Personalized cancer diagnosis

Assignment 3-

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

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

- 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.

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [1]:

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

from mlxtend.classifier import StackingClassifier

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

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:

```
data = pd.read_csv('E:\\Machine Learning\\Assignments\\Personalized_Cancer_Diagnosis\\Data\\traini
ng_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("E:\\Machine Learning\\Assignments\\Personalized_Cancer_Diagnosis\\Data\\tr
aining_text",sep="\|\|",engine="python",names=["ID","TEXT"],skiprows=1)
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

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

Out[3]:

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

3.1.3. Preprocessing of text

In [4]:

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


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

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

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

In [5]:

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

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 172.12538458699999 seconds
```

In [6]:

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

Out[6]:

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

In [7]:

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

Out[7]:

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

In [8]:

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

In [9]:

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

Out[9]:

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

3.1.4. Test, Train and Cross Validation Split

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

In [10]:

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

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

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

In [11]:

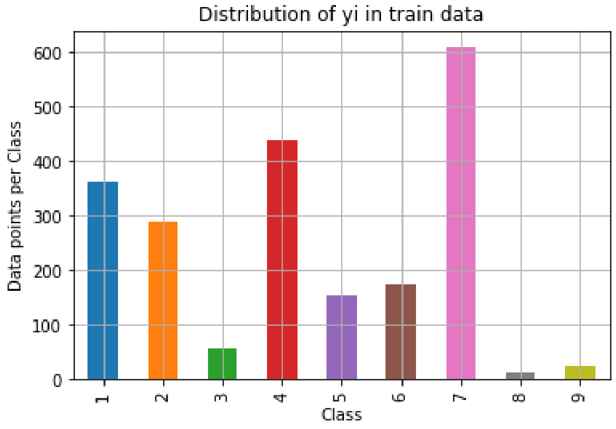
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

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

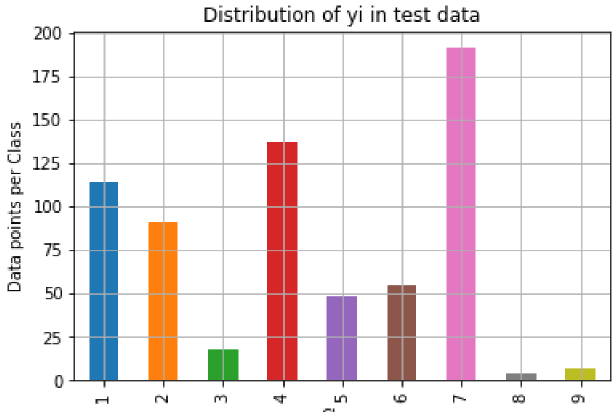
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [12]:

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

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

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


print('-'*80)
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.rou
nd((test_class_distribution.values[i]/test_df.shape[0]*100), 3), '%)')

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

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

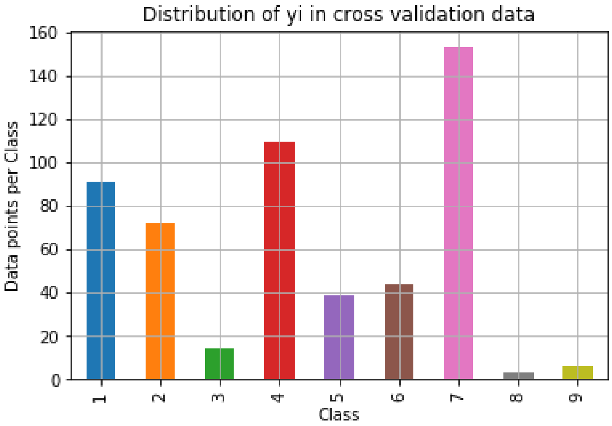


```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298 %)
Number of data points in class 3 : 57 ( 2.684 %)
Number of data points in class 9 : 24 ( 1.13 %)
Number of data points in class 8 : 12 ( 0.565 %)
--------------------------------------------------------------------------------
```



Class

```
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.

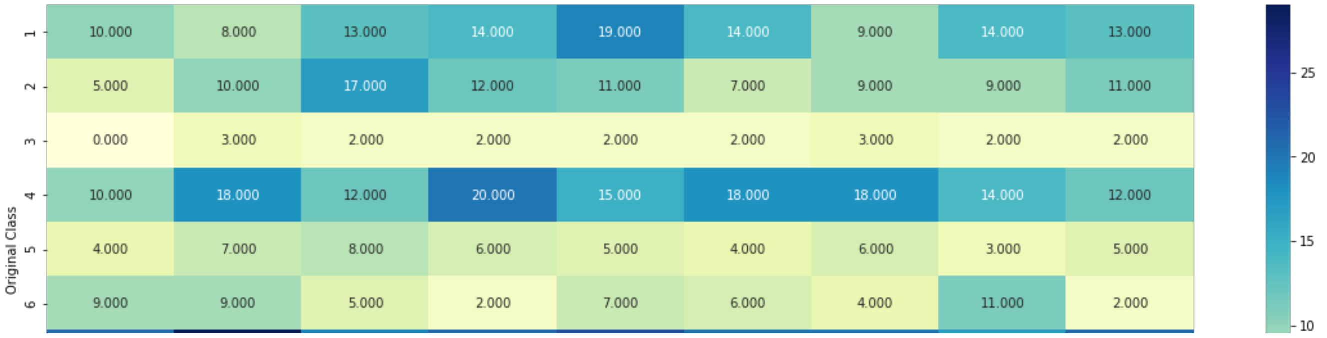
In [13]:

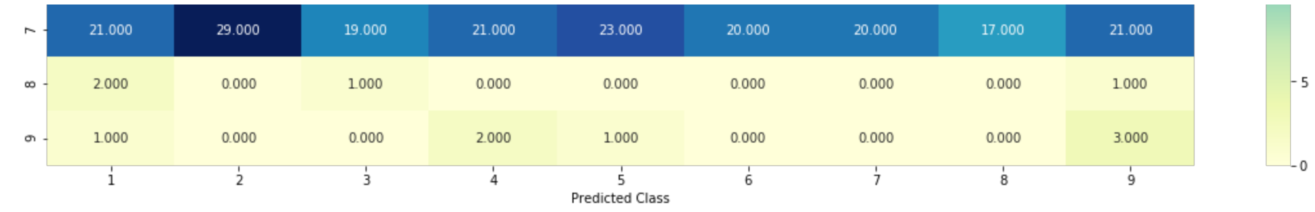
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j
    
    A =(((C.T)/(C.sum(axis=1))).T)
    #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 [14]:

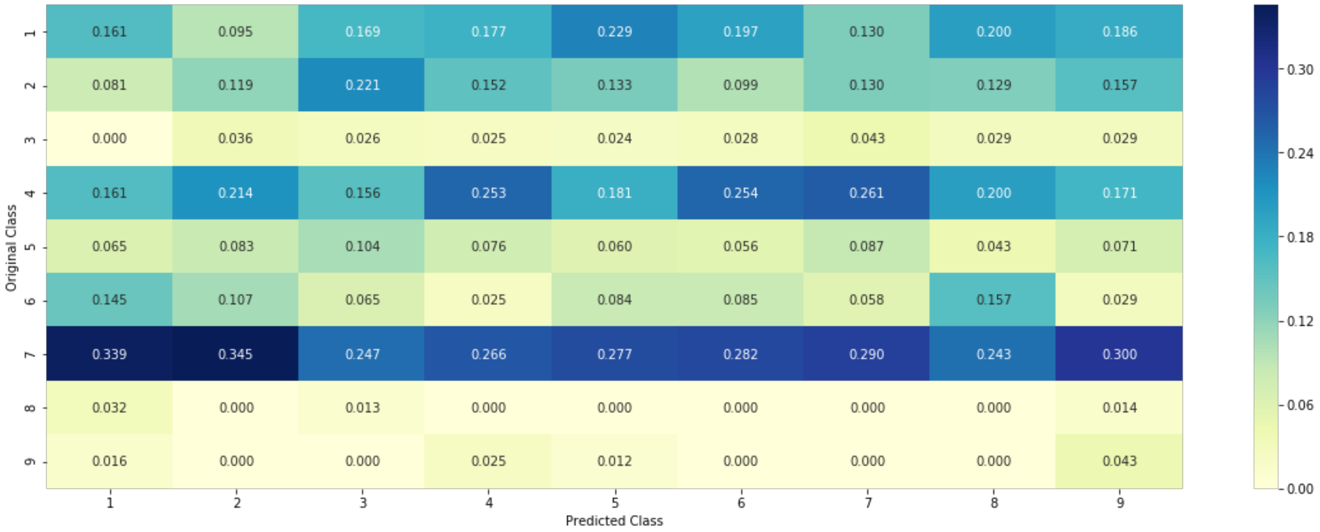
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

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


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

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

```
Log loss on Cross Validation Data using Random Model 2.460858301177185
Log loss on Test Data using Random Model 2.476731515527584
-------------------- Confusion matrix --------------------
```

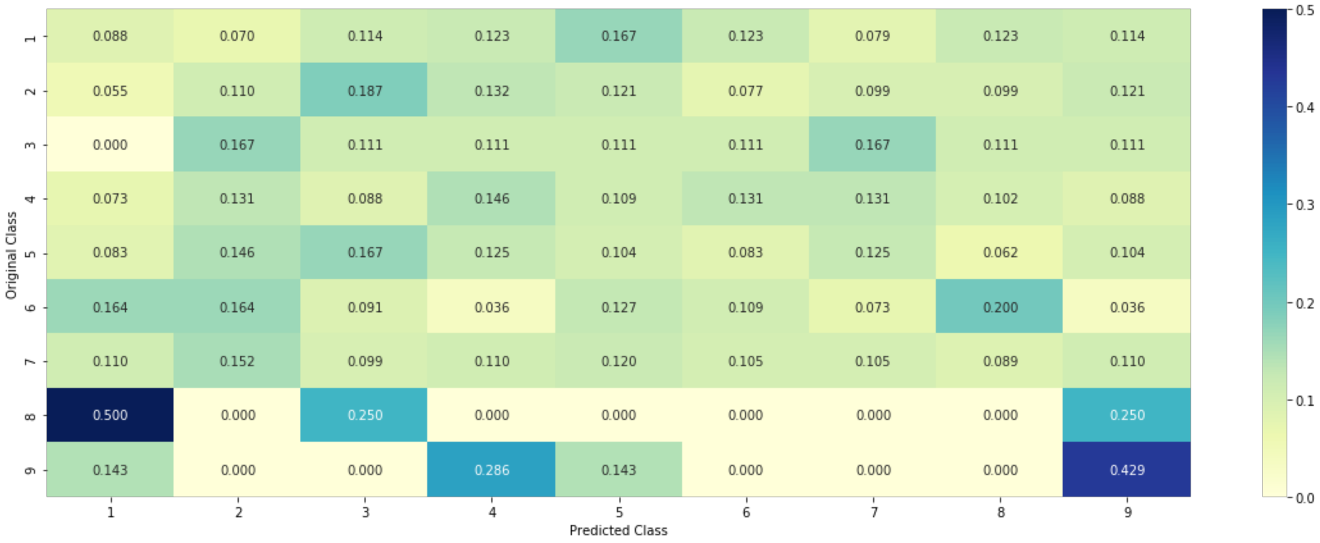




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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



3.3 Univariate Analysis

In [15]:

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

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

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

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

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

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.037878787878787888, 0.068181818181818177,
0.13636363636363635, 0.25, 0.19318181818181818, 0.037878787878787888, 0.037878787878787888,
0.037878787878787888],
    #       'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366,
0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408
163265307, 0.056122448979591837],
    #       'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177,
0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #       'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608,
0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608,
0.060606060606060608, 0.060606060606060608],
    #       'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917,
0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081
761006289, 0.062893081761006289],
    #       'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295,
0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702,
0.066225165562913912, 0.066225165562913912],
    #       'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334,
0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999,
0.066666666666666666, 0.066666666666666666],
    #       ...
    #      }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

In [16]:

```
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 : 229
BRCA1     173
TP53      104
EGFR       92
BRCA2      81
PTEN       75
BRAF       69
KIT        67
ALK        47
ERBB2      45
PDGFRA     40
Name: Gene, dtype: int64
```

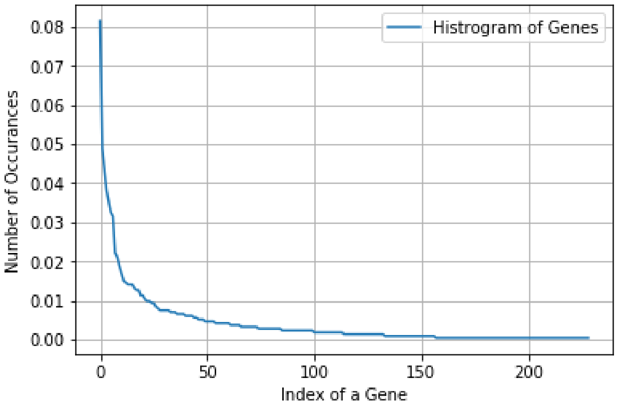
In [17]:

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

```
Ans: There are 229 different categories of genes in the train data, and they are distibuted as fol
lows
```

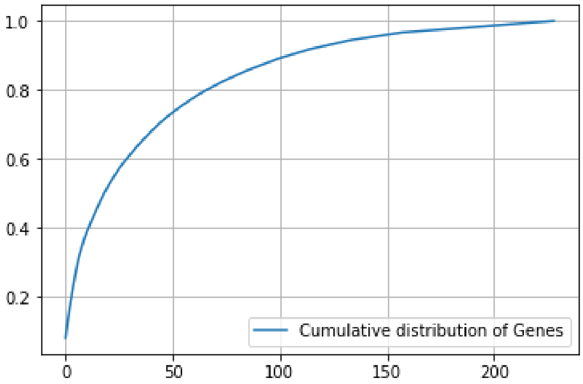
In [18]:

```
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 [19]:

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



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

In [20]:

```
#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
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 [21]:

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

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

In [22]:

```
# 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 [23]:

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

Out[23]:

```
326       ROS1
2641     BRCA1
1746      MSH2
3217     NTRK1
2387    PTPN11
Name: Gene, dtype: object
```

In [24]:

```
gene_vectorizer.get_feature_names()
```

Out[24]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid2',
 'arid5b',
 'asxl1',
 'asxl2',
 'atm',
 'atrx',
 'aurka',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcl2',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk6',
 'cdkn1a',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cdkn2c',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctla4',
 'ctnnb1',
 'ddr2',
 'dicer1',
 'dnmt3b',
 'dusp4',
 'egfr',
 'elf3',
 'ep300',
 'epas1',
 'epcam',
 'erbb2',
 'erbb3',
 'erbb4',
 'ercc2',
 'ercc3',
 'ercc4',
 'erg',
 'errfi1',
 'esr1',
 'etv1',
 'etv6',
 'ewsr1',
 'ezh2',
 'fanca',
 'fancc',
 'fat1',
 'fbxw7',
 'fgf19',
 'fgf4',
 'fgfr1',
 'fgfr2',
 'fgfr3',
 'fgfr4',
 'flt3',
 'foxa1',
 'foxo1',
 'foxp1',
 'gata3',
 'gli1',
 'gna11',
 'gnaq',
 'gnas',
 'h3f3a',
 'hist1h1c',
 'hla',
 'hnf1a',
 'hras',
 'idh1',
 'idh2',
 'igf1r',
 'ikbke',
 'ikzf1',
 'inpp4b',
 'jak1',
 'jak2',
 'jun',
 'kdm5c',
 'kdm6a',
 'kdr',
 'keap1',
 'kit',
 'kmt2a',
 'kmt2b',
 'kmt2c',
 'knstrn',
 'kras',
 'lats1',
 'lats2',
 'map2k1',
 'map2k2',
 'map2k4',
 'map3k1',
 'mapk1',
 'med12',
 'mef2b',
 'men1',
 'met',
 'mga',
 'mlh1',
 'mpl',
 'msh2',
 'msh6',
 'mtor',
 'myc',
 'mycn',
 'myd88',
 'nf1',
 'nf2',
 'nfe2l2',
 'nfkbia',
 'nkx2',
 'notch1',
 'notch2',
 'npm1',
 'nras',
 'nsd1',
 'ntrk1',
 'ntrk2',
 'ntrk3',
 'nup93',
 'pak1',
 'pax8',
 'pbrm1',
 'pdgfra',
 'pdgfrb',
 'pik3ca',
 'pik3cb',
 'pik3cd',
 'pik3r1',
 'pik3r2',
 'pim1',
 'pms1',
 'pms2',
 'pole',
 'ppp2r1a',
 'ppp6c',
 'prdm1',
 'ptch1',
 'pten',
 'ptpn11',
 'ptprd',
 'ptprt',
 'rab35',
 'rac1',
 'rad21',
 'rad50',
 'rad51b',
 'rad51c',
 'rad51d',
 'rad54l',
 'raf1',
 'rara',
 'rasa1',
 'rb1',
 'rbm10',
 'ret',
 'rheb',
 'rhoa',
 'rit1',
 'rnf43',
 'ros1',
 'rras2',
 'runx1',
 'rxra',
 'setd2',
 'sf3b1',
 'shq1',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'srsf2',
 'stag2',
 'stat3',
 'stk11',
 'tcf3',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vhl',
 'xpo1',
 'xrcc2',
 'yap1']
```

In [25]:

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of g
ene feature: (2124, 228)
```

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

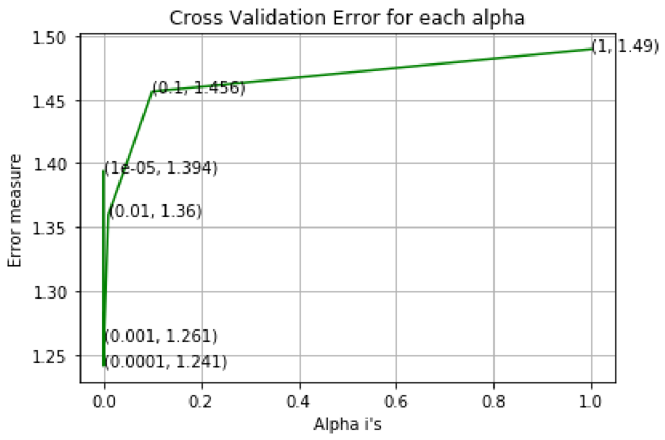
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [26]:

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

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

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

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


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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.3938713009277255
For values of alpha =  0.0001 The log loss is: 1.241083309052142
For values of alpha =  0.001 The log loss is: 1.261084985992044
For values of alpha =  0.01 The log loss is: 1.3596415361560659
For values of alpha =  0.1 The log loss is: 1.4563430207404924
For values of alpha =  1 The log loss is: 1.4895636187177526
```



```
For values of best alpha =  0.0001 The train log loss is: 1.0340129130870106
For values of best alpha =  0.0001 The cross validation log loss is: 1.241083309052142
For values of best alpha =  0.0001 The test log loss is: 1.2256863802045463
```

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 [27]:

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

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  229  genes in train dataset?
Ans
1. In test data 646 out of 665 : 97.14285714285714
2. In cross validation data 510 out of  532 : 95.86466165413535
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

In [28]:

```
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 : 1936
Truncating_Mutations     59
Deletion                 47
Amplification            38
Fusions                  23
Overexpression            4
Q61R                      3
Q61H                      3
G12V                      3
G12A                      2
P130S                     2
Name: Variation, dtype: int64
```

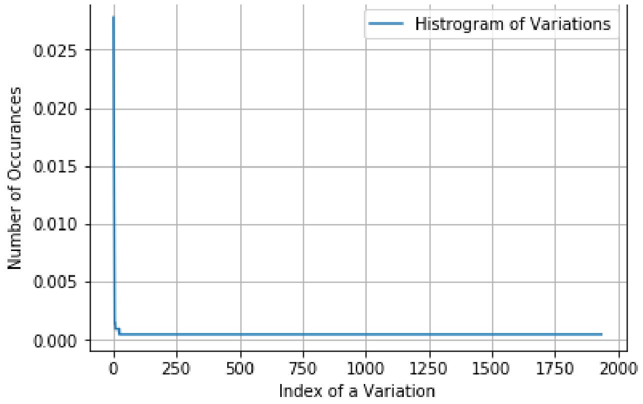
In [29]:

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

