Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [3]:

```
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.cross_validation import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
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 [4]:

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

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

Out[4]:

ID	Gene	Variation	Class

	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 [5]:

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

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

Out[5]:

	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 [6]:

```
# 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 + " "
                string += word + " "

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

In [7]:

```
#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 : 308.716593 seconds
```

In [8]:

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

Out[8]:

	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 [9]:

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

Out[9]:

	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 [10]:

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

In [11]:

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

Out[11]:

	ID	Gene	Variation	Class	TEXT

	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 [12]:

```
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 [13]:

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

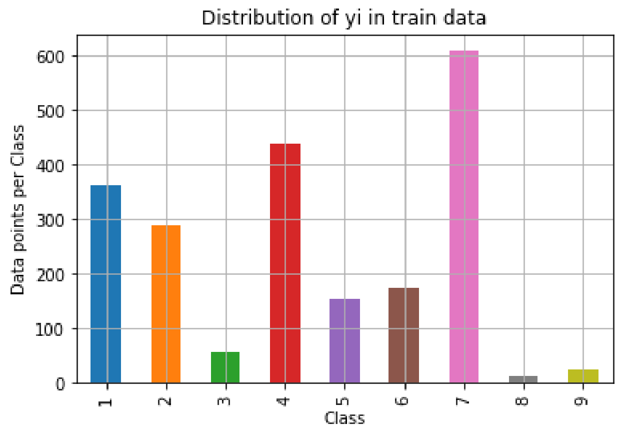
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

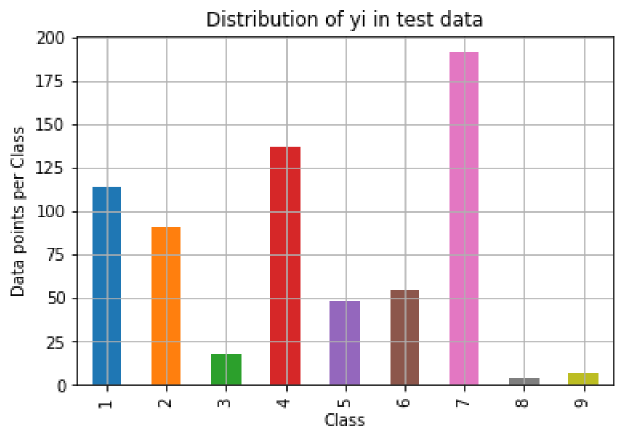
In [14]:

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

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

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


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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
```

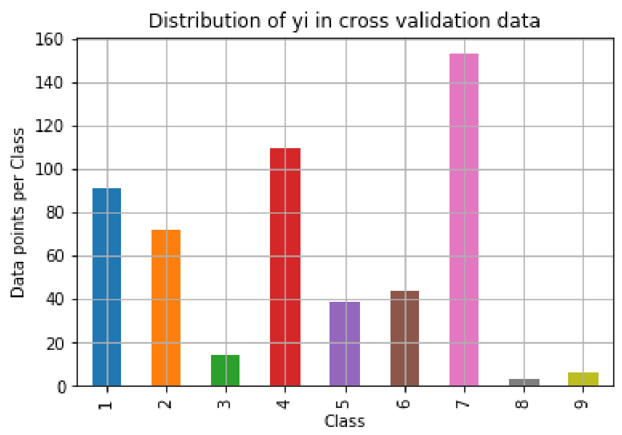
```
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.values[i], '(', np.rou
nd((test_class_distribution.values[i]/test_df.shape[0]*100), 3), '%)')

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

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



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



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



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

3.2 Prediction using a 'Random' Model

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

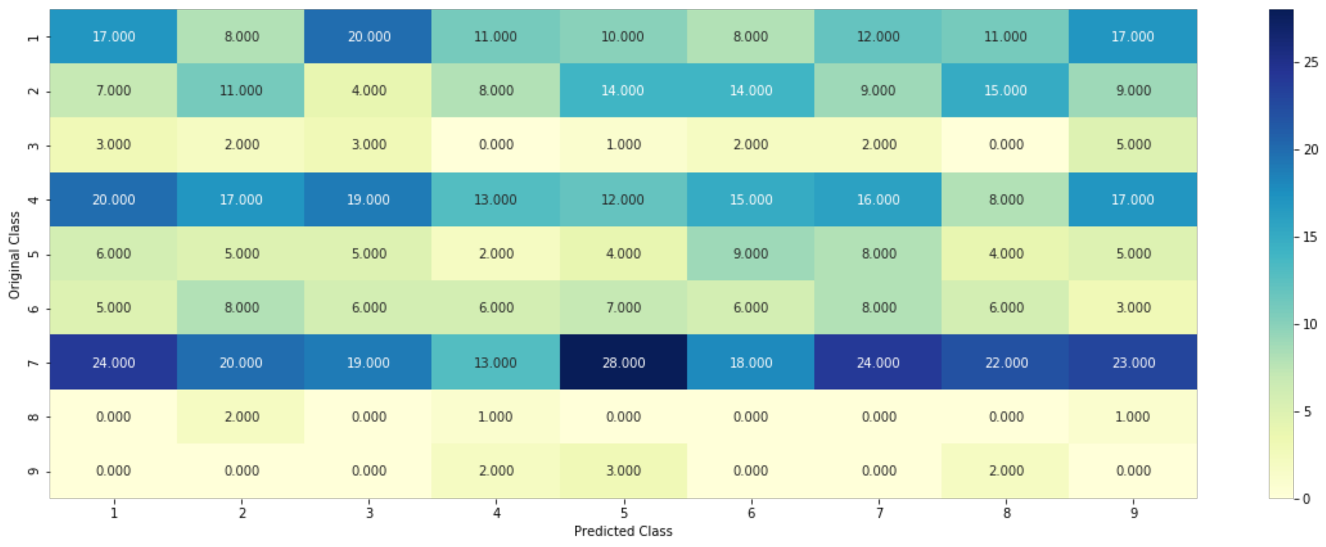
In [15]:

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

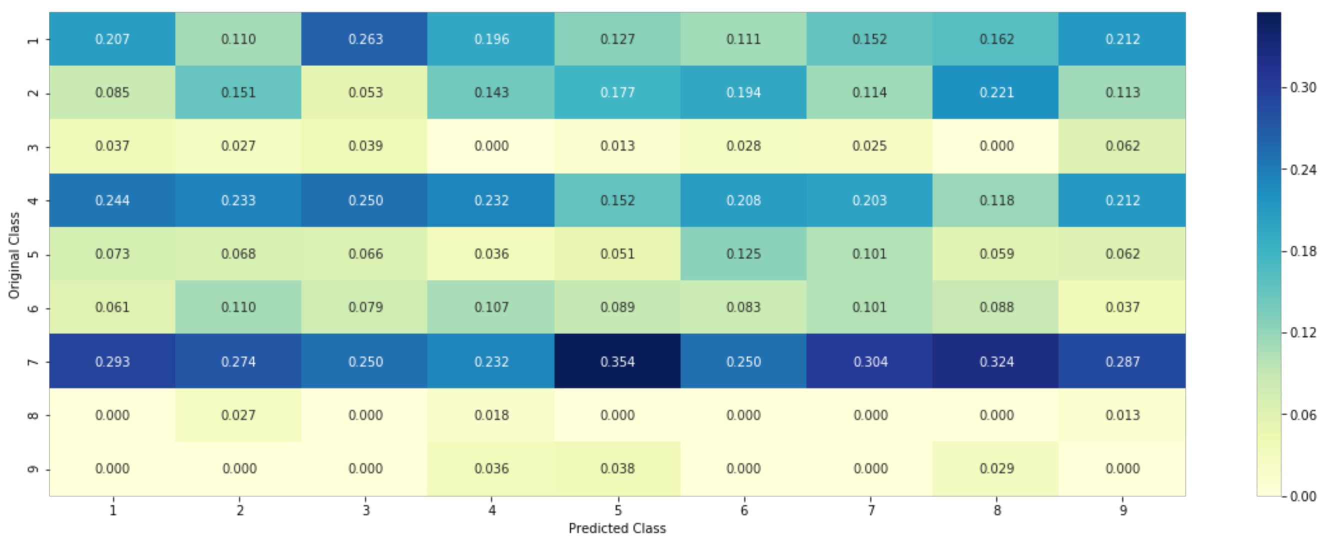
In [16]:

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

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

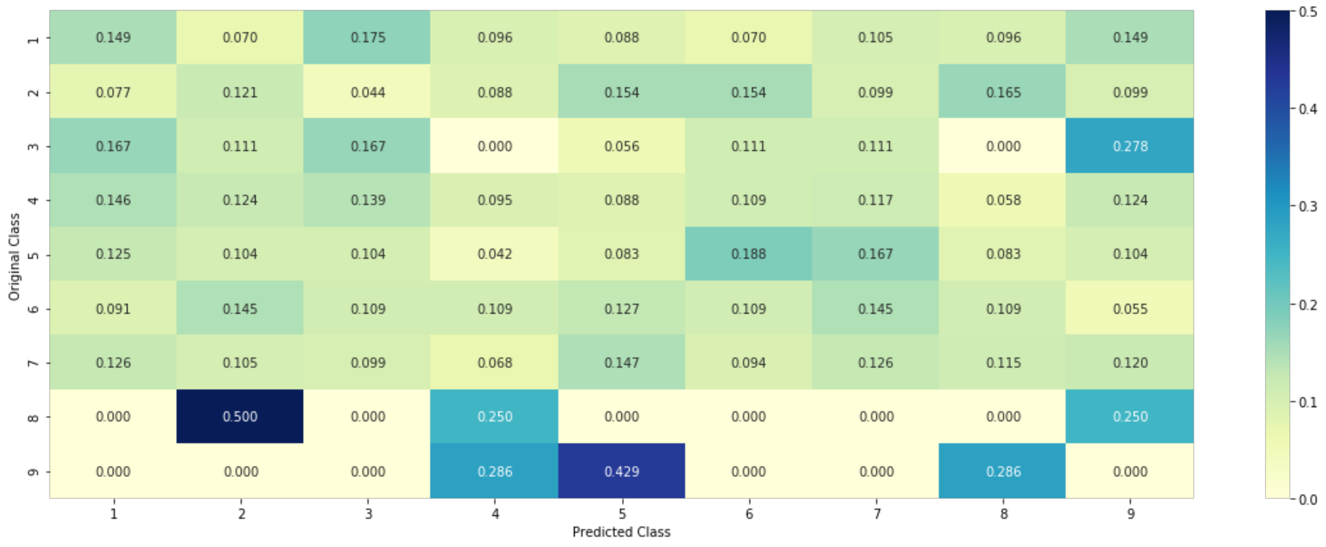
```
Log loss on Cross Validation Data using Random Model 2.461466928286884
Log loss on Test Data using Random Model 2.4568628755858533
-------------------- Confusion matrix --------------------
```



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



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



3.3 Univariate Analysis

In [17]:

```
# 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, 0.037878787878787878788, 0.068181818181818177,
0.13636363636363635, 0.25, 0.19318181818181818, 0.037878787878787878788, 0.037878787878787878788,
0.037878787878787878788],
    #      '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,
0.06666666666666666, 0.06666666666666666],
    #      ...
    #     }
    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 [18]:

```
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 : 232
BRCA1     171
TP53      101
PTEN       88
BRCA2      86
EGFR       86
KIT        65
BRAF       58
PDGFRA     45
ALK        42
ERBB2      39
Name: Gene, dtype: int64
```

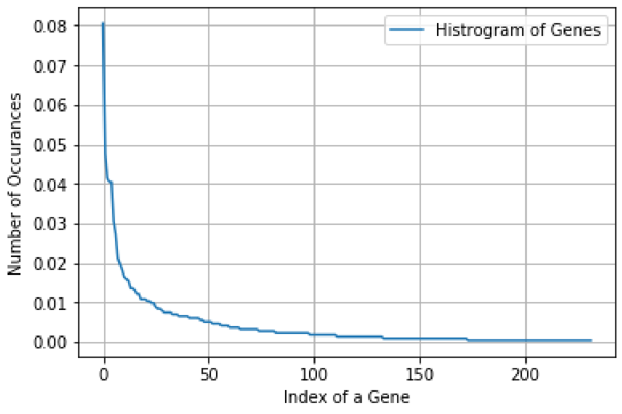
In [19]:

```
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 232 different categories of genes in the train data, and they are distibuted as fol
lows
```

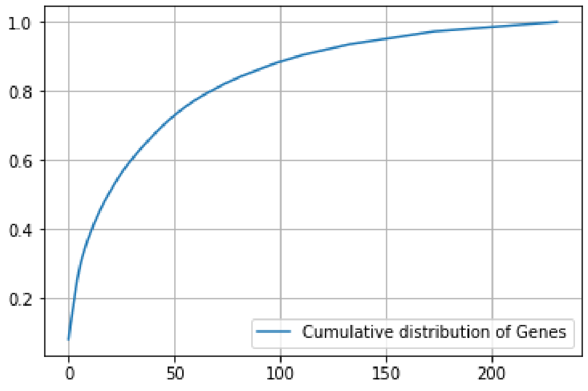
In [20]:

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



In [21]:

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



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

In [22]:

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

In [23]:

```
print("train_gene_feature_responseCoding is converted feature using respone coding method. The 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 [24]:

```
# 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 [25]:

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

Out[25]:

```
1952    CTLA4
541     SMAD2
236      EGFR
299      PAK1
1020     TSC2
Name: Gene, dtype: object
```

In [26]:

```
gene_vectorizer.get_feature_names()
```

Out[26]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl1',
 'asxl2',
 'atm',
 'atrx',
 'aurka',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk6',
 'cdkn1a',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cebpa',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctla4',
 'ctnnb1',
 'ddr2',
 'dicer1',
 'dnmt3a',
 'dnmt3b',
 'dusp4',
 'egfr',
 'eif1ax',
 'elf3',
 'ep300',
 'epas1',
 'epcam',
 'erbb2',
 'erbb3',
 'erbb4',
 'ercc2',
 'ercc4',
 'erg',
 'errfi1',
 'esr1',
 'etv1',
 'etv6',
 'ewsr1',
 'ezh2',
 'fam58a',
 'fanca',
 'fancc',
 'fat1',
 'fbxw7',
 'fgf19',
 'fgf3',
 'fgf4',
 'fgfr1',
 'fgfr2',
 'fgfr3',
 'fgfr4',
 'flt1',
 'flt3',
 'foxa1',
 'foxp1',
 'fubp1',
 'gata3',
 'gli1',
 'gnaq',
 'gnas',
 'h3f3a',
 'hla',
 'hnf1a',
 'hras',
 'idh1',
 'idh2',
 'igf1r',
 'ikbke',
 'il7r',
 'inpp4b',
 'jak1',
 'jak2',
 'kdm5a',
 'kdm5c',
 'kdm6a',
 'kdr',
 'keap1',
 'kit',
 'kmt2a',
 'kmt2c',
 'kmt2d',
 'knstrn',
 'kras',
 'lats1',
 'map2k1',
 'map2k2',
 'map2k4',
 'map3k1',
 'mapk1',
 'mdm2',
 'mdm4',
 'med12',
 'mef2b',
 'met',
 'mga',
 'mlh1',
 'mpl',
 'msh2',
 'msh6',
 'mtor',
 'myc',
 'mycn',
 'myd88',
 'myod1',
 'ncor1',
 'nf1',
 'nf2',
 'nfe2l2',
 'nfkbia',
 'nkx2',
 'notch1',
 'notch2',
 'npm1',
 'nras',
 'nsd1',
 'ntrk1',
 'ntrk2',
 'ntrk3',
 'pak1',
 'pax8',
 'pbrm1',
 'pdgfra',
 'pdgfrb',
 'pik3ca',
 'pik3cb',
 'pik3cd',
 'pik3r1',
 'pik3r2',
 'pik3r3',
 'pim1',
 'pms2',
 'pole',
 'ppp2r1a',
 'ppp6c',
 'prdm1',
 'ptch1',
 'pten',
 'ptpn11',
 'ptprd',
 'ptprt',
 'rab35',
 'rac1',
 'rad21',
 'rad50',
 'rad51b',
 'rad51c',
 'rad54l',
 'raf1',
 'rara',
 'rasa1',
 'rb1',
 'rbm10',
 'ret',
 'rheb',
 'rhoa',
 'rictor',
 'rit1',
 'ros1',
 'runx1',
 'rxra',
 'sdhb',
 'sdhc',
 'setd2',
 'sf3b1',
 'shq1',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'stat3',
 'stk11',
 'tcf3',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'whsc1',
 'xrcc2',
 'yap1']
```

In [27]:

```
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, 232)
```

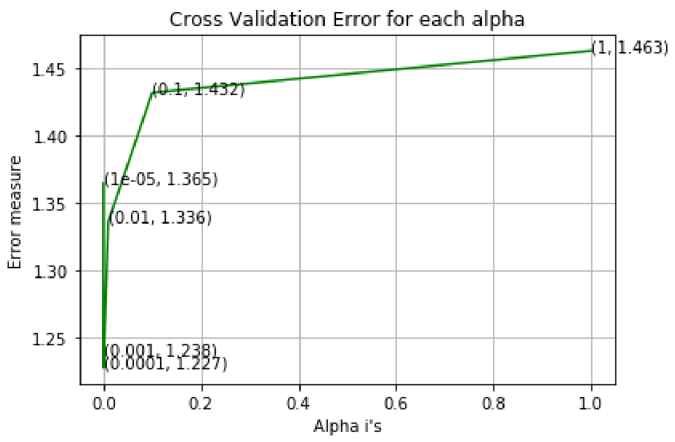
Q4. How good is this gene feature in predicting y_i?

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

In [29]:

```
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.


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: 1.3645544347177747
For values of alpha =  0.0001 The log loss is: 1.227446714793447
For values of alpha =  0.001 The log loss is: 1.237837315623884
For values of alpha =  0.01 The log loss is: 1.3360962303192339
For values of alpha =  0.1 The log loss is: 1.431992482117519
For values of alpha =  1 The log loss is: 1.4631779785295835
```



```
For values of best alpha =  0.0001 The train log loss is: 1.03592349585379
For values of best alpha =  0.0001 The cross validation log loss is: 1.227446714793447
For values of best alpha =  0.0001 The test log loss is: 1.25500401042499
```

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 [30]:

```
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  232  genes in train dataset?
Ans
1. In test data 644 out of 665 : 96.84210526315789
2. In cross validation data 516 out of  532 : 96.99248120300751
```

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 [31]:

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

```
Number of Unique Variations : 1922
Truncating_Mutations    59
Deletion                51
Amplification           47
Fusions                 25
Q61R                     3
T286A                    2
G12V                     2
Y42C                     2
G12C                     2
M1R                      2
Name: Variation, dtype: int64
```

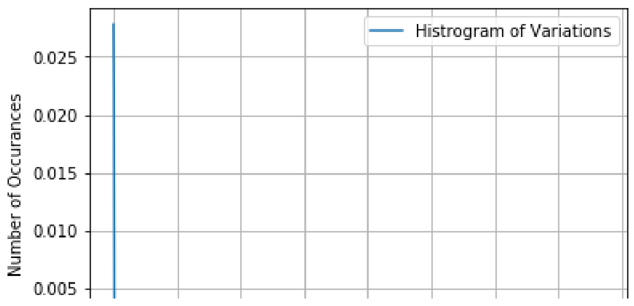
In [32]:

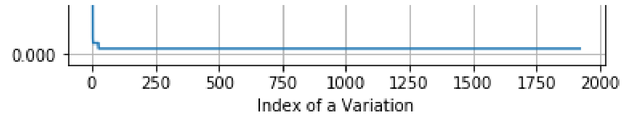
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of variations in the
train data, and they are distibuted as follows",)
```

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

In [33]:

```
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()
```

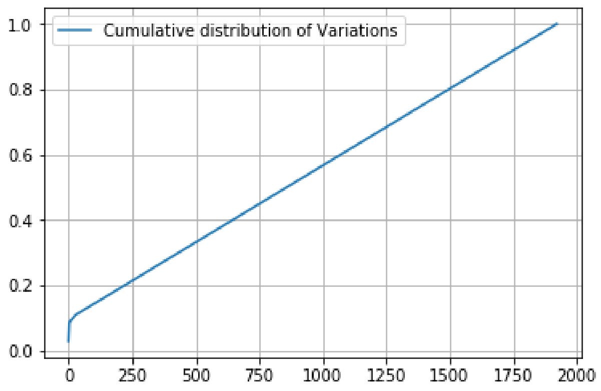




In [34]:

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

```
[0.02777778 0.05178908 0.07391714 ... 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 [35]:

```
# 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 [36]:

```
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 [37]:

```
# 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 [38]:

```
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, 1948)
```

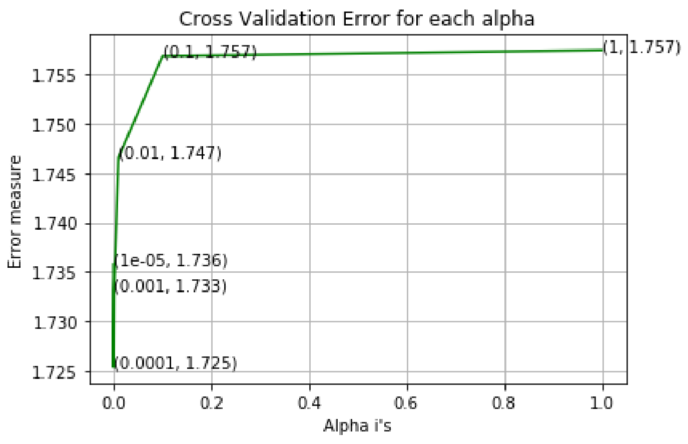
Q10. How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

In [39]:

```
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
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: 1.735767017974844
For values of alpha =  0.0001 The log loss is: 1.7253396930076925
For values of alpha =  0.001 The log loss is: 1.733171898730345
For values of alpha =  0.01 The log loss is: 1.7465618943598367
For values of alpha =  0.1 The log loss is: 1.7568096919792595
For values of alpha =  1 The log loss is: 1.7574073321163524
```



```
For values of best alpha =  0.0001 The train log loss is: 0.7709971733228396
For values of best alpha =  0.0001 The cross validation log loss is: 1.7253396930076925
For values of best alpha =  0.0001 The test log loss is: 1.737147683619957
```

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 [40]:

```
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  1922  genes in test and cross validation data
sets?
Ans
1. In test data 59 out of 665 : 8.87218045112782
2. In cross validation data 59 out of  532 : 11.090225563909774
```

3.2.3 Univariate Analysis on Text Feature using TF-IDF

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

In [41]:

```
# 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 [42]:

```
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 [45]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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


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

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

In [46]:

```
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 [47]:

```
#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 [48]:

```
# 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 [49]:

