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.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

#from mlxtend.classifier import StackingClassifier

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

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [4]:

```
data = pd.read_csv('training_variants.csv')
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
0	0	FAM58A	Truncating Mutations	1

	ID	Gene	Variation	Class
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 + " "
```

```
        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 : 149.86975610564537 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])
```

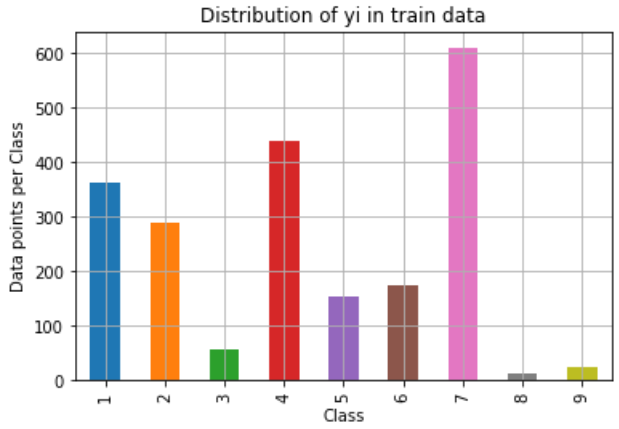
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

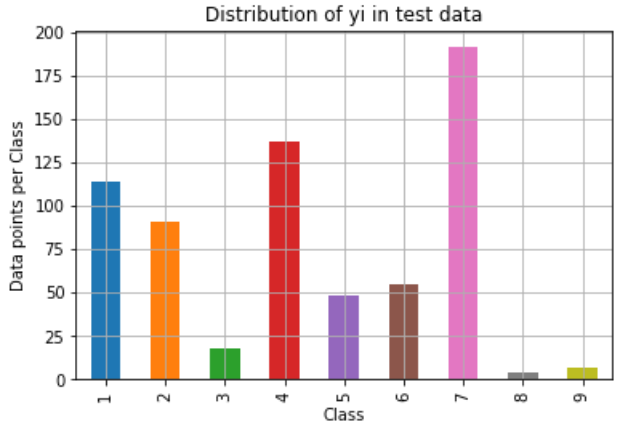
In [14]:

```
# 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), '%)')
```

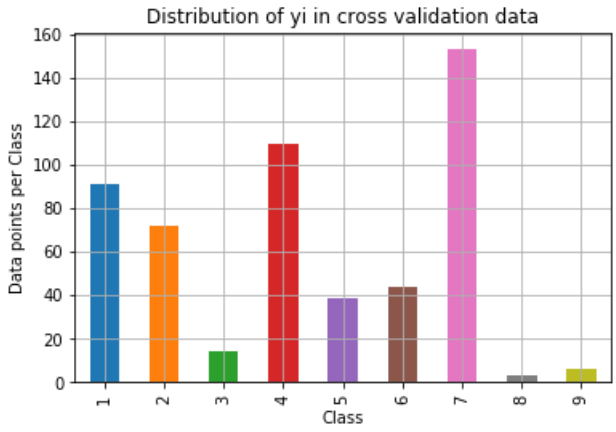



```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298 %)
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]:

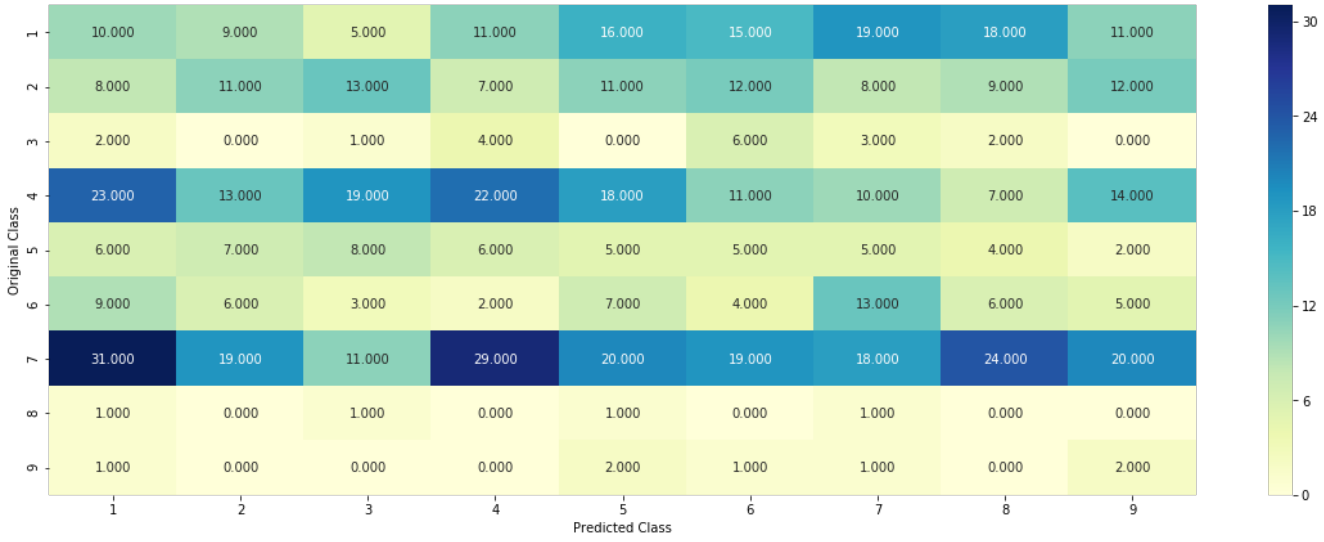
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

    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)
```

```
    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()
```

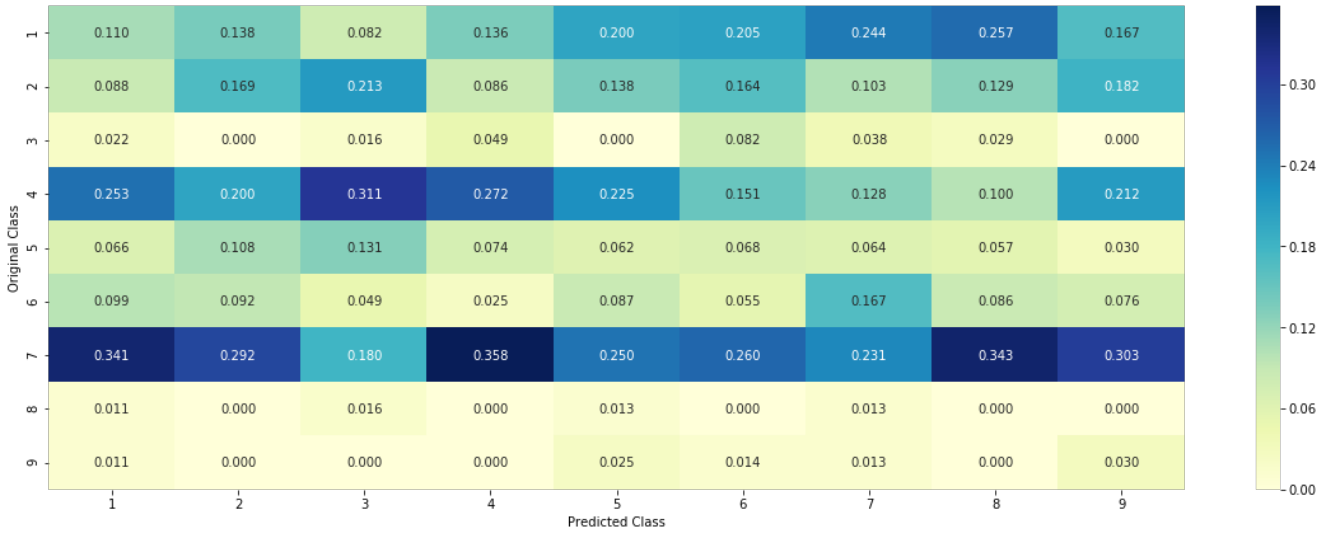
In [16]:

```
# 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)
```