```
Ans: There are 1936 different categories of variations in the train data, and they are distibuted
as follows
```

In [30]:

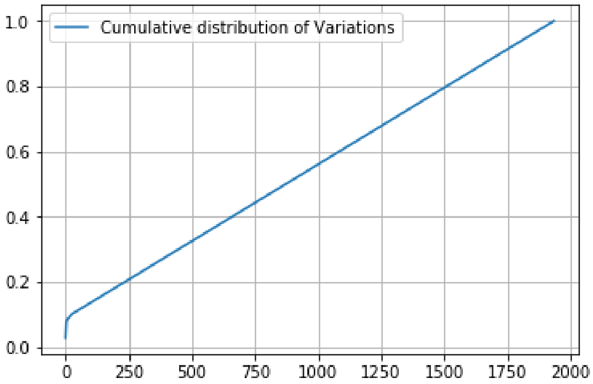
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [31]:

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

```
[0.02777778 0.04990584 0.06779661 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [32]:

```
# 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 [33]:

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

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

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

In [34]:

```
# 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 [35]:

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

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The
shape of Variation feature: (2124, 1967)
```

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

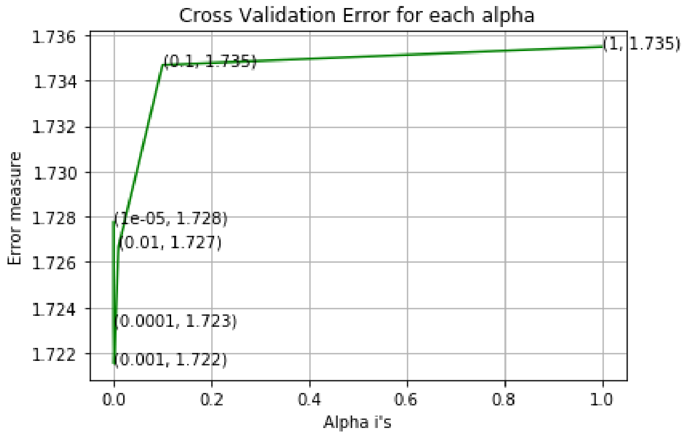
Let's build a model just like the earlier!

In [36]:

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

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

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

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


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

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

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

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


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

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

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

```
For values of alpha =  1e-05 The log loss is: 1.7277622467098064
For values of alpha =  0.0001 The log loss is: 1.7231890943653245
For values of alpha =  0.001 The log loss is: 1.7215094642100297
For values of alpha =  0.01 The log loss is: 1.7266719440969969
For values of alpha =  0.1 The log loss is: 1.7346815214007536
For values of alpha =  1 The log loss is: 1.735483488250195
```



```
For values of best alpha =  0.001 The train log loss is: 1.0778974453913979
For values of best alpha =  0.001 The cross validation log loss is: 1.7215094642100297
For values of best alpha =  0.001 The test log loss is: 1.6869429071351947
```

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 [37]:

```
print("Q12. How many data points are covered by total ", unique_variations.shape[0], " genes in te
st and cross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/test_df.
shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.s
hape[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 77 out of 665 : 11.578947368421053
2. In cross validation data 54 out of  532 : 10.150375939849624
```

3.2.3 Univariate Analysis on Text Feature

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

3. Is the text feature stable across train, test and CV datasets?

In [38]:

```
# 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 [39]:

```
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 [40]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = CountVectorizer(min_df=3,ngram_range=(1,2),max_features=20000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

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


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

```
Total number of unique words in train data : 20000
```

In [41]:

```
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 [42]:

```
#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 [43]:

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

In [44]:

```
# 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 [45]:

```
#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 [46]:

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

```
Counter({120: 186, 118: 180, 115: 178, 119: 171, 116: 169, 112: 164, 117: 160, 114: 158, 113: 154,
121: 151, 130: 147, 126: 147, 129: 143, 123: 139, 128: 136, 125: 136, 133: 135, 124: 133, 122: 131
, 146: 130, 134: 128, 137: 125, 136: 124, 139: 123, 141: 121, 135: 116, 138: 115, 142: 113, 127: 1
12, 132: 111, 150: 109, 147: 109, 162: 107, 149: 106, 143: 106, 131: 104, 140: 103, 148: 102, 145:
101, 153: 98, 152: 97, 156: 96, 154: 95, 144: 94, 151: 93, 161: 91, 163: 90, 160: 85, 171: 84, 157
: 81, 158: 80, 155: 80, 177: 79, 184: 78, 159: 77, 179: 75, 164: 72, 183: 70, 174: 70, 172: 70, 16
9: 70, 168: 70, 175: 69, 188: 68, 170: 68, 176: 67, 195: 65, 182: 65, 189: 64, 167: 64, 181: 63, 1
65: 63, 186: 62, 111: 62, 210: 61, 193: 61, 180: 61, 178: 61, 166: 61, 198: 59, 192: 59, 230: 57,
197: 57, 194: 57, 231: 56, 185: 56, 207: 55, 187: 54, 173: 54, 201: 52, 205: 51, 203: 51, 199: 51,
204: 50, 190: 50, 218: 49, 200: 49, 191: 49, 206: 48, 214: 47, 243: 46, 216: 46, 237: 45, 228: 45,
223: 45, 222: 45, 220: 45, 202: 45, 240: 44, 212: 44, 229: 43, 219: 43, 196: 43, 224: 42, 221: 42,
248: 41, 213: 41, 209: 41, 215: 40, 211: 40, 208: 39, 279: 38, 271: 38, 238: 38, 226: 38, 217: 38,
254: 37, 234: 37, 256: 36, 251: 36, 247: 36, 244: 36, 233: 36, 227: 36, 272: 35, 259: 35, 246: 35,
245: 35, 242: 35, 225: 35, 276: 34, 307: 33, 269: 33, 265: 33, 241: 33, 292: 32, 287: 32, 252: 32,
239: 32, 235: 32, 336: 31, 261: 31, 250: 31, 316: 30, 264: 30, 236: 30, 304: 29, 294: 29, 275: 29,
268: 29, 258: 29, 325: 28, 323: 28, 288: 28, 232: 28, 299: 27, 286: 27, 267: 27, 260: 27, 249: 27,
320: 26, 300: 26, 289: 26, 280: 26, 278: 26, 274: 26, 348: 25, 327: 25, 326: 25, 306: 25, 296: 25,
285: 25, 270: 25, 257: 25, 253: 25, 356: 24, 354: 24, 311: 24, 302: 24, 291: 24, 367: 23, 364: 23,
331: 23, 328: 23, 322: 23, 318: 23, 298: 23, 284: 23, 283: 23, 282: 23, 266: 23, 263: 23, 262: 23,
338: 22, 332: 22, 314: 22, 308: 22, 303: 22, 295: 22, 290: 22, 460: 21, 357: 21, 349: 21, 339: 21,
321: 21, 319: 21, 313: 21, 255: 21, 417: 20, 411: 20, 378: 20, 335: 20, 324: 20, 317: 20, 315: 20,
301: 20, 281: 20, 277: 20, 392: 19, 376: 19, 363: 19, 333: 19, 297: 19, 432: 18, 399: 18, 393: 18,
380: 18, 369: 18, 368: 18, 351: 18, 342: 18, 330: 18, 329: 18, 305: 18, 519: 17, 420: 17, 414: 17,
381: 17, 365: 17, 334: 17, 309: 17, 293: 17, 425: 16, 416: 16, 405: 16, 403: 16, 395: 16, 384: 16,
381: 17, 365: 17, 334: 17, 309: 17, 293: 17, 425: 16, 416: 16, 405: 16, 403: 16, 393: 16, 384: 16,
379: 16, 353: 16, 346: 16, 337: 16, 310: 16, 492: 15, 474: 15, 440: 15, 438: 15, 421: 15, 412: 15,
401: 15, 391: 15, 366: 15, 359: 15, 582: 14, 508: 14, 469: 14, 437: 14, 413: 14, 408: 14, 402: 14,
396: 14, 371: 14, 362: 14, 358: 14, 343: 14, 340: 14, 312: 14, 273: 14, 462: 13, 449: 13, 446: 13,
422: 13, 406: 13, 404: 13, 400: 13, 360: 13, 350: 13, 344: 13, 721: 12, 521: 12, 500: 12, 499: 12,
494: 12, 491: 12, 490: 12, 489: 12, 473: 12, 470: 12, 455: 12, 452: 12, 451: 12, 442: 12, 430: 12,
423: 12, 385: 12, 382: 12, 361: 12, 620: 11, 583: 11, 566: 11, 544: 11, 531: 11, 520: 11, 507: 11,
502: 11, 501: 11, 483: 11, 471: 11, 464: 11, 459: 11, 450: 11, 443: 11, 436: 11, 434: 11, 428: 11,
419: 11, 415: 11, 389: 11, 355: 11, 341: 11, 1047: 10, 680: 10, 672: 10, 631: 10, 617: 10, 593: 10
, 591: 10, 585: 10, 580: 10, 569: 10, 529: 10, 497: 10, 484: 10, 478: 10, 477: 10, 465: 10, 461: 1
0, 453: 10, 448: 10, 447: 10, 445: 10, 444: 10, 441: 10, 439: 10, 418: 10, 388: 10, 377: 10, 374:
10, 352: 10, 345: 10, 819: 9, 776: 9, 744: 9, 704: 9, 681: 9, 639: 9, 609: 9, 606: 9, 592: 9, 589:
9, 562: 9, 556: 9, 547: 9, 534: 9, 530: 9, 517: 9, 516: 9, 512: 9, 504: 9, 503: 9, 498: 9, 493: 9,
487: 9, 481: 9, 475: 9, 456: 9, 427: 9, 410: 9, 398: 9, 387: 9, 375: 9, 372: 9, 370: 9, 1026: 8,
873: 8, 845: 8, 769: 8, 733: 8, 718: 8, 703: 8, 674: 8, 665: 8, 652: 8, 621: 8, 607: 8, 597: 8,
594: 8, 587: 8, 579: 8, 575: 8, 574: 8, 551: 8, 550: 8, 548: 8, 542: 8, 540: 8, 538: 8, 518: 8,
511: 8, 506: 8, 488: 8, 458: 8, 454: 8, 435: 8, 431: 8, 409: 8, 397: 8, 390: 8, 383: 8, 347: 8,
1146: 7, 927: 7, 884: 7, 882: 7, 851: 7, 847: 7, 807: 7, 801: 7, 760: 7, 757: 7, 741: 7, 705: 7,
696: 7, 684: 7, 676: 7, 655: 7, 653: 7, 648: 7, 637: 7, 633: 7, 614: 7, 612: 7, 608: 7, 605: 7,
601: 7, 554: 7, 553: 7, 539: 7, 535: 7, 527: 7, 514: 7, 510: 7, 505: 7, 486: 7, 480: 7, 479: 7,
472: 7, 463: 7, 394: 7, 373: 7, 1588: 6, 1540: 6, 1231: 6, 1218: 6, 1160: 6, 1140: 6, 1072: 6, 1033
: 6, 1025: 6, 984: 6, 977: 6, 967: 6, 956: 6, 939: 6, 906: 6, 871: 6, 864: 6, 863: 6, 836: 6, 835:
6, 827: 6, 809: 6, 803: 6, 792: 6, 783: 6, 781: 6, 777: 6, 770: 6, 765: 6, 763: 6, 755: 6, 743: 6,
734: 6, 710: 6, 692: 6, 689: 6, 671: 6, 670: 6, 661: 6, 654: 6, 646: 6, 645: 6, 632: 6, 629: 6,
626: 6, 624: 6, 622: 6, 613: 6, 610: 6, 602: 6, 599: 6, 598: 6, 596: 6, 586: 6, 577: 6, 572: 6,
570: 6, 564: 6, 563: 6, 561: 6, 557: 6, 555: 6, 536: 6, 532: 6, 528: 6, 526: 6, 524: 6, 523: 6,
522: 6, 515: 6, 513: 6, 496: 6, 482: 6, 468: 6, 457: 6, 433: 6, 429: 6, 407: 6, 386: 6, 1369: 5,
1329: 5, 1293: 5, 1274: 5, 1260: 5, 1243: 5, 1213: 5, 1179: 5, 1130: 5, 1091: 5, 1083: 5, 1066: 5,
1017: 5, 1003: 5, 989: 5, 983: 5, 974: 5, 968: 5, 962: 5, 938: 5, 933: 5, 915: 5, 900: 5, 898: 5,
888: 5, 860: 5, 788: 5, 786: 5, 785: 5, 774: 5, 773: 5, 771: 5, 761: 5, 759: 5, 747: 5, 746: 5,
737: 5, 732: 5, 719: 5, 715: 5, 706: 5, 700: 5, 698: 5, 695: 5, 691: 5, 679: 5, 675: 5, 673: 5,
669: 5, 664: 5, 663: 5, 656: 5, 636: 5, 618: 5, 615: 5, 604: 5, 603: 5, 600: 5, 588: 5, 584: 5,
559: 5, 558: 5, 541: 5, 533: 5, 509: 5, 495: 5, 466: 5, 426: 5, 2657: 4, 2417: 4, 1951: 4, 1645: 4,
1517: 4, 1499: 4, 1449: 4, 1367: 4, 1356: 4, 1318: 4, 1314: 4, 1302: 4, 1261: 4, 1226: 4, 1216: 4,
1211: 4, 1196: 4, 1163: 4, 1158: 4, 1156: 4, 1149: 4, 1141: 4, 1098: 4, 1097: 4, 1094: 4, 1089: 4,
1082: 4, 1076: 4, 1048: 4, 1014: 4, 1013: 4, 1002: 4, 1001: 4, 988: 4, 973: 4, 971: 4, 964: 4, 954:
4, 953: 4, 951: 4, 948: 4, 940: 4, 937: 4, 921: 4, 920: 4, 886: 4, 885: 4, 881: 4, 837: 4, 832: 4,
831: 4, 829: 4, 828: 4, 824: 4, 817: 4, 816: 4, 808: 4, 805: 4, 798: 4, 797: 4, 790: 4, 782: 4,
780: 4, 779: 4, 772: 4, 762: 4, 756: 4, 752: 4, 751: 4, 750: 4, 748: 4, 738: 4, 736: 4, 724: 4,
720: 4, 714: 4, 707: 4, 697: 4, 694: 4, 693: 4, 690: 4, 687: 4, 686: 4, 685: 4, 678: 4, 677: 4,
668: 4, 667: 4, 658: 4, 651: 4, 643: 4, 642: 4, 635: 4, 619: 4, 611: 4, 595: 4, 590: 4, 581: 4,
578: 4, 560: 4, 549: 4, 546: 4, 545: 4, 543: 4, 467: 4, 424: 4, 5462: 3, 3748: 3, 3736: 3, 3438: 3,
3157: 3, 3132: 3, 2977: 3, 2943: 3, 2738: 3, 2468: 3, 2373: 3, 2366: 3, 2288: 3, 2254: 3, 2233: 3,
2170: 3, 2167: 3, 2166: 3, 2155: 3, 2150: 3, 2112: 3, 2106: 3, 2099: 3, 2065: 3, 2019: 3, 1963: 3,
1956: 3, 1954: 3, 1932: 3, 1876: 3, 1866: 3, 1862: 3, 1817: 3, 1786: 3, 1765: 3, 1746: 3, 1734: 3,
1728: 3, 1716: 3, 1689: 3, 1636: 3, 1613: 3, 1606: 3, 1592: 3, 1589: 3, 1565: 3, 1556: 3, 1555: 3,
1545: 3, 1538: 3, 1534: 3, 1519: 3, 1515: 3, 1514: 3, 1498: 3, 1488: 3, 1475: 3, 1464: 3, 1452: 3,
1444: 3, 1439: 3, 1437: 3, 1399: 3, 1389: 3, 1380: 3, 1374: 3, 1372: 3, 1366: 3, 1360: 3, 1349: 3,
1343: 3, 1337: 3, 1336: 3, 1330: 3, 1320: 3, 1317: 3, 1308: 3, 1305: 3, 1299: 3, 1285: 3, 1279: 3,
1277: 3, 1276: 3, 1271: 3, 1265: 3, 1262: 3, 1239: 3, 1228: 3, 1219: 3, 1215: 3, 1198: 3, 1194: 3,
1183: 3, 1177: 3, 1162: 3, 1143: 3, 1133: 3, 1123: 3, 1116: 3, 1102: 3, 1093: 3, 1090: 3, 1088: 3,
1085: 3, 1074: 3, 1060: 3, 1058: 3, 1057: 3, 1056: 3, 1055: 3, 1054: 3, 1052: 3, 1051: 3, 1037: 3,
1035: 3, 1032: 3, 1030: 3, 1022: 3, 1012: 3, 1007: 3, 1004: 3, 993: 3, 991: 3, 986: 3, 980: 3, 978:
3, 975: 3, 972: 3, 966: 3, 965: 3, 963: 3, 960: 3, 947: 3, 946: 3, 945: 3, 944: 3, 943: 3, 942: 3,
941: 3, 932: 3, 928: 3, 924: 3, 914: 3, 910: 3, 909: 3, 907: 3, 905: 3, 902: 3, 893: 3, 892: 3,
890: 3, 889: 3, 880: 3, 878: 3, 877: 3, 875: 3, 867: 3, 866: 3, 850: 3, 849: 3, 848: 3, 844: 3,
841: 3, 840: 3, 839: 3, 838: 3, 821: 3, 814: 3, 810: 3, 806: 3, 804: 3, 802: 3, 796: 3, 794: 3,
793: 3, 791: 3, 789: 3, 787: 3, 768: 3, 767: 3, 758: 3, 753: 3, 745: 3, 740: 3, 735: 3, 727: 3,
726: 3, 725: 3, 723: 3, 722: 3, 716: 3, 711: 3, 709: 3, 702: 3, 701: 3, 688: 3, 683: 3, 662: 3,
660: 3, 659: 3, 650: 3, 649: 3, 647: 3, 644: 3, 641: 3, 630: 3, 623: 3, 616: 3, 576: 3, 573: 3,
568: 3, 537: 3, 476: 3, 20284: 2, 9346: 2, 9048: 2, 8027: 2, 7693: 2, 7260: 2, 6681: 2, 5474: 2, 51
31: 2, 4997: 2, 4863: 2, 4663: 2, 4522: 2, 4459: 2, 4431: 2, 4367: 2, 4358: 2, 4346: 2, 4131: 2, 40
54: 2, 4010: 2, 3884: 2, 3831: 2, 3785: 2, 3615: 2, 3562: 2, 3557: 2, 3542: 2, 3538: 2, 3513: 2, 34
94: 2, 3444: 2, 3434: 2, 3395: 2, 3357: 2, 3350: 2, 3335: 2, 3332: 2, 3301: 2, 3296: 2, 3267: 2, 32
64: 2, 3247: 2, 3228: 2, 3220: 2, 3188: 2, 3167: 2, 3098: 2, 3059: 2, 3026: 2, 2992: 2, 2980: 2, 29
41: 2, 2895: 2, 2878: 2, 2841: 2, 2835: 2, 2824: 2, 2760: 2, 2722: 2, 2715: 2, 2662: 2, 2660: 2, 26
58: 2, 2643: 2, 2612: 2, 2608: 2, 2601: 2, 2598: 2, 2596: 2, 2585: 2, 2581: 2, 2570: 2, 2565: 2, 25
55: 2, 2552: 2, 2501: 2, 2491: 2, 2469: 2, 2463: 2, 2443: 2, 2440: 2, 2435: 2, 2423: 2, 2408: 2, 24
04: 2, 2321: 2, 2313: 2, 2305: 2, 2295: 2, 2281: 2, 2277: 2, 2264: 2, 2258: 2, 2256: 2, 2255: 2, 22
49: 2, 2225: 2, 2221: 2, 2220: 2, 2215: 2, 2208: 2, 2178: 2, 2168: 2, 2158: 2, 2153: 2, 2138: 2, 21
35: 2, 2133: 2, 2119: 2, 2105: 2, 2091: 2, 2086: 2, 2081: 2, 2076: 2, 2072: 2, 2057: 2, 2036: 2, 20
21: 2, 2018: 2, 2016: 2, 2014: 2, 2005: 2, 2000: 2, 1998: 2, 1993: 2, 1986: 2, 1981: 2, 1969: 2, 19
68: 2, 1964: 2, 1958: 2, 1921: 2, 1914: 2, 1911: 2, 1909: 2, 1901: 2, 1897: 2, 1895: 2, 1881: 2, 18
73: 2, 1863: 2, 1858: 2, 1857: 2, 1838: 2, 1828: 2, 1827: 2, 1820: 2, 1816: 2, 1814: 2, 1807: 2, 18
06: 2, 1803: 2, 1795: 2, 1787: 2, 1778: 2, 1770: 2, 1768: 2, 1761: 2, 1749: 2, 1735: 2, 1731: 2, 17
30: 2, 1723: 2, 1718: 2, 1717: 2, 1715: 2, 1707: 2, 1699: 2, 1693: 2, 1691: 2, 1690: 2, 1687: 2, 16
85: 2, 1684: 2, 1677: 2, 1674: 2, 1672: 2, 1662: 2, 1661: 2, 1659: 2, 1657: 2, 1655: 2, 1646: 2, 16
44: 2, 1631: 2, 1615: 2, 1609: 2, 1604: 2, 1602: 2, 1601: 2, 1597: 2, 1595: 2, 1590: 2, 1586: 2, 15
```

