Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

1. https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25
2. https://www.youtube.com/watch?v=UwbuW7oK8rk
3. https://www.youtube.com/watch?v=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
- We have two data files: one conatins the information about the genetic mutations and the other contains the clinical evidence (text) that human experts/pathologists use to classify the genetic mutations.
- Both these data files are have a common column called ID
- Data file's information:
  - training_variants (ID , Gene, Variations, Class)
  - training_text (ID, Text)

2.1.2. Example Data Point

training_variants

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

training_text

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

- Interpretability
- Class probabilities are needed.
- Penalize the errors in class probabilites => Metric is Log-loss.

- No Latency constraints.

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [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,TfidfVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.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
from sklearn.manifold import TSNE
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

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

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:

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

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

Out[2]:

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

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

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

3.1.2. Reading Text Data

In [3]:

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

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

Out[3]:

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

3.1.3. Preprocessing of text

In [4]:

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


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

```
        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 : 148.175293201 seconds
```

In [7]:

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

Out[7]:

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

In [8]:

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

Out[8]:

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

In [9]:

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

In [10]:

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

Out[10]:

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

In [11]:

```
result.to_csv('result.csv')
```

In [2]:

```
result = pd.read_csv('result.csv')
```

In [3]:

```
result.head()
```

Out[3]:

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

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 [3]:

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

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

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

In [4]:

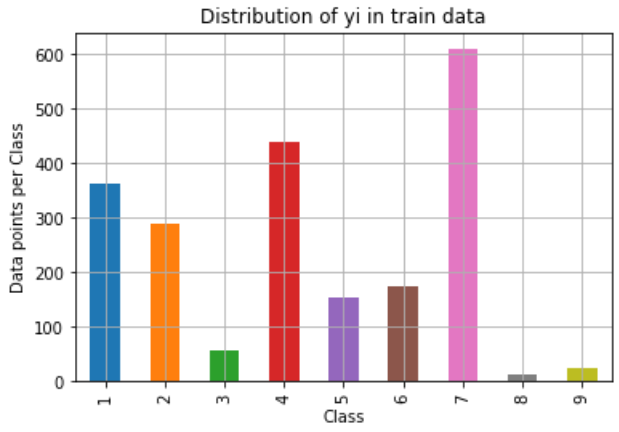
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
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

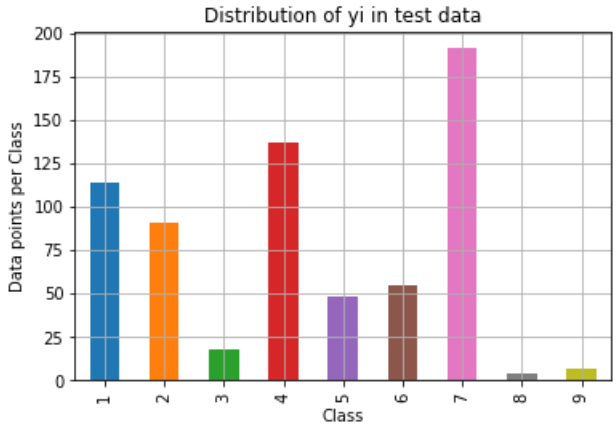
In [5]:

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

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

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


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

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

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

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

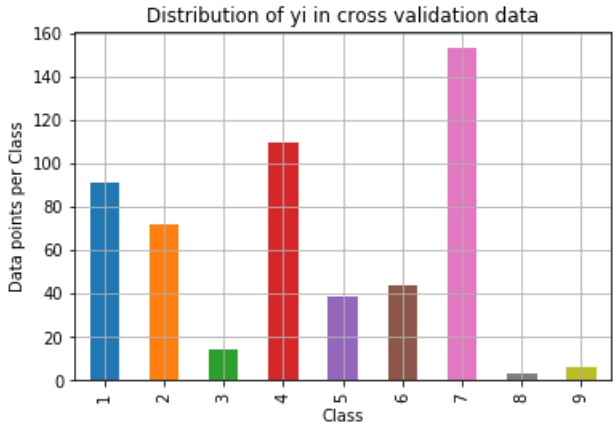



```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
```

```
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298 %)
Number of data points in class 3 : 57 ( 2.684 %)
Number of data points in class 9 : 24 ( 1.13 %)
Number of data points in class 8 : 12 ( 0.565 %)
------------------------------------------------------------------------------------------
```




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




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

3.2 Prediction using a 'Random' Model

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

In [51]:

```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
```

```
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

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

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

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

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

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

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

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

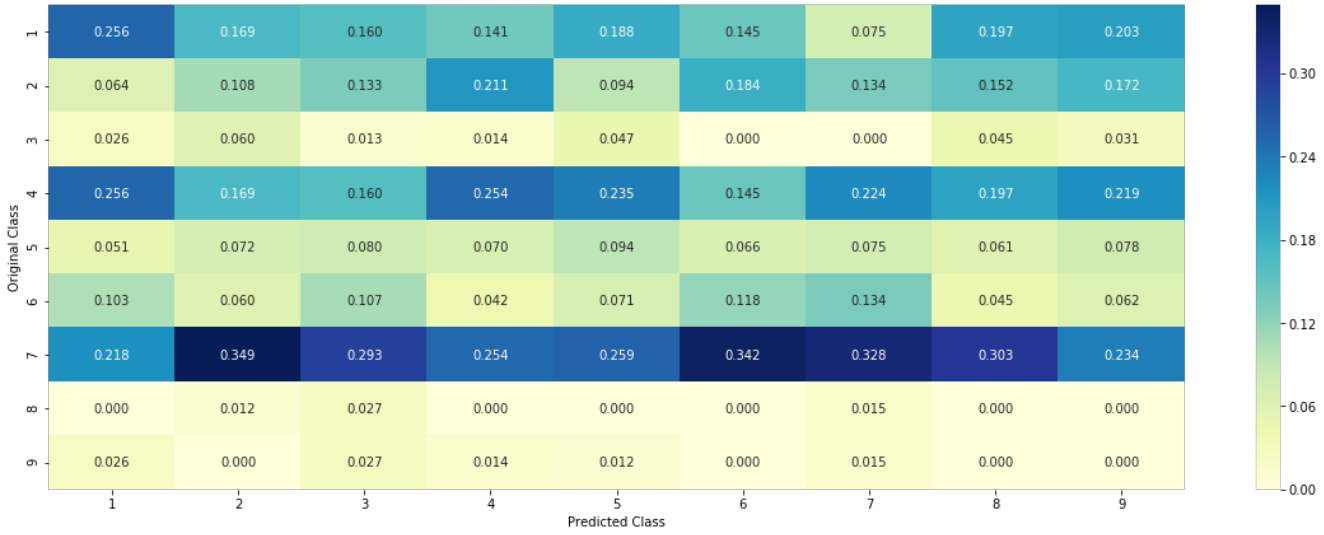
In [8]:

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

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


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

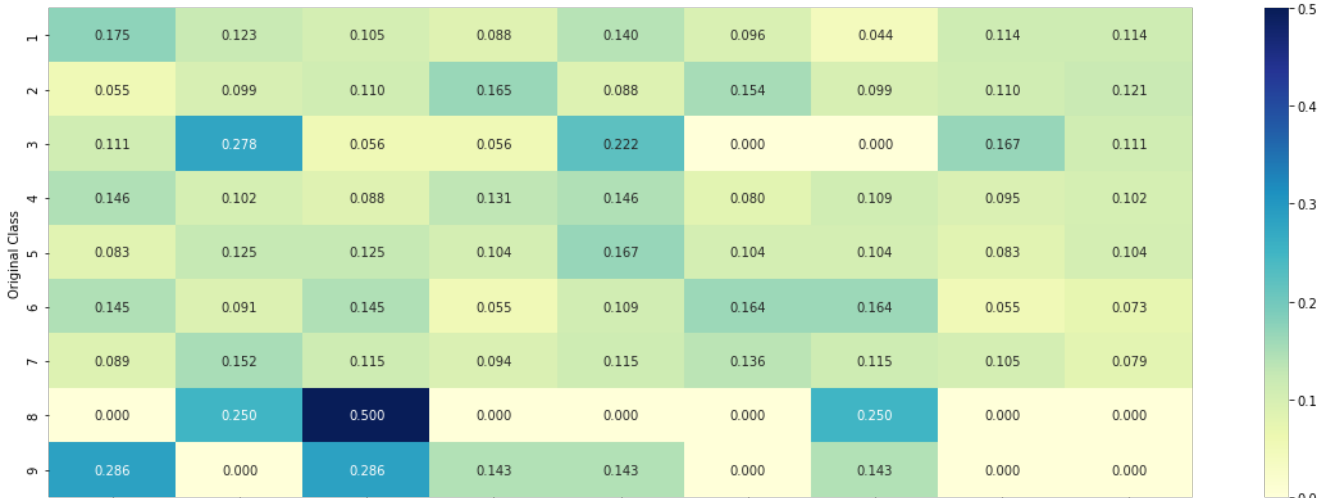
```
print('Log loss on Test Data using Random Model',log_loss(y_test,test_predicted_y, eps=1e-15))

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

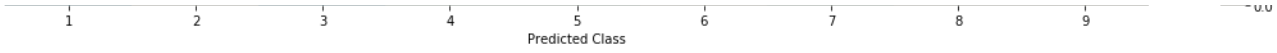
```
Log loss on Cross Validation Data using Random Model 2.5767485019416045
Log loss on Test Data using Random Model 2.4134816530256047
-------------------- Confusion matrix --------------------
```


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




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





3.3 Univariate Analysis

In [6]:

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

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

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

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

```
            # cls_cnt.shape[0] will return the number of rows

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

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177, 0.1363636363636363
5, 0.25, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.2704081632653061
5, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408163265307, 0.0561224489
79591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177, 0.06818181
8181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.078787878787878
782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608, 0.060606060606060608, 0.06060606060
6060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.465408805031446
55, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081761006289, 0.062893081
761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.07284768211920529
5, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702, 0.066225165562913912, 0.06622516556
2913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.0733333333333333
34, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999, 0.066666666666666666, 0.0666666666
66666666],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

In [7]:

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

```
Number of Unique Genes : 241
```

```
BRCA1      164
TP53       100
BRCA2       84
PTEN        84
EGFR        82
KIT         56
ALK         55
BRAF        52
ERBB2       52
PIK3CA      38
Name: Gene, dtype: int64
```

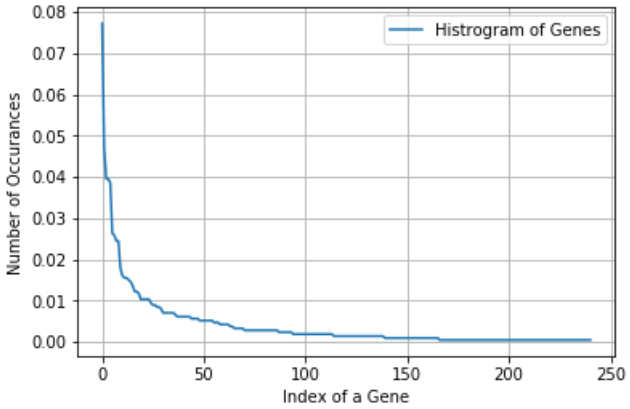
In [8]:

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

```
Ans: There are 241 different categories of genes in the train data, and they are distibuted as follows
```

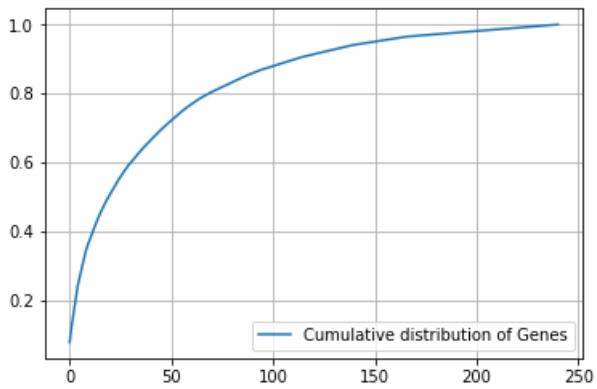
In [9]:

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



In [10]:

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



Class labels for top 5 most occuring genes

In [11]:

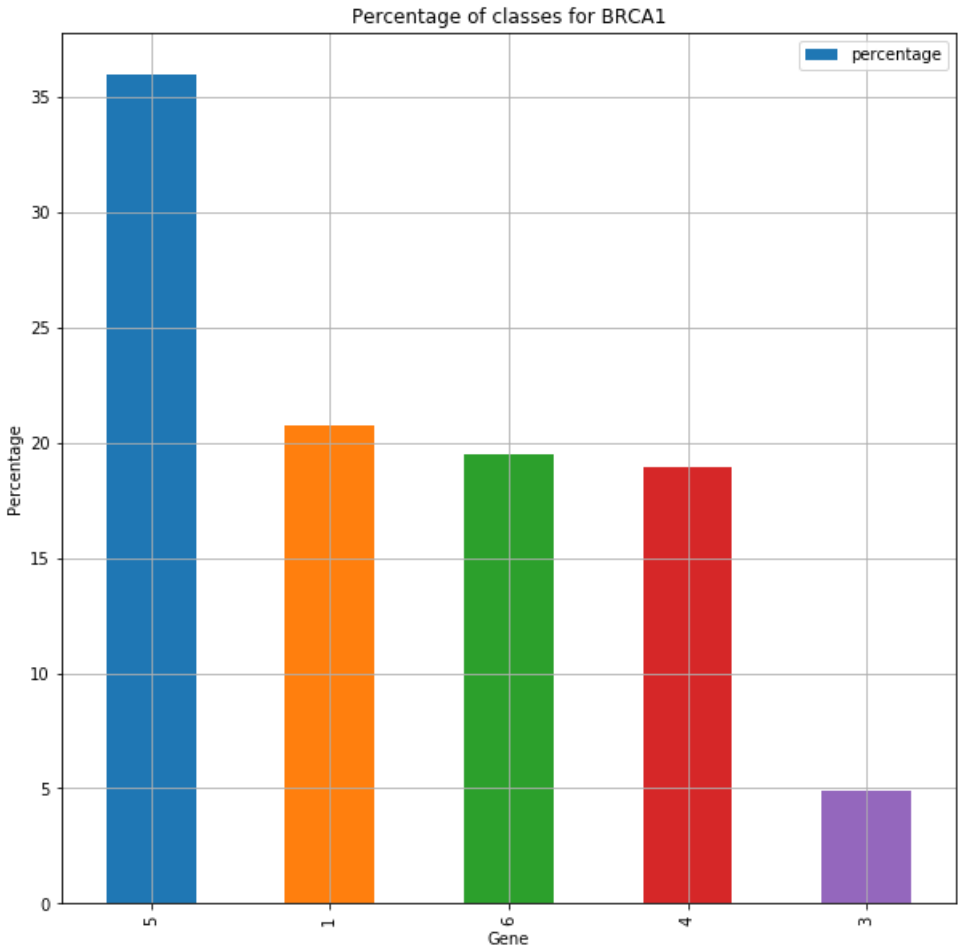
```
unique_genes.head(5)
```

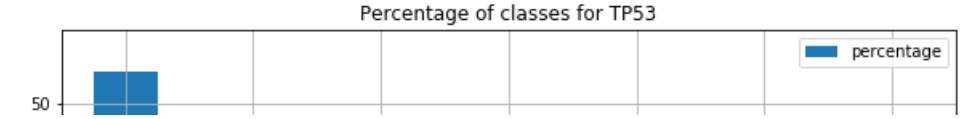
Out[11]:

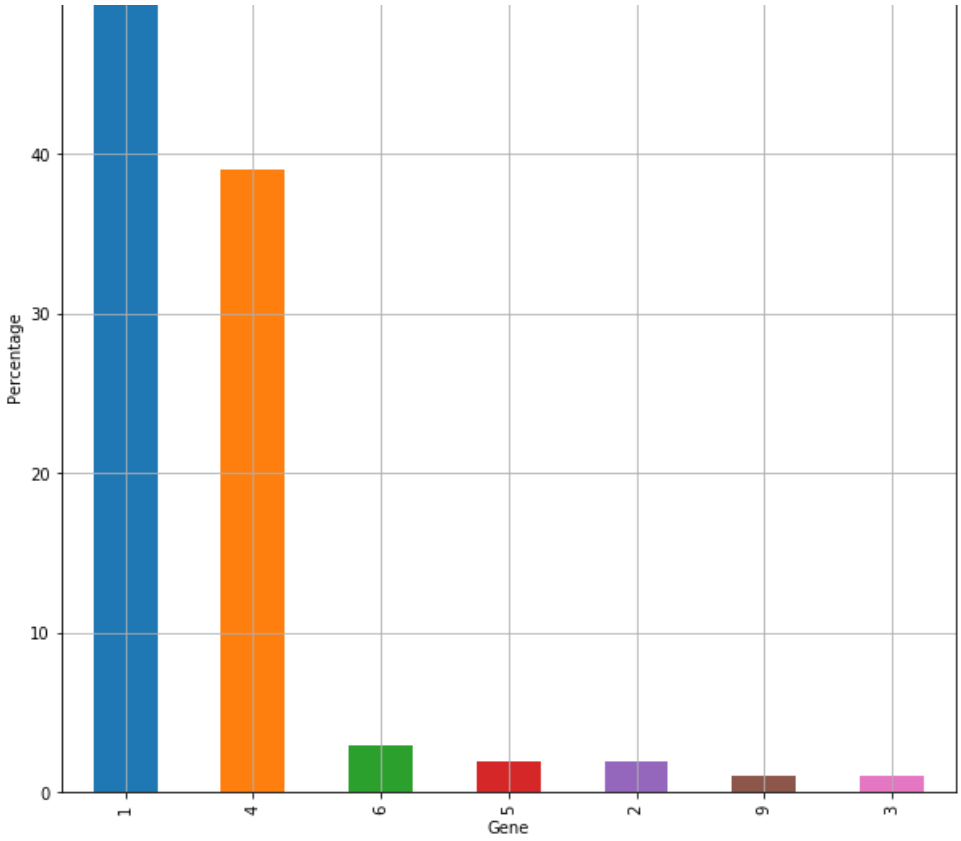
```
BRCA1    164
TP53     100
BRCA2     84
PTEN      84
EGFR      82
Name: Gene, dtype: int64
```

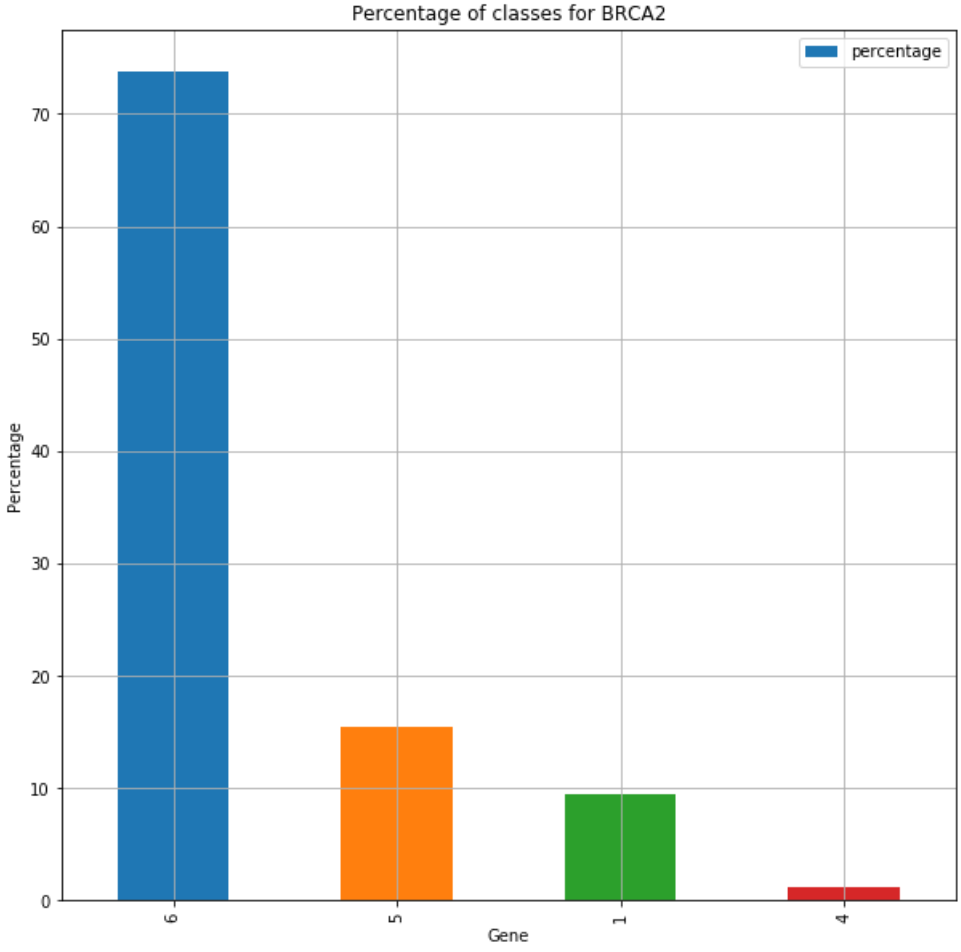
In [12]:

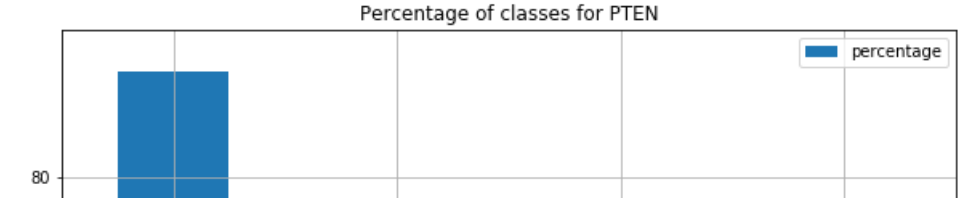
```
for gene in unique_genes.head(5).index:
    #print(gene)
    specificfic_gene = train_df[train_df.Gene == gene]
    val = specificfic_gene.Class.value_counts().reset_index()
    val.columns=['Class', 'Count']

    total = val.Count.sum()
    val['percentage'] = (val.Count/total)*100

    val.plot(x='Class',y='percentage',kind='bar',figsize=(10,10))
    plt.xlabel('Gene')
    plt.ylabel('Percentage')
    plt.title('Percentage of classes for {0}'.format(gene))
    plt.grid()
    plt.show()
```

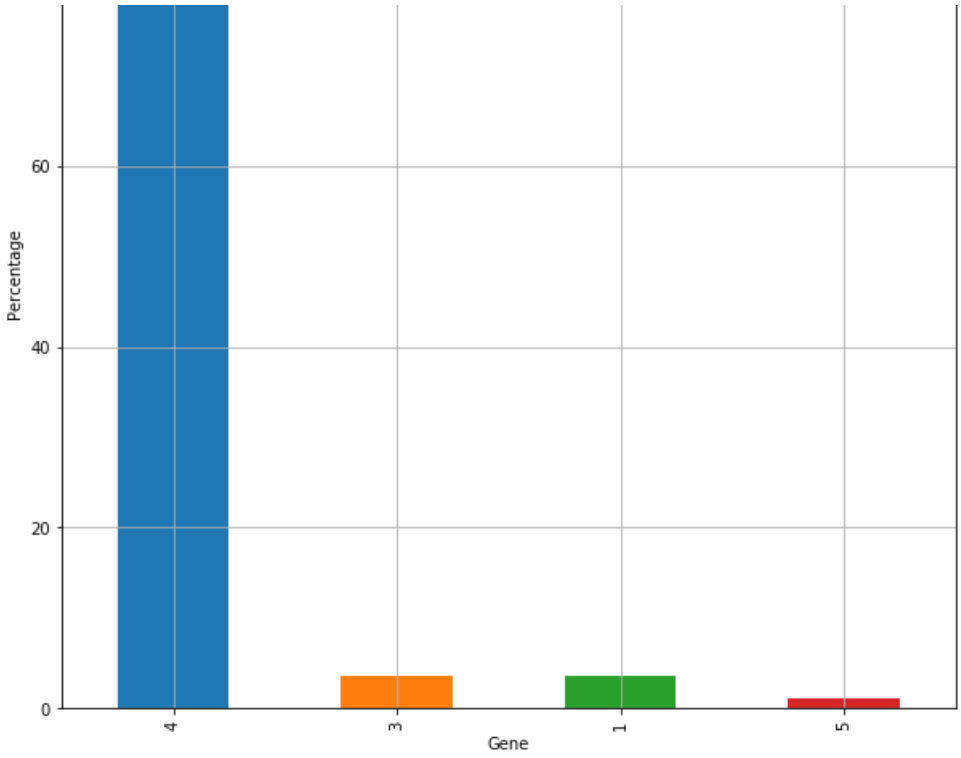


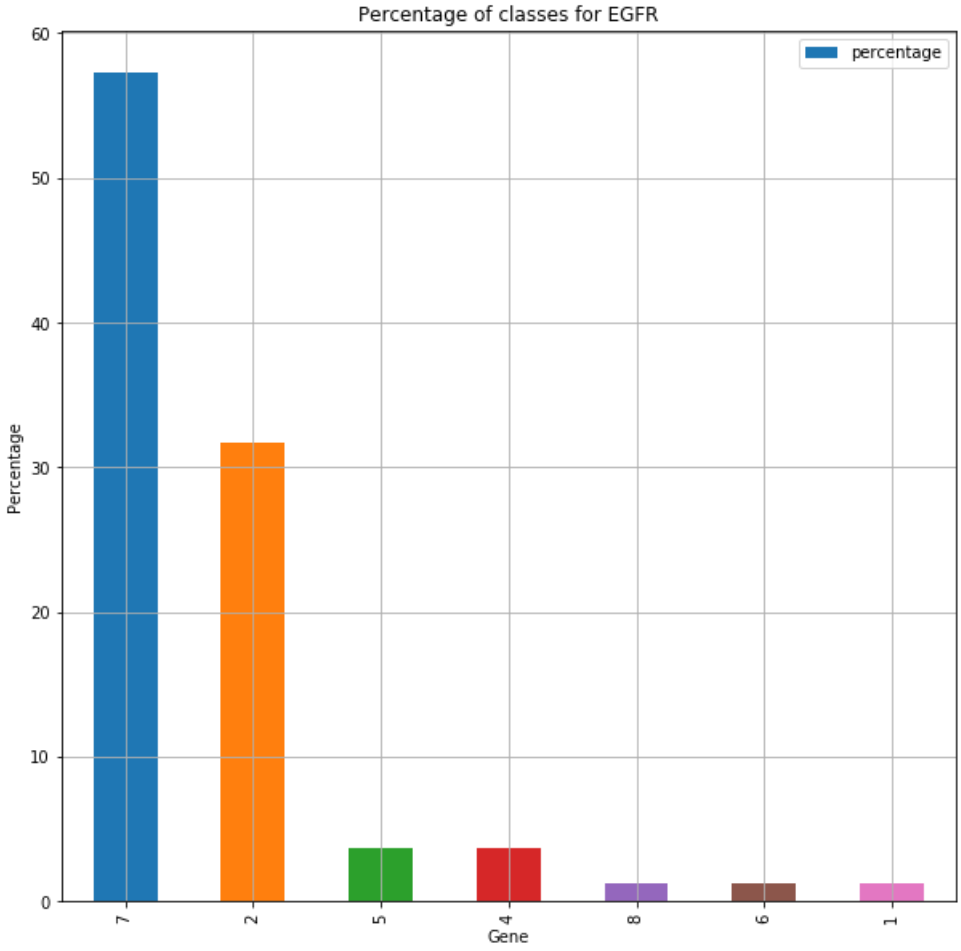



Percentage
0
10
20
30
40
1
4
6
5
2
9
3
Gene


Percentage of classes for BRCA2
percentage
Percentage
0
10
20
30
40
50
60
70
6
5
1
4
Gene


Percentage of classes for PTEN
percentage
80





We can see for some of the gene we got tall bars for specific classes. So we can say we are that much sure that the gene belongs to that specific class.

Q3. How to featurize this Gene feature ?

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

1. One hot Encoding

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 [13]:

```
#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 [14]:

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

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

In [15]:

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

In [19]:

```
#not necessary our main aim is to convert feature to one hot encoding TFIDF doesnot do that
# one-hot encoding of Gene feature.
#gene_vectorizer_ = TfidfVectorizer()
#train_gene_feature_onehotCoding_ = gene_vectorizer_.fit_transform(train_df['Gene'])
#test_gene_feature_onehotCoding_ = gene_vectorizer_.transform(test_df['Gene'])
#cv_gene_feature_onehotCoding_ = gene_vectorizer_.transform(cv_df['Gene'])
```

In [20]:

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

Out[20]:

```
2393    PTPN11
2084     MYD88
2701      BRAF
504       TP53
1046      TSC2
Name: Gene, dtype: object
```

In [21]:

```
#gene_vectorizer.get_feature_names()
```

In [22]:

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene f
eature: (2124, 240)
```

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

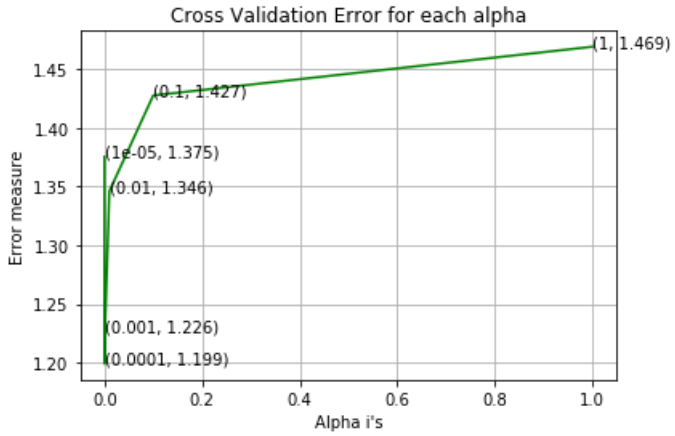
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [23]:

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

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

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

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


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

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


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

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


misclassified_pt = np.count_nonzero((sig_clf.predict(cv_gene_feature_onehotCoding)- y_cv))/y_cv.shape[0
]

columns = ["ML Model","features", "encoding","Vectorizer for text","hyperparameter","train loss","CV lo
ss","test loss","misclassified point(%)","interpretability"]
models_performence = {
    'ML Model':'LR',
```

```
    'features': 'Gene',
    'encoding':'onehot',
    'Vectorizer for text':'NA',
    'hyperparameter':"alpha {0}".format(alpha[best_alpha]),
    'train loss':tr_loss,
    'CV loss':cv_loss,
    'test loss':test_loss,
    'misclassified point(%)':misclassified_pt*100,
    'interpretability':'YES'
}

ittr=0
df = pd.DataFrame(models_performence, columns=columns, index =[0])
```

```
For values of alpha =  1e-05 The log loss is: 1.3752693858721317
For values of alpha =  0.0001 The log loss is: 1.1989127623583515
For values of alpha =  0.001 The log loss is: 1.226145751781846
For values of alpha =  0.01 The log loss is: 1.3458257354407248
For values of alpha =  0.1 The log loss is: 1.4274622483370658
For values of alpha =  1 The log loss is: 1.468909200200966
```




```
For values of best alpha =  0.0001 The train log loss is: 1.038632857279468
For values of best alpha =  0.0001 The cross validation log loss is: 1.1989127623583515
For values of best alpha =  0.0001 The test log loss is: 1.208968450620473
```

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 [24]:

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

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  240  genes in train dataset?
Ans
1. In test data 647 out of 665 : 97.29323308270676
2. In cross validation data 518 out of  532 : 97.36842105263158
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Ans. Variation is a categorical variable

Q8. How many categories are there?

In [16]:

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

```
Number of Unique Variations : 1922
Truncating_Mutations    60
Deletion                52
Amplification           44
Fusions                 24
Overexpression           3
Q61R                     3
E17K                     3
A146V                    2
T73I                     2
G13V                     2
Name: Variation, dtype: int64
```

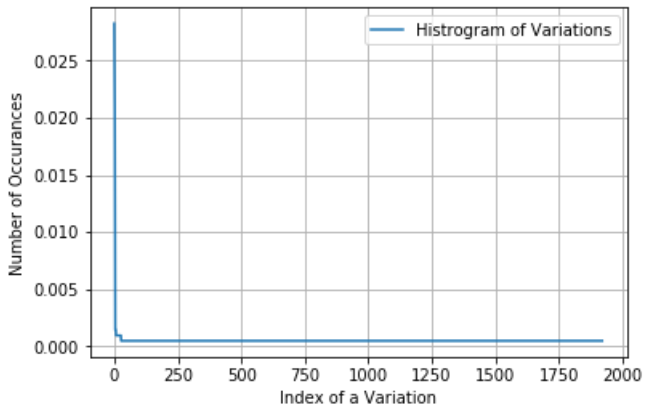
In [17]:

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

```
Ans: There are 1922 different categories of variations in the train data, and they are distibuted as fo
llows
```

In [18]:

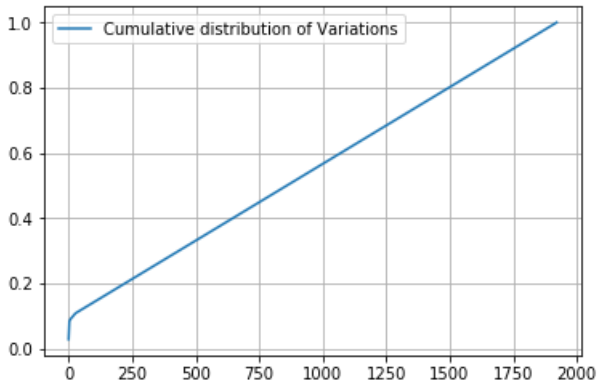
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [19]:

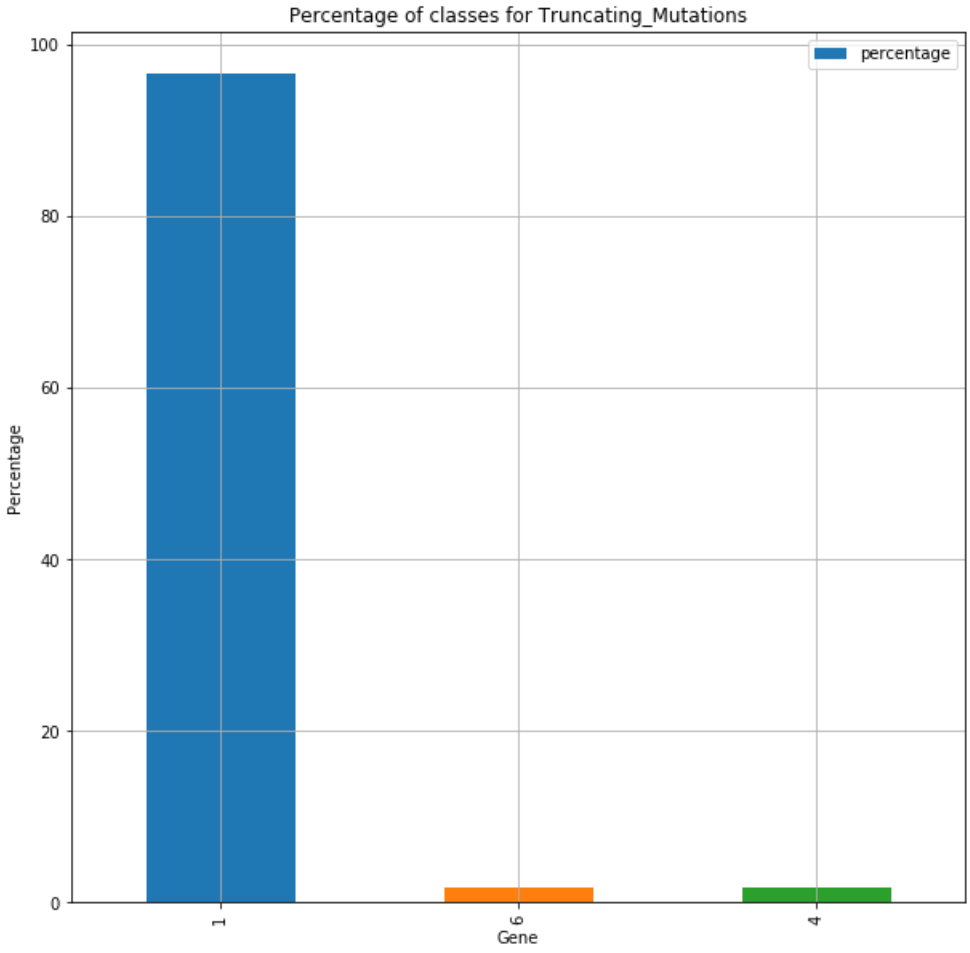
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

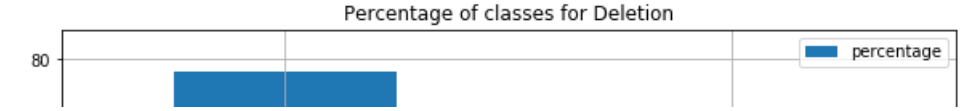
```
[0.02824859 0.0527307  0.07344633 ... 0.99905838 0.99952919 1.        ]
```

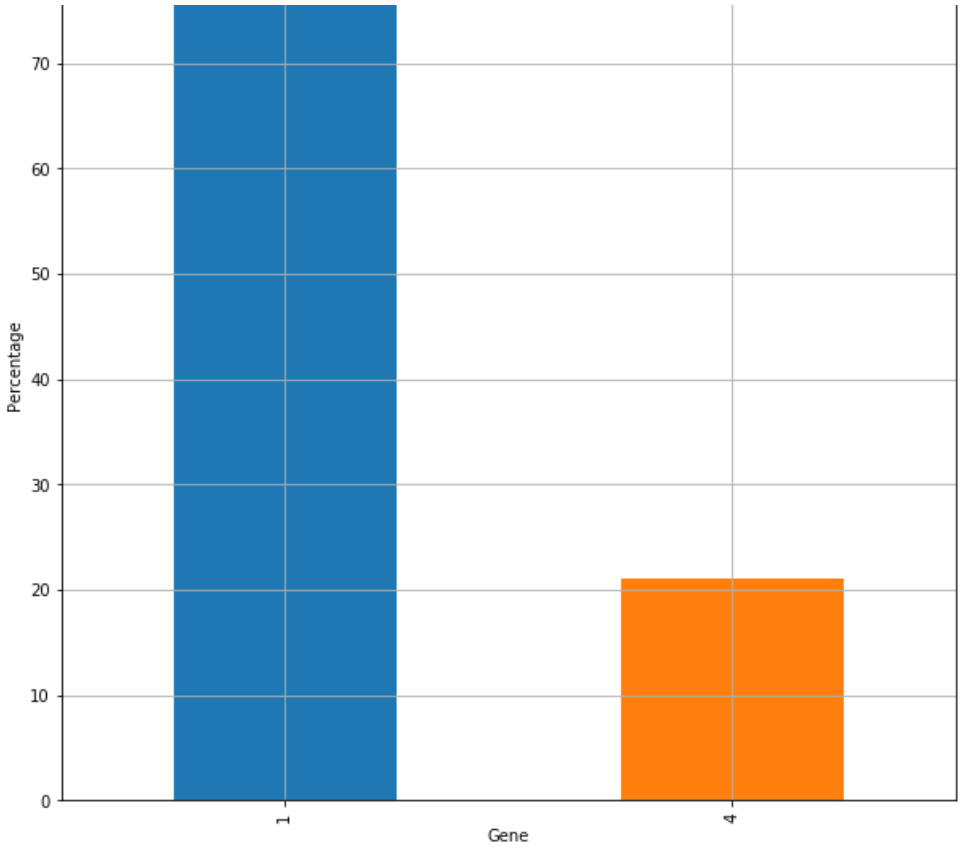


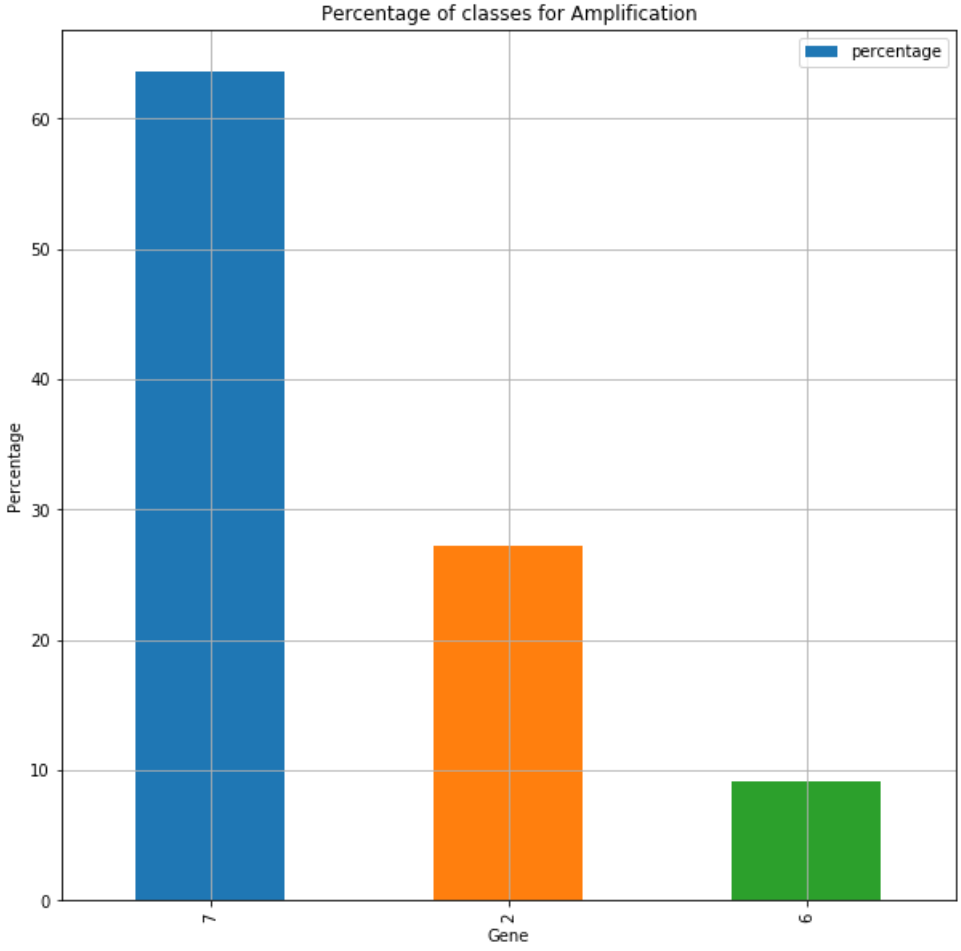
In [20]:

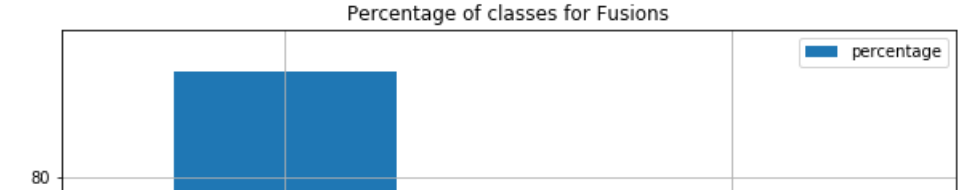
```
for variation in unique_variations.head(5).index:
    #print(gene)
    specificfic_var = train_df[train_df.Variation == variation]
    val_var = specificfic_var.Class.value_counts().reset_index()
    val_var.columns=['Class', 'Count']

    total = val_var.Count.sum()
    val_var['percentage'] = (val_var.Count/total)*100

    val_var.plot(x='Class',y='percentage',kind='bar',figsize=(10,10))
    plt.xlabel('Gene')
    plt.ylabel('Percentage')
    plt.title('Percentage of classes for {0}'.format(variation))
    plt.grid()
    plt.show()
```