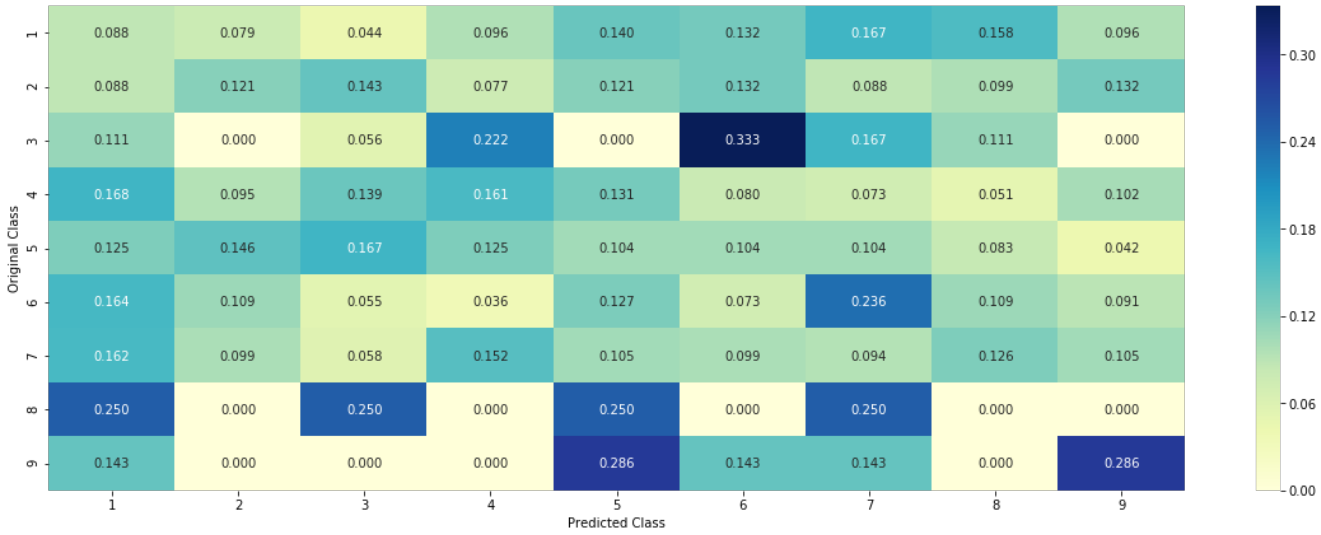
```
Log loss on Cross Validation Data using Random Model 2.517643683888507
Log loss on Test Data using Random Model 2.525770652548291
-------------------- Confusion matrix --------------------
```




```
-------------------- 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
```

```
    #         TP53         100
    #         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, 0.03787878787878788, 0.068181818181818177,
0.13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788,
0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366,
0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408
163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177,
0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608,
0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608,
0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917,
0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081
761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295,
0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702,
0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334,
0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999,
0.066666666666666666, 0.066666666666666666],
```

```
0.0666666666666666, 0.0666666666666666],
    #       ...
    #     }
    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     164
TP53      104
EGFR       91
PTEN       82
BRCA2      79
KIT        70
BRAF       59
ALK        49
ERBB2      39
PIK3CA     35
Name: Gene, dtype: int64
```

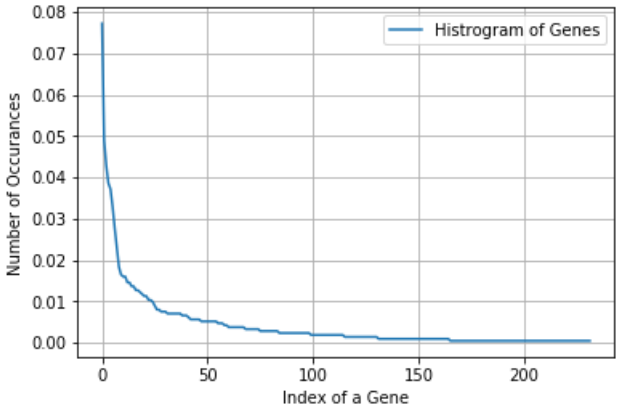
In [19]:

```
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
```

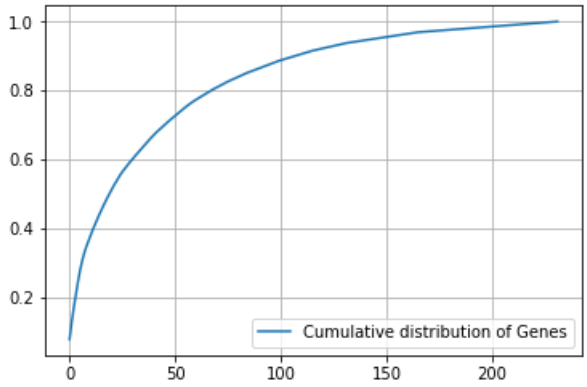
In [20]:

```
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]:

```
2201     PTEN
1350     AKT1
982      ETV6
3210      RB1
1586    CARM1
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',
 'asxl1',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'aurkb',
 'axl',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk6',
 'cdk8',
```