```
44: 2, 1631: 2, 1615: 2, 1609: 2, 1604: 2, 1602: 2, 1601: 2, 1597: 2, 1595: 2, 1590: 2, 1586: 2, 15
82: 2, 1579: 2, 1566: 2, 1561: 2, 1560: 2, 1557: 2, 1551: 2, 1547: 2, 1542: 2, 1529: 2, 1520: 2, 15
08: 2, 1507: 2, 1503: 2, 1496: 2, 1483: 2, 1480: 2, 1479: 2, 1478: 2, 1470: 2, 1460: 2, 1458: 2, 14
55: 2, 1453: 2, 1450: 2, 1447: 2, 1440: 2, 1432: 2, 1426: 2, 1422: 2, 1416: 2, 1415: 2, 1414: 2, 14
13: 2, 1409: 2, 1400: 2, 1391: 2, 1386: 2, 1382: 2, 1373: 2, 1359: 2, 1351: 2, 1350: 2, 1346: 2, 13
38: 2, 1335: 2, 1331: 2, 1322: 2, 1319: 2, 1315: 2, 1312: 2, 1311: 2, 1306: 2, 1301: 2, 1294: 2, 12
92: 2, 1290: 2, 1284: 2, 1283: 2, 1280: 2, 1273: 2, 1270: 2, 1268: 2, 1258: 2, 1256: 2, 1255: 2, 12
53: 2, 1248: 2, 1247: 2, 1245: 2, 1244: 2, 1240: 2, 1237: 2, 1236: 2, 1233: 2, 1229: 2, 1222: 2, 12
17: 2, 1208: 2, 1204: 2, 1195: 2, 1193: 2, 1189: 2, 1188: 2, 1182: 2, 1181: 2, 1180: 2, 1175: 2, 11
74: 2, 1170: 2, 1157: 2, 1148: 2, 1147: 2, 1136: 2, 1131: 2, 1129: 2, 1127: 2, 1125: 2, 1124: 2, 11
18: 2, 1115: 2, 1114: 2, 1106: 2, 1105: 2, 1104: 2, 1103: 2, 1100: 2, 1096: 2, 1095: 2, 1092: 2, 10
73: 2, 1071: 2, 1070: 2, 1065: 2, 1062: 2, 1059: 2, 1053: 2, 1050: 2, 1049: 2, 1046: 2, 1040: 2, 10
39: 2, 1036: 2, 1034: 2, 1028: 2, 1023: 2, 1019: 2, 1016: 2, 1015: 2, 1011: 2, 1010: 2, 1005: 2, 10
00: 2, 998: 2, 996: 2, 994: 2, 982: 2, 981: 2, 961: 2, 955: 2, 952: 2, 950: 2, 949: 2, 936: 2,
929: 2, 925: 2, 918: 2, 917: 2, 911: 2, 908: 2, 904: 2, 897: 2, 896: 2, 894: 2, 887: 2, 883: 2,
876: 2, 874: 2, 869: 2, 865: 2, 862: 2, 861: 2, 859: 2, 857: 2, 856: 2, 855: 2, 853: 2, 843: 2,
834: 2, 833: 2, 830: 2, 826: 2, 823: 2, 822: 2, 815: 2, 813: 2, 812: 2, 811: 2, 800: 2, 799: 2,
778: 2, 766: 2, 754: 2, 749: 2, 739: 2, 731: 2, 728: 2, 717: 2, 712: 2, 708: 2, 666: 2, 657: 2,
638: 2, 634: 2, 628: 2, 627: 2, 625: 2, 571: 2, 567: 2, 565: 2, 552: 2, 525: 2, 485: 2, 153331: 1,
118445: 1, 80660: 1, 68707: 1, 68479: 1, 67949: 1, 67680: 1, 66909: 1, 64525: 1, 63572: 1, 57332:
1, 53912: 1, 48917: 1, 48688: 1, 45856: 1, 45609: 1, 44739: 1, 43637: 1, 42507: 1, 42216: 1, 41778
: 1, 40810: 1, 40146: 1, 39808: 1, 39341: 1, 38890: 1, 38025: 1, 36637: 1, 36457: 1, 36027: 1, 358
83: 1, 35531: 1, 33934: 1, 33820: 1, 33664: 1, 33202: 1, 31998: 1, 31872: 1, 29552: 1, 29380: 1, 2
8234: 1, 26466: 1, 26395: 1, 26259: 1, 25901: 1, 25453: 1, 24735: 1, 24653: 1, 24508: 1, 24400: 1,
24085: 1, 23967: 1, 23881: 1, 23764: 1, 23668: 1, 23036: 1, 22658: 1, 21853: 1, 21765: 1, 21381: 1
, 21291: 1, 21227: 1, 21156: 1, 20392: 1, 20085: 1, 19876: 1, 19732: 1, 19568: 1, 19471: 1, 19193:
1, 19100: 1, 18997: 1, 18772: 1, 18742: 1, 18520: 1, 18427: 1, 18411: 1, 18356: 1, 18188: 1, 18186
: 1, 18184: 1, 18144: 1, 17949: 1, 17905: 1, 17852: 1, 17734: 1, 17688: 1, 17618: 1, 17526: 1, 175
08: 1, 17335: 1, 17318: 1, 17309: 1, 17064: 1, 17051: 1, 16973: 1, 16743: 1, 16519: 1, 16232: 1, 1
6083: 1, 16060: 1, 15960: 1, 15905: 1, 15895: 1, 15775: 1, 15641: 1, 15574: 1, 15418: 1, 15392: 1,
15288: 1, 15242: 1, 15225: 1, 14952: 1, 14899: 1, 14776: 1, 14595: 1, 14586: 1, 14579: 1, 14535: 1
, 14524: 1, 14508: 1, 14470: 1, 14096: 1, 14015: 1, 13931: 1, 13909: 1, 13834: 1, 13726: 1, 13547:
1, 13448: 1, 13361: 1, 13335: 1, 13329: 1, 13192: 1, 13174: 1, 13104: 1, 13014: 1, 12891: 1, 12819
: 1, 12809: 1, 12787: 1, 12775: 1, 12660: 1, 12653: 1, 12649: 1, 12538: 1, 12535: 1, 12527: 1, 124
17: 1, 12410: 1, 12381: 1, 12337: 1, 12321: 1, 12290: 1, 12283: 1, 12281: 1, 12251: 1, 12190: 1, 1
2188: 1, 12177: 1, 12087: 1, 12083: 1, 12078: 1, 12036: 1, 11999: 1, 11961: 1, 11940: 1, 11934: 1,
11916: 1, 11913: 1, 11911: 1, 11611: 1, 11591: 1, 11505: 1, 11503: 1, 11492: 1, 11453: 1, 11447: 1
, 11309: 1, 11197: 1, 11191: 1, 11170: 1, 11080: 1, 10955: 1, 10943: 1, 10887: 1, 10858: 1, 10843:
1, 10836: 1, 10784: 1, 10640: 1, 10593: 1, 10549: 1, 10499: 1, 10485: 1, 10474: 1, 10446: 1, 10368
: 1, 10366: 1, 10346: 1, 10314: 1, 10300: 1, 10294: 1, 10153: 1, 10047: 1, 10029: 1, 10021: 1, 100
09: 1, 9986: 1, 9967: 1, 9920: 1, 9906: 1, 9896: 1, 9872: 1, 9861: 1, 9734: 1, 9714: 1, 9705: 1, 96
64: 1, 9628: 1, 9431: 1, 9430: 1, 9408: 1, 9398: 1, 9313: 1, 9275: 1, 9264: 1, 9213: 1, 9208: 1, 92
03: 1, 9201: 1, 9171: 1, 9137: 1, 9127: 1, 9101: 1, 9070: 1, 9052: 1, 9031: 1, 9024: 1, 8996: 1, 89
23: 1, 8892: 1, 8792: 1, 8760: 1, 8755: 1, 8706: 1, 8663: 1, 8655: 1, 8654: 1, 8621: 1, 8552: 1, 84
68: 1, 8417: 1, 8416: 1, 8403: 1, 8398: 1, 8391: 1, 8368: 1, 8325: 1, 8323: 1, 8297: 1, 8259: 1, 82
34: 1, 8181: 1, 8159: 1, 8140: 1, 8101: 1, 8089: 1, 8068: 1, 8066: 1, 8041: 1, 8036: 1, 8021: 1, 80
07: 1, 7988: 1, 7940: 1, 7930: 1, 7928: 1, 7920: 1, 7886: 1, 7878: 1, 7876: 1, 7874: 1, 7868: 1, 78
34: 1, 7832: 1, 7815: 1, 7808: 1, 7803: 1, 7798: 1, 7742: 1, 7726: 1, 7677: 1, 7644: 1, 7643: 1, 75
92: 1, 7585: 1, 7555: 1, 7549: 1, 7545: 1, 7516: 1, 7509: 1, 7496: 1, 7470: 1, 7463: 1, 7437: 1, 74
09: 1, 7392: 1, 7334: 1, 7314: 1, 7301: 1, 7253: 1, 7213: 1, 7199: 1, 7189: 1, 7171: 1, 7169: 1, 71
61: 1, 7152: 1, 7134: 1, 7133: 1, 7117: 1, 7114: 1, 7072: 1, 7066: 1, 7058: 1, 7041: 1, 7028: 1, 70
03: 1, 6986: 1, 6976: 1, 6970: 1, 6963: 1, 6954: 1, 6937: 1, 6931: 1, 6906: 1, 6892: 1, 6891: 1, 68
49: 1, 6830: 1, 6807: 1, 6802: 1, 6798: 1, 6734: 1, 6720: 1, 6716: 1, 6712: 1, 6686: 1, 6680: 1, 66
76: 1, 6671: 1, 6656: 1, 6646: 1, 6629: 1, 6615: 1, 6613: 1, 6590: 1, 6574: 1, 6572: 1, 6536: 1, 65
16: 1, 6500: 1, 6469: 1, 6457: 1, 6440: 1, 6439: 1, 6434: 1, 6429: 1, 6426: 1, 6405: 1, 6400: 1, 63
96: 1, 6387: 1, 6355: 1, 6346: 1, 6341: 1, 6322: 1, 6318: 1, 6308: 1, 6288: 1, 6274: 1, 6263: 1, 62
56: 1, 6255: 1, 6253: 1, 6233: 1, 6220: 1, 6215: 1, 6206: 1, 6204: 1, 6187: 1, 6157: 1, 6140: 1, 61
25: 1, 6118: 1, 6115: 1, 6113: 1, 6106: 1, 6104: 1, 6102: 1, 6094: 1, 6086: 1, 6077: 1, 6076: 1, 60
67: 1, 6060: 1, 6049: 1, 6044: 1, 6017: 1, 6005: 1, 5992: 1, 5986: 1, 5977: 1, 5958: 1, 5952: 1, 59
23: 1, 5917: 1, 5894: 1, 5866: 1, 5863: 1, 5853: 1, 5843: 1, 5842: 1, 5834: 1, 5793: 1, 5789: 1, 57
82: 1, 5733: 1, 5706: 1, 5703: 1, 5694: 1, 5689: 1, 5653: 1, 5648: 1, 5630: 1, 5617: 1, 5602: 1, 55
87: 1, 5580: 1, 5578: 1, 5574: 1, 5559: 1, 5554: 1, 5552: 1, 5529: 1, 5526: 1, 5515: 1, 5495: 1, 54
60: 1, 5429: 1, 5428: 1, 5422: 1, 5419: 1, 5417: 1, 5408: 1, 5406: 1, 5405: 1, 5361: 1, 5320: 1, 52
98: 1, 5264: 1, 5223: 1, 5217: 1, 5214: 1, 5208: 1, 5206: 1, 5200: 1, 5199: 1, 5192: 1, 5164: 1, 51
61: 1, 5153: 1, 5147: 1, 5133: 1, 5121: 1, 5100: 1, 5099: 1, 5094: 1, 5088: 1, 5076: 1, 5068: 1, 50
52: 1, 5047: 1, 5045: 1, 5042: 1, 5033: 1, 5018: 1, 5016: 1, 5015: 1, 5013: 1, 5009: 1, 4985: 1, 49
76: 1, 4972: 1, 4962: 1, 4959: 1, 4953: 1, 4950: 1, 4947: 1, 4945: 1, 4938: 1, 4935: 1, 4921: 1, 48
96: 1, 4893: 1, 4869: 1, 4859: 1, 4850: 1, 4844: 1, 4834: 1, 4819: 1, 4809: 1, 4786: 1, 4785: 1, 47
76: 1, 4774: 1, 4759: 1, 4757: 1, 4732: 1, 4731: 1, 4724: 1, 4723: 1, 4709: 1, 4707: 1, 4691: 1, 46
50: 1, 4642: 1, 4638: 1, 4633: 1, 4624: 1, 4617: 1, 4578: 1, 4577: 1, 4568: 1, 4563: 1, 4557: 1, 45
52: 1, 4548: 1, 4533: 1, 4532: 1, 4529: 1, 4515: 1, 4513: 1, 4510: 1, 4501: 1, 4493: 1, 4489: 1, 44
85: 1, 4474: 1, 4473: 1, 4461: 1, 4456: 1, 4454: 1, 4453: 1, 4446: 1, 4430: 1, 4425: 1, 4414: 1, 44
09: 1, 4405: 1, 4387: 1, 4383: 1, 4376: 1, 4371: 1, 4364: 1, 4362: 1, 4351: 1, 4349: 1, 4341: 1, 43
39: 1, 4330: 1, 4318: 1, 4315: 1, 4314: 1, 4311: 1, 4299: 1, 4296: 1, 4292: 1, 4278: 1, 4276: 1, 42
75: 1, 4266: 1, 4264: 1, 4261: 1, 4251: 1, 4242: 1, 4238: 1, 4236: 1, 4201: 1, 4197: 1, 4196: 1, 41
84: 1, 4176: 1, 4172: 1, 4168: 1, 4150: 1, 4142: 1, 4112: 1, 4106: 1, 4102: 1, 4097: 1, 4092: 1, 40
78: 1, 4075: 1, 4071: 1, 4070: 1, 4066: 1, 4055: 1, 4050: 1, 4049: 1, 4047: 1, 4046: 1, 4043: 1, 40
37: 1, 4030: 1, 4023: 1, 3997: 1, 3994: 1, 3993: 1, 3992: 1, 3976: 1, 3974: 1, 3970: 1, 3968: 1, 39
```

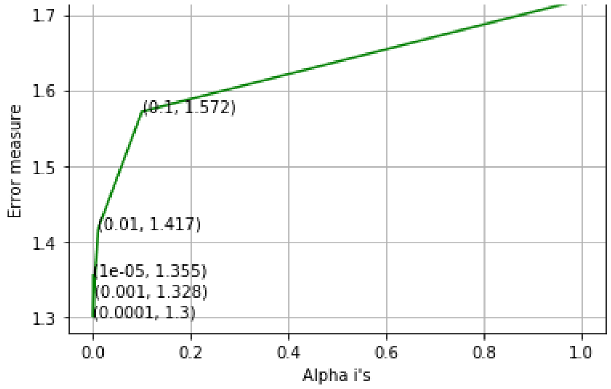
```
37: 1, 4030: 1, 4023: 1, 3997: 1, 3994: 1, 3993: 1, 3992: 1, 3976: 1, 3974: 1, 3970: 1, 3968: 1, 39
65: 1, 3956: 1, 3954: 1, 3952: 1, 3946: 1, 3945: 1, 3923: 1, 3919: 1, 3917: 1, 3912: 1, 3907: 1, 38
99: 1, 3894: 1, 3892: 1, 3881: 1, 3878: 1, 3862: 1, 3849: 1, 3842: 1, 3840: 1, 3837: 1, 3835: 1, 38
20: 1, 3819: 1, 3818: 1, 3809: 1, 3806: 1, 3804: 1, 3801: 1, 3796: 1, 3795: 1, 3793: 1, 3792: 1, 37
84: 1, 3781: 1, 3768: 1, 3764: 1, 3756: 1, 3755: 1, 3751: 1, 3730: 1, 3723: 1, 3722: 1, 3721: 1, 37
13: 1, 3712: 1, 3702: 1, 3694: 1, 3691: 1, 3686: 1, 3683: 1, 3679: 1, 3675: 1, 3670: 1, 3665: 1, 36
63: 1, 3662: 1, 3661: 1, 3660: 1, 3656: 1, 3655: 1, 3651: 1, 3648: 1, 3641: 1, 3632: 1, 3623: 1, 36
20: 1, 3616: 1, 3614: 1, 3612: 1, 3611: 1, 3603: 1, 3600: 1, 3596: 1, 3590: 1, 3581: 1, 3576: 1, 35
73: 1, 3566: 1, 3564: 1, 3559: 1, 3556: 1, 3555: 1, 3553: 1, 3546: 1, 3544: 1, 3528: 1, 3524: 1, 35
23: 1, 3518: 1, 3517: 1, 3514: 1, 3512: 1, 3511: 1, 3507: 1, 3503: 1, 3481: 1, 3471: 1, 3469: 1, 34
68: 1, 3467: 1, 3462: 1, 3458: 1, 3456: 1, 3449: 1, 3448: 1, 3443: 1, 3431: 1, 3416: 1, 3414: 1, 34
09: 1, 3407: 1, 3401: 1, 3397: 1, 3391: 1, 3388: 1, 3383: 1, 3382: 1, 3381: 1, 3373: 1, 3371: 1, 33
70: 1, 3365: 1, 3364: 1, 3363: 1, 3359: 1, 3358: 1, 3344: 1, 3340: 1, 3338: 1, 3336: 1, 3327: 1, 33
26: 1, 3325: 1, 3312: 1, 3308: 1, 3288: 1, 3287: 1, 3283: 1, 3271: 1, 3262: 1, 3257: 1, 3245: 1, 32
43: 1, 3238: 1, 3221: 1, 3218: 1, 3217: 1, 3216: 1, 3215: 1, 3214: 1, 3209: 1, 3199: 1, 3184: 1, 31
78: 1, 3173: 1, 3169: 1, 3163: 1, 3156: 1, 3155: 1, 3154: 1, 3148: 1, 3142: 1, 3139: 1, 3136: 1, 31
34: 1, 3131: 1, 3124: 1, 3120: 1, 3119: 1, 3117: 1, 3106: 1, 3094: 1, 3092: 1, 3085: 1, 3080: 1, 30
79: 1, 3078: 1, 3076: 1, 3073: 1, 3071: 1, 3069: 1, 3067: 1, 3064: 1, 3063: 1, 3058: 1, 3041: 1, 30
40: 1, 3022: 1, 3014: 1, 3010: 1, 3008: 1, 3003: 1, 2994: 1, 2990: 1, 2989: 1, 2987: 1, 2986: 1, 29
85: 1, 2972: 1, 2968: 1, 2964: 1, 2961: 1, 2956: 1, 2952: 1, 2948: 1, 2947: 1, 2939: 1, 2938: 1, 29
37: 1, 2928: 1, 2920: 1, 2909: 1, 2908: 1, 2907: 1, 2906: 1, 2905: 1, 2900: 1, 2894: 1, 2892: 1, 28
90: 1, 2887: 1, 2884: 1, 2882: 1, 2881: 1, 2865: 1, 2863: 1, 2851: 1, 2847: 1, 2846: 1, 2844: 1, 28
42: 1, 2826: 1, 2825: 1, 2822: 1, 2815: 1, 2813: 1, 2808: 1, 2805: 1, 2802: 1, 2800: 1, 2798: 1, 27
93: 1, 2788: 1, 2785: 1, 2782: 1, 2773: 1, 2772: 1, 2758: 1, 2754: 1, 2751: 1, 2748: 1, 2744: 1, 27
43: 1, 2741: 1, 2734: 1, 2732: 1, 2731: 1, 2730: 1, 2724: 1, 2710: 1, 2705: 1, 2697: 1, 2693: 1, 26
91: 1, 2683: 1, 2680: 1, 2676: 1, 2675: 1, 2674: 1, 2671: 1, 2668: 1, 2666: 1, 2659: 1, 2650: 1, 26
46: 1, 2642: 1, 2635: 1, 2631: 1, 2630: 1, 2629: 1, 2624: 1, 2615: 1, 2614: 1, 2611: 1, 2610: 1, 26
07: 1, 2605: 1, 2603: 1, 2602: 1, 2600: 1, 2599: 1, 2590: 1, 2589: 1, 2586: 1, 2584: 1, 2582: 1, 25
75: 1, 2571: 1, 2563: 1, 2562: 1, 2561: 1, 2559: 1, 2557: 1, 2550: 1, 2546: 1, 2543: 1, 2541: 1, 25
39: 1, 2535: 1, 2534: 1, 2533: 1, 2532: 1, 2528: 1, 2525: 1, 2520: 1, 2519: 1, 2511: 1, 2510: 1, 25
04: 1, 2499: 1, 2497: 1, 2482: 1, 2478: 1, 2477: 1, 2476: 1, 2465: 1, 2457: 1, 2452: 1, 2450: 1, 24
47: 1, 2446: 1, 2445: 1, 2441: 1, 2438: 1, 2437: 1, 2436: 1, 2434: 1, 2431: 1, 2430: 1, 2419: 1, 24
16: 1, 2415: 1, 2414: 1, 2399: 1, 2397: 1, 2395: 1, 2391: 1, 2388: 1, 2386: 1, 2384: 1, 2382: 1, 23
79: 1, 2378: 1, 2376: 1, 2368: 1, 2367: 1, 2360: 1, 2359: 1, 2357: 1, 2356: 1, 2349: 1, 2347: 1, 23
45: 1, 2344: 1, 2343: 1, 2338: 1, 2337: 1, 2335: 1, 2330: 1, 2326: 1, 2325: 1, 2322: 1, 2319: 1, 23
18: 1, 2308: 1, 2306: 1, 2303: 1, 2300: 1, 2296: 1, 2284: 1, 2283: 1, 2282: 1, 2276: 1, 2272: 1, 22
69: 1, 2267: 1, 2266: 1, 2265: 1, 2243: 1, 2242: 1, 2241: 1, 2236: 1, 2234: 1, 2231: 1, 2227: 1, 22
26: 1, 2224: 1, 2223: 1, 2222: 1, 2214: 1, 2213: 1, 2212: 1, 2211: 1, 2204: 1, 2195: 1, 2191: 1, 21
90: 1, 2187: 1, 2186: 1, 2183: 1, 2180: 1, 2177: 1, 2176: 1, 2174: 1, 2163: 1, 2162: 1, 2161: 1, 21
54: 1, 2152: 1, 2146: 1, 2144: 1, 2143: 1, 2142: 1, 2140: 1, 2137: 1, 2136: 1, 2130: 1, 2129: 1, 21
27: 1, 2126: 1, 2120: 1, 2116: 1, 2115: 1, 2113: 1, 2109: 1, 2104: 1, 2103: 1, 2102: 1, 2100: 1, 20
98: 1, 2097: 1, 2095: 1, 2093: 1, 2092: 1, 2088: 1, 2084: 1, 2080: 1, 2077: 1, 2074: 1, 2069: 1, 20
67: 1, 2066: 1, 2063: 1, 2062: 1, 2061: 1, 2054: 1, 2053: 1, 2048: 1, 2047: 1, 2043: 1, 2040: 1, 20
34: 1, 2032: 1, 2031: 1, 2026: 1, 2008: 1, 2003: 1, 1994: 1, 1991: 1, 1987: 1, 1985: 1, 1984: 1, 19
83: 1, 1982: 1, 1976: 1, 1974: 1, 1971: 1, 1966: 1, 1965: 1, 1962: 1, 1961: 1, 1959: 1, 1955: 1, 19
50: 1, 1949: 1, 1946: 1, 1945: 1, 1943: 1, 1942: 1, 1939: 1, 1937: 1, 1936: 1, 1935: 1, 1931: 1, 19
27: 1, 1923: 1, 1922: 1, 1919: 1, 1918: 1, 1917: 1, 1916: 1, 1915: 1, 1910: 1, 1907: 1, 1904: 1, 19
00: 1, 1896: 1, 1892: 1, 1891: 1, 1890: 1, 1887: 1, 1885: 1, 1883: 1, 1880: 1, 1878: 1, 1874: 1, 18
72: 1, 1868: 1, 1867: 1, 1860: 1, 1856: 1, 1854: 1, 1853: 1, 1852: 1, 1849: 1, 1847: 1, 1845: 1, 18
44: 1, 1843: 1, 1837: 1, 1833: 1, 1831: 1, 1830: 1, 1826: 1, 1821: 1, 1819: 1, 1815: 1, 1813: 1, 18
12: 1, 1805: 1, 1802: 1, 1799: 1, 1796: 1, 1793: 1, 1790: 1, 1788: 1, 1781: 1, 1777: 1, 1775: 1, 17
73: 1, 1772: 1, 1771: 1, 1769: 1, 1766: 1, 1764: 1, 1763: 1, 1762: 1, 1760: 1, 1759: 1, 1756: 1, 17
54: 1, 1750: 1, 1748: 1, 1744: 1, 1742: 1, 1740: 1, 1737: 1, 1736: 1, 1733: 1, 1732: 1, 1729: 1, 17
27: 1, 1724: 1, 1711: 1, 1709: 1, 1705: 1, 1704: 1, 1703: 1, 1701: 1, 1700: 1, 1698: 1, 1697: 1, 16
96: 1, 1694: 1, 1692: 1, 1688: 1, 1686: 1, 1682: 1, 1681: 1, 1680: 1, 1679: 1, 1678: 1, 1676: 1, 16
71: 1, 1670: 1, 1669: 1, 1666: 1, 1658: 1, 1656: 1, 1654: 1, 1652: 1, 1651: 1, 1647: 1, 1642: 1, 16
40: 1, 1630: 1, 1629: 1, 1628: 1, 1627: 1, 1626: 1, 1625: 1, 1624: 1, 1620: 1, 1618: 1, 1617: 1, 16
16: 1, 1611: 1, 1608: 1, 1605: 1, 1603: 1, 1598: 1, 1594: 1, 1593: 1, 1587: 1, 1585: 1, 1581: 1, 15
80: 1, 1576: 1, 1575: 1, 1574: 1, 1572: 1, 1571: 1, 1568: 1, 1567: 1, 1564: 1, 1562: 1, 1559: 1, 15
58: 1, 1549: 1, 1548: 1, 1543: 1, 1541: 1, 1537: 1, 1536: 1, 1533: 1, 1532: 1, 1530: 1, 1528: 1, 15
27: 1, 1526: 1, 1525: 1, 1524: 1, 1523: 1, 1521: 1, 1516: 1, 1510: 1, 1502: 1, 1501: 1, 1500: 1, 14
97: 1, 1493: 1, 1492: 1, 1489: 1, 1487: 1, 1486: 1, 1484: 1, 1482: 1, 1477: 1, 1473: 1, 1468: 1, 14
65: 1, 1462: 1, 1461: 1, 1459: 1, 1451: 1, 1448: 1, 1446: 1, 1442: 1, 1441: 1, 1436: 1, 1435: 1, 14
33: 1, 1431: 1, 1430: 1, 1428: 1, 1425: 1, 1424: 1, 1423: 1, 1421: 1, 1420: 1, 1419: 1, 1418: 1, 14
17: 1, 1412: 1, 1411: 1, 1410: 1, 1408: 1, 1407: 1, 1406: 1, 1405: 1, 1403: 1, 1402: 1, 1401: 1, 13
96: 1, 1395: 1, 1394: 1, 1392: 1, 1388: 1, 1387: 1, 1381: 1, 1378: 1, 1377: 1, 1376: 1, 1375: 1, 13
70: 1, 1368: 1, 1364: 1, 1363: 1, 1358: 1, 1357: 1, 1355: 1, 1354: 1, 1352: 1, 1347: 1, 1345: 1, 13
44: 1, 1341: 1, 1339: 1, 1334: 1, 1332: 1, 1328: 1, 1326: 1, 1325: 1, 1324: 1, 1323: 1, 1309: 1, 13
07: 1, 1304: 1, 1303: 1, 1297: 1, 1296: 1, 1291: 1, 1286: 1, 1272: 1, 1269: 1, 1267: 1, 1259: 1, 12
57: 1, 1252: 1, 1251: 1, 1249: 1, 1246: 1, 1242: 1, 1241: 1, 1238: 1, 1234: 1, 1232: 1, 1230: 1, 12
27: 1, 1225: 1, 1223: 1, 1221: 1, 1220: 1, 1214: 1, 1210: 1, 1207: 1, 1205: 1, 1203: 1, 1202: 1, 12
00: 1, 1199: 1, 1197: 1, 1191: 1, 1190: 1, 1186: 1, 1185: 1, 1176: 1, 1173: 1, 1172: 1, 1171: 1, 11
68: 1, 1167: 1, 1166: 1, 1161: 1, 1155: 1, 1154: 1, 1152: 1, 1151: 1, 1150: 1, 1145: 1, 1142: 1, 11
39: 1, 1138: 1, 1135: 1, 1134: 1, 1128: 1, 1126: 1, 1121: 1, 1117: 1, 1111: 1, 1109: 1, 1107: 1, 10
86: 1, 1081: 1, 1078: 1, 1077: 1, 1075: 1, 1069: 1, 1068: 1, 1064: 1, 1061: 1, 1045: 1, 1044: 1, 10
43: 1, 1041: 1, 1038: 1, 1024: 1, 1021: 1, 1020: 1, 1008: 1, 997: 1, 990: 1, 987: 1, 985: 1, 970: 1
, 969: 1, 959: 1, 958: 1, 957: 1, 935: 1, 934: 1, 931: 1, 930: 1, 923: 1, 922: 1, 916: 1, 913: 1,
912: 1, 903: 1, 901: 1, 899: 1, 895: 1, 868: 1, 858: 1, 854: 1, 852: 1, 846: 1, 842: 1, 835: 1
```