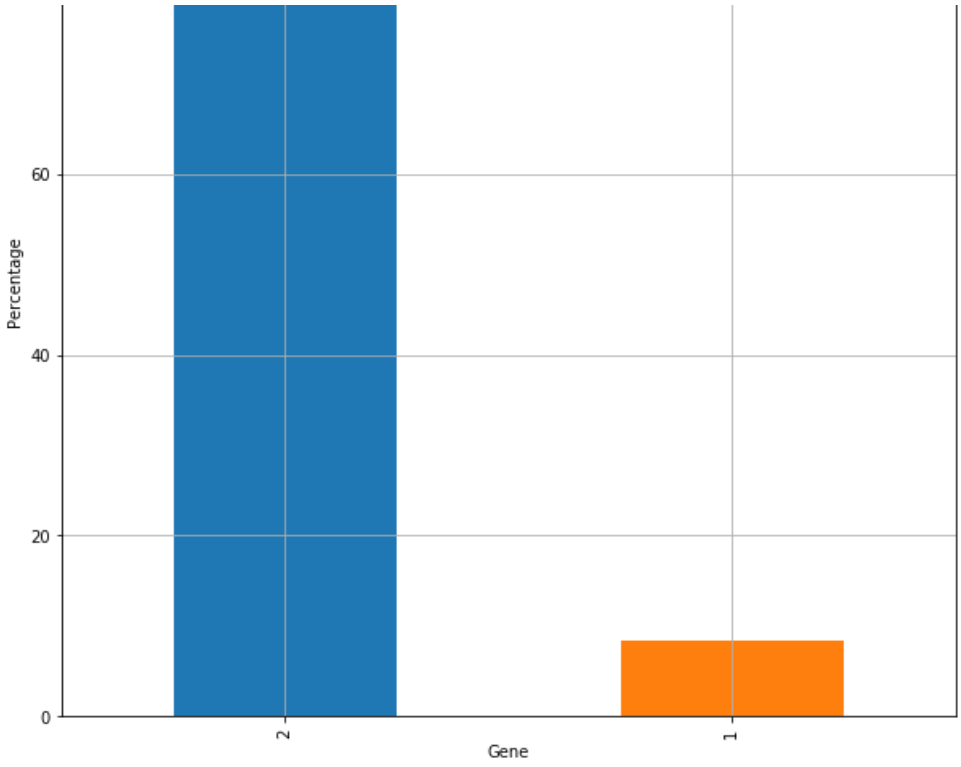


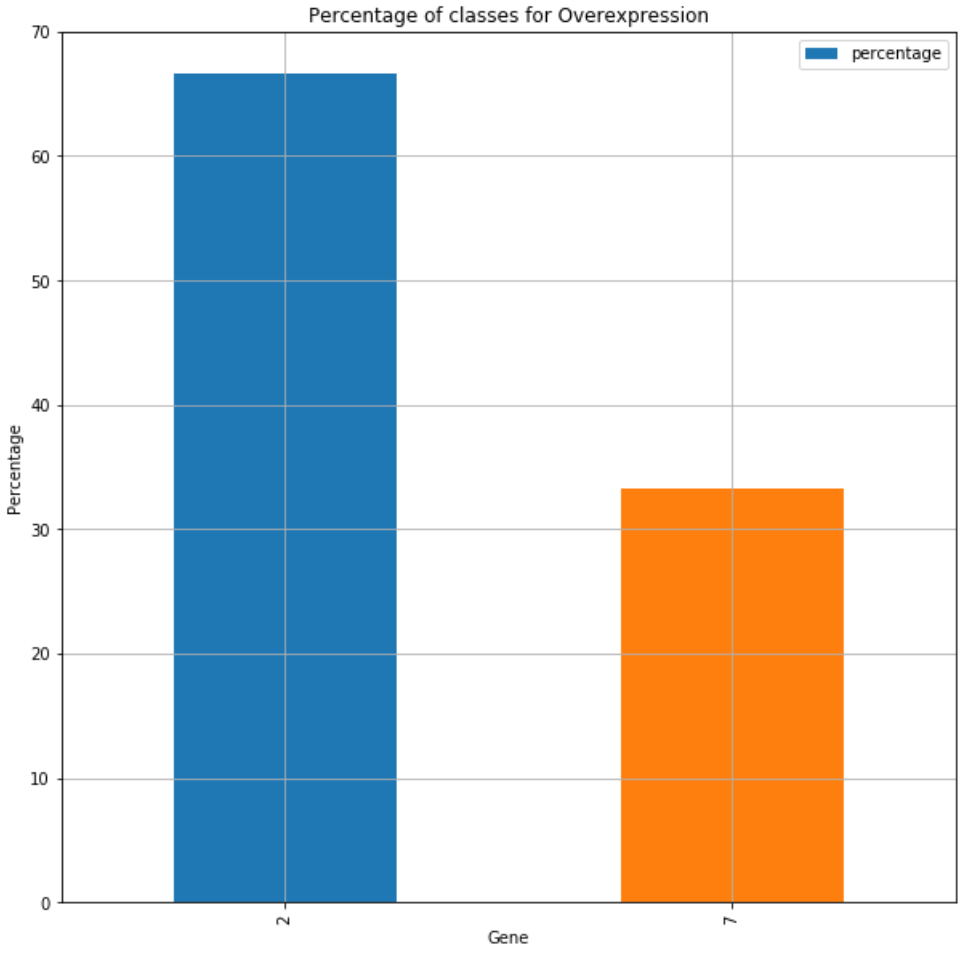



70
60
50
40
30
20
10
0
Percentage
1
4
Gene


Percentage of classes for Amplification
percentage
60
50
40
30
20
10
0
Percentage
7
2
6
Gene


Percentage of classes for Fusions
percentage
80





Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [21]:

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

In [22]:

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

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

In [23]:

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

In [33]:

```
#Not necessary we do one hot encoding by BOW
# 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 [34]:

```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. T
he shape of Variation feature:", train_variation_feature_onehotCoding.shape)
```

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shap
e of Variation feature: (2124, 1968)
```

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

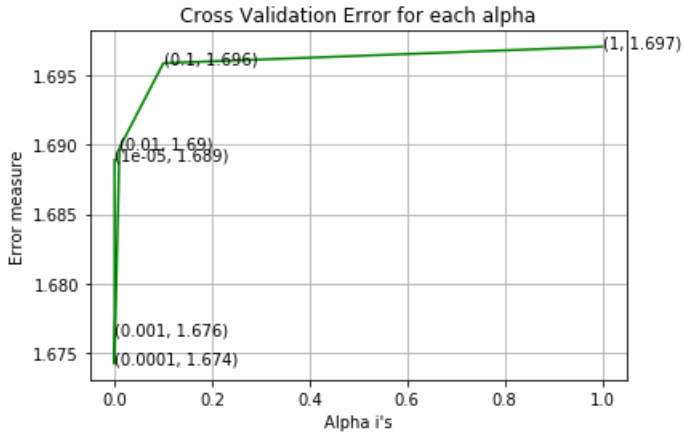
Let's build a model just like the earlier!

In [35]:

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

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

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

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

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


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

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

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

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


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

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

misclassified_pt = np.count_nonzero((sig_clf.predict(cv_variation_feature_onehotCoding)- y_cv))/y_cv.sh
ape[0]

ittr+=1
df.loc[ittr]= ['LR', 'Variance','onehot','NA',"alpha {0}".format(alpha[best_alpha]),tr_loss,cv_loss,tes
t_loss,misclassified_pt*100,'YES']
```

```
For values of alpha =  1e-05 The log loss is: 1.6889082234443813
For values of alpha =  0.0001 The log loss is: 1.6742598606039054
For values of alpha =  0.001 The log loss is: 1.6763562798278302
For values of alpha =  0.01 The log loss is: 1.6897609301312262
For values of alpha =  0.1 The log loss is: 1.6958428821901737
For values of alpha =  1 The log loss is: 1.6970212007232504
```



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

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 [36]:

```
print("Q12. How many data points are covered by total ", unique_variations.shape[0], " genes in test an
d cross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/test_df.shape
[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.shape
[0])*100)
```

```
Q12. How many data points are covered by total  1936  genes in test and cross validation data sets?
Ans
1. In test data 62 out of 665 : 9.323308270676693
2. In cross validation data 68 out of  532 : 12.781954887218044
```

3.2.3 Univariate Analysis on Text Feature

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

In [37]:

```
# 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 [38]:

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

In [39]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = CountVectorizer(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)
vector
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

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


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

```
Total number of unique words in train data : 53198
```

In [40]:

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

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


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

In [41]:

```
#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 [42]:

```
# 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_responseCod
ing.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(a
xis=1)).T
```

In [43]:

```
# 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)
```

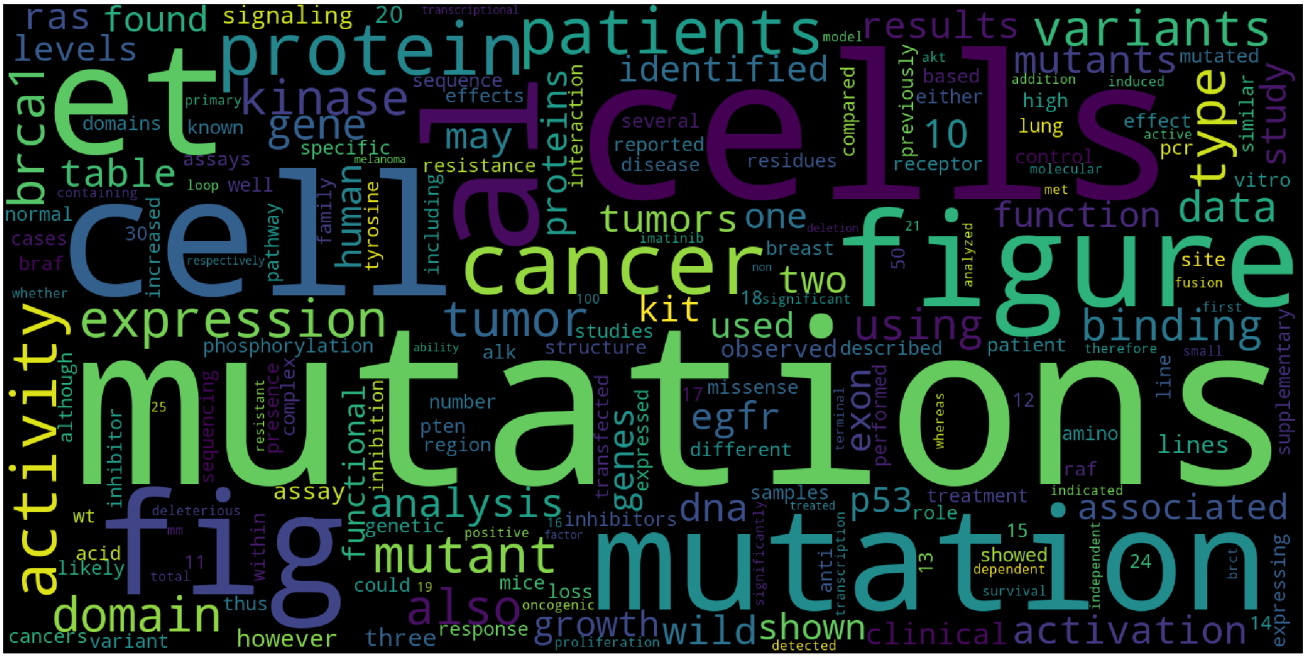
In [44]:

```
#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 [44]:

```
from wordcloud import WordCloud
# Lets first convert the 'result' dictionary to 'list of tuples'
print("Word cloud for most common words")

most_common_word = sorted_text_fea_dict
#Initializing WordCloud using frequencies of tags.
wordcloud = WordCloud(    background_color='black',
                          width=1600,
                          height=800,
                    ).generate_from_frequencies(most_common_word)

fig = plt.figure(figsize=(30,20))
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad=0)
fig.savefig("most_common_word.png")
plt.show()
```

Word cloud for most common words

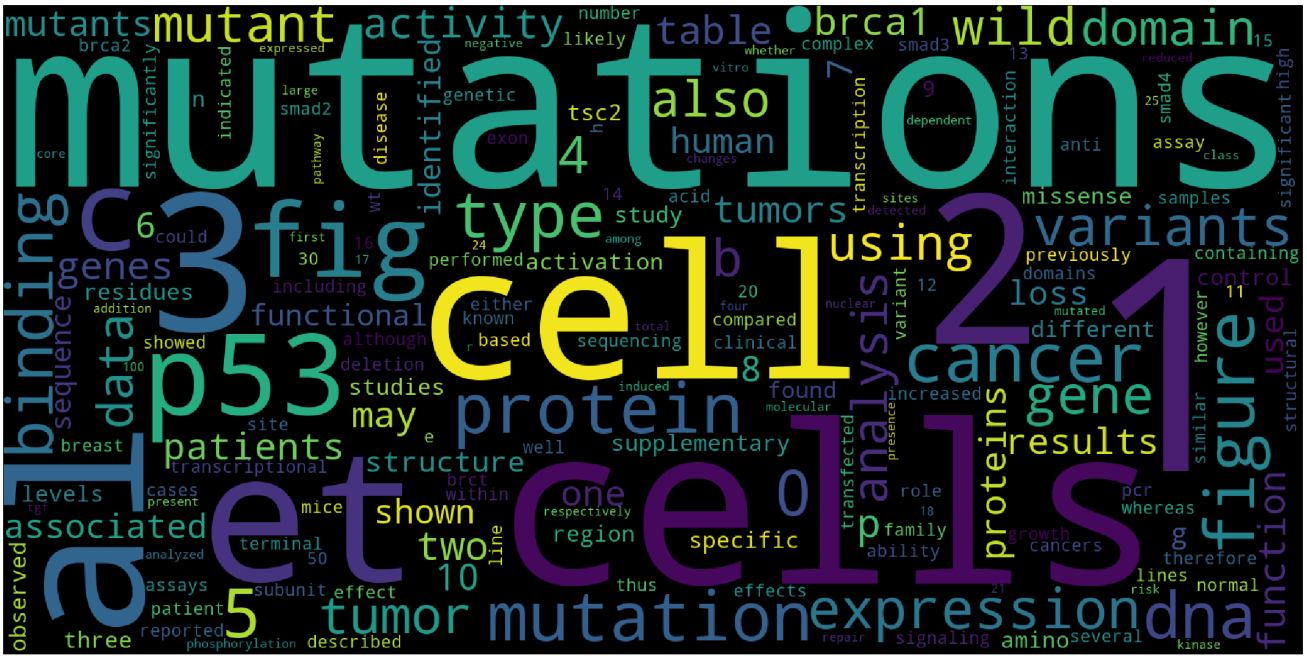


In [45]:

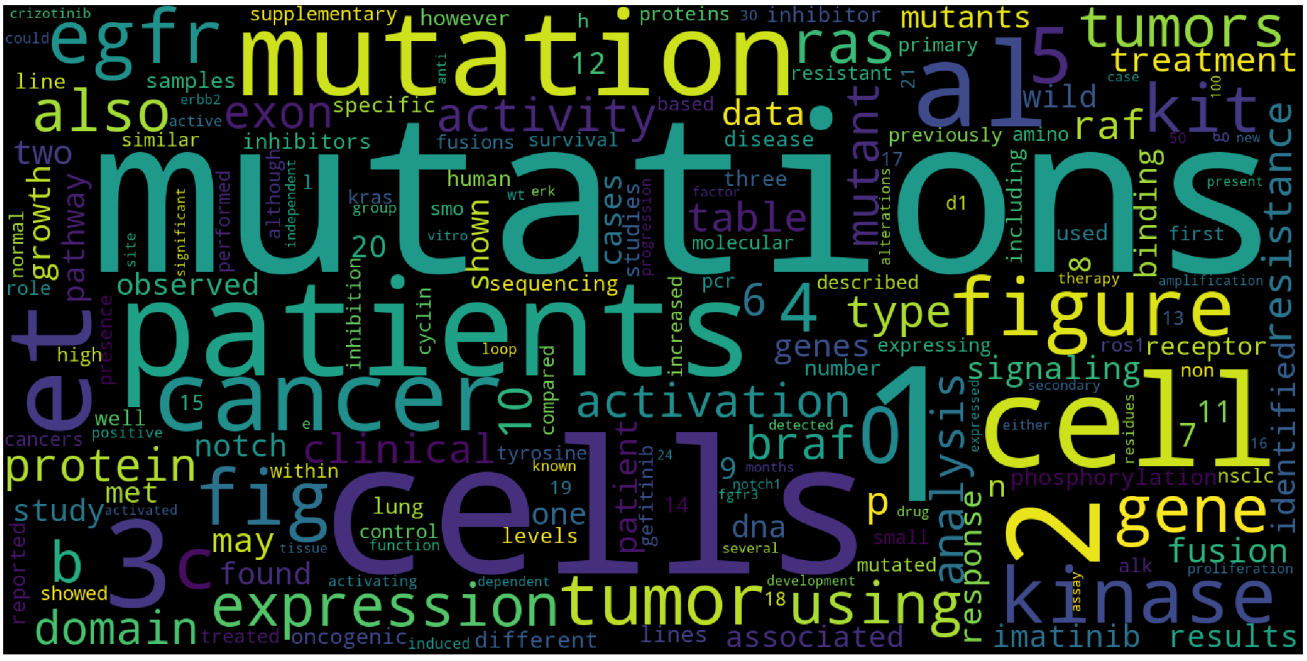
```
print("Word cloud based on text data class wise")
for i in range(9):
    print("most common words for class [{0}]".format(i+1))
    most_common_word_classwise = dict_list[i]
    #Initializing WordCloud using frequencies of tags.
    wordcloud = WordCloud(    background_color='black',
                              width=1600,
                              height=800,
                        ).generate_from_frequencies(most_common_word_classwise)

    fig = plt.figure(figsize=(30,20))
    plt.imshow(wordcloud)
    plt.axis('off')
    plt.tight_layout(pad=0)
```

```
    name = "most_common_class{0}.png".format(i)
    fig.savefig(name)
    plt.show()
```

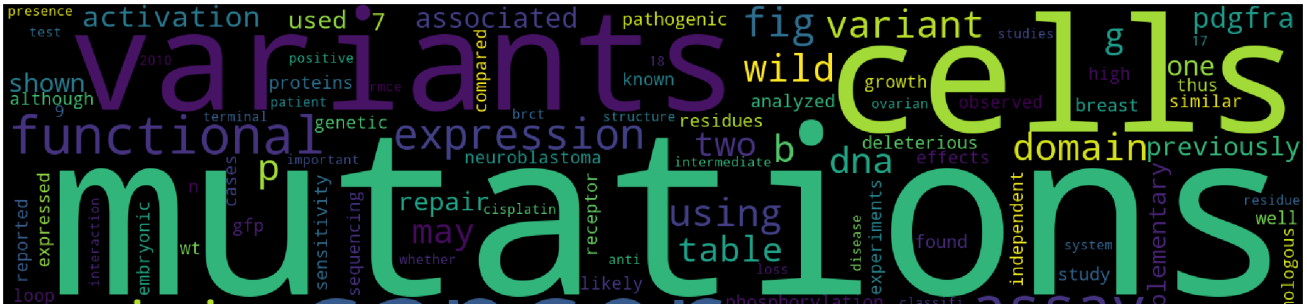
Word cloud based on text data class wise
most common words for class [1]

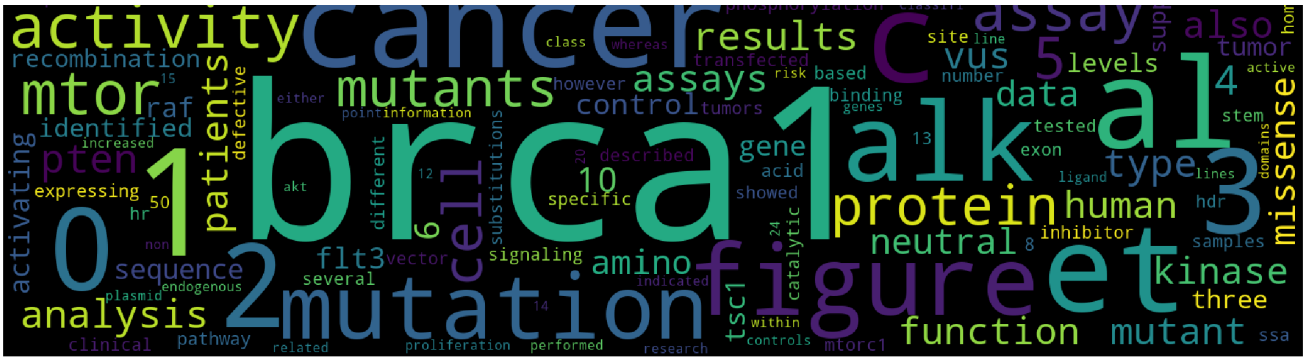


most common words for class [2]

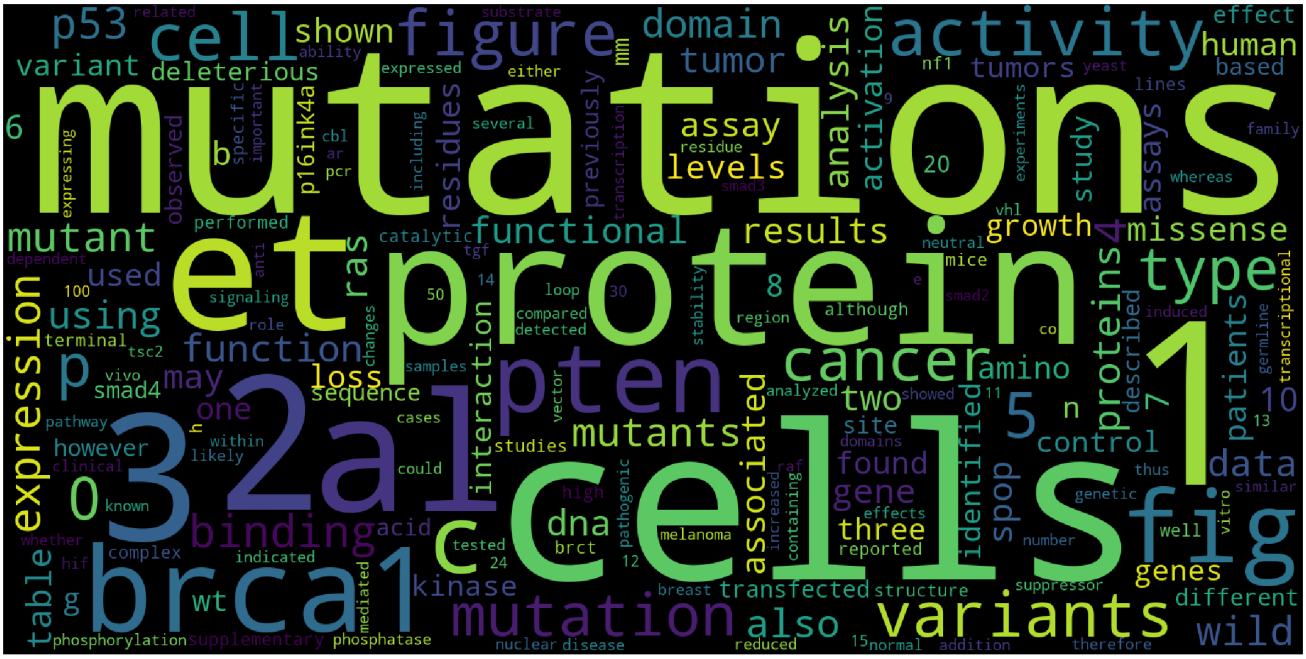


most common words for class [3]

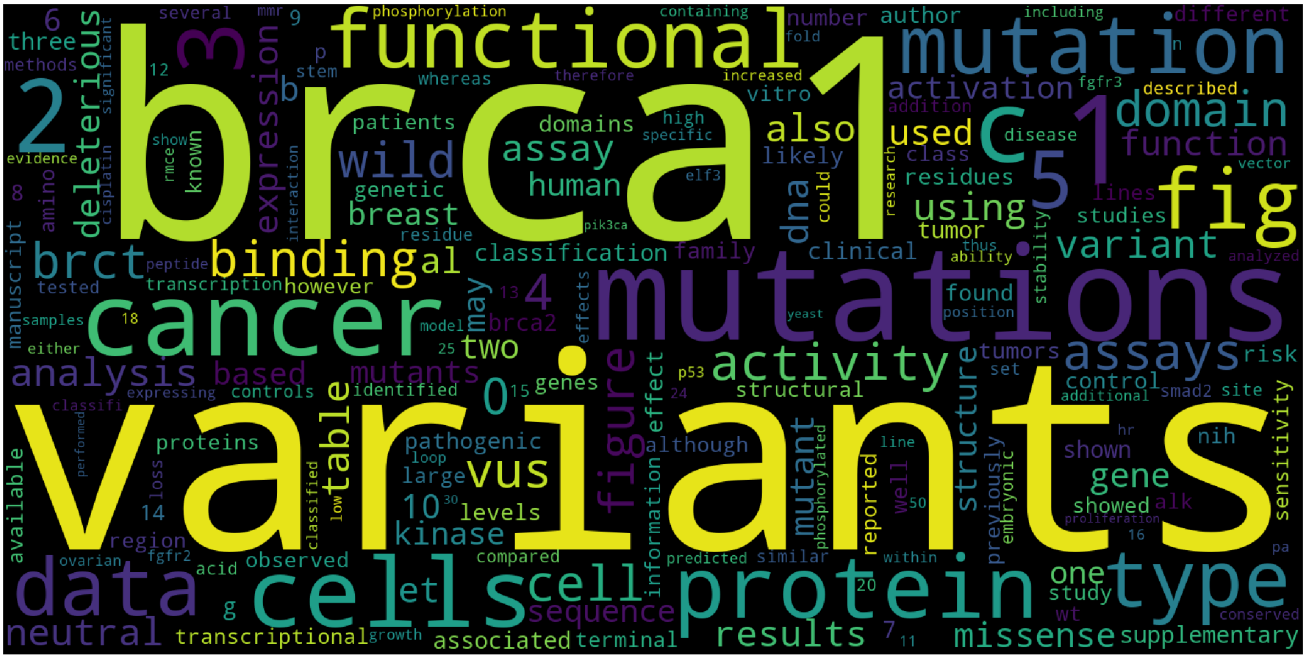




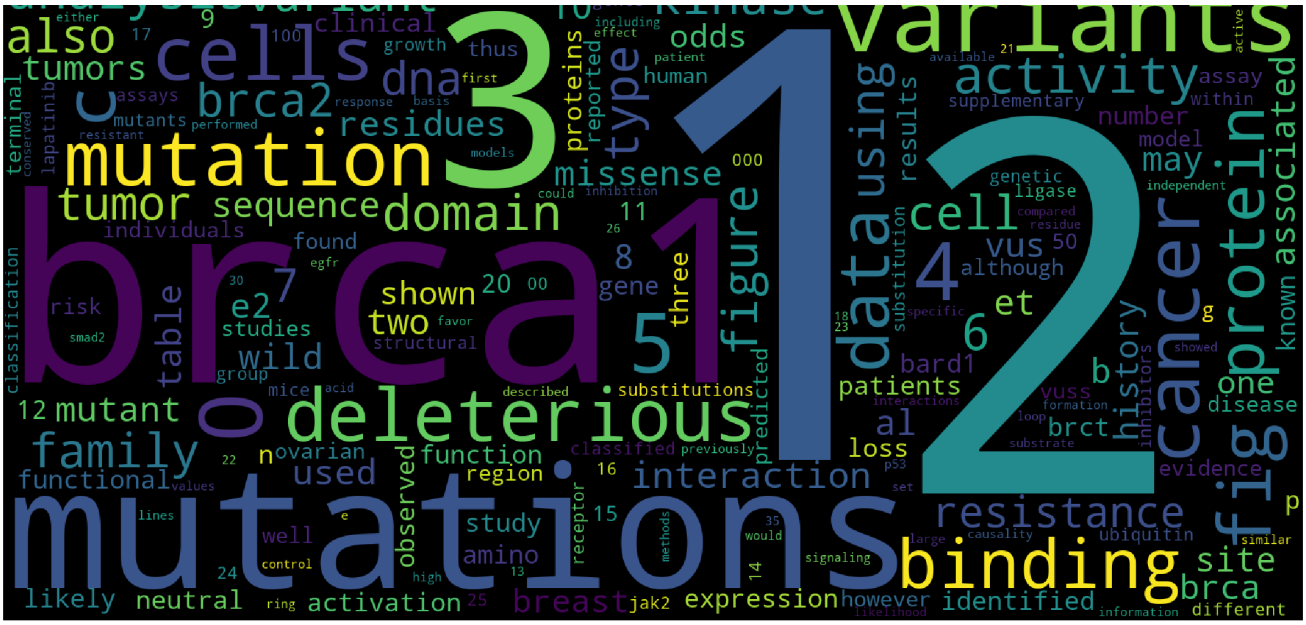
most common words for class [4]



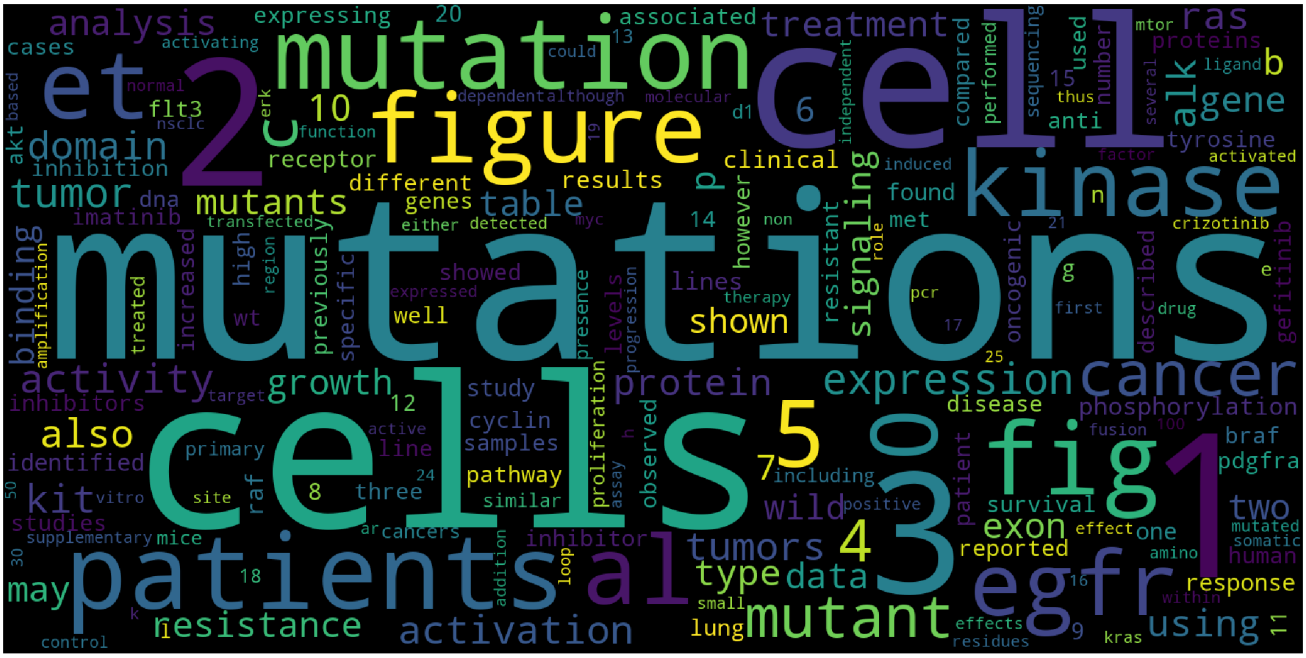
most common words for class [5]



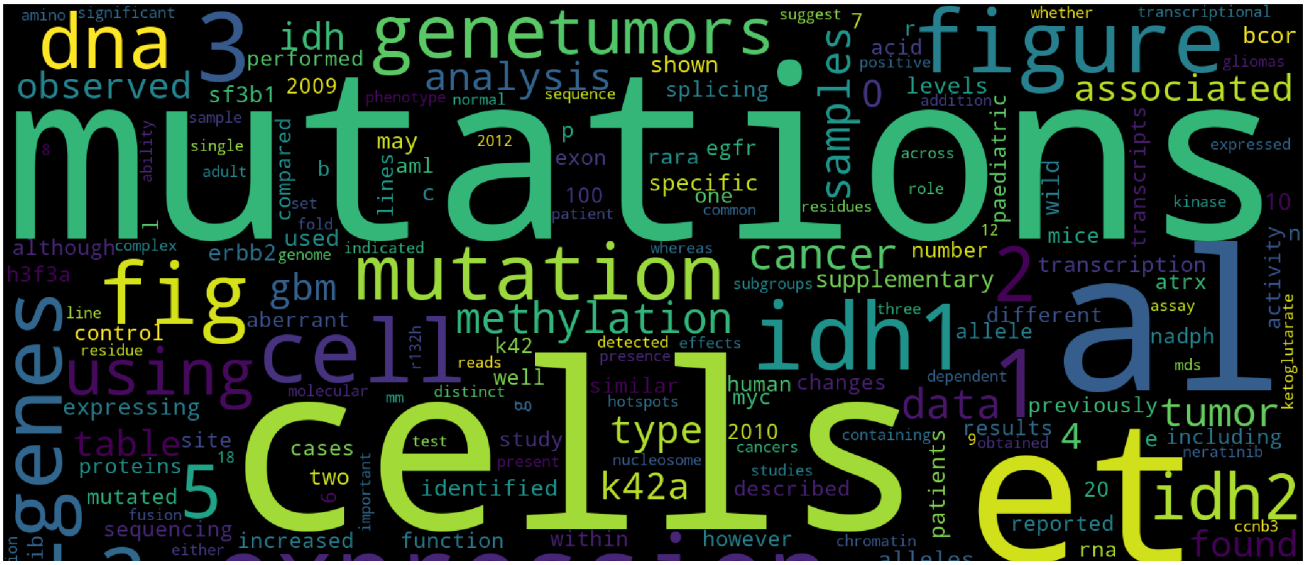
most common words for class [6]



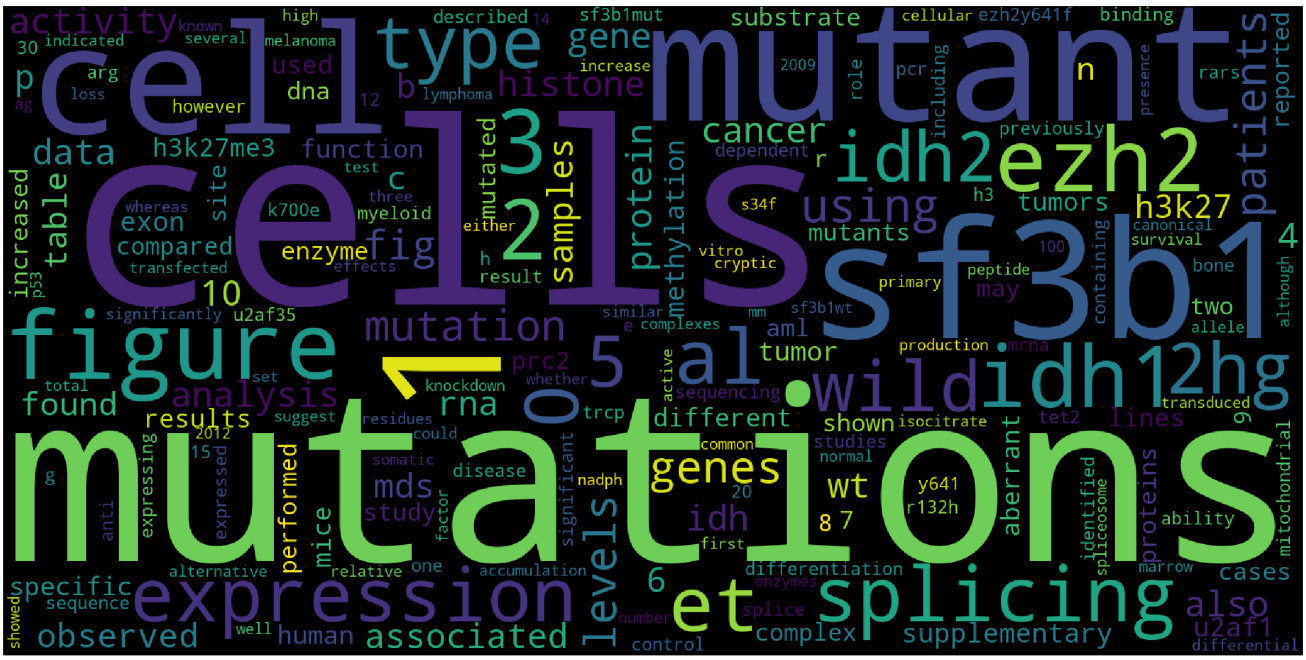
most common words for class [7]



most common words for class [8]



most common words for class [9]



In [46]:

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

```
Counter({3: 5947, 4: 3478, 5: 3061, 6: 2528, 8: 2279, 10: 1899, 7: 1787, 9: 1534, 12: 1131, 11: 1121, 1
4: 1065, 15: 1037, 16: 872, 13: 771, 17: 697, 18: 681, 20: 576, 21: 512, 24: 476, 23: 471, 22: 450, 19:
445, 25: 420, 27: 399, 30: 369, 26: 366, 28: 351, 44: 348, 29: 347, 32: 295, 33: 272, 34: 262, 31: 262,
53: 256, 36: 239, 42: 238, 35: 237, 40: 228, 39: 227, 48: 214, 38: 206, 41: 189, 37: 186, 45: 178, 46:
168, 43: 166, 52: 162, 47: 148, 56: 147, 59: 144, 60: 140, 49: 136, 54: 135, 66: 134, 55: 130, 63: 127,
51: 125, 50: 122, 57: 120, 65: 114, 67: 108, 61: 106, 69: 102, 64: 101, 62: 101, 58: 100, 68: 97, 85: 9
6, 77: 95, 70: 95, 81: 94, 72: 93, 82: 90, 80: 84, 73: 84, 75: 83, 74: 81, 71: 80, 76: 79, 88: 76, 78:
73, 79: 72, 96: 71, 84: 71, 89: 69, 83: 69, 90: 68, 86: 66, 95: 65, 91: 64, 104: 61, 93: 61, 92: 61, 99
: 60, 87: 60, 120: 59, 112: 59, 98: 59, 97: 58, 105: 57, 94: 56, 113: 54, 108: 54, 106: 54, 100: 53, 10
2: 52, 123: 51, 101: 50, 128: 47, 111: 45, 132: 44, 125: 44, 107: 44, 103: 44, 124: 43, 119: 42, 118: 4
2, 114: 42, 110: 41, 156: 40, 144: 39, 117: 39, 109: 39, 140: 37, 129: 37, 127: 37, 141: 36, 139: 36, 1
62: 35, 159: 35, 149: 35, 126: 35, 143: 34, 138: 34, 154: 33, 146: 33, 175: 32, 168: 32, 165: 32, 150:
32, 148: 32, 142: 32, 122: 32, 115: 32, 221: 31, 151: 31, 116: 31, 182: 30, 134: 30, 161: 29, 152: 29,
135: 29, 133: 29, 208: 28, 198: 28, 177: 28, 169: 28, 160: 28, 131: 28, 176: 27, 121: 27, 153: 26, 147:
26, 145: 26, 212: 25, 205: 25, 194: 25, 171: 25, 163: 25, 261: 24, 204: 24, 202: 24, 201: 24, 188: 24,
184: 24, 183: 24, 173: 24, 170: 24, 164: 24, 195: 23, 192: 23, 189: 23, 186: 23, 180: 23, 172: 23, 130:
23, 197: 22, 178: 22, 155: 22, 136: 22, 251: 21, 249: 21, 226: 21, 207: 21, 196: 21, 167: 21, 157: 21,
137: 21, 241: 20, 220: 20, 200: 20, 174: 20, 270: 19, 209: 19, 206: 19, 203: 19, 187: 19, 179: 19, 166:
19, 287: 18, 260: 18, 237: 18, 223: 18, 219: 18, 216: 18, 199: 18, 190: 18, 348: 17, 296: 17, 265: 17,
235: 17, 228: 17, 222: 17, 218: 17, 211: 17, 210: 17, 185: 17, 323: 16, 294: 16, 263: 16, 259: 16, 252:
16, 246: 16, 245: 16, 238: 16, 224: 16, 217: 16, 215: 16, 181: 16, 158: 16, 347: 15, 320: 15, 288: 15,
285: 15, 280: 15, 258: 15, 254: 15, 243: 15, 236: 15, 214: 15, 193: 15, 362: 14, 314: 14, 310: 14, 291:
14, 282: 14, 281: 14, 277: 14, 274: 14, 247: 14, 242: 14, 239: 14, 225: 14, 435: 13, 370: 13, 338: 13,
329: 13, 326: 13, 269: 13, 268: 13, 264: 13, 262: 13, 253: 13, 230: 13, 229: 13, 213: 13, 191: 13, 493:
12, 457: 12, 352: 12, 351: 12, 328: 12, 313: 12, 308: 12, 307: 12, 302: 12, 300: 12, 293: 12, 292: 12,
279: 12, 275: 12, 272: 12, 267: 12, 255: 12, 250: 12, 240: 12, 233: 12, 232: 12, 586: 11, 424: 11, 383:
11, 340: 11, 333: 11, 319: 11, 316: 11, 312: 11, 278: 11, 273: 11, 248: 11, 231: 11, 514: 10, 406: 10,
395: 10, 388: 10, 379: 10, 361: 10, 345: 10, 341: 10, 330: 10, 324: 10, 317: 10, 315: 10, 306: 10, 303:
10, 299: 10, 289: 10, 271: 10, 257: 10, 256: 10, 672: 9, 620: 9, 539: 9, 498: 9, 476: 9, 453: 9, 444: 9
, 431: 9, 421: 9, 415: 9, 405: 9, 389: 9, 385: 9, 368: 9, 367: 9, 359: 9, 357: 9, 356: 9, 336: 9, 304:
9, 286: 9, 234: 9, 227: 9, 738: 8, 661: 8, 650: 8, 567: 8, 563: 8, 537: 8, 533: 8, 496: 8, 465: 8, 462:
8, 446: 8, 439: 8, 433: 8, 417: 8, 399: 8, 374: 8, 372: 8, 342: 8, 337: 8, 311: 8, 297: 8, 290: 8, 283:
8, 276: 8, 833: 7, 774: 7, 719: 7, 708: 7, 668: 7, 651: 7, 644: 7, 633: 7, 597: 7, 582: 7, 551: 7, 541:
7, 525: 7, 508: 7, 499: 7, 491: 7, 470: 7, 469: 7, 468: 7, 441: 7, 425: 7, 422: 7, 420: 7, 418: 7, 413:
7, 410: 7, 407: 7, 396: 7, 393: 7, 382: 7, 375: 7, 366: 7, 364: 7, 363: 7, 355: 7, 353: 7, 350: 7, 331:
7, 322: 7, 321: 7, 305: 7, 295: 7, 284: 7, 1996: 6, 1286: 6, 888: 6, 821: 6, 785: 6, 755: 6, 751: 6, 72
```