```
'cdkn1a',
'cdkn1b',
'cdkn2a',
'cdkn2b',
'cdkn2c',
'chek2',
'cic',
'crebbp',
'ctcf',
'ctnnb1',
'ddr2',
'dicer1',
'dnmt3a',
'dnmt3b',
'egfr',
'elf3',
'ep300',
'epas1',
'erbb2',
'erbb3',
'erbb4',
'ercc2',
'ercc4',
'erg',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fam58a',
'fanca',
'fat1',
'fbxw7',
'fgf19',
'fgf4',
'fgfr1',
'fgfr2',
'fgfr3',
'fgfr4',
'flt1',
'flt3',
'foxa1',
'foxl2',
'foxo1',
'gata3',
'gli1',
'gna11',
'gnaq',
'h3f3a',
'hist1h1c',
'hla',
'hnf1a',
'hras',
'idh1',
'idh2',
'igf1r',
'ikbke',
'il7r',
'jak1',
'jak2',
'kdm5c',
'kdm6a',
'kdr',
'keap1',
'kit',
'klf4',
'kmt2a',
'kmt2b',
'kmt2c',
'kmt2d',
'knstrn',
'kras',
'lats1',
'lats2',
'map2k1',
'map2k2',
'map2k4',
```

```
'map3k1',
'mapk1',
'mdm2',
'med12',
'mef2b',
'men1',
'met',
'mga',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'mycn',
'myd88',
'myod1',
'ncor1',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',
'nkx2',
'notch1',
'notch2',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'nup93',
'pax8',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pik3r3',
'pim1',
'pms1',
'pms2',
'pole',
'ppp2r1a',
'ppp6c',
'prdm1',
'ptch1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rab35',
'rac1',
'rad21',
'rad50',
'rad51c',
'rad54l',
'raf1',
'rara',
'rasa1',
'rb1',
'rbm10',
'ret',
'rheb',
'rhoa',
'rit1',
'rnf43',
'ros1',
'rras2',
'runx1',
'rxra',
'sdhb',
'sdhc',
'setd2',
'sf3b1',
```

```
 'shq1',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'stag2',
 'stat3',
 'stk11',
 'tcf3',
 'tcf7l2',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vhl',
 'whsc1',
 'whsc1l1',
 'xpo1',
 '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, 231)
```

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

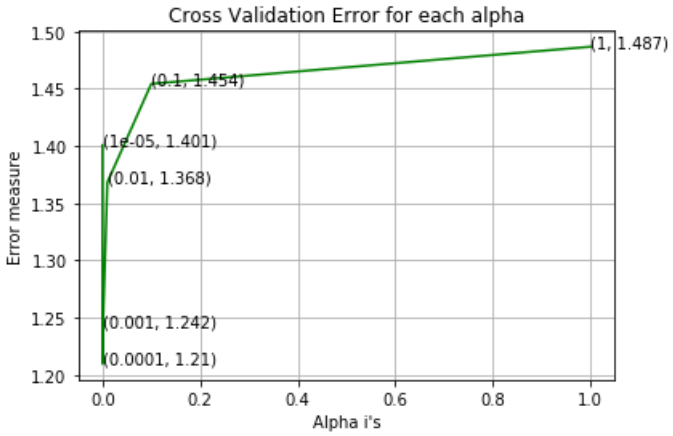
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [28]:

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

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

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

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


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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.400723919485547
For values of alpha =  0.0001 The log loss is: 1.209593674082376
For values of alpha =  0.001 The log loss is: 1.2421911542693114
For values of alpha =  0.01 The log loss is: 1.3682090473953596
For values of alpha =  0.1 The log loss is: 1.4542984396676448
For values of alpha =  1 The log loss is: 1.4867073207866122
```




```
For values of best alpha =  0.0001 The train log loss is: 1.0130616675946895
For values of best alpha =  0.0001 The cross validation log loss is: 1.209593674082376
For values of best alpha =  0.0001 The test log loss is: 1.2225685973164468
```

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

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

In [29]:

```
print("Q6. How many data points in Test and CV datasets are covered by the ", 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 646 out of 665 : 97.14285714285714
2. In cross validation data 511 out of  532 : 96.05263157894737
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

In [30]:

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

```
Number of Unique Variations : 1930
Truncating_Mutations    54
Deletion                49
Amplification           45
Fusions                 22
Overexpression           5
G12V                     3
E17K                     3
Q61L                     3
Q61H                     3
Q61R                     3
Name: Variation, dtype: int64
```

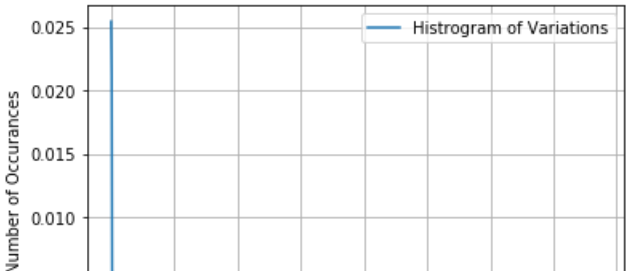
In [31]:

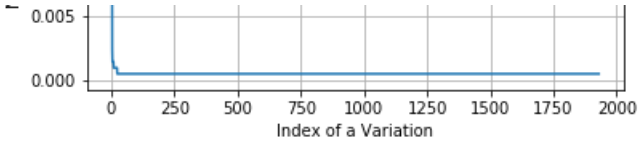
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of variations in the
train data, and they are distibuted as follows",)
```

```
Ans: There are 1930 different categories of variations in the train data, and they are distibuted
as follows
```

In [32]:

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

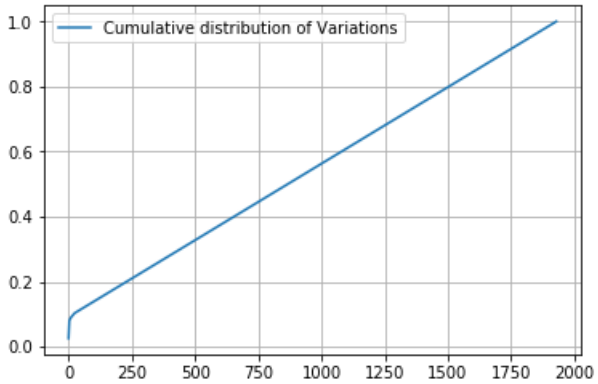




In [33]:

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

```
[0.02542373 0.04849341 0.06967985 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [34]:

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

In [35]:

```
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 [40]:

```
# 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 [41]:

```
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, 1959)
```

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

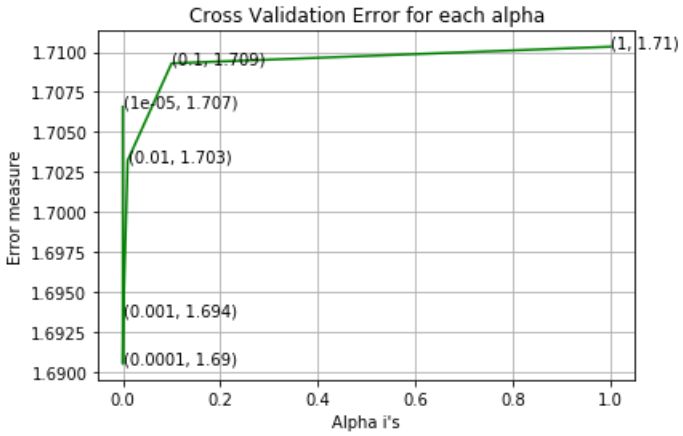
Let's build a model just like the earlier!

In [42]:

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

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

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

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


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

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

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

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


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

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

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