```
# 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 [50]:

```
#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 [51]:

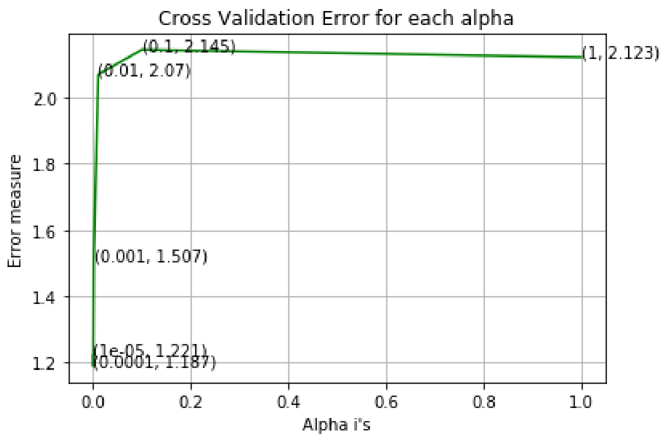
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({51.78482427213675: 1, 14.80142140123754: 1, 17.88522495110822: 1, 5.826510977001123: 1,
6.866219437738137: 1, 7.886165661759031: 1, 8.081188078532092: 1, 9.309288395303211: 1,
10.77064774230815: 1, 11.501146940053268: 1, 12.411077164003903: 1, 13.526167121639158: 1,
14.677030986401148: 1, 15.041851754097543: 1, 16.907890773479796: 1, 17.393316736154127: 1,
18.704724588745687: 1, 19.654631144808: 1, 20.345390779656057: 1, 21.53801774037188: 1,
22.997321178949797: 1, 23.756869400430915: 1, 24.381311847640948: 1, 25.17692562374429: 1,
26.03574231424655: 1, 27.527591245399133: 1, 28.023976315231533: 1, 29.200625652697045: 1,
30.122886875210423: 1, 31.556058492925747: 1, 32.05493004870892: 1, 33.462738272721566: 1,
34.47597743428858: 1, 35.66417349282799: 1, 36.32021330317139: 1, 6.906158416977559: 1,
38.58285108995812: 1, 39.775010044435895: 1, 18.137361023081564: 1, 41.782356861146496: 1,
14.558216568817011: 1, 7.724051174180241: 1, 44.36308152267425: 1, 45.73160564393715: 1, 46.0895281
26412525: 1, 47.83941464666562: 1, 48.265321212350884: 1, 8.41304571149047: 1, 10.89500652036099:
1, 51.48325179219575: 1, 7.404655126418957: 1, 17.452471555383536: 1, 9.42007011073284: 1,
56.976808342884006: 1, 58.78915999885831: 1, 59.43451306404411: 1, 14.021397850255003: 1,
10.865161705071312: 1, 62.86914373655131: 1, 36.222447516009325: 1, 64.75925400974624: 1,
66.47995468659107: 1, 11.595124068593659: 1, 68.33675467867349: 1, 69.57258849246843: 1,
71.09641721273502: 1, 12.2084103070575: 1, 75.20632194427156: 1, 28.59168094019329: 1,
13.444685230479132: 1, 11.19400276781047: 1, 81.2599507903724: 1, 10.370442457123698: 1,
84.27833689303439: 1, 14.435516785276258: 1, 7.234639561492996: 1, 89.92864808783139: 1,
15.203373905815392: 1, 92.00458368889011: 1, 13.569701968645475: 1, 20.844479782828124: 1,
24.500458380735402: 1, 16.20575896584985: 1, 18.08348407016396: 1, 17.36320345336917: 1, 9.00233644
7628638: 1, 17.633766038588554: 1, 7.603408669114954: 1, 18.770489034996643: 1,
108.56751668369235: 1, 18.91178204126431: 1, 12.208870467754107: 1, 111.086737267512: 1,
23.740909982544036: 1, 19.612811961541667: 1, 31.935605581498987: 1, 10.181609139693704: 1,
20.87590459455601: 1, 6.947096221246169: 1, 21.182866691598825: 1, 21.000595881910648: 1,
129.09047149165164: 1, 130.9880437346769: 1, 22.344366776645582: 1, 14.063479529502695: 1,
8.091073605743077: 1, 23.553463030015774: 1, 140.73184878803542: 1, 24.086236162544676: 1,
9.563147542135736: 1, 9.835206599408274: 1, 25.937942521296762: 1, 7.827459793899691: 1,
18.7804983934117: 1, 19.371412146152295: 1, 10.274903156338794: 1, 26.149464075009792: 1,
9.742295356632399: 1, 27.523734228042553: 1, 9.598482943242203: 1, 31.936446419822307: 1,
28.524464680312878: 1, 14.669807019583711: 1, 17.62783040142551: 1, 20.967057999256472: 1,
29.484320881022022: 1, 9.565986218889952: 1, 10.452057184394958: 1, 26.61251293418336: 1,
30.945291397013495: 1, 182.66208758342478: 1, 40.914517396695416: 1, 23.14214233342394: 1,
31.308099969540272: 1, 13.595438884473992: 1, 8.215340472176415: 1, 32.724321456265315: 1,
10.387749361740791: 1, 7.048305304747947: 1, 8.03932732544046: 1, 33.870769129333034: 1,
15.949616113240946: 1, 22.72482878029209: 1, 34.21616568044217: 1, 11.180746025488359: 1,
8.764688932700624: 1, 35.21757276596143: 1, 32.56709269181555: 1, 12.083652283158107: 1,
7.357040357979912: 1, 36.21902577729707: 1, 10.592106113963254: 1, 37.98950685419891: 1,
10.221693938042408: 1, 19.272514199629946: 1, 32.386505098555745: 1, 38.18998663227135: 1,
16.500027409627446: 1, 13.557555383532145: 1, 8.307512482500284: 1, 19.222418239116358: 1,
12.579871710672561: 1, 40.71226052505412: 1, 14.332701701715976: 1, 15.40273119653294: 1,
8.880708030798814: 1, 41.94910752738752: 1, 249.11739832331338: 1, 12.963920165202277: 1,
42.655547873536044: 1, 9.893343884378977: 1, 8.847669789494342: 1, 17.884493679836478: 1,
10.230085502492733: 1, 44.29938988654204: 1, 8.831470324584954: 1, 13.18520877718606: 1, 8.80514571
4867395: 1, 20.405321140975754: 1, 9.430860328967725: 1, 8.898845433351307: 1, 7.598055262491841:
1, 47.640828414085334: 1, 9.829289208538993: 1, 9.622560645682048: 1, 11.129438204319882: 1,
7.576041866406275: 1, 18.74877629962779: 1, 12.856759542454311: 1, 8.931358807726308: 1,
14.257842569538836: 1, 44.590981943910734: 1, 27.71213024038226: 1, 10.933557824929295: 1,
51.99321042992181: 1, 8.353616201839419: 1, 12.263493817942814: 1, 8.396144778924386: 1,
9.374985921361809: 1, 10.481689024296143: 1, 19.25276090679733: 1, 10.008175445487527: 1,
14.347455972863878: 1, 15.426054987953675: 1, 79.12680785902717: 1, 55.84209367913495: 1,
13.349295370601745: 1, 28.185571373976153: 1, 12.537238501247106: 1, 11.602145075421106: 1,
11.076211211700269: 1, 24.93477172438608: 1, 18.191278603078068: 1, 58.9623836440078: 1,
20.6215066996979: 1, 12.524450497121418: 1, 23.654037349613507: 1, 8.527128947549508: 1,
16.06636995155841: 1, 29.778138078774923: 1, 12.478437145203173: 1, 10.117651124140474: 1,
7.8014001544927645: 1, 26.428076481018604: 1, 14.89303286490846: 1, 8.842784183232796: 1,
10.73572452928116: 1, 63.74006883247057: 1, 21.963282070705983: 1, 11.005977843275039: 1,
24.101347356525515: 1, 34.93439949029828: 1, 64.29171123339644: 1, 15.98243508457122: 1, 7.61752297
5471893: 1, 16.1827294932185: 1, 13.025934097931298: 1, 66.79264220410388: 1, 9.126608336486386:
1, 7.306733653315095: 1, 22.421063531888247: 1, 21.6430587879021: 1, 12.684072656271322: 1,
9.187033922883385: 1, 8.532850495435051: 1, 15.80689412110351: 1, 14.73219488518644: 1,
11.657065055009763: 1, 14.980359088892442: 1, 9.67623045966221: 1, 7.4230222374680315: 1,
10.764486384807011: 1, 16.80002309946631: 1, 23.58884864087994: 1, 9.157167099385314: 1, 8.48484758
4397237: 1, 7.859130890890085: 1, 75.33022159609527: 1, 9.50670433512087: 1, 32.58218445546026: 1,
15.475769850718706: 1, 13.368427330837902: 1, 12.287045088620477: 1, 11.500449497559687: 1,
8.43882708766188: 1, 24.757993614273115: 1, 13.49305496836048: 1, 79.12249663505497: 1,
9.095396457679438: 1, 18.946457058411877: 1, 16.770730044592085: 1, 19.824828784571682: 1,
19.302305328110258: 1, 12.008095180803993: 1, 18.908516312682227: 1, 25.057270333366088: 1,
17.08920149702452: 1, 8.966554820986435: 1, 7.468102367893596: 1, 84.22406334731534: 1,
10.588452506924177: 1, 13.603168447577701: 1, 10.361712857277043: 1, 16.84650530957157: 1,
22.07394028226156: 1, 17.160650605086943: 1, 9.175392780583095: 1, 8.79581581619877: 1,
13.97237199078606: 1, 9.162434926542534: 1, 10.622998060583038: 1, 9.940950350366753: 1,
8.180639427309275: 1, 9.700165529624545: 1, 17.60790955664652: 1, 10.012490422055096: 1,
7.841386398589354: 1, 10.609583004431457: 1, 7.962477657452626: 1, 11.852922210349966: 1,
18.131038986834227: 1, 19.636173990978694: 1, 14.626362472000723: 1, 18.660929547274407: 1,
9.77008452589459: 1, 16.850543987405523: 1, 10.086280527909114: 1, 8.557745223833864: 1,
10.280248700109807: 1, 9.200513539927707: 1, 12.121215332408166: 1, 19.96489197290947: 1,
12.66080367646552: 1, 10.907584009571938: 1, 13.855771641062066: 1, 22.971600069400523: 1,
15.853380536469901: 1, 28.72916689646561: 1, 10.480494551521568: 1, 11.298613762055087: 1,
9.583016795075272: 1, 43.17479123333226: 1, 10.471366484417883: 1, 7.407040903147282: 1,
15.20776292035901: 1, 10.590179200428835: 1, 37.671062085692164: 1, 20.244551676240718: 1,
17.207020772963723: 1, 43.43924991134998: 1, 10.456109929760308: 1, 13.985036211534485: 1,
12.418549204190672: 1, 12.561285784797759: 1, 23.379084633008215: 1, 8.805208089808962: 1,
55.3903888690033: 1, 6.988901957238915: 1, 38.85707656367861: 1, 21.967032360769394: 1,
8.518624980993954: 1, 11.975566657420806: 1, 22.239955004378942: 1, 11.701118012004805: 1,
9.1626781856828 8: 1, 30.661768944896775: 1, 13.083922541932626: 1, 15.179983804495707: 1,
10.019160416948589: 1, 9.676891057790732: 1, 11.548421517857777: 1, 22.161274590537175: 1,
26.23941040226884: 1, 15.005171142122952: 1, 8.284813251480196: 1, 113.87535370261571: 1,
11.206556422879302: 1, 14.13137712370403: 1, 7.9400485990930605: 1, 16.823333470313493: 1,
15.555312799141209: 1, 10.703316725032812: 1, 8.606950063145563: 1, 17.523067348003295: 1,
79.42315434734596: 1, 23.4636722697864: 1, 19.878480102720967: 1, 6.670311608214412: 1,
9.408251823752646: 1, 16.964798736418885: 1, 9.28310313512837: 1, 26.774516621666226: 1,
10.409587352286623: 1, 8.101498632179183: 1, 15.318944514525132: 1, 8.141619854590978: 1,
12.841847318317052: 1, 10.007640227867661: 1, 11.094129116930935: 1, 9.283936517966358: 1,
7.053145926064316: 1, 24.818720111324808: 1, 7.8689338262351285: 1, 7.163991066401039: 1,
31.470131619790845: 1, 7.395218332678771: 1, 16.639738779535865: 1, 9.044451661207292: 1,
18.81207458652719: 1, 9.859510220643703: 1, 12.5355848620269: 1, 7.014440183453612: 1,
42.72512344882062: 1, 25.56415050378402: 1, 8.195674448554719: 1, 19.00724077485719: 1,
6.941064348664306: 1, 18.839230425712053: 1, 9.50380041767042: 1, 12.04220260868196: 1,
17.437534393254143: 1, 20.734260539789965: 1, 15.369506459953996: 1, 10.730500803961498: 1,
13.42633214110919: 1, 17.845426274533335: 1, 10.28064375348283: 1, 26.576928499051604: 1,
9.139084383211134: 1, 24.48953443533036: 1, 7.273945754538354: 1, 13.256655754653577: 1,
21.53193452422352: 1, 13.08344441228932: 1, 14.793424624348207: 1, 14.522376027401371: 1,
7.389346007365692: 1, 18.309972341894937: 1, 14.695814114117757: 1, 10.507932051139178: 1,
26.253962587168118: 1, 9.399611352703149: 1, 14.635114775799371: 1, 30.783475674809605: 1,
27.119440623647137: 1, 9.247297897450869: 1, 10.819520203508041: 1, 40.5090273276182: 1,
11.692616087127833: 1, 12.676028224814646: 1, 8.86549456165601: 1, 8.597951977103389: 1,
9.549326617822922: 1, 19.220228814831618: 1, 15.771463791738404: 1, 13.479730406067125: 1,
8.132120461072851: 1, 16.615136601735912: 1, 16.975095052260237: 1, 11.826793747163812: 1,
13.572535055641382: 1, 8.173503446223096: 1, 7.291627320943645: 1, 12.389342215887755: 1,
8.953001644183306: 1, 7.767916084213917: 1, 9.118748430849047: 1, 8.311932916509141: 1,
17.35479760111369: 1, 20.02933560507157: 1, 8.301620086912926: 1, 9.388739315253643: 1,
11.824035401696866: 1, 13.311766462297175: 1, 14.699346567332052: 1, 29.25195438353876: 1,
12.783449479226455: 1, 9.125640681500506: 1, 8.408595010400214: 1, 10.33704038521608: 1, 6.64749711
3443431: 1, 21.07549555978841: 1, 8.883344452175995: 1, 10.370773241049735: 1, 9.1071794370515: 1,
10.735082098580056: 1, 64.19759197481285: 1, 30.115958321429076: 1, 12.583490148938596: 1,
13.587298031691127: 1, 13.330364818521254: 1, 115.31939347018033: 1, 15.548412709995507: 1,
9.094329170384677: 1, 17.64915196653709: 1, 7.889562512826113: 1, 22.883682988632106: 1,
18.902205598225343: 1, 9.108739993871861: 1, 16.8190090298947: 1, 42.66744717826792: 1,
9.008767485494984: 1, 6.948489927761378: 1, 12.779468269069978: 1, 21.180199684026345: 1,
31.687514708735364: 1, 6.635268661679496: 1, 14.573259599376463: 1, 10.030927264066086: 1,
9.415675886142854: 1, 17.261183920596732: 1, 9.509084604894133: 1, 10.553910850083648: 1,
7.3905906346600805: 1, 23.70061877617596: 1, 11.157967717651008: 1, 10.988194214643565: 1,
10.96701690963019: 1, 16.261692724009187: 1, 49.22577697618441: 1, 32.76310050672001: 1,
15.13987653877897: 1, 18.126025896049796: 1, 9.819562530986037: 1, 11.29510877725185: 1,
14.924745847109477: 1, 22.631839072129026: 1, 21.267779702874396: 1, 10.413707142399785: 1,
14.985776615060356: 1, 9.952208041458935: 1, 43.2123824619801: 1, 17.86528123709011: 1,
16.676809881851774: 1, 8.730296693136289: 1, 9.095116235961715: 1, 12.701030345512871: 1,
8.600609527860243: 1, 12.671217843253237: 1, 17.033110912896277: 1, 8.805067410029647: 1,
12.569131235026681: 1, 8.068637100476165: 1, 7.954108831921957: 1, 6.800391946597824: 1, 18.8712906
35405387: 1, 21.685878708188117: 1, 16.692983620094594: 1, 9.735862433016008: 1,
34.363338363760704: 1, 17.930059991686626: 1, 13.475085776590195: 1, 13.661752351613218: 1,
64.1269785257612: 1, 23.91339568083726: 1, 12.376976089972848: 1, 13.979732484634193: 1,
9.400801397968923: 1, 26.976614127162836: 1, 22.850494827712858: 1, 9.305558252409593: 1,
8.21620188309706: 1, 7.608420017968752: 1, 9.826093733023336: 1, 35.38739747467724: 1,
18.031748139728855: 1, 9.778097648715487: 1, 14.856798878860085: 1, 7.683444925702071: 1,
10.771602875905849: 1, 10.66246607914523: 1, 10.504497416251436: 1, 8.422717844620639: 1,
11.748132087115284: 1, 10.981640080323332: 1, 19.47225137650873: 1, 10.941755683562551: 1,
15.197526499466628: 1, 11.84433393580306: 1, 28.512987345412828: 1, 20.198929744308483: 1,
11.221466417200059: 1, 12.916789664100369: 1, 7.436959205921239: 1, 9.87453753015478: 1,
20.228540292447242: 1, 16.038350151333233: 1, 9.659998674697986: 1, 8.939184645382179: 1,
11.429618381572757: 1, 16.063502901876053: 1, 19.08019679096562: 1, 22.893815374590137: 1,
12.17644306902319: 1, 9.678504397049386: 1, 30.331640377887247: 1, 13.515877558920888: 1,
8.11132567625715: 1, 20.969247826573863: 1, 22.78399003519042: 1, 21.529585480259904: 1,
8.898213220716762: 1, 9.077507134977806: 1, 19.35823813132714: 1, 8.22203802500817: 1,
16.16364023536676: 1, 13.171425373507793: 1, 30.61998138145029: 1, 13.000974515057418: 1,
9.754577396196927: 1, 9.156511812385771: 1, 7.663357544866314: 1, 39.119028631698626: 1,
22.366156885982157: 1, 12.720154252601159: 1, 11.320786968662738: 1, 10.483187596610597: 1,
12.0652223861054: 1, 31.079258828760292: 1, 14.331840273519031: 1, 12.47549177787039: 1,
9.765284409025558: 1, 14.936684417724207: 1, 27.05831909195021: 1, 10.17403458572373: 1,
12.144422275172252: 1, 8.038008920293453: 1, 23.802195149480333: 1, 6.500658798438266: 1,
8.182954573501734: 1, 14.32581236120447: 1, 16.529965183910832: 1, 10.760202163739978: 1,
12.176927336379334: 1, 14.39086240801816: 1, 15.147000665561773: 1, 9.389665834662184: 1,
21.825149808950798: 1, 109.84105849580487: 1, 11.604220690851362: 1, 75.82442643539248: 1,
12.178377732610235: 1, 24.686119054944847: 1, 7.312611646312732: 1, 7.9167037206301965: 1,
17.05156309166134: 1, 10.00459631730432: 1, 19.940384581379206: 1, 11.839182401385395: 1,
16.25531353482121: 1, 8.78962311260545: 1, 11.38370738024655: 1, 27.691338447051606: 1,
8.527741174796024: 1, 12.486005058303148: 1, 15.94985606243337: 1, 11.170325189258266: 1,
42.97745815611424: 1, 8.256816604264927: 1, 8.884529339197702: 1, 12.99903802017997: 1,
10.53660238366489: 1, 8.018019565936342: 1, 17.96587161326828: 1, 8.88512958786081: 1,
11.071327774618647: 1, 18.02130583793352: 1, 17.20957391106135: 1, 25.44158001183385: 1, 8.73991155
0638364: 1, 13.123991019770733: 1, 13.011162094102884: 1, 43.75407786326983: 1,
26.621030719228692: 1, 22.279010999752934: 1, 9.004958232537277: 1, 9.029526817151428: 1,
8.326514544989923: 1, 7.527356336249782: 1, 13.215754121251377: 1, 19.20193353178702: 1,
7.987585956594829: 1, 20.30085565831317: 1, 11.205011718575232: 1, 18.21811174573615: 1,
9.459310103175303: 1, 14.05323959258281: 1, 16.724955632121777: 1, 10.665219519910204: 1,
7.79154137336786: 1, 10.112666166275929: 1, 14.33724001279107: 1, 14.95171471825736: 1,
9.478585593153563: 1, 6.760680642323864: 1, 17.056001359938154: 1, 19.242454641163715: 1,
14.51802499798466: 1, 10.402426112387706: 1, 15.157163903395668: 1, 7.571422231150543: 1,
28.072611277505764: 1, 11.188825662231821: 1, 10.75711909756022: 1, 22.13930984072048: 1,
7.8924838895730876: 1, 7.810506137535854: 1, 9.242136793489063: 1, 20.221755290018347: 1,
17.856258362911593: 1, 18.11536572360634: 1, 9.526502601534048: 1, 10.70591683022075: 1, 18.4715512
73557765: 1, 12.485539844957675: 1, 17.53152291522194: 1, 8.30365682962203: 1, 6.559395302524439:
1, 21.88061744551113: 1, 8.954345294153386: 1, 8.85512337197573: 1, 19.68999906687921: 1,
8.887211096287524: 1, 6.376326985192849: 1, 27.52774516617576: 1, 30.55917301227027: 1,
13.021638177612786: 1, 11.795417727369294: 1, 10.116565683565918: 1, 11.663834826833295: 1,
9.023355266045359: 1, 9.348134473262963: 1, 7.4008324738044164: 1, 22.95251566414326: 1,
18.72802386526893: 1, 8.690408190125574: 1, 17.02914963021196: 1, 31.63369611357621: 1,
10.622245759621212: 1, 11.731911233350113: 1, 14.37775722744764: 1, 9.49415219499926: 1,
14.449885094377876: 1, 17.492946434583768: 1, 16.391800341638167: 1, 7.640030569434938: 1,
10.958229192354127: 1, 17.566330760710073: 1, 20.410635132832553: 1, 23.717902375360477: 1,
6.881925245953443: 1, 17.252169992008113: 1, 11.727642014611: 1, 8.326525640231973: 1,
13.997824130579037: 1, 10.736871546178042: 1, 15.05679144389056: 1, 9.158377802075153: 1,
11.153991447748469: 1, 9.593305986615565: 1, 11.449211055531508: 1, 19.524036485124334: 1,
14.575468773474029: 1, 9.313840240142149: 1, 24.15946340112087: 1, 7.780775795728022: 1, 13.7851002
95911958: 1, 9.235867754449258: 1, 33.214426218030134: 1, 16.982468273214963: 1,
8.822225886047146: 1, 9.741633866964323: 1, 9.03995710539542: 1, 12.661934294690683: 1,
15.50498429783236: 1, 25.29519207653878: 1, 11.270671397636862: 1, 8.626920293576337: 1,
8.626266023896079: 1, 12.512555402258823: 1, 11.302225791065375: 1, 10.696810581953374: 1,
17.50284686999781: 1, 9.170648591267835: 1, 16.477300957571007: 1, 19.94025458695525: 1,
10.550090738037804: 1, 11.262309425721412: 1, 26.448423217370998: 1, 12.543218113726132: 1,
9.247271711663746: 1, 17.839378896143444: 1, 12.253487292892546: 1, 7.956018017413716: 1,
35.395022991208414: 1, 18.032401210305395: 1, 9.646091304575842: 1, 10.850554877431554: 1,
9.013818700131909: 1, 14.193286103569124: 1, 27.68113574173996: 1, 9.380692350095307: 1,
10.503340419952469: 1, 18.95136315524252: 1, 8.39349721273452: 1, 7.343429040831411: 1,
10.051080717490295: 1, 19.111770047924907: 1, 9.6775142456717: 1, 12.376557892983966: 1,
15.106350163033751: 1, 16.30237347123327: 1, 15.35498095816685: 1, 9.261521734563702: 1, 8.6509290C
083573: 1, 21.997446078265725: 1, 16.392354921109792: 1, 11.242996455577515: 1, 14.79494900003715:
1, 21.46036449267118: 1, 13.41945059368022: 1, 10.553603392199745: 1, 18.483312653433778: 1,
11.98508227206097: 1, 20.967681889343154: 1, 14.395873307454911: 1, 18.100815055849207: 1,
11.981629926764167: 1, 15.071902797108466: 1, 12.957846381759868: 1, 9.541206473706728: 1,
8.31881775499709: 1, 16.009250625790852: 1, 9.65852388381388: 1, 6.921828163993147: 1,
7.703982305493924: 1, 21.19098340106639: 1, 21.498115945384605: 1, 11.878214565700766: 1,
12.045771909454777: 1, 31.652996128589933: 1, 21.03364185165118: 1, 35.710594094319944: 1,
13.981989365164953: 1, 21.663011535624506: 1, 17.0779099817913: 1, 16.060194965282193: 1,
12.61682322517162: 1, 9.207436588637327: 1, 12.345034301633648: 1, 7.846371894372189: 1,
22.950023908655155: 1, 10.637187126588682: 1, 9.824371526096709: 1, 19.224607733626062: 1,
21.014893179183645: 1, 8.96470029785483: 1, 31.965261946014788: 1, 12.996868849590573: 1,
17.619609622289644: 1, 9.089436481053076: 1, 10.862145578148178: 1, 40.87101703477871: 1,
23.35515216010573: 1, 23.584356011966193: 1, 8.581634377512204: 1, 9.236116972875477: 1,
14.682928392905856: 1, 8.283718441219373: 1, 19.370385445173323: 1, 15.07887597377207: 1,
8.841486320344561: 1, 15.729528517100665: 1, 19.265520069640676: 1, 10.506339741399508: 1,
13.165240992466927: 1, 16.422579087883072: 1, 11.710821628866446: 1, 19.652021264045466: 1,
24.776724105815717: 1, 16.287105041207536: 1, 7.895213916356759: 1, 14.139884094654143: 1,
21.404772545776662: 1, 16.64842718839343: 1, 12.518331781478054: 1, 10.154410125665315: 1,
9.379335559396376: 1, 9.247434359885634: 1, 12.518416705456465: 1, 10.552118945972376: 1,
9.631953610997021: 1, 42.56901844659874: 1, 8.384997295609786: 1, 14.598759441679265: 1,
10.373568116031324: 1, 17.322104366287636: 1, 12.955075355652456: 1, 12.353778372962427: 1,
43.719532850098496: 1, 26.116566533559265: 1, 9.197491836732006: 1, 113.99791188784793: 1,
7.920062772233217: 1, 14.206998095183028: 1, 11.234597115444677: 1, 14.160615245674412: 1,
18.979768625815115: 1, 10.498734984877064: 1, 8.908977291367114: 1, 10.401119797973214: 1,
7.716443203816759: 1, 14.15833325220506: 1, 12.47803380958493: 1, 7.562386931234502: 1,
9.953029567419485: 1, 10.657433646528341: 1, 16.414767141390556: 1, 10.364730368142482: 1,
6.770725022993425: 1, 8.724218647211433: 1, 11.523226905961574: 1, 13.034286552584433: 1,
19.779458666259035: 1, 13.58830967089182: 1, 8.222710457576826: 1, 10.952673944138246: 1,
15.644178272784833: 1, 28.909710849639126: 1, 8.081588041174939: 1, 11.144153792904653: 1,
7.385704402177817: 1, 11.913459016333466: 1, 20.83461253533959: 1, 8.71222013669837: 1,
17.414575761413715: 1, 45.56364411686455: 1, 9.15721674933161: 1, 17.479359294223823: 1,
9.596024517847527: 1, 12.539170235537059: 1, 29.01167679401938: 1, 8.749171492361222: 1,
12.248951364018033: 1, 10.550354423010612: 1, 10.924048135188862: 1, 7.847455352090972: 1,
18.889814299064685: 1, 8.870791474516148: 1, 19.149545987222293: 1, 21.58154448889842: 1,
10.374691196417077: 1, 9.013681054435427: 1, 10.020493049294467: 1, 12.159585473711429: 1,
13.251955774536643: 1, 18.994039713545515: 1, 47.234519860757025: 1, 30.55698967633741: 1,
14.660392604559572: 1, 13.048403771349449: 1, 14.111583517108832: 1, 9.10956834632528: 1,
12.407995589756561: 1, 10.119060841321083: 1, 9.941754095221592: 1, 7.738744693732071: 1,
12.884991594068474: 1, 7.239910490556691: 1, 13.736523252948968: 1, 14.98375309675001: 1,
22.011496670464865: 1, 14.502909420687217: 1, 21.490119668759714: 1, 10.36145496516817: 1,
31.41107065003196: 1, 14.896172393840509: 1, 12.560789922416408: 1, 35.11929445220381: 1,
10.626340384873624: 1, 11.226789896256003: 1, 11.131996658950202: 1, 14.90460442816594: 1,
9.753892784366807: 1, 6.714985896257584: 1, 8.32754070200941: 1, 9.260145576559278: 1,
11.63302999268435: 1, 15.503816162808867: 1, 15.00990851452876: 1, 9.35032100026017: 1,
11.811954818797966: 1, 15.449521208065324: 1, 10.48880338263039: 1, 17.692070129927487: 1,
24.32865638218259: 1, 8.606822874631797: 1, 7.55517961967524: 1, 7.623305339623542: 1,
16.223344020475782: 1, 17.296541135826462: 1, 9.785699020860484: 1, 12.475711889138209: 1,
7.017596211989559: 1, 10.34251219153572: 1, 33.951375981747574: 1, 16.712169216080632: 1,
10.284372291487491: 1, 13.391758319482708: 1, 7.234706059162555: 1, 18.54095823892504: 1,
16.30710624234967: 1, 13.857197083873924: 1, 7.1458749658156915: 1, 8.595922990404201: 1,
10.872615602580082: 1, 19.191639219808362: 1, 20.551743561651723: 1, 11.067912582274166: 1,
7.940592845063295: 1})
```

In [52]:

```
# 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: 1.2211447627167331
For values of alpha =  0.0001 The log loss is: 1.1867348905731983
For values of alpha =  0.001 The log loss is: 1.5067579304482075
For values of alpha =  0.01 The log loss is: 2.0703862543870257
For values of alpha =  0.1 The log loss is: 2.145446094828133
For values of alpha =  1 The log loss is: 2.122603698589289
```



```
For values of best alpha =  0.0001 The train log loss is: 0.8454231198510006
For values of best alpha =  0.0001 The cross validation log loss is: 1.1867348905731983
For values of best alpha =  0.0001 The test log loss is: 1.170156720204002
```

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

Ans. Yes, it seems like!

In [53]:

```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(max_features=1000, 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 [54]:

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

```
94.3 % of word of test data appeared in train data
94.5 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [55]:

```
#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 [56]:

```
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

In [58]:

```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = TfidfVectorizer(max_features=1000, 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 [59]:

```
# 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 [60]:

```
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, 3180)
(number of data points * number of features) in test data =  (665, 3180)
(number of data points * number of features) in cross validation data = (532, 3180)
```

In [61]:

```
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

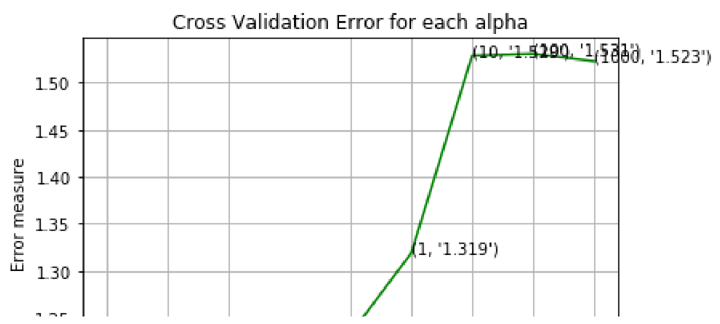
4.1.1.1. Hyper parameter tuning

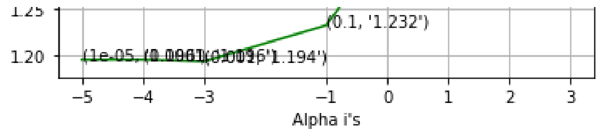
In [62]:

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

```python
# 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 : 1.1959408641140366
for alpha = 0.0001
Log Loss : 1.1959838265062763
for alpha = 0.001
Log Loss : 1.1940322618241044
for alpha = 0.1
Log Loss : 1.232388690116469
for alpha = 1
Log Loss : 1.3187856040631496
for alpha = 10
Log Loss : 1.5290055919622612
for alpha = 100
Log Loss : 1.53096964554861
for alpha = 1000
Log Loss : 1.522936899098048
```





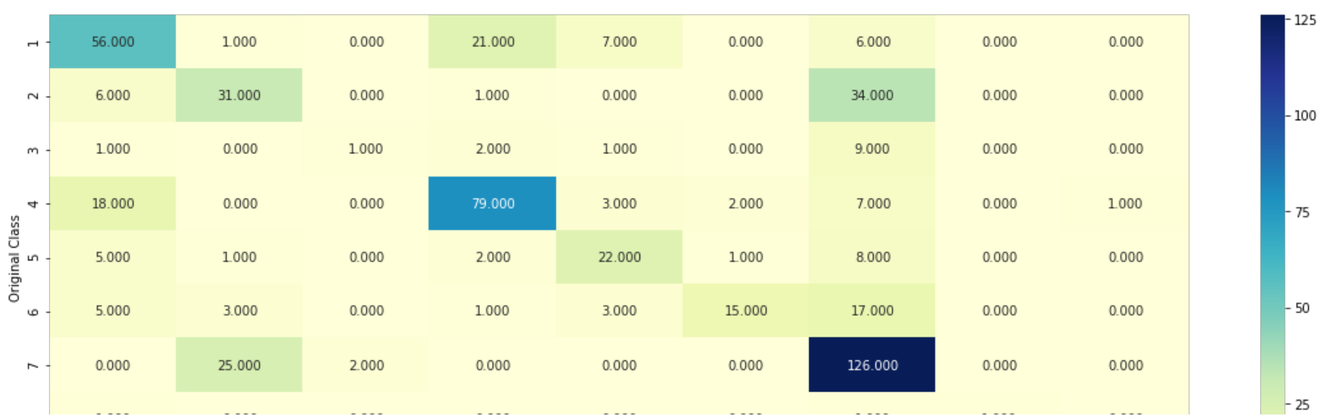
```
For values of best alpha =  0.001 The train log loss is: 0.546484013546373
For values of best alpha =  0.001 The cross validation log loss is: 1.1940322618241044
For values of best alpha =  0.001 The test log loss is: 1.2457340198985563
```

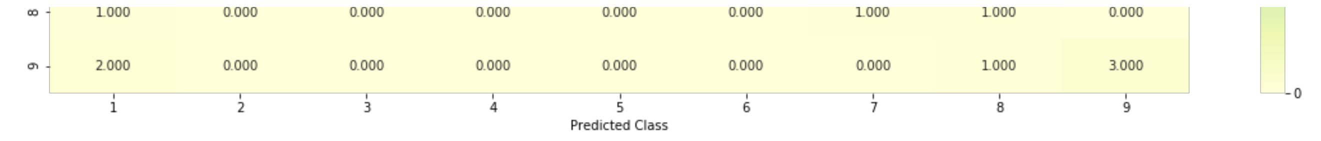
4.1.1.2. Testing the model with best hyper paramters

In [63]:

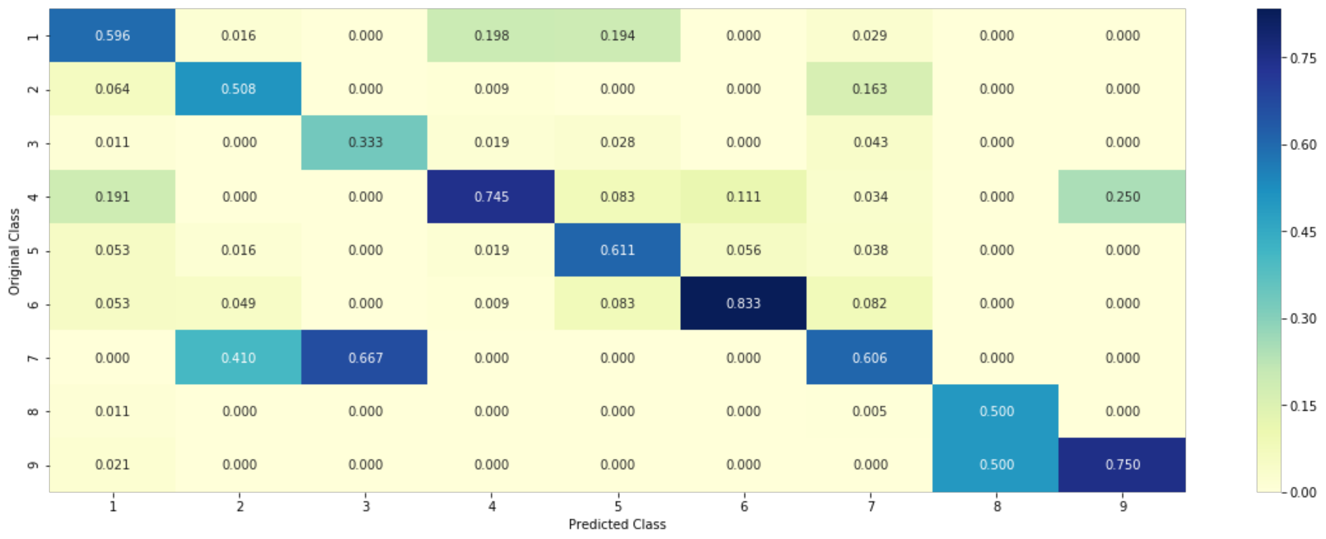
```python
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# 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 : 1.1940322618241044
Number of missclassified point : 0.37218045112781956
-------------------- Confusion matrix --------------------
```

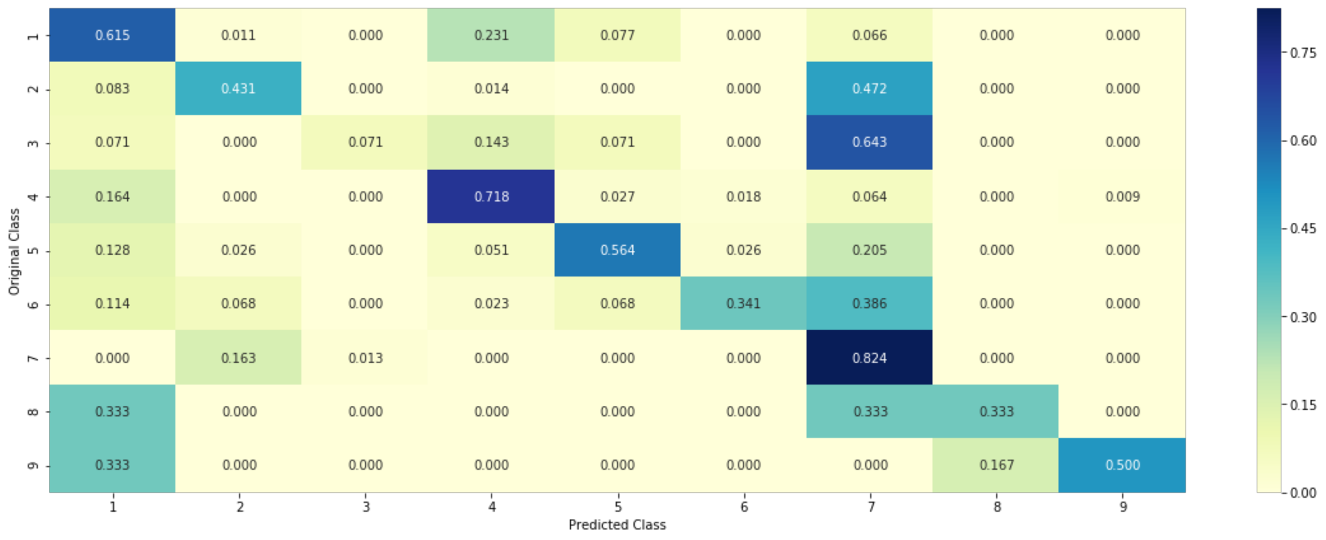




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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



4.1.1.3. Feature Importance, Correctly classified point

In [64]:

```
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 : 7
Predicted Class Probabilities: [[0.061  0.0392 0.0105 0.0692 0.0339 0.0298 0.7523 0.0027 0.0015]]
Actual Class : 7
```

```
--------------------------------------------------
15 Text feature [activation] present in test data point [True]
19 Text feature [activated] present in test data point [True]
21 Text feature [kinase] present in test data point [True]
22 Text feature [downstream] present in test data point [True]
23 Text feature [cells] present in test data point [True]
24 Text feature [expressing] present in test data point [True]
25 Text feature [inhibitor] present in test data point [True]
26 Text feature [signaling] present in test data point [True]
27 Text feature [independent] present in test data point [True]
28 Text feature [factor] present in test data point [True]
29 Text feature [also] present in test data point [True]
30 Text feature [contrast] present in test data point [True]
32 Text feature [growth] present in test data point [True]
33 Text feature [however] present in test data point [True]
34 Text feature [addition] present in test data point [True]
35 Text feature [10] present in test data point [True]
36 Text feature [treatment] present in test data point [True]
37 Text feature [cell] present in test data point [True]
38 Text feature [constitutive] present in test data point [True]
39 Text feature [compared] present in test data point [True]
40 Text feature [higher] present in test data point [True]
41 Text feature [shown] present in test data point [True]
42 Text feature [activating] present in test data point [True]
43 Text feature [similar] present in test data point [True]
44 Text feature [inhibitors] present in test data point [True]
45 Text feature [previously] present in test data point [True]
46 Text feature [presence] present in test data point [True]
47 Text feature [showed] present in test data point [True]
48 Text feature [found] present in test data point [True]
49 Text feature [potential] present in test data point [True]
50 Text feature [well] present in test data point [True]
51 Text feature [absence] present in test data point [True]
52 Text feature [treated] present in test data point [True]
53 Text feature [sensitive] present in test data point [True]
54 Text feature [mutations] present in test data point [True]
55 Text feature [oncogenic] present in test data point [True]
56 Text feature [increased] present in test data point [True]
57 Text feature [may] present in test data point [True]
59 Text feature [without] present in test data point [True]
60 Text feature [recently] present in test data point [True]
61 Text feature [suggest] present in test data point [True]
62 Text feature [activate] present in test data point [True]
63 Text feature [tyrosine] present in test data point [True]
64 Text feature [inhibition] present in test data point [True]
65 Text feature [3b] present in test data point [True]
67 Text feature [figure] present in test data point [True]
68 Text feature [total] present in test data point [True]
69 Text feature [constitutively] present in test data point [True]
70 Text feature [proliferation] present in test data point [True]
71 Text feature [observed] present in test data point [True]
72 Text feature [survival] present in test data point [True]
73 Text feature [3a] present in test data point [True]
75 Text feature [serum] present in test data point [True]
77 Text feature [mechanism] present in test data point [True]
78 Text feature [interestingly] present in test data point [True]
79 Text feature [reported] present in test data point [True]
80 Text feature [therapeutic] present in test data point [True]
81 Text feature [respectively] present in test data point [True]
82 Text feature [two] present in test data point [True]
83 Text feature [mutation] present in test data point [True]
84 Text feature [small] present in test data point [True]
85 Text feature [described] present in test data point [True]
86 Text feature [either] present in test data point [True]
87 Text feature [receptor] present in test data point [True]
88 Text feature [identified] present in test data point [True]
89 Text feature [although] present in test data point [True]
91 Text feature [15] present in test data point [True]
93 Text feature [consistent] present in test data point [True]
94 Text feature [results] present in test data point [True]
95 Text feature [20] present in test data point [True]
96 Text feature [pathway] present in test data point [True]
97 Text feature [using] present in test data point [True]
98 Text feature [including] present in test data point [True]
99 Text feature [mutant] present in test data point [True]
Out of the top  100  features  74 are present in query point
```

4.2. K Nearest Neighbour Classification

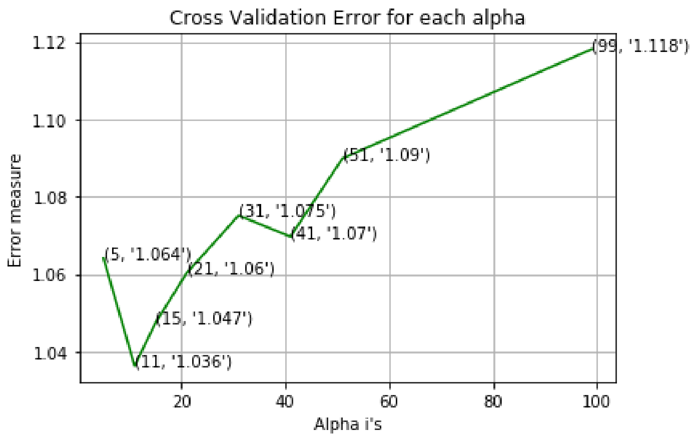
4.2.1. Hyper parameter tuning

In [66]:

```
# 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



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 : 1.0640577743763733
for alpha = 11
Log Loss : 1.0362556069136786
for alpha = 15
Log Loss : 1.0473808883890872
for alpha = 21
Log Loss : 1.0602150109340975
for alpha = 31
Log Loss : 1.075125753031025
for alpha = 41
Log Loss : 1.0696597564796773
for alpha = 51
Log Loss : 1.0898581199451145
for alpha = 99
Log Loss : 1.118080873632053
```



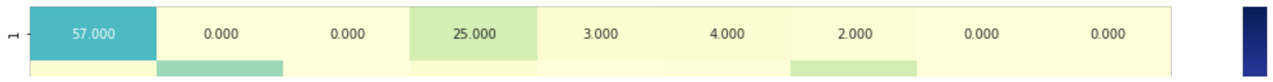
```
For values of best alpha =  11 The train log loss is: 0.6397814598036096
For values of best alpha =  11 The cross validation log loss is: 1.0362556069136786
For values of best alpha =  11 The test log loss is: 1.071611753989241
```

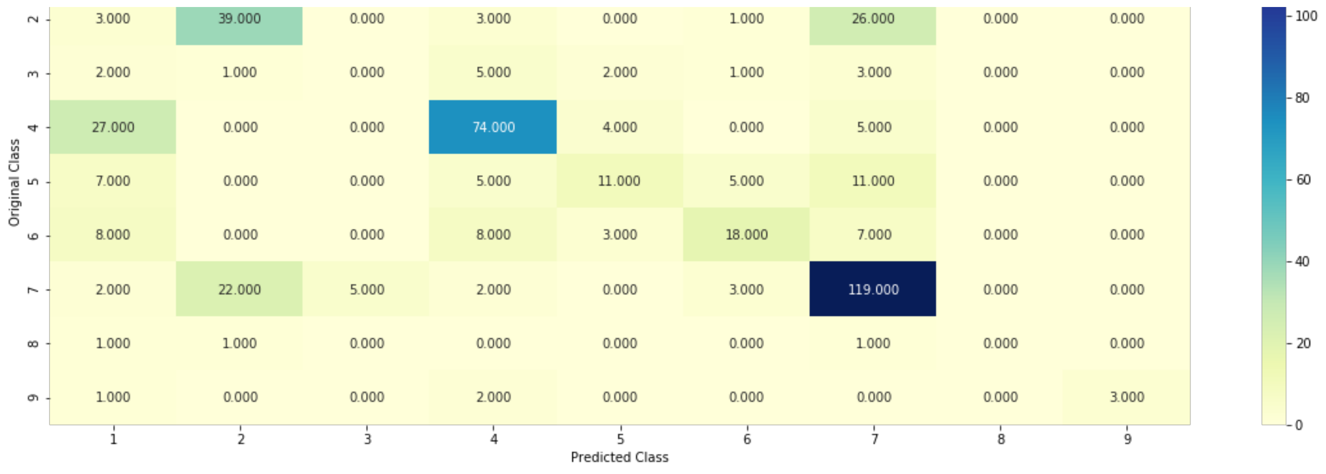
4.2.2. Testing the model with best hyper paramters

In [102]:

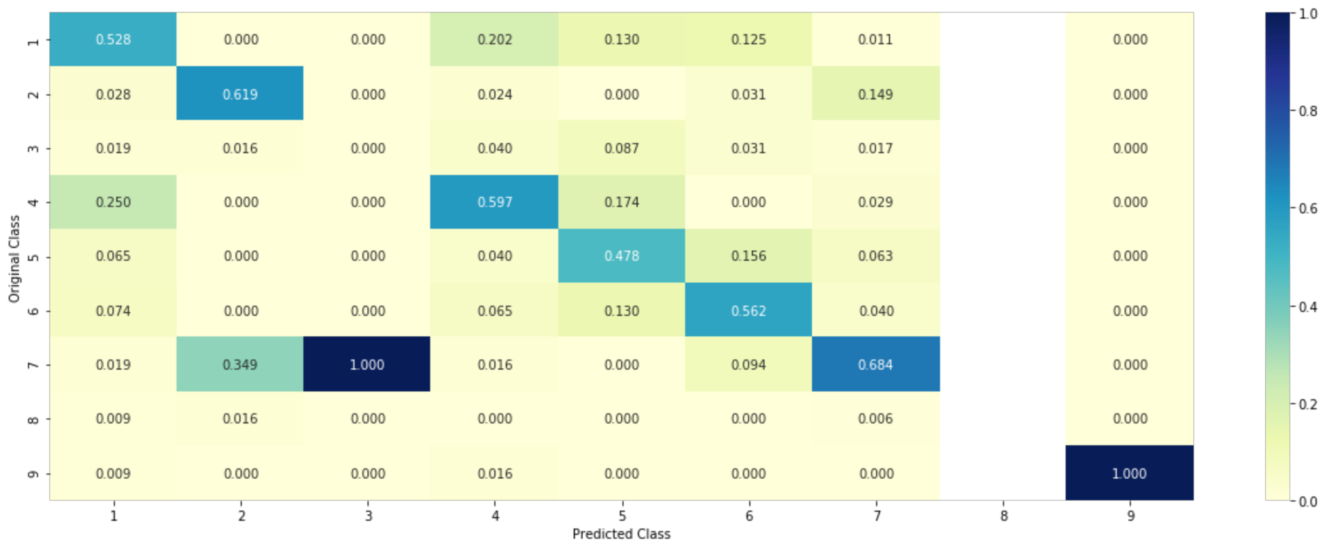
```
# find more about KNeighborsClassifier() here http://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', 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-ne
ighbors-geometric-intuition-with-a-toy-example-1/
#--------------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y, clf)
```