1: 6, 689: 6, 642: 6, 639: 6, 628: 6, 608: 6, 598: 6, 595: 6, 594: 6, 587: 6, 581: 6, 549: 6, 546: 6, 5
20: 6, 515: 6, 512: 6, 503: 6, 489: 6, 477: 6, 473: 6, 434: 6, 428: 6, 423: 6, 416: 6, 409: 6, 404: 6,
403: 6, 398: 6, 394: 6, 392: 6, 391: 6, 380: 6, 376: 6, 373: 6, 371: 6, 360: 6, 358: 6, 354: 6, 349: 6,
343: 6, 339: 6, 334: 6, 327: 6, 309: 6, 301: 6, 266: 6, 244: 6, 1582: 5, 1348: 5, 1161: 5, 958: 5, 896:
5, 881: 5, 879: 5, 851: 5, 835: 5, 830: 5, 822: 5, 805: 5, 779: 5, 777: 5, 741: 5, 714: 5, 680: 5, 678:
5, 671: 5, 662: 5, 659: 5, 658: 5, 656: 5, 654: 5, 647: 5, 618: 5, 607: 5, 603: 5, 596: 5, 592: 5, 574:
5, 559: 5, 556: 5, 554: 5, 550: 5, 543: 5, 542: 5, 535: 5, 532: 5, 527: 5, 526: 5, 524: 5, 522: 5, 501:
5, 494: 5, 488: 5, 487: 5, 482: 5, 463: 5, 461: 5, 460: 5, 452: 5, 451: 5, 450: 5, 449: 5, 448: 5, 447:
5, 442: 5, 438: 5, 437: 5, 427: 5, 426: 5, 419: 5, 412: 5, 402: 5, 401: 5, 400: 5, 378: 5, 344: 5, 332:
5, 318: 5, 1835: 4, 1747: 4, 1705: 4, 1674: 4, 1524: 4, 1414: 4, 1327: 4, 1300: 4, 1285: 4, 1226: 4, 12
01: 4, 1167: 4, 1166: 4, 1160: 4, 1152: 4, 1130: 4, 1078: 4, 1072: 4, 1069: 4, 1039: 4, 1026: 4, 1013:
4, 985: 4, 965: 4, 964: 4, 922: 4, 911: 4, 902: 4, 891: 4, 889: 4, 872: 4, 865: 4, 861: 4, 814: 4, 813:
4, 793: 4, 780: 4, 778: 4, 773: 4, 764: 4, 763: 4, 758: 4, 742: 4, 730: 4, 724: 4, 718: 4, 700: 4, 694:
4, 682: 4, 674: 4, 666: 4, 663: 4, 657: 4, 655: 4, 646: 4, 636: 4, 629: 4, 622: 4, 610: 4, 605: 4, 599:
4, 593: 4, 571: 4, 564: 4, 558: 4, 557: 4, 553: 4, 544: 4, 540: 4, 538: 4, 536: 4, 531: 4, 530: 4, 529:
4, 519: 4, 518: 4, 517: 4, 510: 4, 509: 4, 505: 4, 504: 4, 502: 4, 497: 4, 481: 4, 478: 4, 475: 4, 459:
4, 445: 4, 443: 4, 436: 4, 432: 4, 429: 4, 390: 4, 384: 4, 369: 4, 365: 4, 335: 4, 325: 4, 4142: 3, 349
6: 3, 2934: 3, 2772: 3, 2724: 3, 2691: 3, 2607: 3, 2507: 3, 2316: 3, 2269: 3, 2161: 3, 2068: 3, 2049: 3
, 2017: 3, 2012: 3, 1988: 3, 1911: 3, 1867: 3, 1787: 3, 1779: 3, 1772: 3, 1692: 3, 1665: 3, 1637: 3, 16
02: 3, 1583: 3, 1555: 3, 1551: 3, 1520: 3, 1508: 3, 1507: 3, 1504: 3, 1501: 3, 1481: 3, 1468: 3, 1467:
3, 1447: 3, 1421: 3, 1420: 3, 1419: 3, 1415: 3, 1369: 3, 1361: 3, 1355: 3, 1350: 3, 1346: 3, 1321: 3, 1
313: 3, 1307: 3, 1267: 3, 1248: 3, 1243: 3, 1234: 3, 1231: 3, 1224: 3, 1216: 3, 1211: 3, 1199: 3, 1193:
3, 1177: 3, 1164: 3, 1153: 3, 1123: 3, 1118: 3, 1111: 3, 1109: 3, 1108: 3, 1099: 3, 1092: 3, 1088: 3, 1
086: 3, 1067: 3, 1063: 3, 1050: 3, 1044: 3, 1038: 3, 1031: 3, 1018: 3, 1015: 3, 1014: 3, 1008: 3, 996:
3, 992: 3, 990: 3, 987: 3, 986: 3, 981: 3, 980: 3, 971: 3, 967: 3, 966: 3, 945: 3, 939: 3, 935: 3, 926:
3, 918: 3, 917: 3, 916: 3, 898: 3, 882: 3, 867: 3, 863: 3, 862: 3, 858: 3, 856: 3, 853: 3, 852: 3, 850:
3, 847: 3, 836: 3, 834: 3, 831: 3, 829: 3, 827: 3, 826: 3, 824: 3, 823: 3, 806: 3, 804: 3, 794: 3, 787:
3, 786: 3, 782: 3, 781: 3, 771: 3, 768: 3, 762: 3, 754: 3, 750: 3, 746: 3, 745: 3, 727: 3, 726: 3, 706:
3, 705: 3, 699: 3, 696: 3, 693: 3, 691: 3, 688: 3, 685: 3, 681: 3, 676: 3, 675: 3, 670: 3, 665: 3, 653:
3, 645: 3, 641: 3, 640: 3, 638: 3, 637: 3, 627: 3, 626: 3, 625: 3, 623: 3, 621: 3, 619: 3, 611: 3, 604:
3, 589: 3, 584: 3, 583: 3, 580: 3, 577: 3, 576: 3, 569: 3, 565: 3, 562: 3, 552: 3, 545: 3, 534: 3, 523:
3, 516: 3, 511: 3, 507: 3, 492: 3, 490: 3, 486: 3, 485: 3, 483: 3, 480: 3, 479: 3, 471: 3, 467: 3, 466:
3, 464: 3, 458: 3, 440: 3, 430: 3, 414: 3, 408: 3, 397: 3, 387: 3, 386: 3, 346: 3, 298: 3, 11537: 2, 90
19: 2, 8128: 2, 6560: 2, 6213: 2, 6177: 2, 6154: 2, 6141: 2, 5317: 2, 5172: 2, 5041: 2, 4976: 2, 4889:
2, 4786: 2, 4694: 2, 4423: 2, 4411: 2, 4249: 2, 4234: 2, 4224: 2, 4121: 2, 4060: 2, 3748: 2, 3717: 2, 3
604: 2, 3513: 2, 3475: 2, 3467: 2, 3461: 2, 3458: 2, 3430: 2, 3418: 2, 3387: 2, 3350: 2, 3348: 2, 3346:
2, 3339: 2, 3319: 2, 3306: 2, 3297: 2, 3249: 2, 3228: 2, 3214: 2, 3191: 2, 3140: 2, 3107: 2, 2995: 2, 2
974: 2, 2933: 2, 2880: 2, 2864: 2, 2854: 2, 2792: 2, 2729: 2, 2645: 2, 2640: 2, 2587: 2, 2559: 2, 2542:
2, 2501: 2, 2482: 2, 2454: 2, 2405: 2, 2393: 2, 2386: 2, 2375: 2, 2323: 2, 2305: 2, 2296: 2, 2288: 2, 2
275: 2, 2253: 2, 2219: 2, 2205: 2, 2177: 2, 2167: 2, 2155: 2, 2153: 2, 2127: 2, 2123: 2, 2121: 2, 2120:
2, 2113: 2, 2089: 2, 2064: 2, 2062: 2, 2040: 2, 1989: 2, 1986: 2, 1977: 2, 1975: 2, 1966: 2, 1960: 2, 1
934: 2, 1915: 2, 1907: 2, 1906: 2, 1897: 2, 1895: 2, 1890: 2, 1885: 2, 1879: 2, 1873: 2, 1870: 2, 1860:
2, 1853: 2, 1852: 2, 1841: 2, 1840: 2, 1812: 2, 1806: 2, 1792: 2, 1778: 2, 1774: 2, 1766: 2, 1765: 2, 1
759: 2, 1736: 2, 1735: 2, 1731: 2, 1728: 2, 1723: 2, 1718: 2, 1709: 2, 1696: 2, 1691: 2, 1688: 2, 1685:
2, 1676: 2, 1672: 2, 1669: 2, 1660: 2, 1658: 2, 1649: 2, 1645: 2, 1634: 2, 1632: 2, 1630: 2, 1617: 2, 1
616: 2, 1612: 2, 1608: 2, 1599: 2, 1594: 2, 1585: 2, 1569: 2, 1566: 2, 1559: 2, 1553: 2, 1542: 2, 1529:
2, 1527: 2, 1518: 2, 1510: 2, 1506: 2, 1503: 2, 1484: 2, 1482: 2, 1473: 2, 1471: 2, 1465: 2, 1457: 2, 1
456: 2, 1454: 2, 1453: 2, 1444: 2, 1436: 2, 1433: 2, 1431: 2, 1430: 2, 1427: 2, 1423: 2, 1411: 2, 1410:
2, 1406: 2, 1405: 2, 1393: 2, 1392: 2, 1388: 2, 1382: 2, 1379: 2, 1378: 2, 1376: 2, 1367: 2, 1364: 2, 1
343: 2, 1335: 2, 1330: 2, 1323: 2, 1322: 2, 1315: 2, 1309: 2, 1306: 2, 1305: 2, 1297: 2, 1296: 2, 1295:
2, 1291: 2, 1290: 2, 1288: 2, 1281: 2, 1268: 2, 1263: 2, 1262: 2, 1256: 2, 1252: 2, 1251: 2, 1250: 2, 1
247: 2, 1246: 2, 1240: 2, 1235: 2, 1229: 2, 1228: 2, 1227: 2, 1219: 2, 1213: 2, 1210: 2, 1208: 2, 1203:
2, 1202: 2, 1200: 2, 1192: 2, 1191: 2, 1188: 2, 1187: 2, 1185: 2, 1183: 2, 1182: 2, 1176: 2, 1172: 2, 1
162: 2, 1157: 2, 1155: 2, 1154: 2, 1151: 2, 1148: 2, 1142: 2, 1141: 2, 1121: 2, 1117: 2, 1112: 2, 1097:
2, 1094: 2, 1091: 2, 1089: 2, 1085: 2, 1084: 2, 1083: 2, 1082: 2, 1075: 2, 1073: 2, 1071: 2, 1068: 2, 1
062: 2, 1059: 2, 1058: 2, 1056: 2, 1054: 2, 1053: 2, 1049: 2, 1047: 2, 1040: 2, 1034: 2, 1029: 2, 1028:
2, 1025: 2, 1020: 2, 1017: 2, 1012: 2, 1007: 2, 1002: 2, 1001: 2, 999: 2, 994: 2, 982: 2, 974: 2, 972:
2, 969: 2, 968: 2, 961: 2, 960: 2, 957: 2, 956: 2, 952: 2, 948: 2, 943: 2, 938: 2, 936: 2, 932: 2, 927:
2, 924: 2, 921: 2, 919: 2, 915: 2, 910: 2, 909: 2, 907: 2, 903: 2, 900: 2, 894: 2, 887: 2, 886: 2, 885:
2, 873: 2, 871: 2, 866: 2, 860: 2, 857: 2, 849: 2, 846: 2, 845: 2, 844: 2, 841: 2, 838: 2, 837: 2, 832:
2, 815: 2, 812: 2, 808: 2, 807: 2, 803: 2, 802: 2, 801: 2, 800: 2, 799: 2, 797: 2, 791: 2, 789: 2, 788:
2, 784: 2, 783: 2, 776: 2, 767: 2, 765: 2, 759: 2, 756: 2, 753: 2, 749: 2, 748: 2, 747: 2, 740: 2, 737:
2, 736: 2, 733: 2, 732: 2, 728: 2, 725: 2, 723: 2, 720: 2, 717: 2, 716: 2, 713: 2, 712: 2, 703: 2, 701:
2, 698: 2, 697: 2, 692: 2, 690: 2, 687: 2, 684: 2, 683: 2, 679: 2, 677: 2, 669: 2, 648: 2, 635: 2, 616:
2, 614: 2, 613: 2, 612: 2, 609: 2, 606: 2, 602: 2, 601: 2, 590: 2, 588: 2, 585: 2, 579: 2, 573: 2, 568:
2, 566: 2, 560: 2, 555: 2, 548: 2, 547: 2, 528: 2, 521: 2, 513: 2, 506: 2, 500: 2, 472: 2, 455: 2, 454:
2, 411: 2, 381: 2, 377: 2, 153412: 1, 120638: 1, 81199: 1, 68488: 1, 68348: 1, 68335: 1, 68072: 1, 6430
8: 1, 63823: 1, 54847: 1, 54574: 1, 50020: 1, 49262: 1, 47099: 1, 47042: 1, 44432: 1, 44077: 1, 42978:
1, 42869: 1, 41914: 1, 41373: 1, 41054: 1, 40817: 1, 39098: 1, 38982: 1, 37890: 1, 36808: 1, 36150: 1,
34933: 1, 34335: 1, 34232: 1, 34186: 1, 33743: 1, 33110: 1, 32370: 1, 31870: 1, 29969: 1, 29851: 1, 282
55: 1, 26937: 1, 26446: 1, 26194: 1, 25899: 1, 25518: 1, 25202: 1, 25056: 1, 24772: 1, 24645: 1, 24561:
1, 24337: 1, 24204: 1, 23838: 1, 23347: 1, 22704: 1, 22531: 1, 22088: 1, 22085: 1, 21764: 1, 21657: 1,
21252: 1, 20922: 1, 20548: 1, 20515: 1, 20299: 1, 20288: 1, 20230: 1, 19986: 1, 19967: 1, 19858: 1, 196
27: 1, 19624: 1, 19142: 1, 18928: 1, 18892: 1, 18679: 1, 18647: 1, 18503: 1, 18479: 1, 18450: 1, 18254:
1, 18101: 1, 18016: 1, 17898: 1, 17881: 1, 17838: 1, 17794: 1, 17793: 1, 17493: 1, 17308: 1, 17231: 1,
17205: 1, 17164: 1, 17127: 1, 17047: 1, 17013: 1, 16950: 1, 16777: 1, 16552: 1, 16373: 1, 16257: 1, 161
19: 1, 16106: 1, 15994: 1, 15978: 1, 15971: 1, 15934: 1, 15764: 1, 15683: 1, 15333: 1, 15327: 1, 15150:
1, 15005: 1, 14989: 1, 14827: 1, 14791: 1, 14434: 1, 14419: 1, 14262: 1, 14213: 1, 14087: 1, 14083: 1,

1, 15095: 1, 14888: 1, 14827: 1, 14791: 1, 14434: 1, 14412: 1, 14262: 1, 14212: 1, 14087: 1, 14083: 1,
13996: 1, 13984: 1, 13724: 1, 13712: 1, 13711: 1, 13460: 1, 13458: 1, 13453: 1, 13411: 1, 13398: 1, 133
27: 1, 13300: 1, 13237: 1, 13212: 1, 13185: 1, 13153: 1, 13118: 1, 13090: 1, 13074: 1, 12728: 1, 12709:
1, 12569: 1, 12536: 1, 12528: 1, 12517: 1, 12460: 1, 12457: 1, 12406: 1, 12403: 1, 12359: 1, 12357: 1,
12321: 1, 12304: 1, 12255: 1, 12218: 1, 12190: 1, 12140: 1, 12104: 1, 12098: 1, 12058: 1, 12043: 1, 120
40: 1, 11999: 1, 11951: 1, 11833: 1, 11763: 1, 11755: 1, 11732: 1, 11726: 1, 11592: 1, 11549: 1, 11540:
1, 11433: 1, 11410: 1, 11337: 1, 11269: 1, 11230: 1, 11214: 1, 11209: 1, 11205: 1, 10974: 1, 10956: 1,
10916: 1, 10887: 1, 10857: 1, 10816: 1, 10769: 1, 10740: 1, 10731: 1, 10718: 1, 10422: 1, 10367: 1, 103
53: 1, 10271: 1, 10264: 1, 10239: 1, 10207: 1, 10196: 1, 10191: 1, 10088: 1, 10057: 1, 10050: 1, 10041:
1, 9993: 1, 9962: 1, 9941: 1, 9885: 1, 9824: 1, 9797: 1, 9748: 1, 9723: 1, 9655: 1, 9620: 1, 9594: 1, 9
542: 1, 9524: 1, 9505: 1, 9490: 1, 9482: 1, 9477: 1, 9414: 1, 9355: 1, 9349: 1, 9348: 1, 9344: 1, 9212:
1, 9193: 1, 9125: 1, 9117: 1, 9089: 1, 9068: 1, 9037: 1, 9030: 1, 8986: 1, 8949: 1, 8945: 1, 8840: 1, 8
836: 1, 8793: 1, 8787: 1, 8771: 1, 8760: 1, 8712: 1, 8698: 1, 8584: 1, 8576: 1, 8564: 1, 8563: 1, 8562:
1, 8531: 1, 8488: 1, 8440: 1, 8419: 1, 8382: 1, 8376: 1, 8346: 1, 8338: 1, 8326: 1, 8314: 1, 8237: 1, 8
199: 1, 8183: 1, 8174: 1, 8148: 1, 8147: 1, 8098: 1, 8083: 1, 8066: 1, 8065: 1, 8012: 1, 8011: 1, 7998:
1, 7972: 1, 7937: 1, 7932: 1, 7923: 1, 7918: 1, 7917: 1, 7908: 1, 7891: 1, 7878: 1, 7829: 1, 7827: 1, 7
810: 1, 7804: 1, 7783: 1, 7761: 1, 7736: 1, 7727: 1, 7713: 1, 7668: 1, 7663: 1, 7649: 1, 7619: 1, 7606:
1, 7599: 1, 7576: 1, 7569: 1, 7556: 1, 7546: 1, 7495: 1, 7483: 1, 7440: 1, 7407: 1, 7364: 1, 7348: 1, 7
303: 1, 7292: 1, 7287: 1, 7256: 1, 7248: 1, 7226: 1, 7210: 1, 7192: 1, 7191: 1, 7163: 1, 7158: 1, 7156:
1, 7153: 1, 7151: 1, 7145: 1, 7114: 1, 7108: 1, 7104: 1, 7082: 1, 7053: 1, 7052: 1, 7049: 1, 7021: 1, 7
017: 1, 7012: 1, 7006: 1, 6975: 1, 6949: 1, 6938: 1, 6907: 1, 6904: 1, 6878: 1, 6869: 1, 6860: 1, 6852:
1, 6829: 1, 6813: 1, 6808: 1, 6801: 1, 6764: 1, 6763: 1, 6758: 1, 6740: 1, 6735: 1, 6727: 1, 6717: 1, 6
707: 1, 6693: 1, 6684: 1, 6643: 1, 6537: 1, 6530: 1, 6517: 1, 6489: 1, 6476: 1, 6459: 1, 6443: 1, 6440:
1, 6421: 1, 6409: 1, 6353: 1, 6322: 1, 6272: 1, 6265: 1, 6243: 1, 6208: 1, 6191: 1, 6188: 1, 6179: 1, 6
175: 1, 6170: 1, 6156: 1, 6134: 1, 6112: 1, 6096: 1, 6090: 1, 6088: 1, 6057: 1, 6056: 1, 6053: 1, 6048:
1, 6028: 1, 6027: 1, 6020: 1, 6012: 1, 5993: 1, 5990: 1, 5989: 1, 5964: 1, 5938: 1, 5937: 1, 5926: 1, 5
889: 1, 5876: 1, 5868: 1, 5852: 1, 5849: 1, 5837: 1, 5817: 1, 5797: 1, 5794: 1, 5793: 1, 5742: 1, 5727:
1, 5681: 1, 5676: 1, 5671: 1, 5668: 1, 5667: 1, 5650: 1, 5641: 1, 5618: 1, 5609: 1, 5608: 1, 5597: 1, 5
594: 1, 5593: 1, 5592: 1, 5590: 1, 5564: 1, 5553: 1, 5551: 1, 5539: 1, 5528: 1, 5517: 1, 5485: 1, 5474:
1, 5466: 1, 5454: 1, 5442: 1, 5438: 1, 5437: 1, 5436: 1, 5433: 1, 5418: 1, 5333: 1, 5329: 1, 5314: 1, 5
313: 1, 5305: 1, 5284: 1, 5282: 1, 5279: 1, 5274: 1, 5264: 1, 5263: 1, 5231: 1, 5221: 1, 5214: 1, 5181:
1, 5162: 1, 5152: 1, 5138: 1, 5137: 1, 5129: 1, 5128: 1, 5114: 1, 5112: 1, 5094: 1, 5090: 1, 5085: 1, 5
082: 1, 5068: 1, 5064: 1, 5047: 1, 5028: 1, 5026: 1, 5010: 1, 5006: 1, 5000: 1, 4989: 1, 4985: 1, 4974:
1, 4965: 1, 4953: 1, 4950: 1, 4947: 1, 4935: 1, 4920: 1, 4907: 1, 4899: 1, 4885: 1, 4861: 1, 4851: 1, 4
850: 1, 4843: 1, 4827: 1, 4823: 1, 4822: 1, 4817: 1, 4792: 1, 4779: 1, 4764: 1, 4761: 1, 4755: 1, 4750:
1, 4746: 1, 4743: 1, 4736: 1, 4733: 1, 4693: 1, 4676: 1, 4659: 1, 4646: 1, 4632: 1, 4627: 1, 4613: 1, 4
605: 1, 4585: 1, 4584: 1, 4569: 1, 4567: 1, 4566: 1, 4562: 1, 4551: 1, 4547: 1, 4545: 1, 4534: 1, 4530:
1, 4515: 1, 4512: 1, 4505: 1, 4504: 1, 4495: 1, 4476: 1, 4469: 1, 4454: 1, 4446: 1, 4445: 1, 4437: 1, 4
412: 1, 4408: 1, 4405: 1, 4394: 1, 4389: 1, 4384: 1, 4373: 1, 4372: 1, 4371: 1, 4363: 1, 4362: 1, 4345:
1, 4343: 1, 4336: 1, 4333: 1, 4331: 1, 4310: 1, 4309: 1, 4307: 1, 4299: 1, 4288: 1, 4282: 1, 4270: 1, 4
261: 1, 4260: 1, 4259: 1, 4248: 1, 4236: 1, 4233: 1, 4223: 1, 4222: 1, 4215: 1, 4209: 1, 4208: 1, 4205:
1, 4198: 1, 4171: 1, 4162: 1, 4158: 1, 4156: 1, 4152: 1, 4141: 1, 4134: 1, 4131: 1, 4130: 1, 4128: 1, 4
114: 1, 4113: 1, 4109: 1, 4105: 1, 4080: 1, 4075: 1, 4074: 1, 4070: 1, 4053: 1, 4038: 1, 4033: 1, 4031:
1, 4030: 1, 4028: 1, 4024: 1, 4011: 1, 4005: 1, 3999: 1, 3980: 1, 3979: 1, 3968: 1, 3963: 1, 3957: 1, 3
955: 1, 3941: 1, 3940: 1, 3930: 1, 3929: 1, 3926: 1, 3925: 1, 3924: 1, 3923: 1, 3920: 1, 3916: 1, 3903:
1, 3895: 1, 3879: 1, 3878: 1, 3874: 1, 3868: 1, 3867: 1, 3865: 1, 3854: 1, 3851: 1, 3840: 1, 3833: 1, 3
823: 1, 3819: 1, 3804: 1, 3801: 1, 3800: 1, 3785: 1, 3779: 1, 3778: 1, 3766: 1, 3762: 1, 3758: 1, 3756:
1, 3755: 1, 3744: 1, 3738: 1, 3732: 1, 3715: 1, 3712: 1, 3707: 1, 3694: 1, 3691: 1, 3677: 1, 3671: 1, 3
665: 1, 3664: 1, 3651: 1, 3644: 1, 3642: 1, 3641: 1, 3640: 1, 3639: 1, 3635: 1, 3633: 1, 3631: 1, 3628:
1, 3626: 1, 3625: 1, 3621: 1, 3620: 1, 3617: 1, 3613: 1, 3611: 1, 3608: 1, 3603: 1, 3599: 1, 3597: 1, 3
591: 1, 3589: 1, 3588: 1, 3587: 1, 3580: 1, 3576: 1, 3567: 1, 3562: 1, 3560: 1, 3558: 1, 3555: 1, 3554:
1, 3553: 1, 3546: 1, 3545: 1, 3532: 1, 3519: 1, 3516: 1, 3510: 1, 3508: 1, 3494: 1, 3493: 1, 3492: 1, 3
485: 1, 3484: 1, 3480: 1, 3474: 1, 3471: 1, 3460: 1, 3456: 1, 3451: 1, 3445: 1, 3417: 1, 3410: 1, 3404:
1, 3391: 1, 3388: 1, 3386: 1, 3379: 1, 3372: 1, 3370: 1, 3369: 1, 3364: 1, 3362: 1, 3359: 1, 3354: 1, 3
349: 1, 3345: 1, 3327: 1, 3323: 1, 3321: 1, 3314: 1, 3313: 1, 3311: 1, 3304: 1, 3299: 1, 3296: 1, 3293:
1, 3289: 1, 3286: 1, 3267: 1, 3261: 1, 3260: 1, 3255: 1, 3252: 1, 3251: 1, 3245: 1, 3241: 1, 3238: 1, 3
232: 1, 3230: 1, 3221: 1, 3216: 1, 3213: 1, 3200: 1, 3198: 1, 3197: 1, 3195: 1, 3179: 1, 3178: 1, 3177:
1, 3174: 1, 3169: 1, 3167: 1, 3165: 1, 3157: 1, 3154: 1, 3139: 1, 3135: 1, 3129: 1, 3127: 1, 3125: 1, 3
119: 1, 3118: 1, 3113: 1, 3108: 1, 3099: 1, 3098: 1, 3096: 1, 3095: 1, 3090: 1, 3078: 1, 3077: 1, 3076:
1, 3074: 1, 3072: 1, 3059: 1, 3053: 1, 3051: 1, 3043: 1, 3039: 1, 3035: 1, 3031: 1, 3028: 1, 3027: 1, 3
020: 1, 3017: 1, 3016: 1, 3011: 1, 3003: 1, 2998: 1, 2990: 1, 2983: 1, 2979: 1, 2976: 1, 2964: 1, 2960:
1, 2959: 1, 2951: 1, 2949: 1, 2940: 1, 2935: 1, 2926: 1, 2924: 1, 2914: 1, 2913: 1, 2908: 1, 2894: 1, 2
892: 1, 2887: 1, 2885: 1, 2881: 1, 2869: 1, 2853: 1, 2851: 1, 2850: 1, 2846: 1, 2829: 1, 2814: 1, 2812:
1, 2809: 1, 2802: 1, 2798: 1, 2797: 1, 2776: 1, 2775: 1, 2773: 1, 2763: 1, 2756: 1, 2755: 1, 2744: 1, 2
743: 1, 2731: 1, 2725: 1, 2719: 1, 2717: 1, 2716: 1, 2715: 1, 2713: 1, 2712: 1, 2710: 1, 2709: 1, 2703:
1, 2700: 1, 2697: 1, 2696: 1, 2693: 1, 2689: 1, 2683: 1, 2682: 1, 2681: 1, 2680: 1, 2679: 1, 2678: 1, 2
674: 1, 2668: 1, 2664: 1, 2663: 1, 2660: 1, 2657: 1, 2652: 1, 2636: 1, 2633: 1, 2631: 1, 2628: 1, 2626:
1, 2624: 1, 2620: 1, 2617: 1, 2615: 1, 2613: 1, 2612: 1, 2606: 1, 2603: 1, 2596: 1, 2593: 1, 2586: 1, 2
580: 1, 2578: 1, 2577: 1, 2576: 1, 2575: 1, 2569: 1, 2565: 1, 2564: 1, 2560: 1, 2558: 1, 2556: 1, 2551:
1, 2538: 1, 2533: 1, 2529: 1, 2526: 1, 2513: 1, 2512: 1, 2509: 1, 2508: 1, 2506: 1, 2503: 1, 2494: 1, 2
491: 1, 2490: 1, 2483: 1, 2481: 1, 2479: 1, 2477: 1, 2476: 1, 2473: 1, 2467: 1, 2463: 1, 2462: 1, 2460:
1, 2458: 1, 2457: 1, 2451: 1, 2444: 1, 2440: 1, 2439: 1, 2438: 1, 2431: 1, 2430: 1, 2428: 1, 2426: 1, 2
425: 1, 2424: 1, 2423: 1, 2421: 1, 2418: 1, 2417: 1, 2409: 1, 2399: 1, 2396: 1, 2394: 1, 2383: 1, 2382:
1, 2378: 1, 2377: 1, 2372: 1, 2368: 1, 2366: 1, 2364: 1, 2362: 1, 2350: 1, 2349: 1, 2348: 1, 2346: 1, 2
342: 1, 2338: 1, 2337: 1, 2334: 1, 2333: 1, 2332: 1, 2329: 1, 2328: 1, 2327: 1, 2324: 1, 2320: 1, 2319:
1, 2318: 1, 2317: 1, 2312: 1, 2303: 1, 2301: 1, 2298: 1, 2297: 1, 2293: 1, 2292: 1, 2287: 1, 2286: 1, 2
280: 1, 2278: 1, 2268: 1, 2267: 1, 2266: 1, 2265: 1, 2263: 1, 2260: 1, 2259: 1, 2255: 1, 2254: 1, 2250:

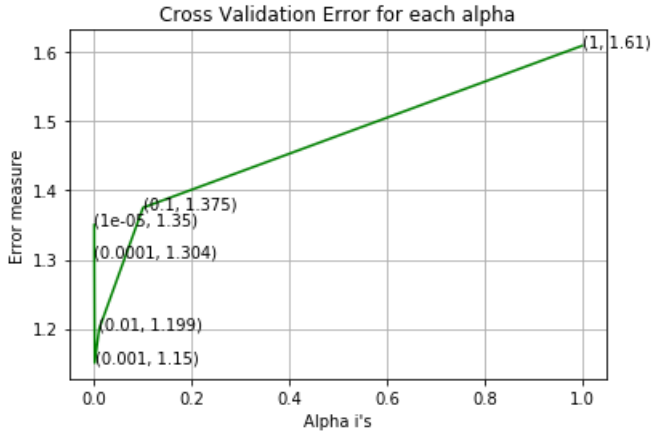
1, 2247: 1, 2246: 1, 2245: 1, 2235: 1, 2234: 1, 2228: 1, 2227: 1, 2226: 1, 2224: 1, 2223: 1, 2214: 1, 2
213: 1, 2210: 1, 2207: 1, 2206: 1, 2199: 1, 2193: 1, 2190: 1, 2189: 1, 2186: 1, 2185: 1, 2178: 1, 2173:
1, 2168: 1, 2165: 1, 2164: 1, 2158: 1, 2154: 1, 2151: 1, 2147: 1, 2141: 1, 2139: 1, 2138: 1, 2133: 1, 2
131: 1, 2130: 1, 2129: 1, 2112: 1, 2110: 1, 2107: 1, 2105: 1, 2101: 1, 2096: 1, 2095: 1, 2091: 1, 2088:
1, 2086: 1, 2083: 1, 2082: 1, 2076: 1, 2075: 1, 2073: 1, 2063: 1, 2060: 1, 2058: 1, 2056: 1, 2050: 1, 2
045: 1, 2042: 1, 2041: 1, 2037: 1, 2028: 1, 2027: 1, 2024: 1, 2022: 1, 2021: 1, 2019: 1, 2015: 1, 2013:
1, 2010: 1, 2008: 1, 2007: 1, 2006: 1, 2001: 1, 2000: 1, 1998: 1, 1997: 1, 1994: 1, 1993: 1, 1981: 1, 1
979: 1, 1978: 1, 1974: 1, 1970: 1, 1969: 1, 1968: 1, 1967: 1, 1964: 1, 1963: 1, 1962: 1, 1958: 1, 1957:
1, 1956: 1, 1953: 1, 1952: 1, 1950: 1, 1949: 1, 1946: 1, 1942: 1, 1940: 1, 1938: 1, 1937: 1, 1935: 1, 1
932: 1, 1931: 1, 1928: 1, 1925: 1, 1922: 1, 1921: 1, 1919: 1, 1918: 1, 1913: 1, 1910: 1, 1905: 1, 1904:
1, 1903: 1, 1900: 1, 1898: 1, 1892: 1, 1891: 1, 1888: 1, 1886: 1, 1883: 1, 1881: 1, 1880: 1, 1878: 1, 1
872: 1, 1869: 1, 1864: 1, 1861: 1, 1855: 1, 1846: 1, 1845: 1, 1844: 1, 1836: 1, 1833: 1, 1830: 1, 1827:
1, 1826: 1, 1825: 1, 1823: 1, 1822: 1, 1821: 1, 1817: 1, 1816: 1, 1811: 1, 1808: 1, 1807: 1, 1805: 1, 1
801: 1, 1800: 1, 1799: 1, 1795: 1, 1789: 1, 1788: 1, 1782: 1, 1781: 1, 1775: 1, 1773: 1, 1771: 1, 1770:
1, 1769: 1, 1761: 1, 1760: 1, 1755: 1, 1753: 1, 1752: 1, 1750: 1, 1748: 1, 1746: 1, 1745: 1, 1744: 1, 1
741: 1, 1740: 1, 1738: 1, 1737: 1, 1733: 1, 1730: 1, 1729: 1, 1727: 1, 1726: 1, 1725: 1, 1715: 1, 1714:
1, 1711: 1, 1710: 1, 1708: 1, 1707: 1, 1704: 1, 1703: 1, 1702: 1, 1701: 1, 1699: 1, 1697: 1, 1695: 1, 1
689: 1, 1687: 1, 1682: 1, 1678: 1, 1673: 1, 1671: 1, 1666: 1, 1663: 1, 1659: 1, 1656: 1, 1655: 1, 1652:
1, 1650: 1, 1648: 1, 1644: 1, 1643: 1, 1641: 1, 1639: 1, 1638: 1, 1635: 1, 1627: 1, 1626: 1, 1625: 1, 1
624: 1, 1614: 1, 1609: 1, 1606: 1, 1604: 1, 1600: 1, 1598: 1, 1591: 1, 1590: 1, 1587: 1, 1580: 1, 1579:
1, 1577: 1, 1576: 1, 1575: 1, 1573: 1, 1572: 1, 1570: 1, 1567: 1, 1564: 1, 1560: 1, 1552: 1, 1550: 1, 1
549: 1, 1545: 1, 1544: 1, 1540: 1, 1536: 1, 1535: 1, 1531: 1, 1528: 1, 1525: 1, 1521: 1, 1515: 1, 1512:
1, 1505: 1, 1502: 1, 1500: 1, 1499: 1, 1498: 1, 1496: 1, 1493: 1, 1492: 1, 1491: 1, 1490: 1, 1488: 1, 1
486: 1, 1485: 1, 1478: 1, 1475: 1, 1474: 1, 1462: 1, 1460: 1, 1459: 1, 1458: 1, 1455: 1, 1451: 1, 1449:
1, 1445: 1, 1443: 1, 1440: 1, 1439: 1, 1434: 1, 1429: 1, 1417: 1, 1409: 1, 1408: 1, 1407: 1, 1401: 1, 1
398: 1, 1395: 1, 1391: 1, 1386: 1, 1384: 1, 1383: 1, 1381: 1, 1375: 1, 1374: 1, 1363: 1, 1360: 1, 1359:
1, 1357: 1, 1356: 1, 1354: 1, 1349: 1, 1345: 1, 1344: 1, 1342: 1, 1341: 1, 1340: 1, 1338: 1, 1336: 1, 1
331: 1, 1328: 1, 1325: 1, 1324: 1, 1320: 1, 1318: 1, 1314: 1, 1308: 1, 1302: 1, 1301: 1, 1298: 1, 1294:
1, 1293: 1, 1292: 1, 1282: 1, 1280: 1, 1279: 1, 1278: 1, 1277: 1, 1275: 1, 1273: 1, 1271: 1, 1270: 1, 1
269: 1, 1266: 1, 1264: 1, 1261: 1, 1260: 1, 1258: 1, 1257: 1, 1255: 1, 1253: 1, 1249: 1, 1245: 1, 1244:
1, 1242: 1, 1239: 1, 1237: 1, 1236: 1, 1233: 1, 1230: 1, 1225: 1, 1223: 1, 1222: 1, 1221: 1, 1220: 1, 1
215: 1, 1214: 1, 1212: 1, 1209: 1, 1207: 1, 1204: 1, 1198: 1, 1195: 1, 1190: 1, 1189: 1, 1184: 1, 1175:
1, 1174: 1, 1173: 1, 1171: 1, 1170: 1, 1163: 1, 1156: 1, 1147: 1, 1146: 1, 1144: 1, 1140: 1, 1137: 1, 1
134: 1, 1132: 1, 1131: 1, 1128: 1, 1127: 1, 1125: 1, 1116: 1, 1115: 1, 1113: 1, 1107: 1, 1105: 1, 1104:
1, 1102: 1, 1101: 1, 1100: 1, 1098: 1, 1095: 1, 1093: 1, 1090: 1, 1087: 1, 1077: 1, 1076: 1, 1074: 1, 1
066: 1, 1061: 1, 1057: 1, 1055: 1, 1052: 1, 1051: 1, 1048: 1, 1046: 1, 1045: 1, 1041: 1, 1036: 1, 1033:
1, 1032: 1, 1030: 1, 1024: 1, 1023: 1, 1022: 1, 1021: 1, 1016: 1, 1009: 1, 1006: 1, 1003: 1, 998: 1, 99
7: 1, 995: 1, 993: 1, 991: 1, 989: 1, 988: 1, 983: 1, 978: 1, 973: 1, 970: 1, 951: 1, 950: 1, 949: 1, 9
47: 1, 946: 1, 944: 1, 942: 1, 941: 1, 937: 1, 934: 1, 933: 1, 929: 1, 928: 1, 925: 1, 923: 1, 920: 1,
914: 1, 913: 1, 908: 1, 906: 1, 905: 1, 901: 1, 897: 1, 893: 1, 892: 1, 884: 1, 883: 1, 878: 1, 877: 1,
876: 1, 870: 1, 868: 1, 859: 1, 855: 1, 854: 1, 848: 1, 843: 1, 842: 1, 840: 1, 828: 1, 825: 1, 820: 1,
818: 1, 816: 1, 811: 1, 810: 1, 809: 1, 798: 1, 796: 1, 795: 1, 775: 1, 770: 1, 769: 1, 766: 1, 761: 1,
760: 1, 752: 1, 744: 1, 743: 1, 739: 1, 734: 1, 731: 1, 722: 1, 711: 1, 710: 1, 709: 1, 704: 1, 702: 1,
695: 1, 667: 1, 664: 1, 660: 1, 649: 1, 634: 1, 632: 1, 631: 1, 630: 1, 624: 1, 617: 1, 615: 1, 591: 1,
578: 1, 575: 1, 572: 1, 570: 1, 484: 1, 474: 1, 456: 1})

In [47]:

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

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

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

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


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

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

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

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


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

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

misclassified_pt = np.count_nonzero((sig_clf.predict(cv_text_feature_onehotCoding)- y_cv))/y_cv.shape[0
]

ittr+=1
df.loc[ittr]= ['LR', 'Variance','onehot','BOW',"alpha {0}".format(alpha[best_alpha]),tr_loss,cv_loss,te
st_loss,misclassified_pt*100,'YES']
```

```
For values of alpha =  1e-05 The log loss is: 1.350196392824388
For values of alpha =  0.0001 The log loss is: 1.3041677340084359
For values of alpha =  0.001 The log loss is: 1.1502328932946062
For values of alpha =  0.01 The log loss is: 1.1993038815002164
For values of alpha =  0.1 The log loss is: 1.3751945330463895
For values of alpha =  1 The log loss is: 1.609852462670334
```




```
For values of best alpha =  0.001 The train log loss is: 0.7461926772473694
For values of best alpha =  0.001 The cross validation log loss is: 1.1502328932946062
For values of best alpha =  0.001 The test log loss is: 1.242717238436523
```

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

Ans. Yes, it seems like!

In [48]:

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

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

In [49]:

```
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")
```

```
96.098 % of word of test data appeared in train data
98.847 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [47]:

```
#Data preparation for ML models.

#Misc. functionns for ML models


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

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

In [48]:

```
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 [49]:

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

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

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

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

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

Stacking the three types of features

In [53]:

```
# 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]]

#stacking for BOW
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCod
ing))
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))


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


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



#stacking for BOW
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_re
sponseCoding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variation_feature_respo
nseCoding))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_feature_responseCod
ing))

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 [55]:

```
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.shap
e)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 55358)
(number of data points * number of features) in test data =  (665, 55358)
(number of data points * number of features) in cross validation data = (532, 55358)
```

In [56]:

```
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.sh
ape)
```

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

4.1. Base Line Model

4.1.1. Naive Bayes

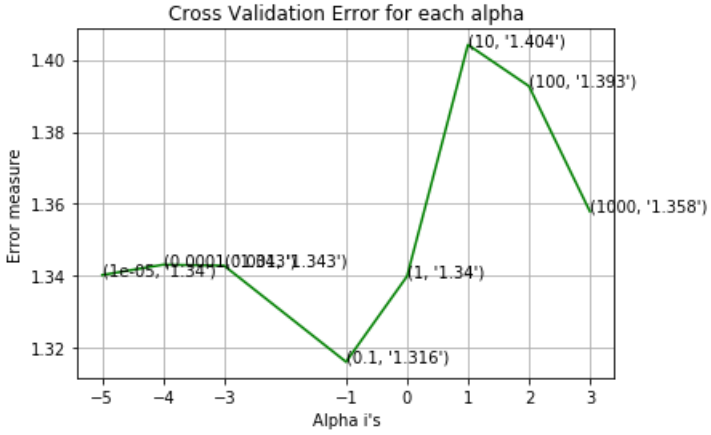
4.1.1.1. Hyper parameter tuning

In [57]:

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

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


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


alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
```

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

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


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


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

```
for alpha = 1e-05
Log Loss : 1.3402917774955008
for alpha = 0.0001
Log Loss : 1.343158775878964
for alpha = 0.001
Log Loss : 1.3428848399044193
for alpha = 0.1
Log Loss : 1.31621802255848
for alpha = 1
Log Loss : 1.3399257609743143
for alpha = 10
Log Loss : 1.4041470090830361
for alpha = 100
Log Loss : 1.3926678414138793
for alpha = 1000
Log Loss : 1.3579572733762608
```



```
For values of best alpha =  0.1 The train log loss is: 0.8676217056984667
For values of best alpha =  0.1 The cross validation log loss is: 1.31621802255848
For values of best alpha =  0.1 The test log loss is: 1.2884030587007582
```

4.1.1.2. Testing the model with best hyper paramters

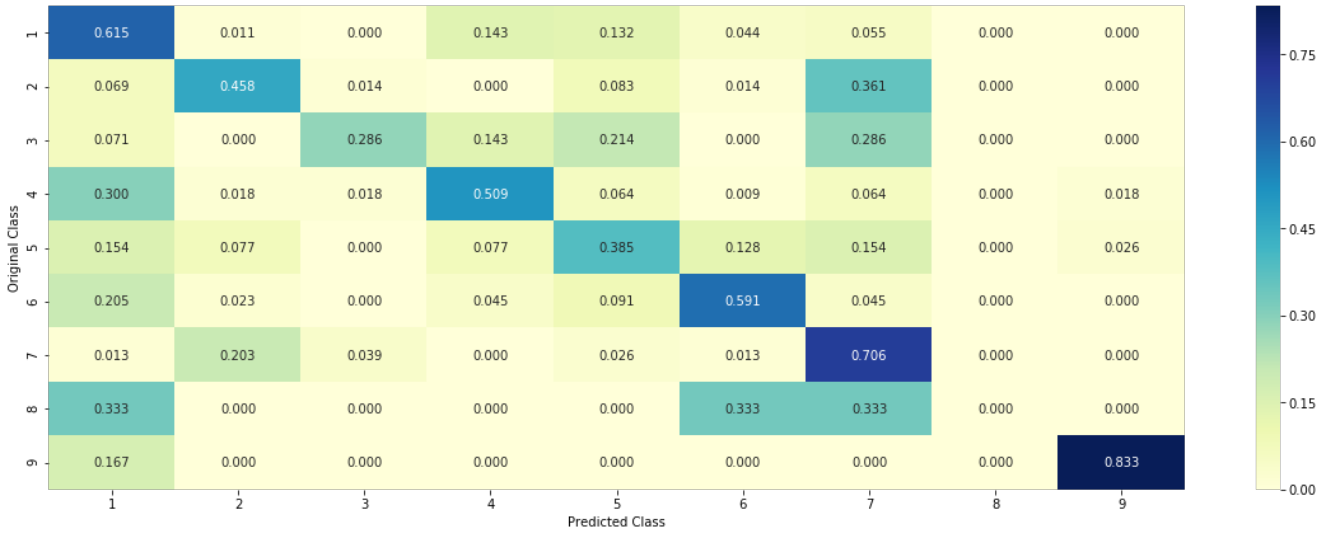
In [58]:

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

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


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

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

```
Log Loss : 1.31621802255848
Number of missclassified point : 0.43045112781954886
-------------------- Confusion matrix --------------------
```

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


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



In [59]:

```
ittr+=1
df.loc[ittr]= ['NB', 'ALL','onehot','BOW',"alpha {0}".format(alpha[best_alpha]),tr_loss,cv_loss,test_lo
ss,misclassified_pt*100,'YES']
```

4.1.1.3. Feature Importance, Correctly classified point

In [60]:

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

```
Predicted Class : 5
Predicted Class Probabilities: [[0.1076 0.0941 0.018  0.1412 0.4241 0.0503 0.1553 0.0056 0.0039]]
Actual Class : 6
```

```
Actual Class : 6
--------------------------------------------------
6 Text feature [assays] present in test data point [True]
8 Text feature [functional] present in test data point [True]
10 Text feature [variants] present in test data point [True]
17 Text feature [neutral] present in test data point [True]
18 Text feature [uncertain] present in test data point [True]
23 Text feature [predict] present in test data point [True]
24 Text feature [controls] present in test data point [True]
25 Text feature [variant] present in test data point [True]
26 Text feature [usefulness] present in test data point [True]
30 Text feature [based] present in test data point [True]
33 Text feature [likely] present in test data point [True]
39 Text feature [functionally] present in test data point [True]
40 Text feature [v1736a] present in test data point [True]
45 Text feature [sequence] present in test data point [True]
46 Text feature [assay] present in test data point [True]
50 Text feature [d1739g] present in test data point [True]
Out of the top  100  features  16 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [61]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.076  0.0665 0.0127 0.1    0.0334 0.0358 0.6689 0.004  0.0027]]
Actual Class : 7
--------------------------------------------------
18 Text feature [presence] present in test data point [True]
19 Text feature [kinase] present in test data point [True]
20 Text feature [downstream] present in test data point [True]
22 Text feature [inhibitor] present in test data point [True]
23 Text feature [well] present in test data point [True]
24 Text feature [cell] present in test data point [True]
25 Text feature [recently] present in test data point [True]
26 Text feature [contrast] present in test data point [True]
27 Text feature [compared] present in test data point [True]
28 Text feature [growth] present in test data point [True]
29 Text feature [factor] present in test data point [True]
30 Text feature [independent] present in test data point [True]
31 Text feature [shown] present in test data point [True]
32 Text feature [potential] present in test data point [True]
33 Text feature [treated] present in test data point [True]
34 Text feature [however] present in test data point [True]
35 Text feature [cells] present in test data point [True]
36 Text feature [showed] present in test data point [True]
37 Text feature [suggest] present in test data point [True]
38 Text feature [higher] present in test data point [True]
39 Text feature [also] present in test data point [True]
40 Text feature [expressing] present in test data point [True]
42 Text feature [similar] present in test data point [True]
43 Text feature [mutations] present in test data point [True]
44 Text feature [previously] present in test data point [True]
45 Text feature [10] present in test data point [True]
46 Text feature [addition] present in test data point [True]
47 Text feature [inhibition] present in test data point [True]
48 Text feature [observed] present in test data point [True]
49 Text feature [studies] present in test data point [True]
50 Text feature [increased] present in test data point [True]
51 Text feature [found] present in test data point [True]
52 Text feature [1a] present in test data point [True]
53 Text feature [may] present in test data point [True]
54 Text feature [concentrations] present in test data point [True]
```

```
55 Text feature [obtained] present in test data point [True]
56 Text feature [inhibitors] present in test data point [True]
58 Text feature [described] present in test data point [True]
59 Text feature [respectively] present in test data point [True]
60 Text feature [12] present in test data point [True]
61 Text feature [new] present in test data point [True]
62 Text feature [activated] present in test data point [True]
63 Text feature [due] present in test data point [True]
65 Text feature [reported] present in test data point [True]
66 Text feature [three] present in test data point [True]
67 Text feature [small] present in test data point [True]
68 Text feature [proliferation] present in test data point [True]
69 Text feature [including] present in test data point [True]
70 Text feature [3a] present in test data point [True]
71 Text feature [total] present in test data point [True]
72 Text feature [followed] present in test data point [True]
73 Text feature [interestingly] present in test data point [True]
74 Text feature [3b] present in test data point [True]
75 Text feature [without] present in test data point [True]
76 Text feature [using] present in test data point [True]
77 Text feature [report] present in test data point [True]
78 Text feature [fig] present in test data point [True]
79 Text feature [although] present in test data point [True]
80 Text feature [occur] present in test data point [True]
81 Text feature [confirmed] present in test data point [True]
82 Text feature [either] present in test data point [True]
83 Text feature [hours] present in test data point [True]
85 Text feature [mutation] present in test data point [True]
88 Text feature [recent] present in test data point [True]
89 Text feature [leading] present in test data point [True]
90 Text feature [identified] present in test data point [True]
91 Text feature [two] present in test data point [True]
93 Text feature [furthermore] present in test data point [True]
95 Text feature [increase] present in test data point [True]
96 Text feature [15] present in test data point [True]
97 Text feature [whereas] present in test data point [True]
98 Text feature [inhibited] present in test data point [True]
99 Text feature [mutant] present in test data point [True]
Out of the top  100  features  73 are present in query point
```

4.2. K Nearest Neighbour Classification

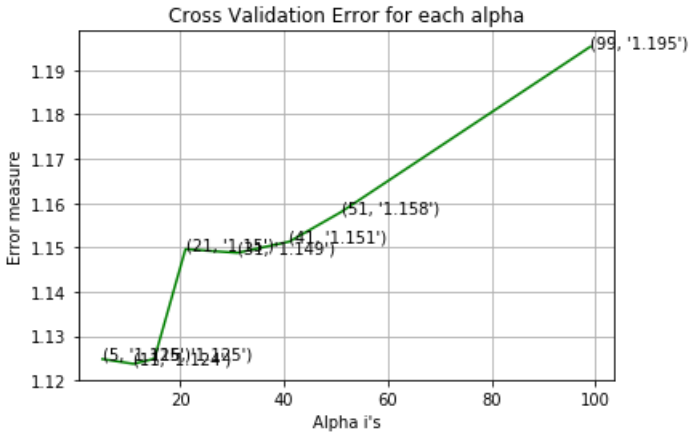
4.2.1. Hyper parameter tuning

In [62]:

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

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


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

```
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------


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

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


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



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

"""predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, pred
ict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_
cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predic
t_y, labels=clf.classes_, eps=1e-15))
"""
```

```
for alpha = 5
Log Loss : 1.124815602380621
for alpha = 11
Log Loss : 1.1237390019189601
for alpha = 15
Log Loss : 1.1248852577595658
for alpha = 21
Log Loss : 1.149601382096524
for alpha = 31
Log Loss : 1.1487765852173837
for alpha = 41
Log Loss : 1.1513006636705145
for alpha = 51
Log Loss : 1.1579870455213954
for alpha = 99
Log Loss : 1.195248383986407
```




```
For values of best alpha =  11 The train log loss is: 0.6355934476643952
For values of best alpha =  11 The cross validation log loss is: 1.1237390019189601
For values of best alpha =  11 The test log loss is: 1.0918146811112799
```

Out[62]:

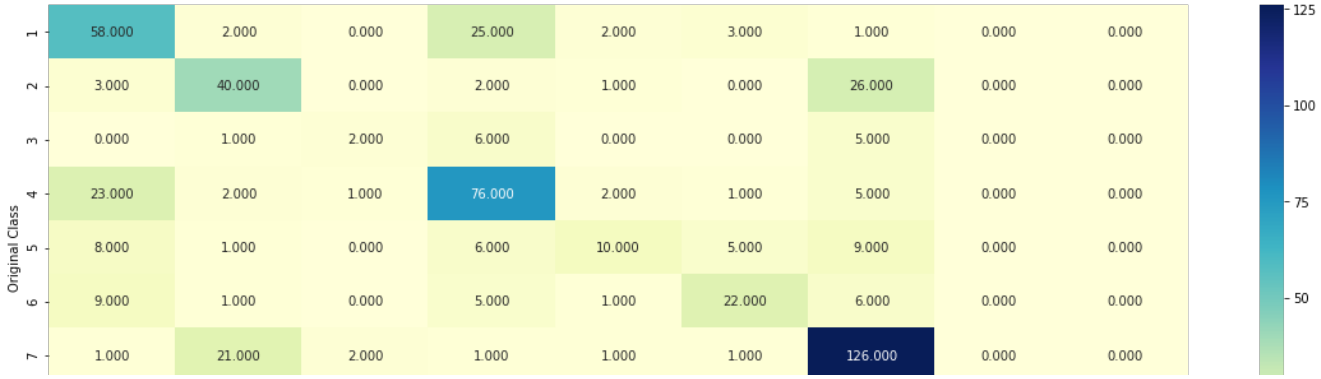
```
'predict_y = sig_clf.predict_proba(train_x_responseCoding)\nprint(\'For values of best alpha = \', alph
a[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))\n
predict_y = sig_clf.predict_proba(cv_x_responseCoding)\nprint(\'For values of best alpha = \', alpha[be
st_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15
))\npredict_y = sig_clf.predict_proba(test_x_responseCoding)\nprint(\'For values of best alpha = \', al
pha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))\n
'
```

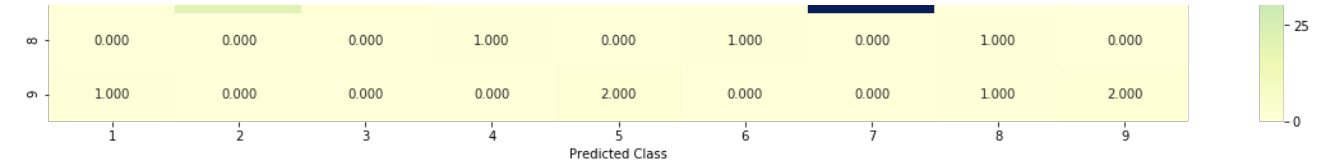
4.2.2. Testing the model with best hyper paramters

In [63]:

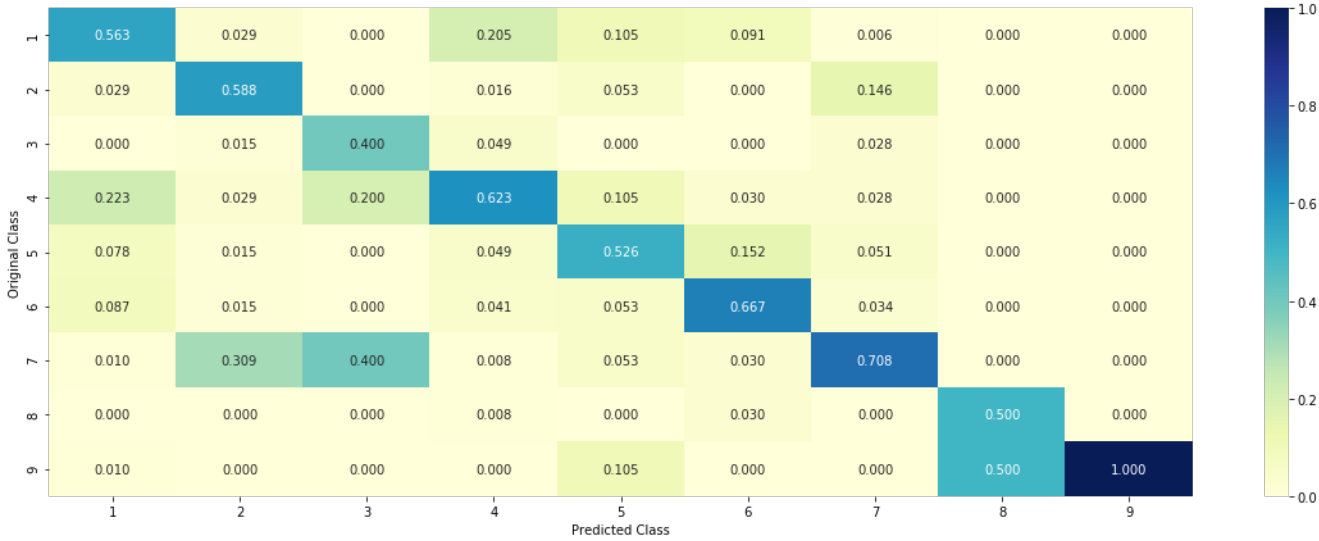
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.
neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski',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

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

```
Log loss : 1.1237390019189601
Number of mis-classified points : 0.36654135338345867
-------------------- Confusion matrix --------------------
```

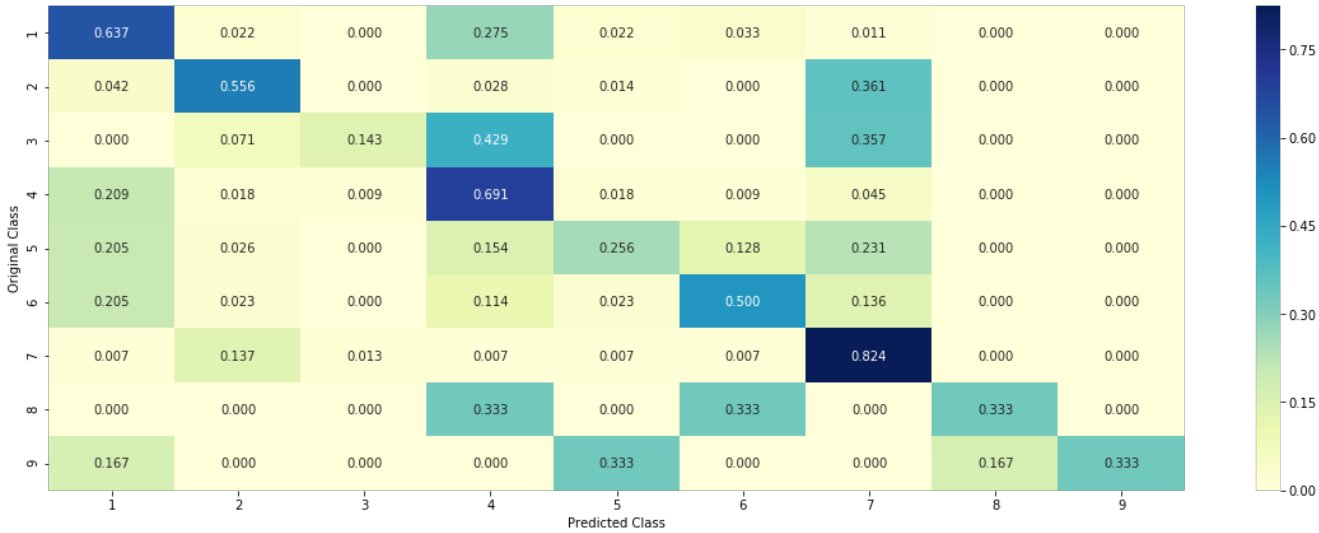





```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



In [64]:

```
ittr+=1
df.loc[ittr]= ['KNN', 'ALL','response','BOW',"alpha {0}".format(alpha[best_alpha]),tr_loss,cv_loss,test
_loss,misclassified_pt*100,'NO']
```

4.2.3.Sample Query point -1

In [65]:

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

test_point_index = 1
predicted_cls = sig_clf.predict(test_x_responseCoding[0].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
```

```
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), alpha[best_alpha])
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to classes",train_y[neig
hbors[1][0]])
print("Fequency 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 [5 6 1 4 5 5 6 1 6 6 1]
Fequency of nearest points : Counter({6: 4, 5: 3, 1: 3, 4: 1})
```

4.2.4. Sample Query Point-2

In [66]:

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

test_point_index = 100

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

```
Predicted Class : 7
Actual Class : 7
the k value for knn is 11 and the nearest neighbours of the test points belongs to classes [7 7 7 7 7 7
7 7 7 7 6]
Fequency of nearest points : Counter({7: 10, 6: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

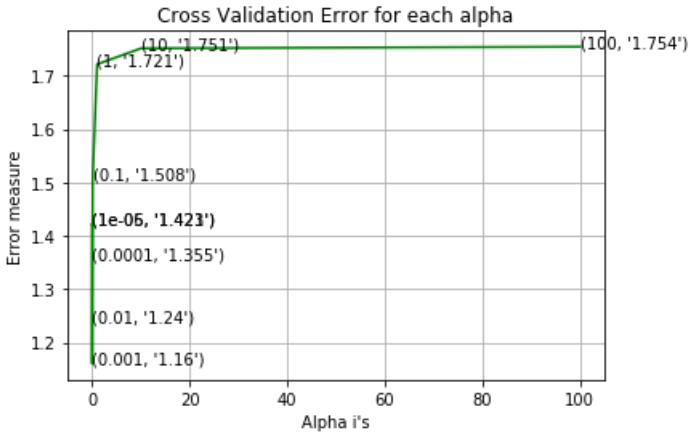
In [67]:

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

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

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


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

```
# 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_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

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

```
for alpha = 1e-06
Log Loss : 1.421276297650619
for alpha = 1e-05
Log Loss : 1.422557064611508
for alpha = 0.0001
Log Loss : 1.3547649964269222
for alpha = 0.001
Log Loss : 1.1597793089273998
for alpha = 0.01
Log Loss : 1.239614576061078
for alpha = 0.1
Log Loss : 1.5075770096749268
for alpha = 1
```

```
Log Loss : 1.7214313893808526
for alpha = 10
Log Loss : 1.7512758902346415
for alpha = 100
Log Loss : 1.754320034591301
```




```
For values of best alpha =  0.001 The train log loss is: 0.6227642994252047
For values of best alpha =  0.001 The cross validation log loss is: 1.1597793089273998
For values of best alpha =  0.001 The test log loss is: 1.1781370305207564
```

Out[67]:

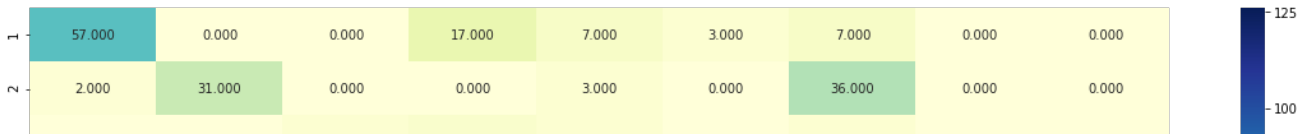
```
'predict_y = sig_clf.predict_proba(train_x_onehotCoding)\nprint(\'For values of best alpha = \', alpha[
best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))\npr
edict_y = sig_clf.predict_proba(cv_x_onehotCoding)\nprint(\'For values of best alpha = \',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

4.3.1.2. Testing the model with best hyper paramters

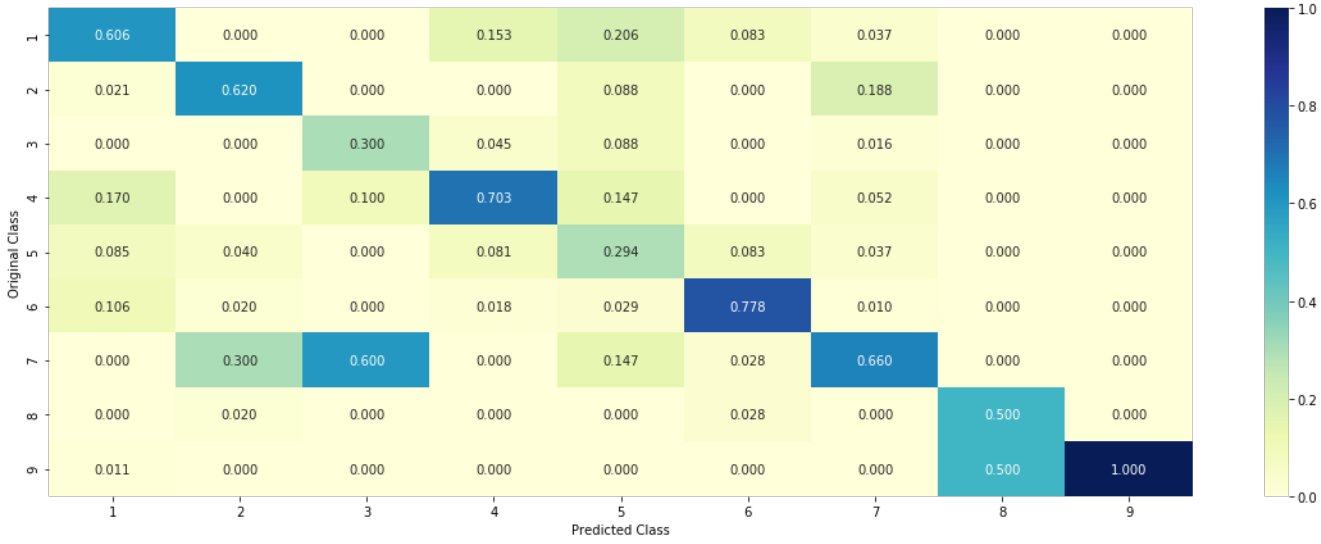
In [68]:

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

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

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

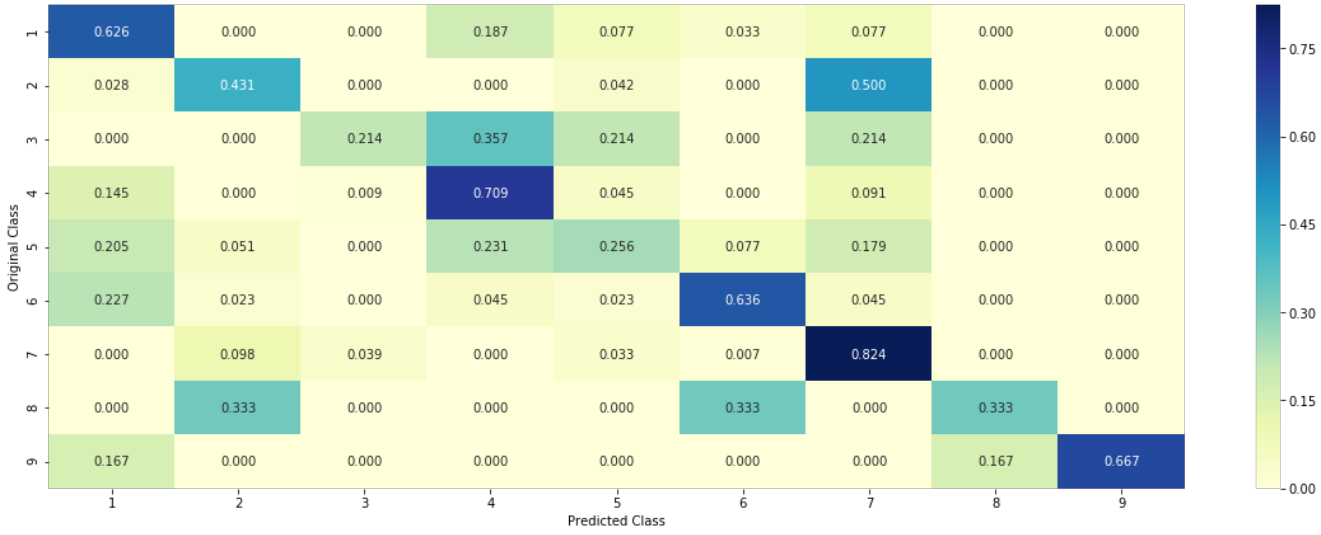
```
Log loss : 1.1597793089273998
Number of mis-classified points : 0.36466165413533835
-------------------- Confusion matrix --------------------
```



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



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



In [69]:

```
ittr+=1
df.loc[ittr]= ['LR-Balanced', 'ALL','onehot','BOW',"alpha {0}".format(alpha[best_alpha]),tr_loss,cv_los
s,test_loss,misclassified_pt*100,'YES']
```

4.3.1.3. Feature Importance

In [70]:

In [70]:

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

4.3.1.3.1. Correctly Classified point

In [71]:

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.2258 0.047  0.0037 0.0111 0.2869 0.3878 0.0306 0.0038 0.0033]]
Actual Class : 6
--------------------------------------------------
90 Text feature [v1804d] present in test data point [True]
261 Text feature [901] present in test data point [True]
368 Text feature [oophorectomy] 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 [72]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.015  0.065  0.0011 0.0168 0.0037 0.0017 0.8911 0.0045 0.0011]]
Actual Class : 7
--------------------------------------------------
```

```
97 Text feature [activated] present in test data point [True]
103 Text feature [receptor] present in test data point [True]
105 Text feature [subclinical] present in test data point [True]
128 Text feature [cysteine] present in test data point [True]
140 Text feature [swab] present in test data point [True]
141 Text feature [hyperplasia] present in test data point [True]
145 Text feature [ligand] present in test data point [True]
159 Text feature [technology] present in test data point [True]
170 Text feature [transformed] present in test data point [True]
187 Text feature [cylinders] present in test data point [True]
190 Text feature [750] present in test data point [True]
204 Text feature [murine] present in test data point [True]
209 Text feature [inhibited] present in test data point [True]
222 Text feature [extracellular] present in test data point [True]
228 Text feature [downstream] present in test data point [True]
247 Text feature [refractory] present in test data point [True]
250 Text feature [hazards] present in test data point [True]
262 Text feature [209k] present in test data point [True]
266 Text feature [attractive] present in test data point [True]
271 Text feature [oestrogen] present in test data point [True]
274 Text feature [neu] present in test data point [True]
292 Text feature [warrants] present in test data point [True]
325 Text feature [nsclcs] present in test data point [True]
345 Text feature [inhibitor] present in test data point [True]
414 Text feature [enhancing] present in test data point [True]
480 Text feature [to3] present in test data point [True]
484 Text feature [transformation] present in test data point [True]
493 Text feature [il] present in test data point [True]
Out of the top  500  features  28 are present in query point
```

4.3.2. Without Class balancing

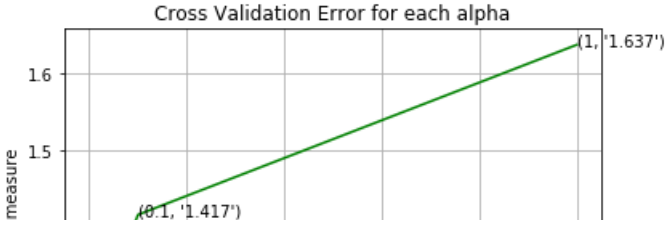
4.3.2.1. Hyper paramter tuning

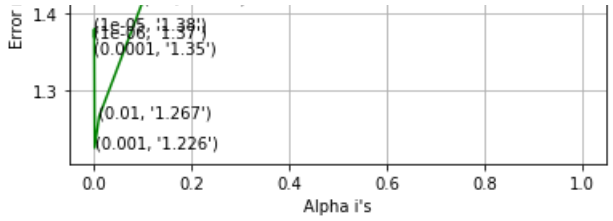
In [73]:

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

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

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



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

alpha = [10 ** x for x in range(-6, 1)]
```

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

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=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)
tr_loss = log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",tr_loss)
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
cv_loss = log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",cv_loss)
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
test_loss = log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",test_loss)

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

"""
```

```
for alpha = 1e-06
Log Loss : 1.370262847176083
for alpha = 1e-05
Log Loss : 1.3800761369767958
for alpha = 0.0001
Log Loss : 1.3502895559584402
for alpha = 0.001
Log Loss : 1.2260609393072237
for alpha = 0.01
Log Loss : 1.2666036745934437
for alpha = 0.1
Log Loss : 1.4168304691321894
for alpha = 1
Log Loss : 1.6372494373886533
```






```
For values of best alpha =  0.001 The train log loss is: 0.6212634777686672
For values of best alpha =  0.001 The cross validation log loss is: 1.2260609393072237
For values of best alpha =  0.001 The test log loss is: 1.2124093698352558
```

Out[73]:

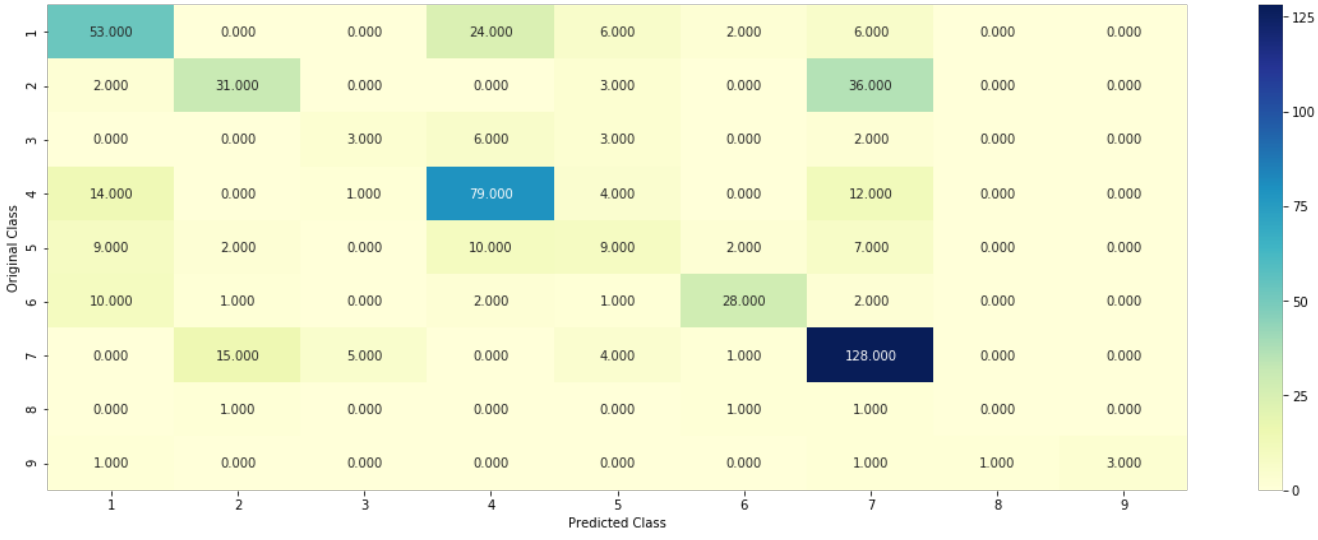
```
'\npredict_y = sig_clf.predict_proba(train_x_onehotCoding)\nprint(\'For values of best alpha = \', alph
a[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))\n
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)\nprint(\'For values of best alpha =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

4.3.2.2. Testing model with best hyper parameters

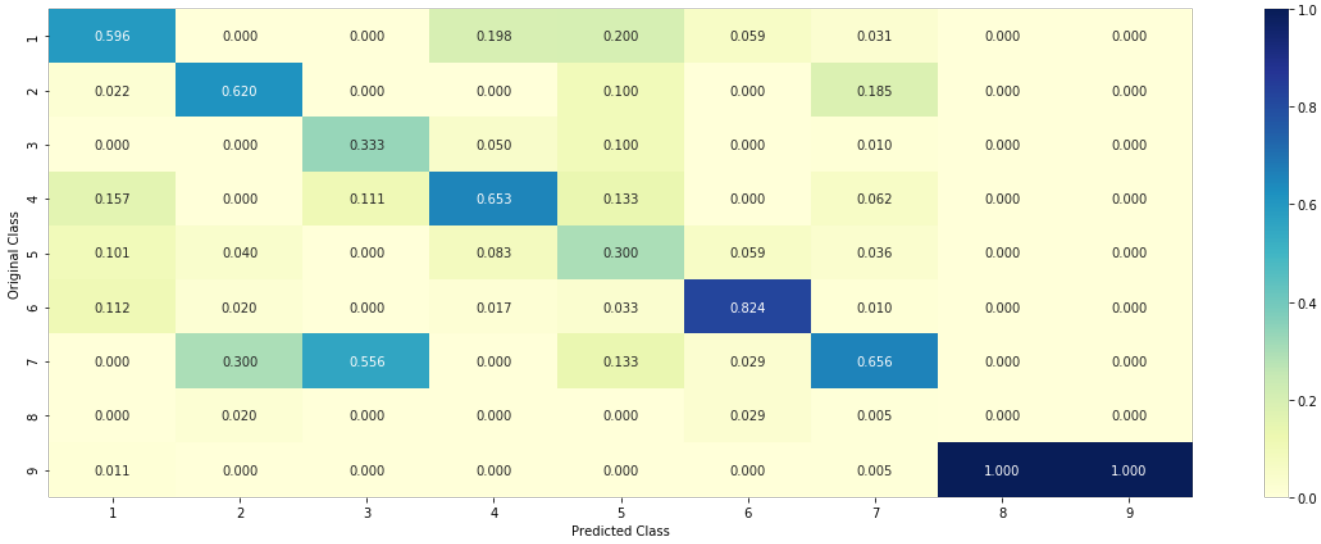
In [74]:

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

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

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

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

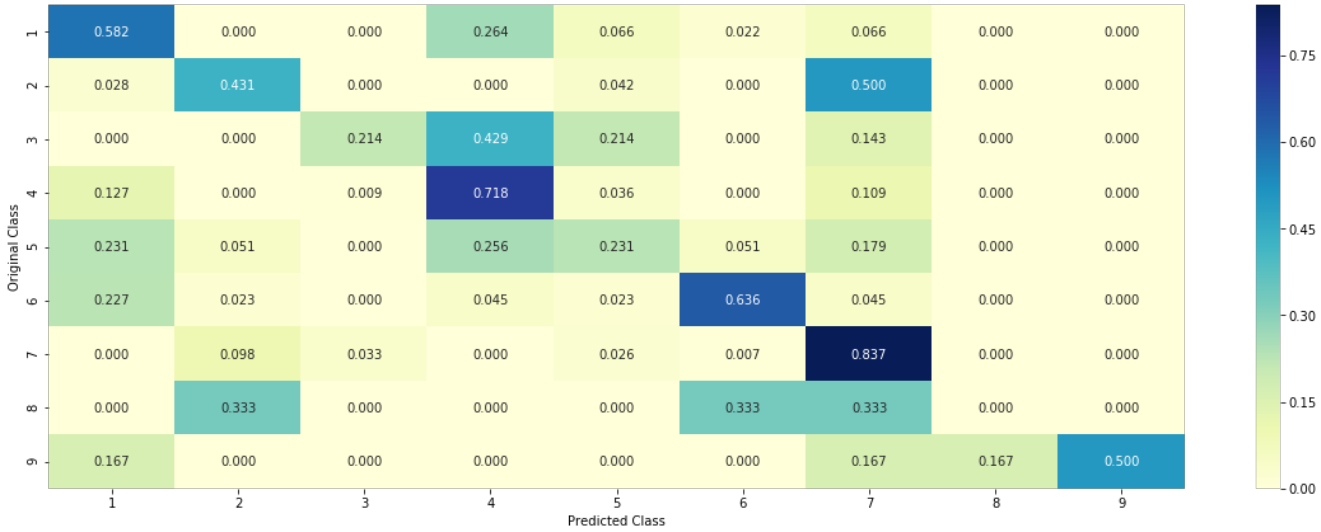
```
Log loss : 1.2260609393072237
Number of mis-classified points : 0.37218045112781956
-------------------- Confusion matrix --------------------
```



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




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



In [75]:

```
ittr+=1
df.loc[ittr]= ['LR-Imbalanced', 'ALL','onehot','BOW',"alpha {0}".format(alpha[best_alpha]),tr_loss,cv_l
oss,test_loss,misclassified_pt*100,'YES']
```

4.3.2.3. Feature Importance, Correctly Classified point

In [76]:

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

```
Predicted Class : 6
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.2285 0.0494 0.0049 0.0092 0.274  0.3915 0.0365 0.0039 0.0021]]
Actual Class : 6
--------------------------------------------------
115 Text feature [v1804d] present in test data point [True]
242 Text feature [901] present in test data point [True]
376 Text feature [oophorectomy] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [77]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[1.950e-02 5.220e-02 1.000e-04 1.990e-02 2.400e-03 1.200e-03 8.994e-01
  5.200e-03 0.000e+00]]
Actual Class : 7
--------------------------------------------------
115 Text feature [swab] present in test data point [True]
148 Text feature [subclinical] present in test data point [True]
165 Text feature [cysteine] present in test data point [True]
175 Text feature [hyperplasia] present in test data point [True]
206 Text feature [technology] present in test data point [True]
222 Text feature [activated] present in test data point [True]
224 Text feature [inhibited] present in test data point [True]
250 Text feature [extracellular] present in test data point [True]
262 Text feature [receptor] present in test data point [True]
320 Text feature [transformed] present in test data point [True]
330 Text feature [murine] present in test data point [True]
337 Text feature [ligand] present in test data point [True]
340 Text feature [cylinders] present in test data point [True]
351 Text feature [warrants] present in test data point [True]
375 Text feature [209k] present in test data point [True]
396 Text feature [downstream] present in test data point [True]
427 Text feature [inhibitor] present in test data point [True]
488 Text feature [oestrogen] present in test data point [True]
Out of the top  500  features  18 are present in query point
```

4.4. Linear Support Vector Machines

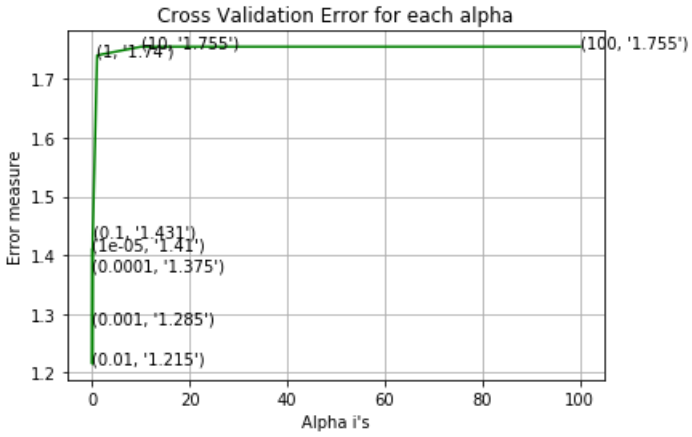
4.4.1. Hyper paramter tuning

In [78]:

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

# ----------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.
001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_
state=None)

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

```
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/mathematical-deri
vation-copy-8/
# --------------------------------



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

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

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


best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge', rando
m_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)
tr_loss = log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",tr_loss)
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
cv_loss = log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",cv_loss)
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
test_loss = log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",test_loss)

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

"""
```

```
for C = 1e-05
Log Loss : 1.4099403656709937
for C = 0.0001
Log Loss : 1.3747201348491975
for C = 0.001
Log Loss : 1.2849530080492744
for C = 0.01
Log Loss : 1.2149625494565932
for C = 0.1
Log Loss : 1.430594904714664
for C = 1
Log Loss : 1.7402346426163358
for C = 10
Log Loss : 1.7548270793775116
for C = 100
Log Loss : 1.7548270464478577
```




```
For values of best alpha =  0.01 The train log loss is: 0.7439068546698516
For values of best alpha =  0.01 The cross validation log loss is: 1.2149625494565932
For values of best alpha =  0.01 The test log loss is: 1.1997506464033854
```

Out[78]:

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

4.4.2. Testing model with best hyper parameters

In [79]:

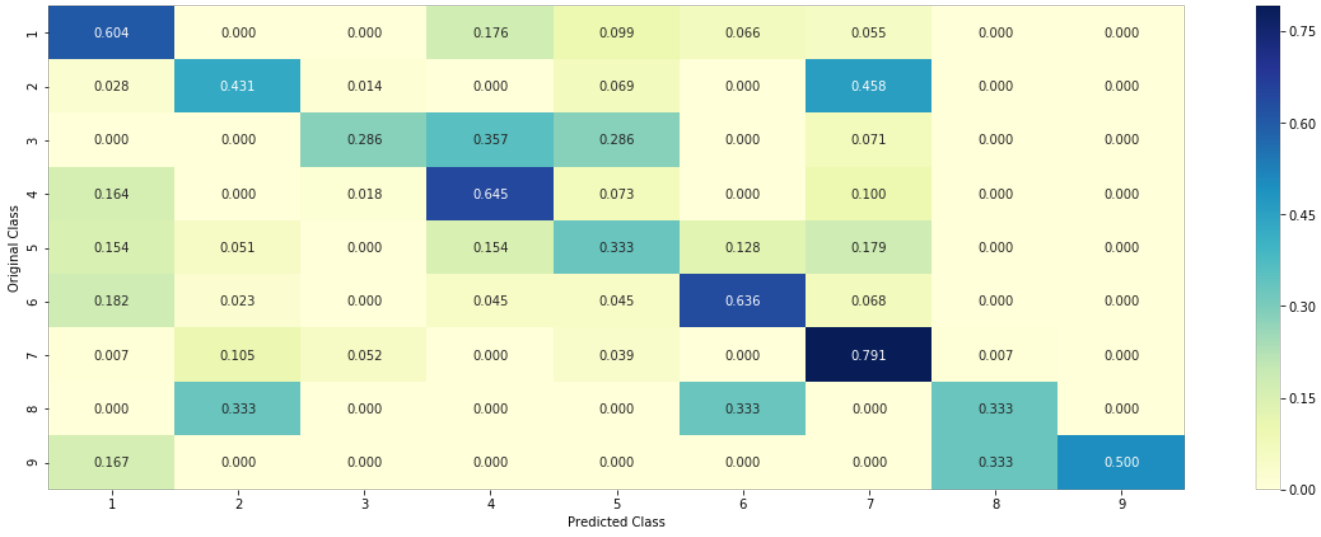
```
# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modul
es/generated/sklearn.svm.SVC.html

# ----------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.
001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_
state=None)

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


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

```
balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

```
Log loss : 1.2149625494565932
Number of mis-classified points : 0.38533834586466165
-------------------- Confusion matrix --------------------
```


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


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



In [80]:

```
ittr+=1
df.loc[ittr]= ['Linear-SVM', 'ALL','onehot','BOW',"alpha {0}".format(alpha[best_alpha]),tr_loss,cv_loss
,test_loss,misclassified_pt*100,'YES']
```

4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [81]:

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

```
Predicted Class : 5
Predicted Class Probabilities: [[0.1632 0.0822 0.0114 0.0668 0.2796 0.2656 0.1193 0.0064 0.0054]]
Actual Class : 6
--------------------------------------------------
422 Text feature [ura3] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [82]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[3.990e-02 5.140e-02 1.200e-03 3.440e-02 8.000e-03 5.600e-03 8.565e-01
  2.200e-03 8.000e-04]]
Actual Class : 7
--------------------------------------------------
44 Text feature [cysteine] present in test data point [True]
64 Text feature [cylinders] present in test data point [True]
72 Text feature [subclinical] present in test data point [True]
73 Text feature [activated] present in test data point [True]
75 Text feature [209k] present in test data point [True]
121 Text feature [oestrogen] present in test data point [True]
126 Text feature [transformed] present in test data point [True]
147 Text feature [expressing] present in test data point [True]
148 Text feature [technology] present in test data point [True]
166 Text feature [nitrocellulose] present in test data point [True]
198 Text feature [receptor] present in test data point [True]
225 Text feature [downstream] present in test data point [True]
242 Text feature [ill] present in test data point [True]
```

```
242 Text feature [11] present in test data point [True]
248 Text feature [swab] present in test data point [True]
257 Text feature [ligand] present in test data point [True]
263 Text feature [transformation] present in test data point [True]
265 Text feature [extracellular] present in test data point [True]
266 Text feature [concentrations] present in test data point [True]
298 Text feature [to3] present in test data point [True]
337 Text feature [hazards] present in test data point [True]
340 Text feature [hyperplasia] present in test data point [True]
393 Text feature [warrants] present in test data point [True]
394 Text feature [neu] present in test data point [True]
404 Text feature [inhibitor] present in test data point [True]
405 Text feature [inhibited] present in test data point [True]
436 Text feature [attractive] present in test data point [True]
438 Text feature [murine] present in test data point [True]
448 Text feature [enhancing] present in test data point [True]
Out of the top  500  features  28 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [83]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_sample
s_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impur
ity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, war
m_start=False,
# class_weight=None)