```
For values of alpha =  1e-05 The log loss is: 1.706575187191091
For values of alpha =  0.0001 The log loss is: 1.6904878066664548
For values of alpha =  0.001 The log loss is: 1.6935071119158236
For values of alpha =  0.01 The log loss is: 1.7032138984768395
For values of alpha =  0.1 The log loss is: 1.7092947327344439
For values of alpha =  1 The log loss is: 1.7103343396331578
```




```
For values of best alpha =  0.0001 The train log loss is: 0.7099309388158609
For values of best alpha =  0.0001 The cross validation log loss is: 1.6904878066664548
For values of best alpha =  0.0001 The test log loss is: 1.7106661240342929
```

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 [43]:

```
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  1930  genes in test and cross validation data
sets?
Ans
1. In test data 63 out of 665 : 9.473684210526317
2. In cross validation data 61 out of  532 : 11.466165413533833
```

3.2.3 Univariate Analysis on Text Feature

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

In [38]:

```
# cls_text is a data frame
# for every row in data fram consider the 'TEXT'
# split the words by space
# make a dict with those words
# increment its count whenever we see that word

def extract dictionary paddle(cls text):
```

```
    dictionary = defaultdict(int)
    for index, row in cls_text.iterrows():
        for word in row['TEXT'].split():
            dictionary[word] +=1
    return dictionary
```

In [39]:

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

++

TASK 1 : TF-Idf with Top 2500 Features

++

In [36]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

# building a CountVectorizer with all the words that occured minimum 5 times in train data
text_vectorizer = TfidfVectorizer(max_features = 2500, min_df = 5)
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("++"*30)
print("Total Number Of Unique Words in Train Data :", len(train_text_features))
print("++"*30)
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Total Number Of Unique Words in Train Data : 2500
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

In [44]:

```
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 [45]:

```
#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 [46]:

```
# 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 [47]:

```
# 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 [48]:

```
#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 [49]:

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

```
Counter({197.83990216773944: 1, 135.86889354437196: 1, 119.87056125122628: 1, 102.44263364165974:
1, 96.09488634118074: 1, 96.05509942529139: 1, 95.56673365455389: 1, 92.78959373281819: 1,
92.31814556022982: 1, 84.69062835580459: 1, 82.67302310483255: 1, 75.16837426525943: 1,
72.62080523669842: 1, 71.97341205473235: 1, 71.41033940749497: 1, 68.24334493915605: 1,
63.748632735803795: 1, 62.50578023461656: 1, 61.750742156671656: 1, 60.232455479220455: 1,
59.02943135568096: 1, 58.53503744265358: 1, 55.710402606925484: 1, 54.76837175888124: 1,
53.23610083277169: 1, 52.052234209595426: 1, 51.67808676113479: 1, 51.30033313120515: 1,
50.80539501482721: 1, 50.79117401956985: 1, 50.11172521736339: 1, 49.96604882804489: 1,
49.50615592189898: 1, 49.097643656063006: 1, 48.57649770994604: 1, 46.94599974380654: 1,
46.019382350746895: 1, 44.06654318739959: 1, 43.93052114052727: 1, 43.375283724067515: 1,
42.924393100165425: 1, 41.49283810809194: 1, 41.27147370672764: 1, 40.59838463634959: 1, 40.4594523
6964931: 1, 40.35248545647748: 1, 37.97201662550041: 1, 37.72468100744833: 1, 37.091553831801455:
1, 36.90843723325788: 1, 36.04245696649999: 1, 35.23408651764933: 1, 35.01444484714573: 1,
34.96002747607442: 1, 34.959826709520115: 1, 33.987849626726145: 1, 33.762297795343294: 1,
33.66619135249106: 1, 33.624549806142156: 1, 33.442742546942476: 1, 33.280542764990415: 1,
33.208421209008215: 1, 32.954303443234906: 1, 32.9336854828316: 1, 32.460031005468664: 1,
32.33886525786131: 1, 31.648388851287923: 1, 31.54796465917949: 1, 31.286391138134427: 1,
30.82070236553367: 1, 30.566630995193997: 1, 29.78982430841987: 1, 29.7791640102571l6: 1,
29.45481129533731: 1, 29.256043283538435: 1, 29.192073479107872: 1, 28.83641764095739: 1,
```