```
912: 1, 903: 1, 901: 1, 899: 1, 895: 1, 868: 1, 858: 1, 854: 1, 852: 1, 846: 1, 842: 1, 825: 1,
820: 1, 818: 1, 795: 1, 784: 1, 775: 1, 764: 1, 742: 1, 730: 1, 713: 1, 682: 1, 640: 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_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.355477637257163
For values of alpha =  0.0001 The log loss is: 1.2998028946016753
For values of alpha =  0.001 The log loss is: 1.3279066411498839
For values of alpha =  0.01 The log loss is: 1.4166539639992124
For values of alpha =  0.1 The log loss is: 1.5719278827985916
For values of alpha =  1 The log loss is: 1.71987234905709
```



```
For values of best alpha =  0.0001 The train log loss is: 0.8792465542563729
For values of best alpha =  0.0001 The cross validation log loss is: 1.2998028946016753
For values of best alpha =  0.0001 The test log loss is: 1.1945849213919335
```

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")
```

```
30.559 % of word of test data appeared in train data
34.912 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [50]:

```
#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 [51]:

```
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 [52]:

```
# 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_r
o))
        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]]

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

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

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

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).tocsr()
cv_y = np.array(list(cv_df['Class']))


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

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

In [54]:

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

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 22195)
(number of data points * number of features) in test data =  (665, 22195)
(number of data points * number of features) in cross validation data = (532, 22195)
```

In [55]:

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

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

4.1. Base Line Model

4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

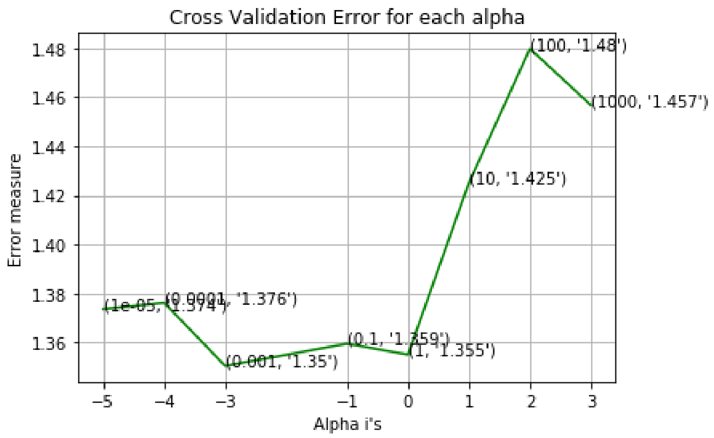
In [56]:

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

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

```
algorithm-1/
# ------------------------


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


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

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


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


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

```
for alpha = 1e-05
Log Loss : 1.373582713678396
for alpha = 0.0001
Log Loss : 1.3761428299486618
for alpha = 0.001
Log Loss : 1.3504060247529628
for alpha = 0.1
Log Loss : 1.3594271602674226
for alpha = 1
Log Loss : 1.3550109619324844
for alpha = 10
Log Loss : 1.4251043902204346
for alpha = 100
Log Loss : 1.47980024289176
```

```
for alpha = 1000
Log Loss : 1.4567928823212968
```



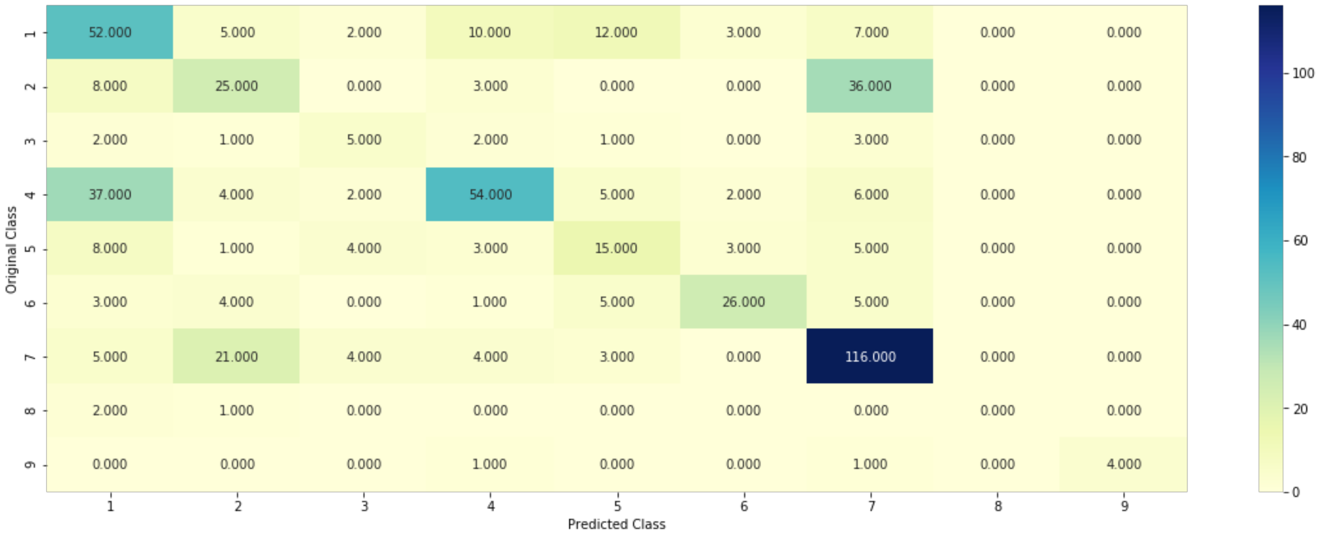
```
For values of best alpha =  0.001 The train log loss is: 0.8680118600981226
For values of best alpha =  0.001 The cross validation log loss is: 1.3504060247529628
For values of best alpha =  0.001 The test log loss is: 1.2705561383472463
```

4.1.1.2. Testing the model with best hyper paramters

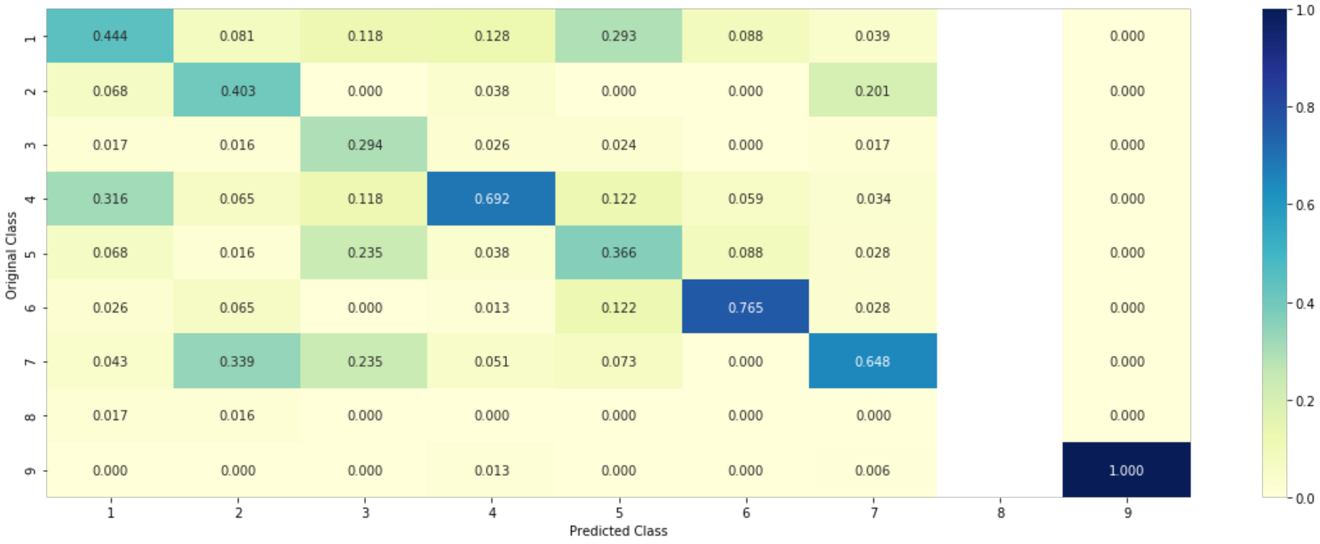
In [57]:

```
# 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-algorithm-1/
# -----------------------


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

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

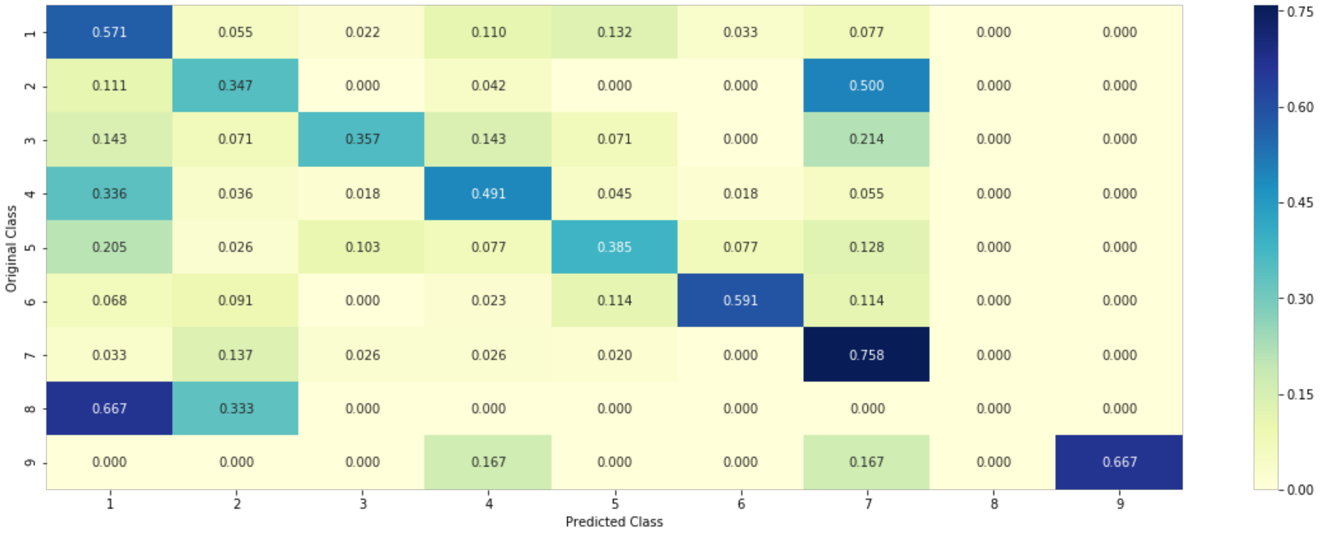
```
Log Loss : 1.3504060247529628
Number of missclassified point : 0.4417293233082707
-------------------- Confusion matrix --------------------
```



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



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



4.1.1.3. Feature Importance, Correctly classified point

In [58]:

```
test_point_index = 1
```

```
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
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 : 4
Predicted Class Probabilities: [[0.0642 0.0612 0.0112 0.4992 0.0341 0.0345 0.2874 0.0049 0.0034]]
Actual Class : 4
--------------------------------------------------
76 Text feature [explained] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [59]:

```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
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: [[0.0718 0.0686 0.0123 0.0999 0.0378 0.0401 0.6604 0.0055 0.0037]]
Actual Class : 5
--------------------------------------------------
17 Text feature [16] present in test data point [True]
32 Text feature [counts] present in test data point [True]
Out of the top  100  features  2 are present in query point
```

4.2. K Nearest Neighbour Classification

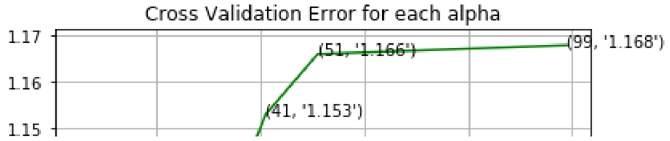
4.2.1. Hyper parameter tuning

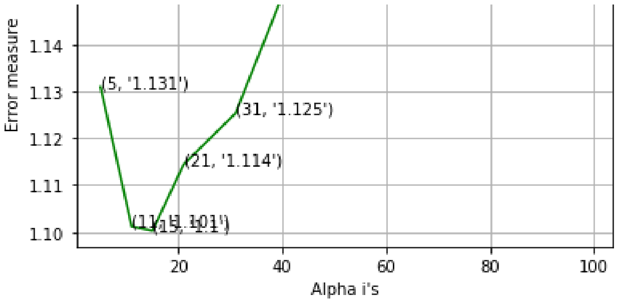
In [101]:

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

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


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

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


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

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


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

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

```
for alpha = 5
Log Loss : 1.1309170126692265
for alpha = 11
Log Loss : 1.1012397762291362
for alpha = 15
Log Loss : 1.1002748803755749
for alpha = 21
Log Loss : 1.1144110925957647
for alpha = 31
Log Loss : 1.1253206995500455
for alpha = 41
Log Loss : 1.1530939773909168
for alpha = 51
Log Loss : 1.1659585643007098
for alpha = 99
Log Loss : 1.1678505034822115
```





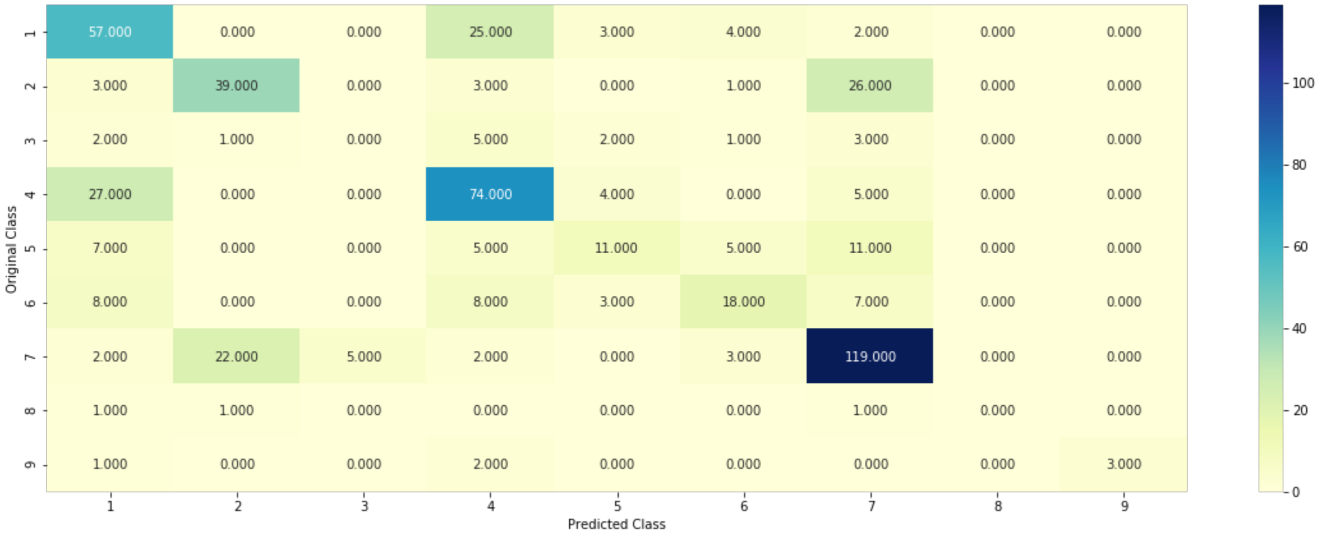
```
For values of best alpha =  15 The train log loss is: 0.7056892871225193
For values of best alpha =  15 The cross validation log loss is: 1.1002748803755749
For values of best alpha =  15 The test log loss is: 1.0911901980302394
```

4.2.2. Testing the model with best hyper paramters

In [102]:

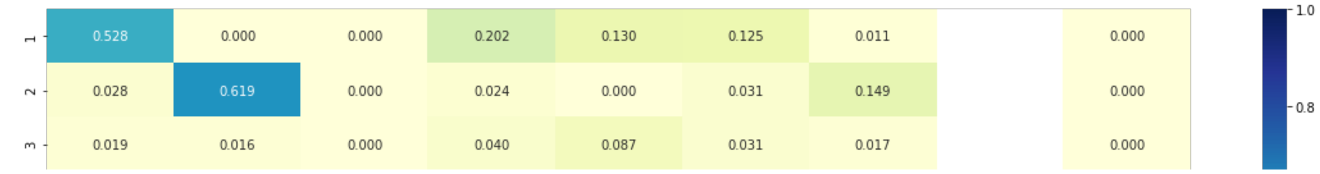
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski',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

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

```
Log loss : 1.1002748803755749
Number of mis-classified points : 0.39661654135338345
-------------------- Confusion matrix --------------------
```

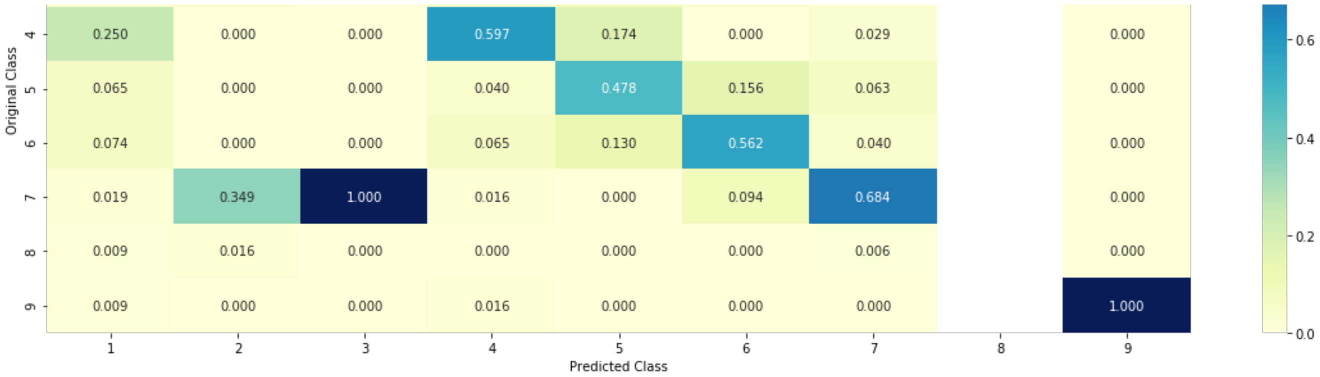


Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	57.000	0.000	0.000	25.000	3.000	4.000	2.000	0.000	0.000
2	3.000	39.000	0.000	3.000	0.000	1.000	26.000	0.000	0.000
3	2.000	1.000	0.000	5.000	2.000	1.000	3.000	0.000	0.000
4	27.000	0.000	0.000	74.000	4.000	0.000	5.000	0.000	0.000
5	7.000	0.000	0.000	5.000	11.000	5.000	11.000	0.000	0.000
6	8.000	0.000	0.000	8.000	3.000	18.000	7.000	0.000	0.000
7	2.000	22.000	5.000	2.000	0.000	3.000	119.000	0.000	0.000
8	1.000	1.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000
9	1.000	0.000	0.000	2.000	0.000	0.000	0.000	0.000	3.000

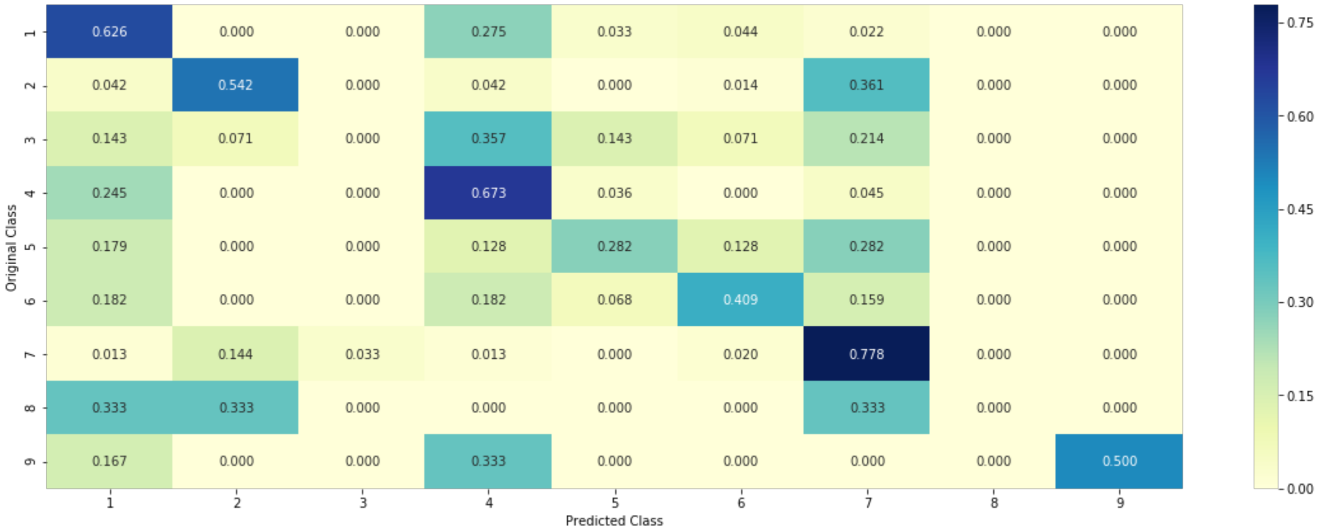
```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



	1	2	3	4	5	6	7	8	9
1	0.528	0.000	0.000	0.202	0.130	0.125	0.011		0.000
2	0.028	0.619	0.000	0.024	0.000	0.031	0.149		0.000
3	0.019	0.016	0.000	0.040	0.087	0.031	0.017		0.000



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



4.2.3.Sample Query point -1

In [103]:

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

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

```
Predicted Class : 6
Actual Class : 7
The  15  nearest neighbours of the test points belongs to classes [7 7 7 6 6 7 6 7 6 7 7 7 7 7 6]
Fequency of nearest points : Counter({7: 10, 6: 5})
```

4.2.4. Sample Query Point-2

In [104]:

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

```
test_point_index = 100

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

```
Predicted Class : 7
Actual Class : 7
the k value for knn is 15 and the nearest neighbours of the test points belongs to classes [2 7 5
7 7 2 7 7 7 2 7 7 7 7 7]
Fequency of nearest points : Counter({7: 11, 2: 3, 5: 1})
```

4.3. Logistic Regression

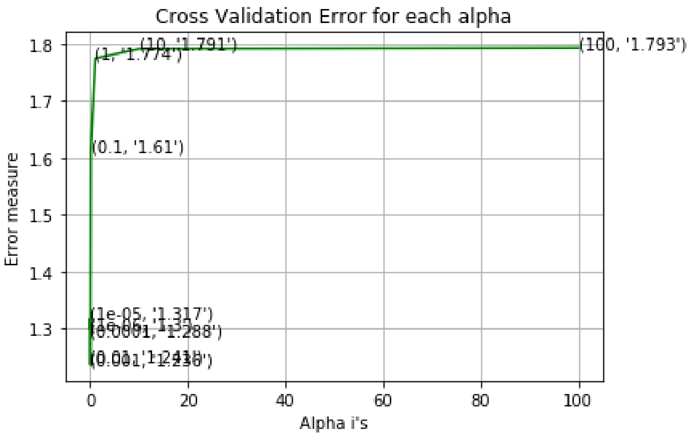
4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [60]:

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

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

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



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

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.2999394776735835
for alpha = 1e-05
Log Loss : 1.3174492130606075
for alpha = 0.0001
Log Loss : 1.2879274815810555
for alpha = 0.001
Log Loss : 1.2359709121465652
for alpha = 0.01
Log Loss : 1.2411232567310144
for alpha = 0.1
Log Loss : 1.610205773117858
for alpha = 1
Log Loss : 1.7741039034361896
for alpha = 10
Log Loss : 1.7913302222418734
for alpha = 100
Log Loss : 1.7929852386303775
```



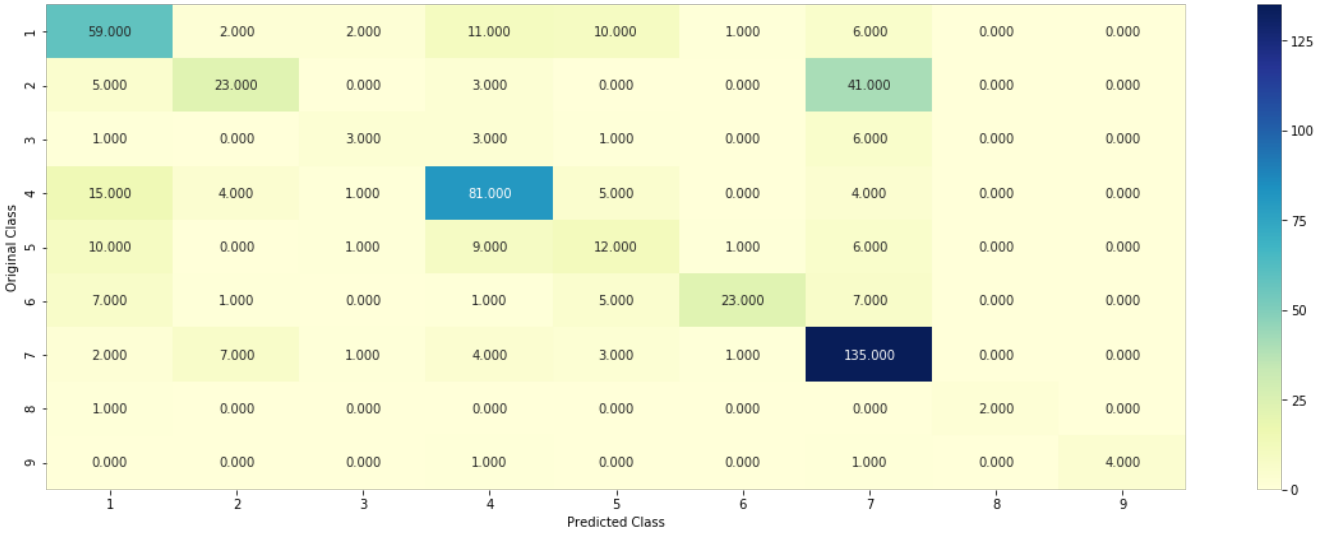
```
For values of best alpha =  0.001 The train log loss is: 0.5659176450527719
For values of best alpha =  0.001 The cross validation log loss is: 1.2359709121465652
For values of best alpha =  0.001 The test log loss is: 1.0810567308032386
```

4.3.1.2. Testing the model with best hyper paramters

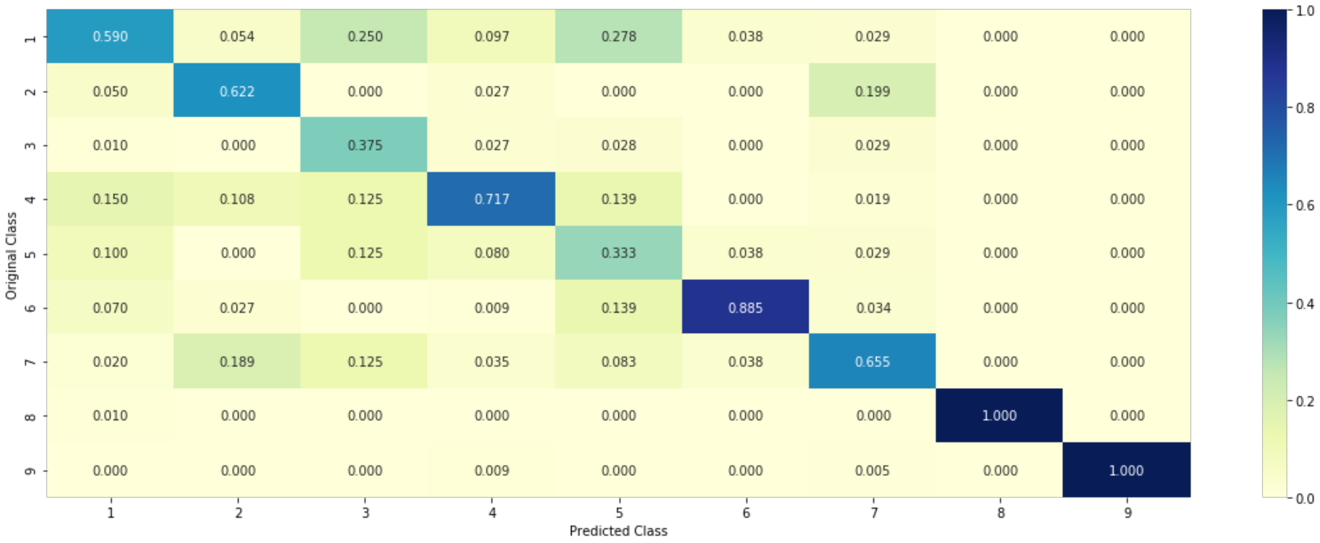
In [61]:

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

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

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

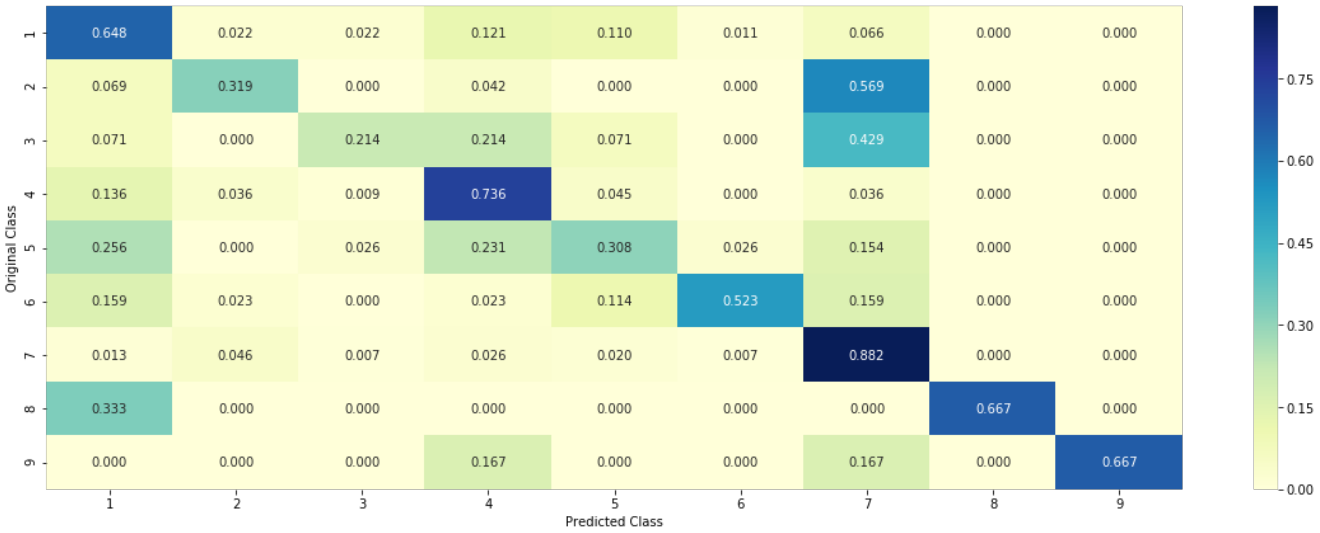
```
Log loss : 1.2359709121465652
Number of mis-classified points : 0.35714285714285715
-------------------- Confusion matrix --------------------
```



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



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



4.3.1.3. Feature Importance

In [62]:

```
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 [63]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 50
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
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 : 4
Predicted Class Probabilities: [[7.400e-03 2.200e-03 1.100e-03 7.948e-01 1.400e-03 6.000e-04 1.880
e-01
  3.400e-03 1.000e-03]]
```

```
 5.400e-05 1.000e-05]]
Actual Class : 4
--------------------------------------------------
26 Text feature [cbf] present in test data point [True]
Out of the top  50  features  1 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [64]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
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: [[0.0111 0.1065 0.0064 0.0237 0.0237 0.4423 0.3773 0.0064 0.0026]]
Actual Class : 5
--------------------------------------------------
90 Text feature [euthanized] present in test data point [True]
160 Text feature [development] present in test data point [True]
162 Text feature [ctrl] present in test data point [True]
239 Text feature [94080] present in test data point [True]
245 Text feature [charged] present in test data point [True]
314 Text feature [enforced] present in test data point [True]
319 Text feature [accompanied] present in test data point [True]
324 Text feature [conformational] present in test data point [True]
466 Text feature [effective] present in test data point [True]
Out of the top  500  features  9 are present in query point
```

4.3.2. Without Class balancing

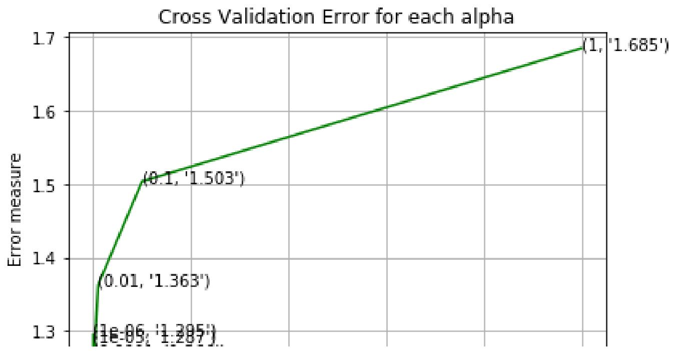
4.3.2.1. Hyper paramter tuning

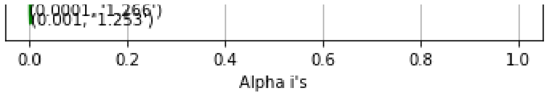
In [65]:

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

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

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



# find more about CalibratedClassifierCV here at http://scikit-
learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
```

```
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.29473632073341
for alpha = 1e-05
Log Loss : 1.2866110688789305
for alpha = 0.0001
Log Loss : 1.266492844219918
for alpha = 0.001
Log Loss : 1.2528324424007187
for alpha = 0.01
Log Loss : 1.362732151742255
for alpha = 0.1
Log Loss : 1.503465014301471
for alpha = 1
Log Loss : 1.684810474363498
```





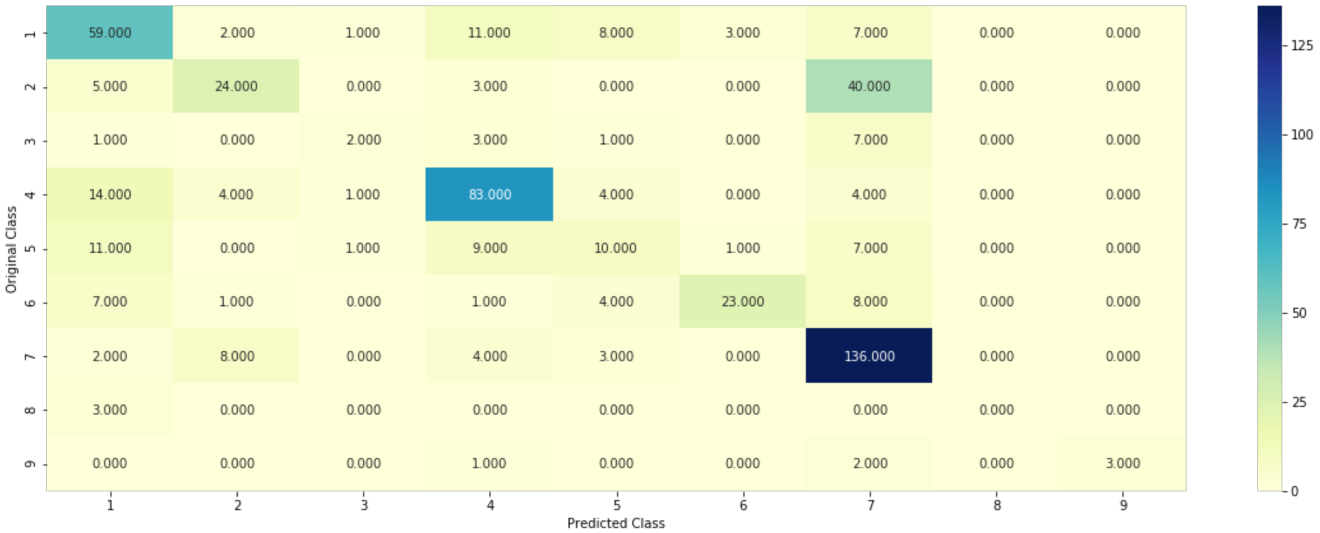
```
For values of best alpha =  0.001 The train log loss is: 0.5661459004698538
For values of best alpha =  0.001 The cross validation log loss is: 1.2528324424007187
For values of best alpha =  0.001 The test log loss is: 1.091072105549384
```

4.3.2.2. Testing model with best hyper parameters

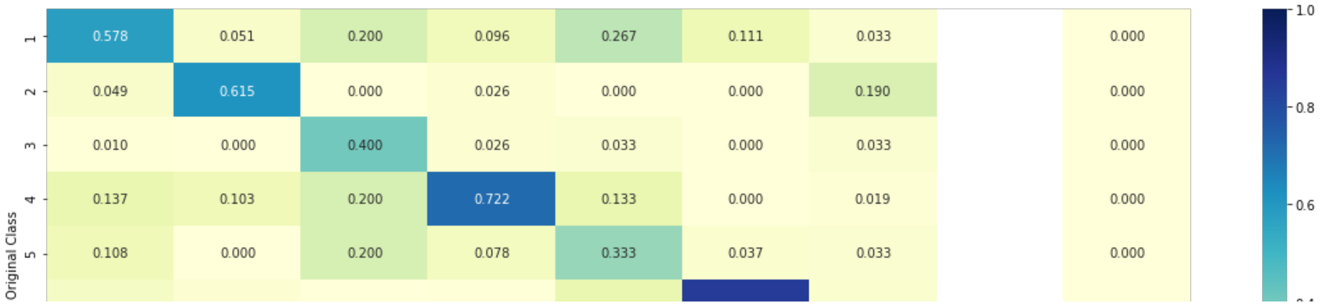
In [66]:

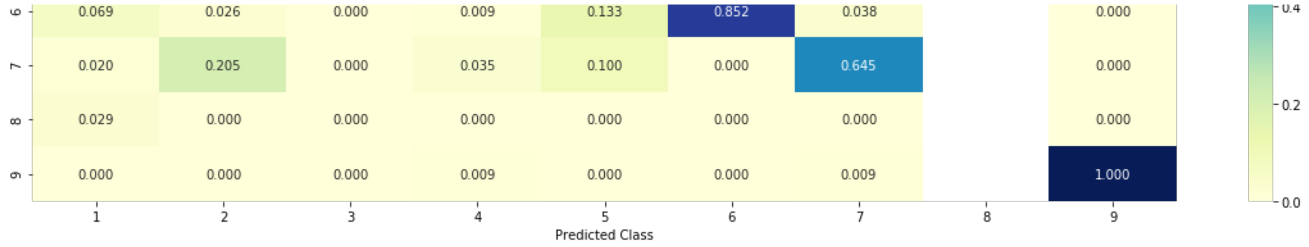
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,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

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

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

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

```
Log loss : 1.2528324424007187
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```

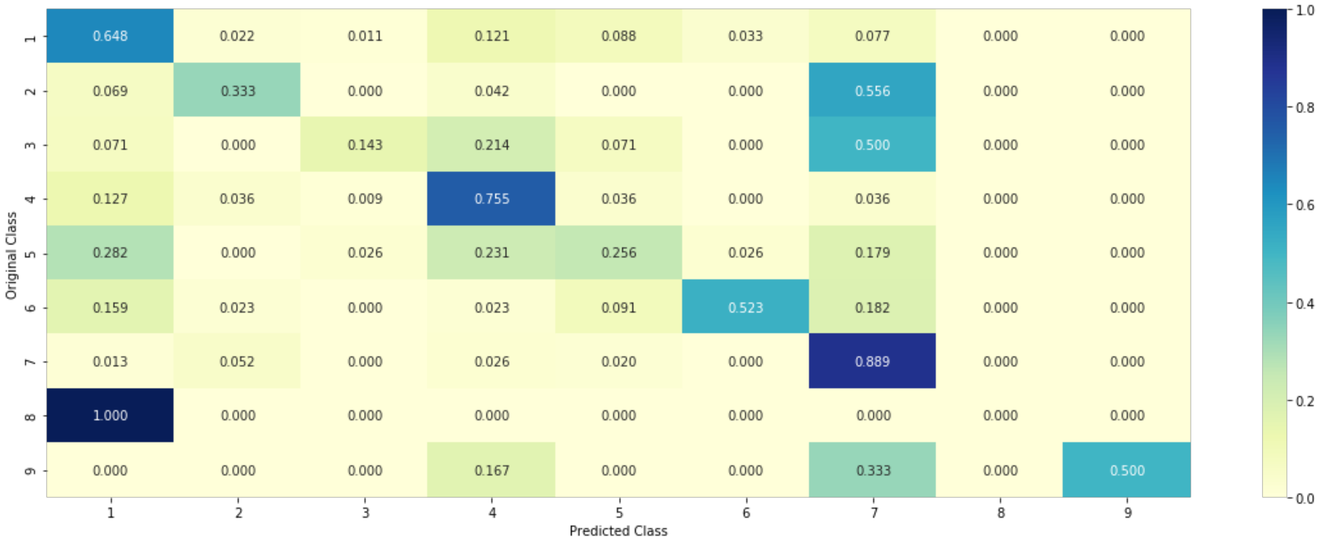


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



4.3.2.3. Feature Importance, Correctly Classified point

In [67]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
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 : 4
Predicted Class Probabilities: [[7.800e-03 1.800e-03 5.000e-04 8.035e-01 1.000e-03 4.000e-04 1.811
e-01
  3.800e-03 0.000e+00]]
Actual Class : 4
--------------------------------------------------
65 Text feature [cbf] present in test data point [True]
222 Text feature [clear] present in test data point [True]
295 Text feature [blotting] present in test data point [True]
304 Text feature [density] present in test data point [True]
319 Text feature [balanced] present in test data point [True]
401 Text feature [exons] present in test data point [True]
454 Text feature [culture] present in test data point [True]
Out of the top  500  features  7 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [68]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
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: [[0.0121 0.1135 0.0058 0.0272 0.0226 0.4187 0.3919 0.0075 0.0008]]
Actual Class : 5
--------------------------------------------------
131 Text feature [euthanized] present in test data point [True]
142 Text feature [development] present in test data point [True]
152 Text feature [ctrl] present in test data point [True]
251 Text feature [charged] present in test data point [True]
263 Text feature [94080] present in test data point [True]
305 Text feature [accompanied] present in test data point [True]
387 Text feature [conformational] present in test data point [True]
390 Text feature [enforced] present in test data point [True]
454 Text feature [effective] present in test data point [True]
487 Text feature [association] present in test data point [True]
Out of the top  500  features  10 are present in query point
```

4.4. Linear Support Vector Machines

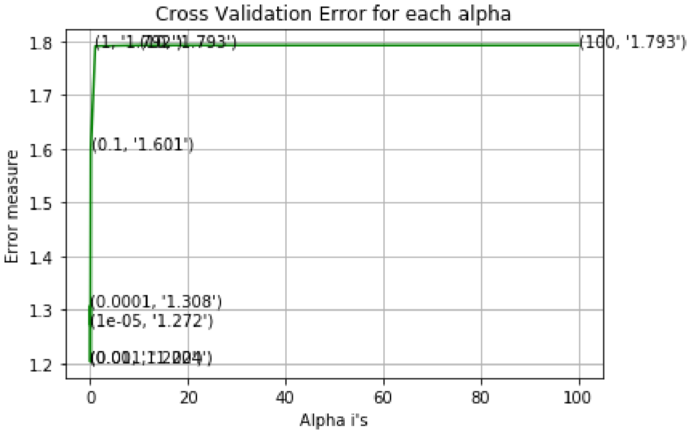
4.4.1. Hyper paramter tuning

In [69]:

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

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

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



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

alpha = [10 ** x for x in range(-5, 3)]
cv_log_error_array = []
```

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

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


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

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

```
for C = 1e-05
Log Loss : 1.2720627356169079
for C = 0.0001
Log Loss : 1.3077156189313521
for C = 0.001
Log Loss : 1.2038655837068433
for C = 0.01
Log Loss : 1.2022250262145127
for C = 0.1
Log Loss : 1.6008800282089042
for C = 1
Log Loss : 1.7918128834508797
for C = 10
Log Loss : 1.7932438941351054
for C = 100
Log Loss : 1.7932438986029786
```



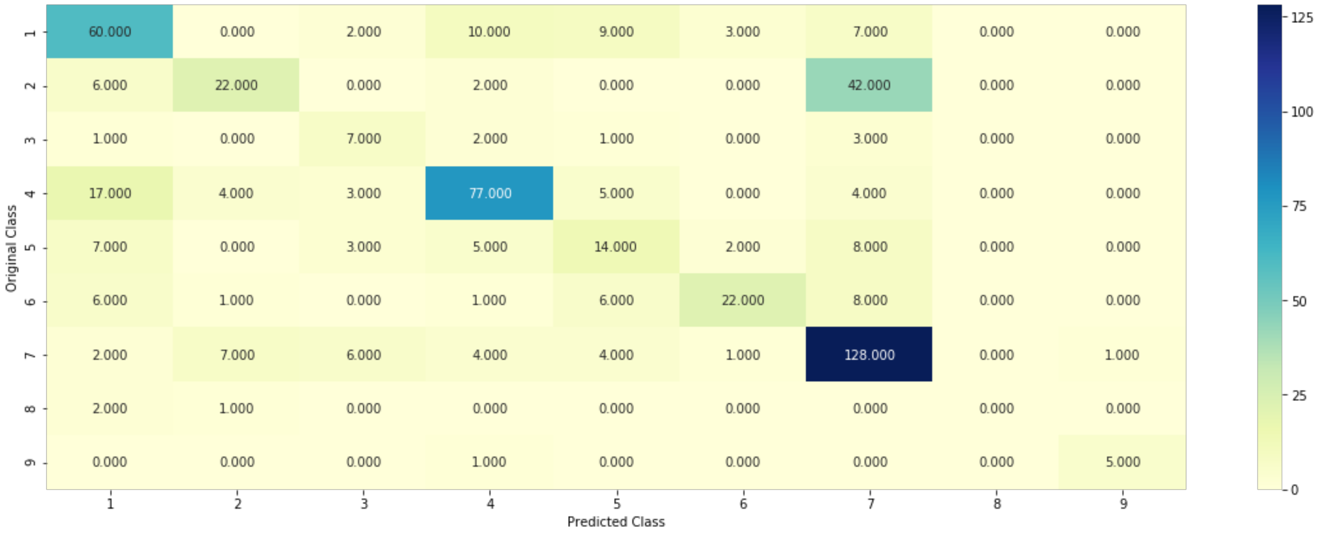
```
For values of best alpha =  0.01 The train log loss is: 0.7847783698836049
For values of best alpha =  0.01 The cross validation log loss is: 1.2022250262145127
For values of best alpha =  0.01 The test log loss is: 1.1368060709237942
```

4.4.2. Testing model with best hyper parameters

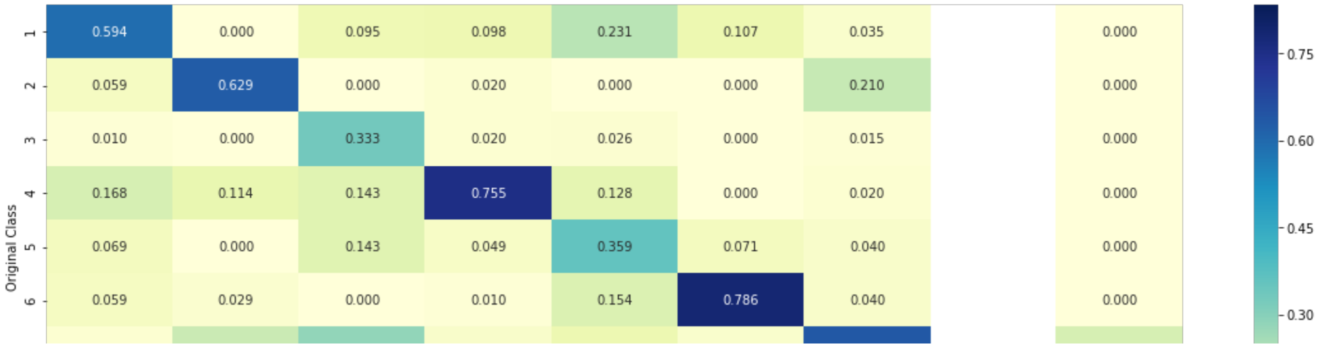
In [70]:

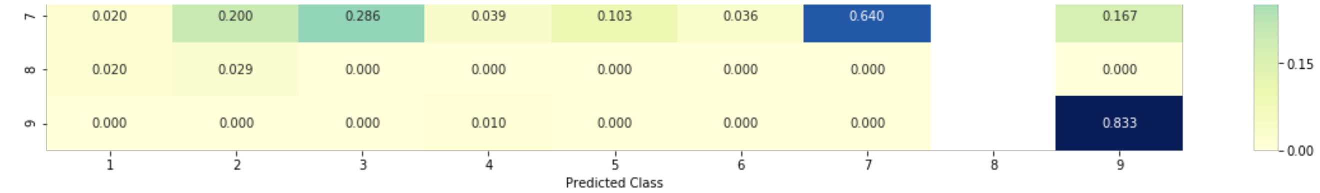
```
# read more about support vector machines with linear kernals here http://scikit-
learn.org/stable/modules/generated/sklearn.svm.SVC.html