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

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

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


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

alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42, n_
jobs=-1)
        clf.fit(train_x_onehotCoding, train_y)
```

```
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_onehotCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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



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

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.296135736006595
for n_estimators = 100 and max depth =  10
Log Loss : 1.2372823970699784
for n_estimators = 200 and max depth =  5
Log Loss : 1.2737314403164237
for n_estimators = 200 and max depth =  10
Log Loss : 1.2232365882617
for n_estimators = 500 and max depth =  5
Log Loss : 1.2613048069851045
for n_estimators = 500 and max depth =  10
Log Loss : 1.2192185933922843
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2631172082862072
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2218604669767523
for n_estimators = 2000 and max depth =  5
Log Loss : 1.261818070387963
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2188123626176308
For values of best alpha =  2000 The train log loss is: 0.6948034550121251
For values of best alpha =  2000 The cross validation log loss is: 1.2188123626176308
For values of best alpha =  2000 The test log loss is: 1.1647008714914011
```

Out[83]:

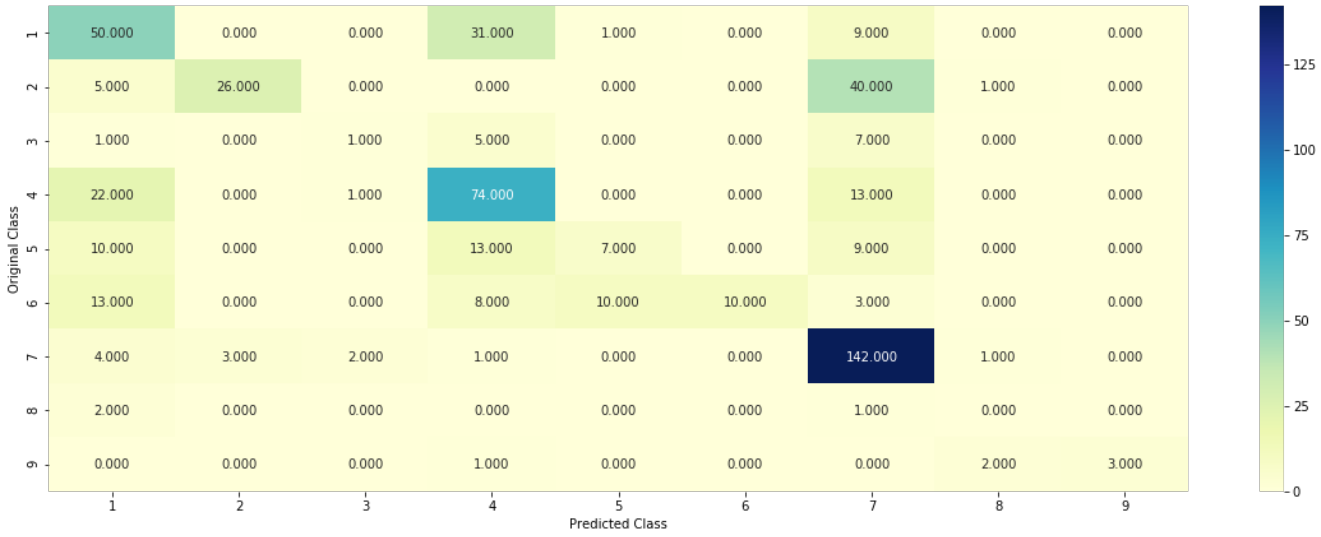
```
'\npredict_y = sig_clf.predict_proba(train_x_onehotCoding)\nprint(\'For values of best estimator = \', alpha[int(best_alpha/2)], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))\npredict_y = sig_clf.predict_proba(cv_x_onehotCoding)\nprint(\'For values of best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

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

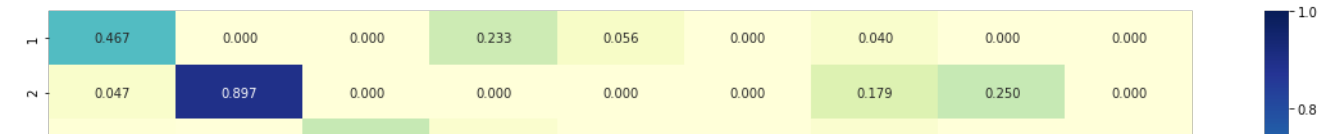
In [84]:

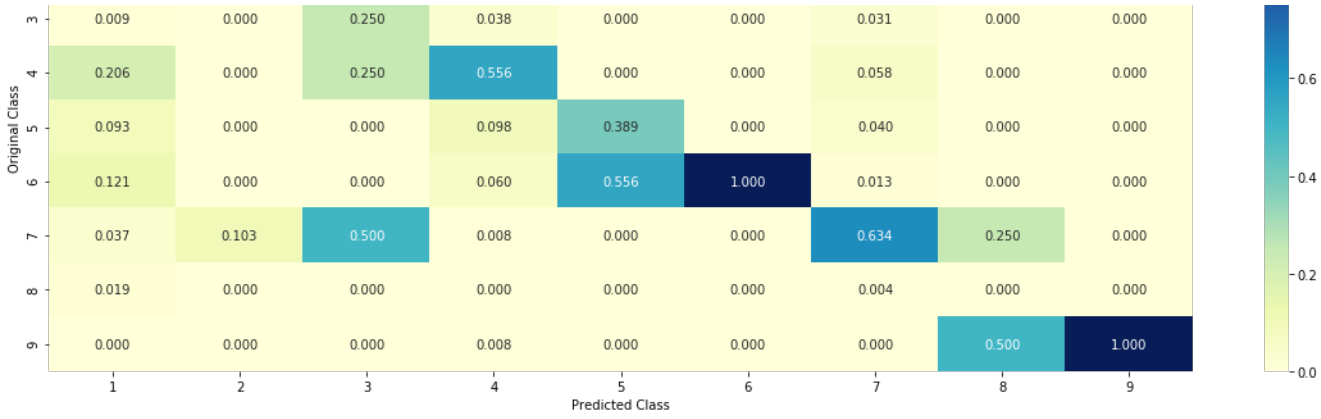
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None,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

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

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

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

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

```
Log loss : 1.2188123626176308
Number of mis-classified points : 0.4116541353383459
-------------------- Confusion matrix --------------------
```

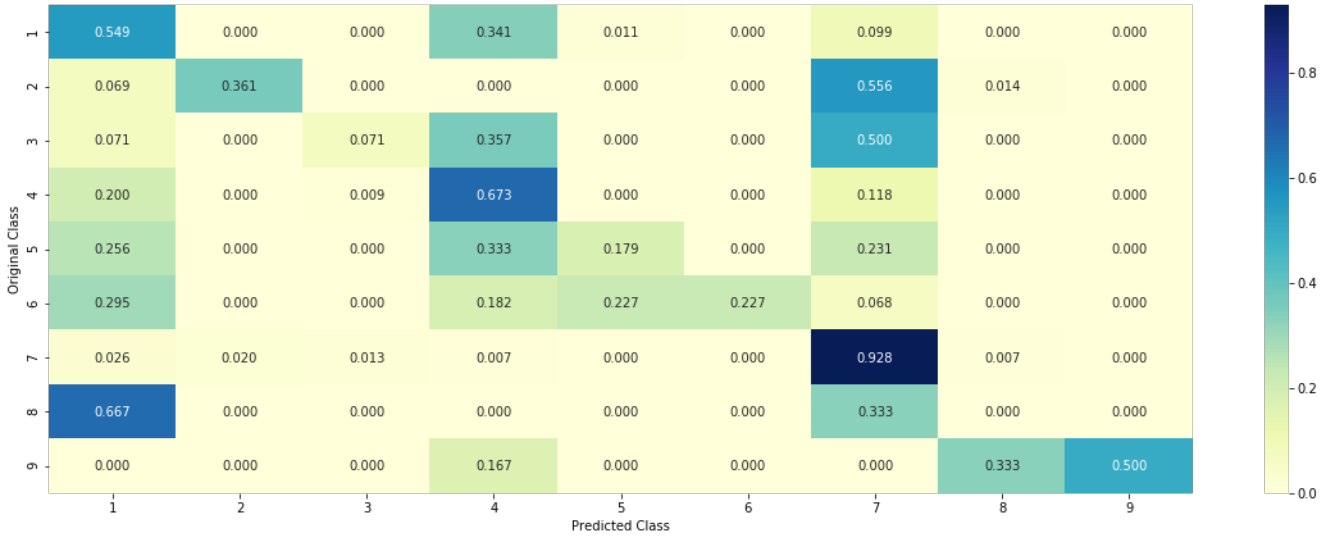



```
-------------------- Precision matrix (Column Sum=1) --------------------
```





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



In [85]:

```
ittr+=1
df.loc[ittr]= ['RF', 'ALL','onehot','BOW',"alpha {0}".format(alpha[int(best_alpha/2)]),tr_loss,cv_loss,
test_loss,misclassified_pt*100,'YES']
```

4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [86]:

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

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.366  0.0508 0.0223 0.2332 0.1537 0.1146 0.0438 0.0066 0.009 ]]
Actual Class : 6
--------------------------------------------------
5 Text feature [activation] present in test data point [True]
6 Text feature [nonsense] present in test data point [True]
9 Text feature [function] present in test data point [True]
11 Text feature [missense] present in test data point [True]
14 Text feature [treatment] present in test data point [True]
15 Text feature [variants] present in test data point [True]
19 Text feature [suppressor] present in test data point [True]
20 Text feature [therapy] present in test data point [True]
23 Text feature [growth] present in test data point [True]
24 Text feature [cells] present in test data point [True]
25 Text feature [functional] present in test data point [True]
27 Text feature [stability] present in test data point [True]
30 Text feature [deleterious] present in test data point [True]
33 Text feature [brca1] present in test data point [True]
34 Text feature [inhibition] present in test data point [True]
54 Text feature [frameshift] present in test data point [True]
55 Text feature [brct] present in test data point [True]
57 Text feature [cell] present in test data point [True]
58 Text feature [neutral] present in test data point [True]
59 Text feature [yeast] present in test data point [True]
65 Text feature [expressing] present in test data point [True]
66 Text feature [expression] present in test data point [True]
72 Text feature [protein] present in test data point [True]
80 Text feature [unclassified] present in test data point [True]
83 Text feature [dna] present in test data point [True]
84 Text feature [clinical] present in test data point [True]
85 Text feature [lines] present in test data point [True]
86 Text feature [variant] present in test data point [True]
89 Text feature [respond] present in test data point [True]
90 Text feature [proteins] present in test data point [True]
Out of the top  100  features  30 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [87]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1178 0.1229 0.0166 0.0875 0.0461 0.0387 0.5492 0.0116 0.0096]]
Actuall Class : 7
--------------------------------------------------
1 Text feature [kinase] present in test data point [True]
2 Text feature [inhibitors] present in test data point [True]
4 Text feature [activated] present in test data point [True]
8 Text feature [signaling] present in test data point [True]
9 Text feature [function] present in test data point [True]
12 Text feature [inhibitor] present in test data point [True]
13 Text feature [oncogenic] present in test data point [True]
14 Text feature [treatment] present in test data point [True]
18 Text feature [loss] present in test data point [True]
19 Text feature [suppressor] present in test data point [True]
20 Text feature [therapy] present in test data point [True]
22 Text feature [trials] present in test data point [True]
23 Text feature [growth] present in test data point [True]
24 Text feature [cells] present in test data point [True]
25 Text feature [functional] present in test data point [True]
26 Text feature [egfr] present in test data point [True]
28 Text feature [downstream] present in test data point [True]
29 Text feature [inhibited] present in test data point [True]
```

```
34 Text feature [inhibition] present in test data point [True]
38 Text feature [nsclc] present in test data point [True]
39 Text feature [treated] present in test data point [True]
40 Text feature [receptor] present in test data point [True]
43 Text feature [therapeutic] present in test data point [True]
46 Text feature [proliferation] present in test data point [True]
52 Text feature [months] present in test data point [True]
56 Text feature [extracellular] present in test data point [True]
57 Text feature [cell] present in test data point [True]
62 Text feature [ligand] present in test data point [True]
63 Text feature [patients] present in test data point [True]
65 Text feature [expressing] present in test data point [True]
66 Text feature [expression] present in test data point [True]
67 Text feature [response] present in test data point [True]
69 Text feature [activate] present in test data point [True]
70 Text feature [efficacy] present in test data point [True]
72 Text feature [protein] present in test data point [True]
73 Text feature [phosphatase] present in test data point [True]
74 Text feature [likelihood] present in test data point [True]
81 Text feature [factor] present in test data point [True]
83 Text feature [dna] present in test data point [True]
84 Text feature [clinical] present in test data point [True]
85 Text feature [lines] present in test data point [True]
88 Text feature [lung] present in test data point [True]
90 Text feature [proteins] present in test data point [True]
92 Text feature [il] present in test data point [True]
94 Text feature [survival] present in test data point [True]
97 Text feature [serum] present in test data point [True]
98 Text feature [p53] present in test data point [True]
99 Text feature [concentrations] present in test data point [True]
Out of the top  100  features  48 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [88]:

```
# ---------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_sample
s_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impur
ity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, war
m_start=False,
# class_weight=None)

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

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

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


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

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

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

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


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

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

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.034544305631865
for n_estimators = 10 and max depth =  3
Log Loss : 1.7757338459273817
for n_estimators = 10 and max depth =  5
Log Loss : 1.3221951751547376
for n_estimators = 10 and max depth =  10
Log Loss : 2.0278625867336846
for n_estimators = 50 and max depth =  2
Log Loss : 1.7141074158015 35
for n_estimators = 50 and max depth =  3
Log Loss : 1.4491449417997677
for n_estimators = 50 and max depth =  5
Log Loss : 1.4648157328294078
for n_estimators = 50 and max depth =  10
Log Loss : 1.896867957818881
for n_estimators = 100 and max depth =  2
```

```
for n_estimators = 100 and max depth =  2
Log Loss : 1.5686291977609046
for n_estimators = 100 and max depth =  3
Log Loss : 1.4713129778264629
for n_estimators = 100 and max depth =  5
Log Loss : 1.3278362111393849
for n_estimators = 100 and max depth =  10
Log Loss : 1.763422334598137
for n_estimators = 200 and max depth =  2
Log Loss : 1.6053706447126417
for n_estimators = 200 and max depth =  3
Log Loss : 1.4669066353242424
for n_estimators = 200 and max depth =  5
Log Loss : 1.3926024403499713
for n_estimators = 200 and max depth =  10
Log Loss : 1.7778548574795279
for n_estimators = 500 and max depth =  2
Log Loss : 1.669180845170103
for n_estimators = 500 and max depth =  3
Log Loss : 1.5292715167212325
for n_estimators = 500 and max depth =  5
Log Loss : 1.4125566570096657
for n_estimators = 500 and max depth =  10
Log Loss : 1.808665314628538
for n_estimators = 1000 and max depth =  2
Log Loss : 1.636920610310959
for n_estimators = 1000 and max depth =  3
Log Loss : 1.5340573421771253
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3910053094026542
for n_estimators = 1000 and max depth =  10
Log Loss : 1.8257432551837651
For values of best alpha =  10 The train log loss is: 0.06998483267081859
For values of best alpha =  10 The cross validation log loss is: 1.3221951751547376
For values of best alpha =  10 The test log loss is: 1.313385140746074
```

Out[88]:

```
'\npredict_y = sig_clf.predict_proba(train_x_responseCoding)\nprint(\'For values of best alpha = \', al
pha[int(best_alpha/4)], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=
1e-15))\npredict_y = sig_clf.predict_proba(cv_x_responseCoding)\nprint(\'For values of best alpha = \',
alpha[int(best_alpha/4)], "The cross validation log loss is:",log_loss(y_cv, predict_y, labels=clf.clas
ses_, eps=1e-15))\npredict_y = sig_clf.predict_proba(test_x_responseCoding)\nprint(\'For values of best
alpha = \', alpha[int(best_alpha/4)], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.cl
asses_, eps=1e-15))\n'
```

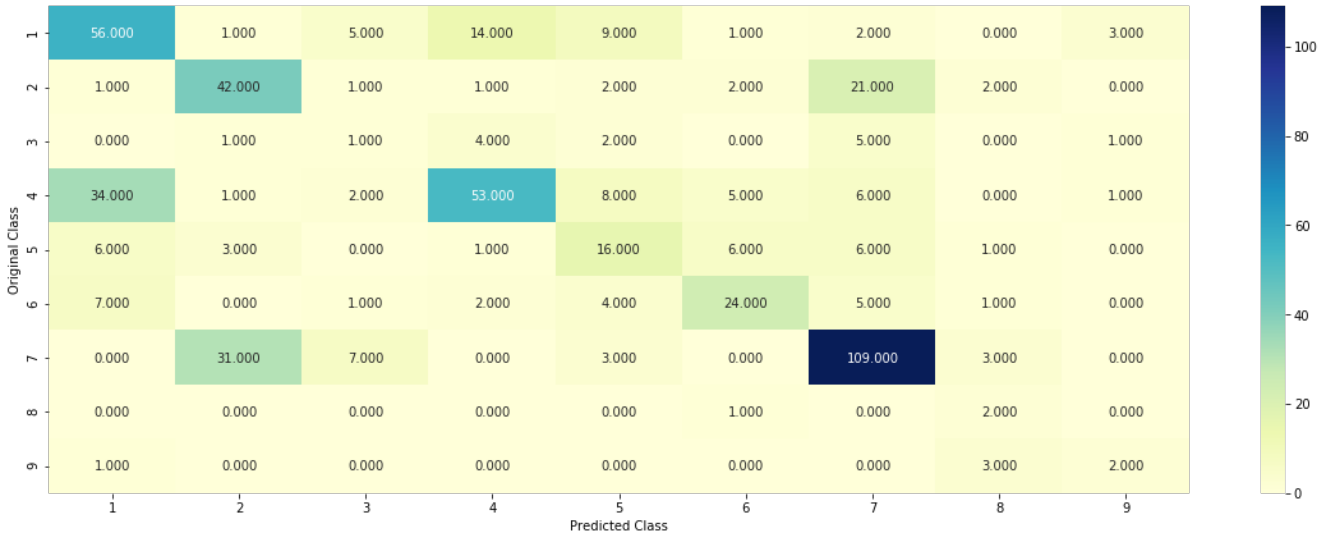
4.5.4. Testing model with best hyper parameters (Response Coding)

In [89]:

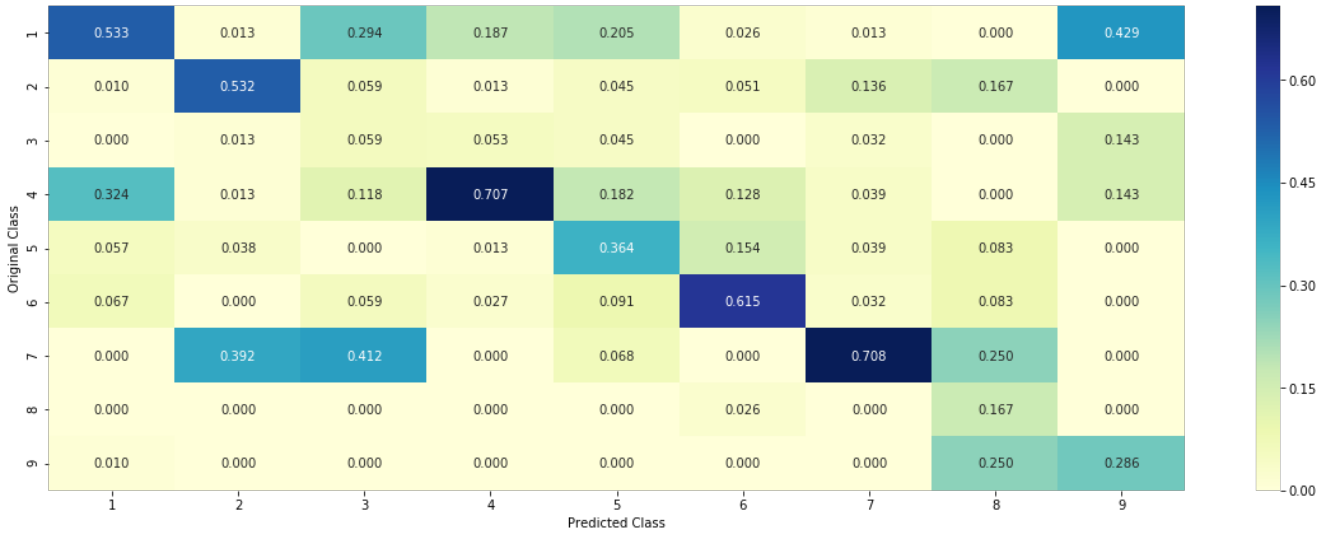
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_sample
s_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impur
ity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, war
m_start=False,
# class_weight=None)

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

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

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

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

```
4)], criterion='gini', max_features='auto',random_state=42)
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCoding,cv_y, clf)
```

```
Log loss : 1.3221951751547376
Number of mis-classified points : 0.4266917293233083
-------------------- Confusion matrix --------------------
```




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




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

In [90]:

```
ittr+=1
df.loc[ittr]= ['RF', 'ALL','response','BOW',"alpha {0}".format(alpha[int(best_alpha/4)]),tr_loss,cv_los
s,test_loss,misclassified_pt*100,'YES']
```

4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [91]:

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


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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0765 0.0056 0.2277 0.1224 0.2549 0.2943 0.0024 0.0103 0.0058]]
Actual Class : 6
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [92]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0025 0.1038 0.0052 0.0065 0.0015 0.0062 0.8291 0.0106 0.0345]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [93]:

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

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

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


# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
# ---------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

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


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

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

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

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


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

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', random_state=0)
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


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

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehotCoding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding))))
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
```

```
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba
s=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sclf.p
redict_proba(cv_x_onehotCoding))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

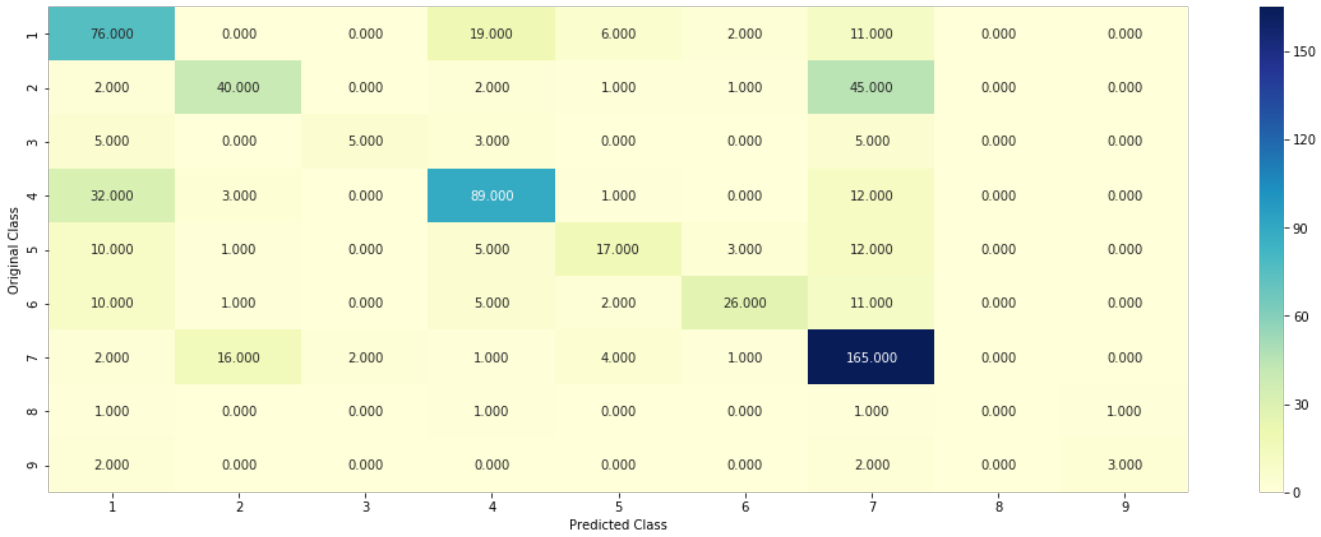
```
Logistic Regression :  Log Loss: 1.17
Support vector machines : Log Loss: 1.74
Naive Bayes : Log Loss: 1.34
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.179
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.045
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.555
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.194
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.314
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.570
```

4.7.2 testing the model with the best hyper parameters

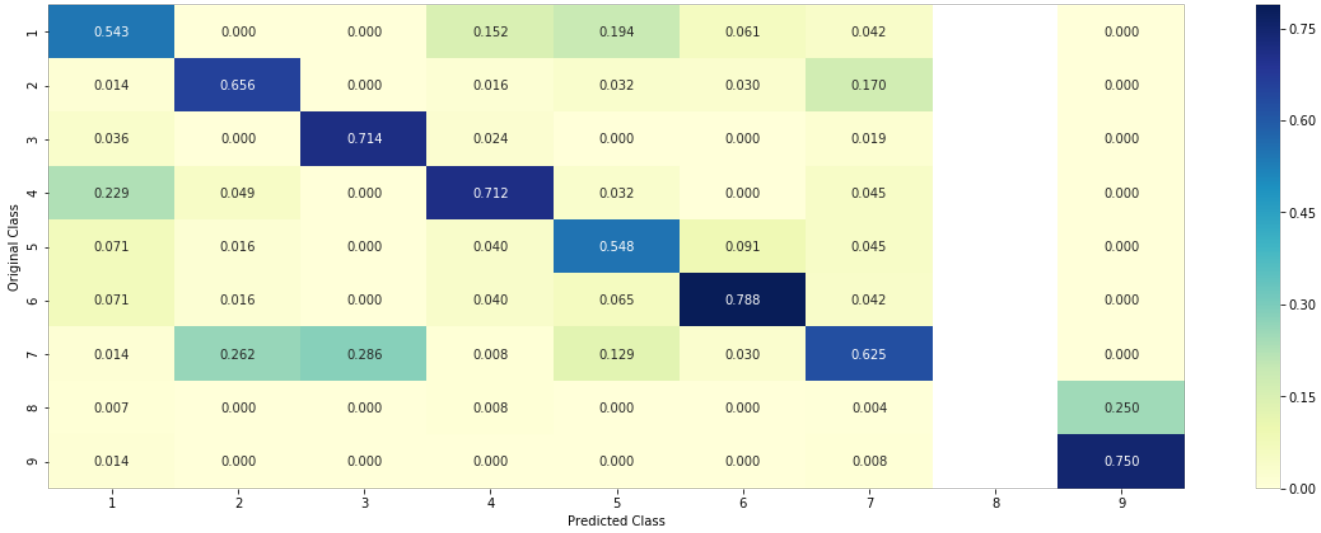
In [94]:

```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=Tr
ue)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

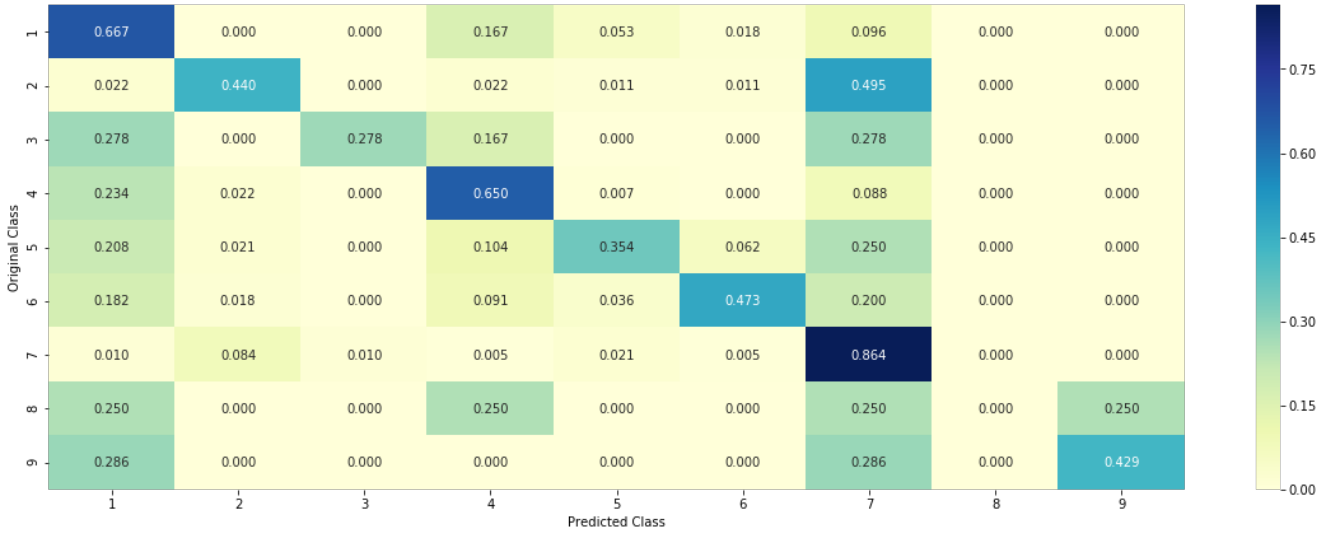
```
Log loss (train) on the stacking classifier : 0.665202059583961
Log loss (CV) on the stacking classifier : 1.1944176426301407
Log loss (test) on the stacking classifier : 1.1610434421769167
Number of missclassified point : 0.3669172932330827
-------------------- Confusion matrix --------------------
```




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



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



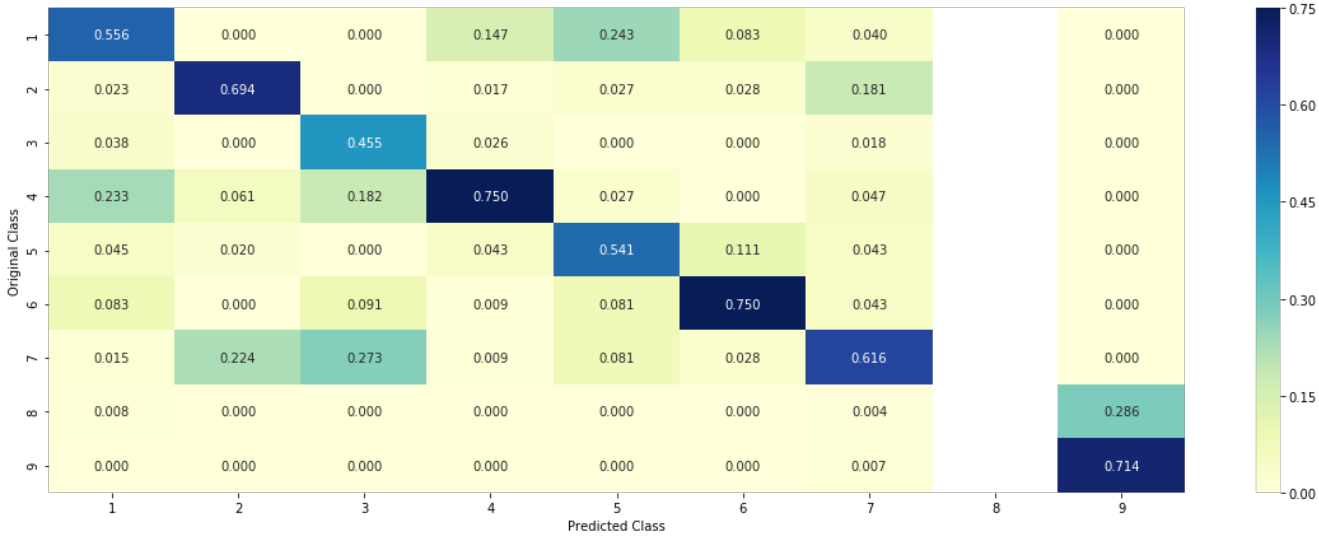
4.7.3 Maximum Voting classifier

In [95]:

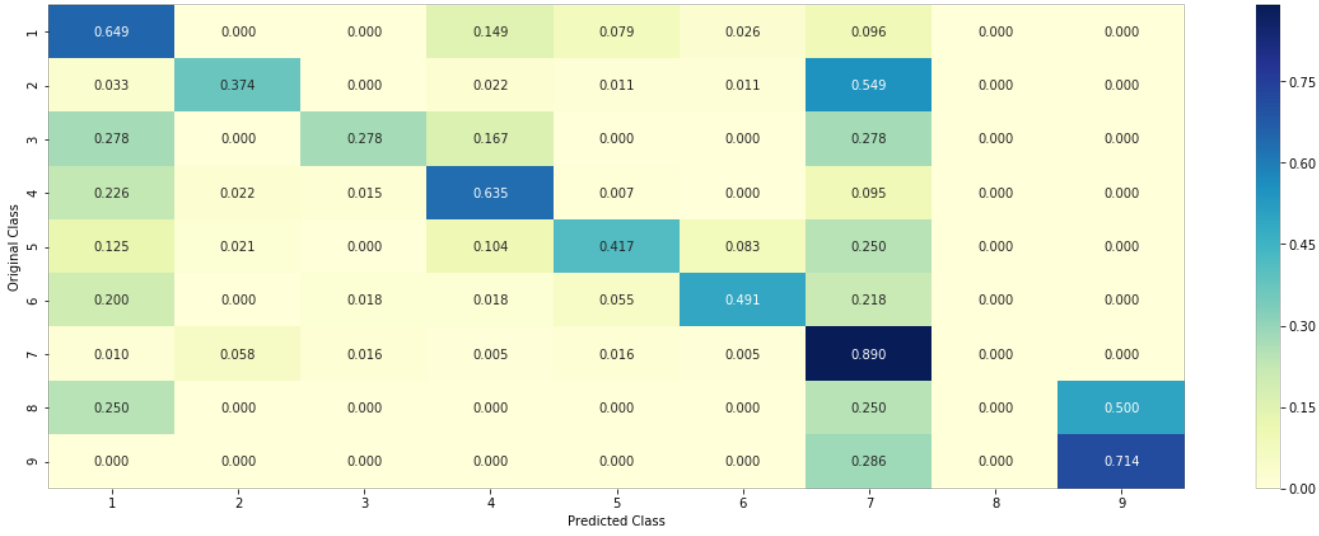
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
tr_loss =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

```
Log loss (train) on the VotingClassifier : 0.9079621886852434
Log loss (CV) on the VotingClassifier : 1.2580971096727878
Log loss (test) on the VotingClassifier : 1.2472013744397252
Number of missclassified point : 0.36541353383458647
---------- Confusion matrix ----------
```

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



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



In [96]:

```
ittr+=1
df.loc[ittr]= ['stacking', 'ALL','onehot','BOW',"NA",tr_loss,cv_loss,test_loss,misclassified_pt*100,'NO
']
```

5. Assignments

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

5.1 Below models are for TFIDF vectrozier

stacking features with text featured with TFIDF

In [97]:

```
text_vectorizer = TfidfVectorizer()
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
#normalize
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)
#process test data
test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEXT'])
test_text_feature_onehotCoding = normalize(test_text_feature_onehotCoding, axis=0)
#process CV data
cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])
cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding, axis=0)


#stacking for TFIDF
train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_onehotCoding)).tocsr()
test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_onehotCoding)).tocsr()
cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).tocsr()
```

Naive Bayes with TFIDF

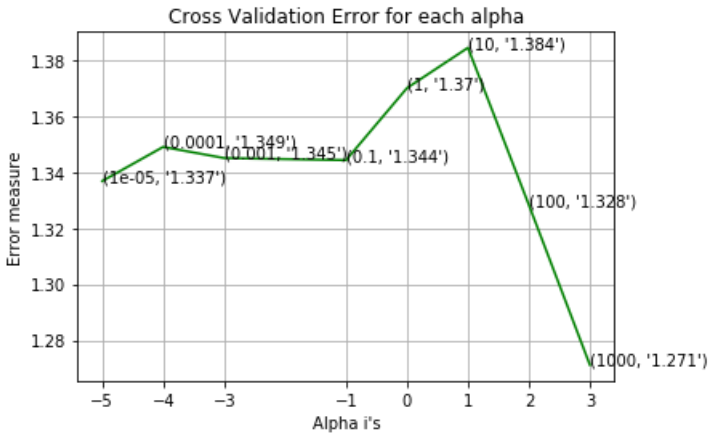
Apply All the models with tf-idf features (Replace CountVectorizer with tfidfVectorizer and run the same cells)

In [98]:

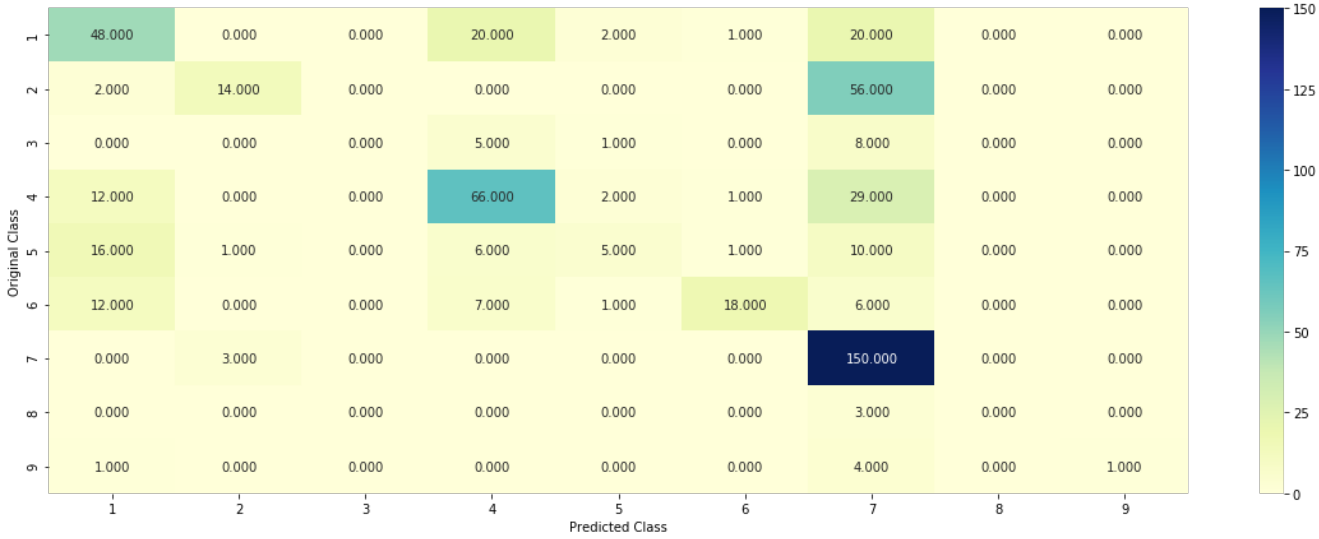
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated
/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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

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

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


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


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



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
misclassified_pt = np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)- cv_y))/cv_y.shape[0]
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", misclassified_pt)
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))

ittr+=1
df.loc[ittr]= ['NB', 'ALL','onehot','TFIDF',"alpha {0}".format(alpha[best_alpha]),tr_loss,cv_loss,test_
loss,misclassified_pt*100,'YES']
```

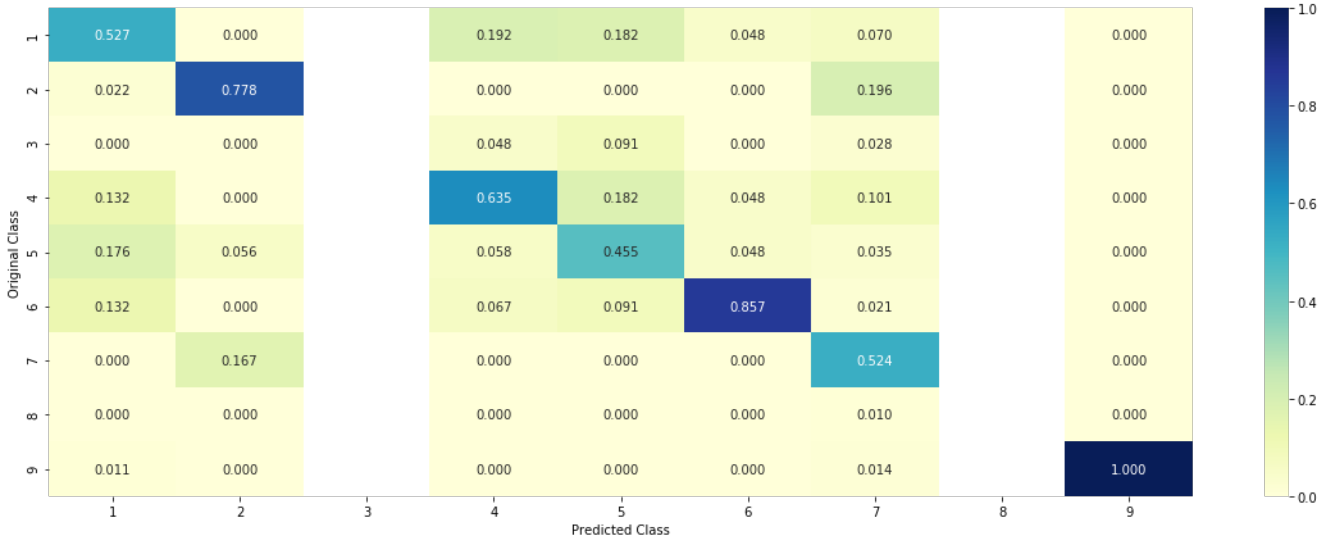
```
for alpha = 1e-05
Log Loss : 1.3368241212816123
for alpha = 0.0001
Log Loss : 1.3490270195322214
for alpha = 0.001
Log Loss : 1.3450398224401292
for alpha = 0.1
Log Loss : 1.34429870617996
for alpha = 1
Log Loss : 1.3700490611522531
for alpha = 10
Log Loss : 1.384453721750188
for alpha = 100
Log Loss : 1.3282649418423922
for alpha = 1000
Log Loss : 1.2711701634756623
```



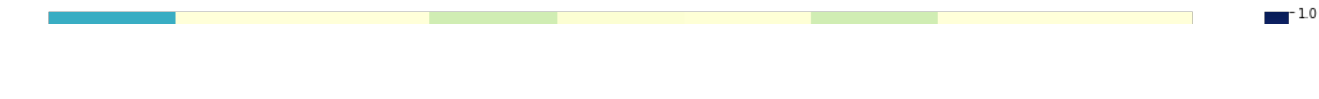

```
For values of best alpha =  1000 The train log loss is: 0.8572589896625962
For values of best alpha =  1000 The cross validation log loss is: 1.2711701634756623
For values of best alpha =  1000 The test log loss is: 1.2340965960430728
Log Loss : 1.2711701634756623
Number of missclassified point : 0.4323308270676692
-------------------- Confusion matrix --------------------
```