28.690503002422123: 1, 28.380574184637165: 1, 28.353411312757522: 1, 27.832277090576593: 1,
27.635553905034552: 1, 27.448569335853623: 1, 27.3775903602297: 1, 27.360755878524788: 1,
27.157012718589588: 1, 27.133714159856233: 1, 27.06769857946202: 1, 27.021875114298762: 1,
26.91780173370655: 1, 26.899544458215818: 1, 26.513756036638327: 1, 26.493953957248664: 1,
25.88264519462012: 1, 25.807987084468934: 1, 25.782325238866438: 1, 25.712041062364715: 1,
25.4314236546557: 1, 25.422189138837883: 1, 25.39424394158756: 1, 25.34632348780844: 1,
25.301210781747983: 1, 25.28361372514612: 1, 25.280721805480116: 1, 25.002985057403894: 1,
24.912137329484032: 1, 24.803380743872516: 1, 24.688807067016462: 1, 24.6103953317011 84: 1,
24.452952206885588: 1, 24.411406751867318: 1, 24.280392874633723: 1, 24.2100227269511 52: 1,
24.155089448329136: 1, 24.089826063471897: 1, 24.068374657932193: 1, 24.0398580718897: 1,
24.02352472120543: 1, 23.990790688569415: 1, 23.872391316825144: 1, 23.69322314093248: 1,
23.677925584875737: 1, 23.547257781197178: 1, 23.396590923319867: 1, 23.378326306256625: 1,
23.341100481113056: 1, 23.297187297795524: 1, 23.28977149499426: 1, 23.068482492672526: 1,
22.856330481239773: 1, 22.83152183315355: 1, 22.800529227617766: 1, 22.53525317049051: 1,
22.45582234272283: 1, 22.296733114327537: 1, 22.289049299141436: 1, 22.15940921861767: 1,
22.117315777097176: 1, 21.79187719068866: 1, 21.68630407794915: 1, 21.608463895096925: 1,
21.458109356963902: 1, 21.424096859564944: 1, 21.417900718813783: 1, 21.289522245753417: 1,
21.01836540037423: 1, 20.949720759104007: 1, 20.90402970820092: 1, 20.848270072099496: 1,
20.532646505185287: 1, 20.527720533515517: 1, 20.45339516422031: 1, 20.45062144695618: 1,
20.400503764153893: 1, 20.36391095949687: 1, 20.328245053225608: 1, 20.233988965886173: 1,
20.149885824124894: 1, 19.977393383431803: 1, 19.96733467435919: 1, 19.867617222088935: 1,
19.845011976063873: 1, 19.83300881084406: 1, 19.667792328669996: 1, 19.6313065828201 16: 1,
19.261650839967526: 1, 19.223435745464357: 1, 19.18935973417 14: 1, 19.06184345674091: 1,
19.05207019001389: 1, 19.01849123977777: 1, 18.93900107233414: 1, 18.914771049999228: 1,
18.914077554424505: 1, 18.90146485437037: 1, 18.751659884751 96: 1, 18.635534952419 28: 1, 18.5889768
6450289: 1, 18.58157938890609: 1, 18.468926288960034: 1, 18.36444200677419: 1, 18.34310955017073:
1, 18.24849082172013: 1, 18.177104063039582: 1, 18.14441140004325: 1, 18.126343934662064: 1,
18.111872834841098: 1, 18.09671129783795: 1, 18.057086195901476: 1, 18.028844619165678: 1,
17.904338898835142: 1, 17.88494707484555: 1, 17.866121420250 58: 1, 17.752817313747695: 1,
17.68377714588016: 1, 17.590630362426754: 1, 17.589652623487 83: 1, 17.565496527624877: 1,
17.530003435648943: 1, 17.500736685377206: 1, 17.423856375839687: 1, 17.389583251189748: 1,
17.38614033097498: 1, 17.33992330613464: 1, 17.331163479407493: 1, 17.316917789942067: 1,
17.29406890669379: 1, 17.28080751410478: 1, 17.254400545388737: 1, 17.24969344912712: 1,
17.22972690142567: 1, 17.17421614885719: 1, 17.104868647947132: 1, 17.098737927506406: 1,
17.087675910087786: 1, 17.017781472380452: 1, 17.008168862048 13: 1, 16.985027854327143: 1,
16.975972566622094: 1, 16.92253135575267: 1, 16.922103478836043: 1, 16.908484650392285: 1,
16.85942488006543: 1, 16.818857950987624: 1, 16.810450250270193: 1, 16.806416711077713: 1,
16.805856839920242: 1, 16.802051005160752: 1, 16.725854702439 86: 1, 16.697166125941404: 1,
16.688434134262305: 1, 16.632393748456057: 1, 16.630731609193983: 1, 16.588684215799514: 1,
16.466503585968493: 1, 16.462596924546432: 1, 16.405519510521 44: 1, 16.364846203775 35: 1,
16.355076974470776: 1, 16.332187184040 86: 1, 16.299109617080212: 1, 16.256398748183322: 1,
16.226270820188574: 1, 16.197713934464165: 1, 16.169238826225733: 1, 16.168886079867 84: 1,
16.131556938115448: 1, 16.106883283191 36: 1, 16.044028475741502: 1, 15.896854906432194: 1,
15.85411342507669: 1, 15.82076882101117: 1, 15.802105699539 91: 1, 15.793161600116305: 1,
15.692598487872903: 1, 15.692250178575636: 1, 15.685230758875354: 1, 15.672616998403276: 1,
15.644709121276577: 1, 15.618344374766899: 1, 15.601186658908533: 1, 15.593458804315821: 1,
15.575728026175753: 1, 15.568386415765353: 1, 15.552008448556 68: 1, 15.551556205985214: 1,
15.528537774077286: 1, 15.448719012173074: 1, 15.408493424795013: 1, 15.402382203384878: 1,
15.35582081406351: 1, 15.235966775620398: 1, 15.172260756212973: 1, 15.139437738873907: 1,
15.067889093759055: 1, 15.01068561711751: 1, 15.001629592945998: 1, 14.997282742439252: 1,
14.992968997720364: 1, 14.977328098101237: 1, 14.953435274229262: 1, 14.895164174121776: 1,
14.85969706037967: 1, 14.853033989989985: 1, 14.840511781961258: 1, 14.810662007521675: 1,
14.808575947358705: 1, 14.796379833204092: 1, 14.773039937962 19: 1, 14.771889529846295: 1,
14.762395207867145: 1, 14.740316137735816: 1, 14.717263642182251: 1, 14.709925528540149: 1,
14.60205157870507: 1, 14.553524413353896: 1, 14.531052265630956: 1, 14.506846302113894: 1,
14.458145634762381: 1, 14.456339737419567: 1, 14.449634924187922: 1, 14.443601550947943: 1,
14.431736660190332: 1, 14.364165310812526: 1, 14.302422557172783: 1, 14.283094928792499: 1,
14.28308861791992: 1, 14.267669708074518: 1, 14.261156353790474: 1, 14.160728848755966: 1,
14.158215087729037: 1, 14.155346861251697: 1, 14.102770990015255: 1, 14.064425183675512: 1,
14.06119600090138: 1, 14.020888093440233: 1, 13.996248180821333: 1, 13.983478135715952: 1,
13.974299715807208: 1, 13.95653493837564: 1, 13.956345275333767: 1, 13.87818823530447: 1,
13.85965709562428: 1, 13.846716016001674: 1, 13.833940688129594: 1, 13.83176701187541: 1,
13.720852749037581: 1, 13.709998898207163: 1, 13.689589949048798: 1, 13.650898598610382: 1,
13.629973009751524: 1, 13.585468594862862: 1, 13.566928567861 74: 1, 13.532615486927275: 1,
13.484111092551414: 1, 13.436797232206684: 1, 13.408098351332 95: 1, 13.403415613325192: 1,
13.374128487268099: 1, 13.353375823331342: 1, 13.332939976046847: 1, 13.324853287511106: 1,
13.319380244950251: 1, 13.279321646368256: 1, 13.276143458718 67: 1, 13.274104549756776: 1,
13.25229271567913: 1, 13.246877025539177: 1, 13.211150567447538: 1, 13.186359176672 62: 1,
13.175699192938053: 1, 13.121303628424771: 1, 13.068811559984 47: 1, 13.03893768017447: 1,
13.036107713258618: 1, 13.021144540723135: 1, 13.006889781728505: 1, 12.928616542457743: 1,
12.927893753742044: 1, 12.910473454594102: 1, 12.896337952888537: 1, 12.889582641590547: 1,
12.884674520431252: 1, 12.874553982179778: 1, 12.860989311368327: 1, 12.831556128630462: 1,
12.82664930340718: 1, 12.822024365707549: 1, 12.779305515902754: 1, 12.769253258493242: 1,
12.767275490057317: 1, 12.76413581069566: 1, 12.704609937369336: 1, 12.680257966338088: 1,
12.592900177726614: 1, 12.579821264363625: 1, 12.555064213371095: 1, 12.541021327152862: 1,
12.531989181049822: 1, 12.507236557759253: 1, 12.505877719892462: 1, 12.494596116638004: 1,
12.484589387803526: 1, 12.459408610247788: 1, 12.442482457587763: 1, 12.431704017667 63: 1,