```
Log loss : 1.1002748803755749
Number of mis-classified points : 0.3966165413533835
-------------------- Confusion matrix --------------------
```

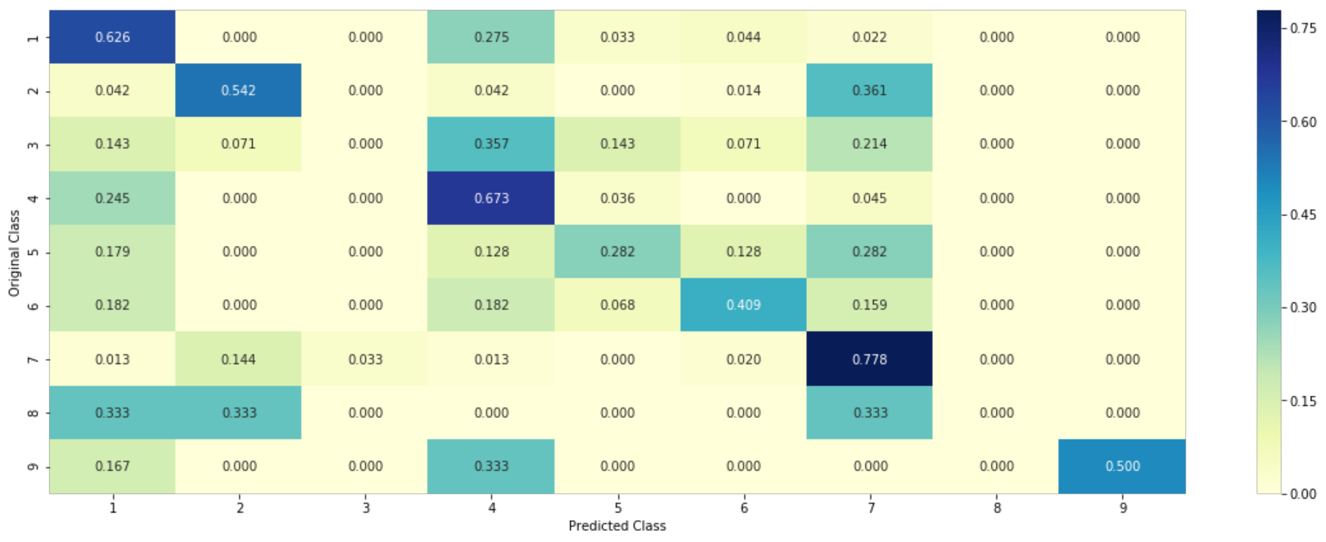




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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



4.2.3.Sample Query point -1

In [67]:

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

```
test_point_index = 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 : 4
Actual Class : 7
The  11  nearest neighbours of the test points belongs to classes [7 7 7 7 7 7 7 7 7 7 7]
Fequency of nearest points : Counter({7: 11})
```

4.2.4. Sample Query Point-2

In [68]:

```
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 : 1
Actual Class : 1
the k value for knn is 11 and the nearest neighbours of the test points belongs to classes [1 1 6
1 1 1 7 1 1 6 1]
Fequency of nearest points : Counter({1: 8, 6: 2, 7: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

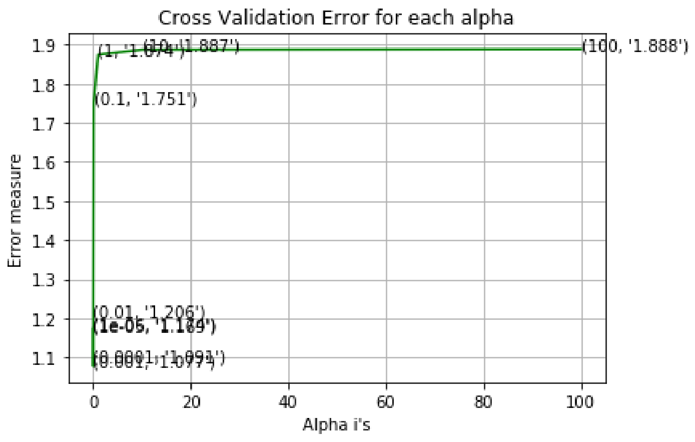
4.3.1.1. Hyper paramter tuning

In [70]:

```
# 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/
#------------------------------
```

```python
# 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


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 : 1.173513894387702
for alpha = 1e-05
Log Loss : 1.1690771438074514
for alpha = 0.0001
Log Loss : 1.090904957299963
for alpha = 0.001
Log Loss : 1.0772379261715657
for alpha = 0.01
Log Loss : 1.2060181862008477
for alpha = 0.1
Log Loss : 1.7509536957546354
for alpha = 1
Log Loss : 1.874443513989209
for alpha = 10
Log Loss : 1.8867902910414287
for alpha = 100
Log Loss : 1.8880661980967286
```



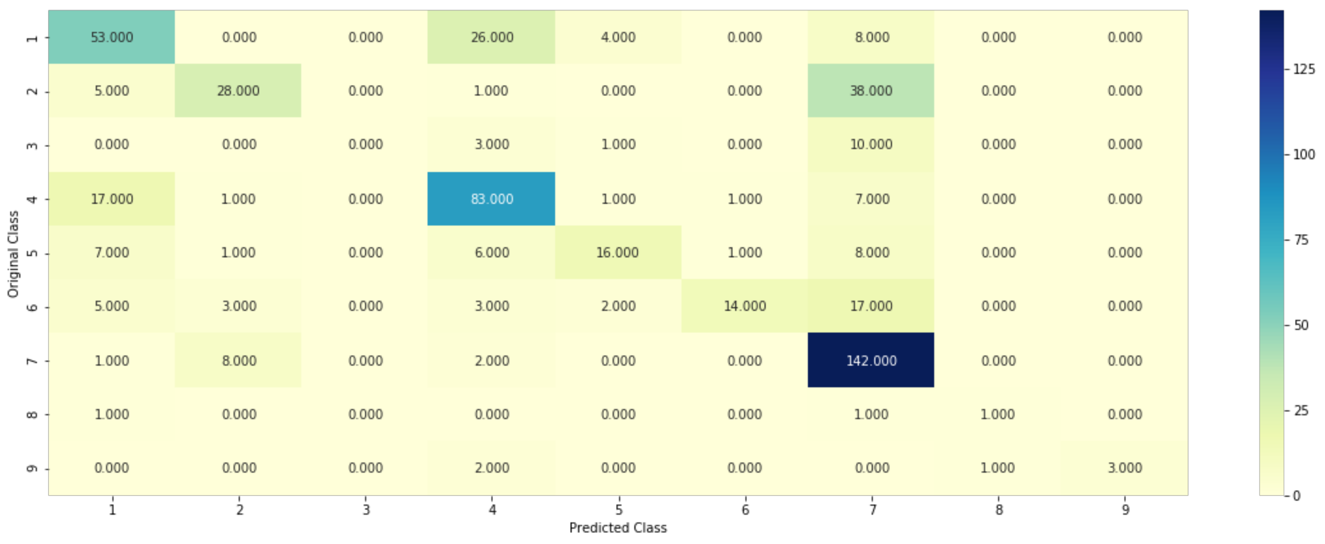
```
For values of best alpha =  0.001 The train log loss is: 0.726332218972466
For values of best alpha =  0.001 The cross validation log loss is: 1.0772379261715657
For values of best alpha =  0.001 The test log loss is: 1.0928089578037763
```

4.3.1.2. Testing the model with best hyper paramters

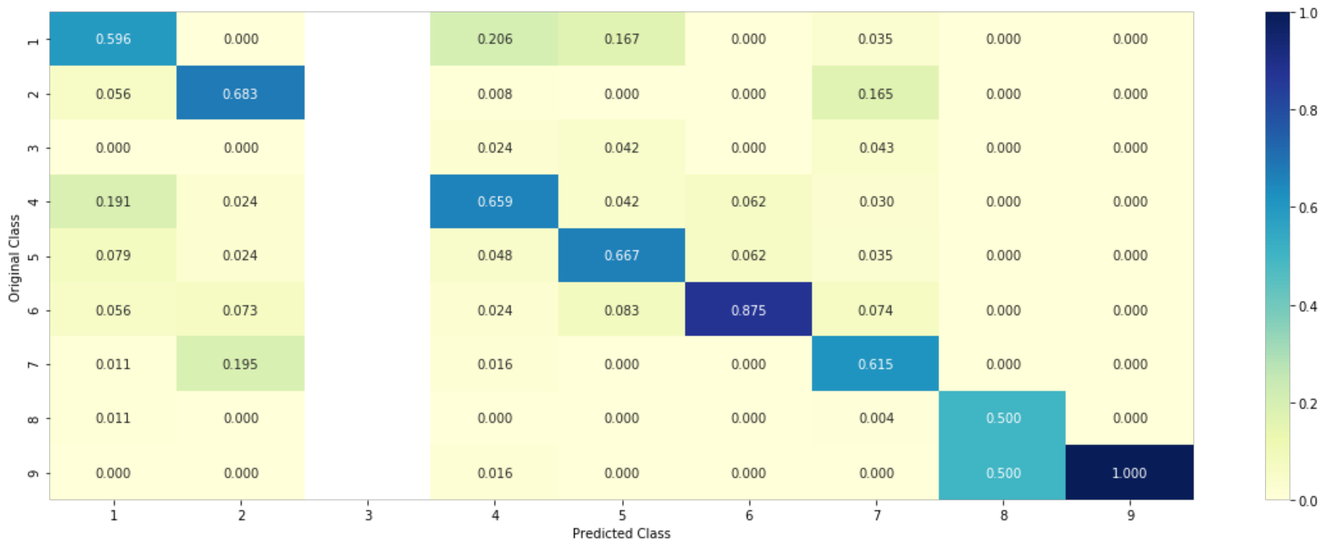
In [71]:

```
# 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/
#------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

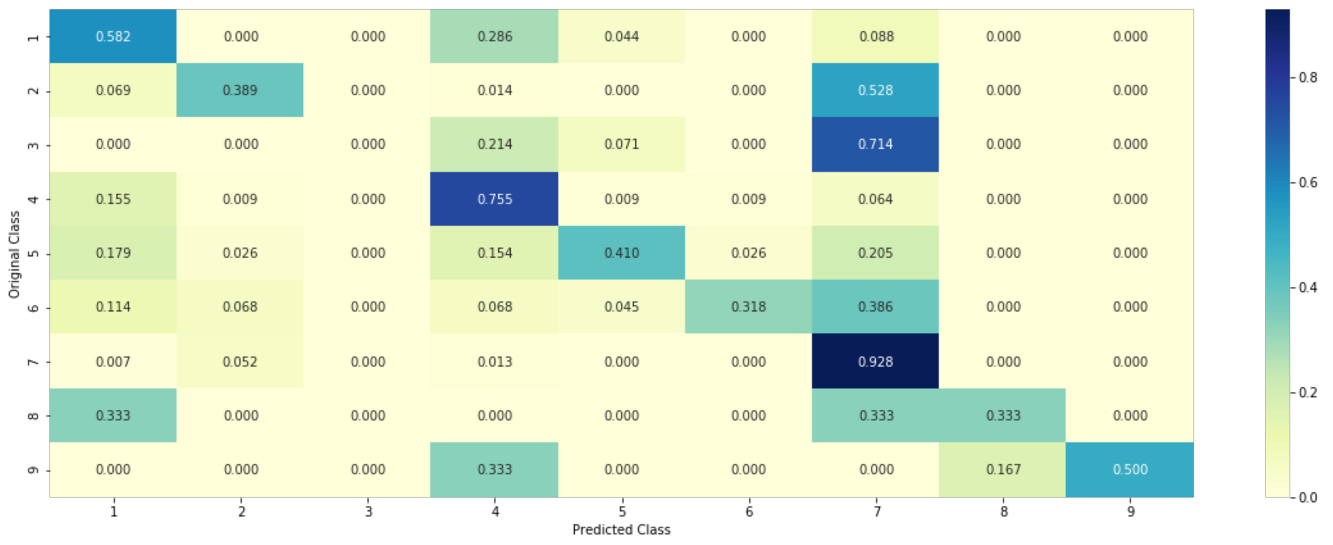
```
Log loss : 1.0772379261715657
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```



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



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



4.3.1.3. Feature Importance

In [72]:

```python
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 [73]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[1.620e-02 8.700e-03 7.700e-03 2.780e-02 1.430e-02 8.500e-03 9.151
e-01
  1.400e-03 4.000e-04]]
Actual Class : 7
--------------------------------------------------
20 Text feature [constitutive] present in test data point [True]
24 Text feature [activation] present in test data point [True]
25 Text feature [activated] present in test data point [True]
27 Text feature [transformed] present in test data point [True]
30 Text feature [oncogenic] present in test data point [True]
31 Text feature [downstream] present in test data point [True]
36 Text feature [ligand] present in test data point [True]
38 Text feature [egf] present in test data point [True]
51 Text feature [transformation] present in test data point [True]
52 Text feature [transforming] present in test data point [True]
57 Text feature [serum] present in test data point [True]
60 Text feature [factor] present in test data point [True]
64 Text feature [extracellular] present in test data point [True]
68 Text feature [signaling] present in test data point [True]
69 Text feature [activating] present in test data point [True]
70 Text feature [expressing] present in test data point [True]
74 Text feature [overexpression] present in test data point [True]
78 Text feature [activate] present in test data point [True]
79 Text feature [il] present in test data point [True]
82 Text feature [epithelial] present in test data point [True]
86 Text feature [constitutively] present in test data point [True]
88 Text feature [express] present in test data point [True]
93 Text feature [tyrosine] present in test data point [True]
96 Text feature [lung] present in test data point [True]
97 Text feature [ras] present in test data point [True]
106 Text feature [inhibitor] present in test data point [True]
107 Text feature [receptors] present in test data point [True]
109 Text feature [leukemia] present in test data point [True]
110 Text feature [days] present in test data point [True]
111 Text feature [egfr] present in test data point [True]
112 Text feature [promote] present in test data point [True]
115 Text feature [colony] present in test data point [True]
120 Text feature [adenocarcinoma] present in test data point [True]
122 Text feature [absence] present in test data point [True]
123 Text feature [3b] present in test data point [True]
124 Text feature [nras] present in test data point [True]
127 Text feature [cultured] present in test data point [True]
129 Text feature [kinase] present in test data point [True]
131 Text feature [growth] present in test data point [True]
133 Text feature [positive] present in test data point [True]
134 Text feature [elevated] present in test data point [True]
136 Text feature [survival] present in test data point [True]
144 Text feature [bone] present in test data point [True]
150 Text feature [carcinoma] present in test data point [True]
152 Text feature [stimulation] present in test data point [True]
157 Text feature [2a] present in test data point [True]
158 Text feature [cells] present in test data point [True]
164 Text feature [presence] present in test data point [True]
167 Text feature [independent] present in test data point [True]
```

```
186 Text feature [somatic] present in test data point [True]
189 Text feature [day] present in test data point [True]
197 Text feature [codon] present in test data point [True]
219 Text feature [inhibited] present in test data point [True]
228 Text feature [fold] present in test data point [True]
253 Text feature [her2] present in test data point [True]
254 Text feature [per] present in test data point [True]
286 Text feature [effective] present in test data point [True]
297 Text feature [akt1] present in test data point [True]
325 Text feature [kras] present in test data point [True]
378 Text feature [proliferation] present in test data point [True]
381 Text feature [pi3k] present in test data point [True]
409 Text feature [factors] present in test data point [True]
411 Text feature [tissue] present in test data point [True]
438 Text feature [braf] present in test data point [True]
Out of the top  500  features  64 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [77]:

```
test_point_index = 110
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: [[2.680e-02 1.702e-01 1.400e-03 3.310e-02 9.600e-03 3.900e-03 7.537
e-01
  1.200e-03 1.000e-04]]
Actual Class : 2
--------------------------------------------------
24 Text feature [activation] present in test data point [True]
31 Text feature [downstream] present in test data point [True]
57 Text feature [serum] present in test data point [True]
60 Text feature [factor] present in test data point [True]
61 Text feature [malignant] present in test data point [True]
70 Text feature [expressing] present in test data point [True]
78 Text feature [activate] present in test data point [True]
79 Text feature [il] present in test data point [True]
82 Text feature [epithelial] present in test data point [True]
86 Text feature [constitutively] present in test data point [True]
88 Text feature [express] present in test data point [True]
96 Text feature [lung] present in test data point [True]
114 Text feature [61] present in test data point [True]
116 Text feature [carcinomas] present in test data point [True]
129 Text feature [kinase] present in test data point [True]
131 Text feature [growth] present in test data point [True]
133 Text feature [positive] present in test data point [True]
136 Text feature [survival] present in test data point [True]
144 Text feature [bone] present in test data point [True]
150 Text feature [carcinoma] present in test data point [True]
153 Text feature [grade] present in test data point [True]
157 Text feature [2a] present in test data point [True]
158 Text feature [cells] present in test data point [True]
164 Text feature [presence] present in test data point [True]
167 Text feature [independent] present in test data point [True]
186 Text feature [somatic] present in test data point [True]
198 Text feature [fusion] present in test data point [True]
286 Text feature [effective] present in test data point [True]
300 Text feature [phosphorylation] present in test data point [True]
378 Text feature [proliferation] present in test data point [True]
409 Text feature [factors] present in test data point [True]
411 Text feature [tissue] present in test data point [True]
483 Text feature [ret] present in test data point [True]
Out of the top  500  features  33 are present in query point
```

4.3.2. Without Class balancing

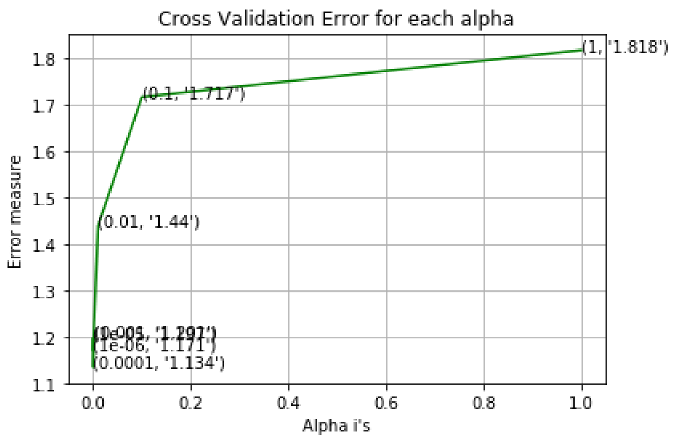
4.3.2.1. Hyper paramter tuning

In [78]:

```
# 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


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 : 1.1714346732927656
for alpha = 1e-05
Log Loss : 1.197286213328466
for alpha = 0.0001
Log Loss : 1.1337687987669227
for alpha = 0.001
Log Loss : 1.2013262215170972
for alpha = 0.01
Log Loss : 1.4398027422067956
for alpha = 0.1
Log Loss : 1.7172074795799352
for alpha = 1
Log Loss : 1.817954465559327
```



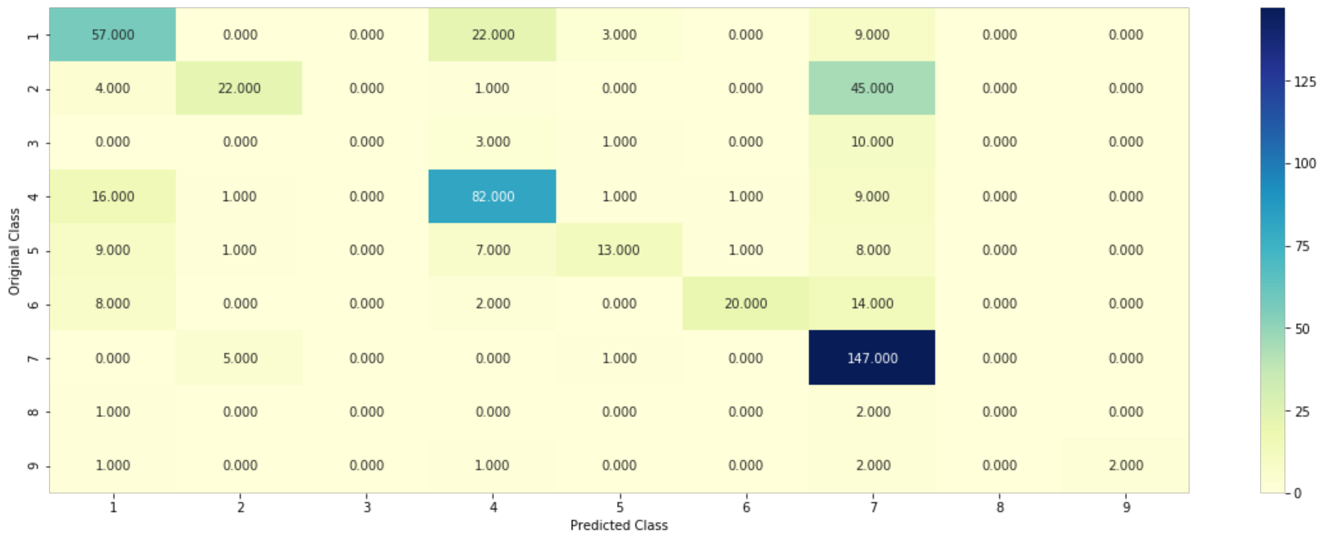
```
For values of best alpha =  0.0001 The train log loss is: 0.4438309067765205
For values of best alpha =  0.0001 The cross validation log loss is: 1.1337687987669227
For values of best alpha =  0.0001 The test log loss is: 1.1064860886699643
```

4.3.2.2. Testing model with best hyper parameters

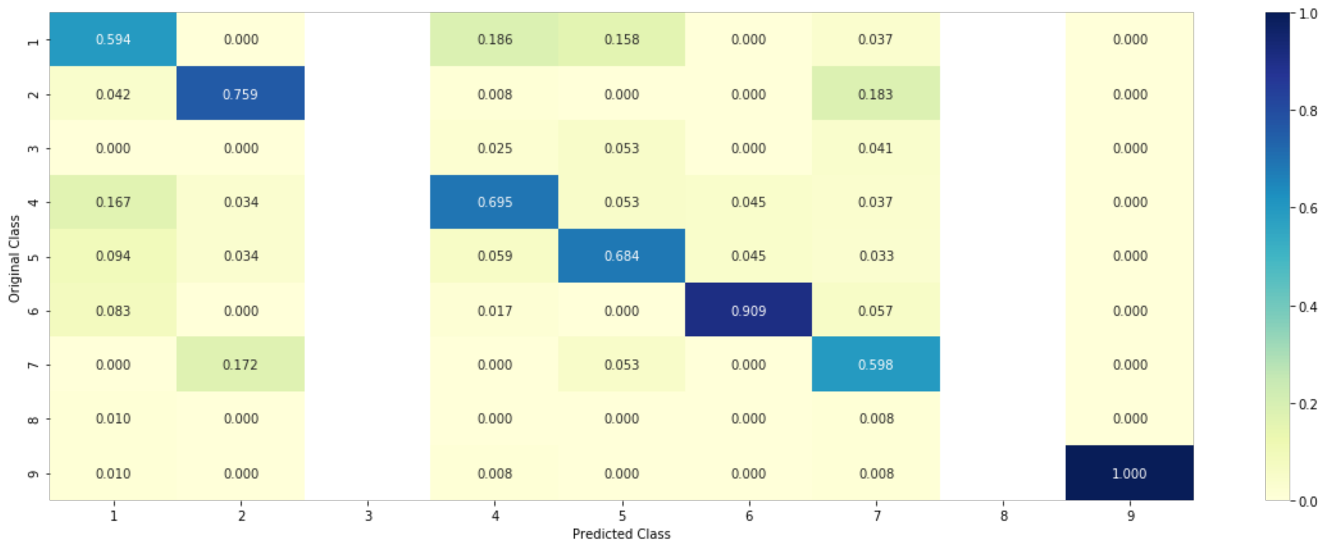
In [79]:

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

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

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

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

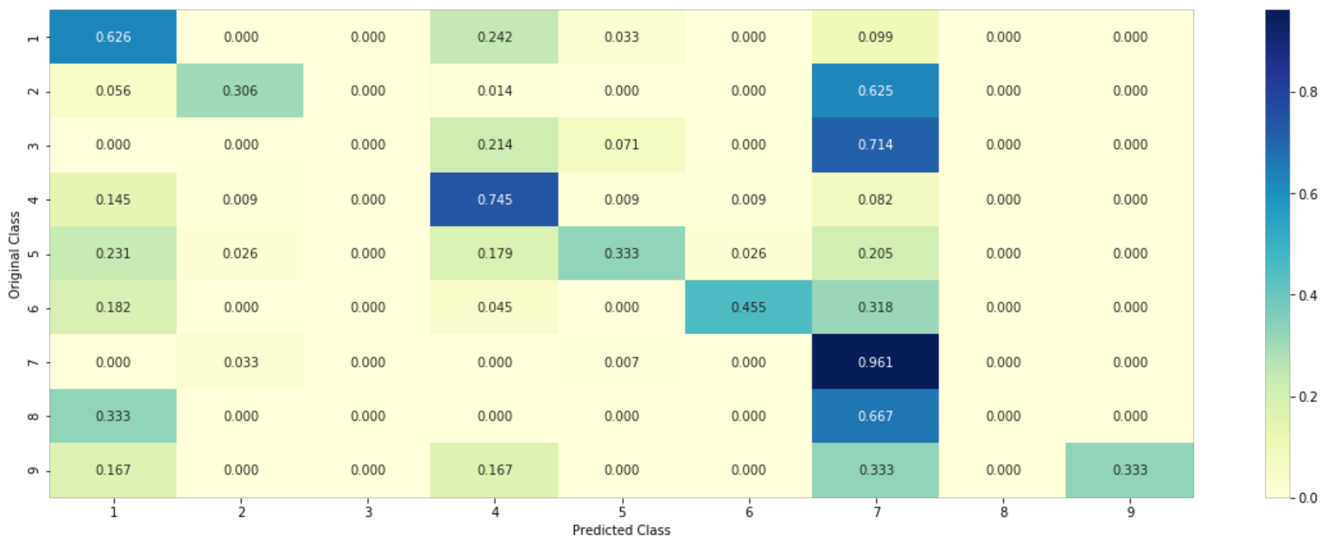
```
Log loss : 1.1337687987669227
Number of mis-classified points : 0.35526315789473684
-------------------- Confusion matrix --------------------
```



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



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



4.3.2.3. Feature Importance, Correctly Classified point

In [80]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
```

```python
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 : 7
Predicted Class Probabilities: [[1.870e-02 2.900e-03 5.000e-03 1.950e-02 1.360e-02 2.000e-03 9.372
e-01
  1.000e-03 1.000e-04]]
Actual Class : 7
--------------------------------------------------
21 Text feature [downstream] present in test data point [True]
23 Text feature [transformed] present in test data point [True]
28 Text feature [activation] present in test data point [True]
36 Text feature [activated] present in test data point [True]
38 Text feature [overexpression] present in test data point [True]
59 Text feature [oncogenic] present in test data point [True]
66 Text feature [colony] present in test data point [True]
79 Text feature [ligand] present in test data point [True]
97 Text feature [derived] present in test data point [True]
98 Text feature [express] present in test data point [True]
100 Text feature [3b] present in test data point [True]
101 Text feature [egf] present in test data point [True]
111 Text feature [somatic] present in test data point [True]
113 Text feature [factor] present in test data point [True]
117 Text feature [fold] present in test data point [True]
128 Text feature [codon] present in test data point [True]
152 Text feature [constitutive] present in test data point [True]
161 Text feature [lung] present in test data point [True]
193 Text feature [epithelial] present in test data point [True]
195 Text feature [2a] present in test data point [True]
204 Text feature [extracellular] present in test data point [True]
205 Text feature [presence] present in test data point [True]
214 Text feature [per] present in test data point [True]
237 Text feature [days] present in test data point [True]
241 Text feature [distinct] present in test data point [True]
259 Text feature [egfr] present in test data point [True]
272 Text feature [size] present in test data point [True]
285 Text feature [her2] present in test data point [True]
296 Text feature [positive] present in test data point [True]
302 Text feature [occur] present in test data point [True]
304 Text feature [adenocarcinoma] present in test data point [True]
313 Text feature [serum] present in test data point [True]
316 Text feature [3a] present in test data point [True]
325 Text feature [signaling] present in test data point [True]
333 Text feature [high] present in test data point [True]
342 Text feature [transforming] present in test data point [True]
345 Text feature [addition] present in test data point [True]
356 Text feature [open] present in test data point [True]
364 Text feature [day] present in test data point [True]
366 Text feature [frequent] present in test data point [True]
374 Text feature [leukemia] present in test data point [True]
384 Text feature [inhibitor] present in test data point [True]
388 Text feature [transformation] present in test data point [True]
389 Text feature [elevated] present in test data point [True]
415 Text feature [lead] present in test data point [True]
429 Text feature [cancers] present in test data point [True]
433 Text feature [within] present in test data point [True]
437 Text feature [cultured] present in test data point [True]
447 Text feature [receptors] present in test data point [True]
451 Text feature [activate] present in test data point [True]
462 Text feature [position] present in test data point [True]
471 Text feature [ras] present in test data point [True]
477 Text feature [expressing] present in test data point [True]
479 Text feature [2002] present in test data point [True]
482 Text feature [cells] present in test data point [True]
486 Text feature [increased] present in test data point [True]
```

```
488 Text feature [carcinoma] present in test data point [True]
490 Text feature [70] present in test data point [True]
493 Text feature [observations] present in test data point [True]
497 Text feature [bone] present in test data point [True]
498 Text feature [total] present in test data point [True]
Out of the top  500  features  61 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [84]:

```
test_point_index = 107
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: [[0.0767 0.1275 0.0017 0.4377 0.0136 0.0034 0.3361 0.0024 0.0007]]
Actual Class : 8
--------------------------------------------------
80 Text feature [suppressor] present in test data point [True]
98 Text feature [tagged] present in test data point [True]
103 Text feature [inactivation] present in test data point [True]
127 Text feature [suggesting] present in test data point [True]
142 Text feature [ca] present in test data point [True]
149 Text feature [germline] present in test data point [True]
159 Text feature [reduced] present in test data point [True]
161 Text feature [transfected] present in test data point [True]
165 Text feature [mammalian] present in test data point [True]
169 Text feature [show] present in test data point [True]
177 Text feature [half] present in test data point [True]
179 Text feature [specifically] present in test data point [True]
184 Text feature [see] present in test data point [True]
185 Text feature [flag] present in test data point [True]
188 Text feature [protein] present in test data point [True]
202 Text feature [tumorigenesis] present in test data point [True]
212 Text feature [iii] present in test data point [True]
226 Text feature [endogenous] present in test data point [True]
234 Text feature [lack] present in test data point [True]
235 Text feature [phenotype] present in test data point [True]
252 Text feature [catalytic] present in test data point [True]
257 Text feature [stability] present in test data point [True]
261 Text feature [washed] present in test data point [True]
267 Text feature [displayed] present in test data point [True]
271 Text feature [due] present in test data point [True]
283 Text feature [right] present in test data point [True]
286 Text feature [bind] present in test data point [True]
296 Text feature [functional] present in test data point [True]
304 Text feature [phosphatase] present in test data point [True]
308 Text feature [determine] present in test data point [True]
309 Text feature [mrna] present in test data point [True]
315 Text feature [methylation] present in test data point [True]
317 Text feature [transfection] present in test data point [True]
331 Text feature [loss] present in test data point [True]
337 Text feature [involved] present in test data point [True]
339 Text feature [altered] present in test data point [True]
344 Text feature [intermediate] present in test data point [True]
345 Text feature [predicted] present in test data point [True]
357 Text feature [high] present in test data point [True]
361 Text feature [changes] present in test data point [True]
363 Text feature [presented] present in test data point [True]
370 Text feature [38] present in test data point [True]
377 Text feature [plasmid] present in test data point [True]
378 Text feature [western] present in test data point [True]
380 Text feature [express] present in test data point [True]
382 Text feature [repair] present in test data point [True]
```