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

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


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

```
Log loss : 1.2022250262145127
Number of mis-classified points : 0.37030075187969924
-------------------- Confusion matrix --------------------
```

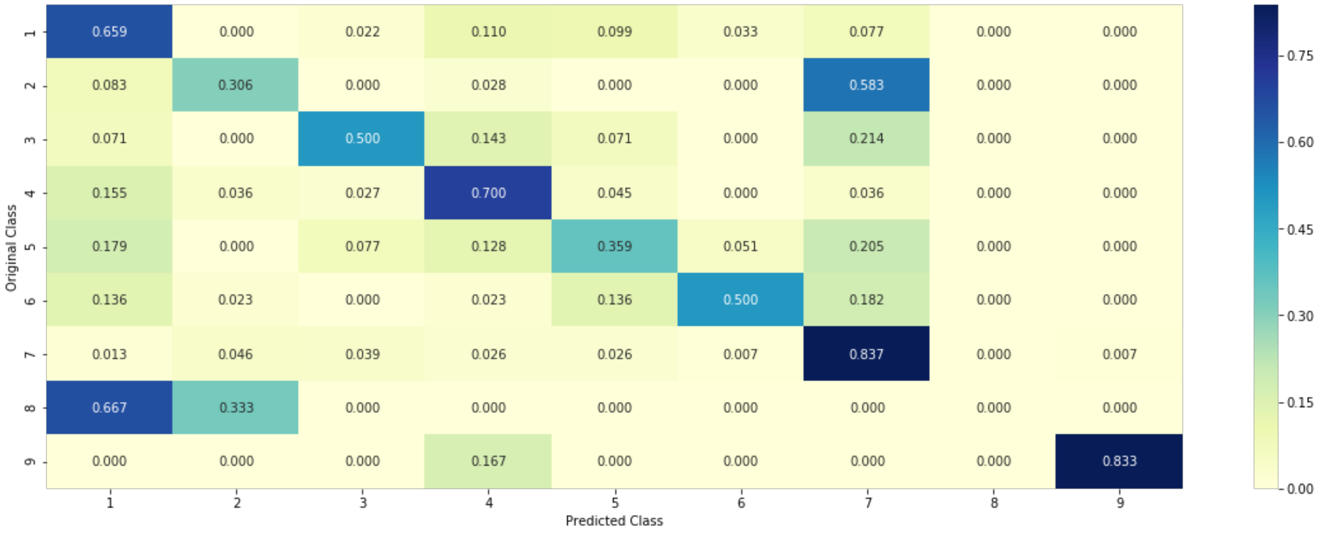


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [71]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
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 : 4
Predicted Class Probabilities: [[2.380e-02 6.000e-03 3.900e-03 7.574e-01 7.000e-03 4.400e-03 1.942
e-01
  3.000e-03 3.000e-04]]
Actual Class : 4
--------------------------------------------------
34 Text feature [cbf] present in test data point [True]
103 Text feature [exons] present in test data point [True]
126 Text feature [43] present in test data point [True]
175 Text feature [blotting] present in test data point [True]
190 Text feature [facscalibur] present in test data point [True]
222 Text feature [bio] present in test data point [True]
267 Text feature [consider] present in test data point [True]
270 Text feature [common] present in test data point [True]
279 Text feature [cytometric] present in test data point [True]
280 Text feature [collaborate] present in test data point [True]
307 Text feature [city] present in test data point [True]
331 Text feature [currently] present in test data point [True]
340 Text feature [collectively] present in test data point [True]
```

```
340 Text feature [collectively] present in test data point [True]
385 Text feature [crisis] present in test data point [True]
390 Text feature [d171n] present in test data point [True]
420 Text feature [collaborates] present in test data point [True]
429 Text feature [analyzer] present in test data point [True]
432 Text feature [express] present in test data point [True]
480 Text feature [36] present in test data point [True]
Out of the top  500  features  19 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [72]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", 
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
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: [[0.0468 0.1226 0.0116 0.0716 0.0549 0.3026 0.3805 0.007  0.0025]]
Actual Class : 5
--------------------------------------------------
52 Text feature [cl] present in test data point [True]
147 Text feature [american] present in test data point [True]
165 Text feature [16] present in test data point [True]
187 Text feature [d477g] present in test data point [True]
206 Text feature [embryonic] present in test data point [True]
256 Text feature [2a] present in test data point [True]
306 Text feature [cm] present in test data point [True]
312 Text feature [120] present in test data point [True]
328 Text feature [allprep] present in test data point [True]
369 Text feature [449] present in test data point [True]
380 Text feature [ability] present in test data point [True]
399 Text feature [array] present in test data point [True]
407 Text feature [ca] present in test data point [True]
475 Text feature [assays] present in test data point [True]
478 Text feature [6b] present in test data point [True]
Out of the top  500  features  15 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [73]:

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

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

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).
# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-construction-2/
# --------------------------------


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

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

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

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

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2633914831739768
for n_estimators = 100 and max depth =  10
Log Loss : 1.2185448940913008
for n_estimators = 200 and max depth =  5
Log Loss : 1.2657597312782587
for n_estimators = 200 and max depth =  10
Log Loss : 1.2115271513207009
for n_estimators = 500 and max depth =  5
```

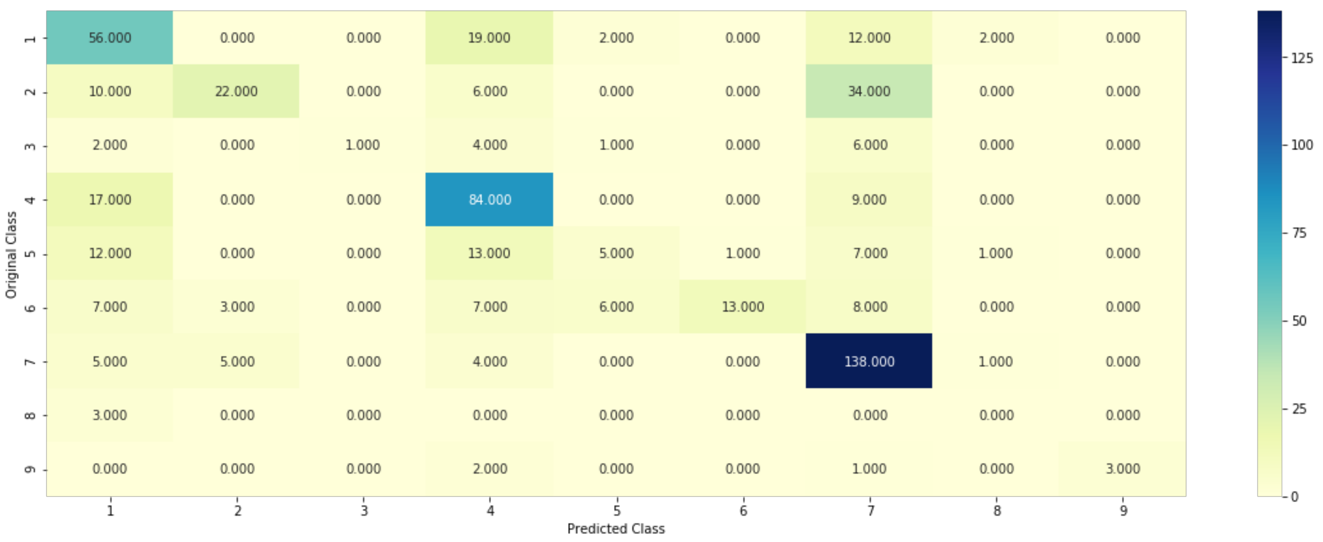
```
Log Loss : 1.2597920074299445
for n_estimators = 500 and max depth =  10
Log Loss : 1.2085279509799574
for n_estimators = 1000 and max depth =  5
Log Loss : 1.260156119169178
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2066313349098094
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2550146222207985
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2025950083016586
For values of best estimator =  2000 The train log loss is: 0.646795340093057
For values of best estimator =  2000 The cross validation log loss is: 1.2025950083016586
For values of best estimator =  2000 The test log loss is: 1.132256555035761
```

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

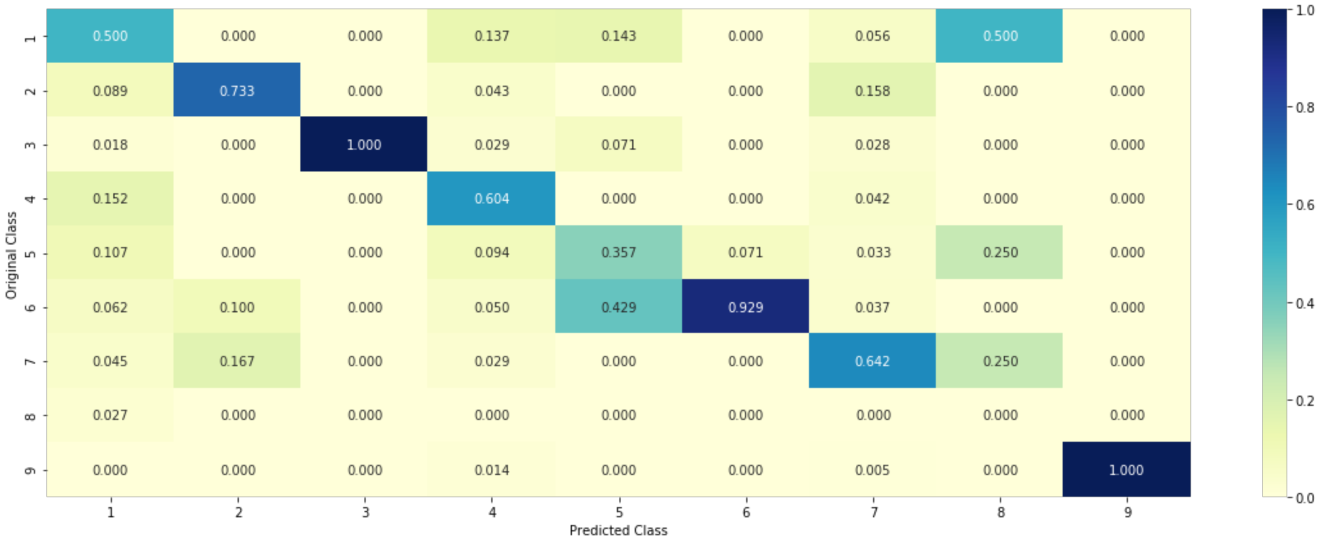
In [74]:

```
# --------------------------------
# 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)
```

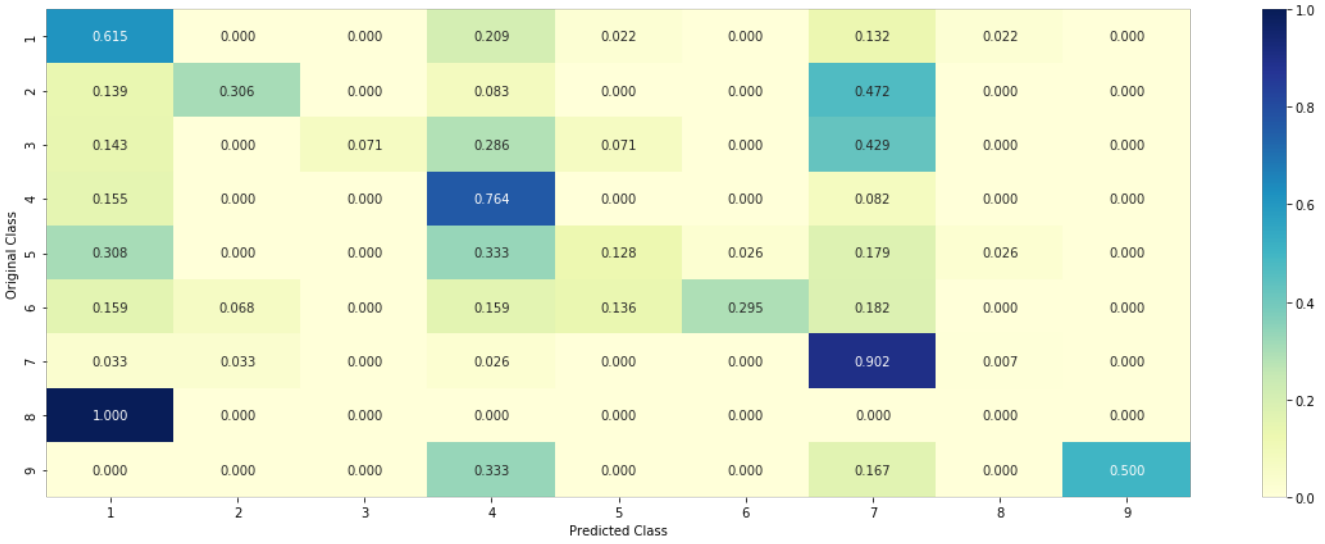
```
Log loss : 1.2025950083016586
Number of mis-classified points : 0.39473684210526316
-------------------- Confusion matrix --------------------
```



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



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [75]:

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

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
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 : 4
Predicted Class : 4
Predicted Class Probabilities: [[0.2692 0.0954 0.0226 0.363  0.052  0.0456 0.1283 0.0082 0.0157]]
Actual Class : 4
--------------------------------------------------
4 Text feature [16] present in test data point [True]
38 Text feature [blasts] present in test data point [True]
Out of the top  100  features  2 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [76]:

```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
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: [[0.1354 0.1429 0.0261 0.1194 0.0581 0.1    0.3994 0.0092 0.0096]]
Actuall Class : 5
--------------------------------------------------
4 Text feature [16] present in test data point [True]
19 Text feature [adults] present in test data point [True]
23 Text feature [daily] present in test data point [True]
64 Text feature [absent] present in test data point [True]
85 Text feature [embryonic] present in test data point [True]
Out of the top  100  features  5 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [77]:

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

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

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

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


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

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

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

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

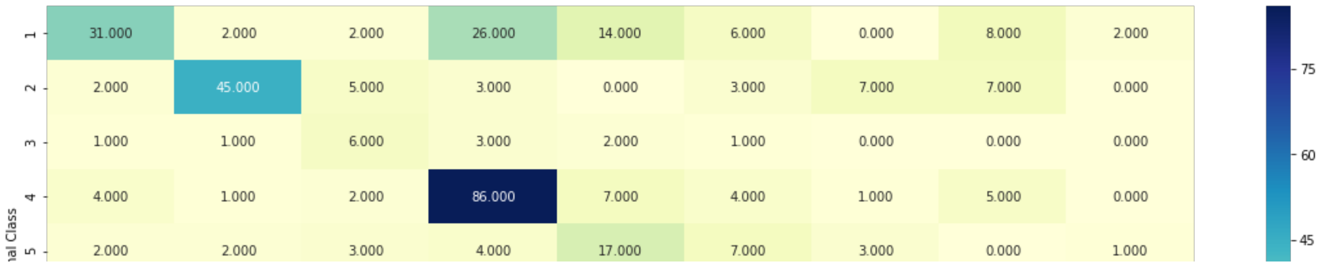
```
for n_estimators = 10 and max depth =  2
Log Loss : 2.30579709462392
for n_estimators = 10 and max depth =  3
Log Loss : 1.8233095556520926
for n_estimators = 10 and max depth =  5
Log Loss : 1.5386614102617446
for n_estimators = 10 and max depth =  10
Log Loss : 1.8005287935879848
for n_estimators = 50 and max depth =  2
Log Loss : 1.8254499515456843
for n_estimators = 50 and max depth =  3
Log Loss : 1.4716320104760638
for n_estimators = 50 and max depth =  5
Log Loss : 1.4530000015347897
for n_estimators = 50 and max depth =  10
Log Loss : 1.805745911151148
for n_estimators = 100 and max depth =  2
Log Loss : 1.6778281613144799
for n_estimators = 100 and max depth =  3
Log Loss : 1.5441085430648338
for n_estimators = 100 and max depth =  5
Log Loss : 1.4806152627307225
for n_estimators = 100 and max depth =  10
Log Loss : 1.833830161955343
for n_estimators = 200 and max depth =  2
for n_estimators = 200 and max depth =  2
Log Loss : 1.7654979466404417
for n_estimators = 200 and max depth =  3
Log Loss : 1.5784347585969452
for n_estimators = 200 and max depth =  5
Log Loss : 1.4757763784265217
for n_estimators = 200 and max depth =  10
Log Loss : 1.7851663966248867
for n_estimators = 500 and max depth =  2
Log Loss : 1.8260321793301235
for n_estimators = 500 and max depth =  3
Log Loss : 1.6656415600363867
for n_estimators = 500 and max depth =  5
Log Loss : 1.4693592952686492
for n_estimators = 500 and max depth =  10
Log Loss : 1.7884349471386365
for n_estimators = 1000 and max depth =  2
Log Loss : 1.7966894058319445
for n_estimators = 1000 and max depth =  3
Log Loss : 1.675284475896871
for n_estimators = 1000 and max depth =  5
Log Loss : 1.4509894403006207
for n_estimators = 1000 and max depth =  10
Log Loss : 1.7820672176482721
For values of best alpha =  1000 The train log loss is: 0.05106115818042505
For values of best alpha =  1000 The cross validation log loss is: 1.4509894403006207
For values of best alpha =  1000 The test log loss is: 1.4191271401466348
```

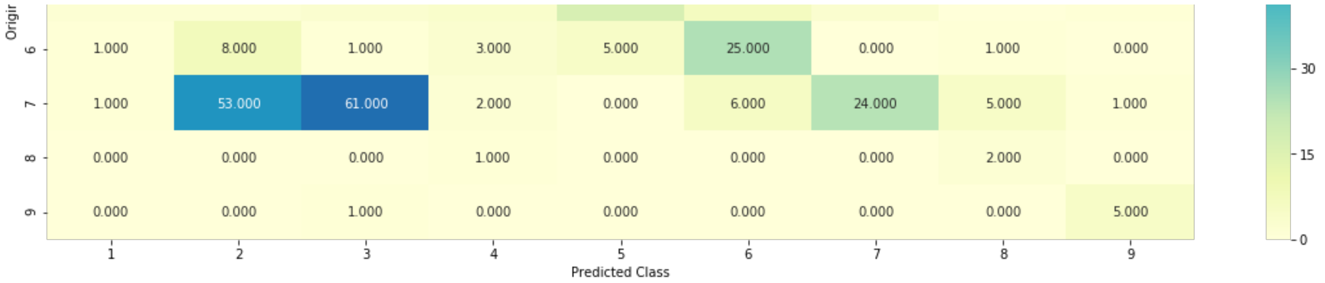
4.5.4. Testing model with best hyper parameters (Response Coding)

In [78]:

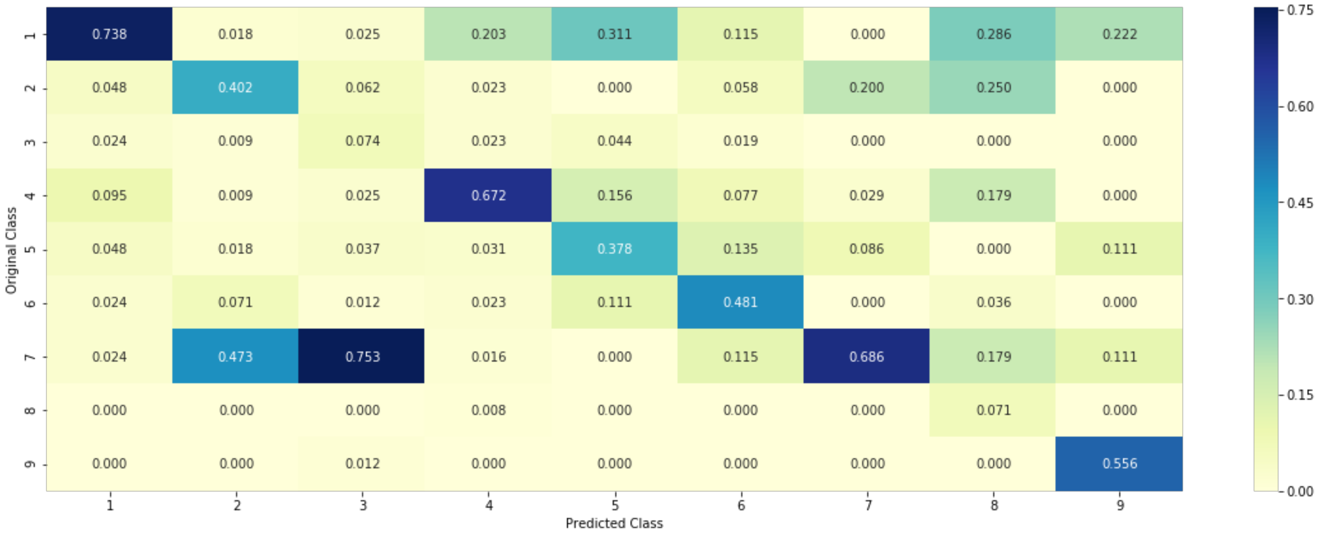
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None,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