12.427633225632068: 1, 12.415882235450782: 1, 12.408382950478643: 1, 12.385775493117643: 1,
12.37257496301067: 1, 12.349765105950581: 1, 12.349258889985459: 1, 12.325884207275672: 1,
12.321526967499402: 1, 12.290678476787035: 1, 12.285197404344443: 1, 12.265018829424788: 1,
12.209465367884897: 1, 12.208145444641154: 1, 12.196532572689355: 1, 12.19472573001066: 1,
12.182338603866057: 1, 12.162314833422917: 1, 12.160540636492204: 1, 12.151556873448401: 1,
12.148760650806043: 1, 12.110457415585136: 1, 12.10973374686979: 1, 12.104851084739623: 1,
12.095958591936192: 1, 12.089578579831887: 1, 12.080417931767153: 1, 12.08026005097759: 1,
12.07620442978962: 1, 12.02696456241555: 1, 12.01253337257609: 1, 11.988878745937447: 1,
11.98864603551158: 1, 11.89845989554156: 1, 11.883592198756006: 1, 11.865417219689142: 1,
11.85394763501776: 1, 11.803256691085714: 1, 11.791155074363246: 1, 11.785190150274692: 1,
11.782677806910765: 1, 11.759447657117075: 1, 11.737206038598828: 1, 11.735401123670941: 1,
11.682165328710335: 1, 11.620835333122121: 1, 11.618785116548006: 1, 11.61721239526164: 1,
11.6134469297477: 1, 11.612142671553574: 1, 11.598750654158472: 1, 11.584693497396268: 1,
11.541052785133186: 1, 11.53758796674875: 1, 11.532610597271207: 1, 11.509786007652595: 1,
11.504191841408563: 1, 11.484307641846646: 1, 11.454518024645402: 1, 11.428835038662013: 1,
11.412940219143504: 1, 11.399185175515099: 1, 11.398417389649392: 1, 11.38963308045661: 1,
11.357218233238724: 1, 11.353856525699866: 1, 11.327251596012337: 1, 11.305742421486325: 1,
11.300882231833919: 1, 11.274548334524406: 1, 11.265482855638329: 1, 11.258309423372475: 1,
11.228861290309993: 1, 11.222100858870755: 1, 11.212884117865608: 1, 11.19223000443725: 1,
11.170086414515616: 1, 11.154466563329072: 1, 11.148275905039096: 1, 11.144018390788698: 1,
11.142045653821652: 1, 11.13896234677091: 1, 11.079293193551296: 1, 11.074797535958268: 1,
11.07368608187461: 1, 11.063857838021956: 1, 11.060658190026203: 1, 11.018325469364335: 1,
11.002117263834176: 1, 10.998401833334679: 1, 10.997433115599868: 1, 10.996669760215989: 1,
10.967741684464642: 1, 10.958455899456903: 1, 10.947913742817367: 1, 10.942534893616184: 1,
10.907643421474807: 1, 10.877300248511101: 1, 10.873849087387665: 1, 10.85494432953844: 1,
10.844409311028707: 1, 10.827349284657695: 1, 10.825484324487912: 1, 10.81681439702012: 1,
10.798179732932438: 1, 10.792008399028477: 1, 10.786377709328585: 1, 10.783202320210586: 1,
10.773936409788087: 1, 10.769954387976961: 1, 10.762398879931919: 1, 10.750517625321528: 1,
10.721422062157751: 1, 10.693826864977568: 1, 10.620362836929033: 1, 10.619875080113888: 1,
10.598607803373607: 1, 10.598585291877471: 1, 10.577526208675042: 1, 10.573097720210528: 1,
10.555438053599627: 1, 10.514306701937937: 1, 10.494352027369759: 1, 10.486108096379054: 1,
10.48423611658016: 1, 10.479440948210918: 1, 10.477634809703066: 1, 10.461318290264948: 1,
10.448888919941144: 1, 10.444728560837236: 1, 10.418575550911852: 1, 10.389046427301125: 1,
10.369941221997404: 1, 10.367795823146736: 1, 10.362586118746957: 1, 10.362462536203589: 1,
10.345442823960653: 1, 10.342833320601883: 1, 10.336107542208829: 1, 10.332497378999795: 1,
10.329495144956892: 1, 10.328132823604859: 1, 10.30587404706583: 1, 10.2933515827221 46: 1,
10.260759579768862: 1, 10.248549765637327: 1, 10.241591625015266: 1, 10.173346696334098: 1,
10.169409251152153: 1, 10.149036166055565: 1, 10.145623869055163: 1, 10.142168213948057: 1,
10.098859211590192: 1, 10.062258916318067: 1, 10.05428074683287: 1, 10.042786939219502: 1,
10.023379917473484: 1, 10.018987928433003: 1, 10.015848602854891: 1, 10.004174137563933: 1,
9.984420356322161: 1, 9.97055469585815: 1, 9.963494383107275: 1, 9.947529731557411: 1,
9.90947582724858: 1, 9.857210557532122: 1, 9.85701838179636: 1, 9.807968185823242: 1,
9.788002235178846: 1, 9.787173232452405: 1, 9.780821930899108: 1, 9.75659261641084: 1,
9.744221747718623: 1, 9.743953363049389: 1, 9.743805955478456: 1, 9.736033868189471: 1,
9.729573851354205: 1, 9.72664220596731: 1, 9.698135358436186: 1, 9.682304913304366: 1,
9.666630873756915: 1, 9.646991876542844: 1, 9.632486959885393: 1, 9.631322975490235: 1,
9.62512570844919: 1, 9.615086936170487: 1, 9.614262806512485: 1, 9.612544796297296: 1,
9.607899967051694: 1, 9.601287764818947: 1, 9.600945865463835: 1, 9.577254064084453: 1,
9.571747106388635: 1, 9.56991967341301: 1, 9.56723087881606: 1, 9.522566883531782: 1,
9.513759963852543: 1, 9.512459587075533: 1, 9.477023883867957: 1, 9.456902128566709: 1,
9.445728377056009: 1, 9.440583878323151: 1, 9.40857632495524: 1, 9.384246754643508: 1,
9.369370807882728: 1, 9.368107948030994: 1, 9.329687553877259: 1, 9.326695435752125: 1,
9.289083626226546: 1, 9.278530023172044: 1, 9.256318635896534: 1, 9.249611960201928: 1,
9.240205263781816: 1, 9.225124580153702: 1, 9.219320968888862: 1, 9.198424243963597: 1,
9.191146258839279: 1, 9.185574289697092: 1, 9.168852436019657: 1, 9.168622991674376: 1,
9.16085253784031: 1, 9.138262589389086: 1, 9.13778442369545: 1, 9.118284439517911: 1,
9.11763519177333: 1, 9.104186939350136: 1, 9.095066430436676: 1, 9.094034587595441: 1,
9.092857364605242: 1, 9.087467436635823: 1, 9.085875248490797: 1, 9.072929133183862: 1,
9.072603659880174: 1, 9.06518178612193: 1, 9.064002911344563: 1, 9.056962093103639: 1,
9.042083186217821: 1, 9.02184628862549: 1, 9.017995072530553: 1, 8.999662502295093: 1,
8.979490827467615: 1, 8.967520244198923: 1, 8.961916402811868: 1, 8.957777161416962: 1,
8.955201735476196: 1, 8.954269486450938: 1, 8.952676222436029: 1, 8.923483674251022: 1,
8.918849789898832: 1, 8.918117614720593: 1, 8.906970419794959: 1, 8.895012218883412: 1,
8.882792044554913: 1, 8.860605817565318: 1, 8.849981853194278: 1, 8.848577762871663: 1,
8.834086203519526: 1, 8.832910887736583: 1, 8.832857144794868: 1, 8.820418045146088: 1,
8.815090442015181: 1, 8.813132639415688: 1, 8.811338544078923: 1, 8.793013370482951: 1,
8.7901312605691: 1, 8.789670276694709: 1, 8.780190316353192: 1, 8.746296010442622: 1, 8.73457309904
4425: 1, 8.70281690945093: 1, 8.701666919967797: 1, 8.699858539151345: 1, 8.697747821071028: 1,
8.694577966513 34: 1, 8.681285247519135: 1, 8.672531434236893: 1, 8.65141561977822: 1,
8.65041794622365: 1, 8.63712274535347: 1, 8.614667444886473: 1, 8.608307906405926: 1,
8.602674816907045: 1, 8.595160062367587: 1, 8.58374644115312: 1, 8.582311795818471: 1,
8.577480651364128: 1, 8.559689358679979: 1, 8.53613589913884: 1, 8.533870257974103: 1,
8.522837062643893: 1, 8.515387646827344: 1, 8.508704865761723: 1, 8.5033913314916: 1,
8.499971383107638: 1, 8.494701181708464: 1, 8.489493888430221: 1, 8.488228040899916: 1,
8.483254962321924: 1, 8.48081523888285: 1, 8.479808964198247: 1, 8.478547791380594: 1,
8.477664464066715: 1, 8.45854395636274: 1, 8.444150449164466: 1, 8.43092725034387: 1,
8.429512832377291: 1, 8.42052852789045: 1, 8.407497800740884: 1, 8.39474361451996: 1,