```
392 Text feature [family] present in test data point [True]
398 Text feature [activity] present in test data point [True]
401 Text feature [isolated] present in test data point [True]
405 Text feature [despite] present in test data point [True]
409 Text feature [indicate] present in test data point [True]
411 Text feature [similarly] present in test data point [True]
412 Text feature [left] present in test data point [True]
413 Text feature [transduced] present in test data point [True]
416 Text feature [nuclear] present in test data point [True]
417 Text feature [assay] present in test data point [True]
423 Text feature [low] present in test data point [True]
424 Text feature [university] present in test data point [True]
426 Text feature [suggest] present in test data point [True]
428 Text feature [terminal] present in test data point [True]
431 Text feature [cycle] present in test data point [True]
435 Text feature [rate] present in test data point [True]
436 Text feature [39] present in test data point [True]
438 Text feature [product] present in test data point [True]
441 Text feature [insertions] present in test data point [True]
442 Text feature [29] present in test data point [True]
445 Text feature [2000] present in test data point [True]
447 Text feature [alterations] present in test data point [True]
450 Text feature [linked] present in test data point [True]
452 Text feature [appears] present in test data point [True]
453 Text feature [levels] present in test data point [True]
456 Text feature [described] present in test data point [True]
457 Text feature [proteins] present in test data point [True]
460 Text feature [interact] present in test data point [True]
463 Text feature [leads] present in test data point [True]
465 Text feature [function] present in test data point [True]
468 Text feature [myc] present in test data point [True]
472 Text feature [age] present in test data point [True]
476 Text feature [vector] present in test data point [True]
478 Text feature [regulation] present in test data point [True]
479 Text feature [purified] present in test data point [True]
481 Text feature [stained] present in test data point [True]
482 Text feature [dominant] present in test data point [True]
483 Text feature [3b] present in test data point [True]
487 Text feature [reaction] present in test data point [True]
488 Text feature [characterized] present in test data point [True]
489 Text feature [correlation] present in test data point [True]
490 Text feature [full] present in test data point [True]
491 Text feature [44] present in test data point [True]
493 Text feature [ovarian] present in test data point [True]
496 Text feature [localization] present in test data point [True]
499 Text feature [12] present in test data point [True]
Out of the top  500  features  92 are present in query point
```

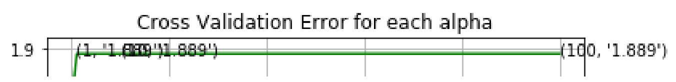
4.4. Linear Support Vector Machines

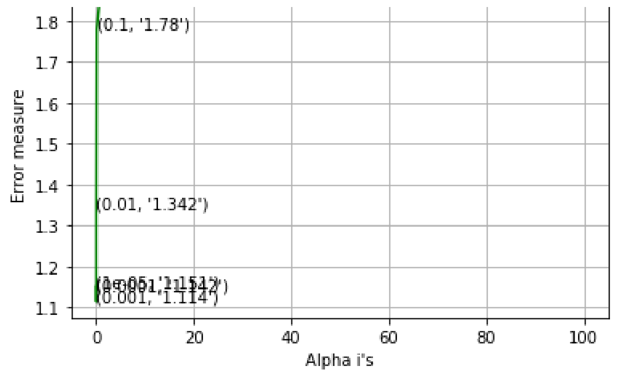
4.4.1. Hyper paramter tuning

In [85]:

```
# 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

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 : 1.1514592307171332
for C = 0.0001
Log Loss : 1.1418990748003925
for C = 0.001
Log Loss : 1.1140148575239075
for C = 0.01
Log Loss : 1.3419954923724646
for C = 0.1
Log Loss : 1.7797798513232725
for C = 1
Log Loss : 1.8887253072241317
for C = 10
Log Loss : 1.8887253772268362
for C = 100
Log Loss : 1.8887253825802615
```





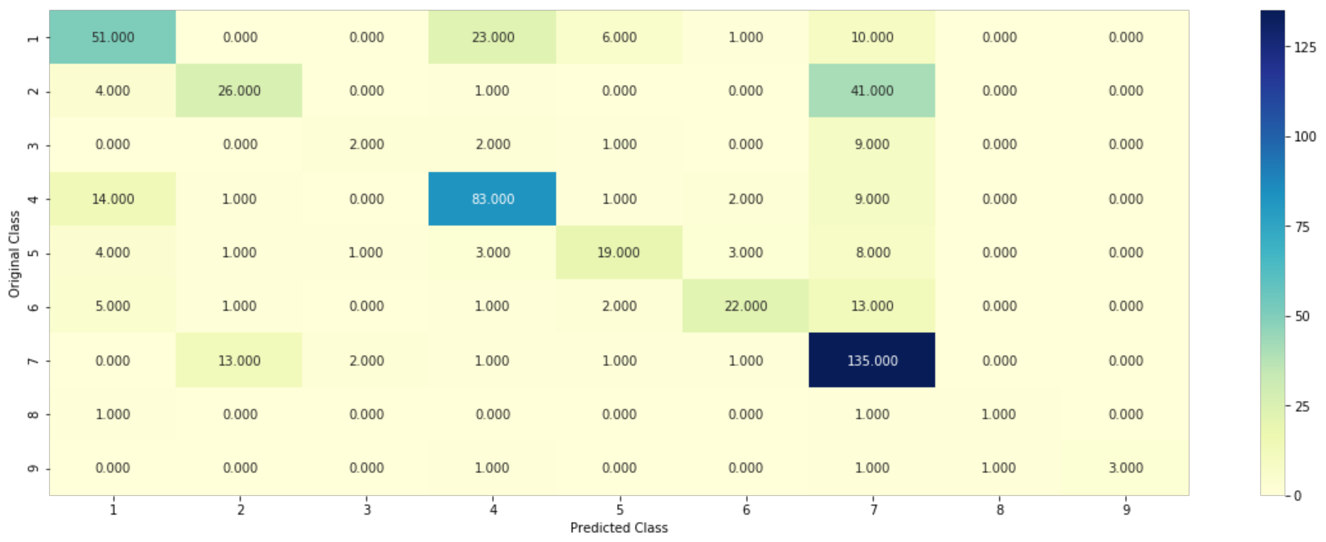
```
For values of best alpha =  0.001 The train log loss is: 0.6095234222537416
For values of best alpha =  0.001 The cross validation log loss is: 1.1140148575239075
For values of best alpha =  0.001 The test log loss is: 1.1411770625309539
```

4.4.2. Testing model with best hyper parameters

In [86]:

```
# 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)
```

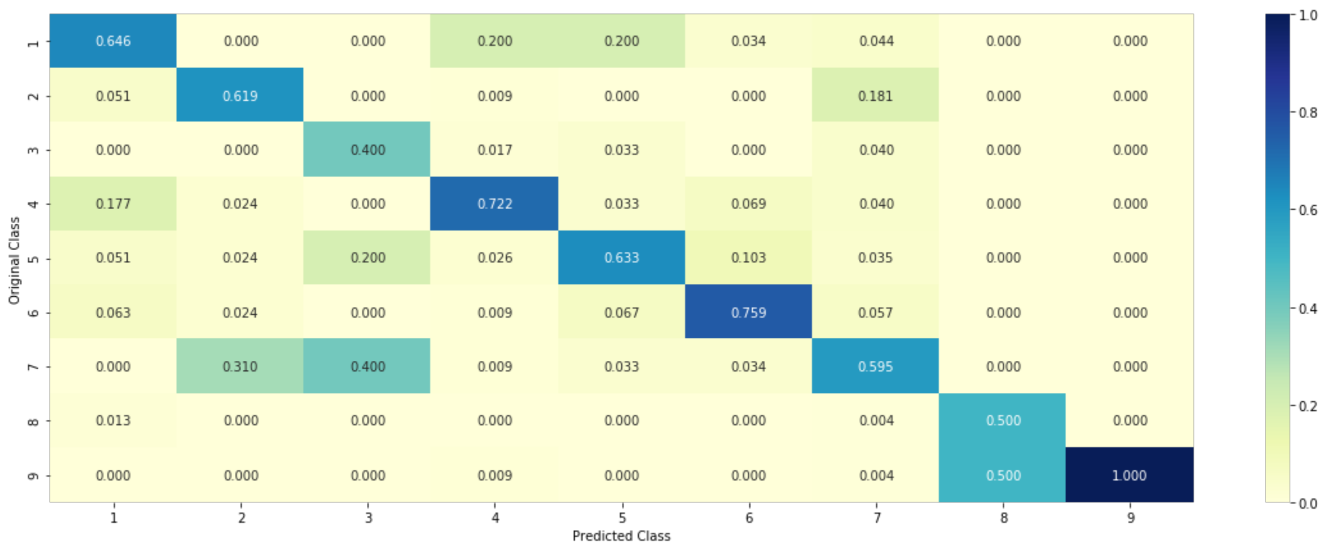
```
Log loss : 1.1140148575239075
Number of mis-classified points : 0.35714285714285715
-------------------- Confusion matrix --------------------
```



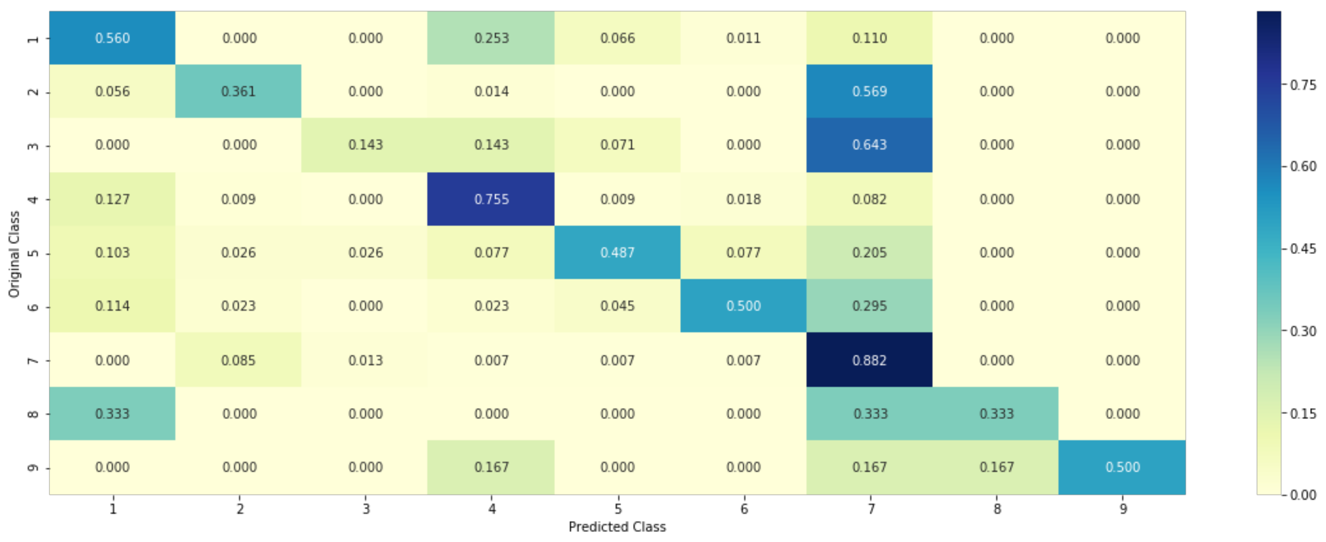
	1	2	3	4	5	6	7	8	9
1	51.000	0.000	0.000	23.000	6.000	1.000	10.000	0.000	0.000
2	4.000	26.000	0.000	1.000	0.000	0.000	41.000	0.000	0.000
3	0.000	0.000	2.000	2.000	1.000	0.000	9.000	0.000	0.000
4	14.000	1.000	0.000	83.000	1.000	2.000	9.000	0.000	0.000
5	4.000	1.000	1.000	3.000	19.000	3.000	8.000	0.000	0.000
6	5.000	1.000	0.000	1.000	2.000	22.000	13.000	0.000	0.000
7	0.000	13.000	2.000	1.000	1.000	1.000	135.000	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	0.000	1.000	1.000	0.000
9	0.000	0.000	0.000	1.000	0.000	0.000	1.000	1.000	3.000

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

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



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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [87]:

```python
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 : 7
Predicted Class Probabilities: [[0.0472 0.0076 0.014  0.0601 0.0263 0.0052 0.8373 0.0015 0.001 ]]
Actual Class : 7
```

```
--------------------------------------------------
17 Text feature [activation] present in test data point [True]
18 Text feature [constitutive] present in test data point [True]
19 Text feature [transformed] present in test data point [True]
29 Text feature [extracellular] present in test data point [True]
30 Text feature [downstream] present in test data point [True]
31 Text feature [activated] present in test data point [True]
32 Text feature [ligand] present in test data point [True]
33 Text feature [egf] present in test data point [True]
41 Text feature [oncogenic] present in test data point [True]
42 Text feature [2a] present in test data point [True]
43 Text feature [codon] present in test data point [True]
44 Text feature [express] present in test data point [True]
46 Text feature [factor] present in test data point [True]
67 Text feature [derived] present in test data point [True]
69 Text feature [colony] present in test data point [True]
72 Text feature [leukemia] present in test data point [True]
73 Text feature [overexpression] present in test data point [True]
74 Text feature [day] present in test data point [True]
75 Text feature [presence] present in test data point [True]
76 Text feature [egfr] present in test data point [True]
77 Text feature [3b] present in test data point [True]
78 Text feature [lead] present in test data point [True]
79 Text feature [lung] present in test data point [True]
81 Text feature [somatic] present in test data point [True]
82 Text feature [expressing] present in test data point [True]
84 Text feature [epithelial] present in test data point [True]
85 Text feature [positive] present in test data point [True]
262 Text feature [activate] present in test data point [True]
263 Text feature [transformation] present in test data point [True]
264 Text feature [fold] present in test data point [True]
266 Text feature [adenocarcinoma] present in test data point [True]
267 Text feature [absence] present in test data point [True]
268 Text feature [serum] present in test data point [True]
269 Text feature [signaling] present in test data point [True]
272 Text feature [addition] present in test data point [True]
273 Text feature [cancers] present in test data point [True]
274 Text feature [occur] present in test data point [True]
275 Text feature [days] present in test data point [True]
276 Text feature [transforming] present in test data point [True]
277 Text feature [gfp] present in test data point [True]
280 Text feature [higher] present in test data point [True]
282 Text feature [inhibitor] present in test data point [True]
283 Text feature [cells] present in test data point [True]
284 Text feature [her2] present in test data point [True]
285 Text feature [size] present in test data point [True]
287 Text feature [open] present in test data point [True]
453 Text feature [showed] present in test data point [True]
455 Text feature [3a] present in test data point [True]
456 Text feature [high] present in test data point [True]
458 Text feature [without] present in test data point [True]
459 Text feature [frequent] present in test data point [True]
461 Text feature [factors] present in test data point [True]
462 Text feature [per] present in test data point [True]
464 Text feature [carcinoma] present in test data point [True]
467 Text feature [sensitive] present in test data point [True]
470 Text feature [ras] present in test data point [True]
471 Text feature [mutants] present in test data point [True]
472 Text feature [inhibited] present in test data point [True]
474 Text feature [distinct] present in test data point [True]
475 Text feature [bone] present in test data point [True]
477 Text feature [s3] present in test data point [True]
479 Text feature [transduced] present in test data point [True]
483 Text feature [independent] present in test data point [True]
486 Text feature [examined] present in test data point [True]
487 Text feature [tyrosine] present in test data point [True]
489 Text feature [increase] present in test data point [True]
492 Text feature [2b] present in test data point [True]
493 Text feature [2001] present in test data point [True]
495 Text feature [cultured] present in test data point [True]
496 Text feature [common] present in test data point [True]
497 Text feature [within] present in test data point [True]
499 Text feature [activating] present in test data point [True]
Out of the top  500  features  72 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [89]:

```
test_point_index = 107
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: [[0.1045 0.108  0.0064 0.3727 0.0289 0.0073 0.3584 0.0067 0.0071]]
Actual Class : 8
--------------------------------------------------
22 Text feature [suppressor] present in test data point [True]
28 Text feature [inactivation] present in test data point [True]
30 Text feature [suggesting] present in test data point [True]
33 Text feature [tagged] present in test data point [True]
37 Text feature [flag] present in test data point [True]
39 Text feature [germline] present in test data point [True]
40 Text feature [transfected] present in test data point [True]
41 Text feature [show] present in test data point [True]
222 Text feature [lack] present in test data point [True]
224 Text feature [see] present in test data point [True]
225 Text feature [stability] present in test data point [True]
226 Text feature [mammalian] present in test data point [True]
228 Text feature [catalytic] present in test data point [True]
229 Text feature [reduced] present in test data point [True]
230 Text feature [ca] present in test data point [True]
231 Text feature [purified] present in test data point [True]
233 Text feature [due] present in test data point [True]
234 Text feature [iii] present in test data point [True]
235 Text feature [altered] present in test data point [True]
236 Text feature [half] present in test data point [True]
237 Text feature [changes] present in test data point [True]
334 Text feature [determine] present in test data point [True]
335 Text feature [university] present in test data point [True]
336 Text feature [involved] present in test data point [True]
338 Text feature [washed] present in test data point [True]
339 Text feature [high] present in test data point [True]
340 Text feature [tumorigenesis] present in test data point [True]
342 Text feature [functional] present in test data point [True]
343 Text feature [predicted] present in test data point [True]
344 Text feature [presented] present in test data point [True]
345 Text feature [repair] present in test data point [True]
346 Text feature [rate] present in test data point [True]
351 Text feature [specifically] present in test data point [True]
352 Text feature [myc] present in test data point [True]
353 Text feature [38] present in test data point [True]
354 Text feature [localization] present in test data point [True]
356 Text feature [protein] present in test data point [True]
358 Text feature [reaction] present in test data point [True]
359 Text feature [phosphatase] present in test data point [True]
360 Text feature [phenotype] present in test data point [True]
362 Text feature [loss] present in test data point [True]
364 Text feature [alterations] present in test data point [True]
403 Text feature [transduced] present in test data point [True]
405 Text feature [direct] present in test data point [True]
407 Text feature [endogenous] present in test data point [True]
411 Text feature [right] present in test data point [True]
412 Text feature [12] present in test data point [True]
413 Text feature [family] present in test data point [True]
414 Text feature [intermediate] present in test data point [True]
415 Text feature [despite] present in test data point [True]
416 Text feature [still] present in test data point [True]
417 Text feature [linked] present in test data point [True]
418 Text feature [activity] present in test data point [True]
419 Text feature [notch1] present in test data point [True]
420 Text feature [2000] present in test data point [True]
424 Text feature [low] present in test data point [True]
```

```
424 Text feature [low] present in test data point [True]
426 Text feature [appears] present in test data point [True]
427 Text feature [displayed] present in test data point [True]
428 Text feature [fact] present in test data point [True]
429 Text feature [dominant] present in test data point [True]
431 Text feature [characterized] present in test data point [True]
433 Text feature [figure] present in test data point [True]
436 Text feature [western] present in test data point [True]
437 Text feature [negative] present in test data point [True]
442 Text feature [indicate] present in test data point [True]
443 Text feature [nuclear] present in test data point [True]
444 Text feature [isolated] present in test data point [True]
445 Text feature [complex] present in test data point [True]
446 Text feature [mechanisms] present in test data point [True]
447 Text feature [product] present in test data point [True]
449 Text feature [levels] present in test data point [True]
450 Text feature [29] present in test data point [True]
451 Text feature [cycle] present in test data point [True]
453 Text feature [various] present in test data point [True]
454 Text feature [methylation] present in test data point [True]
455 Text feature [hours] present in test data point [True]
456 Text feature [stained] present in test data point [True]
457 Text feature [testing] present in test data point [True]
458 Text feature [transfection] present in test data point [True]
460 Text feature [described] present in test data point [True]
461 Text feature [genetic] present in test data point [True]
462 Text feature [recent] present in test data point [True]
463 Text feature [rates] present in test data point [True]
469 Text feature [test] present in test data point [True]
470 Text feature [rna] present in test data point [True]
471 Text feature [aberrant] present in test data point [True]
472 Text feature [proteins] present in test data point [True]
473 Text feature [locus] present in test data point [True]
474 Text feature [mice] present in test data point [True]
477 Text feature [plasmid] present in test data point [True]
478 Text feature [able] present in test data point [True]
479 Text feature [bind] present in test data point [True]
480 Text feature [incubated] present in test data point [True]
481 Text feature [3b] present in test data point [True]
Out of the top  500  features  94 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [90]:

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


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 : 1.2334032199289395
for n_estimators = 100 and max depth =  10
Log Loss : 1.2687022639691015
for n_estimators = 200 and max depth =  5
Log Loss : 1.2144559720382253
for n_estimators = 200 and max depth =  10
Log Loss : 1.2574926159027042
for n_estimators = 500 and max depth =  5
Log Loss : 1.2106435147520442
for n_estimators = 500 and max depth =  10
Log Loss : 1.2533936499595932
for n_estimators = 1000 and max depth =  5
Log Loss : 1.206806868043842
for n_estimators = 1000 and max depth =  10
Log Loss : 1.249065839984148
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2053850610593415
for n_estimators = 2000 and max depth =  10
Log Loss : 1.247616667612019
```

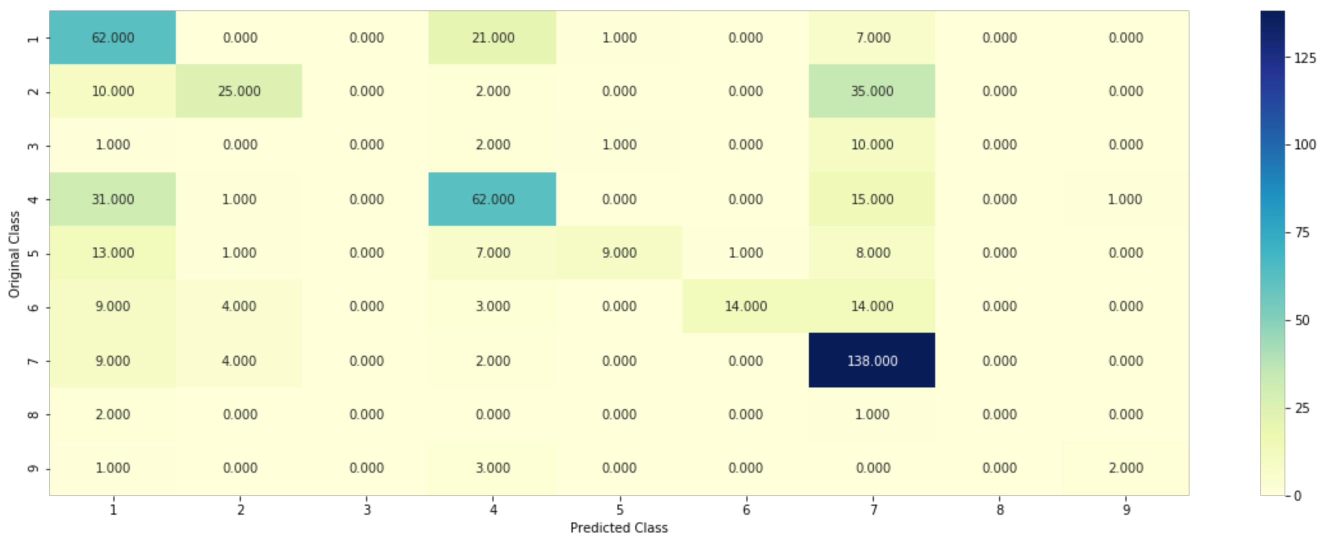
```
For values of best estimator =  2000 The train log loss is: 0.8718462712696455
For values of best estimator =  2000 The cross validation log loss is: 1.2053850610593415
For values of best estimator =  2000 The test log loss is: 1.2020733518052715
```

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

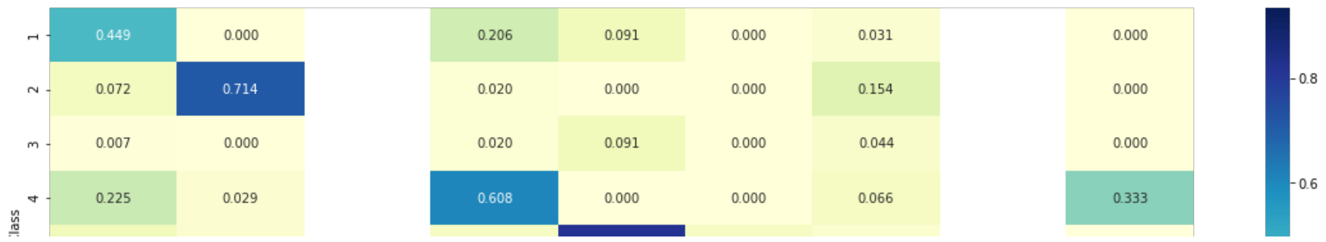
In [91]:

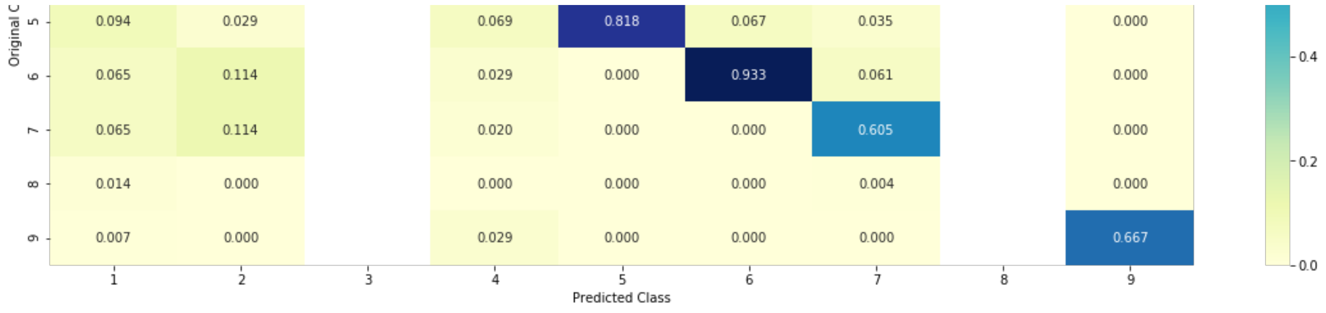
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None,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

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

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

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

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

```
Log loss : 1.2053850610593415
Number of mis-classified points : 0.41353383458646614
-------------------- Confusion matrix --------------------
```

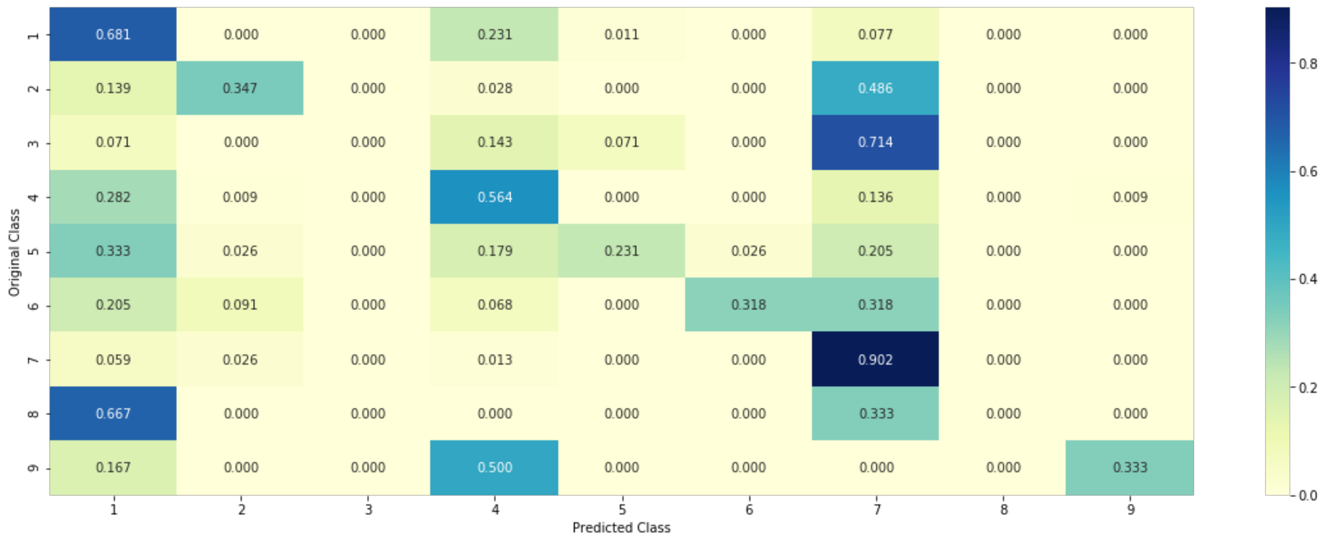


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [92]:

```
# 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 : 7
Predicted Class Probabilities: [[0.0395 0.114  0.0221 0.0308 0.042  0.0316 0.7121 0.0053 0.0026]]
Actual Class : 7
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [tyrosine] present in test data point [True]
4 Text feature [function] present in test data point [True]
5 Text feature [activation] present in test data point [True]
6 Text feature [activated] present in test data point [True]
7 Text feature [inhibitors] present in test data point [True]
```

```
8 Text feature [constitutive] present in test data point [True]
9 Text feature [missense] present in test data point [True]
11 Text feature [treatment] present in test data point [True]
13 Text feature [loss] present in test data point [True]
15 Text feature [receptor] present in test data point [True]
17 Text feature [inhibitor] present in test data point [True]
19 Text feature [oncogenic] present in test data point [True]
20 Text feature [signaling] present in test data point [True]
23 Text feature [constitutively] present in test data point [True]
24 Text feature [transforming] present in test data point [True]
25 Text feature [protein] present in test data point [True]
26 Text feature [extracellular] present in test data point [True]
29 Text feature [cells] present in test data point [True]
30 Text feature [growth] present in test data point [True]
32 Text feature [therapeutic] present in test data point [True]
33 Text feature [functional] present in test data point [True]
38 Text feature [cell] present in test data point [True]
41 Text feature [kinases] present in test data point [True]
51 Text feature [mek] present in test data point [True]
55 Text feature [downstream] present in test data point [True]
56 Text feature [expression] present in test data point [True]
57 Text feature [activate] present in test data point [True]
58 Text feature [proteins] present in test data point [True]
60 Text feature [efficacy] present in test data point [True]
61 Text feature [treated] present in test data point [True]
63 Text feature [inhibition] present in test data point [True]
65 Text feature [functions] present in test data point [True]
69 Text feature [ovarian] present in test data point [True]
70 Text feature [resistance] present in test data point [True]
71 Text feature [ability] present in test data point [True]
72 Text feature [affect] present in test data point [True]
74 Text feature [dna] present in test data point [True]
77 Text feature [patients] present in test data point [True]
78 Text feature [clinical] present in test data point [True]
80 Text feature [activity] present in test data point [True]
81 Text feature [nsclc] present in test data point [True]
82 Text feature [inhibited] present in test data point [True]
84 Text feature [lines] present in test data point [True]
87 Text feature [egfr] present in test data point [True]
89 Text feature [lung] present in test data point [True]
90 Text feature [tagged] present in test data point [True]
91 Text feature [dose] present in test data point [True]
95 Text feature [wild] present in test data point [True]
96 Text feature [ligand] present in test data point [True]
97 Text feature [proliferation] present in test data point [True]
99 Text feature [ras] present in test data point [True]
Out of the top  100  features  53 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [93]:

```
test_point_index = 107
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 : 1
Predicted Class Probabilities: [[0.4366 0.0731 0.0137 0.146  0.0452 0.038  0.1247 0.0172 0.1055]]
Actuall Class : 8
--------------------------------------------------
3 Text feature [suppressor] present in test data point [True]
4 Text feature [function] present in test data point [True]
5 Text feature [activation] present in test data point [True]
6 Text feature [activated] present in test data point [True]
7 Text feature [inhibitors] present in test data point [True]
8 Text feature [constitutive] present in test data point [True]
```

```
11 Text feature [treatment] present in test data point [True]
13 Text feature [loss] present in test data point [True]
14 Text feature [stability] present in test data point [True]
15 Text feature [receptor] present in test data point [True]
17 Text feature [inhibitor] present in test data point [True]
19 Text feature [oncogenic] present in test data point [True]
20 Text feature [signaling] present in test data point [True]
23 Text feature [constitutively] present in test data point [True]
25 Text feature [protein] present in test data point [True]
29 Text feature [cells] present in test data point [True]
30 Text feature [growth] present in test data point [True]
32 Text feature [therapeutic] present in test data point [True]
33 Text feature [functional] present in test data point [True]
34 Text feature [therapy] present in test data point [True]
35 Text feature [variants] present in test data point [True]
36 Text feature [phosphatase] present in test data point [True]
38 Text feature [cell] present in test data point [True]
42 Text feature [months] present in test data point [True]
43 Text feature [57] present in test data point [True]
45 Text feature [repair] present in test data point [True]
47 Text feature [brca] present in test data point [True]
55 Text feature [downstream] present in test data point [True]
56 Text feature [expression] present in test data point [True]
57 Text feature [activate] present in test data point [True]
58 Text feature [proteins] present in test data point [True]
61 Text feature [treated] present in test data point [True]
62 Text feature [predicted] present in test data point [True]
63 Text feature [inhibition] present in test data point [True]
65 Text feature [functions] present in test data point [True]
67 Text feature [advanced] present in test data point [True]
69 Text feature [ovarian] present in test data point [True]
71 Text feature [ability] present in test data point [True]
72 Text feature [affect] present in test data point [True]
74 Text feature [dna] present in test data point [True]
75 Text feature [inactivation] present in test data point [True]
77 Text feature [patients] present in test data point [True]
78 Text feature [clinical] present in test data point [True]
80 Text feature [activity] present in test data point [True]
82 Text feature [inhibited] present in test data point [True]
83 Text feature [nuclear] present in test data point [True]
84 Text feature [lines] present in test data point [True]
85 Text feature [variant] present in test data point [True]
88 Text feature [mammalian] present in test data point [True]
89 Text feature [lung] present in test data point [True]
90 Text feature [tagged] present in test data point [True]
93 Text feature [catalytic] present in test data point [True]
95 Text feature [wild] present in test data point [True]
97 Text feature [proliferation] present in test data point [True]
Out of the top  100  features  54 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [94]:

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

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 : 2.1496365380109683
for n_estimators = 10 and max depth =  3
Log Loss : 1.679124241710989
for n_estimators = 10 and max depth =  5
Log Loss : 1.4181277144701456
for n_estimators = 10 and max depth =  10
Log Loss : 2.019555031730607
for n_estimators = 50 and max depth =  2
Log Loss : 1.7547191533890396
for n_estimators = 50 and max depth =  3
Log Loss : 1.4490225468554727
for n_estimators = 50 and max depth =  5
Log Loss : 1.3990857967627308
for n_estimators = 50 and max depth =  10
```

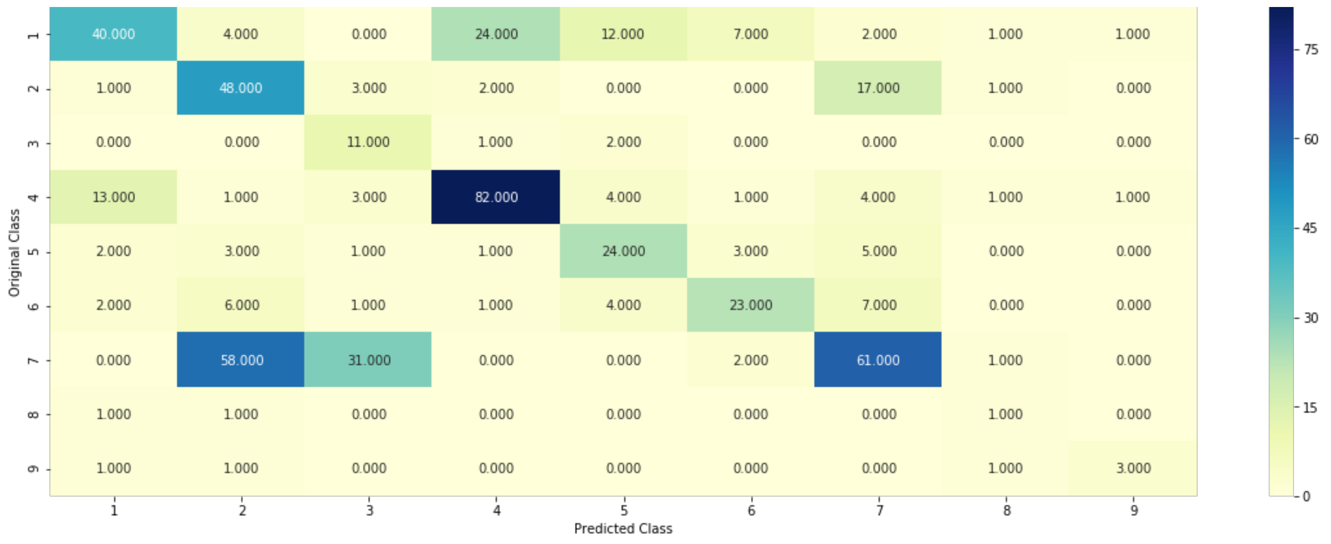
```
Log Loss : 1.7581300470782224
for n_estimators = 100 and max depth =  2
Log Loss : 1.6539761174710264
for n_estimators = 100 and max depth =  3
Log Loss : 1.456904051074765
for n_estimators = 100 and max depth =  5
Log Loss : 1.306538010159577
for n_estimators = 100 and max depth =  10
Log Loss : 1.7416134199808646
for n_estimators = 200 and max depth =  2
Log Loss : 1.624990297672473
for n_estimators = 200 and max depth =  3
Log Loss : 1.4786164351057505
for n_estimators = 200 and max depth =  5
Log Loss : 1.386818997936361
for n_estimators = 200 and max depth =  10
Log Loss : 1.776141092326142
for n_estimators = 500 and max depth =  2
Log Loss : 1.7045422910129615
for n_estimators = 500 and max depth =  3
Log Loss : 1.562344702680422
for n_estimators = 500 and max depth =  5
Log Loss : 1.3942295902813193
for n_estimators = 500 and max depth =  10
Log Loss : 1.7417778389162208
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6682926246048218
for n_estimators = 1000 and max depth =  3
Log Loss : 1.5519250085446086
for n_estimators = 1000 and max depth =  5
Log Loss : 1.4158206388476329
for n_estimators = 1000 and max depth =  10
Log Loss : 1.7159583267395597
For values of best alpha =  100 The train log loss is: 0.06299003394985564
For values of best alpha =  100 The cross validation log loss is: 1.3065380101595772
For values of best alpha =  100 The test log loss is: 1.2897945703648426
```

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

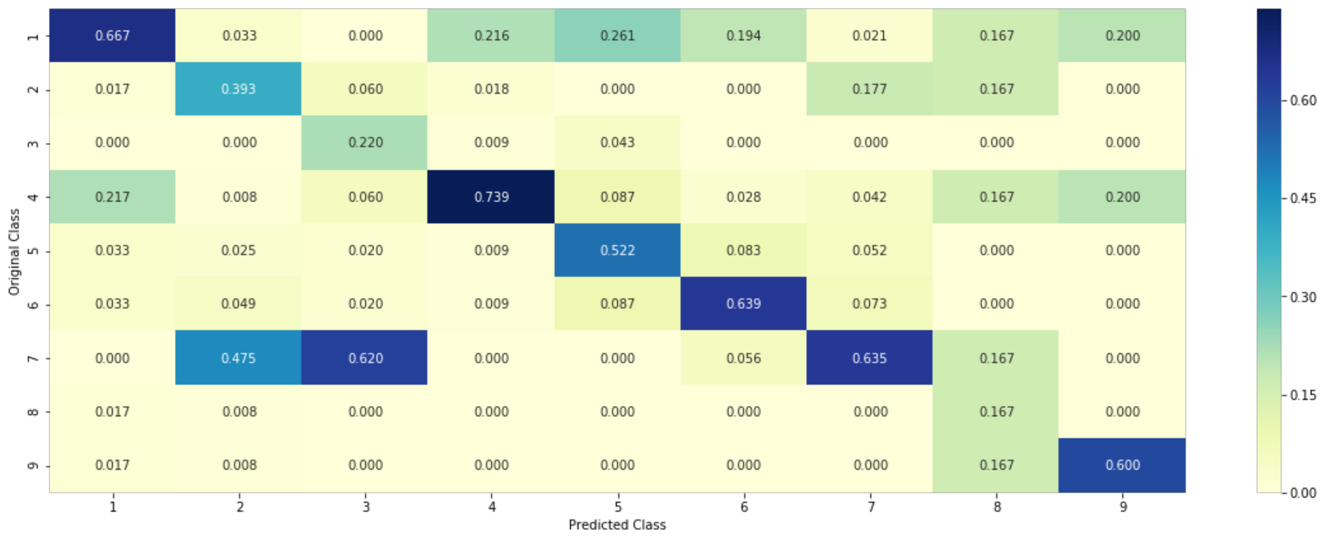
In [95]:

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

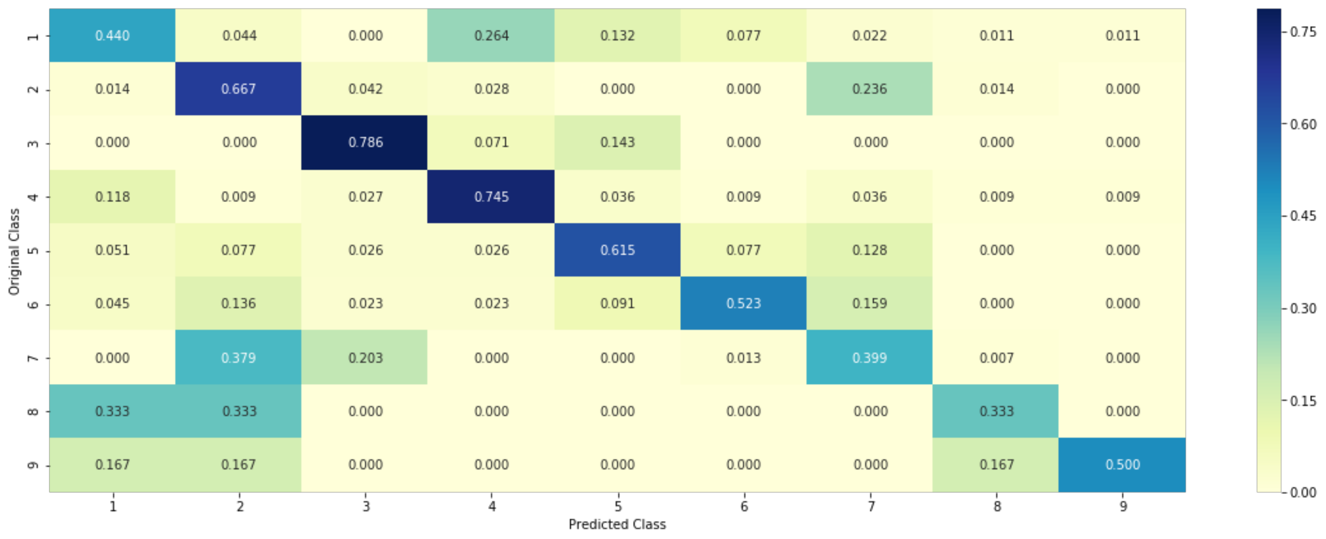
```
Log loss : 1.306538010159577
Number of mis-classified points : 0.4492481203007519
-------------------- Confusion matrix --------------------
```



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



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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [96]:

```
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 = 10
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: [[0.3467 0.022  0.1299 0.3668 0.0381 0.049  0.0064 0.0206 0.0204]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Variation 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 [97]:

```
test_point_index = 107
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")
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0708 0.2383 0.0662 0.0732 0.0298 0.0344 0.0308 0.238  0.2185]]
Actual Class : 8
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Variation 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 [98]:

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

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

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


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

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


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

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

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

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


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

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


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

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

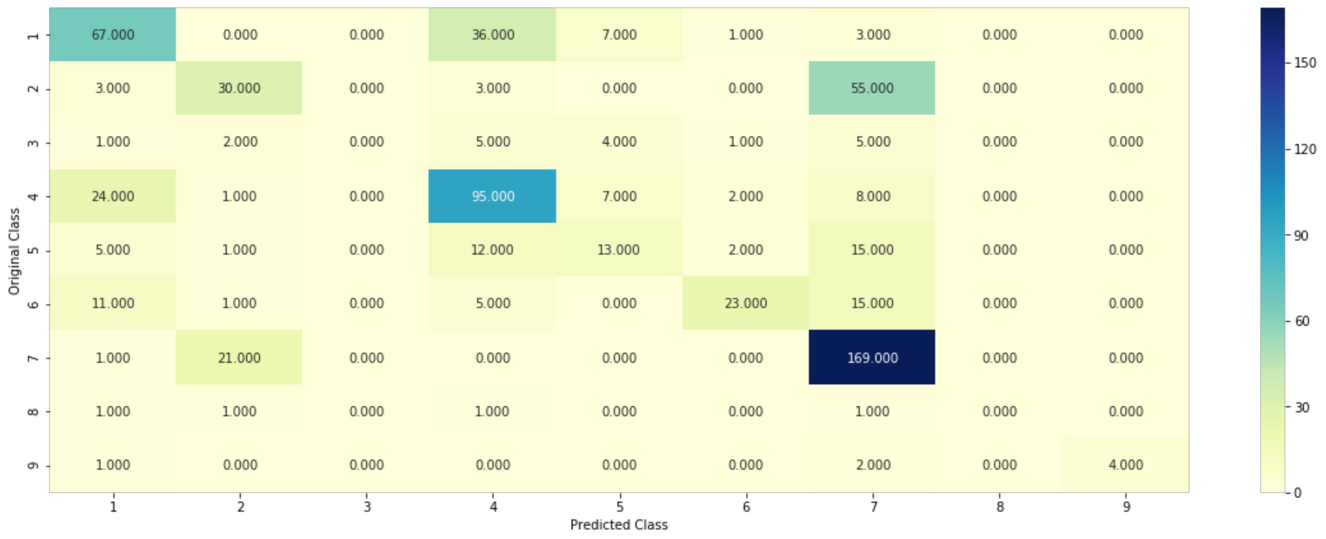
```
Logistic Regression :  Log Loss: 1.07
Support vector machines : Log Loss: 1.89
Naive Bayes : Log Loss: 1.19
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.032
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.504
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.165
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.366
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.835
```

4.7.2 testing the model with the best hyper parameters

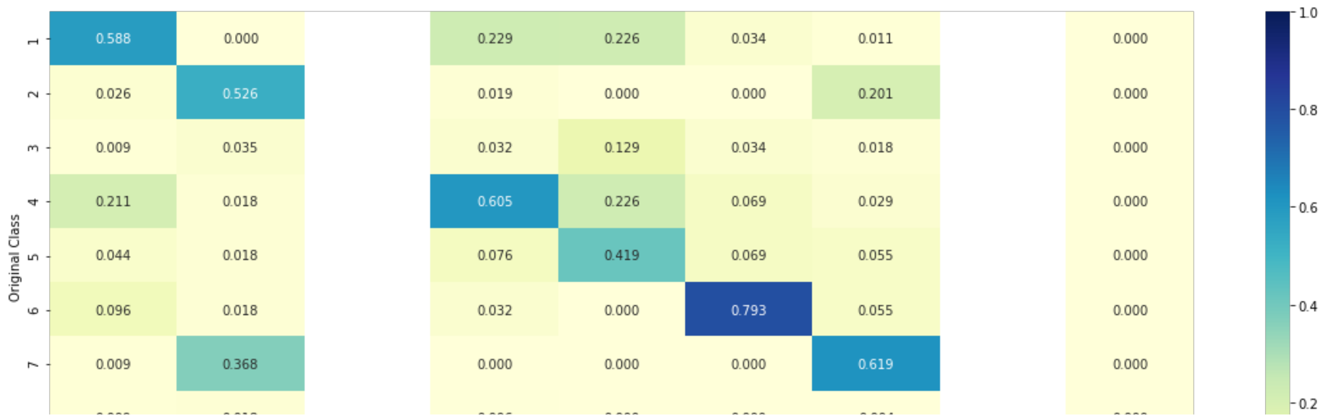
In [99]:

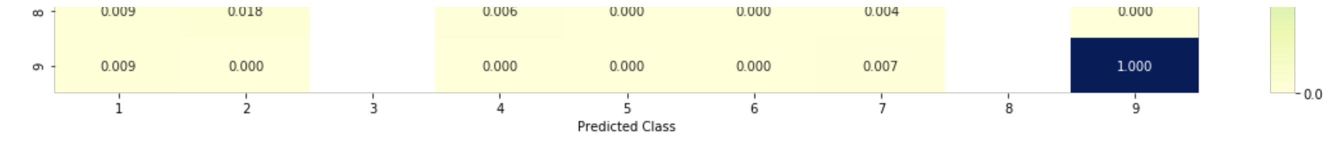
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba
s=True)
sclf.fit(train_x_onehotCoding, train_y)

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 : 0.5516407625330033
Log loss (CV) on the stacking classifier : 1.1650007314503938
Log loss (test) on the stacking classifier : 1.217427795929318
Number of missclassified point : 0.3969924812030075
-------------------- Confusion matrix --------------------
```

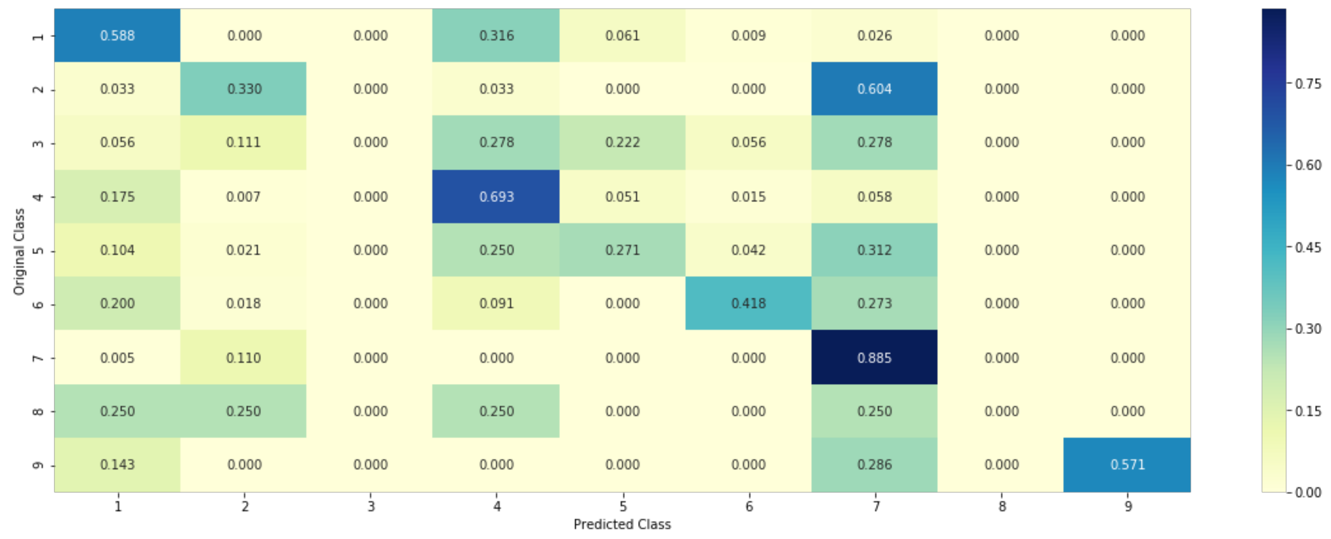


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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

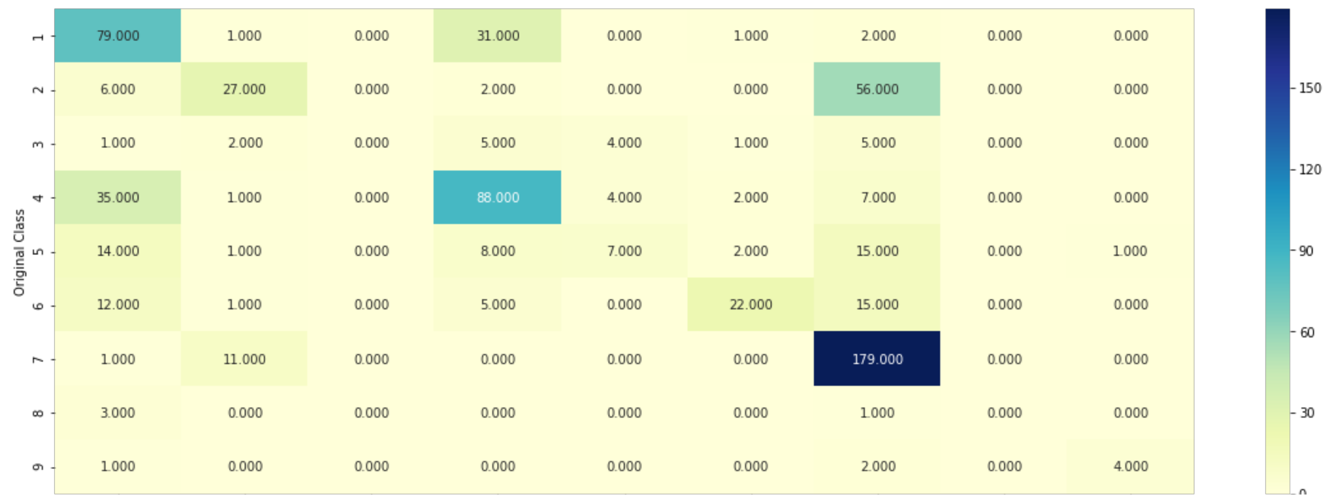


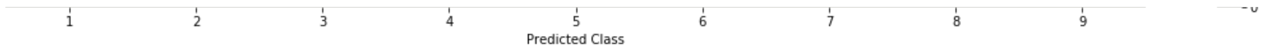
4.7.3 Maximum Voting classifier

In [100]:

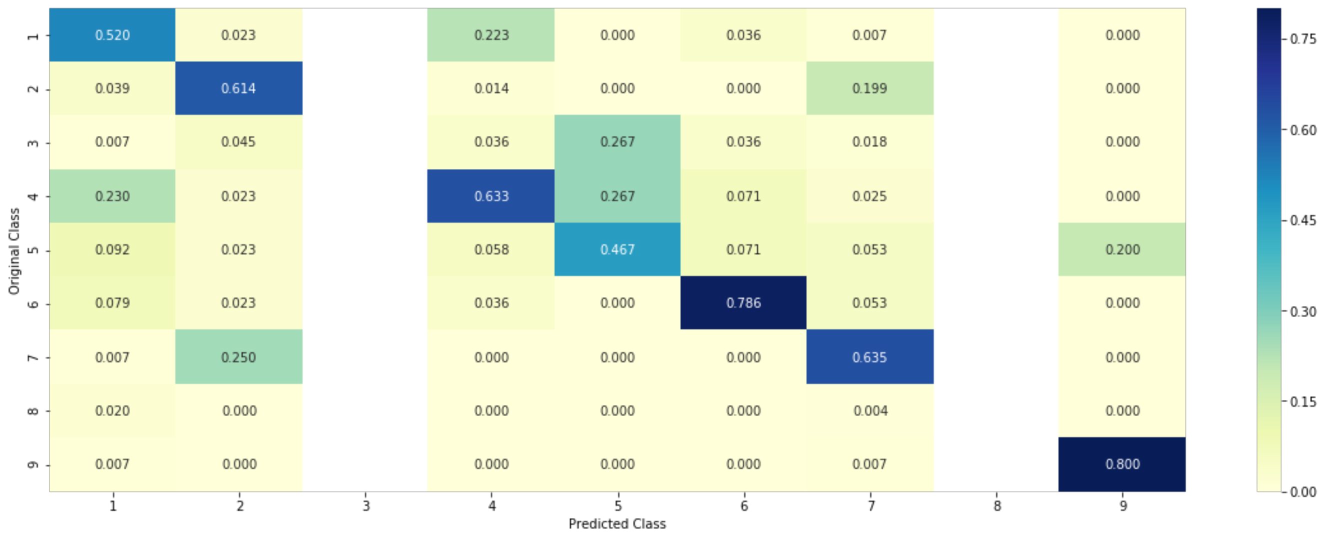
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting=
'soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log 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 : 0.8386643489783056
Log loss (CV) on the VotingClassifier : 1.217328958028138
Log loss (test) on the VotingClassifier : 1.249026484227864
Number of missclassified point : 0.3894736842105263
-------------------- Confusion matrix --------------------
```

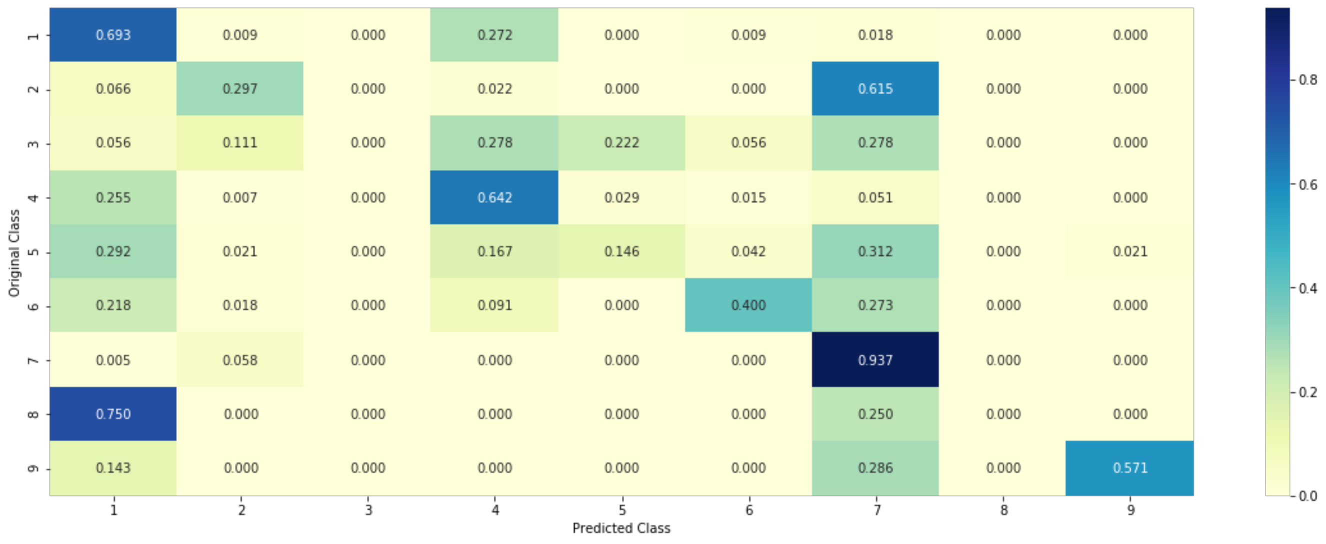




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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



5. Model Comparision

In [103]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Model", "Train loss", "CV loss", "Test loss", "% of misclassified points"]

x.add_row(['NB - one hot encoding', 0.546, 1.194, 1.245, 37.2])
x.add_row(['KNN - Response Coding', 0.639, 1.036, 1.071, 39.6])
x.add_row(['Logistic Regression(Balanced) - one hot encoding', 0.726, 1.077, 1.092, 36.0])
x.add_row(['Logistic Regression(Un-Balanced) - one hot encoding', 0.443, 1.133, 1.106, 35.5])
x.add_row(['Linear SVM - one hot encoding', 0.609, 1.114, 1.141, 35.7])
x.add_row(['Random Forest Classifier - one hot encoding', 0.871, 1.205, 1.202, 41.3])
x.add_row(['Random Forest Classifier - Response Coding', 0.062, 1.306, 1.289, 44.9])
x.add_row(['Stacking Model - one hot encoding', 0.551, 1.165, 1.217, 39.6])
x.add_row(['Maximum Voting - one hot encoding', 0.838, 1.217, 1.249, 38.9])

print(x)
```

```
+-------------------------------------------------------+------------+---------+-----------+---------
---------------+
|                        Model                          | Train loss | CV loss | Test loss | % of
misclassified points |
+-------------------------------------------------------+------------+---------+-----------+---------
---------------+
|                NB - one hot encoding                  |   0.546    |  1.194  |   1.245   |
37.2           |
|                KNN - Response Coding                  |   0.639    |  1.036  |   1.071   |
39.6           |
|   Logistic Regression(Balanced) - one hot encoding   |   0.726    |  1.077  |   1.092   |
36.0           |
| Logistic Regression(Un-Balanced) - one hot encoding |   0.443    |  1.133  |   1.106   |
35.5           |
|            Linear SVM - one hot encoding              |   0.609    |  1.114  |   1.141   |
35.7           |
|     Random Forest Classifier - one hot encoding       |   0.871    |  1.205  |   1.202   |
41.3           |
|      Random Forest Classifier - Response Coding       |   0.062    |  1.306  |   1.289   |
44.9           |
|          Stacking Model - one hot encoding            |   0.551    |  1.165  |   1.217   |
39.6           |
|          Maximum Voting - one hot encoding            |   0.838    |  1.217  |   1.249   |
38.9           |
+-------------------------------------------------------+------------+---------+-----------+---------
---------------+
```

6. Unigram and Bigram on text Feature

6.1 Unigram

6.1.2 Uni- Variate analysis:

In [106]:

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

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of 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 : 53174
```

In [107]:

```
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 [109]:

```
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 [110]:

```
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 [111]:

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

```
Counter({3: 5526, 4: 3822, 6: 2974, 5: 2540, 7: 2217, 9: 1910, 8: 1748, 10: 1498, 12: 1424, 11: 987
, 15: 893, 14: 888, 16: 865, 13: 828, 20: 697, 18: 637, 24: 570, 17: 557, 21: 554, 19: 517, 25: 46
4, 22: 401, 30: 386, 27: 384, 26: 369, 23: 350, 28: 326, 46: 287, 31: 285, 32: 278, 36: 277, 29: 2
73, 33: 268, 35: 262, 42: 259, 53: 238, 34: 223, 40: 219, 39: 212, 37: 198, 38: 192, 43: 181, 45:
177, 44: 175, 54: 172, 57: 167, 48: 163, 52: 159, 41: 157, 50: 156, 55: 155, 47: 147, 51: 147, 49:
146, 60: 144, 56: 124, 63: 122, 67: 122, 64: 113, 61: 107, 72: 107, 59: 99, 70: 98, 62: 94, 84:
94, 68: 92, 65: 91, 69: 91, 58: 89, 76: 88, 71: 87, 77: 87, 66: 83, 74: 80, 79: 80, 81: 80, 78: 7
8, 90: 78, 75: 77, 73: 76, 85: 75, 82: 72, 80: 71, 91: 69, 89: 66, 93: 63, 106: 63, 87: 62, 97: 6
1, 99: 61, 104: 61, 88: 60, 96: 60, 101: 59, 94: 58, 95: 57, 108: 57, 83: 55, 92: 55, 100: 55, 11
1: 55, 126: 55, 86: 54, 113: 54, 110: 53, 114: 52, 135: 52, 98: 51, 103: 48, 109: 48, 120: 48,
102: 46, 107: 46, 105: 45, 117: 44, 119: 44, 112: 43, 140: 43, 115: 42, 122: 41, 129: 41, 136: 41,
139: 40, 124: 39, 132: 39, 154: 39, 116: 38, 128: 38, 150: 37, 121: 36, 123: 36, 138: 35, 145: 35,
118: 34, 147: 34, 148: 34, 159: 34, 125: 33, 137: 32, 127: 31, 149: 31, 161: 31, 180: 31, 141: 30,
144: 30, 130: 29, 131: 29, 134: 29, 146: 29, 191: 29, 133: 28, 142: 28, 152: 28, 153: 28, 169: 28,
175: 28, 189: 28, 156: 27, 168: 27, 177: 27, 178: 27, 193: 27, 143: 26, 164: 26, 166: 26, 197: 26,
160: 25, 162: 25, 170: 25, 207: 24, 167: 23, 174: 23, 183: 23, 222: 23, 223: 23, 242: 23, 158: 22,
171: 22, 184: 22, 206: 22, 209: 22, 244: 22, 155: 21, 157: 21, 163: 21, 187: 21, 194: 21, 195: 21,
199: 21, 202: 21, 215: 21, 228: 21, 310: 21, 165: 20, 172: 20, 173: 20, 176: 20, 188: 20, 210: 20,
212: 20, 234: 20, 240: 20, 151: 19, 179: 19, 185: 19, 201: 19, 208: 19, 229: 19, 252: 19, 271: 19,
181: 18, 186: 18, 204: 18, 216: 18, 218: 18, 224: 18, 236: 18, 253: 18, 256: 18, 265: 18, 276: 18,
214: 17, 217: 17, 235: 17, 245: 17, 257: 17, 279: 17, 293: 17, 203: 16, 205: 16, 213: 16, 237: 16,
250: 16, 263: 16, 270: 16, 282: 16, 301: 16, 305: 16, 200: 15, 219: 15, 221: 15, 225: 15, 182: 14,
196: 14, 198: 14, 211: 14, 220: 14, 230: 14, 246: 14, 249: 14, 258: 14, 274: 14, 281: 14, 300: 14,
302: 14, 311: 14, 336: 14, 347: 14, 226: 13, 227: 13, 233: 13, 248: 13, 254: 13, 262: 13, 267: 13,
272: 13, 303: 13, 304: 13, 306: 13, 312: 13, 331: 13, 413: 13, 456: 13, 287: 13, 192: 12, 241: 12,
247: 12, 288: 12, 291: 12, 326: 12, 345: 12, 563: 12, 259: 11, 260: 11, 273: 11, 278: 11, 280: 11,
283: 11, 286: 11, 289: 11, 294: 11, 316: 11, 327: 11, 335: 11, 348: 11, 350: 11, 354: 11, 360: 11,
367: 11, 388: 11, 417: 11, 429: 11, 232: 10, 238: 10, 239: 10, 261: 10, 269: 10, 277: 10, 285: 10,
292: 10, 318: 10, 319: 10, 332: 10, 340: 10, 400: 10, 408: 10, 430: 10, 484: 10, 373: 10, 243: 9,
251: 9, 255: 9, 266: 9, 290: 9, 296: 9, 307: 9, 309: 9, 314: 9, 315: 9, 320: 9, 321: 9, 324: 9,
329: 9, 334: 9, 337: 9, 341: 9, 342: 9, 346: 9, 355: 9, 358: 9, 377: 9, 403: 9, 419: 9, 433: 9,
447: 9, 462: 9, 482: 9, 231: 8, 295: 8, 299: 8, 323: 8, 325: 8, 333: 8, 344: 8, 351: 8, 353: 8,
363: 8, 366: 8, 371: 8, 372: 8, 374: 8, 378: 8, 381: 8, 390: 8, 402: 8, 405: 8, 415: 8, 427: 8,
441: 8, 463: 8, 501: 8, 502: 8, 620: 8, 480: 8, 570: 8, 190: 7, 264: 7, 268: 7, 284: 7, 328: 7,
330: 7, 356: 7, 362: 7, 368: 7, 376: 7, 382: 7, 383: 7, 384: 7, 389: 7, 412: 7, 422: 7, 423: 7,
425: 7, 431: 7, 437: 7, 444: 7, 449: 7, 457: 7, 460: 7, 461: 7, 469: 7, 481: 7, 488: 7, 496: 7,
507: 7, 528: 7, 551: 7, 556: 7, 581: 7, 596: 7, 608: 7, 610: 7, 612: 7, 616: 7, 673: 7, 695: 7,
735: 7, 1289: 7, 275: 6, 298: 6, 308: 6, 339: 6, 343: 6, 349: 6, 357: 6, 359: 6, 361: 6, 364: 6,
365: 6, 370: 6, 379: 6, 380: 6, 385: 6, 386: 6, 394: 6, 395: 6, 396: 6, 406: 6, 407: 6, 411: 6,
424: 6, 426: 6, 450: 6, 454: 6, 455: 6, 459: 6, 465: 6, 477: 6, 495: 6, 498: 6, 500: 6, 505: 6,
513: 6, 519: 6, 522: 6, 526: 6, 538: 6, 547: 6, 548: 6, 562: 6, 565: 6, 568: 6, 576: 6, 582: 6,
594: 6, 681: 6, 693: 6, 714: 6, 741: 6, 746: 6, 845: 6, 1173: 6, 1191: 6, 313: 5, 317: 5, 352: 5,
369: 5, 391: 5, 392: 5, 393: 5, 397: 5, 409: 5, 414: 5, 418: 5, 420: 5, 421: 5, 428: 5, 434: 5,
448: 5, 451: 5, 452: 5, 464: 5, 466: 5, 470: 5, 476: 5, 479: 5, 485: 5, 487: 5, 493: 5, 497: 5,
504: 5, 509: 5, 510: 5, 515: 5, 518: 5, 524: 5, 529: 5, 530: 5, 536: 5, 539: 5, 549: 5, 555: 5,
571: 5, 578: 5, 585: 5, 604: 5, 621: 5, 632: 5, 636: 5, 647: 5, 654: 5, 657: 5, 663: 5, 691: 5,
697: 5, 707: 5, 721: 5, 754: 5, 768: 5, 797: 5, 846: 5, 945: 5, 964: 5, 972: 5, 994: 5, 995: 5,
1072: 5, 1076: 5, 1158: 5, 1233: 5, 1236: 5, 297: 4, 375: 4, 399: 4, 401: 4, 404: 4, 416: 4, 432: 4
, 436: 4, 438: 4, 439: 4, 443: 4, 446: 4, 468: 4, 472: 4, 473: 4, 474: 4, 475: 4, 483: 4, 486: 4,
491: 4, 503: 4, 511: 4, 512: 4, 520: 4, 521: 4, 523: 4, 534: 4, 537: 4, 540: 4, 541: 4, 544: 4,
545: 4, 546: 4, 558: 4, 561: 4, 573: 4, 575: 4, 577: 4, 584: 4, 587: 4, 591: 4, 592: 4, 600: 4,
601: 4, 609: 4, 613: 4, 614: 4, 619: 4, 623: 4, 624: 4, 626: 4, 633: 4, 635: 4, 641: 4, 648: 4,
649: 4, 653: 4, 655: 4, 665: 4, 666: 4, 680: 4, 687: 4, 690: 4, 698: 4, 699: 4, 700: 4, 701: 4,
702: 4, 711: 4, 712: 4, 717: 4, 730: 4, 732: 4, 737: 4, 739: 4, 778: 4, 780: 4, 781: 4, 782: 4,
795: 4, 801: 4, 802: 4, 807: 4, 811: 4, 813: 4, 819: 4, 827: 4, 829: 4, 836: 4, 844: 4, 871: 4,
876: 4, 877: 4, 970: 4, 1002: 4, 1012: 4, 1127: 4, 1183: 4, 1230: 4, 1253: 4, 1264: 4, 1329: 4, 134
4: 4, 1375: 4, 1495: 4, 1538: 4, 1567: 4, 1612: 4, 1754: 4, 1786: 4, 1795: 4, 756: 4, 322: 3, 338:
3, 387: 3, 435: 3, 445: 3, 453: 3, 458: 3, 471: 3, 489: 3, 492: 3, 506: 3, 508: 3, 514: 3, 517: 3,
525: 3, 527: 3, 531: 3, 533: 3, 535: 3, 542: 3, 564: 3, 566: 3, 569: 3, 572: 3, 579: 3, 586: 3,
593: 3, 597: 3, 598: 3, 606: 3, 617: 3, 625: 3, 637: 3, 645: 3, 646: 3, 651: 3, 658: 3, 659: 3,
661: 3, 662: 3, 664: 3, 667: 3, 674: 3, 677: 3, 678: 3, 684: 3, 689: 3, 694: 3, 706: 3, 710: 3,
713: 3, 722: 3, 748: 3, 749: 3, 750: 3, 751: 3, 758: 3, 765: 3, 766: 3, 775: 3, 785: 3, 788: 3,
790: 3, 791: 3, 803: 3, 806: 3, 809: 3, 814: 3, 815: 3, 820: 3, 821: 3, 822: 3, 830: 3, 834: 3,
835: 3, 837: 3, 866: 3, 868: 3, 872: 3, 873: 3, 898: 3, 911: 3, 915: 3, 916: 3, 920: 3, 930: 3,
933: 3, 936: 3, 939: 3, 949: 3, 957: 3, 959: 3, 962: 3, 966: 3, 973: 3, 977: 3, 990: 3, 991: 3,
1000: 3, 1009: 3, 1017: 3, 1045: 3, 1061: 3, 1069: 3, 1094: 3, 1100: 3, 1110: 3, 1111: 3, 1113: 3,
1117: 3, 1122: 3, 1151: 3, 1160: 3, 1163: 3, 1172: 3, 1180: 3, 1187: 3, 1204: 3, 1238: 3, 1247: 3,
1250: 3, 1252: 3, 1255: 3, 1265: 3, 1266: 3, 1269: 3, 1277: 3, 1286: 3, 1298: 3, 1300: 3, 1311: 3,
1340: 3, 1348: 3, 1365: 3, 1366: 3, 1368: 3, 1475: 3, 1480: 3, 1499: 3, 1557: 3, 1622: 3, 1705: 3,
1708: 3, 1715: 3, 1763: 3, 1764: 3, 1816: 3, 1878: 3, 1931: 3, 2055: 3, 2065: 3, 2078: 3, 2137: 3,
2155: 3, 2269: 3, 2322: 3, 2335: 3, 2440: 3, 516: 3, 3457: 3, 3910: 3, 895: 3, 987: 3, 1234: 3, 166
71: 2, 398: 2, 410: 2, 499: 2, 543: 2, 550: 2, 552: 2, 557: 2, 559: 2, 560: 2, 580: 2, 583: 2,
590: 2, 599: 2, 602: 2, 603: 2, 607: 2, 611: 2, 627: 2, 629: 2, 630: 2, 631: 2, 634: 2, 638: 2,
640: 2, 643: 2, 644: 2, 650: 2, 656: 2, 660: 2, 668: 2, 669: 2, 672: 2, 675: 2, 676: 2, 679: 2,
683: 2, 688: 2, 692: 2, 703: 2, 704: 2, 705: 2, 715: 2, 718: 2, 719: 2, 724: 2, 725: 2, 728: 2,
729: 2, 733: 2, 734: 2, 736: 2, 743: 2, 744: 2, 747: 2, 753: 2, 762: 2, 763: 2, 769: 2, 770: 2,
771: 2, 772: 2, 776: 2, 777: 2, 784: 2, 787: 2, 792: 2, 793: 2, 794: 2, 798: 2, 800: 2, 810: 2,
812: 2, 816: 2, 818: 2, 826: 2, 828: 2, 831: 2, 841: 2, 843: 2, 848: 2, 851: 2, 852: 2, 854: 2,
856: 2, 857: 2, 859: 2, 863: 2, 864: 2, 865: 2, 867: 2, 875: 2, 878: 2, 881: 2, 882: 2, 883: 2,
884: 2, 885: 2, 889: 2, 890: 2, 894: 2, 896: 2, 897: 2, 912: 2, 913: 2, 914: 2, 918: 2, 922: 2,
924: 2, 927: 2, 928: 2, 941: 2, 942: 2, 944: 2, 946: 2, 952: 2, 954: 2, 955: 2, 956: 2, 958: 2,
963: 2, 968: 2, 969: 2, 975: 2, 979: 2, 983: 2, 988: 2, 992: 2, 999: 2, 1003: 2, 1006: 2, 1533: 2,
1008: 2, 1015: 2, 1019: 2, 1022: 2, 1024: 2, 1025: 2, 1027: 2, 1028: 2, 1029: 2, 1036: 2, 1038: 2,
1040: 2, 1041: 2, 1046: 2, 1048: 2, 1049: 2, 1051: 2, 1052: 2, 1054: 2, 1057: 2, 1059: 2, 1063: 2,
1064: 2, 1071: 2, 1073: 2, 1078: 2, 1080: 2, 1097: 2, 1098: 2, 1099: 2, 1549: 2, 1108: 2, 1119: 2,
1120: 2, 1126: 2, 1131: 2, 1134: 2, 1137: 2, 1142: 2, 1144: 2, 1147: 2, 1149: 2, 1153: 2, 1162: 2,
1169: 2, 1179: 2, 1184: 2, 1188: 2, 1189: 2, 1193: 2, 1194: 2, 1198: 2, 1200: 2, 1201: 2, 1212: 2,
1213: 2, 1221: 2, 1223: 2, 1226: 2, 1227: 2, 1231: 2, 1235: 2, 1243: 2, 1257: 2, 1258: 2, 1278: 2,
1281: 2, 1285: 2, 1291: 2, 1293: 2, 1294: 2, 1295: 2, 1301: 2, 1302: 2, 1308: 2, 1312: 2, 1316: 2,
1322: 2, 1326: 2, 1330: 2, 1335: 2, 1336: 2, 1341: 2, 1345: 2, 1347: 2, 1362: 2, 1370: 2, 1371: 2,
1386: 2, 1390: 2, 1391: 2, 1395: 2, 1396: 2, 1404: 2, 1409: 2, 1411: 2, 1420: 2, 1424: 2, 1429: 2,
1436: 2, 1437: 2, 1439: 2, 1443: 2, 1447: 2, 1453: 2, 1461: 2, 1463: 2, 1471: 2, 1473: 2, 1489: 2,
1496: 2, 1497: 2, 1502: 2, 1509: 2, 1510: 2, 1518: 2, 1521: 2, 1522: 2, 1527: 2, 1542: 2, 1580: 2,
1582: 2, 1584: 2, 1589: 2, 1602: 2, 1605: 2, 1608: 2, 1614: 2, 1615: 2, 1617: 2, 1624: 2, 1627: 2,
1628: 2, 1630: 2, 1631: 2, 1635: 2, 1644: 2, 1647: 2, 1653: 2, 1654: 2, 1658: 2, 1676: 2, 1677: 2,
1681: 2, 1689: 2, 1698: 2, 1701: 2, 1719: 2, 1720: 2, 1725: 2, 1737: 2, 1743: 2, 1744: 2, 1761: 2,
1777: 2, 1784: 2, 1789: 2, 1793: 2, 1797: 2, 1803: 2, 1807: 2, 1809: 2, 1830: 2, 1834: 2, 1842: 2,
1853: 2, 1859: 2, 1864: 2, 1885: 2, 1889: 2, 1892: 2, 1900: 2, 1919: 2, 1924: 2, 1927: 2, 1934: 2,
1949: 2, 1955: 2, 1964: 2, 1972: 2, 1985: 2, 1987: 2, 1997: 2, 1998: 2, 2000: 2, 2001: 2, 2010: 2,
2012: 2, 2016: 2, 2024: 2, 2041: 2, 2057: 2, 2085: 2, 2103: 2, 2106: 2, 2114: 2, 2122: 2, 2168: 2,
2171: 2, 2198: 2, 2228: 2, 2263: 2, 2277: 2, 2312: 2, 2317: 2, 2358: 2, 2363: 2, 2367: 2, 2390: 2,
2394: 2, 2402: 2, 2426: 2, 2450: 2, 2466: 2, 2467: 2, 2490: 2, 2494: 2, 2502: 2, 2542: 2, 2547: 2,
2549: 2, 2559: 2, 2609: 2, 2637: 2, 2641: 2, 2647: 2, 2662: 2, 2686: 2, 2768: 2, 2811: 2, 2855: 2,
2874: 2, 2884: 2, 2893: 2, 2905: 2, 3005: 2, 3048: 2, 3052: 2, 3060: 2, 3063: 2, 3087: 2, 532: 2, 3
250: 2, 3316: 2, 3381: 2, 3404: 2, 3488: 2, 3522: 2, 3555: 2, 1961: 2, 3589: 2, 3591: 2, 3593: 2, 3
639: 2, 3695: 2, 3732: 2, 3880: 2, 4006: 2, 4077: 2, 4141: 2, 4276: 2, 4381: 2, 4397: 2, 4419: 2, 4
537: 2, 4744: 2, 13063: 2, 5021: 2, 886: 2, 5362: 2, 5628: 2, 5653: 2, 5734: 2, 6189: 2, 1096: 2, 6
655: 2, 6738: 2, 6906: 2, 1177: 2, 15291: 2, 12128: 2, 1207: 2, 1245: 2, 8254: 1, 442: 1, 467: 1, 4
78: 1, 8672: 1, 490: 1, 494: 1, 8708: 1, 16916: 1, 553: 1, 567: 1, 8762: 1, 588: 1, 589: 1, 595: 1,
605: 1, 615: 1, 618: 1, 622: 1, 639: 1, 642: 1, 652: 1, 670: 1, 682: 1, 685: 1, 709: 1, 716: 1,
720: 1, 723: 1, 726: 1, 738: 1, 740: 1, 742: 1, 752: 1, 755: 1, 8948: 1, 759: 1, 761: 1, 764: 1,
1493: 1, 773: 1, 774: 1, 779: 1, 783: 1, 786: 1, 789: 1, 796: 1, 804: 1, 16518: 1, 823: 1, 824: 1,
825: 1, 833: 1, 838: 1, 839: 1, 842: 1, 847: 1, 9045: 1, 855: 1, 860: 1, 861: 1, 862: 1, 869: 1,
870: 1, 874: 1, 879: 1, 880: 1, 9078: 1, 887: 1, 891: 1, 892: 1, 893: 1, 16533: 1, 901: 1, 902: 1,
903: 1, 905: 1, 17290: 1, 910: 1, 917: 1, 919: 1, 923: 1, 65690: 1, 926: 1, 931: 1, 932: 1, 934: 1
, 935: 1, 937: 1, 943: 1, 947: 1, 948: 1, 950: 1, 951: 1, 953: 1, 960: 1, 961: 1, 965: 1, 967: 1,
974: 1, 976: 1, 9170: 1, 980: 1, 981: 1, 984: 1, 9177: 1, 9179: 1, 989: 1, 993: 1, 998: 1, 1001: 1,
1004: 1, 1005: 1, 1010: 1, 1011: 1, 1013: 1, 1014: 1, 1016: 1, 1020: 1, 1021: 1, 1026: 1, 1030: 1,
1031: 1, 1033: 1, 9226: 1, 1035: 1, 8365: 1, 1042: 1, 1043: 1, 1044: 1, 1047: 1, 1050: 1, 1053: 1,
1055: 1, 1056: 1, 2763: 1, 1066: 1, 1068: 1, 1074: 1, 1077: 1, 1079: 1, 1082: 1, 1083: 1, 1084: 1,
1085: 1, 1086: 1, 1088: 1, 25665: 1, 1090: 1, 1091: 1, 1093: 1, 1095: 1, 9288: 1, 1102: 1, 1104: 1,
24760: 1, 1107: 1, 1109: 1, 1112: 1, 1114: 1, 1116: 1, 1118: 1, 1121: 1, 1125: 1, 1128: 1, 1129: 1,
1133: 1, 1135: 1, 1136: 1, 1138: 1, 1139: 1, 1145: 1, 1146: 1, 1154: 1, 1155: 1, 1156: 1, 9349: 1,
9313: 1, 1161: 1, 1164: 1, 1165: 1, 1167: 1, 9360: 1, 1170: 1, 9363: 1, 1174: 1, 1175: 1, 1176: 1,
9369: 1, 1178: 1, 1181: 1, 1185: 1, 1186: 1, 1190: 1, 1192: 1, 1195: 1, 1196: 1, 1199: 1, 1202: 1,
1203: 1, 1205: 1, 9399: 1, 1210: 1, 1211: 1, 1214: 1, 1216: 1, 1217: 1, 1219: 1, 1222: 1, 1224: 1,
1225: 1, 1229: 1, 1232: 1, 9426: 1, 1237: 1, 1239: 1, 1240: 1, 42205: 1, 1249: 1, 1251: 1, 1254: 1,
1259: 1, 1261: 1, 1262: 1, 1263: 1, 16409: 1, 1270: 1, 1271: 1, 1273: 1, 1274: 1, 1276: 1, 1279: 1,
1280: 1, 1287: 1, 1290: 1, 25872: 1, 1297: 1, 1299: 1, 1303: 1, 1306: 1, 1309: 1, 1310: 1, 9505: 1,
1317: 1, 1318: 1, 1319: 1, 25896: 1, 1323: 1, 1324: 1, 1325: 1, 1327: 1, 1338: 1, 25915: 1, 1349: 1
, 1350: 1, 1351: 1, 1352: 1, 1353: 1, 1354: 1, 1358: 1, 1359: 1, 1360: 1, 1363: 1, 1364: 1, 1372: 1
, 1373: 1, 1374: 1, 1379: 1, 1381: 1, 1384: 1, 1385: 1, 17771: 1, 1388: 1, 1389: 1, 1394: 1, 1398:
1, 1402: 1, 8426: 1, 1406: 1, 1407: 1, 1408: 1, 9604: 1, 1413: 1, 17801: 1, 1419: 1, 1421: 1, 1423:
1, 1425: 1, 1426: 1, 1428: 1, 1433: 1, 1434: 1, 1435: 1, 1442: 1, 1444: 1, 1446: 1, 1449: 1, 1412:
1, 1454: 1, 1455: 1, 1456: 1, 1457: 1, 1459: 1, 1460: 1, 9654: 1, 1464: 1, 1465: 1, 1468: 1, 1470:
1, 1477: 1, 1479: 1, 1481: 1, 1483: 1, 1484: 1, 1485: 1, 1486: 1, 1487: 1, 1490: 1, 1492: 1, 9685:
1, 1494: 1, 1501: 1, 1503: 1, 1504: 1, 1505: 1, 17890: 1, 1508: 1, 17898: 1, 1516: 1, 1519: 1, 1520
: 1, 1525: 1, 1526: 1, 1528: 1, 1529: 1, 1530: 1, 1531: 1, 17917: 1, 1541: 1, 1544: 1, 1546: 1, 154
7: 1, 17933: 1, 1550: 1, 1552: 1, 1556: 1, 1558: 1, 1560: 1, 1561: 1, 1563: 1, 1566: 1, 1569: 1, 15
70: 1, 1571: 1, 1575: 1, 1576: 1, 1578: 1, 1579: 1, 1585: 1, 1586: 1, 1591: 1, 1593: 1, 1595: 1, 97
88: 1, 1599: 1, 1417: 1, 1632: 1, 1606: 1, 1611: 1, 1613: 1, 1616: 1, 1618: 1, 1626: 1, 1629: 1, 98
24: 1, 1633: 1, 1636: 1, 9831: 1, 1639: 1, 1649: 1, 1651: 1, 1652: 1, 1659: 1, 1663: 1, 1666: 1, 16
43: 1, 1669: 1, 1671: 1, 1674: 1, 1678: 1, 1679: 1, 1683: 1, 1684: 1, 1687: 1, 42648: 1, 1692: 1, 1
694: 1, 1697: 1, 1699: 1, 1700: 1, 1703: 1, 1704: 1, 1706: 1, 1707: 1, 1709: 1, 1710: 1, 1711: 1, 1
712: 1, 1714: 1, 1717: 1, 1721: 1, 9914: 1, 1727: 1, 1728: 1, 1732: 1, 8241: 1, 1734: 1, 1738: 1, 1
739: 1, 1745: 1, 1746: 1, 4387: 1, 1750: 1, 1751: 1, 1755: 1, 1756: 1, 1762: 1, 1765: 1, 1766: 1, 1
768: 1, 1769: 1, 1771: 1, 1772: 1, 1773: 1, 1780: 1, 1787: 1, 1790: 1, 1796: 1, 1798: 1, 1799: 1, 1
6684: 1, 1805: 1, 1808: 1, 1810: 1, 1811: 1, 1815: 1, 1818: 1, 1820: 1, 1821: 1, 1822: 1, 1827: 1,
1828: 1, 16689: 1, 1833: 1, 1838: 1, 1841: 1, 1844: 1, 1845: 1, 10039: 1, 1848: 1, 1850: 1, 1852: 1
, 1856: 1, 1857: 1, 1860: 1, 1865: 1, 1866: 1, 1867: 1, 10060: 1, 1871: 1, 1872: 1, 1875: 1, 1877:
1, 1881: 1, 1882: 1, 1883: 1, 1887: 1, 1894: 1, 1896: 1, 1897: 1, 1898: 1, 1899: 1, 10096: 1, 1905:
1, 1908: 1, 1911: 1, 18296: 1, 1916: 1, 1917: 1, 1918: 1, 10113: 1, 10118: 1, 1933: 1, 1935: 1, 193
6: 1, 1937: 1, 1940: 1, 1941: 1, 1944: 1, 1950: 1, 10149: 1, 1958: 1, 10153: 1, 8519: 1, 18353: 1,
1974: 1, 1976: 1, 1979: 1, 1982: 1, 1984: 1, 1991: 1, 1993: 1, 12621: 1, 2003: 1, 2005: 1, 2006: 1,
2008: 1, 2014: 1, 2015: 1, 2017: 1, 2023: 1, 2027: 1, 2032: 1, 2033: 1, 2034: 1, 2037: 1, 2039: 1,
2043: 1, 2045: 1, 2049: 1, 2051: 1, 2058: 1, 2059: 1, 2066: 1, 2079: 1, 2084: 1, 2087: 1, 2088: 1,
2089: 1, 2090: 1, 2092: 1, 2094: 1, 2099: 1, 2101: 1, 2104: 1, 2109: 1, 2111: 1, 2112: 1, 12640: 1,
2119: 1, 5815: 1, 2124: 1, 2125: 1, 2126: 1, 2128: 1, 2129: 1, 2130: 1, 19470: 1, 2134: 1, 2135: 1,
2138: 1, 2140: 1, 1722: 1, 2142: 1, 2143: 1, 2144: 1, 1723: 1, 2158: 1, 2159: 1, 2166: 1, 2167: 1,
2169: 1, 2172: 1, 2175: 1, 18560: 1, 2177: 1, 2178: 1, 2180: 1, 2181: 1, 2184: 1, 2187: 1, 2188: 1,
2189: 1, 2194: 1, 2195: 1, 2196: 1, 2197: 1, 2199: 1, 2201: 1, 2203: 1, 2207: 1, 2210: 1, 2213: 1,
2220: 1, 2222: 1, 2226: 1, 2229: 1, 2230: 1, 2233: 1, 2234: 1, 2235: 1, 2236: 1, 2237: 1, 2245: 1,
2246: 1, 2249: 1, 18636: 1, 10445: 1, 2264: 1, 2265: 1, 2266: 1, 2270: 1, 2275: 1, 2281: 1, 2289: 1
, 2291: 1, 2295: 1, 2302: 1, 2303: 1, 2309: 1, 8577: 1, 2319: 1, 2328: 1, 10530: 1, 2345: 1, 10539:
1, 2348: 1, 2352: 1, 2353: 1, 2354: 1, 2359: 1, 2361: 1, 8262: 1, 2365: 1, 2369: 1, 2373: 1, 2374:
1, 2375: 1, 2377: 1, 2379: 1, 2381: 1, 2389: 1, 2391: 1, 2393: 1, 10588: 1, 2398: 1, 2399: 1, 2403:
1, 2404: 1, 2405: 1, 8593: 1, 2410: 1, 2411: 1, 2413: 1, 2421: 1, 2427: 1, 2428: 1, 2429: 1, 2432:
1, 2433: 1, 2434: 1, 2435: 1, 2436: 1, 2441: 1, 2452: 1, 2455: 1, 2458: 1, 2464: 1, 2465: 1, 2468:
1, 2469: 1, 2475: 1, 2477: 1, 2479: 1, 2480: 1, 2482: 1, 2483: 1, 2484: 1, 2486: 1, 2488: 1, 2489:
1, 3017: 1, 18879: 1, 2497: 1, 2498: 1, 2503: 1, 2506: 1, 2511: 1, 2512: 1, 41379: 1, 2517: 1, 2520
: 1, 2524: 1, 2526: 1, 2529: 1, 2530: 1, 2531: 1, 2532: 1, 2536: 1, 2538: 1, 2539: 1, 2546: 1, 2548
: 1, 2556: 1, 10750: 1, 2560: 1, 2562: 1, 2566: 1, 2569: 1, 8269: 1, 3161: 1, 2584: 1, 2589: 1, 259
1: 1, 2592: 1, 2594: 1, 2596: 1, 10794: 1, 2606: 1, 2608: 1, 2613: 1, 2614: 1, 2617: 1, 2618: 1, 26
23: 1, 10820: 1, 2636: 1, 10832: 1, 2650: 1, 2651: 1, 8801: 1, 2654: 1, 2659: 1, 2661: 1, 2663: 1,
2665: 1, 10002: 1, 8545: 1, 2674: 1, 2676: 1, 2677: 1, 2679: 1, 2682: 1, 41408: 1, 2690: 1, 2694: 1
, 2695: 1, 2696: 1, 2699: 1, 2702: 1, 2712: 1, 2713: 1, 19102: 1, 2721: 1, 2724: 1, 2726: 1, 2729:
1, 2732: 1, 10925: 1, 2735: 1, 10931: 1, 2743: 1, 2745: 1, 2747: 1, 2748: 1, 2755: 1, 2758: 1, 2760
: 1, 19147: 1, 2774: 1, 2783: 1, 2786: 1, 2790: 1, 2808: 1, 2810: 1, 2813: 1, 11005: 1, 2821: 1, 28
27: 1, 2830: 1, 2833: 1, 2840: 1, 2848: 1, 2870: 1, 2873: 1, 2876: 1, 2882: 1, 2888: 1, 2890: 1, 18
47: 1, 2894: 1, 2899: 1, 2902: 1, 2904: 1, 3215: 1, 2909: 1, 2912: 1, 2919: 1, 2920: 1, 2921: 1, 11
119: 1, 2931: 1, 2932: 1, 2933: 1, 2948: 1, 2952: 1, 2953: 1, 2961: 1, 2974: 1, 2980: 1, 2982: 1, 2
988: 1, 2990: 1, 2994: 1, 2996: 1, 2997: 1, 19387: 1, 3006: 1, 11199: 1, 3010: 1, 1868: 1, 3019: 1,
3028: 1, 3030: 1, 3035: 1, 8698: 1, 3040: 1, 3043: 1, 3046: 1, 3056: 1, 3061: 1, 35833: 1, 3070: 1,
3075: 1, 3084: 1, 3085: 1, 3086: 1, 11295: 1, 3104: 1, 3106: 1, 3110: 1, 3111: 1, 3117: 1, 3118: 1,
3128: 1, 3129: 1, 3139: 1, 3141: 1, 3144: 1, 3147: 1, 3154: 1, 3155: 1, 3156: 1, 3158: 1, 11353: 1,
3164: 1, 3166: 1, 3170: 1, 3171: 1, 3175: 1, 3176: 1, 3182: 1, 11395: 1, 3207: 1, 11403: 1, 3214: 1
, 11407: 1, 8728: 1, 3219: 1, 8729: 1, 3228: 1, 3231: 1, 1904: 1, 3235: 1, 3239: 1, 3240: 1, 3249:
1, 3252: 1, 3253: 1, 3256: 1, 3258: 1, 3262: 1, 8292: 1, 11461: 1, 3270: 1, 3274: 1, 3283: 1, 3285:
1, 3287: 1, 3288: 1, 3290: 1, 3295: 1, 3296: 1, 3307: 1, 3308: 1, 11502: 1, 3311: 1, 27897: 1, 3323
: 1, 3324: 1, 3326: 1, 3332: 1, 36105: 1, 3338: 1, 11538: 1, 3349: 1, 3353: 1, 8751: 1, 8753: 1, 11
563: 1, 3377: 1, 3378: 1, 3379: 1, 3382: 1, 3383: 1, 3401: 1, 3409: 1, 3410: 1, 3412: 1, 3418: 1, 3
423: 1, 3424: 1, 3431: 1, 3432: 1, 3436: 1, 3448: 1, 3450: 1, 3451: 1, 3452: 1, 3454: 1, 3458: 1, 3
463: 1, 3467: 1, 3469: 1, 3475: 1, 3310: 1, 3480: 1, 3482: 1, 3483: 1, 3490: 1, 3494: 1, 3495: 1, 3
497: 1, 19883: 1, 3500: 1, 3507: 1, 3513: 1, 11712: 1, 3529: 1, 3530: 1, 3534: 1, 3535: 1, 3541: 1,
3543: 1, 3548: 1, 3556: 1, 3564: 1, 3566: 1, 11762: 1, 11763: 1, 3577: 1, 3582: 1, 3605: 1, 11799:
1, 3609: 1, 3610: 1, 3614: 1, 3616: 1, 3618: 1, 3620: 1, 3622: 1, 1969: 1, 3626: 1, 3627: 1, 11833:
```

```
1, 12895: 1, 3651: 1, 3655: 1, 44623: 1, 3672: 1, 3676: 1, 3684: 1, 3691: 1, 3692: 1, 3702: 1, 3703
: 1, 11898: 1, 3715: 1, 3717: 1, 3718: 1, 3720: 1, 3721: 1, 8813: 1, 3733: 1, 36505: 1, 3743: 1, 33
55: 1, 3749: 1, 3751: 1, 3755: 1, 3757: 1, 3759: 1, 3761: 1, 3767: 1, 11962: 1, 3772: 1, 11965: 1,
28357: 1, 3787: 1, 20179: 1, 3800: 1, 3802: 1, 3811: 1, 3817: 1, 3820: 1, 3824: 1, 12018: 1, 3830:
1, 3833: 1, 3840: 1, 3855: 1, 12062: 1, 3871: 1, 12065: 1, 3874: 1, 3879: 1, 3900: 1, 3903: 1, 3915
: 1, 3920: 1, 3932: 1, 3934: 1, 3936: 1, 3938: 1, 3943: 1, 3944: 1, 3952: 1, 12154: 1, 3971: 1, 397
2: 1, 8854: 1, 3974: 1, 20359: 1, 17047: 1, 12175: 1, 3985: 1, 3986: 1, 3996: 1, 3999: 1, 4003: 1,
4005: 1, 12205: 1, 4014: 1, 4023: 1, 4031: 1, 4032: 1, 12229: 1, 4038: 1, 12962: 1, 12238: 1, 12241
: 1, 4051: 1, 4055: 1, 4057: 1, 4059: 1, 4064: 1, 4066: 1, 4067: 1, 4076: 1, 4078: 1, 4085: 1, 4087
: 1, 4088: 1, 12281: 1, 4097: 1, 4113: 1, 18435: 1, 12309: 1, 12318: 1, 12319: 1, 4145: 1, 4146: 1,
4148: 1, 4150: 1, 4153: 1, 4160: 1, 4164: 1, 4166: 1, 4169: 1, 4176: 1, 4181: 1, 4203: 1, 8893: 1,
12409: 1, 4232: 1, 4235: 1, 4238: 1, 4239: 1, 4242: 1, 4244: 1, 4261: 1, 4262: 1, 4266: 1, 1506: 1,
17873: 1, 4275: 1, 4283: 1, 4286: 1, 4287: 1, 4289: 1, 4305: 1, 4312: 1, 4318: 1, 12514: 1, 4328: 1
, 4346: 1, 4353: 1, 12553: 1, 4368: 1, 4373: 1, 4374: 1, 4386: 1, 20771: 1, 12591: 1, 4407: 1, 4408
: 1, 4409: 1, 4417: 1, 4420: 1, 4427: 1, 4429: 1, 4441: 1, 20832: 1, 12647: 1, 4456: 1, 4465: 1, 44
72: 1, 3477: 1, 4484: 1, 39692: 1, 12690: 1, 4501: 1, 4510: 1, 11675: 1, 4522: 1, 8946: 1, 4535: 1,
4539: 1, 4557: 1, 4561: 1, 12774: 1, 4600: 1, 4603: 1, 4613: 1, 4614: 1, 25345: 1, 4619: 1, 12813:
1, 4628: 1, 4644: 1, 4650: 1, 148232: 1, 29235: 1, 4664: 1, 4667: 1, 53830: 1, 4680: 1, 4686: 1, 46
89: 1, 66318: 1, 4696: 1, 4703: 1, 4725: 1, 21110: 1, 12926: 1, 4747: 1, 119436: 1, 4749: 1, 12950:
1, 4759: 1, 4766: 1, 4770: 1, 4771: 1, 4780: 1, 4783: 1, 49950: 1, 4793: 1, 4796: 1, 4803: 1, 4808:
1, 4810: 1, 39033: 1, 4825: 1, 4826: 1, 4827: 1, 4831: 1, 4840: 1, 4841: 1, 4845: 1, 45817: 1, 4858
: 1, 13052: 1, 2176: 1, 45870: 1, 4892: 1, 4894: 1, 4902: 1, 37678: 1, 9011: 1, 4921: 1, 13117: 1,
4928: 1, 4933: 1, 4938: 1, 4942: 1, 4954: 1, 4968: 1, 9021: 1, 4994: 1, 5014: 1, 5030: 1, 3570: 1,
5054: 1, 3575: 1, 5070: 1, 5079: 1, 5083: 1, 5090: 1, 5098: 1, 5099: 1, 5111: 1, 5115: 1, 5125: 1,
5130: 1, 5158: 1, 5171: 1, 5174: 1, 5182: 1, 5206: 1, 5225: 1, 5230: 1, 5239: 1, 5242: 1, 5245: 1,
5257: 1, 5269: 1, 5281: 1, 5291: 1, 5292: 1, 5298: 1, 5301: 1, 5304: 1, 5309: 1, 5328: 1, 9725: 1,
5342: 1, 21728: 1, 21748: 1, 9086: 1, 5367: 1, 5374: 1, 5376: 1, 13574: 1, 13591: 1, 5403: 1, 5412:
1, 5424: 1, 906: 1, 13653: 1, 5462: 1, 5467: 1, 3643: 1, 9106: 1, 33684: 1, 21886: 1, 21398: 1,
5519: 1, 5521: 1, 38291: 1, 5543: 1, 925: 1, 5556: 1, 5560: 1, 5578: 1, 54751: 1, 17320: 1, 5620: 1
, 5626: 1, 5642: 1, 5648: 1, 5673: 1, 5678: 1, 5682: 1, 63042: 1, 13894: 1, 5703: 1, 5704: 1, 5711:
1, 5723: 1, 22111: 1, 5740: 1, 5742: 1, 5750: 1, 5752: 1, 5764: 1, 5774: 1, 5776: 1, 9157: 1, 9161:
1, 5832: 1, 5833: 1, 5849: 1, 63194: 1, 5866: 1, 978: 1, 5883: 1, 5908: 1, 14101: 1, 79638: 1, 985:
1, 5925: 1, 5942: 1, 5947: 1, 14140: 1, 65666: 1, 5967: 1, 6000: 1, 9193: 1, 6013: 1, 6020: 1, 1007
: 1, 14245: 1, 6060: 1, 14255: 1, 6068: 1, 6085: 1, 14278: 1, 6100: 1, 16570: 1, 6108: 1, 8224: 1,
6140: 1, 6146: 1, 6150: 1, 6154: 1, 6155: 1, 6157: 1, 14352: 1, 18777: 1, 14377: 1, 6187: 1, 6201:
1, 1034: 1, 6210: 1, 6215: 1, 6216: 1, 1039: 1, 6238: 1, 6241: 1, 2407: 1, 6253: 1, 6254: 1, 6258:
1, 6265: 1, 8225: 1, 6302: 1, 6321: 1, 6327: 1, 6337: 1, 22723: 1, 6341: 1, 14545: 1, 6367: 1, 6373
: 1, 9258: 1, 6420: 1, 6463: 1, 6474: 1, 6489: 1, 6495: 1, 14712: 1, 6531: 1, 1089: 1, 6542: 1, 147
43: 1, 6571: 1, 6572: 1, 6574: 1, 6589: 1, 6590: 1, 6597: 1, 9292: 1, 6604: 1, 6638: 1, 6656: 1, 17
494: 1, 6680: 1, 31265: 1, 6699: 1, 14892: 1, 6707: 1, 6713: 1, 6717: 1, 39495: 1, 14924: 1, 9647:
1, 6812: 1, 6814: 1, 6815: 1, 6859: 1, 15073: 1, 6889: 1, 6890: 1, 6891: 1, 15089: 1, 6914: 1, 6924
: 1, 6928: 1, 6929: 1, 15122: 1, 15132: 1, 6962: 1, 6963: 1, 7004: 1, 7006: 1, 7007: 1, 7013: 1, 11
71: 1, 7057: 1, 7086: 1, 10741: 1, 7105: 1, 7110: 1, 7120: 1, 7123: 1, 7133: 1, 66727: 1, 10058: 1,
7191: 1, 15400: 1, 7210: 1, 23605: 1, 7283: 1, 23688: 1, 7306: 1, 7310: 1, 7312: 1, 1062: 1, 7327:
1, 7340: 1, 7344: 1, 7346: 1, 7372: 1, 15577: 1, 7386: 1, 15595: 1, 9427: 1, 7430: 1, 7434: 1, 7436
: 1, 15632: 1, 7447: 1, 8342: 1, 7463: 1, 7467: 1, 15676: 1, 7521: 1, 7527: 1, 8435: 1, 7565: 1, 40
369: 1, 15807: 1, 7618: 1, 7621: 1, 7623: 1, 7667: 1, 24053: 1, 32252: 1, 7684: 1, 7705: 1, 7706: 1
, 7707: 1, 7709: 1, 24121: 1, 32317: 1, 24157: 1, 1296: 1, 48739: 1, 805: 1, 7802: 1, 2669: 1, 2671
: 1, 7839: 1, 7842: 1, 7865: 1, 7867: 1, 1313: 1, 4045: 1, 25893: 1, 4049: 1, 7924: 1, 32502: 1, 79
38: 1, 2689: 1, 17710: 1, 7981: 1, 7985: 1, 7986: 1, 16185: 1, 8024: 1, 8056: 1, 32653: 1, 8088: 1,
24474: 1, 8093: 1, 32681: 1, 8111: 1, 16637: 1, 8119: 1, 8131: 1, 8156: 1})
```

6.2 Machine Learning model

In [112]:

```python
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 [113]:

```
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']))
```

In [114]:

```
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, 55354)
(number of data points * number of features) in test data =  (665, 55354)
(number of data points * number of features) in cross validation data = (532, 55354)
```

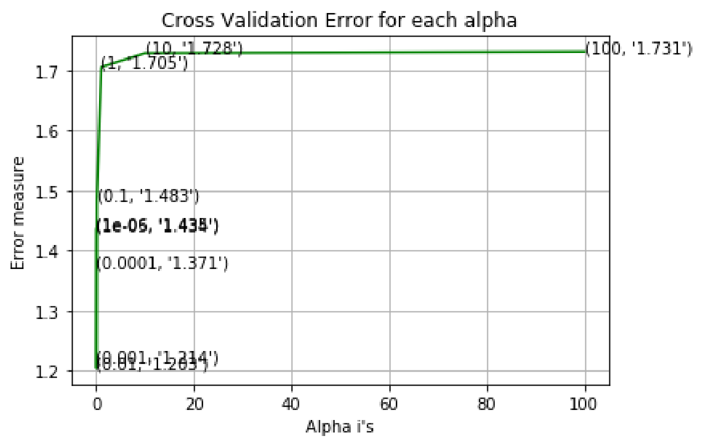
6.2.1 Logistic Regression

6.2.1.1 Hyper paramter tuning

In [117]:

```
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 : 1.4350990724314585
for alpha = 1e-05
Log Loss : 1.434291767498118
for alpha = 0.0001
Log Loss : 1.370805310805254
for alpha = 0.001
Log Loss : 1.2142087413175042
for alpha = 0.01
Log Loss : 1.2033810663566311
for alpha = 0.1
Log Loss : 1.483494612674095
for alpha = 1
Log Loss : 1.7051644370896655
for alpha = 10
Log Loss : 1.7283293343689994
for alpha = 100
Log Loss : 1.7307018566440107
```