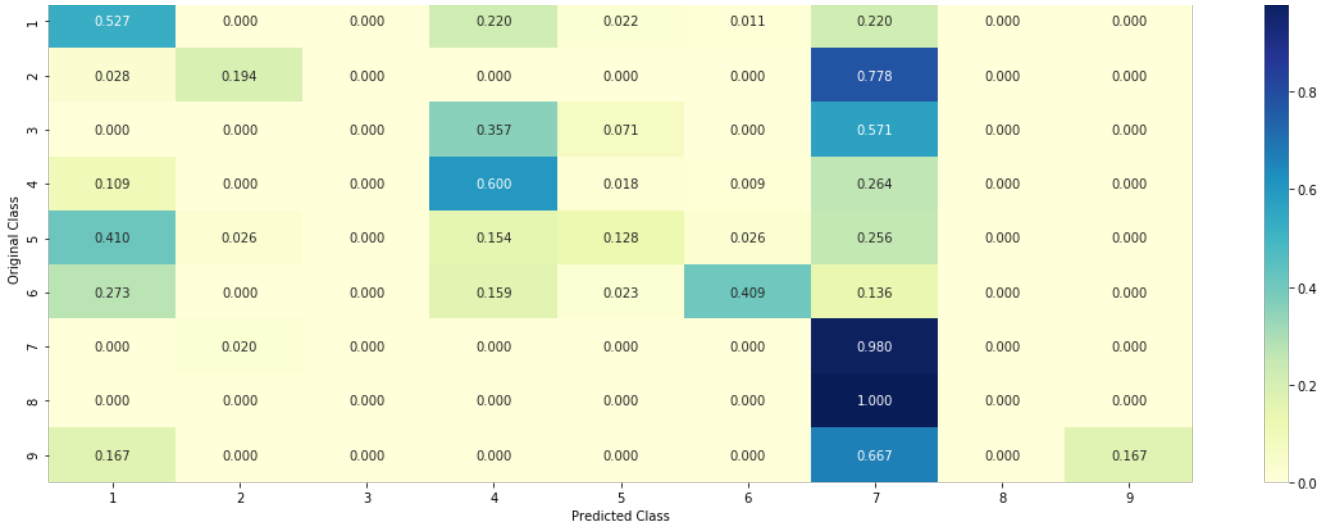



```
-------------------- Precision matrix (Column Sum=1) --------------------
```




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





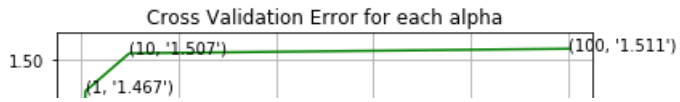
Logistic(Balanced) regression with TFIDF

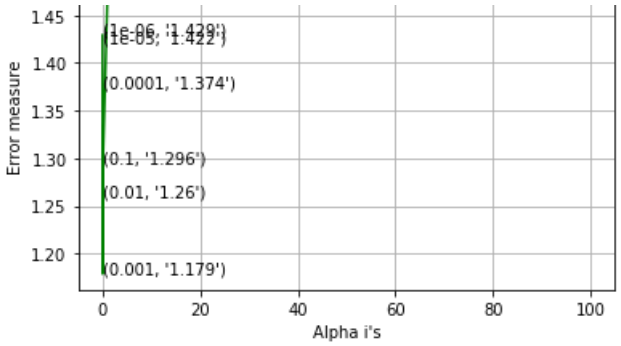
In [99]:

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

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

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


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

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

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
```

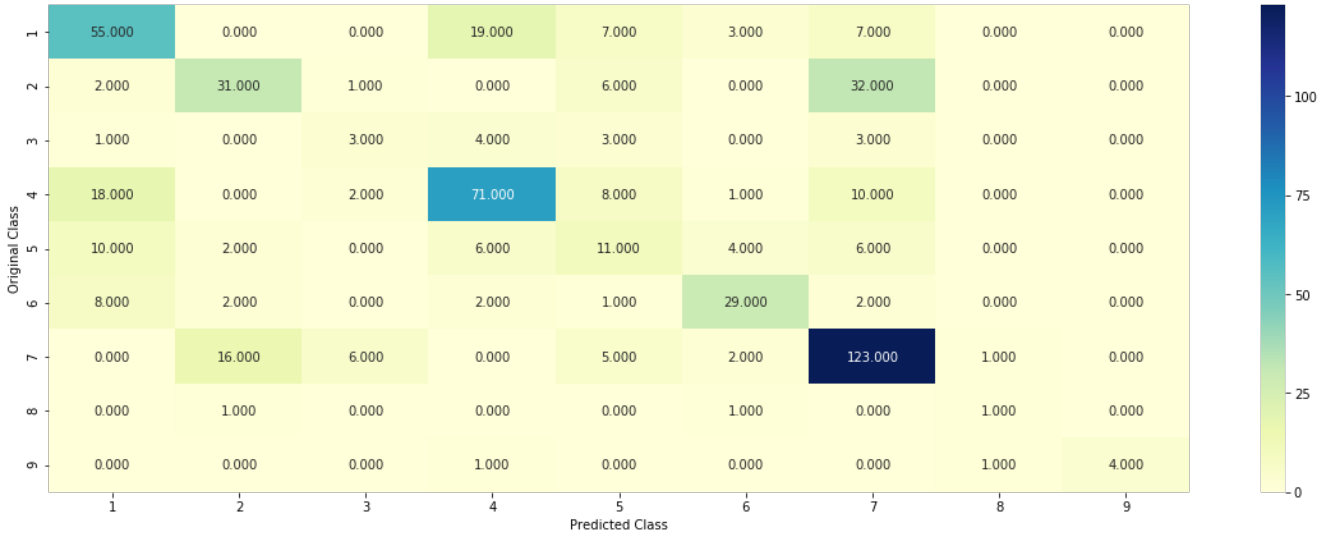
```
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


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

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


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

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

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

ittr+=1
df.loc[ittr]= ['LR-Balanced', 'ALL','onehot','TFIDF',"alpha {0}".format(alpha[best_alpha]),tr_loss,cv_l
oss,test_loss,misclassified_pt*100,'YES']
```

```
for alpha = 1e-06
Log Loss : 1.4291954556793074
for alpha = 1e-05
Log Loss : 1.4216951903833837
for alpha = 0.0001
Log Loss : 1.3736684737499547
for alpha = 0.001
Log Loss : 1.1789250368242568
for alpha = 0.01
Log Loss : 1.259511199924155
for alpha = 0.1
Log Loss : 1.295672000679568
for alpha = 1
Log Loss : 1.4667791795280494
for alpha = 10
Log Loss : 1.5065355271179004
for alpha = 100
Log Loss : 1.5113511334406793
```

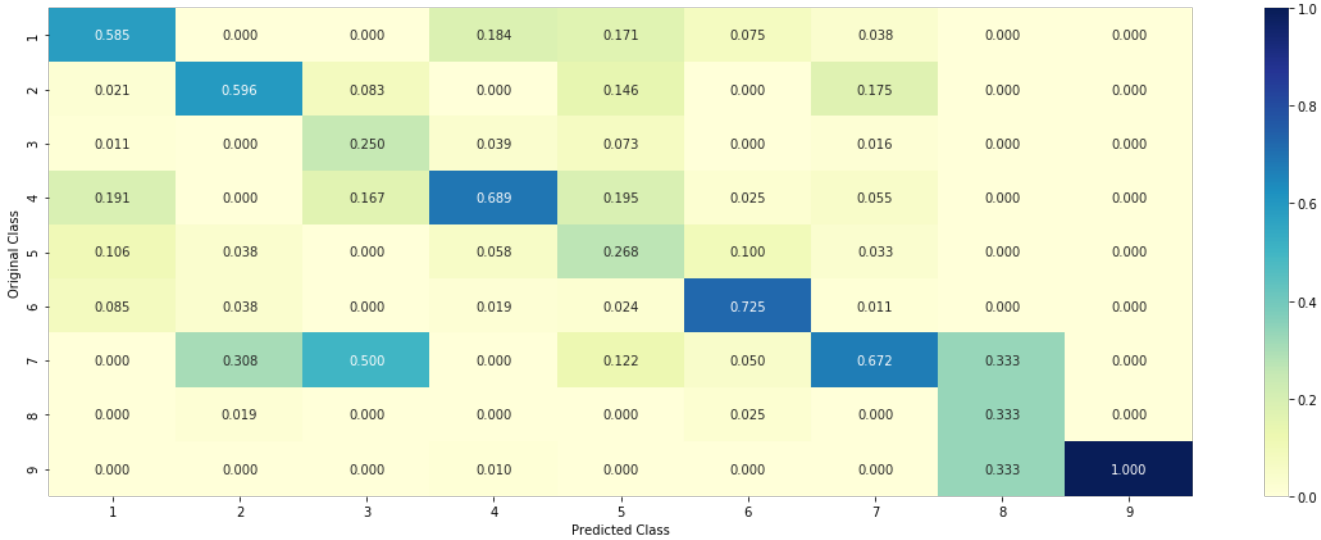




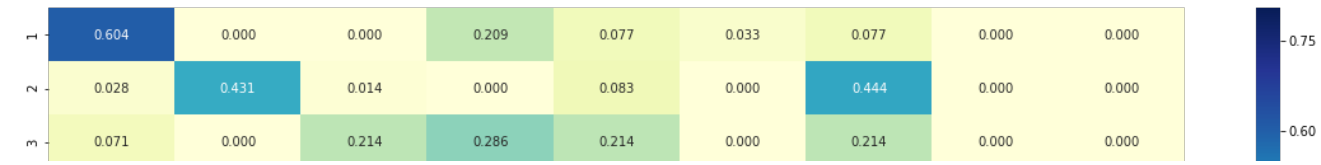

```
For values of best alpha =  0.001 The train log loss is: 0.5706610604132623
For values of best alpha =  0.001 The cross validation log loss is: 1.1789250368242568
For values of best alpha =  0.001 The test log loss is: 1.140753798900384
Log loss : 1.1789250368242568
Number of mis-classified points : 0.38345864661654133
-------------------- Confusion matrix --------------------
```

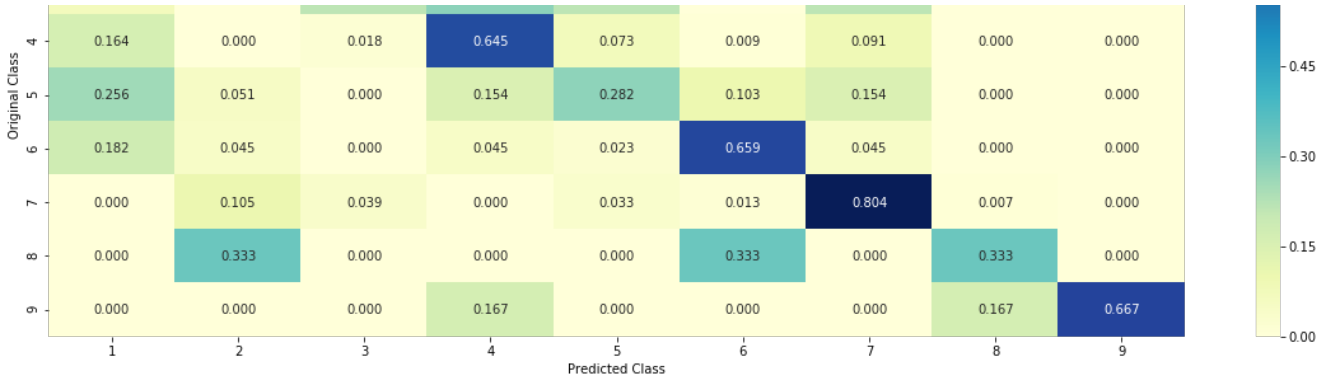



```
-------------------- Precision matrix (Column Sum=1) --------------------
```




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





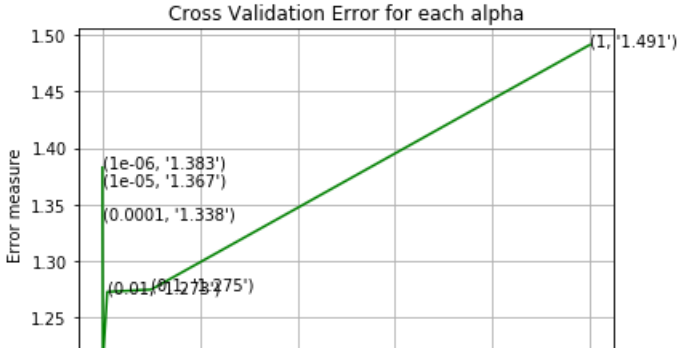
Logistic regression (Imbalanced data) with TFIDF

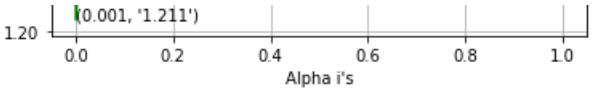
In [100]:

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

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

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



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

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

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

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



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

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

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

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

ittr+=1
df.loc[ittr]= ['LR-Imbalanced', 'ALL','onehot','TFIDF',"alpha {0}".format(alpha[best_alpha]),tr_loss,cv_loss,test_loss,misclassified_pt*100,'YES']
```

```
for alpha = 1e-06
Log Loss : 1.3827958324343965
for alpha = 1e-05
Log Loss : 1.3665163418335673
for alpha = 0.0001
Log Loss : 1.3380742986078897
for alpha = 0.001
Log Loss : 1.210615430453461
for alpha = 0.01
Log Loss : 1.2725936477284756
for alpha = 0.1
Log Loss : 1.2746668260984144
for alpha = 1
Log Loss : 1.491390117692734
```

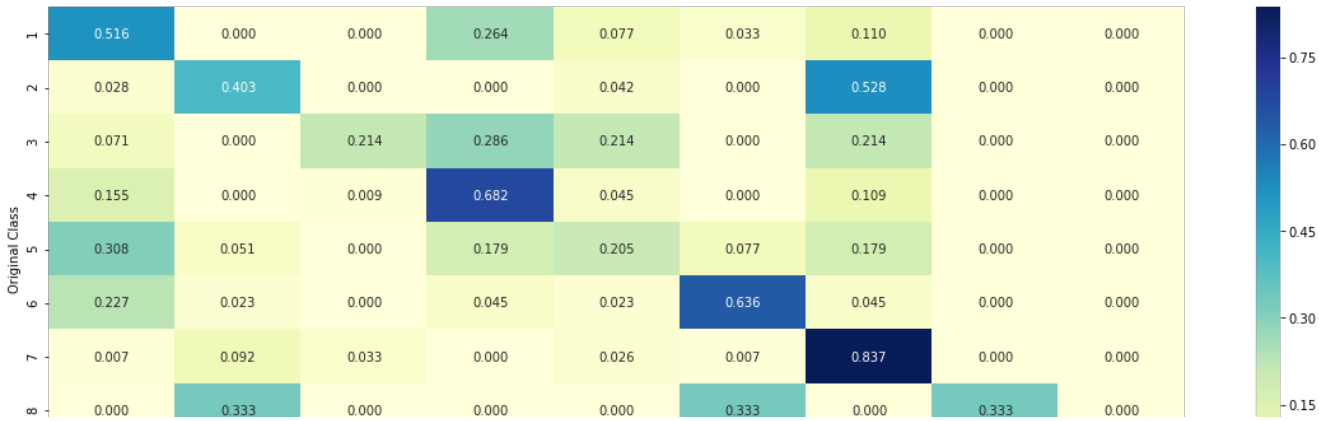


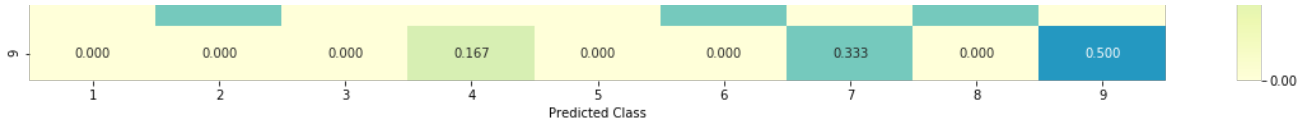



```
For values of best alpha =  0.001 The train log loss is: 0.5678977708363465
For values of best alpha =  0.001 The cross validation log loss is: 1.210615430453461
For values of best alpha =  0.001 The test log loss is: 1.1702497058695709
Log loss : 1.210615430453461
Number of mis-classified points : 0.39473684210526316
-------------------- Confusion matrix --------------------
```


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


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





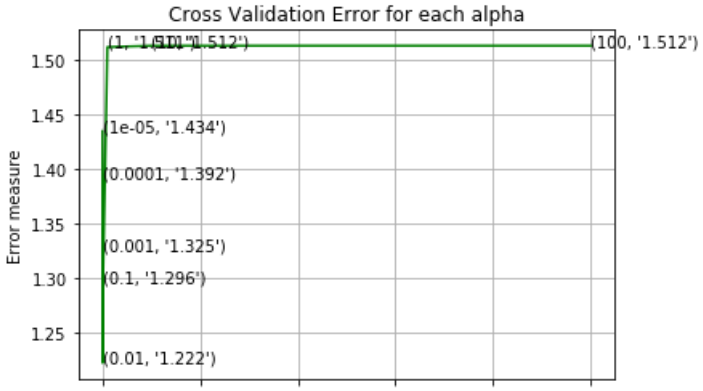
Linear Support vector with TFIDF

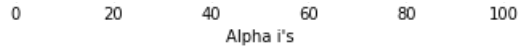
In [101]:

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

# ------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

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



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

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

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


best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge', 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)
tr_loss = log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",tr_loss)
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
cv_loss = log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",cv_loss)
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
test_loss = log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",test_loss)


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

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

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


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



ittr+=1
df.loc[ittr]= ['Linear-SVM', 'ALL','onehot','TFIDF',"alpha {0}".format(alpha[best_alpha]),tr_loss,cv_loss,test_loss,misclassified_pt*100,'YES']
```

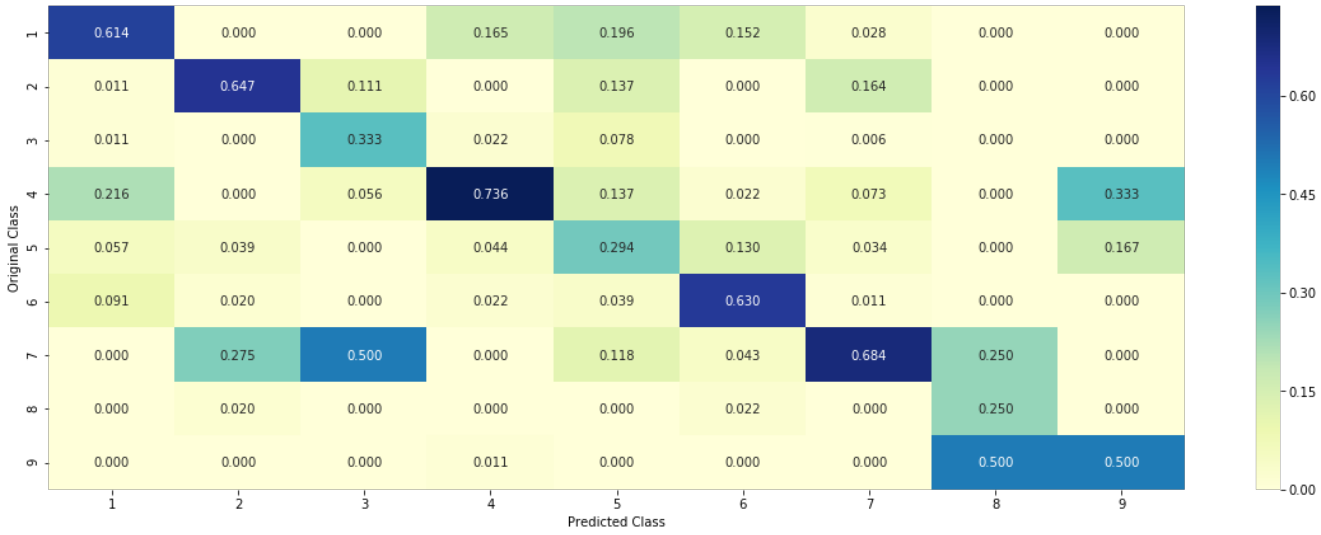
```
for C = 1e-05
Log Loss : 1.4343453208156982
for C = 0.0001
Log Loss : 1.3918388525802343
for C = 0.001
Log Loss : 1.3251617589233753
for C = 0.01
Log Loss : 1.2223450020519724
for C = 0.1
Log Loss : 1.295560479584867
for C = 1
Log Loss : 1.5111624068525444
for C = 10
Log Loss : 1.5123897199211311
for C = 100
Log Loss : 1.5123897721964261
```



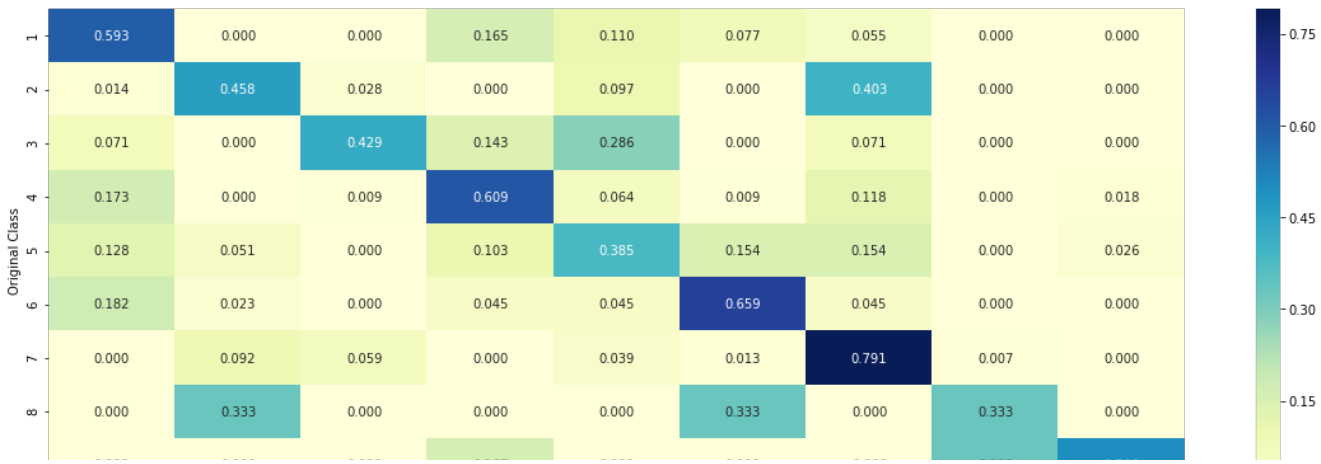



```
For values of best alpha =  0.01 The train log loss is: 0.6678987418910446
For values of best alpha =  0.01 The cross validation log loss is: 1.2223450020519724
For values of best alpha =  0.01 The test log loss is: 1.1895598933222642
Log loss : 1.2223450020519724
Number of mis-classified points : 0.3815789473684211
-------------------- Confusion matrix --------------------
```


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




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



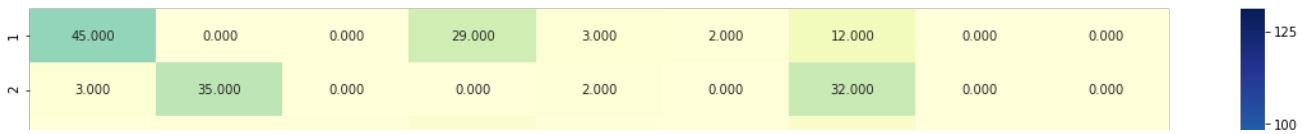
randomForest with TFIDF

In [102]:

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

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

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

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


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

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


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



predict_y = sig_clf.predict_proba(train_x_onehotCoding)
```

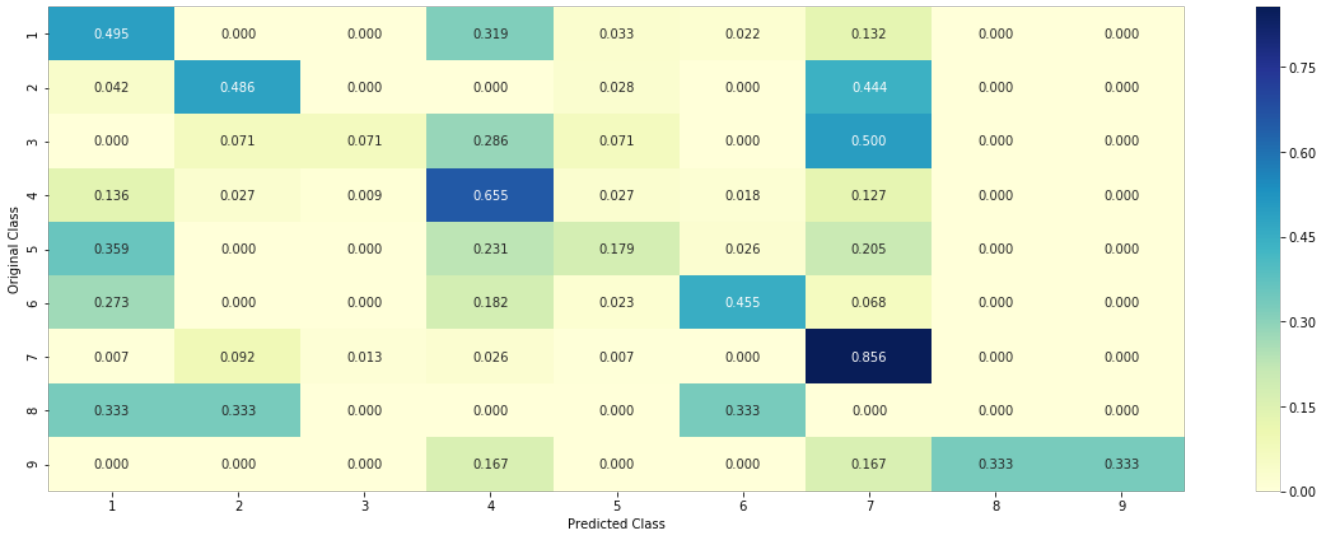
```
tr_loss = log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[int(best_alpha/2)], "The train log loss is:",tr_loss)
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
cv_loss = log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15)
print('For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


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

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

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

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

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


ittr+=1
df.loc[ittr]= ['RF', 'ALL','onehot','TFIDF',"alpha {0}".format(alpha[int(best_alpha/2)]),tr_loss,cv_loss,test_loss,misclassified_pt*100,'YES']
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.32012423708061
for n_estimators = 100 and max depth =  10
Log Loss : 1.2477493587369601
for n_estimators = 200 and max depth =  5
Log Loss : 1.3034931762125983
for n_estimators = 200 and max depth =  10
Log Loss : 1.2342769403614486
for n_estimators = 500 and max depth =  5
Log Loss : 1.2969613508165598
for n_estimators = 500 and max depth =  10
Log Loss : 1.230132203905445
for n_estimators = 1000 and max depth =  5
Log Loss : 1.28795438963387
for n_estimators = 1000 and max depth =  10
Log Loss : 1.226762040470033
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2821171482343343
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2254535661966535
For values of best alpha =  2000 The train log loss is: 0.6748423216859906
For values of best alpha =  2000 The cross validation log loss is: 1.225453566196628
For values of best alpha =  2000 The test log loss is: 1.1468068302231404
Log loss : 1.2254535661965993
Number of mis-classified points : 0.4116541353383459
-------------------- Confusion matrix --------------------
```



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

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



stacking model with TFIDF

In [103]:

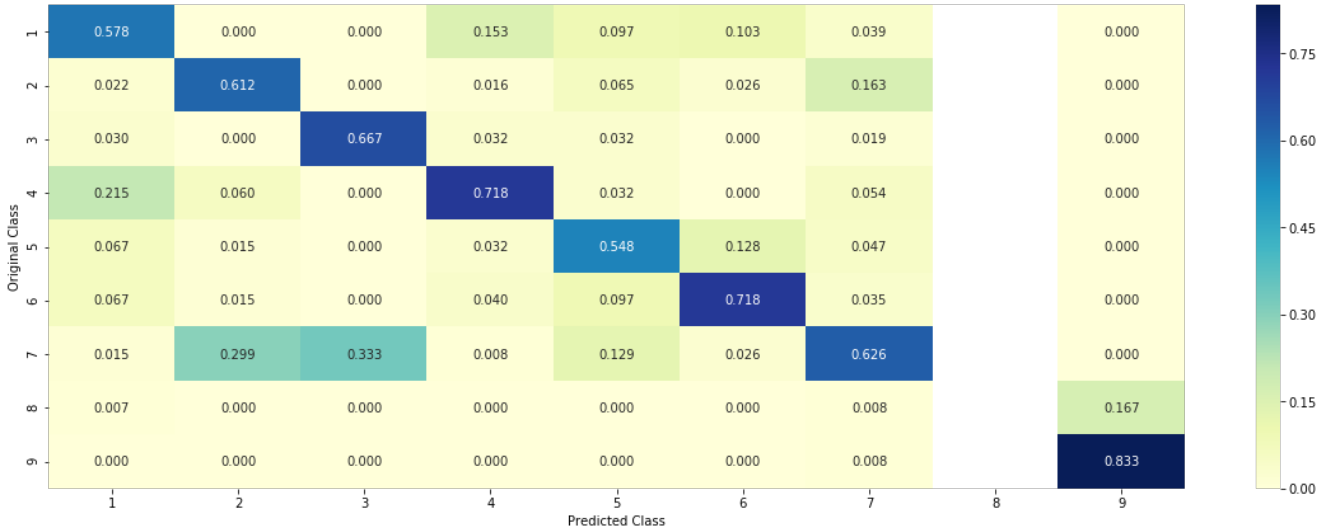
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
```

```
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
# ----------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

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


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

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

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

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


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

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', random_state=0)
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


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

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

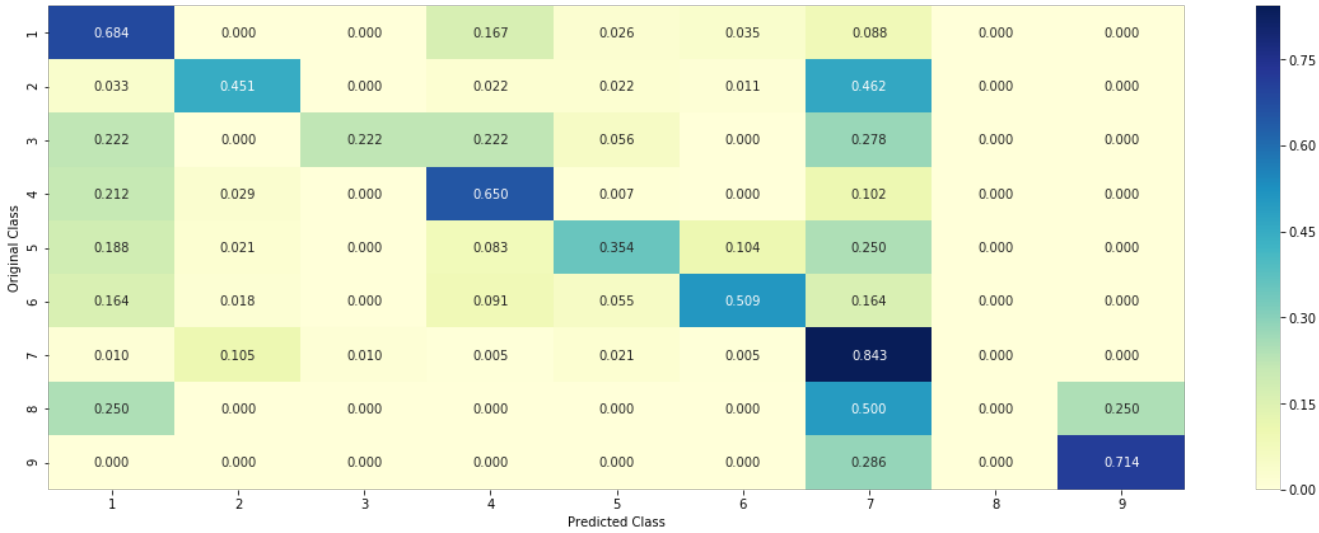
```
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(cv_x_onehot
oding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding))))
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]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



lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=Tr
ue)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='sof
t')
vclf.fit(train_x_onehotCoding, train_y)
tr_loss = log_loss(train_y, vclf.predict_proba(train_x_onehotCoding))
cv_loss = log_loss(cv_y, vclf.predict_proba(cv_x_onehotCoding))
test_loss = log_loss(test_y, vclf.predict_proba(test_x_onehotCoding))
misclassified_pt = np.count_nonzero((vclf.predict(test_x_onehotCoding)- test_y))/test_y.shape[0]
print("Log loss (train) on the VotingClassifier :", tr_loss)
print("Log loss (CV) on the VotingClassifier :", cv_loss)
print("Log loss (test) on the VotingClassifier :", test_loss)
print("Number of missclassified point :", misclassified_pt)
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))

ittr+=1
df.loc[ittr]= ['stacking', 'ALL','onehot','TFIDF','NA',tr_loss,cv_loss,test_loss,misclassified_pt*100,'
NO']
```

```
Logistic Regression :  Log Loss: 1.18
Support vector machines : Log Loss: 1.51
Naive Bayes : Log Loss: 1.35
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.179
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.042
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.544
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.200
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.327
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.561
Log loss (train) on the stacking classifier : 0.6356319068025486
Log loss (CV) on the stacking classifier : 1.2003304796291647
Log loss (test) on the stacking classifier : 1.163376241846243
Number of missclassified point : 0.36390977443609024
-------------------- Confusion matrix --------------------
```

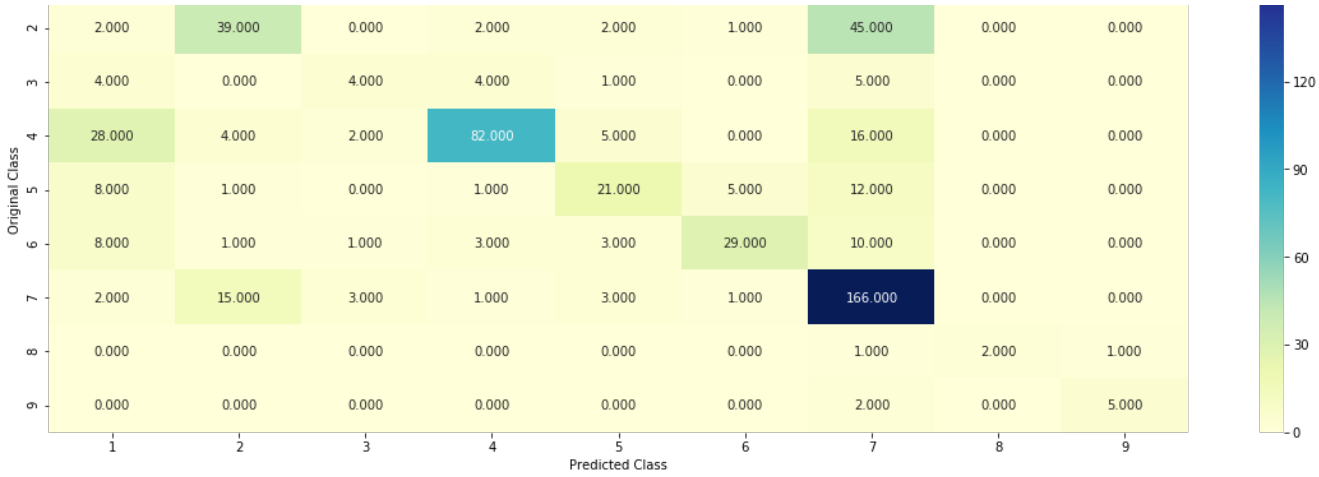
---------- Precision matrix (Column Sum=1) ----------



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

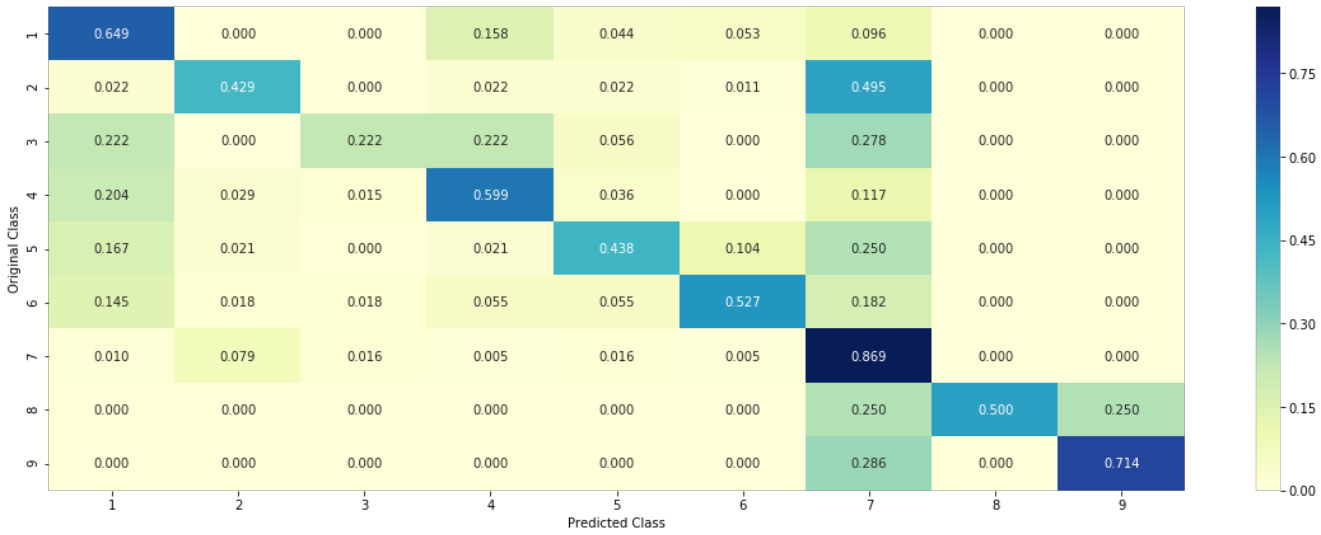


Log loss (train) on the VotingClassifier : 0.8477646097582059
Log loss (CV) on the VotingClassifier : 1.2162560204637465
Log loss (test) on the VotingClassifier : 1.2026876565888904
Number of missclassified point : 0.36541353383458647
---------- Confusion matrix ----------



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

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



5.2 Models for top 1000 words

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

stacking features with text featured with TFIDF

In [104]:

```
text_vectorizer = TfidfVectorizer(max_features=1000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
#normalize
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)
#process test data
test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEXT'])
test_text_feature_onehotCoding = normalize(test_text_feature_onehotCoding, axis=0)
#process CV data
cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])
cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding, axis=0)

#stacking for TFIDF with best 1000 words
train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_onehotCoding)).tocsr()
test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_onehotCoding)).tocsr()
cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).tocsr()
```

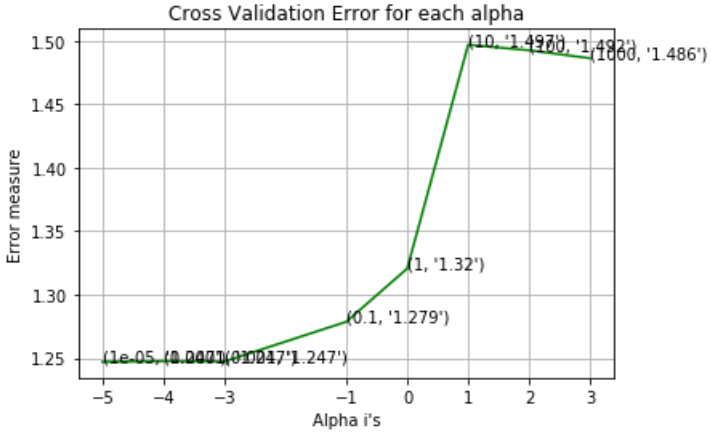
5.2.1 Naive Bayes with top 1k word

In [105]:

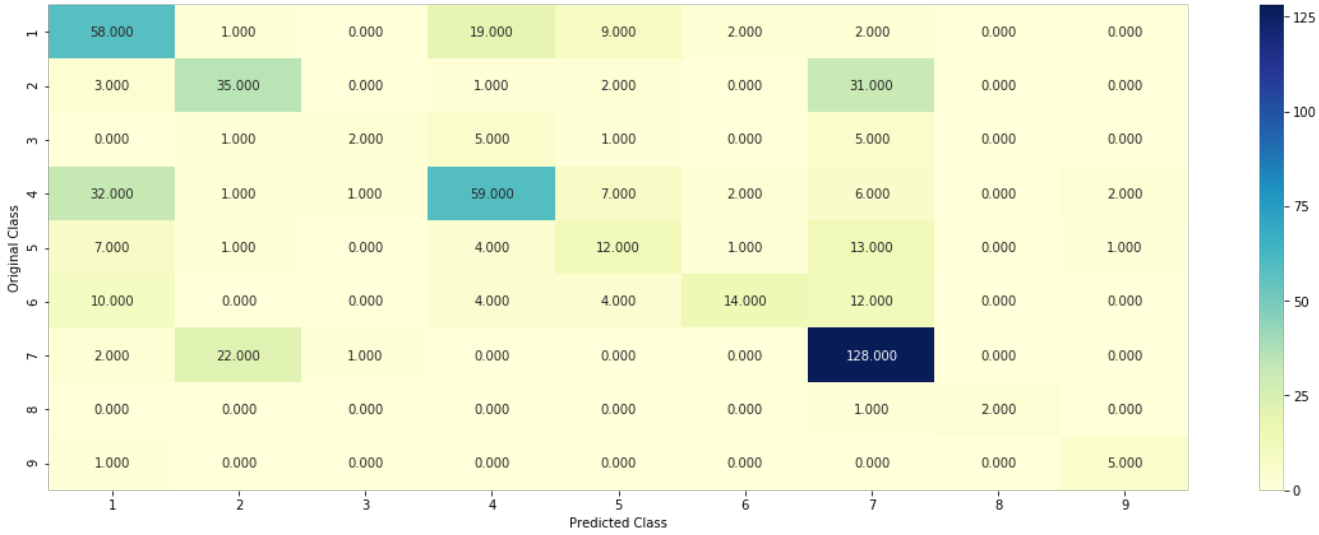
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated
/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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


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

fig, ax = plt.subplots()
ax.plot(np.log10(alpha), cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv_log_error_array[i]))
plt.grid()
plt.xticks(np.log10(alpha))
```