8.3867794051158 38: 1, 8.380656652518075: 1, 8.379179012771642: 1, 8.351989580068848: 1,
8.34718320631845: 1, 8.34345551346589: 1, 8.342736407774629: 1, 8.337208654462135: 1,
8.32927879420529: 1, 8.328275994820283: 1, 8.308948442501999: 1, 8.30256284099919: 1,
8.302067710835951: 1, 8.298820273906982: 1, 8.295738899138925: 1, 8.29182003679958: 1,
8.281021906431057: 1, 8.276428136836794: 1, 8.271494678360355: 1, 8.2707267725328: 1,
8.269624176409344: 1, 8.267347697849395: 1, 8.262133799252533: 1, 8.244451052753558: 1,
8.240319169531798: 1, 8.23998063901605: 1, 8.236418161505611: 1, 8.230057460689231: 1,
8.229659915386184: 1, 8.224898188748032: 1, 8.216266369433809: 1, 8.207977322223751: 1,
8.204481586928278: 1, 8.203814781935069: 1, 8.200043459498117: 1, 8.187734027373208: 1,
8.18140525014631: 1, 8.178942143544933: 1, 8.163160429916466: 1, 8.161431749474474: 1,
8.146425749360366: 1, 8.136671930556037: 1, 8.134008228857478: 1, 8.113419375378935: 1,
8.110337307681704: 1, 8.100278522237115: 1, 8.09918278754038: 1, 8.09230492803897: 1,
8.071808392483048: 1, 8.069595725375251: 1, 8.06753117020511: 1, 8.06214036481133: 1,
8.060238934201713: 1, 8.056384363463946: 1, 8.03618036206428: 1, 8.032077811257023: 1,
8.03197158804962: 1, 8.018608500209881: 1, 8.017236060457027: 1, 8.017203205567625: 1,
8.011836689139058: 1, 8.009622661674426: 1, 8.009283117138477: 1, 7.997225576123142: 1,
7.987989906095763: 1, 7.9840521726291485: 1, 7.981084198371811: 1, 7.975324570273151: 1,
7.9663481093965265: 1, 7.955241326696996: 1, 7.938496735300309: 1, 7.937912889399008: 1, 7.93464343
6887528: 1, 7.934001645657203: 1, 7.927096412676784: 1, 7.916031886485493: 1, 7.91406328948566: 1,
7.913979972004647: 1, 7.909772411021918: 1, 7.88717582321603: 1, 7.874925842456993: 1,
7.871307140963604: 1, 7.86982247191802: 1, 7.868607477075747: 1, 7.868561908442242: 1,
7.867025292993086: 1, 7.863457480666333: 1, 7.862670811551753: 1, 7.8549934793215 19: 1,
7.854050154945546: 1, 7.832075036497772: 1, 7.808283831349865: 1, 7.8070130410056 14: 1,
7.798532545593114: 1, 7.796857372453128: 1, 7.774130017483922: 1, 7.766798326346965: 1,
7.766410638974734: 1, 7.7575062796644785: 1, 7.7443468484583935: 1, 7.7381401436908375: 1,
7.735487283673596: 1, 7.72900303710068: 1, 7.721769341398417: 1, 7.7121789634375455: 1,
7.690416721309513: 1, 7.686330475393919: 1, 7.6849448278819 15: 1, 7.6833408929189 23: 1,
7.679907908671941: 1, 7.675541312640001: 1, 7.6735305191166 34: 1, 7.6647405449256 53: 1,
7.660887950545406: 1, 7.6524025933059665: 1, 7.63244815180383: 1, 7.6312944719155 08: 1,
7.629934626772302: 1, 7.628270388362361: 1, 7.625278786485603: 1, 7.6209967883066 93: 1,
7.6201710829098035: 1, 7.608173899344139: 1, 7.603493196903212: 1, 7.602685378699327: 1, 7.60146399
47589995: 1, 7.5993986398147575: 1, 7.586182531935829: 1, 7.565232148951733: 1, 7.564109931949308:
1, 7.554612979099763: 1, 7.553882780369152: 1, 7.548594525061861: 1, 7.544018206073974: 1,
7.54005859209542: 1, 7.530994319326818: 1, 7.528875102151539: 1, 7.52867989820415: 1,
7.527557640087915: 1, 7.525993121641607: 1, 7.5145992329065 08: 1, 7.4992469464386 67: 1,
7.481823803604705: 1, 7.4675372377421 98: 1, 7.4614965767836 87: 1, 7.4606535807118 25: 1,
7.454883060699015: 1, 7.451047550228095: 1, 7.45040098348422: 1, 7.4500482740631 08: 1,
7.438493849018461: 1, 7.4252081047805065: 1, 7.4161914970075 63: 1, 7.4120899935357 46: 1,
7.411789364841766: 1, 7.3934630312354885: 1, 7.3933768879835995: 1, 7.3801075248054815: 1,
7.369900558863092: 1, 7.360445374301054: 1, 7.359837642786673: 1, 7.3551397972590 24: 1,
7.3474128717691825: 1, 7.346312097232608: 1, 7.344238847554616: 1, 7.3378160705175 85: 1, 7.33160772
159249: 1, 7.330982524966748: 1, 7.3203228610640 15: 1, 7.3157968164590095: 1, 7.3151173457720 89:
1, 7.31467637374689: 1, 7.313795556931123: 1, 7.31309244333496: 1, 7.3117218069237 14: 1,
7.299423071709969: 1, 7.299230666194835: 1, 7.2973189004355 12: 1, 7.2958780074482 02: 1,
7.28988840920002: 1, 7.288606336473661: 1, 7.2875180659539 53: 1, 7.2743314977500 72: 1,
7.2605688840659655: 1, 7.259645004387862: 1, 7.2475291683004865: 1, 7.2425073543256 48: 1,
7.238791238845942: 1, 7.2295673769489275: 1, 7.225713016931602: 1, 7.2234831807023285: 1,
7.209248198875999: 1, 7.202698112019632: 1, 7.19708540940229: 1, 7.1869341839914 69: 1,
7.17443792262702: 1, 7.172490068790381: 1, 7.1676737875198 76: 1, 7.1340950418613 92: 1,
7.133187626677679: 1, 7.128626176300223: 1, 7.128585453016969: 1, 7.1279555647605 98: 1,
7.108318539986739: 1, 7.100291897469341: 1, 7.0918117239873 28: 1, 7.0914491009417 27: 1,
7.084761142645159: 1, 7.071609836900281: 1, 7.0691391120003315: 1, 7.0631106690167 21: 1,
7.060432252409182: 1, 7.052728614525939: 1, 7.0520989979885655: 1, 7.0341562924444 5: 1,
7.027194174923725: 1, 7.018331584903702: 1, 7.009307562096141: 1, 6.9898612284894 78: 1,
6.989467443526059: 1, 6.982648290326046: 1, 6.977153574615564: 1, 6.9719636012102 41: 1,
6.971315398645628: 1, 6.966237163137486: 1, 6.96210479428352: 1, 6.9616502482991 44: 1,
6.949535671315719: 1, 6.948961630906544: 1, 6.939048602047666: 1, 6.9356146415279 68: 1,
6.935326438832486: 1, 6.926571541073043: 1, 6.923490976023719: 1, 6.9231799362788 78: 1,
6.919982578538691: 1, 6.917184716681746: 1, 6.9163494972014865: 1, 6.9147157742770 44: 1,
6.914635473006434: 1, 6.913322620650706: 1, 6.911329763222925: 1, 6.9088326965710 2: 1,
6.908796487904694: 1, 6.903371372643669: 1, 6.897686373261474: 1, 6.8965732520890 16: 1,
6.893790632357957: 1, 6.8854899820401245: 1, 6.873559677147868: 1, 6.8714125865238 28: 1,
6.861863992315541: 1, 6.8588503831667325: 1, 6.850257906614328: 1, 6.8496274933157 82: 1,
6.849586968039911: 1, 6.8477947425025 12: 1, 6.8453021712758 48: 1, 6.8425975712788 64: 1,
6.830975594888198: 1, 6.829844521268866: 1, 6.829569045678761: 1, 6.8275496791829 78: 1,
6.825664270127208: 1, 6.81835060635053: 1, 6.8162765983279 22: 1, 6.8139523402340 24: 1,
6.798871280129824: 1, 6.796456607217549: 1, 6.793658604587225: 1, 6.7923147353652 69: 1,
6.785955953292016: 1, 6.783227326328375: 1, 6.782400056202889: 1, 6.7777478903809 11: 1,
6.775251992113857: 1, 6.774222111265714: 1, 6.773054482175008: 1, 6.7728708002935 15: 1,
6.758115493589656: 1, 6.756335793549205: 1, 6.751894353172971: 1, 6.7413574500175 06: 1,
6.736966840350991: 1, 6.728285113734412: 1, 6.7277584762619185: 1, 6.7125686212312 21: 1,
6.708566719764542: 1, 6.705452166345614: 1, 6.7039403363496675: 1, 6.6997771424857975: 1,
6.693713231780451: 1, 6.692076411279032: 1, 6.690777870814985: 1, 6.6877853875710 82: 1,
6.686321507889621: 1, 6.682529192243995: 1, 6.6789336197725 02: 1, 6.6759851137787205: 1,
6.673553087460841: 1, 6.661437205946631: 1, 6.648182097307839: 1, 6.6476781253261 77: 1,
6.647378050654257: 1, 6.6392043433081795: 1, 6.639065859935073: 1, 6.6378591319252 46: 1,
6.633163648470709: 1, 6.630441574435035: 1, 6.624829423070266: 1, 6.6231583173067 62: 1,