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

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

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

clf = RandomForestClassifier(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 : 1.4509894403006207
Number of mis-classified points : 0.5469924812030075
-------------------- Confusion matrix --------------------
```

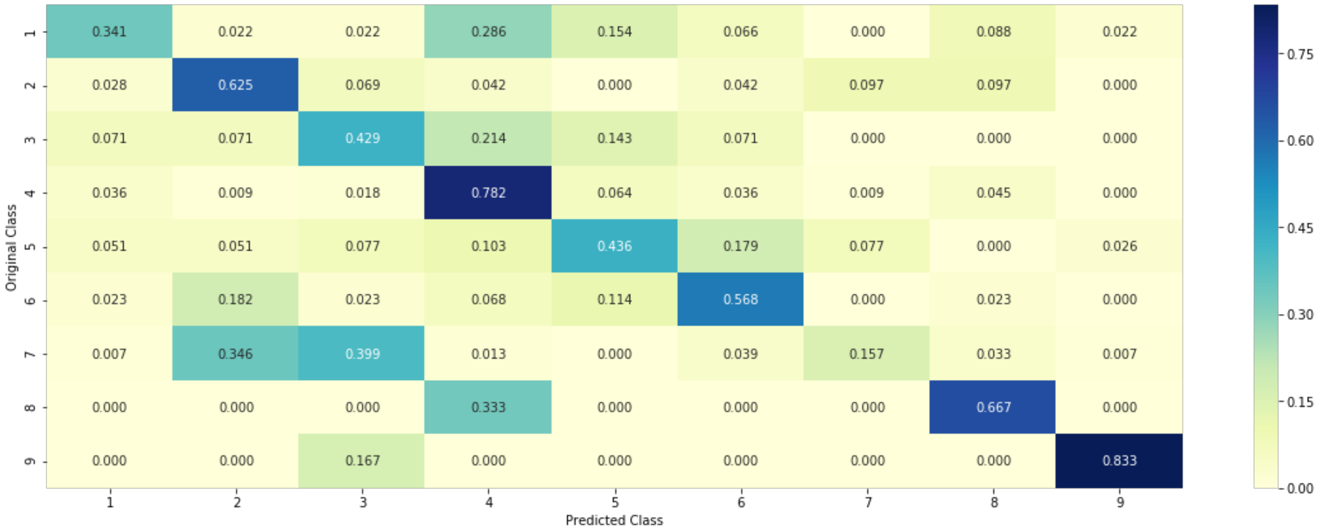




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [79]:

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


test_point_index = 1
```

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0742 0.02   0.1152 0.6157 0.0299 0.0472 0.01   0.0366 0.0511]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [80]:

```
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 : 2
Predicted Class Probabilities: [[0.0164 0.3935 0.0851 0.021  0.0297 0.2812 0.1258 0.036  0.0113]]
Actual Class : 5
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [81]:

```
# 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_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
verbose=0, warm_start=False,
# class_weight=None)

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

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

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


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

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


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

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

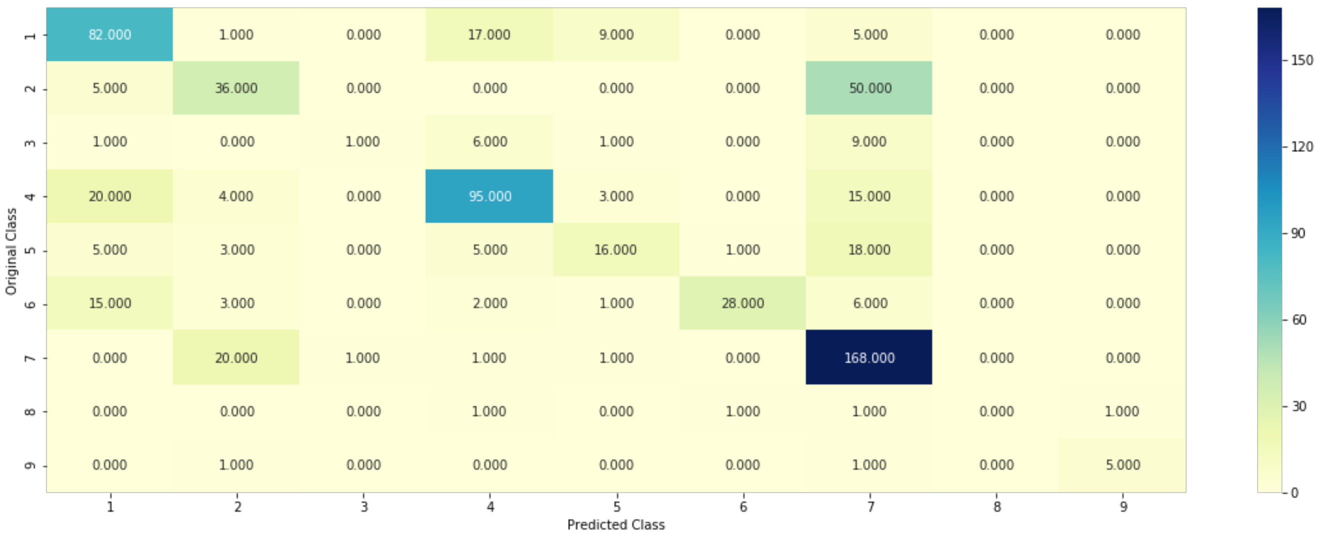
```
Logistic Regression :  Log Loss: 1.21
Support vector machines : Log Loss: 1.79
Naive Bayes : Log Loss: 1.35
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.033
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.522
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.213
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.378
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.706
```

4.7.2 testing the model with the best hyper parameters

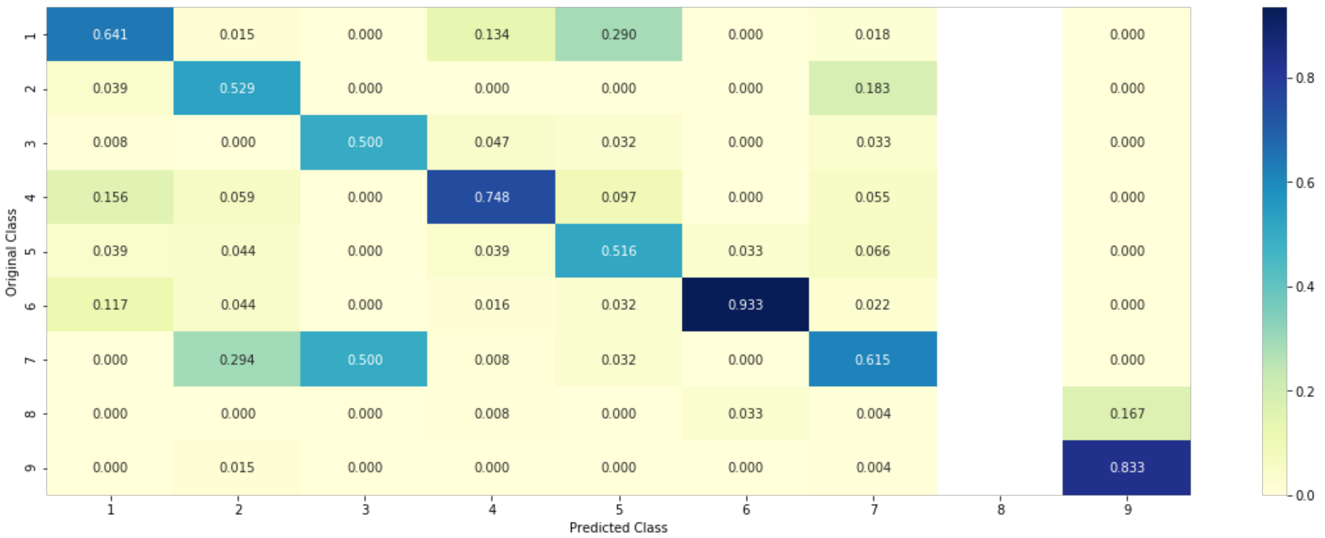
In [82]:

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

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

log_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))
```

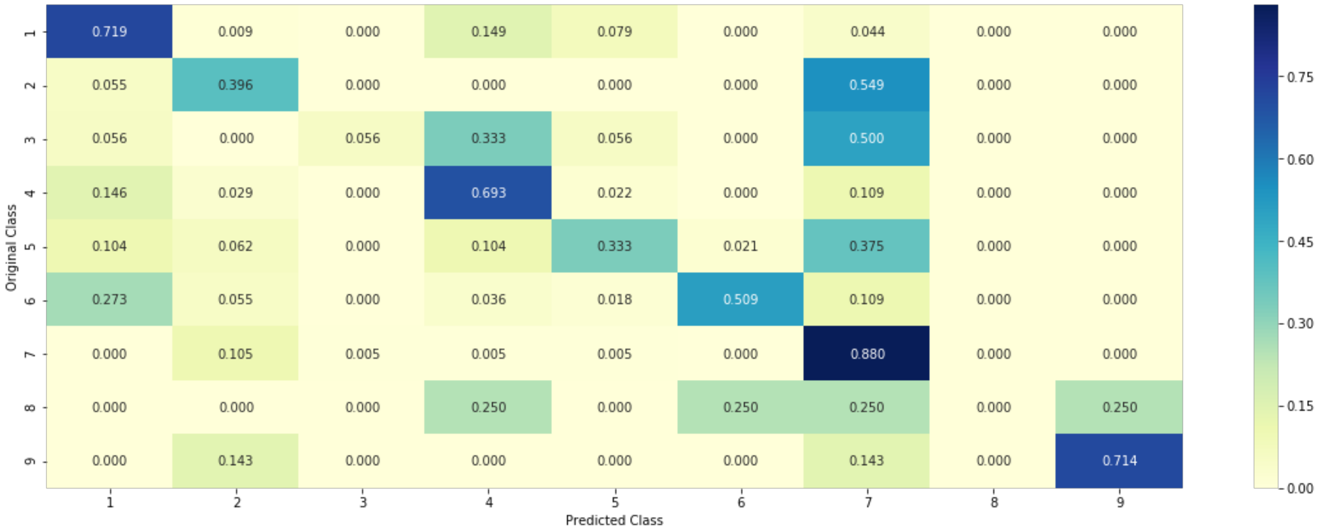
```
Log loss (train) on the stacking classifier : 0.6684467781013348
Log loss (CV) on the stacking classifier : 1.2126977684295641
Log loss (test) on the stacking classifier : 1.1255728789130635
Number of missclassified point : 0.3518796992481203
-------------------- Confusion matrix --------------------
```



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



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

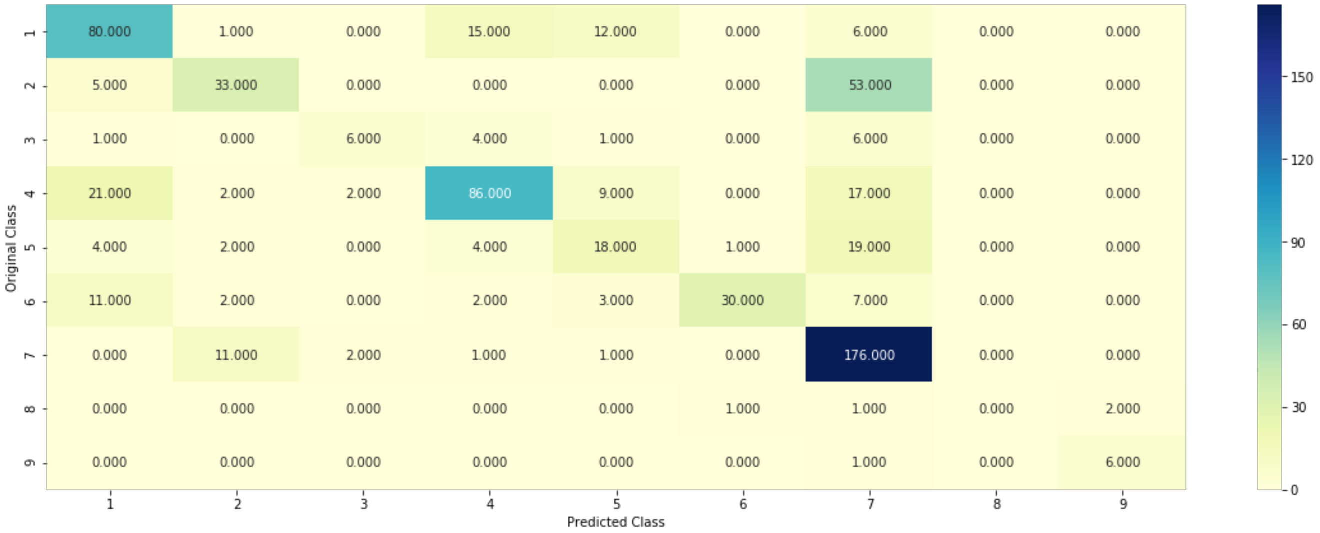


4.7.3 Maximum Voting classifier

In [83]:

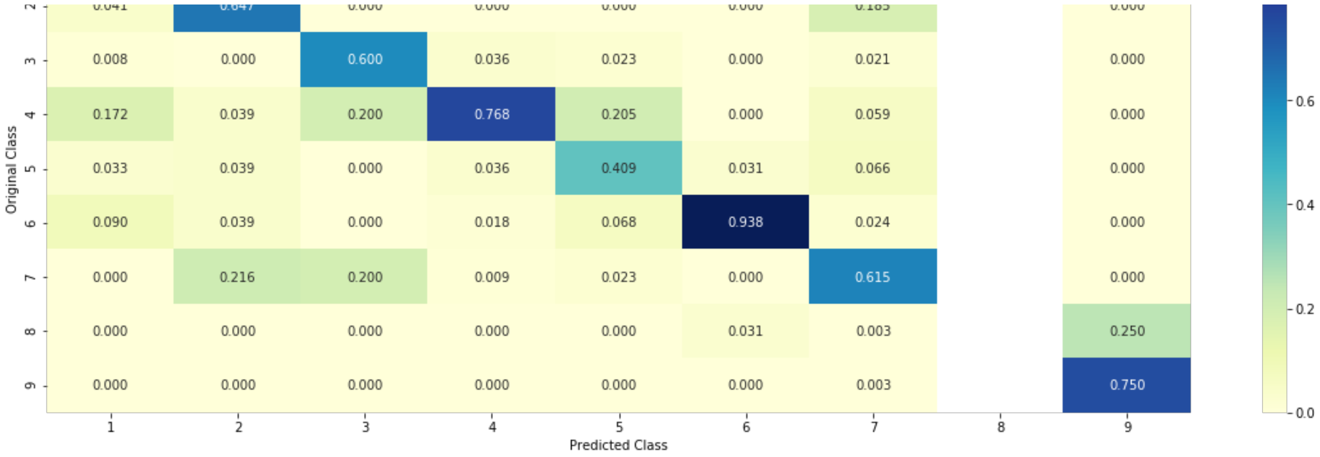
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting=
'soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log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

```
Log loss (train) on the VotingClassifier : 0.8997566841171555
Log loss (CV) on the VotingClassifier : 1.2403290644956153
Log loss (test) on the VotingClassifier : 1.1931646260242685
Number of missclassified point : 0.3458646616541353
-------------------- Confusion matrix --------------------
```

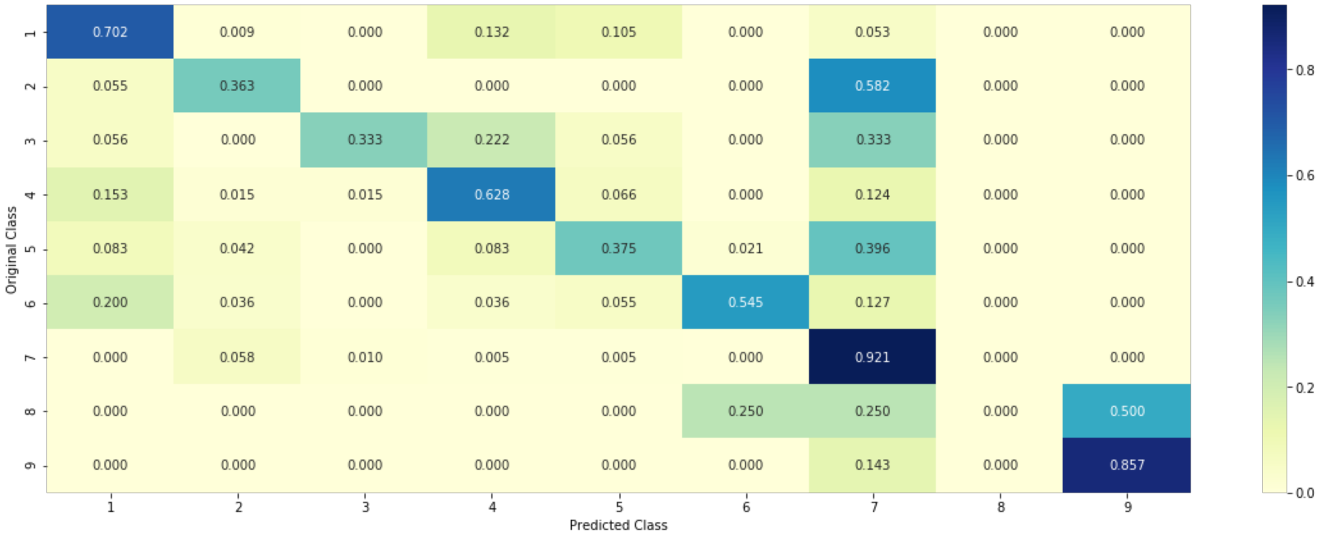


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```

1 0.656 0.020 0.000 0.134 0.273 0.000 0.021 0.000
0.8



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



5. Conclusions

5.1 Steps taken

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

5.2 Model Comparision

In [85]:

```
#https://www.kaggle.com/premvardhan/amazon-fine-food-reviews-analysis-using-knn
models = pd.DataFrame({'Model': ['Naive Bayes', "K-NN","Logistic Regression with Class Balancing",'
Logistic Regression without Class Balancing','Linear SVM','Random forest','Random forest(response
encoding)','Stacking classifier','Maximum voting classifier'], 'Log-Loss': [1.35,1.10,1.23,1.25,1.
20,1.20,1.45,1.21,1.24], 'Percent of Misclassified points':[.44,.39,0.35,.36,.37,.39,.54,.35,.34] ,
'Hyperparameter': [.001,15,.001,.001,.01,2000,100,.1,0]}, columns = ["Model", "Log-Loss", "Percent
of Misclassified points", "Hyperparameter"])
models
```

Out[85]:

	Model	Log-Loss	Percent of Misclassified points	Hyperparameter
0	Naive Bayes	1.35	0.44	0.001
1	K-NN	1.10	0.39	15.000
2	Logistic Regression with Class Balancing	1.23	0.35	0.001

	Model	Log-Loss	Percent of Misclassified points	Hyperparameter
3	Logistic Regression without Class Balancing	1.25	0.36	0.001
4	Linear SVM	1.20	0.37	0.010
5	Random forest	1.20	0.39	2000.000
6	Random forest(response encoding)	1.45	0.54	100.000
7	Stacking classifier	1.21	0.35	0.100
8	Maximum voting classifier	1.24	0.34	0.000

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```