```
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


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


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



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
misclassified_pt = np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)- cv_y))/cv_y.shape[0]
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", misclassified_pt)
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))

ittr+=1
df.loc[ittr]= ['NB', 'ALL','onehot','TFIDF_top1000',"alpha {0}".format(alpha[best_alpha]),tr_loss,cv_lo
ss,test_loss,misclassified_pt*100,'YES']
```

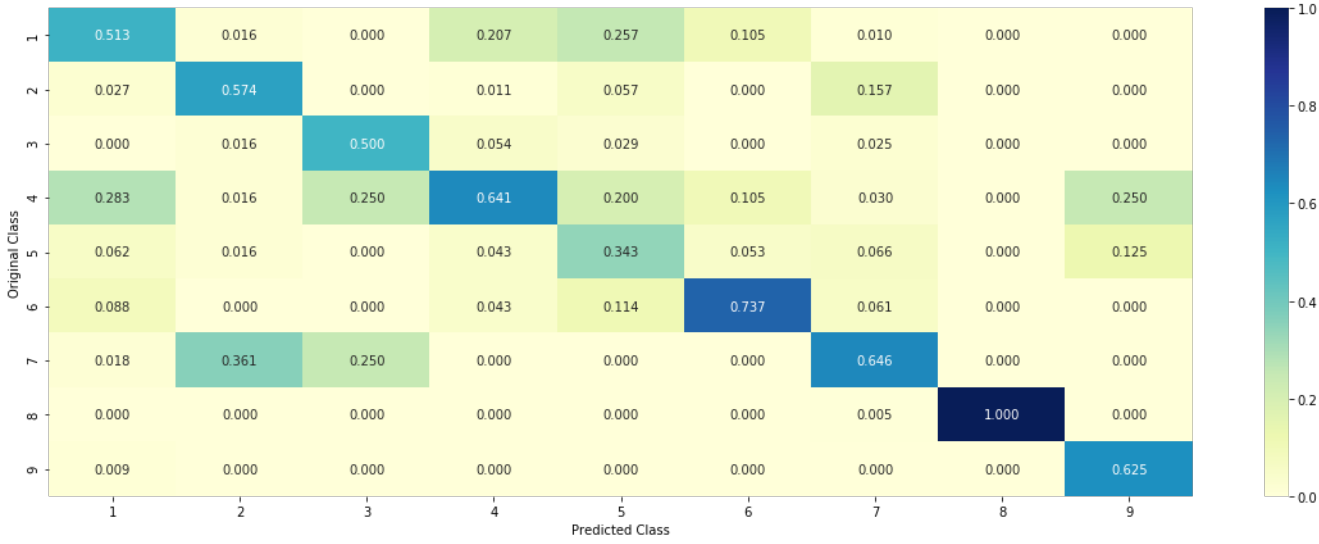
```
for alpha = 1e-05
Log Loss : 1.2469869062589594
for alpha = 0.0001
Log Loss : 1.2474918980057637
for alpha = 0.001
Log Loss : 1.2472434592451678
for alpha = 0.1
Log Loss : 1.2785292989567618
for alpha = 1
Log Loss : 1.3204812267405293
for alpha = 10
Log Loss : 1.4968104904834518
for alpha = 100
Log Loss : 1.492397331452136
for alpha = 1000
Log Loss : 1.4861664158219396
```



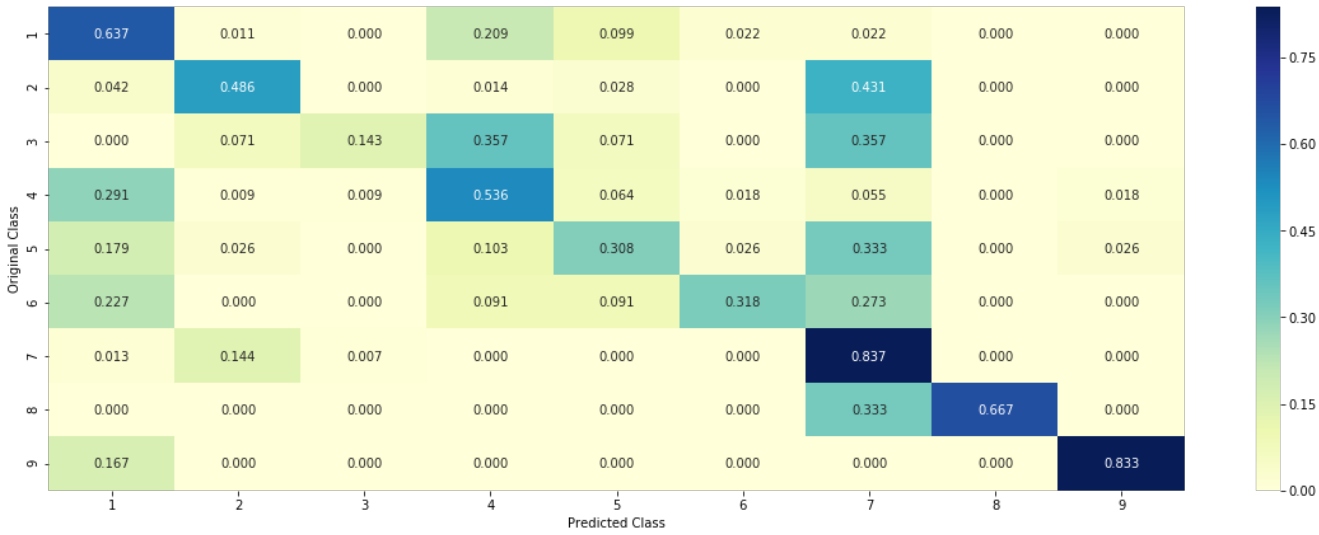
```
For values of best alpha =  1e-05 The train log loss is: 0.49662899061153
For values of best alpha =  1e-05 The cross validation log loss is: 1.2469869062589594
For values of best alpha =  1e-05 The test log loss is: 1.2102543414180853
Log Loss : 1.2469869062589594
Number of missclassified point : 0.40789473684210525
-------------------- Confusion matrix --------------------
```




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




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



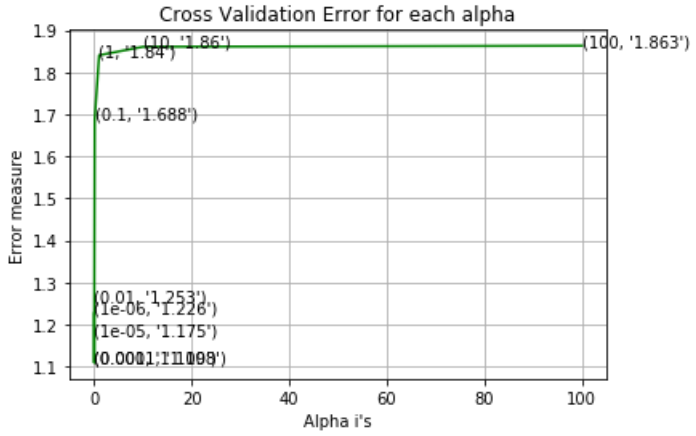
5.2.2 LR(Balanced) with top 1k word

In [106]:

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

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

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


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

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

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


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

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

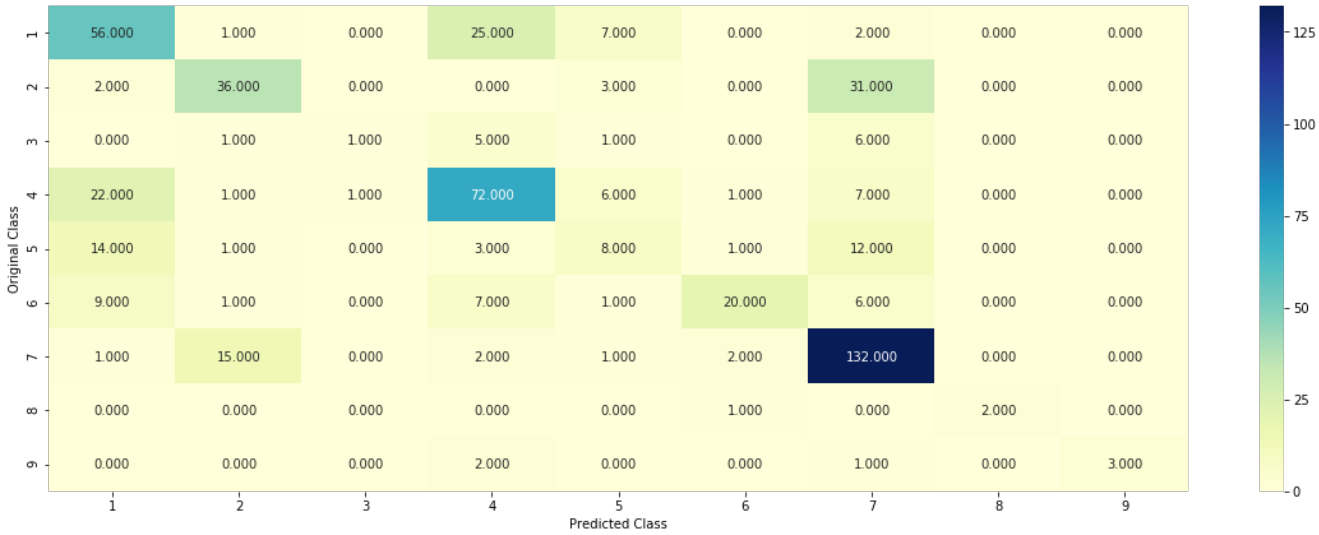
```
cv_loss = log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",cv_loss)
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
test_loss = log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15)
print('For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


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

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

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

ittr+=1
df.loc[ittr]= ['LR-Balanced', 'ALL','onehot','TFIDF_top1000',"alpha {0}".format(alpha[best_alpha]),tr_loss,cv_loss,test_loss,misclassified_pt*100,'YES']
```

```
for alpha = 1e-06
Log Loss : 1.2259621638686045
for alpha = 1e-05
Log Loss : 1.1746349808337646
for alpha = 0.0001
Log Loss : 1.1084303925859684
for alpha = 0.001
Log Loss : 1.1094362814665306
for alpha = 0.01
Log Loss : 1.2526087586184769
for alpha = 0.1
Log Loss : 1.6883498590008819
for alpha = 1
Log Loss : 1.840347632335724
for alpha = 10
Log Loss : 1.860459327232688
for alpha = 100
Log Loss : 1.8629530568272863
```

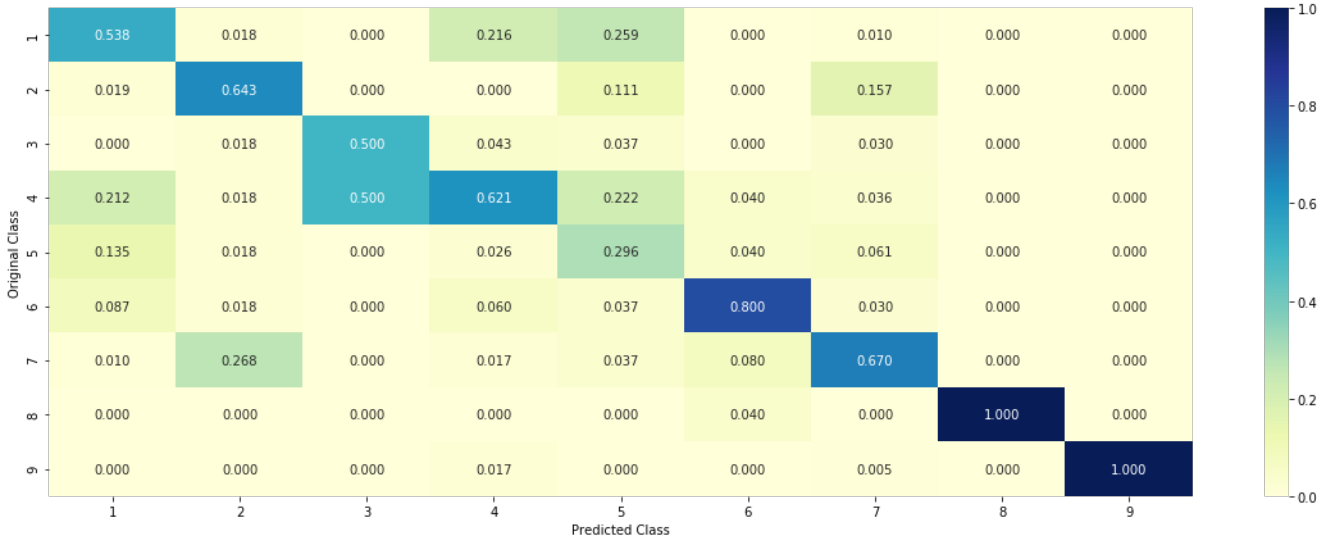



```
For values of best alpha =  0.0001 The train log loss is: 0.4211955119427828
For values of best alpha =  0.0001 The cross validation log loss is: 1.1084303925859684
For values of best alpha =  0.0001 The test log loss is: 1.0337990541461897
Log loss : 1.1084303925859684
Number of mis-classified points : 0.37969924812030076
```

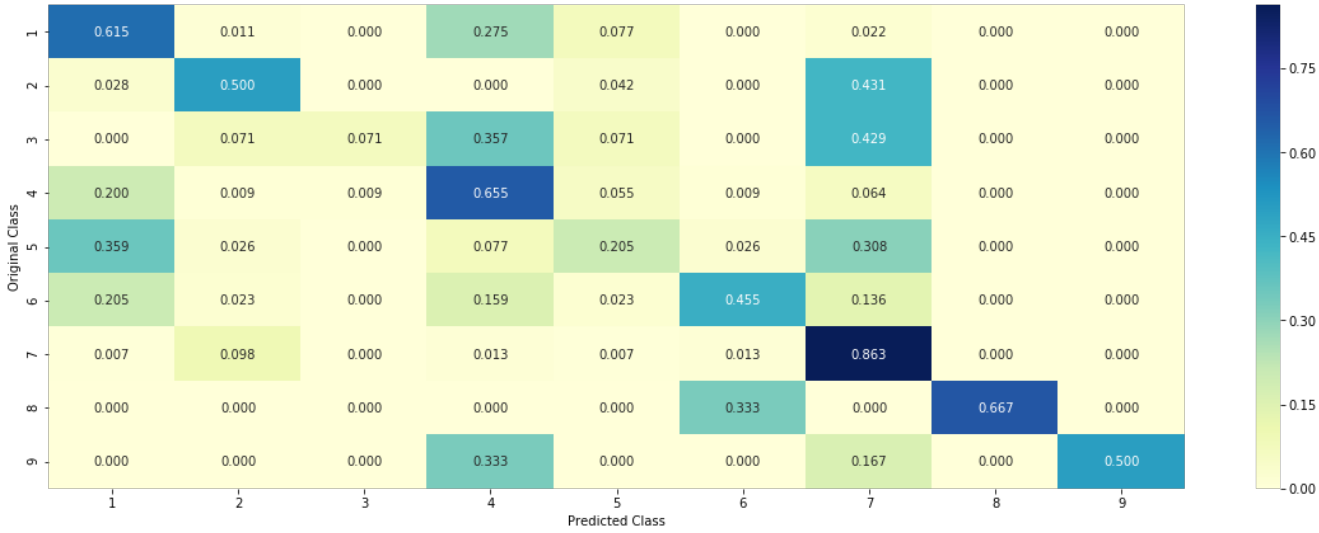
```
-------------------- Confusion matrix --------------------
```




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




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



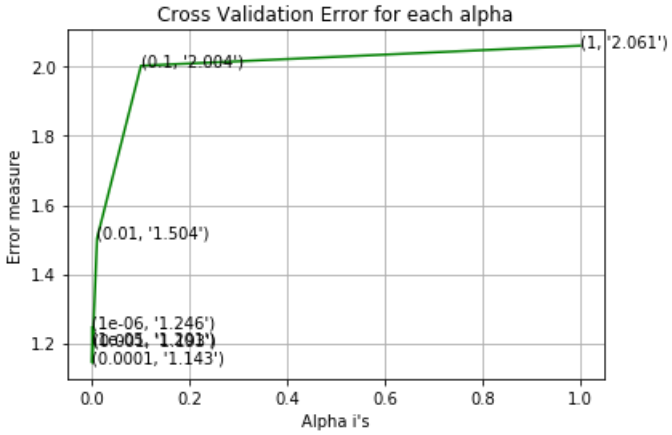
5.2.3 LR(Imbalanced) with top 1k word

In [107]:

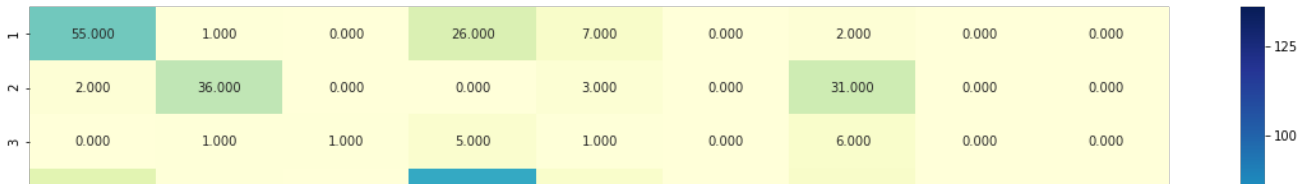
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,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

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

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



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

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

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=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)
tr_loss = log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",tr_loss)
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
cv_loss = log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",cv_loss)
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
test_loss = log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",test_loss)
```

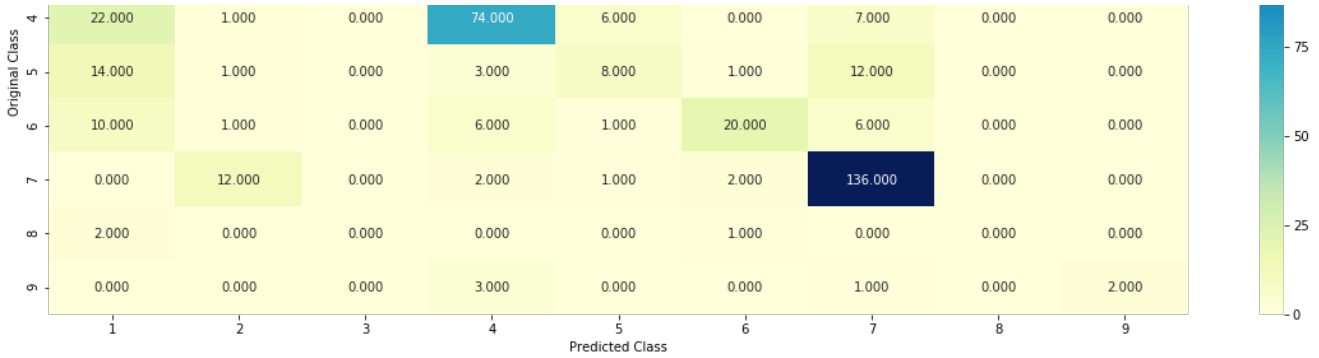
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,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

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

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

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

ittr+=1
df.loc[ittr]= ['LR-Imbalanced', 'ALL','onehot','TFIDF_top1000',"alpha {0}".format(alpha[best_alpha]),tr_loss,cv_loss,test_loss,misclassified_pt*100,'YES']
```

```
for alpha = 1e-06
Log Loss : 1.2457623353361844
for alpha = 1e-05
Log Loss : 1.2012727021965672
for alpha = 0.0001
Log Loss : 1.1428251049939113
for alpha = 0.001
Log Loss : 1.1929752464154515
for alpha = 0.01
Log Loss : 1.5042370103263791
for alpha = 0.1
Log Loss : 2.0035013681587643
for alpha = 1
Log Loss : 2.06137776664582
```



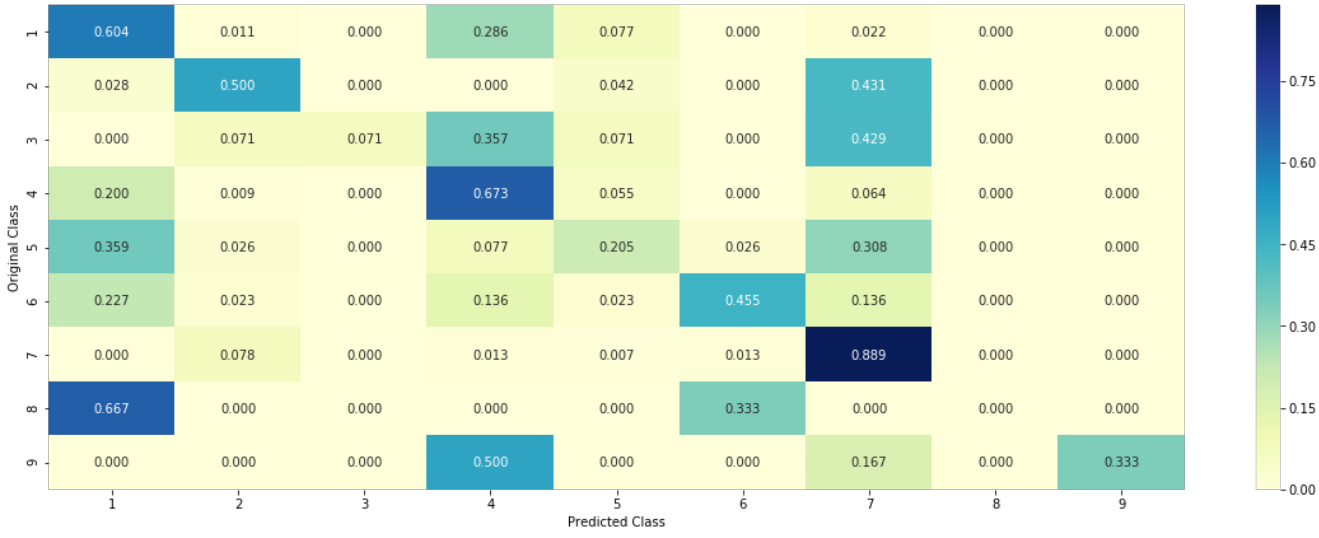

```
For values of best alpha =  0.0001 The train log loss is: 0.4135944107316236
For values of best alpha =  0.0001 The cross validation log loss is: 1.1428251049939113
For values of best alpha =  0.0001 The test log loss is: 1.0502787693856983
Log loss : 1.1428251049939113
Number of mis-classified points : 0.37593984962406013
-------------------- Confusion matrix --------------------
```





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

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



5.2.4 Linear LVM with top 1k words

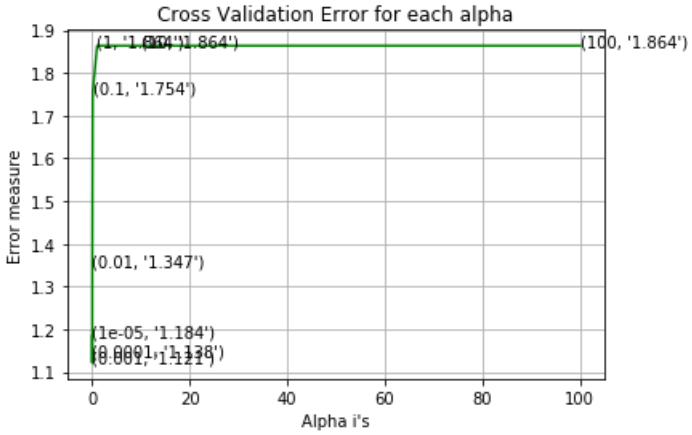
In [108]:

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

# ------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random
```

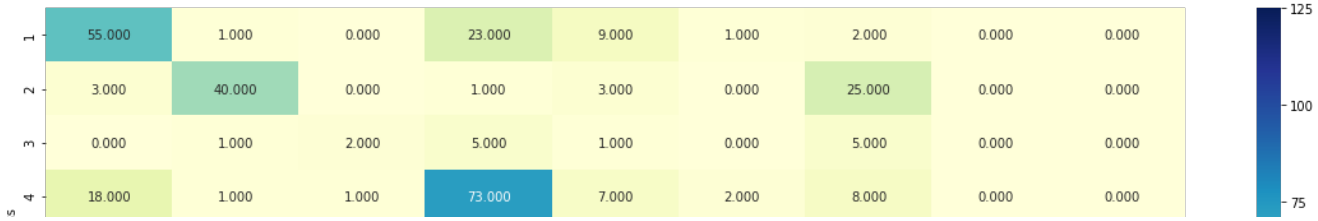
```
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_
state=None)

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



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

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

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


best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge', rando
m_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)
tr_loss = log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",tr_loss)
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
cv_loss = log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",cv_loss)
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
test_loss = log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",test_loss)


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

# --------------------------------
# default parameters
```

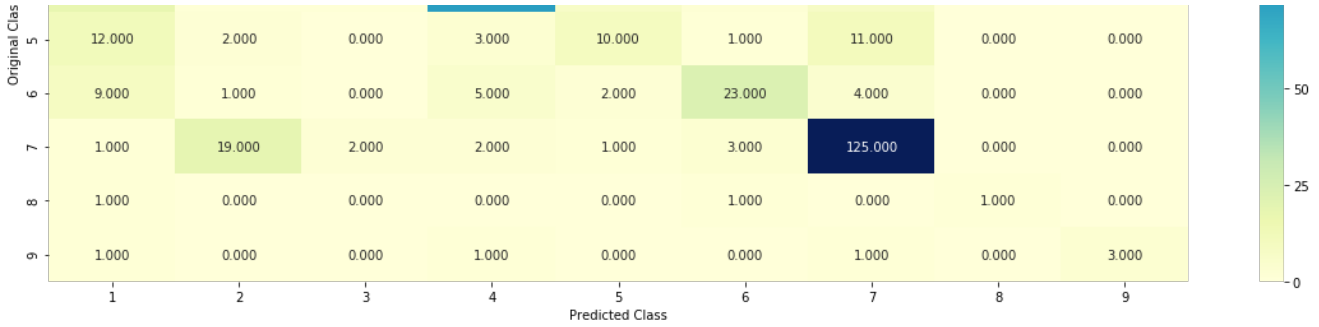
```
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.
001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_
state=None)

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


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



ittr+=1
df.loc[ittr]= ['Linear-SVM', 'ALL','onehot','TFIDF_top1000',"alpha {0}".format(alpha[best_alpha]),tr_lo
ss,cv_loss,test_loss,misclassified_pt*100,'YES']
```

```
for C = 1e-05
Log Loss : 1.1837884878178444
for C = 0.0001
Log Loss : 1.1383973247686408
for C = 0.001
Log Loss : 1.12106190740585
for C = 0.01
Log Loss : 1.3469630131290093
for C = 0.1
Log Loss : 1.753818181532665
for C = 1
Log Loss : 1.863717695900653
for C = 10
Log Loss : 1.8637177524687543
for C = 100
Log Loss : 1.8637177754465537
```

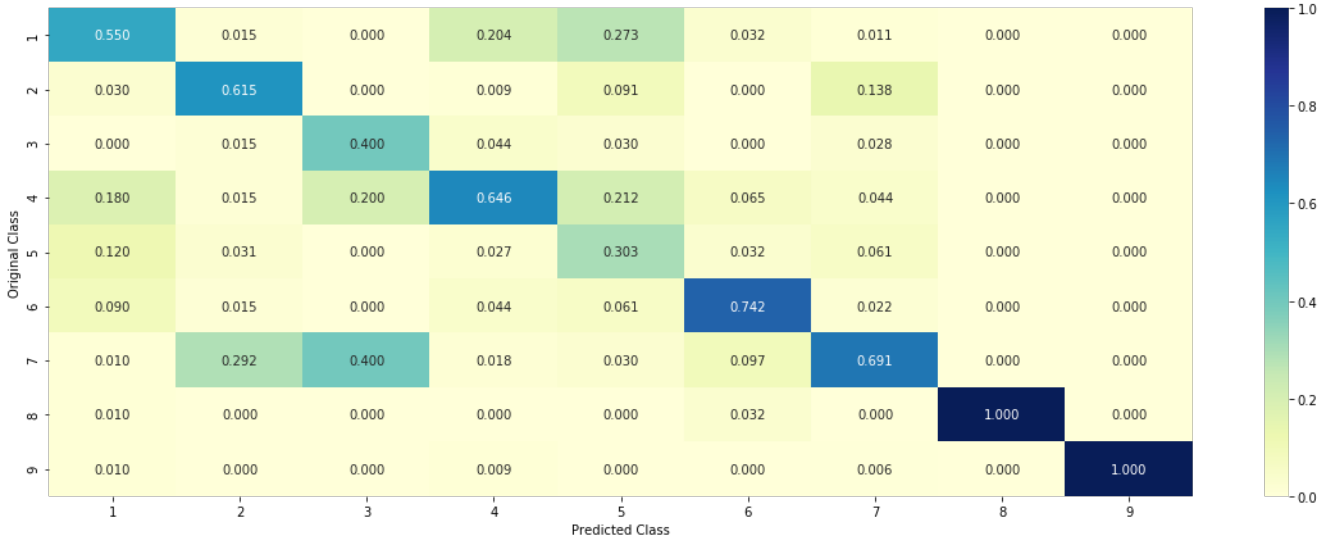



```
For values of best alpha =  0.001 The train log loss is: 0.5659933108434574
For values of best alpha =  0.001 The cross validation log loss is: 1.12106190740585
For values of best alpha =  0.001 The test log loss is: 1.0863079081162335
Log loss : 1.12106190740585
Number of mis-classified points : 0.37593984962406013
-------------------- Confusion matrix --------------------
```

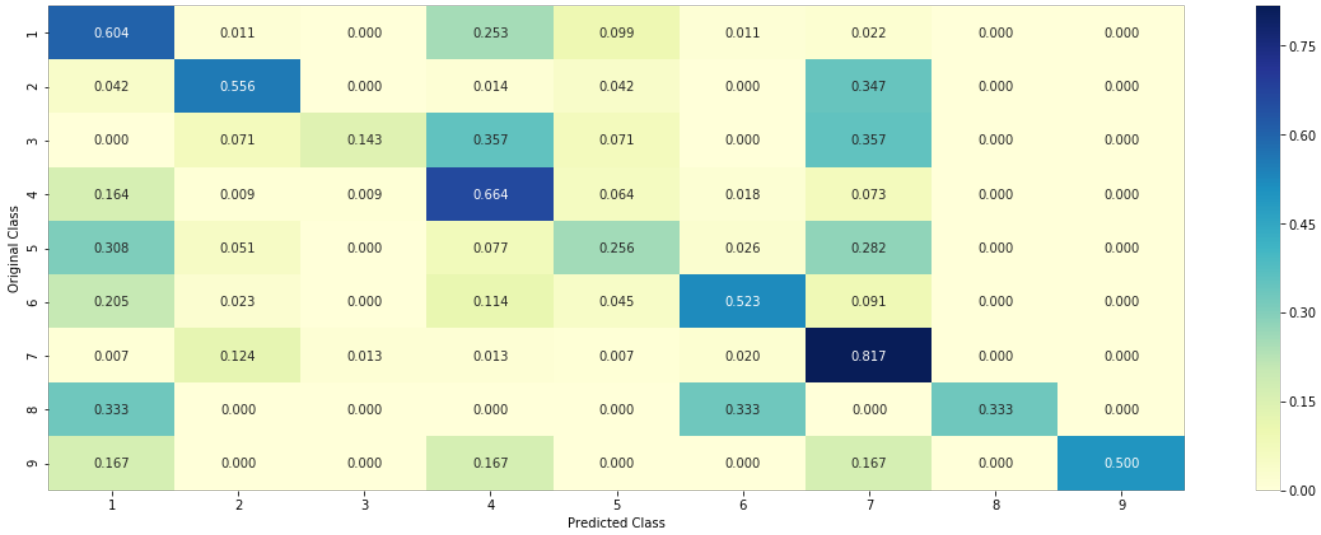




---------- Precision matrix (Columm Sum=1) ----------



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



5.2.5 RF with top 1k words

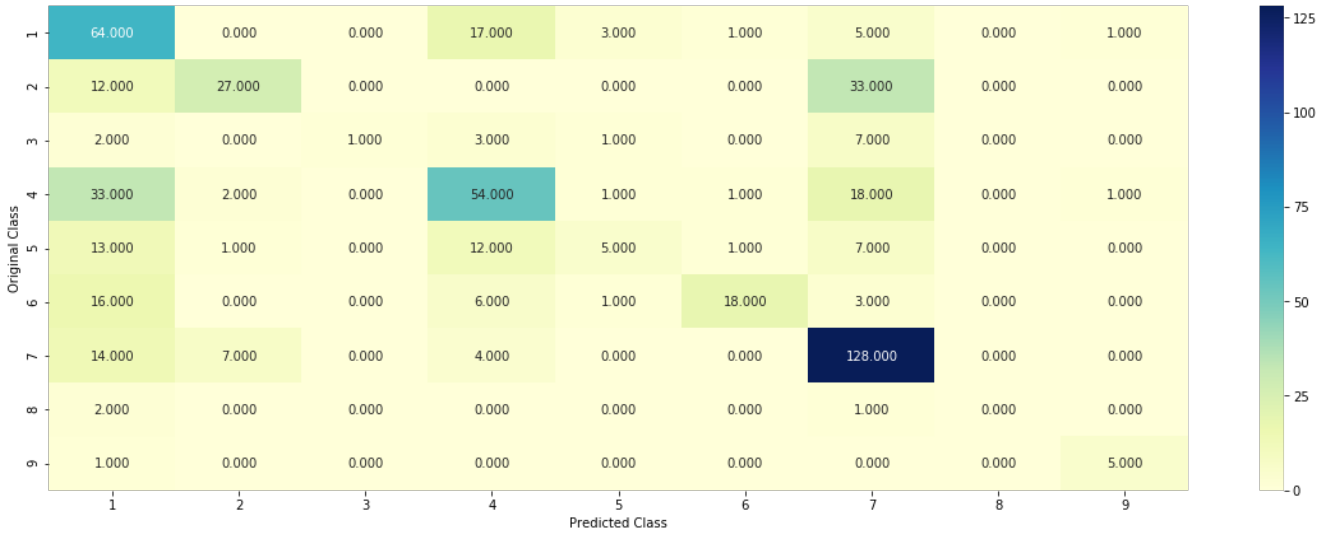
In [109]:

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

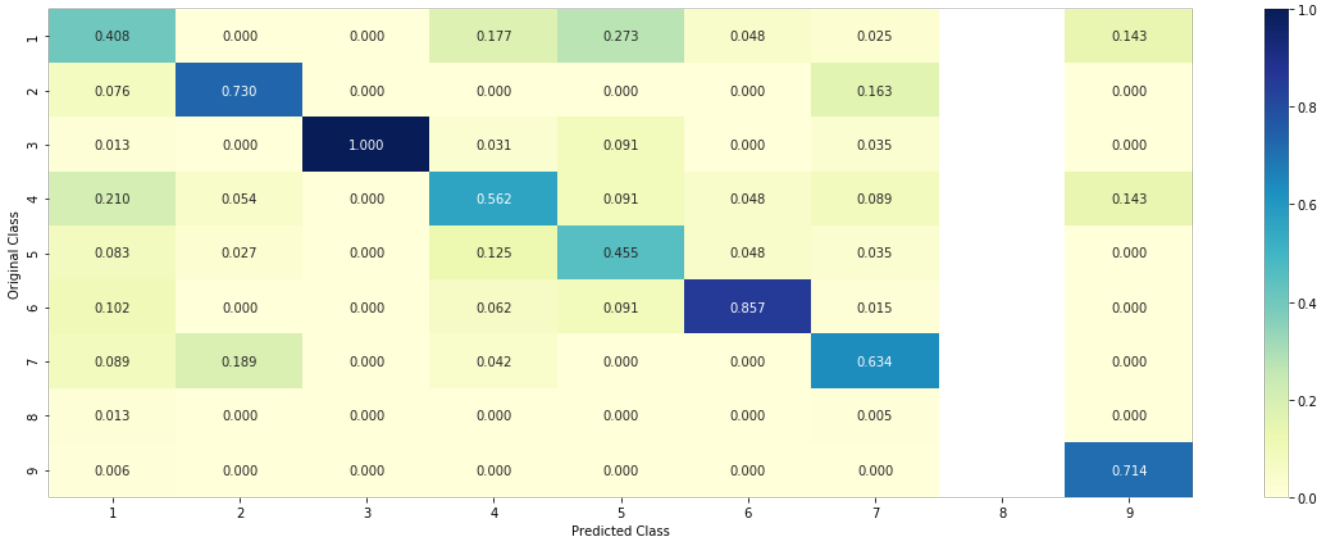
```
# class_weight None)

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

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

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


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

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


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



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


# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_sample
s_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impur
ity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, war
m_start=False,
# class_weight=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-and
-their-construction-2/
# --------------------------------

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


ittr+=1
df.loc[ittr]= ['RF', 'ALL','onehot','TFIDF_top1000',"alpha {0}".format(alpha[int(best_alpha/2)]),tr_los
s,cv_loss,test_loss,misclassified_pt*100,'YES']
```

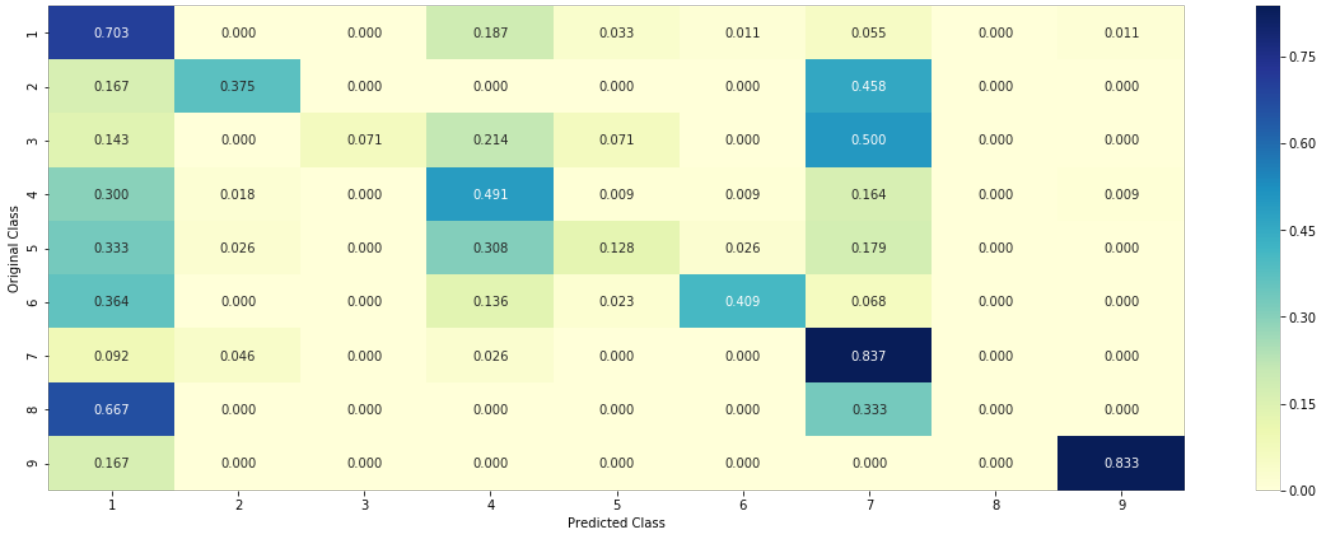
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2854883629435079
for n_estimators = 100 and max depth =  10
Log Loss : 1.2895054540543227
for n_estimators = 200 and max depth =  5
Log Loss : 1.2701931871514858
for n_estimators = 200 and max depth =  10
Log Loss : 1.2825259077141418
for n_estimators = 500 and max depth =  5
Log Loss : 1.2642559223381773
for n_estimators = 500 and max depth =  10
Log Loss : 1.2822159178557884
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2555728912016149
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2782147421359922
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2539047180539313
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2788069988309334
For values of best alpha =  2000 The train log loss is: 0.8360494302575426
For values of best alpha =  2000 The cross validation log loss is: 1.2539047180539313
For values of best alpha =  2000 The test log loss is: 1.204630257178646
Log loss : 1.2539047180539313
Number of mis-classified points : 0.4323308270676692
-------------------- Confusion matrix --------------------
```




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



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



5.2.6 Stacking with top 1k words

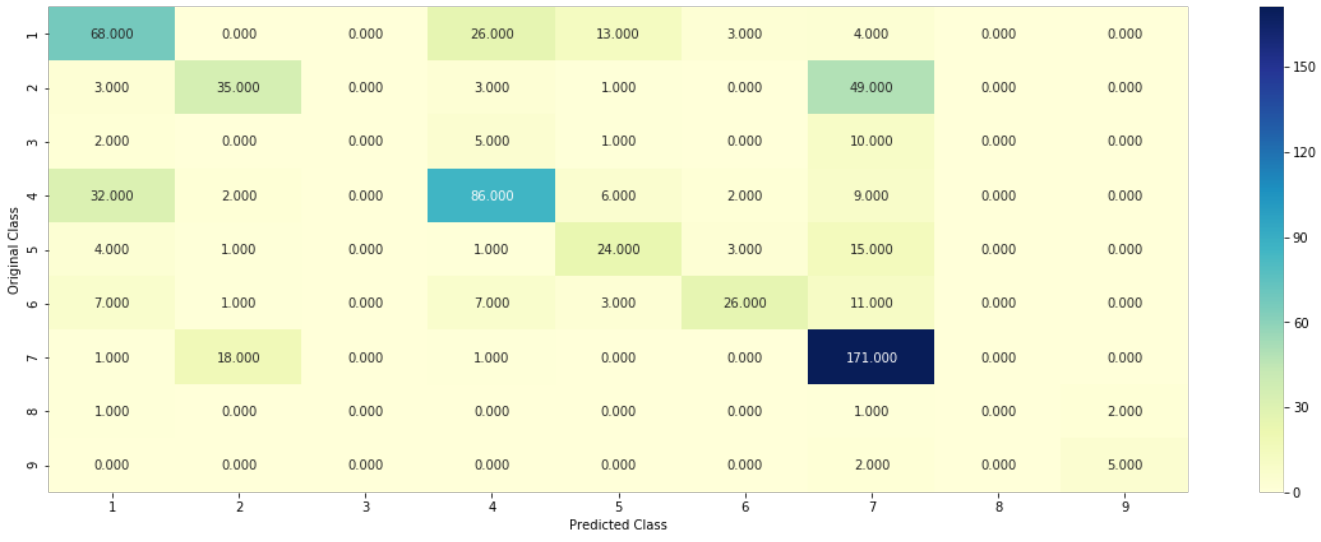
In [110]:

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

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

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


# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
# ------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.
```

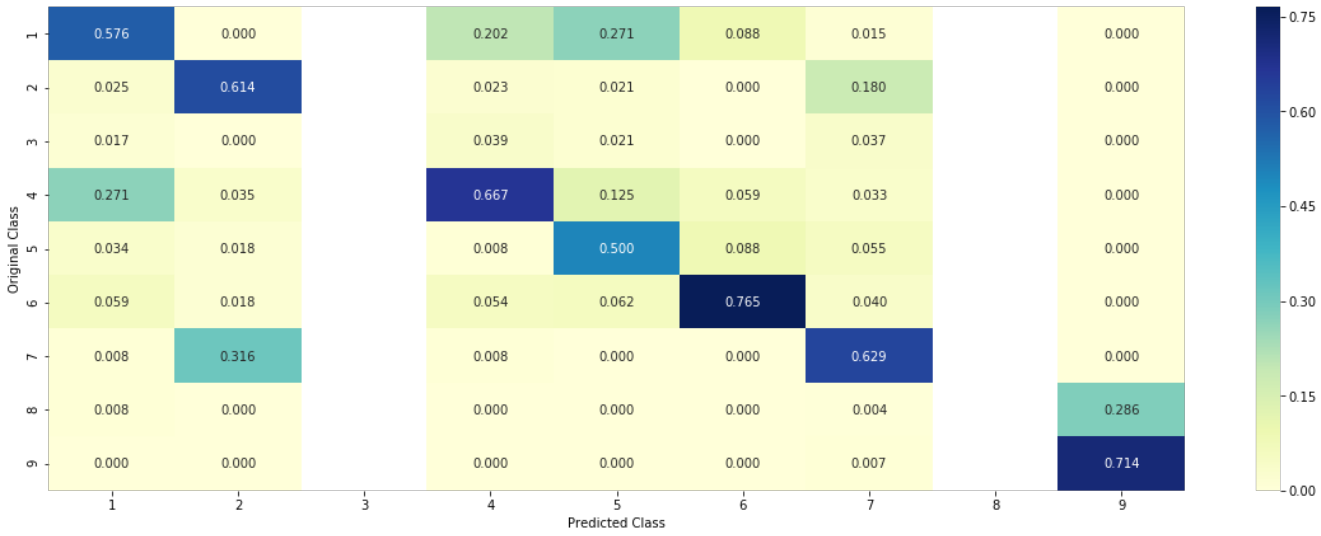
```
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.
001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_
state=None)

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


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

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

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

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


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

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', random_state=0)
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


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

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