```
For values of best alpha =  0.01 The train log loss is: 0.8057610643901808
For values of best alpha =  0.01 The cross validation log loss is: 1.2033810663566311
For values of best alpha =  0.01 The test log loss is: 1.1143023989870882
```

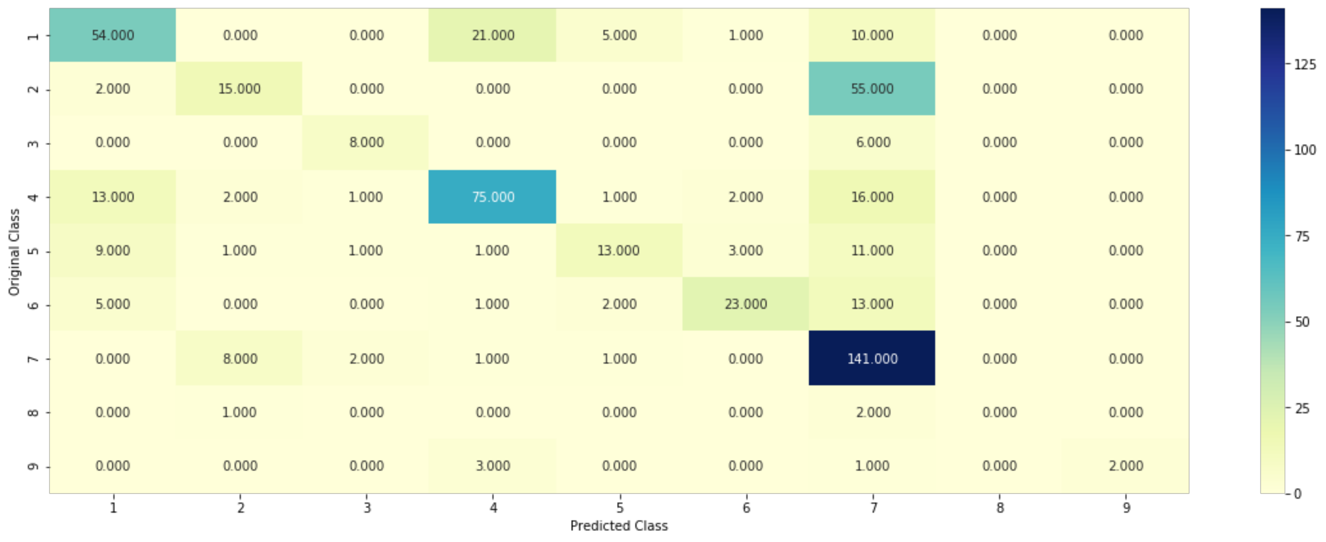
6.2.1.2 Testing the model with best hyper paramters

3.2.1.2 Testing the model with best hyper paramters

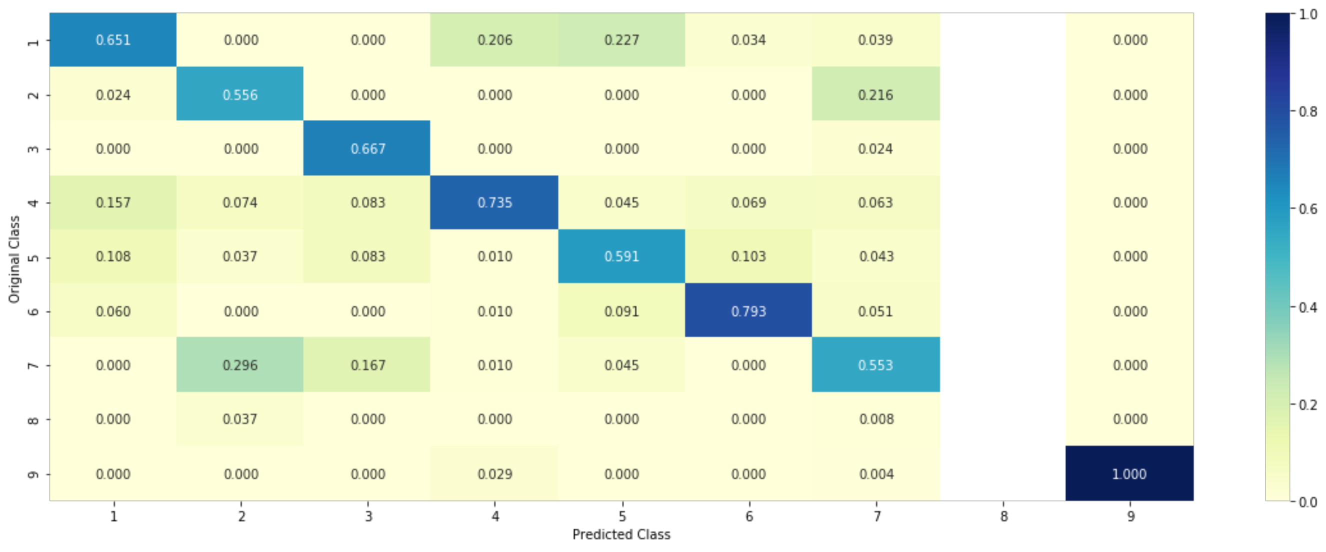
In [119]:

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

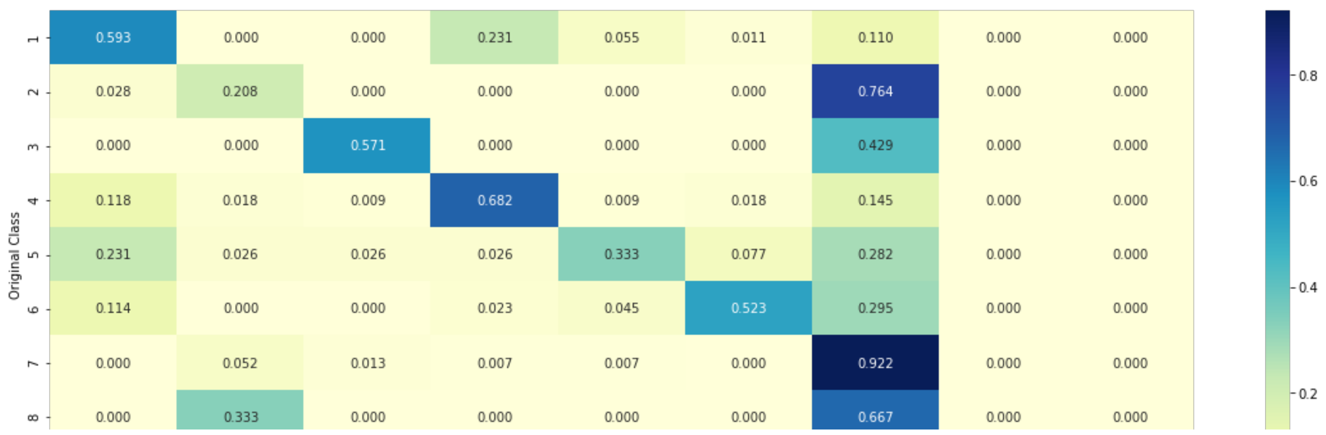
```
Log loss : 1.2033810663566311
Number of mis-classified points : 0.37781954887218044
-------------------- Confusion matrix --------------------
```

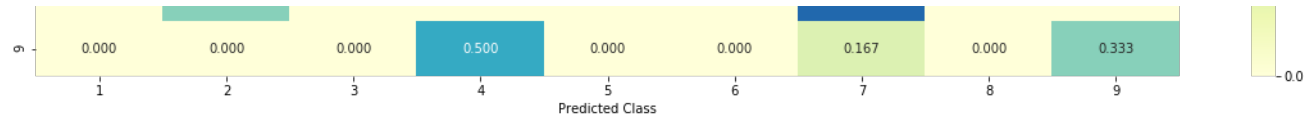


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





6.2.1.3 Feature Importance

In [121]:

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
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: [[0.022  0.0332 0.0043 0.0317 0.0138 0.0065 0.8831 0.0043 0.0012]]
Actual Class : 7
--------------------------------------------------
13 Text feature [constitutive] present in test data point [True]
20 Text feature [ligand] present in test data point [True]
31 Text feature [nude] present in test data point [True]
40 Text feature [egf] present in test data point [True]
44 Text feature [activated] present in test data point [True]
54 Text feature [constitutively] present in test data point [True]
56 Text feature [extracellular] present in test data point [True]
62 Text feature [pertuzumab] present in test data point [True]
73 Text feature [balb] present in test data point [True]
90 Text feature [therapeutics] present in test data point [True]
112 Text feature [lung] present in test data point [True]
113 Text feature [oncogenes] present in test data point [True]
117 Text feature [implanted] present in test data point [True]
120 Text feature [receptors] present in test data point [True]
143 Text feature [amplification] present in test data point [True]
146 Text feature [tyrosine] present in test data point [True]
153 Text feature [pulp] present in test data point [True]
180 Text feature [transformed] present in test data point [True]
183 Text feature [il] present in test data point [True]
193 Text feature [jeong] present in test data point [True]
194 Text feature [independence] present in test data point [True]
200 Text feature [transforming] present in test data point [True]
207 Text feature [macdonald] present in test data point [True]
216 Text feature [yarden] present in test data point [True]
310 Text feature [infiltrating] present in test data point [True]
321 Text feature [ocum] present in test data point [True]
322 Text feature [neuregulins] present in test data point [True]
328 Text feature [adenocarcinoma] present in test data point [True]
335 Text feature [prognostic] present in test data point [True]
349 Text feature [phosphotransferase] present in test data point [True]
351 Text feature [transform] present in test data point [True]
365 Text feature [downstream] present in test data point [True]
433 Text feature [1ivo] present in test data point [True]
436 Text feature [receptor] present in test data point [True]
439 Text feature [sliwkowski] present in test data point [True]
443 Text feature [transformation] present in test data point [True]
464 Text feature [oncogenic] present in test data point [True]
499 Text feature [pinkas] present in test data point [True]
Out of the top  500  features  38 are present in query point
```

In [122]:

```
test_point_index = 107
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: [[0.2449 0.1252 0.0028 0.1019 0.0157 0.0017 0.4808 0.0257 0.0013]]
Actual Class : 8
--------------------------------------------------
13 Text feature [constitutive] present in test data point [True]
27 Text feature [subcutaneously] present in test data point [True]
31 Text feature [nude] present in test data point [True]
43 Text feature [stat] present in test data point [True]
44 Text feature [activated] present in test data point [True]
51 Text feature [murine] present in test data point [True]
54 Text feature [constitutively] present in test data point [True]
69 Text feature [eco] present in test data point [True]
90 Text feature [therapeutics] present in test data point [True]
112 Text feature [lung] present in test data point [True]
113 Text feature [oncogenes] present in test data point [True]
120 Text feature [receptors] present in test data point [True]
129 Text feature [malignant] present in test data point [True]
133 Text feature [gains] present in test data point [True]
143 Text feature [amplification] present in test data point [True]
180 Text feature [transformed] present in test data point [True]
183 Text feature [il] present in test data point [True]
199 Text feature [iiic] present in test data point [True]
246 Text feature [strongest] present in test data point [True]
252 Text feature [fos] present in test data point [True]
335 Text feature [prognostic] present in test data point [True]
365 Text feature [downstream] present in test data point [True]
436 Text feature [receptor] present in test data point [True]
443 Text feature [transformation] present in test data point [True]
449 Text feature [rarely] present in test data point [True]
459 Text feature [immunoglobulin] present in test data point [True]
464 Text feature [oncogenic] present in test data point [True]
478 Text feature [advanced] present in test data point [True]
486 Text feature [grew] present in test data point [True]
Out of the top  500  features  29 are present in query point
```

6.3 Bigrams

In [124]:

```
# 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))
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 : 773888
```

In [125]:

```
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 [126]:

```
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 [127]:

```
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 [128]:

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

```
Counter({3: 158976, 4: 100355, 5: 64807, 6: 58811, 7: 46809, 8: 38617, 9: 35660, 10: 30984, 12: 21
249, 11: 18232, 13: 13440, 14: 12553, 15: 11464, 16: 10333, 20: 8316, 18: 7607, 17: 7497, 19: 7088
, 24: 5903, 21: 5300, 25: 4511, 22: 4426, 30: 4267, 26: 4125, 27: 3704, 23: 3556, 46: 3015, 28: 29
17, 31: 2503, 29: 2390, 42: 2310, 32: 2252, 53: 2170, 33: 1970, 35: 1862, 34: 1842, 36: 1817, 40:
1510, 37: 1427, 38: 1419, 67: 1392, 55: 1369, 39: 1365, 48: 1191, 43: 1173, 41: 1161, 47: 1116, 45
: 1053, 50: 1040, 44: 1029, 49: 974, 54: 954, 52: 870, 51: 853, 60: 821, 57: 819, 56: 809, 58: 681
, 63: 621, 59: 611, 62: 585, 68: 579, 61: 570, 69: 546, 64: 543, 84: 541, 66: 530, 70: 528, 72: 50
2, 65: 499, 71: 466, 73: 431, 75: 417, 76: 415, 74: 407, 78: 387, 81: 378, 79: 374, 77: 361, 80: 3
45, 90: 337, 83: 320, 86: 316, 82: 309, 85: 301, 87: 283, 92: 280, 93: 276, 88: 273, 91: 272, 89:
255, 97: 254, 106: 253, 94: 234, 95: 230, 99: 225, 101: 225, 100: 217, 110: 217, 96: 213, 104: 200
, 108: 197, 98: 195, 102: 195, 105: 195, 111: 193, 109: 188, 113: 187, 126: 187, 112: 184, 107: 18
3, 103: 173, 120: 170, 115: 157, 114: 155, 116: 154, 134: 154, 129: 149, 119: 145, 135: 142, 117:
138, 122: 137, 130: 137, 127: 135, 128: 135, 125: 133, 136: 133, 124: 132, 121: 131, 123: 131, 118
: 125, 138: 125, 140: 123, 133: 122, 131: 118, 137: 115, 144: 112, 132: 111, 141: 108, 150: 105, 1
45: 100, 147: 100, 143: 98, 146: 98, 159: 98, 139: 96, 154: 96, 142: 94, 153: 94, 149: 89, 151: 89
, 148: 87, 156: 87, 166: 84, 168: 83, 164: 82, 152: 80, 155: 80, 160: 80, 161: 80, 162: 80, 175: 7
9, 158: 78, 165: 78, 179: 77, 180: 76, 191: 76, 172: 75, 174: 73, 157: 70, 170: 70, 171: 70, 193:
70, 163: 67, 167: 67, 173: 67, 184: 66, 186: 65, 197: 65, 178: 64, 195: 62, 189: 60, 169: 59, 182:
59, 202: 59, 176: 58, 199: 56, 177: 55, 181: 54, 185: 54, 194: 53, 206: 53, 183: 52, 207: 52, 212:
52, 209: 50, 265: 50, 204: 49, 192: 48, 200: 48, 223: 48, 187: 47, 225: 47, 196: 46, 198: 46, 205:
46, 208: 45, 210: 45, 222: 45, 228: 45, 236: 45, 188: 44, 214: 44, 190: 43, 203: 43, 216: 43, 234:
43, 242: 43, 201: 42, 213: 42, 224: 42, 245: 42, 252: 42, 233: 41, 215: 40, 217: 40, 221: 40, 229:
40, 231: 40, 256: 40, 227: 39, 270: 39, 218: 38, 219: 38, 235: 38, 239: 37, 240: 37, 244: 37, 248:
37, 238: 36, 253: 36, 293: 36, 211: 35, 246: 35, 250: 35, 271: 35, 230: 34, 267: 34, 272: 33, 301:
33, 310: 33, 220: 32, 232: 32, 257: 32, 258: 32, 279: 32, 259: 31, 290: 31, 237: 30, 280: 30, 287:
30, 288: 30, 247: 29, 254: 29, 261: 29, 276: 29, 281: 29, 251: 28, 255: 28, 260: 28, 263: 28, 273:
28, 274: 28, 300: 28, 302: 28, 305: 28, 336: 28, 226: 27, 282: 27, 289: 27, 306: 27, 312: 27, 241:
26, 294: 26, 325: 26, 243: 25, 264: 25, 278: 25, 291: 25, 304: 25, 311: 25, 326: 25, 277: 24, 292:
24, 303: 24, 390: 24, 262: 23, 283: 23, 307: 23, 314: 23, 320: 23, 249: 22, 266: 22, 284: 22, 286:
```

```
22, 328: 22, 345: 22, 379: 22, 268: 21, 269: 21, 299: 21, 329: 21, 348: 21, 350: 21, 296: 20, 315:
20, 316: 20, 318: 20, 319: 20, 327: 20, 331: 20, 332: 20, 337: 20, 340: 20, 342: 20, 354: 20, 285:
19, 330: 19, 334: 19, 335: 19, 347: 19, 353: 19, 372: 19, 378: 19, 408: 19, 413: 19, 417: 19, 275:
18, 295: 18, 298: 18, 309: 18, 321: 18, 324: 18, 333: 18, 351: 18, 365: 18, 371: 18, 376: 18, 383:
18, 388: 18, 405: 18, 429: 18, 297: 17, 313: 17, 363: 17, 367: 17, 380: 17, 456: 17, 460: 17, 344:
16, 346: 16, 355: 16, 364: 16, 400: 16, 403: 16, 419: 16, 462: 16, 323: 15, 349: 15, 357: 15, 358:
15, 368: 15, 377: 15, 412: 15, 431: 15, 449: 15, 450: 15, 308: 14, 322: 14, 338: 14, 359: 14, 360:
14, 361: 14, 370: 14, 381: 14, 382: 14, 426: 14, 430: 14, 436: 14, 447: 14, 341: 13, 356: 13, 362:
13, 366: 13, 373: 13, 384: 13, 407: 13, 421: 13, 422: 13, 459: 13, 466: 13, 469: 13, 475: 13, 477:
13, 488: 13, 563: 13, 339: 12, 374: 12, 391: 12, 394: 12, 398: 12, 402: 12, 404: 12, 414: 12, 420:
12, 427: 12, 433: 12, 441: 12, 465: 12, 480: 12, 484: 12, 493: 12, 501: 12, 565: 12, 570: 12, 673:
12, 735: 12, 317: 11, 343: 11, 392: 11, 393: 11, 395: 11, 397: 11, 411: 11, 415: 11, 425: 11, 437:
11, 443: 11, 451: 11, 454: 11, 481: 11, 496: 11, 500: 11, 515: 11, 525: 11, 551: 11, 576: 11, 581:
11, 401: 11, 369: 10, 389: 10, 399: 10, 406: 10, 416: 10, 423: 10, 424: 10, 428: 10, 434: 10, 444:
10, 457: 10, 458: 10, 461: 10, 471: 10, 476: 10, 482: 10, 495: 10, 498: 10, 502: 10, 504: 10, 507:
10, 518: 10, 519: 10, 522: 10, 541: 10, 549: 10, 594: 10, 616: 10, 620: 10, 621: 10, 352: 9, 386:
9, 396: 9, 438: 9, 445: 9, 452: 9, 455: 9, 463: 9, 464: 9, 470: 9, 503: 9, 509: 9, 528: 9, 530: 9,
533: 9, 556: 9, 577: 9, 582: 9, 585: 9, 596: 9, 612: 9, 624: 9, 654: 9, 664: 9, 737: 9, 539: 9,
385: 8, 409: 8, 432: 8, 448: 8, 474: 8, 485: 8, 486: 8, 489: 8, 505: 8, 512: 8, 513: 8, 517: 8,
523: 8, 526: 8, 532: 8, 535: 8, 536: 8, 544: 8, 545: 8, 546: 8, 548: 8, 555: 8, 562: 8, 564: 8,
572: 8, 575: 8, 583: 8, 592: 8, 608: 8, 610: 8, 613: 8, 633: 8, 656: 8, 680: 8, 684: 8, 693: 8,
699: 8, 754: 8, 795: 8, 1106: 8, 375: 7, 387: 7, 410: 7, 418: 7, 439: 7, 446: 7, 453: 7, 468: 7,
472: 7, 479: 7, 497: 7, 510: 7, 511: 7, 521: 7, 527: 7, 529: 7, 537: 7, 547: 7, 560: 7, 578: 7,
584: 7, 598: 7, 614: 7, 637: 7, 641: 7, 648: 7, 650: 7, 657: 7, 663: 7, 681: 7, 689: 7, 695: 7,
697: 7, 702: 7, 707: 7, 710: 7, 712: 7, 717: 7, 730: 7, 739: 7, 746: 7, 750: 7, 768: 7, 845: 7,
846: 7, 898: 7, 1076: 7, 1289: 7, 473: 6, 478: 6, 483: 6, 487: 6, 499: 6, 506: 6, 520: 6, 524: 6,
531: 6, 538: 6, 540: 6, 542: 6, 543: 6, 550: 6, 568: 6, 571: 6, 587: 6, 591: 6, 593: 6, 600: 6,
601: 6, 603: 6, 604: 6, 617: 6, 635: 6, 640: 6, 647: 6, 655: 6, 658: 6, 665: 6, 666: 6, 687: 6,
690: 6, 691: 6, 713: 6, 714: 6, 741: 6, 743: 6, 748: 6, 802: 6, 815: 6, 816: 6, 819: 6, 822: 6,
837: 6, 945: 6, 972: 6, 994: 6, 1173: 6, 1191: 6, 440: 5, 491: 5, 494: 5, 534: 5, 557: 5, 558: 5,
559: 5, 566: 5, 569: 5, 573: 5, 579: 5, 602: 5, 609: 5, 611: 5, 618: 5, 623: 5, 626: 5, 632: 5,
636: 5, 643: 5, 646: 5, 649: 5, 662: 5, 667: 5, 669: 5, 677: 5, 678: 5, 688: 5, 698: 5, 700: 5,
701: 5, 711: 5, 715: 5, 721: 5, 729: 5, 756: 5, 765: 5, 775: 5, 778: 5, 780: 5, 785: 5, 788: 5,
797: 5, 809: 5, 814: 5, 842: 5, 867: 5, 871: 5, 877: 5, 905: 5, 918: 5, 955: 5, 962: 5, 964: 5,
995: 5, 1012: 5, 1072: 5, 1158: 5, 1233: 5, 1236: 5, 1264: 5, 1286: 5, 1612: 5, 561: 5, 936: 5, 435
: 4, 442: 4, 467: 4, 492: 4, 508: 4, 514: 4, 554: 4, 586: 4, 597: 4, 599: 4, 605: 4, 606: 4, 607:
4, 619: 4, 625: 4, 627: 4, 630: 4, 631: 4, 642: 4, 651: 4, 653: 4, 660: 4, 661: 4, 668: 4, 672: 4,
674: 4, 682: 4, 703: 4, 706: 4, 720: 4, 722: 4, 723: 4, 724: 4, 726: 4, 732: 4, 734: 4, 742: 4,
745: 4, 749: 4, 751: 4, 753: 4, 761: 4, 763: 4, 766: 4, 769: 4, 774: 4, 779: 4, 781: 4, 782: 4,
790: 4, 801: 4, 806: 4, 807: 4, 810: 4, 811: 4, 813: 4, 825: 4, 826: 4, 827: 4, 829: 4, 830: 4,
835: 4, 836: 4, 844: 4, 848: 4, 851: 4, 852: 4, 864: 4, 866: 4, 868: 4, 872: 4, 876: 4, 895: 4,
896: 4, 911: 4, 920: 4, 925: 4, 934: 4, 939: 4, 949: 4, 959: 4, 963: 4, 969: 4, 970: 4, 973: 4,
977: 4, 978: 4, 984: 4, 987: 4, 997: 4, 1000: 4, 1002: 4, 1009: 4, 1017: 4, 1019: 4, 1020: 4, 1035:
4, 1045: 4, 1046: 4, 1057: 4, 1084: 4, 1110: 4, 1127: 4, 1134: 4, 1144: 4, 1151: 4, 1163: 4, 1180:
4, 1183: 4, 1187: 4, 1193: 4, 1204: 4, 1213: 4, 1230: 4, 1253: 4, 1255: 4, 1311: 4, 1329: 4, 1344:
4, 1375: 4, 1424: 4, 1475: 4, 1480: 4, 1495: 4, 1502: 4, 1538: 4, 1567: 4, 1708: 4, 1754: 4, 1764:
4, 1786: 4, 1795: 4, 886: 4, 1100: 4, 490: 3, 516: 3, 567: 3, 574: 3, 588: 3, 589: 3, 590: 3, 615:
3, 629: 3, 634: 3, 638: 3, 639: 3, 645: 3, 652: 3, 659: 3, 676: 3, 679: 3, 683: 3, 685: 3, 692: 3,
694: 3, 696: 3, 704: 3, 705: 3, 716: 3, 728: 3, 733: 3, 736: 3, 738: 3, 744: 3, 747: 3, 752: 3,
755: 3, 758: 3, 764: 3, 767: 3, 770: 3, 772: 3, 787: 3, 791: 3, 794: 3, 803: 3, 812: 3, 820: 3,
821: 3, 831: 3, 834: 3, 839: 3, 843: 3, 854: 3, 856: 3, 857: 3, 859: 3, 862: 3, 863: 3, 870: 3,
873: 3, 878: 3, 881: 3, 884: 3, 885: 3, 892: 3, 897: 3, 910: 3, 912: 3, 915: 3, 916: 3, 922: 3,
924: 3, 927: 3, 928: 3, 930: 3, 933: 3, 935: 3, 942: 3, 946: 3, 952: 3, 954: 3, 956: 3, 957: 3,
966: 3, 988: 3, 990: 3, 991: 3, 998: 3, 1003: 3, 1006: 3, 1008: 3, 1022: 3, 1024: 3, 1027: 3, 1028:
3, 1038: 3, 1041: 3, 1048: 3, 1052: 3, 1061: 3, 1062: 3, 1063: 3, 1069: 3, 1071: 3, 1073: 3, 1093:
3, 1094: 3, 1096: 3, 1098: 3, 1102: 3, 1108: 3, 1109: 3, 1111: 3, 1112: 3, 1113: 3, 1117: 3, 1122:
3, 1129: 3, 1131: 3, 1135: 3, 1142: 3, 1160: 3, 1169: 3, 1172: 3, 1179: 3, 1188: 3, 1198: 3, 1226:
3, 1234: 3, 1235: 3, 1237: 3, 1238: 3, 1245: 3, 1247: 3, 1250: 3, 1252: 3, 1258: 3, 1265: 3, 1266:
3, 1269: 3, 1271: 3, 1277: 3, 1294: 3, 1298: 3, 1300: 3, 1301: 3, 1308: 3, 1322: 3, 1326: 3, 1327:
3, 1340: 3, 1348: 3, 1363: 3, 1365: 3, 1366: 3, 1368: 3, 1370: 3, 1404: 3, 1423: 3, 1447: 3, 1471:
3, 1473: 3, 1489: 3, 1497: 3, 1499: 3, 1510: 3, 1521: 3, 1542: 3, 1557: 3, 1615: 3, 1622: 3, 1628:
3, 1635: 3, 1701: 3, 1703: 3, 1705: 3, 1715: 3, 1719: 3, 1763: 3, 1797: 3, 1816: 3, 1828: 3, 1834:
3, 1878: 3, 1889: 3, 1931: 3, 1964: 3, 1998: 3, 2012: 3, 2055: 3, 2065: 3, 2078: 3, 2137: 3, 2155:
3, 2189: 3, 2269: 3, 2322: 3, 2335: 3, 2440: 3, 2547: 3, 2637: 3, 2686: 3, 3052: 3, 3457: 3, 3593:
3, 3910: 3, 798: 3, 1034: 3, 552: 2, 553: 2, 580: 2, 595: 2, 622: 2, 628: 2, 644: 2, 670: 2, 675:
2, 686: 2, 718: 2, 719: 2, 725: 2, 731: 2, 759: 2, 762: 2, 771: 2, 773: 2, 777: 2, 783: 2, 784: 2,
792: 2, 793: 2, 796: 2, 800: 2, 818: 2, 824: 2, 828: 2, 840: 2, 841: 2, 853: 2, 855: 2, 861: 2,
865: 2, 875: 2, 879: 2, 882: 2, 883: 2, 889: 2, 890: 2, 891: 2, 894: 2, 901: 2, 903: 2, 908: 2,
913: 2, 914: 2, 919: 2, 929: 2, 941: 2, 944: 2, 950: 2, 958: 2, 965: 2, 968: 2, 975: 2, 976: 2,
979: 2, 980: 2, 983: 2, 989: 2, 992: 2, 993: 2, 999: 2, 1005: 2, 1007: 2, 1010: 2, 1011: 2, 1014: 2
, 1015: 2, 1023: 2, 1025: 2, 1029: 2, 1030: 2, 1036: 2, 1040: 2, 1042: 2, 1044: 2, 1049: 2, 1051: 2
, 1054: 2, 1059: 2, 1064: 2, 1070: 2, 1077: 2, 1078: 2, 1079: 2, 1080: 2, 1082: 2, 1088: 2, 1091: 2
, 1095: 2, 1097: 2, 1099: 2, 1549: 2, 1107: 2, 1115: 2, 1119: 2, 1120: 2, 1126: 2, 1133: 2, 1137: 2
, 1138: 2, 1146: 2, 1147: 2, 1149: 2, 1152: 2, 1153: 2, 1154: 2, 1162: 2, 1165: 2, 1184: 2, 1189: 2
, 1190: 2, 1192: 2, 1194: 2, 1195: 2, 1200: 2, 1201: 2, 1203: 2, 1207: 2, 1211: 2, 1212: 2, 1218: 2
, 1221: 2, 1224: 2, 1227: 2, 1229: 2, 1231: 2, 1243: 2, 1257: 2, 1262: 2, 1263: 2, 1276: 2, 1278: 2
, 1281: 2, 1285: 2, 1287: 2, 1288: 2, 1290: 2, 1291: 2, 1293: 2, 1295: 2, 1296: 2, 1299: 2, 1302: 2
, 1303: 2, 1305: 2, 1312: 2, 1316: 2, 1323: 2, 1330: 2, 1335: 2, 1336: 2, 1338: 2, 1341: 2, 1345: 2
, 1347: 2, 1350: 2, 1353: 2, 1354: 2, 1362: 2, 1371: 2, 1379: 2, 1384: 2, 1386: 2, 1390: 2, 1391: 2
, 1395: 2, 1396: 2, 1398: 2, 1402: 2, 1409: 2, 1411: 2, 1417: 2, 1419: 2, 1420: 2, 1422: 2, 1428: 2
, 1429: 2, 1436: 2, 1437: 2, 1439: 2, 1443: 2, 1444: 2, 1453: 2, 1459: 2, 1461: 2, 1463: 2, 1484: 2
, 1490: 2, 1496: 2, 1508: 2, 1509: 2, 1513: 2, 1518: 2, 1522: 2, 1525: 2, 1526: 2, 1527: 2, 1561: 2
, 1580: 2, 1582: 2, 1584: 2, 1589: 2, 1593: 2, 1595: 2, 1602: 2, 1605: 2, 1608: 2, 1614: 2, 1617: 2
, 1624: 2, 1627: 2, 1629: 2, 1630: 2, 1631: 2, 1632: 2, 1644: 2, 1647: 2, 1649: 2, 1653: 2, 1654: 2
, 1658: 2, 1659: 2, 1663: 2, 1676: 2, 1677: 2, 1681: 2, 1687: 2, 1689: 2, 1694: 2, 1698: 2, 1704: 2
, 1712: 2, 1720: 2, 16671: 2, 1725: 2, 1737: 2, 1743: 2, 1744: 2, 1761: 2, 1777: 2, 1784: 2, 1787:
2, 1789: 2, 1793: 2, 1796: 2, 1803: 2, 1807: 2, 1809: 2, 1827: 2, 1830: 2, 1841: 2, 1842: 2, 1844:
2, 1847: 2, 1853: 2, 1859: 2, 1864: 2, 1885: 2, 1892: 2, 1896: 2, 1900: 2, 1919: 2, 1924: 2, 1927:
2, 1934: 2, 1935: 2, 1949: 2, 1955: 2, 1961: 2, 1972: 2, 1985: 2, 1987: 2, 1997: 2, 2000: 2, 2001:
2, 2004: 2, 2010: 2, 2014: 2, 2016: 2, 2023: 2, 2024: 2, 2032: 2, 2041: 2, 2057: 2, 2084: 2, 2085:
2, 2088: 2, 2103: 2, 2106: 2, 2114: 2, 2122: 2, 2126: 2, 2168: 2, 2171: 2, 2175: 2, 2181: 2, 2198:
2, 2222: 2, 2228: 2, 2246: 2, 2262: 2, 2263: 2, 2266: 2, 2312: 2, 2317: 2, 2348: 2, 2358: 2, 2363:
2, 2365: 2, 2367: 2, 2390: 2, 2391: 2, 2394: 2, 2402: 2, 2426: 2, 2441: 2, 2450: 2, 2464: 2, 2466:
2, 2467: 2, 2490: 2, 2494: 2, 2498: 2, 2502: 2, 2542: 2, 2549: 2, 2559: 2, 2608: 2, 2609: 2, 2613:
2, 2641: 2, 2647: 2, 2662: 2, 2665: 2, 2694: 2, 2768: 2, 2808: 2, 2811: 2, 2855: 2, 2874: 2, 2884:
2, 2893: 2, 2905: 2, 3005: 2, 3048: 2, 3060: 2, 3063: 2, 3087: 2, 3156: 2, 3161: 2, 3250: 2, 3308:
2, 3316: 2, 3381: 2, 3383: 2, 3404: 2, 3410: 2, 3424: 2, 3488: 2, 3522: 2, 3555: 2, 3589: 2, 3591:
2, 3626: 2, 3639: 2, 3695: 2, 3732: 2, 3880: 2, 1493: 2, 12128: 2, 4006: 2, 4077: 2, 4141: 2, 1533:
2, 4276: 2, 4381: 2, 4397: 2, 4419: 2, 4537: 2, 4600: 2, 4603: 2, 4614: 2, 776: 2, 4744: 2, 13063:
2, 5021: 2, 5362: 2, 2277: 2, 5628: 2, 5653: 2, 5734: 2, 6189: 2, 6258: 2, 1103: 2, 6655: 2, 6738:
2, 6906: 2, 1177: 2, 15291: 2, 1223: 2, 8254: 1, 8593: 1, 9647: 1, 8753: 1, 671: 1, 709: 1, 727: 1,
740: 1, 757: 1, 760: 1, 148232: 1, 786: 1, 789: 1, 49950: 1, 804: 1, 805: 1, 808: 1, 823: 1, 832:
1, 833: 1, 838: 1, 847: 1, 849: 1, 858: 1, 860: 1, 869: 1, 874: 1, 880: 1, 9078: 1, 887: 1, 893:
1, 899: 1, 900: 1, 902: 1, 904: 1, 906: 1, 907: 1, 917: 1, 921: 1, 923: 1, 926: 1, 931: 1, 932: 1,
17320: 1, 937: 1, 938: 1, 940: 1, 943: 1, 947: 1, 948: 1, 951: 1, 953: 1, 960: 1, 961: 1, 967: 1,
974: 1, 981: 1, 985: 1, 986: 1, 996: 1, 9193: 1, 1004: 1, 1013: 1, 1016: 1, 1018: 1, 1021: 1, 1026:
1, 1031: 1, 1033: 1, 9226: 1, 1037: 1, 8365: 1, 1043: 1, 1047: 1, 1050: 1, 1053: 1, 1055: 1, 1056:
1, 1060: 1, 9258: 1, 1067: 1, 1068: 1, 1074: 1, 1083: 1, 1085: 1, 1086: 1, 1087: 1, 1089: 1, 1090:
1, 9292: 1, 1104: 1, 1105: 1, 1114: 1, 1116: 1, 1118: 1, 1121: 1, 1125: 1, 1128: 1, 1130: 1, 1132:
1, 1136: 1, 1139: 1, 1140: 1, 1143: 1, 1145: 1, 1155: 1, 1156: 1, 9349: 1, 9313: 1, 1161: 1, 1164:
1, 1167: 1, 9360: 1, 1170: 1, 9363: 1, 1174: 1, 1175: 1, 1176: 1, 9369: 1, 1178: 1, 1181: 1, 1185:
1, 1186: 1, 1196: 1, 1197: 1, 1199: 1, 1202: 1, 1205: 1, 1210: 1, 1214: 1, 1216: 1, 1217: 1, 1219:
1, 1220: 1, 1222: 1, 8224: 1, 1225: 1, 1232: 1, 1239: 1, 1240: 1, 1246: 1, 1249: 1, 1251: 1, 1254:
1, 1256: 1, 1259: 1, 1261: 1, 1268: 1, 1270: 1, 1273: 1, 1274: 1, 1279: 1, 1280: 1, 1284: 1, 1297:
1, 1306: 1, 1309: 1, 1310: 1, 9505: 1, 25893: 1, 1318: 1, 1319: 1, 25896: 1, 1324: 1, 1325: 1, 1337
: 1, 25915: 1, 1349: 1, 1351: 1, 1352: 1, 1356: 1, 1358: 1, 1359: 1, 1360: 1, 1364: 1, 1369: 1, 137
2: 1, 1373: 1, 1374: 1, 1376: 1, 1381: 1, 1385: 1, 1387: 1, 1388: 1, 1389: 1, 1392: 1, 1394: 1, 842
6: 1, 1406: 1, 1407: 1, 1408: 1, 9604: 1, 1413: 1, 1415: 1, 1416: 1, 1421: 1, 1425: 1, 1426: 1, 143
1: 1, 1432: 1, 1433: 1, 1434: 1, 1435: 1, 1442: 1, 1446: 1, 1449: 1, 1412: 1, 1454: 1, 1455: 1, 145
6: 1, 1457: 1, 1460: 1, 9654: 1, 1464: 1, 1465: 1, 1468: 1, 1470: 1, 1474: 1, 1477: 1, 1479: 1, 148
1: 1, 1483: 1, 1485: 1, 1486: 1, 1487: 1, 1488: 1, 1492: 1, 9685: 1, 1494: 1, 1501: 1, 1503: 1, 150
4: 1, 1505: 1, 1506: 1, 17898: 1, 1516: 1, 1517: 1, 1519: 1, 1520: 1, 1524: 1, 1528: 1, 1529: 1, 15
30: 1, 1531: 1, 1532: 1, 9725: 1, 1536: 1, 1541: 1, 1544: 1, 1545: 1, 1546: 1, 1547: 1, 1548: 1, 17
933: 1, 1550: 1, 1552: 1, 1553: 1, 1556: 1, 1558: 1, 1560: 1, 1563: 1, 1565: 1, 1566: 1, 1569: 1, 1
570: 1, 1571: 1, 1572: 1, 19147: 1, 1575: 1, 1576: 1, 1578: 1, 1579: 1, 1581: 1, 1585: 1, 1586: 1,
1588: 1, 1590: 1, 1591: 1, 1596: 1, 1598: 1, 1599: 1, 1601: 1, 1606: 1, 1611: 1, 1613: 1, 1616: 1,
1618: 1, 1623: 1, 1626: 1, 1633: 1, 1636: 1, 9831: 1, 1643: 1, 1646: 1, 1651: 1, 1652: 1, 1657: 1,
1665: 1, 1666: 1, 1669: 1, 1671: 1, 1674: 1, 1675: 1, 1678: 1, 1679: 1, 1683: 1, 1684: 1, 1688: 1,
1692: 1, 1697: 1, 1699: 1, 1700: 1, 1706: 1, 1707: 1, 1709: 1, 1710: 1, 1711: 1, 1714: 1, 1717: 1,
1721: 1, 9914: 1, 1727: 1, 1728: 1, 1730: 1, 1732: 1, 8241: 1, 1734: 1, 1738: 1, 1739: 1, 1745: 1,
1746: 1, 4387: 1, 1750: 1, 1751: 1, 1755: 1, 1756: 1, 1757: 1, 1758: 1, 1762: 1, 1765: 1, 1766: 1,
1768: 1, 1769: 1, 1771: 1, 1772: 1, 1773: 1, 1780: 1, 1790: 1, 1792: 1, 1798: 1, 1799: 1, 16684: 1,
1805: 1, 1808: 1, 10002: 1, 1811: 1, 1814: 1, 1815: 1, 1818: 1, 1820: 1, 1821: 1, 1822: 1, 1826: 1,
1829: 1, 16689: 1, 1832: 1, 1833: 1, 1836: 1, 1838: 1, 1840: 1, 1845: 1, 1848: 1, 1850: 1, 1852: 1,
1856: 1, 1857: 1, 1860: 1, 1865: 1, 1866: 1, 10060: 1, 1871: 1, 1872: 1, 1875: 1, 1877: 1, 1880: 1,
1881: 1, 1882: 1, 1883: 1, 1639: 1, 1887: 1, 1894: 1, 1897: 1, 1898: 1, 1899: 1, 10096: 1, 1905: 1,
1908: 1, 1911: 1, 18296: 1, 1914: 1, 1916: 1, 1917: 1, 1918: 1, 10113: 1, 1925: 1, 10118: 1, 1928:
1, 1929: 1, 1933: 1, 1936: 1, 1937: 1, 1940: 1, 1941: 1, 1944: 1, 1950: 1, 1954: 1, 1957: 1, 1958:
1, 8519: 1, 1967: 1, 1969: 1, 1970: 1, 1973: 1, 1974: 1, 1976: 1, 1977: 1, 4703: 1, 1979: 1, 1982:
1, 1984: 1, 1991: 1, 1993: 1, 12621: 1, 2003: 1, 2005: 1, 2006: 1, 2008: 1, 2015: 1, 2017: 1, 2025:
1, 2027: 1, 2033: 1, 2034: 1, 2037: 1, 2039: 1, 2043: 1, 2045: 1, 2049: 1, 2051: 1, 2052: 1, 2058:
1, 2059: 1, 2060: 1, 2066: 1, 2067: 1, 2079: 1, 2087: 1, 2089: 1, 2090: 1, 2092: 1, 2094: 1, 2095:
1, 2099: 1, 2101: 1, 2104: 1, 2109: 1, 2111: 1, 17801: 1, 12640: 1, 2115: 1, 8545: 1, 5815: 1, 2124
: 1, 2125: 1, 2127: 1, 2128: 1, 2129: 1, 2130: 1, 19470: 1, 2134: 1, 2135: 1, 2136: 1, 2138: 1, 214
0: 1, 1722: 1, 2142: 1, 2143: 1, 2144: 1, 1723: 1, 2158: 1, 2159: 1, 2166: 1, 2167: 1, 2169: 1, 217
2: 1, 2174: 1, 18560: 1, 2177: 1, 2178: 1, 2180: 1, 2184: 1, 2187: 1, 2188: 1, 2190: 1, 2194: 1, 21
95: 1, 2196: 1, 2197: 1, 2199: 1, 2201: 1, 2203: 1, 2205: 1, 2207: 1, 2210: 1, 2211: 1, 2213: 1, 22
20: 1, 2226: 1, 2229: 1, 2230: 1, 2233: 1, 2234: 1, 2235: 1, 2236: 1, 2237: 1, 2244: 1, 2245: 1, 22
49: 1, 18636: 1, 10445: 1, 2264: 1, 2265: 1, 2270: 1, 2273: 1, 2275: 1, 18661: 1, 2279: 1, 2281: 1,
2289: 1, 2291: 1, 2295: 1, 2302: 1, 2303: 1, 2309: 1, 8577: 1, 2319: 1, 2328: 1, 10530: 1, 2339: 1,
2345: 1, 2346: 1, 10539: 1, 2352: 1, 2353: 1, 2354: 1, 2357: 1, 2359: 1, 2361: 1, 8262: 1, 2369: 1,
2370: 1, 2373: 1, 2374: 1, 2375: 1, 2376: 1, 2377: 1, 2379: 1, 2381: 1, 2386: 1, 2389: 1, 18777: 1,
10588: 1, 2398: 1, 2399: 1, 2403: 1, 2404: 1, 2405: 1, 2410: 1, 2411: 1, 2412: 1, 2413: 1, 2421: 1,
2427: 1, 2428: 1, 2429: 1, 2432: 1, 2433: 1, 2434: 1, 2435: 1, 2436: 1, 2452: 1, 2455: 1, 2458: 1,
```

2460: 1, 2463: 1, 2465: 1, 2468: 1, 2469: 1, 2470: 1, 2471: 1, 2475: 1, 2477: 1, 2479: 1, 2480: 1,
2482: 1, 2483: 1, 2484: 1, 2486: 1, 2488: 1, 2489: 1, 3017: 1, 18879: 1, 2496: 1, 2497: 1, 2503: 1,
2506: 1, 2511: 1, 2512: 1, 41379: 1, 2517: 1, 2520: 1, 2524: 1, 2526: 1, 2527: 1, 2529: 1, 2530: 1,
2531: 1, 2532: 1, 2536: 1, 2538: 1, 2539: 1, 2546: 1, 2548: 1, 2553: 1, 2556: 1, 10750: 1, 2560: 1,
2562: 1, 2566: 1, 2569: 1, 2571: 1, 8269: 1, 2577: 1, 11353: 1, 2584: 1, 2586: 1, 2589: 1, 2591: 1,
2592: 1, 2594: 1, 2596: 1, 10794: 1, 2606: 1, 2614: 1, 2617: 1, 2618: 1, 2623: 1, 10820: 1, 2636: 1
, 10832: 1, 2644: 1, 2650: 1, 2651: 1, 2654: 1, 2659: 1, 2661: 1, 2663: 1, 2669: 1, 2671: 1, 2674:
1, 2676: 1, 2677: 1, 2679: 1, 2680: 1, 2682: 1, 14101: 1, 41408: 1, 2690: 1, 2695: 1, 2696: 1, 2699
: 1, 2702: 1, 2712: 1, 2713: 1, 19102: 1, 2721: 1, 2724: 1, 2726: 1, 2729: 1, 2732: 1, 10925: 1, 27
35: 1, 10931: 1, 2742: 1, 2743: 1, 2745: 1, 2746: 1, 2747: 1, 2748: 1, 2755: 1, 2758: 1, 2760: 1, 2
763: 1, 2774: 1, 2783: 1, 2786: 1, 2789: 1, 2790: 1, 2796: 1, 2805: 1, 2807: 1, 2810: 1, 2813: 1, 1
1005: 1, 2821: 1, 42648: 1, 2827: 1, 2830: 1, 2833: 1, 2840: 1, 2843: 1, 2848: 1, 2851: 1, 2870: 1,
2873: 1, 2876: 1, 8672: 1, 2882: 1, 2888: 1, 2889: 1, 2890: 1, 10039: 1, 2894: 1, 2899: 1, 2902: 1,
2904: 1, 11407: 1, 2909: 1, 2912: 1, 2919: 1, 2920: 1, 2921: 1, 11119: 1, 35698: 1, 2931: 1, 2932:
1, 2933: 1, 2948: 1, 2952: 1, 2953: 1, 2961: 1, 2962: 1, 2974: 1, 2980: 1, 2982: 1, 2988: 1, 2990:
1, 2994: 1, 2996: 1, 2997: 1, 19387: 1, 3006: 1, 11199: 1, 3010: 1, 1867: 1, 1868: 1, 3019: 1, 3028
: 1, 3030: 1, 3035: 1, 8698: 1, 3040: 1, 3043: 1, 3046: 1, 3049: 1, 3056: 1, 3061: 1, 35833: 1, 307
0: 1, 3075: 1, 9824: 1, 3084: 1, 3085: 1, 3086: 1, 3097: 1, 11295: 1, 3104: 1, 3106: 1, 3110: 1, 31
11: 1, 3115: 1, 3117: 1, 3118: 1, 3121: 1, 3128: 1, 3129: 1, 3139: 1, 3141: 1, 3144: 1, 3147: 1, 31
54: 1, 3155: 1, 3158: 1, 3164: 1, 3166: 1, 3168: 1, 3170: 1, 3171: 1, 3175: 1, 3176: 1, 3182: 1, 16
916: 1, 11395: 1, 3207: 1, 11403: 1, 3214: 1, 3215: 1, 8728: 1, 3219: 1, 8729: 1, 3228: 1, 3231: 1,
1904: 1, 3235: 1, 3239: 1, 3240: 1, 3249: 1, 3252: 1, 3253: 1, 3256: 1, 3258: 1, 3262: 1, 8292: 1,
3268: 1, 11461: 1, 3270: 1, 3274: 1, 3283: 1, 3285: 1, 3287: 1, 3288: 1, 3290: 1, 3295: 1, 3296: 1,
3307: 1, 11502: 1, 3311: 1, 27897: 1, 3323: 1, 3324: 1, 3326: 1, 3332: 1, 36105: 1, 3338: 1, 3343:
1, 11538: 1, 3347: 1, 3349: 1, 3353: 1, 3354: 1, 8751: 1, 3367: 1, 11563: 1, 3377: 1, 3378: 1, 3379
: 1, 3382: 1, 3401: 1, 3409: 1, 3412: 1, 3418: 1, 8762: 1, 3423: 1, 3431: 1, 3432: 1, 3436: 1, 3442
: 1, 3448: 1, 3450: 1, 3451: 1, 3452: 1, 3454: 1, 3458: 1, 3463: 1, 3467: 1, 3469: 1, 3475: 1, 3310
: 1, 3480: 1, 3482: 1, 11675: 1, 3490: 1, 3494: 1, 3495: 1, 3497: 1, 19883: 1, 3500: 1, 3507: 1, 35
13: 1, 3517: 1, 11712: 1, 3529: 1, 3530: 1, 3534: 1, 3535: 1, 3541: 1, 3543: 1, 3548: 1, 10149: 1,
3553: 1, 3556: 1, 3564: 1, 3566: 1, 3570: 1, 11763: 1, 3575: 1, 3577: 1, 3582: 1, 3605: 1, 11799: 1
, 3609: 1, 3610: 1, 3614: 1, 3616: 1, 3618: 1, 3620: 1, 3622: 1, 18353: 1, 3627: 1, 11833: 1, 3643:
1, 3651: 1, 8801: 1, 44623: 1, 3672: 1, 3676: 1, 3684: 1, 3691: 1, 3692: 1, 3702: 1, 3703: 1, 11898
: 1, 3715: 1, 3717: 1, 3718: 1, 3720: 1, 3721: 1, 8813: 1, 3733: 1, 36505: 1, 3743: 1, 3355: 1, 374
9: 1, 3751: 1, 3755: 1, 3757: 1, 3759: 1, 3761: 1, 3767: 1, 11962: 1, 3772: 1, 11965: 1, 28357: 1,
3784: 1, 3787: 1, 20179: 1, 3799: 1, 3800: 1, 3802: 1, 3811: 1, 3817: 1, 3820: 1, 3824: 1, 12018: 1
, 10195: 1, 3830: 1, 3833: 1, 3840: 1, 3855: 1, 3860: 1, 12062: 1, 3871: 1, 12065: 1, 3874: 1, 3879
: 1, 3900: 1, 3903: 1, 3915: 1, 3920: 1, 3932: 1, 3934: 1, 3938: 1, 3943: 1, 3944: 1, 3952: 1, 1215
4: 1, 3971: 1, 3972: 1, 8854: 1, 3974: 1, 20359: 1, 17047: 1, 12175: 1, 3985: 1, 3986: 1, 3996: 1,
3999: 1, 4003: 1, 4005: 1, 4010: 1, 12205: 1, 4014: 1, 4766: 1, 4023: 1, 4031: 1, 4032: 1, 12229: 1
, 4038: 1, 4770: 1, 12238: 1, 4049: 1, 4051: 1, 4055: 1, 4057: 1, 4059: 1, 4064: 1, 4066: 1, 4067:
1, 4076: 1, 4078: 1, 4085: 1, 4087: 1, 4088: 1, 12281: 1, 4097: 1, 4113: 1, 18435: 1, 12309: 1, 123
18: 1, 12319: 1, 4145: 1, 4146: 1, 4148: 1, 4150: 1, 4153: 1, 4160: 1, 4164: 1, 4166: 1, 4169: 1, 4
176: 1, 4181: 1, 4203: 1, 8893: 1, 12409: 1, 4232: 1, 4235: 1, 4238: 1, 4239: 1, 4242: 1, 4244: 1,
4261: 1, 4262: 1, 4264: 1, 4266: 1, 17873: 1, 4275: 1, 4283: 1, 4286: 1, 4287: 1, 4289: 1, 4305: 1,
4312: 1, 10153: 1, 4318: 1, 12514: 1, 4328: 1, 4346: 1, 4353: 1, 12553: 1, 4368: 1, 4373: 1, 4374:
1, 4386: 1, 20771: 1, 12591: 1, 4407: 1, 4408: 1, 4409: 1, 4417: 1, 4420: 1, 4427: 1, 4429: 1, 4441
: 1, 20832: 1, 12647: 1, 4456: 1, 4465: 1, 4472: 1, 3477: 1, 2112: 1, 4484: 1, 39692: 1, 12690: 1,
4501: 1, 4510: 1, 3483: 1, 4522: 1, 2119: 1, 8946: 1, 4535: 1, 4539: 1, 16518: 1, 4557: 1, 4561: 1,
4566: 1, 12774: 1, 2133: 1, 4613: 1, 25345: 1, 4619: 1, 12813: 1, 4628: 1, 4644: 1, 4650: 1, 29235:
1, 4664: 1, 4667: 1, 53830: 1, 4680: 1, 4686: 1, 4689: 1, 66318: 1, 4696: 1, 12895: 1, 4725: 1, 211
10: 1, 12926: 1, 4885: 1, 4747: 1, 119436: 1, 4749: 1, 12950: 1, 4759: 1, 8342: 1, 12962: 1, 4771:
1, 4780: 1, 4783: 1, 17890: 1, 4793: 1, 4796: 1, 4803: 1, 4808: 1, 4810: 1, 6265: 1, 4825: 1, 4826:
1, 4827: 1, 4831: 1, 4840: 1, 4841: 1, 4845: 1, 45817: 1, 4858: 1, 13052: 1, 2176: 1, 37678: 1, 489
2: 1, 4894: 1, 4902: 1, 45870: 1, 9011: 1, 4921: 1, 13117: 1, 4928: 1, 4933: 1, 4938: 1, 4942: 1, 4
945: 1, 4954: 1, 4968: 1, 9021: 1, 16533: 1, 4994: 1, 5014: 1, 5030: 1, 11762: 1, 5054: 1, 5070: 1,
5079: 1, 5083: 1, 5090: 1, 5098: 1, 5099: 1, 5111: 1, 5115: 1, 9045: 1, 5125: 1, 5130: 1, 5144: 1,
5158: 1, 5171: 1, 5174: 1, 5182: 1, 5197: 1, 5206: 1, 1810: 1, 5225: 1, 5230: 1, 5239: 1, 5242: 1,
5245: 1, 5257: 1, 5269: 1, 5281: 1, 5291: 1, 5292: 1, 5296: 1, 5298: 1, 5301: 1, 5304: 1, 5308: 1,
5309: 1, 4127: 1, 5328: 1, 5342: 1, 21728: 1, 21748: 1, 9086: 1, 5367: 1, 5374: 1, 5376: 1, 13574:
1, 13591: 1, 5403: 1, 5412: 1, 5424: 1, 17290: 1, 5454: 1, 13653: 1, 5462: 1, 5467: 1, 9106: 1, 336
84: 1, 21886: 1, 21398: 1, 5519: 1, 5521: 1, 38291: 1, 5543: 1, 3655: 1, 5556: 1, 5560: 1, 5578: 1,
17917: 1, 54751: 1, 5620: 1, 5626: 1, 5642: 1, 5648: 1, 5673: 1, 5678: 1, 5682: 1, 63042: 1, 13894:
1, 5703: 1, 5704: 1, 5711: 1, 5723: 1, 22111: 1, 5740: 1, 5742: 1, 5750: 1, 5752: 1, 5764: 1, 5774:
1, 5776: 1, 9157: 1, 9161: 1, 8377: 1, 5832: 1, 5833: 1, 5849: 1, 63194: 1, 5866: 1, 9170: 1, 5883:
1, 5908: 1, 5909: 1, 79638: 1, 9177: 1, 5916: 1, 9179: 1, 5925: 1, 5942: 1, 5947: 1, 14140: 1, 6566
6: 1, 5967: 1, 6000: 1, 1001: 1, 6013: 1, 6020: 1, 14245: 1, 6060: 1, 14255: 1, 6068: 1, 17771: 1,
2930: 1, 6085: 1, 14278: 1, 6100: 1, 16570: 1, 6108: 1, 6140: 1, 6146: 1, 6150: 1, 6154: 1, 6155: 1
, 6157: 1, 14352: 1, 2393: 1, 14377: 1, 6187: 1, 1405: 1, 6201: 1, 6210: 1, 6215: 1, 6216: 1, 1039:
1, 6238: 1, 6241: 1, 2407: 1, 6253: 1, 6254: 1, 39033: 1, 8225: 1, 24760: 1, 6302: 1, 6321: 1, 6327
: 1, 6337: 1, 22723: 1, 6341: 1, 14545: 1, 16409: 1, 6367: 1, 6373: 1, 1066: 1, 6420: 1, 6463: 1, 6
474: 1, 6489: 1, 6495: 1, 14712: 1, 6531: 1, 25665: 1, 6542: 1, 14743: 1, 6557: 1, 6571: 1, 6572: 1
, 6574: 1, 9288: 1, 8948: 1, 6587: 1, 6589: 1, 6590: 1, 6597: 1, 6604: 1, 6638: 1, 6656: 1, 17494:
1, 65690: 1, 6680: 1, 31265: 1, 6699: 1, 14892: 1, 6707: 1, 6713: 1, 6717: 1, 6725: 1, 39495: 1, 14
924: 1, 6812: 1, 6814: 1, 6815: 1, 6859: 1, 15073: 1, 6889: 1, 6890: 1, 6891: 1, 15089: 1, 6912: 1,
6914: 1, 6924: 1, 6928: 1, 6929: 1, 15122: 1, 15132: 1, 6962: 1, 6963: 1, 7004: 1, 7006: 1, 7007: 1
, 1168: 1, 7013: 1, 1171: 1, 7057: 1, 7086: 1, 10741: 1, 7105: 1, 7110: 1, 7120: 1, 7123: 1, 7133:
1, 7135: 1, 66727: 1, 10058: 1, 7191: 1, 15400: 1, 7210: 1, 23605: 1, 9788: 1, 3936: 1, 9399: 1, 72

```
83: 1, 23688: 1, 7306: 1, 7310: 1, 7312: 1, 7327: 1, 7340: 1, 7344: 1, 7346: 1, 7372: 1, 15577: 1,
7386: 1, 15595: 1, 9426: 1, 9427: 1, 7430: 1, 7434: 1, 7436: 1, 15632: 1, 7447: 1, 7463: 1, 7467: 1
, 42205: 1, 15676: 1, 7521: 1, 7527: 1, 8435: 1, 7565: 1, 8708: 1, 40369: 1, 15807: 1, 7618: 1, 762
1: 1, 7623: 1, 7667: 1, 24053: 1, 32252: 1, 7684: 1, 65034: 1, 7705: 1, 7706: 1, 7707: 1, 7709: 1,
24121: 1, 32317: 1, 24157: 1, 25872: 1, 48739: 1, 7791: 1, 7802: 1, 7817: 1, 7839: 1, 7842: 1, 7865
: 1, 7867: 1, 1313: 1, 4045: 1, 1317: 1, 12241: 1, 7924: 1, 32502: 1, 7938: 1, 2689: 1, 17710: 1, 7
981: 1, 7985: 1, 7986: 1, 16185: 1, 8024: 1, 8056: 1, 32653: 1, 8088: 1, 24474: 1, 8093: 1, 32681:
1, 8111: 1, 16637: 1, 8119: 1, 8131: 1, 8156: 1})
```

In [129]:

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

In [130]:

```python
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']))
```

In [131]:

```python
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)
```

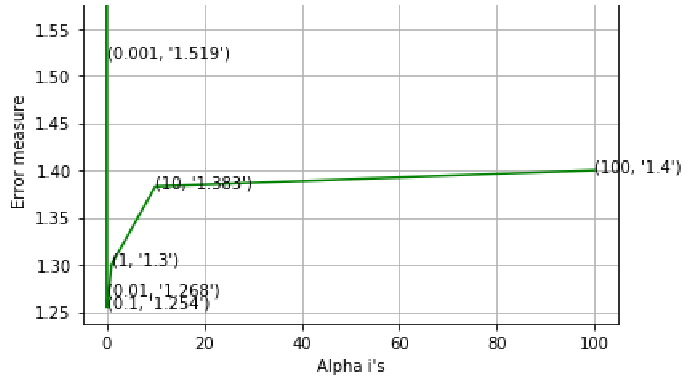
```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 776068)
(number of data points * number of features) in test data =  (665, 776068)
(number of data points * number of features) in cross validation data = (532, 776068)
```

6.3.1 Logistic Regression

In [133]:

```
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 : 1.585467177235256
for alpha = 1e-05
Log Loss : 1.586270084971086
for alpha = 0.0001
Log Loss : 1.5897149173978415
for alpha = 0.001
Log Loss : 1.5193765554894854
for alpha = 0.01
Log Loss : 1.2678047949128457
for alpha = 0.1
Log Loss : 1.254491077524204
for alpha = 1
Log Loss : 1.300300461352358
for alpha = 10
Log Loss : 1.382868362769988
for alpha = 100
Log Loss : 1.3998625264967144
```

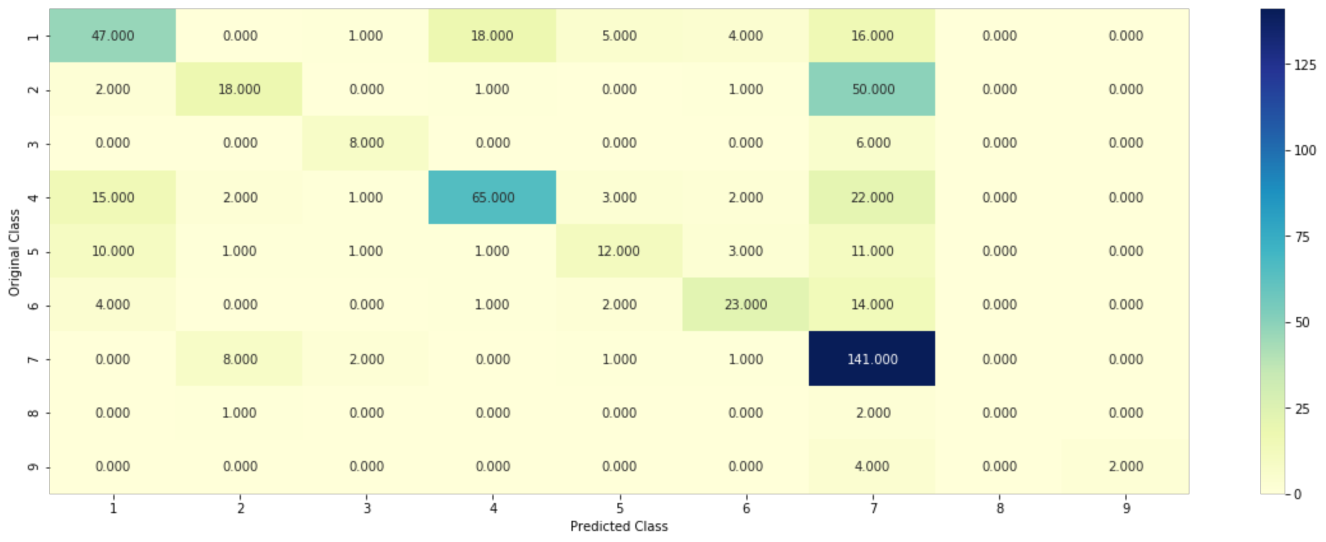


```
For values of best alpha =  0.1 The train log loss is: 0.8893795197108413
For values of best alpha =  0.1 The cross validation log loss is: 1.254491077524204
For values of best alpha =  0.1 The test log loss is: 1.1848644028206798
```

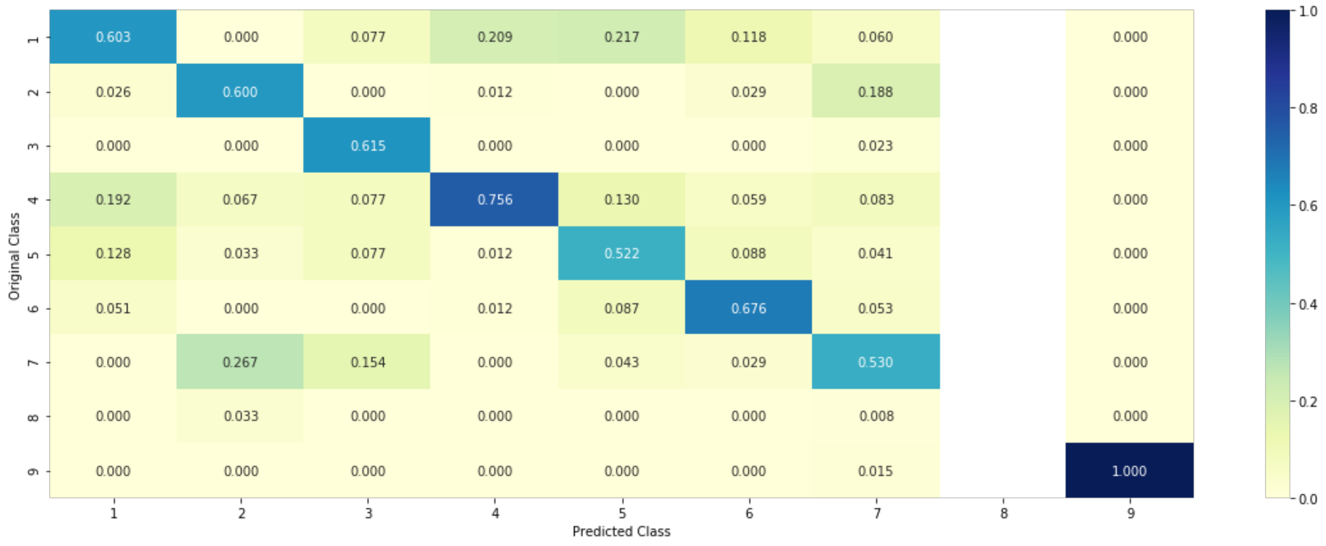
In [134]:

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

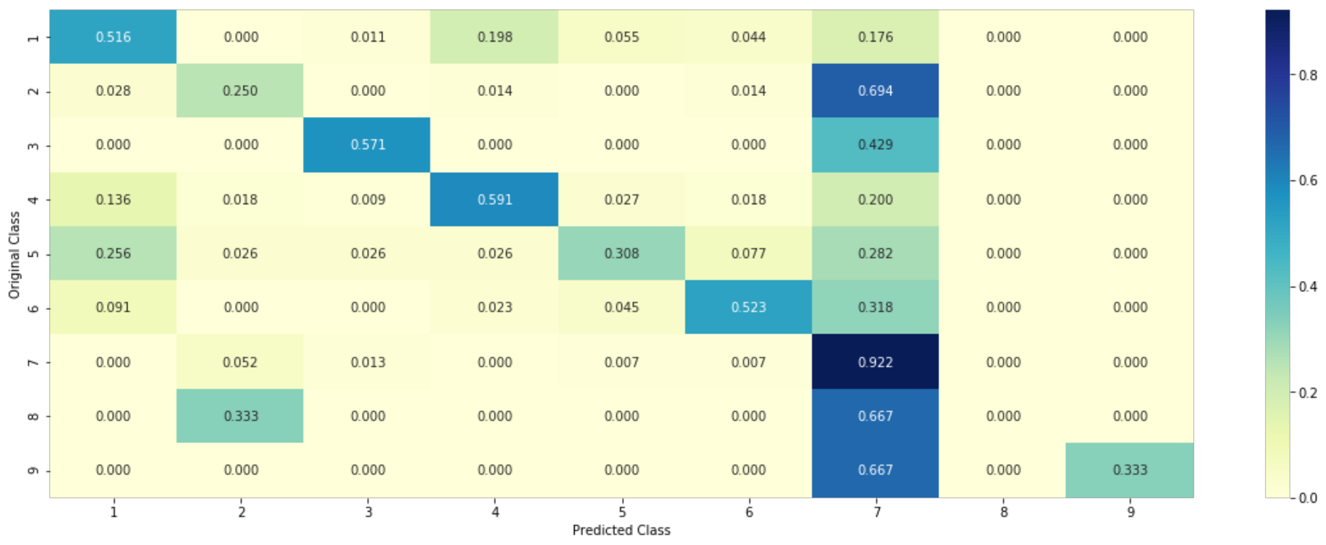
```
Log loss : 1.254491077524204
Number of mis-classified points : 0.40601503759398494
-------------------- Confusion matrix --------------------
```



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



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



7. Model Comparision:

In [135]:

```
x = PrettyTable()

x.field_names = ["Model", "Vectorizer", "Train loss", "CV loss", "Test loss", "% of misclassified p
oints"]

x.add_row(['Logistic Regression(Balanced) - one hot encoding', 'Unigram', 0.805, 1.203, 1.114, 37.
7])
x.add_row(['Logistic Regression(Balanced) - one hot encoding', 'Biigram', 0.889, 1.254, 1.184, 40.
6])

print(x)
```

```
+--------------------------------------------------+------------+------------+---------+-----------
-------------------------+
|                      Model                       | Vectorizer | Train loss | CV loss | Test loss 
% of misclassified points |
+--------------------------------------------------+------------+------------+---------+-----------
-------------------------+
| Logistic Regression(Balanced) - one hot encoding |  Unigram   |   0.805    |  1.203  |   1.114   
|           37.7            |
| Logistic Regression(Balanced) - one hot encoding |  Biigram   |   0.889    |  1.254  |   1.184   
|           40.6            |
+--------------------------------------------------+------------+------------+---------+-----------
-------------------------+
```

8. Conclusion

1. Initially we had two files in which our data exist - a. training_variants, b. training_text. training_variants has features ID, gene, variaton, class and the training_text has features ID, text.
2. We've preprocessed the text data and merged the two files into one based on the ID's.
3. After merging two files, we've checked for any null values and found null values in the text and replaced the null values with gene + variation.
4. Now that we got the complete data set, we've splitted the data in train, cv and test in the ratio of 64:16:20 respectively.
5. After splitting the data we've built a random model to make sure the real models we train are better based on the random model.
6. Now, we've performed the uni-variate analysis on individual feature using logistic regression and found that text and gene features are more useful and more stable compared to the variation feature. I've used one hot encoding and response coding for gene and variation features and TFIDF vectorizer for text feature.
7. Now that we've performed univariate analysis, the next step was to apply different machine learning models with hyperparameter tuning and compared the results.
8. After training models on TF-IDF, we've also trained logistic regression models on BoW using unigrams and bigrams

8. After training models on TF-IDF, we've also trained logistic regression models on BoW using unigrams and bigrams.

```
In [ ]:
```