```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=Tr
ue)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='sof
t')
vclf.fit(train_x_onehotCoding, train_y)
tr_loss = log_loss(train_y, vclf.predict_proba(train_x_onehotCoding))
cv_loss = log_loss(cv_y, vclf.predict_proba(cv_x_onehotCoding))
test_loss = log_loss(test_y, vclf.predict_proba(test_x_onehotCoding))
misclassified_pt = np.count_nonzero((vclf.predict(test_x_onehotCoding)- test_y))/test_y.shape[0]
print("Log loss (train) on the VotingClassifier :", tr_loss)
print("Log loss (CV) on the VotingClassifier :", cv_loss)
print("Log loss (test) on the VotingClassifier :", test_loss)
print("Number of missclassified point :", misclassified_pt)
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))

ittr+=1
df.loc[ittr]= ['stacking', 'ALL','onehot','TFIDF_top1000','NA',tr_loss,cv_loss,test_loss,misclassified_
pt*100,'NO']
```

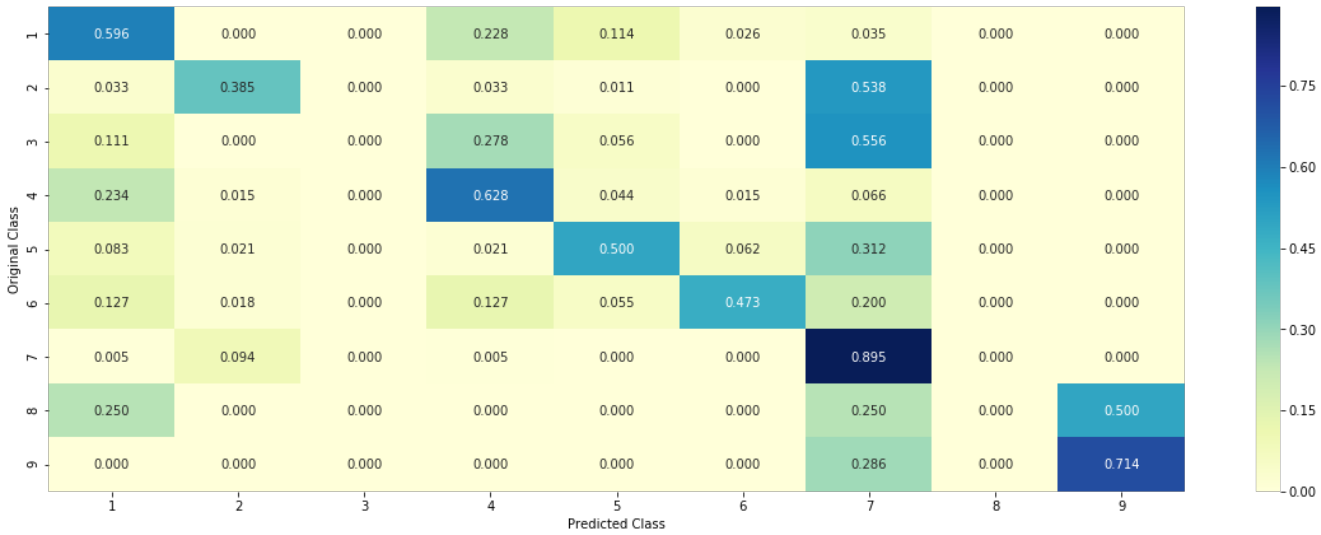
```
Logistic Regression :  Log Loss: 1.11
Support vector machines : Log Loss: 1.86
Naive Bayes : Log Loss: 1.25
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.177
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.031
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.511
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.248
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.532
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 2.038
Log loss (train) on the stacking classifier : 0.5191480537581522
Log loss (CV) on the stacking classifier : 1.2476341542843055
Log loss (test) on the stacking classifier : 1.1867381675919637
Number of missclassified point : 0.37593984962406013
-------------------- Confusion matrix --------------------
```



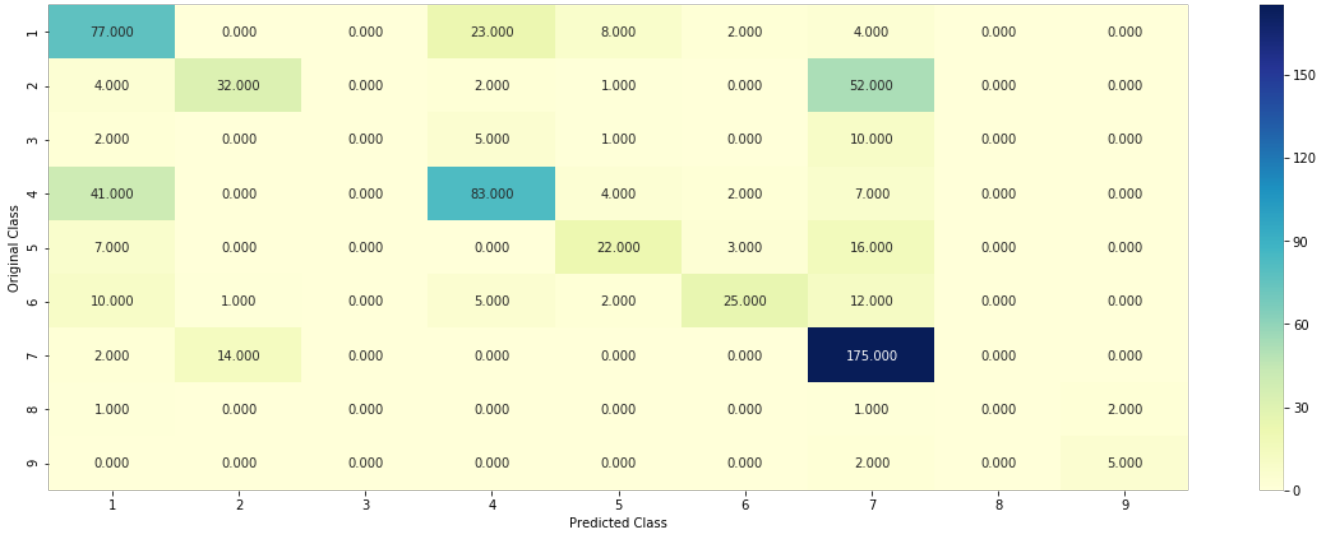
```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



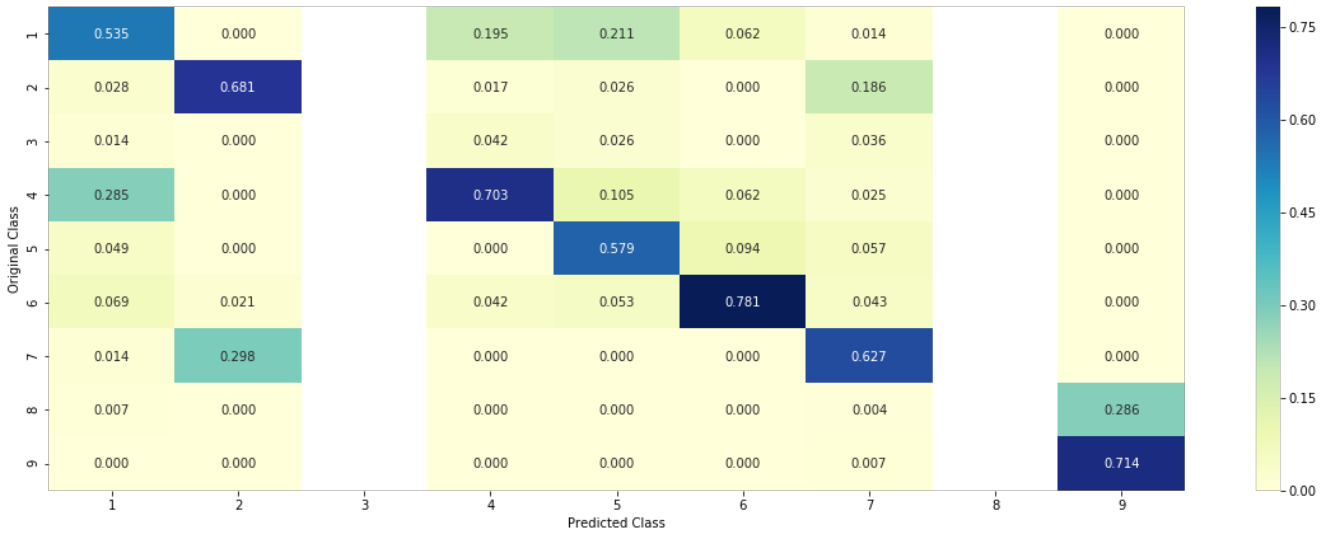

```
-------------------- Recall matrix (Row sum=1) --------------------
```



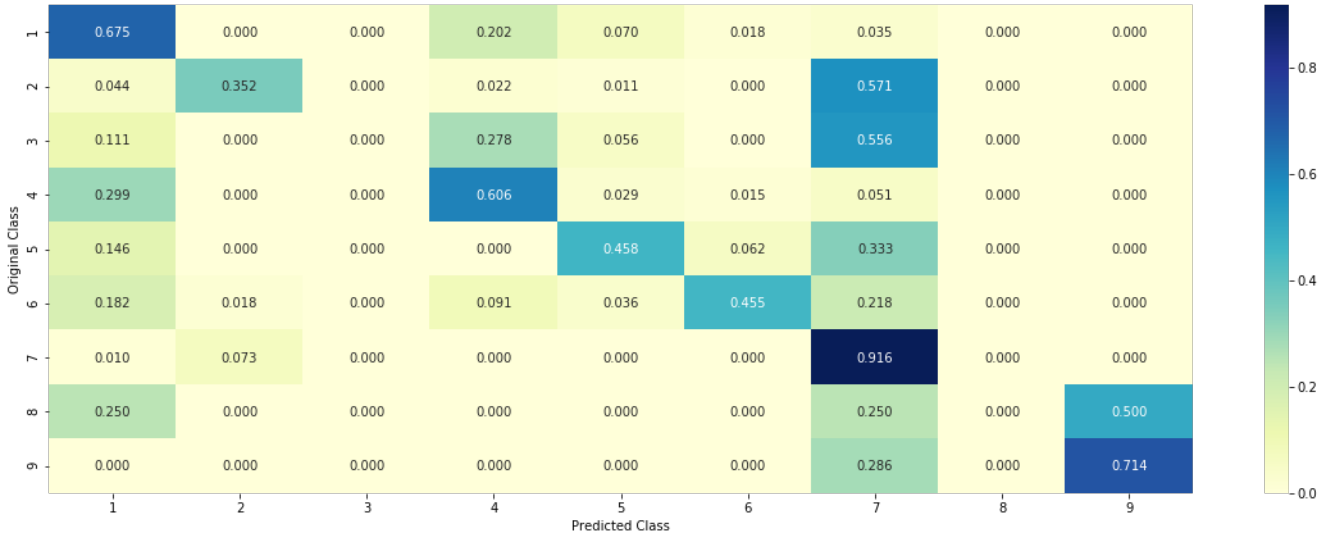

```
Log loss (train) on the VotingClassifier : 0.8106092779198745
Log loss (CV) on the VotingClassifier : 1.2501810040984211
Log loss (test) on the VotingClassifier : 1.2146641431961276
Number of missclassified point : 0.3699248120300752
-------------------- Confusion matrix --------------------
```



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




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



In [111]:

```
df
```

Out[111]:

	ML Model	features	encoding	Vectorizer for text	hyperparameter	train loss	CV loss	test loss	misclassified point(%)	interpretability
0	LR	Gene	onehot	NA	alpha 0.0001	1.023730	1.247220	1.227799	48.872180	YES
1	LR	Variance	onehot	NA	alpha 0.0001	0.744162	1.700559	1.715161	66.165414	YES
2	LR	Variance	onehot	BOW	alpha 0.001	0.762503	1.284959	1.258912	40.601504	YES
3	NB	ALL	onehot	BOW	alpha 0.1	0.867622	1.316218	1.288403	43.045113	YES
4	KNN	ALL	response	BOW	alpha 11	0.635593	1.123739	1.091815	43.045113	NO
5	LR-Balanced	ALL	onehot	BOW	alpha 0.001	0.622764	1.159779	1.178137	43.045113	YES
6	LR-Imbalanced	ALL	onehot	BOW	alpha 0.001	0.621263	1.226061	1.212409	43.045113	YES
7	Linear-SVM	ALL	onehot	BOW	alpha 0.01	0.743907	1.214963	1.199751	43.045113	YES
8	RF	ALL	onehot	BOW	alpha 2000	0.694803	1.218812	1.164701	43.045113	YES
9	RF	ALL	response	BOW	alpha 10	0.069985	1.322195	1.313385	43.045113	YES

	ML Model	features	encoding	Vectorizer for text	hyperparameter	train loss	CV loss	test loss	misclassified point(%)	interpretability
10	stacking	ALL	onehot	BOW	NA	0.907982	1.250097	1.247291	36.541353	NO
11	NB	ALL	onehot	TFIDF	alpha 1000	0.857259	1.271170	1.234097	43.233083	YES
12	LR-Balanced	ALL	onehot	TFIDF	alpha 0.001	0.570661	1.178925	1.140754	43.233083	YES
13	LR-Imbalanced	ALL	onehot	TFIDF	alpha 0.001	0.567898	1.210615	1.170250	43.233083	YES
14	Linear-SVM	ALL	onehot	TFIDF	alpha 0.01	0.667899	1.222345	1.189560	43.233083	YES
15	RF	ALL	onehot	TFIDF	alpha 2000	0.674842	1.225454	1.146807	43.233083	YES
16	stacking	ALL	onehot	TFIDF	NA	0.847765	1.216256	1.202688	36.541353	NO
17	NB	ALL	onehot	TFIDF_top1000	alpha 1e-05	0.496629	1.246987	1.210254	40.789474	YES
18	LR-Balanced	ALL	onehot	TFIDF_top1000	alpha 0.0001	0.421196	1.108430	1.033799	40.789474	YES
19	LR-Imbalanced	ALL	onehot	TFIDF_top1000	alpha 0.0001	0.413594	1.142825	1.050279	40.789474	YES
20	Linear-SVM	ALL	onehot	TFIDF_top1000	alpha 0.001	0.565993	1.121062	1.086308	40.789474	YES
21	RF	ALL	onehot	TFIDF_top1000	alpha 2000	0.836049	1.253905	1.204630	40.789474	YES
22	stacking	ALL	onehot	TFIDF_top1000	NA	0.810609	1.250181	1.214664	36.992481	NO

LR with CountVectorizer (unigrams and bigrams)

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

Prepare data for unigram and biagram

In [112]:

```
text_vectorizer = CountVectorizer(ngram_range =(1,2))
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
#normalize
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)
#process test data
test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEXT'])
test_text_feature_onehotCoding = normalize(test_text_feature_onehotCoding, axis=0)
#process CV data
cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])
cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding, axis=0)

train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_onehotCoding)).tocsr()
test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_onehotCoding)).tocsr()
cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).tocsr()
```

LR (Balanced)

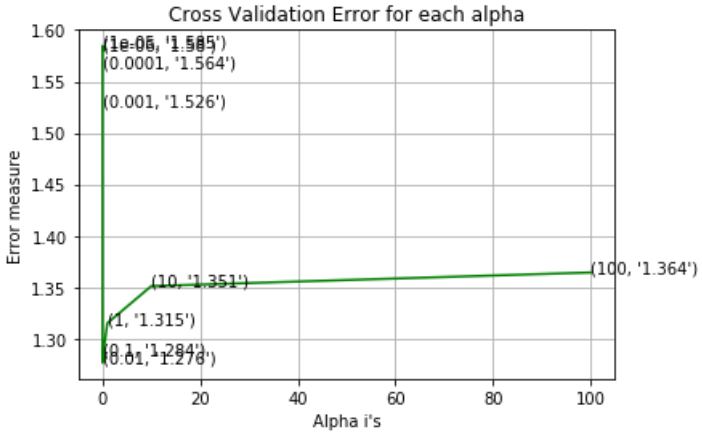
In [113]:

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

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

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

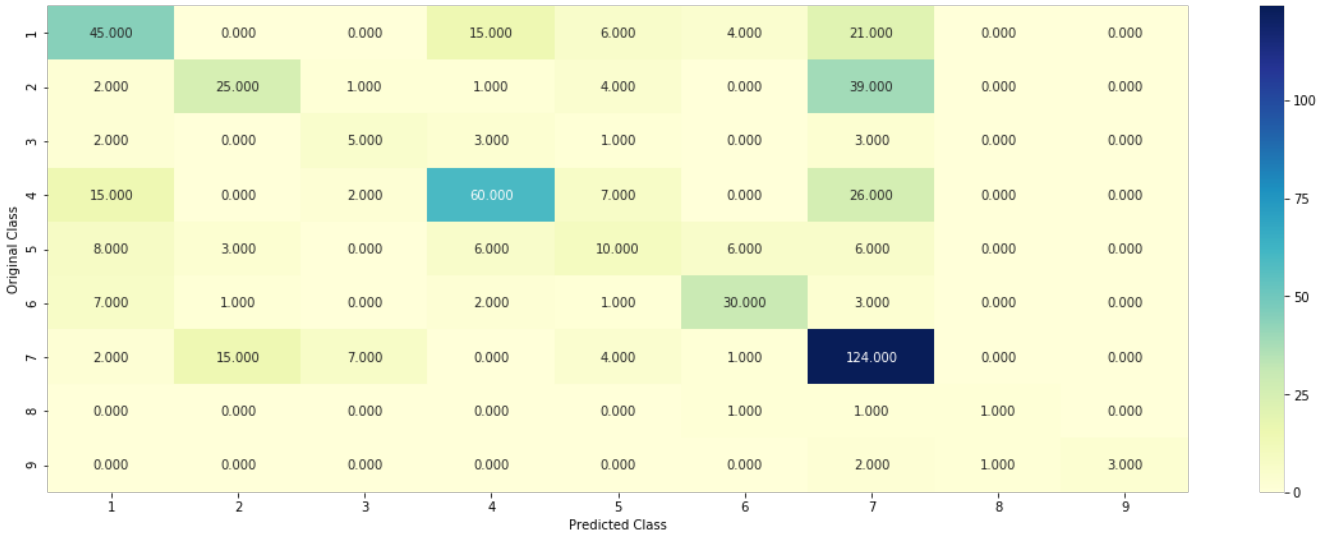
```
# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of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

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

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


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

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


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

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

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

```
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)

ittr+=1
df.loc[ittr]= ['LR-Balanced', 'ALL','onehot','BOW_NGRAM',"alpha {0}".format(alpha[best_alpha]),tr_loss,
cv_loss,test_loss,misclassified_pt*100,'YES']
```

```
for alpha = 1e-06
Log Loss : 1.5802589257740431
for alpha = 1e-05
Log Loss : 1.5847638969943694
for alpha = 0.0001
Log Loss : 1.564049340155952
for alpha = 0.001
Log Loss : 1.5259403629276216
for alpha = 0.01
Log Loss : 1.2764212832934339
for alpha = 0.1
Log Loss : 1.284227553079052
for alpha = 1
Log Loss : 1.31502823113231
for alpha = 10
Log Loss : 1.3510720124271762
for alpha = 100
Log Loss : 1.3642833097082714
```

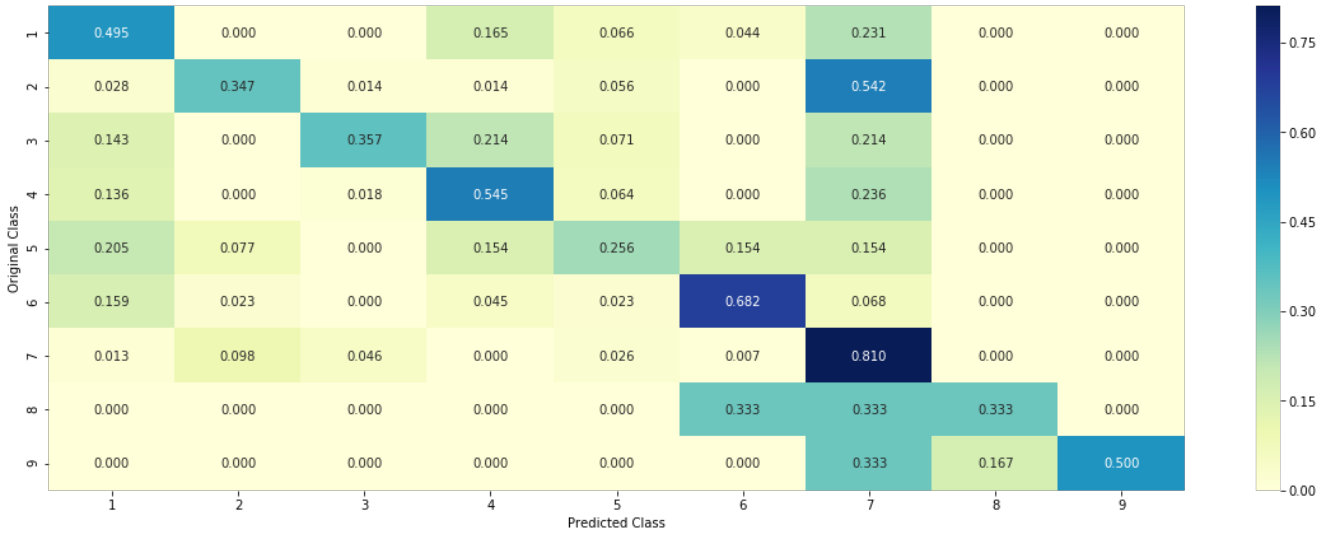



```
For values of best alpha =  0.01 The train log loss is: 0.788850288959823
For values of best alpha =  0.01 The cross validation log loss is: 1.2764212832934339
For values of best alpha =  0.01 The test log loss is: 1.2642937047615677
Log loss : 1.2764212832934339
Number of mis-classified points : 0.43045112781954886
-------------------- Confusion matrix --------------------
```




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

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



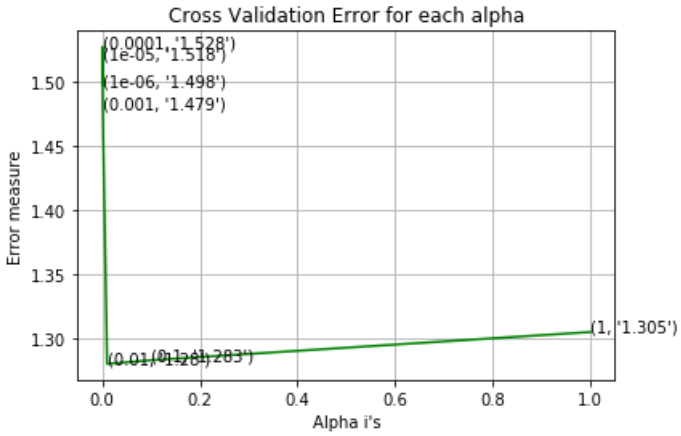
LR Imbalanced

In [114]:

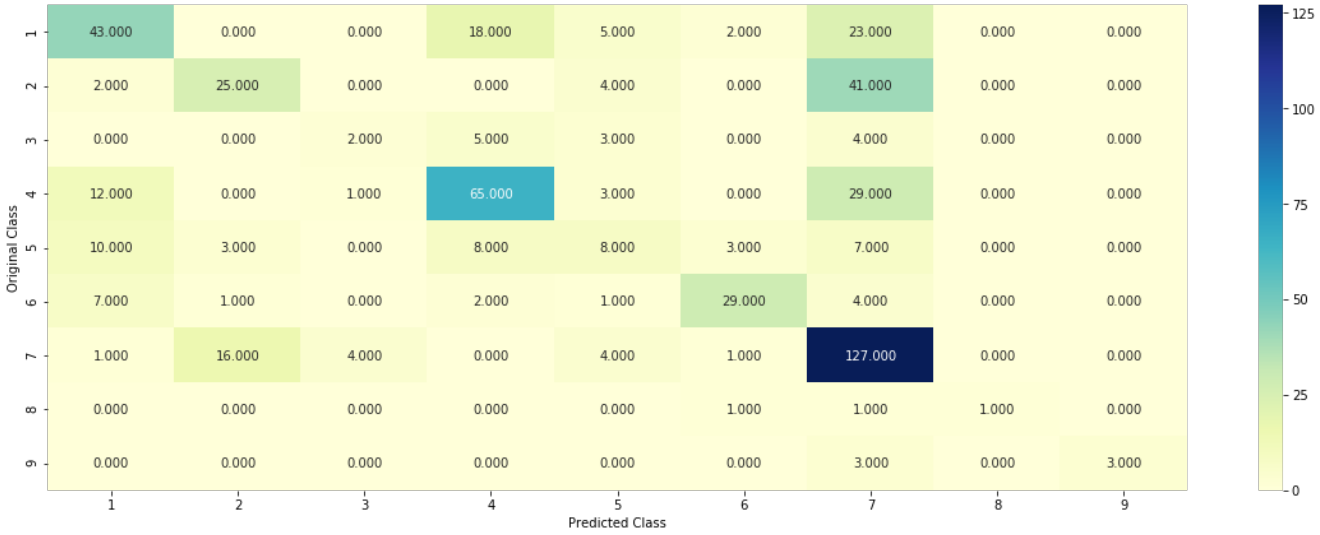
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,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

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

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



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

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

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

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=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)
tr_loss = log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",tr_loss)
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
cv_loss = log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",cv_loss)
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
test_loss = log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",test_loss)



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

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

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

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

ittr+=1
df.loc[ittr]= ['LR-Imbalanced', 'ALL','onehot','BOW_NGRAM',"alpha {0}".format(alpha[best_alpha]),tr_los
s,cv_loss,test_loss,misclassified_pt*100,'YES']
```

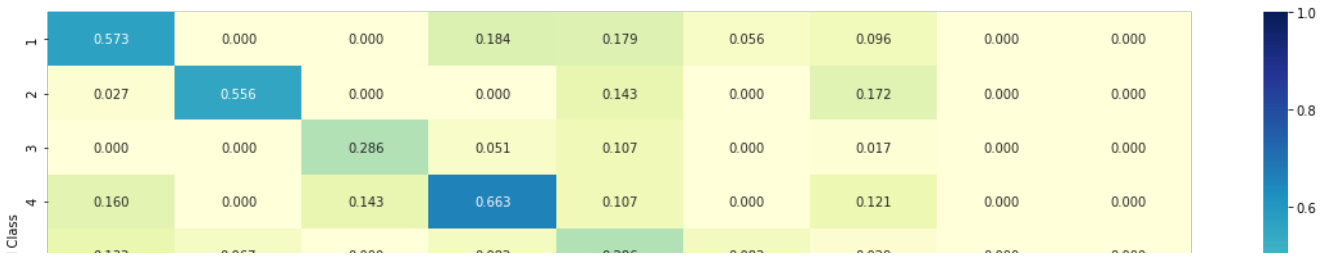
```
for alpha = 1e-06
Log Loss : 1.4975305026909922
for alpha = 1e-05
Log Loss : 1.5182924055305316
for alpha = 0.0001
Log Loss : 1.5275013751181394
for alpha = 0.001
Log Loss : 1.4789293233830487
for alpha = 0.01
Log Loss : 1.2798262019052655
for alpha = 0.1
Log Loss : 1.282672780976192
for alpha = 1
Log Loss : 1.3046706211831867
```

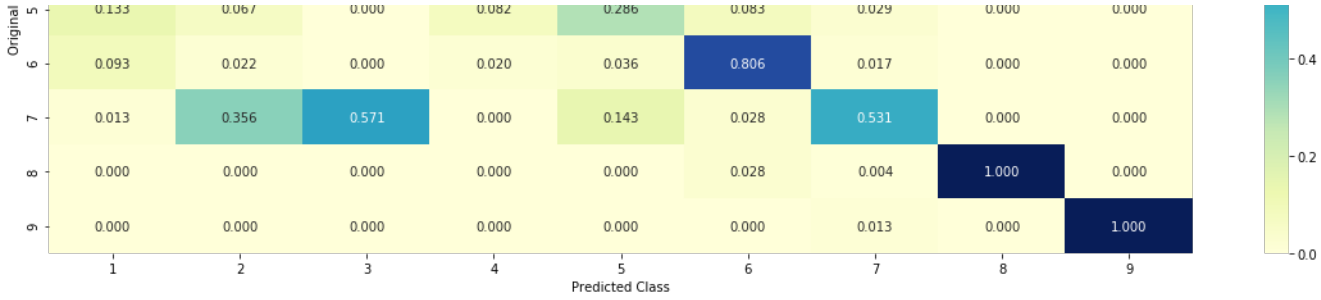



```
For values of best alpha =  0.01 The train log loss is: 0.7643480625083388
For values of best alpha =  0.01 The cross validation log loss is: 1.2798262019052655
For values of best alpha =  0.01 The test log loss is: 1.27462526422163
Log loss : 1.2798262019052655
Number of mis-classified points : 0.43045112781954886
-------------------- Confusion matrix --------------------
```

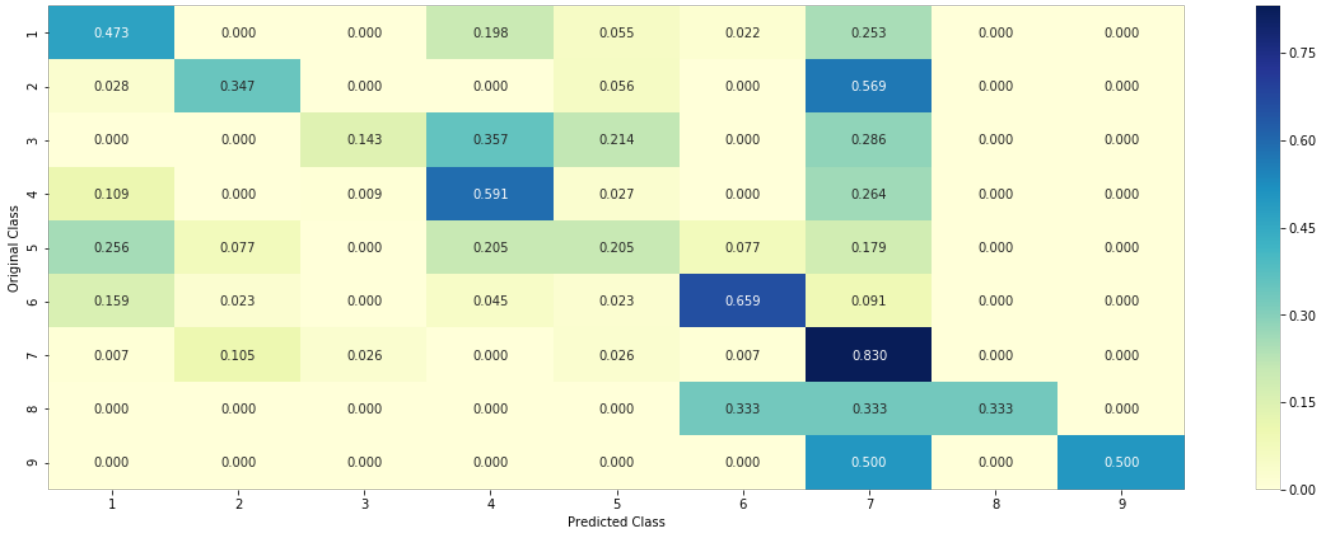



```
-------------------- Precision matrix (Column Sum=1) --------------------
```





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



In [115]:

```
df
```

Out[115]:

	ML Model	features	encoding	Vectorizer for text	hyperparameter	train loss	CV loss	test loss	misclassified point(%)	interpretability
0	LR	Gene	onehot	NA	alpha 0.0001	1.023730	1.247220	1.227799	48.872180	YES
1	LR	Variance	onehot	NA	alpha 0.0001	0.744162	1.700559	1.715161	66.165414	YES
2	LR	Variance	onehot	BOW	alpha 0.001	0.762503	1.284959	1.258912	40.601504	YES
3	NB	ALL	onehot	BOW	alpha 0.1	0.867622	1.316218	1.288403	43.045113	YES
4	KNN	ALL	response	BOW	alpha 11	0.635593	1.123739	1.091815	43.045113	NO
5	LR-Balanced	ALL	onehot	BOW	alpha 0.001	0.622764	1.159779	1.178137	43.045113	YES
6	LR-Imbalanced	ALL	onehot	BOW	alpha 0.001	0.621263	1.226061	1.212409	43.045113	YES
7	Linear-SVM	ALL	onehot	BOW	alpha 0.01	0.743907	1.214963	1.199751	43.045113	YES
8	RF	ALL	onehot	BOW	alpha 2000	0.694803	1.218812	1.164701	43.045113	YES
9	RF	ALL	response	BOW	alpha 10	0.069985	1.322195	1.313385	43.045113	YES
10	stacking	ALL	onehot	BOW	NA	0.907962	1.258097	1.247201	36.541353	NO
11	NB	ALL	onehot	TFIDF	alpha 1000	0.857259	1.271170	1.234097	43.233083	YES
12	LR-Balanced	ALL	onehot	TFIDF	alpha 0.001	0.570661	1.178925	1.140754	43.233083	YES
13	LR-Imbalanced	ALL	onehot	TFIDF	alpha 0.001	0.567898	1.210615	1.170250	43.233083	YES
14	Linear-SVM	ALL	onehot	TFIDF	alpha 0.01	0.667899	1.222345	1.189560	43.233083	YES
15	RF	ALL	onehot	TFIDF	alpha 2000	0.674842	1.225454	1.146807	43.233083	YES
16	stacking	ALL	onehot	TFIDF	NA	0.847765	1.216256	1.202688	36.541353	NO
17	NB	ALL	onehot	TFIDF_top1000	alpha 1e-05	0.496629	1.246987	1.210254	40.789474	YES

	ML Model	features	encoding	text	hyperparameter	train loss	CV loss	test loss	misclassified point(%)	interpretability
18	LR-Balanced	ALL	onehot	TFIDF_top1000	alpha 0.0001	0.421446	1.108430	1.033799	40.789474	YES
19	LR-Imbalanced	ALL	onehot	TFIDF_top1000	alpha 0.0001	0.413594	1.142825	1.050279	40.789474	YES
20	Linear-SVM	ALL	onehot	TFIDF_top1000	alpha 0.001	0.565993	1.121062	1.086308	40.789474	YES
21	RF	ALL	onehot	TFIDF_top1000	alpha 2000	0.836049	1.253905	1.204630	40.789474	YES
22	stacking	ALL	onehot	TFIDF_top1000	NA	0.810609	1.250181	1.214664	36.992481	NO
23	LR-Balanced	ALL	onehot	BOW_NGRAM	alpha 0.01	0.788850	1.276421	1.264294	36.992481	YES
24	LR-Imbalanced	ALL	onehot	BOW_NGRAM	alpha 0.01	0.764348	1.279826	1.274625	36.992481	YES

Try any of the feature engineering techniques discussed in the course to reduce the CV and test log-loss to a value less than 1.0

Till now we have got best performance for Logistic regression balanced data set and for featurization we have used TFIDF with top 1000 words.

We are going to use some featurization technique to reduce the train and test error even more

In [24]:

```
train_y = np.array(list(train_df['Class']))
test_y = np.array(list(test_df['Class']))
cv_y = np.array(list(cv_df['Class']))
```

In [25]:

```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCod
ing))
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))
```

In [30]:

```
text_vectorizer = TfidfVectorizer(max_features=2000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
#normalize
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)
#process test data
test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEXT'])
test_text_feature_onehotCoding = normalize(test_text_feature_onehotCoding, axis=0)
#process CV data
cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])
cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding, axis=0)

"""train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_onehotCoding)).tocsr(
)
test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_onehotCoding)).tocsr()
cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).tocsr()"""

train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_onehotCoding))
test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_onehotCoding))
cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding))
```

In [31]:

```
text_vectorizer_ = TfidfVectorizer()
```

In [41]:

```
from tqdm import tqdm
top_1000_fe = text_vectorizer.get_feature_names()
#train_df_['TEXT'] = [[x+" "+x_*10 for x_ in x.split() if x_ in top_1000_fe] for x in train_df['TEXT']]

out_list = []
for x in tqdm(train_df['TEXT']):
    x__= "feat "
```

```
    #out_list_ = []
    for x_ in top_1000_fe:
        if x_ in x:
            x__ = x__ + x_*50 + " "
        else:
            continue

    out_list.append(x__)

text_vectorizer_fit = text_vectorizer_.fit(out_list)
myfeature_train = text_vectorizer_fit.transform(out_list)
```

```
100%|██████████| 2124/2124 [02:12<0
0:00, 16.08it/s]
```

In [42]:

```
#train_df_['TEXT'] = [[x+" "+x_*10 for x_ in x.split() if x_ in top_1000_fe] for x in train_df['TEXT']]

out_list = []
for x in tqdm(test_df['TEXT']):
    x__ = "feat "
    #out_list_ = []
    for x_ in top_1000_fe:
        if x_ in x:
            x__ = x__ + x_*50 + " "
        else:
            continue

    out_list.append(x__)

myfeature_test = text_vectorizer_fit.transform(out_list)
```

```
100%|██████████| 665/665 [00:39<0
0:00, 15.51it/s]
```

In [43]:

```
#train_df_['TEXT'] = [[x+" "+x_*10 for x_ in x.split() if x_ in top_1000_fe] for x in train_df['TEXT']]

"""out_list = []
for x in tqdm(cv_df['TEXT']):
    for x_ in top_1000_fe:
        if x_ in x:
            out_list.append("myfeat"+" "+x_*10)
        else:
            out_list.append("myfeat")"""

out_list = []
for x in tqdm(cv_df['TEXT']):
    x__ = "feat "
    #out_list_ = []
    for x_ in top_1000_fe:
        if x_ in x:
            x__ = x__ + x_*50 + " "
        else:
            continue

    out_list.append(x__)

myfeature_cv = text_vectorizer_fit.transform(out_list)
```

```
100%|██████████| 532/532 [00:31<0
0:00, 20.16it/s]
```

In [44]:

```
train_x_onehotCoding_ = hstack((train_x_onehotCoding, myfeature_train)).tocsr()
test_x_onehotCoding_ = hstack((test_x_onehotCoding, myfeature_test)).tocsr()
cv_x_onehotCoding_ = hstack((cv_x_onehotCoding, myfeature_cv)).tocsr()
```

In [45]:

```
print(train_x_onehotCoding.shape)
print(test_x_onehotCoding.shape)
print(cv_x_onehotCoding.shape)
```

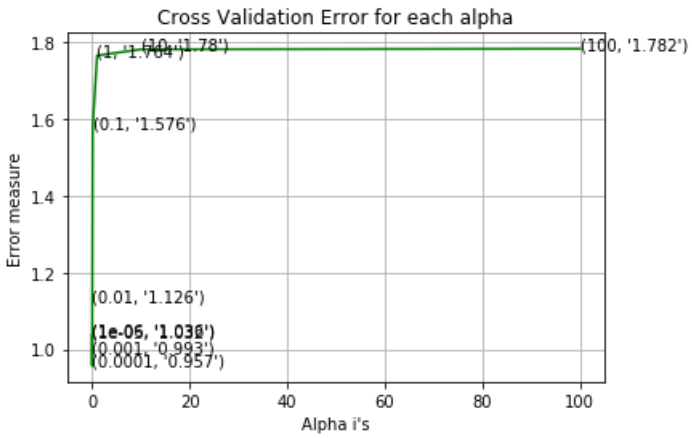
```
(2124, 4194)
(665, 4194)
(532, 4194)
```

In [53]:

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

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

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


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

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

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


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

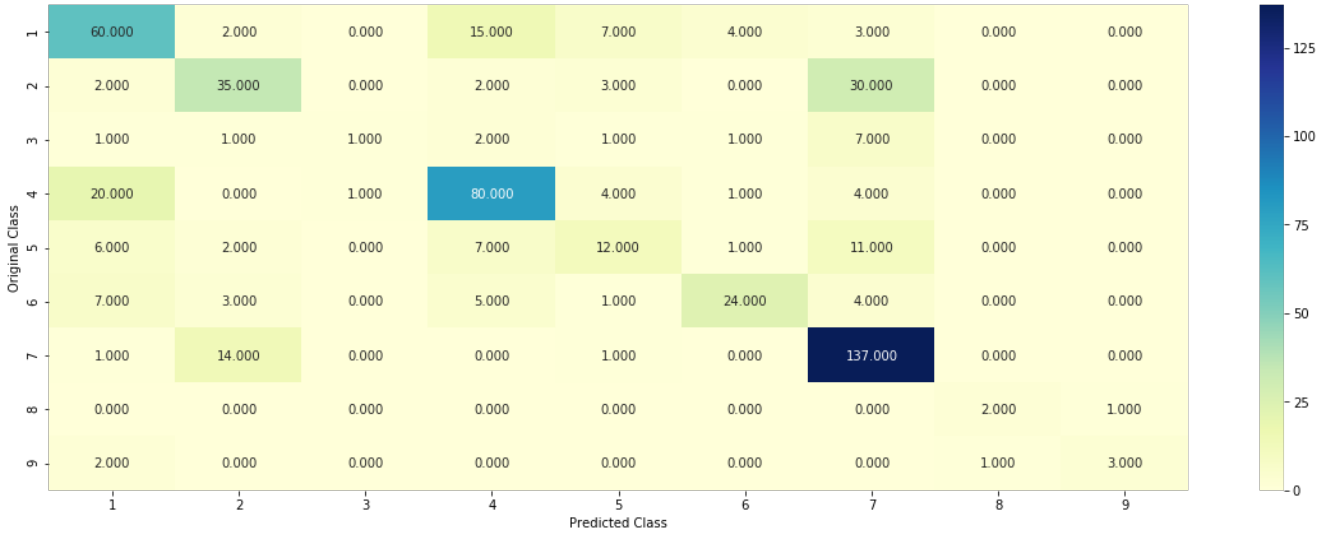
```
clf.fit(train_x_onehotCoding_, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding_, train_y)

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


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

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

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

```
for alpha = 1e-06
Log Loss : 1.0362578439898422
for alpha = 1e-05
Log Loss : 1.032085319361448
for alpha = 0.0001
Log Loss : 0.9571145324604985
for alpha = 0.001
Log Loss : 0.992946384785905
for alpha = 0.01
Log Loss : 1.1262979612780297
for alpha = 0.1
Log Loss : 1.5760463949121886
for alpha = 1
Log Loss : 1.7640981908171363
for alpha = 10
Log Loss : 1.7801839631332008
for alpha = 100
Log Loss : 1.7817781413638805
```



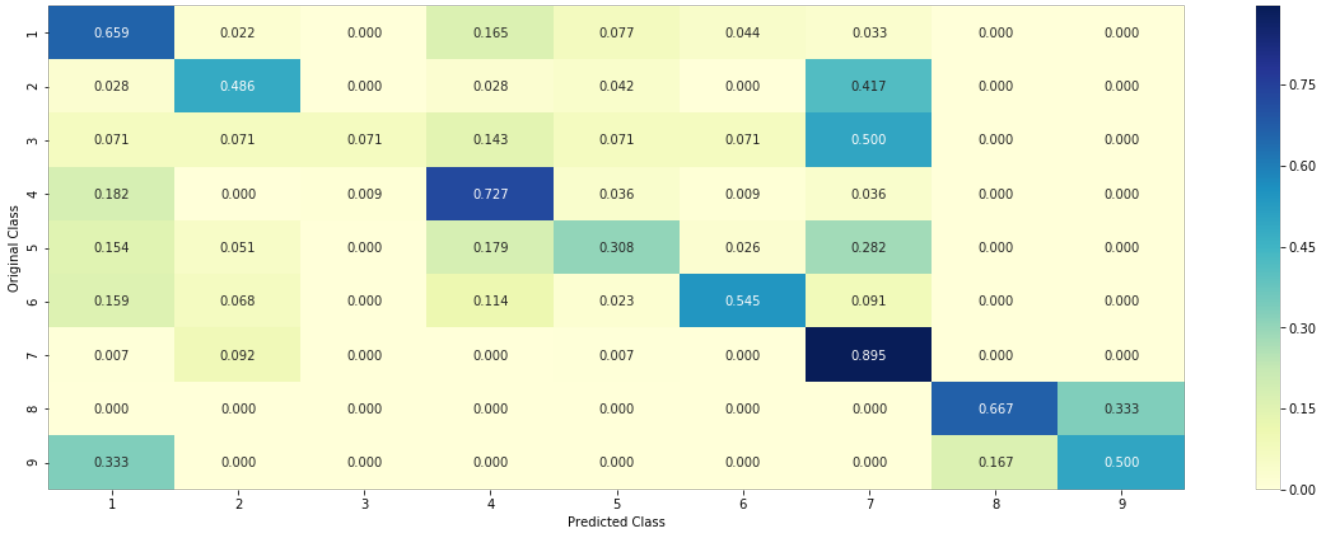

```
For values of best alpha =  0.0001 The train log loss is: 0.4078555132843562
```

```
For values of best alpha =  0.0001 The cross validation log loss is: 0.9571145324604985
For values of best alpha =  0.0001 The test log loss is: 0.9559826288633524
Log loss : 0.9571145324604985
Number of mis-classified points : 0.33458646616541354
-------------------- Confusion matrix --------------------
```




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


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



After featurization our Train Log loss: 0.4078 CV log loss:0.9571 Test log loss: 0.9559

It does not seem like an over fitting model

In the above featurization techinque we used the below algorithm

1. Get the top 2000 important words from text data
2. see if that specific important feature is present in the text data
3. if present give more waitage to the important feature word by adding it to a string multiple times
4. Get TFIDF of the strings whery we have added important features multiple times sample wise

Why we used this technique ?

As in the EDA of text data we saw some classed are hugely depend on some word, (we have got proper word clouds class wise) so we decided to play with these words to improve the accuracy. For every model we used to see class 8 was not properly handled and it has a very less data point. But after our featurization technique we can handle the class 8 preety well.