6.623003768246648: 1, 6.6176161847463035: 1, 6.608697265331079: 1, 6.6023667828753348: 1,
6.592701135801955: 1, 6.592666636073295: 1, 6.583452024198199: 1, 6.563255107838499: 1,
6.554299326591166: 1, 6.537541544090135: 1, 6.530203445789561: 1, 6.528558691128355: 1,
6.524766215844754: 1, 6.524674247287523: 1, 6.5202239727079805: 1, 6.519172197494929: 1,
6.519113729805788: 1, 6.512407563555805: 1, 6.511306652767088: 1, 6.509682156795283: 1,
6.506366617088208: 1, 6.499725713491925: 1, 6.499621350486389: 1, 6.498668313713559: 1,
6.497172832121124: 1, 6.495696645439061: 1, 6.495012215351852: 1, 6.490388200997174: 1,
6.489340798235220: 1, 6.486937976011501: 1, 6.484849409774177: 1, 6.472576900917222: 1,
6.472194431509924: 1, 6.463755938481455: 1, 6.462249264074069: 1, 6.462208372544202: 1,
6.461464487586231: 1, 6.454927581871196: 1, 6.453735188202409: 1, 6.451720405230460: 1,
6.450868546252095: 1, 6.448236387940654: 1, 6.441282935380251: 1, 6.437083277153498: 1,
6.432636798085861: 1, 6.432314585943541 6: 1, 6.431696553014884: 1, 6.425647943937587: 1,
6.412050736312310 5: 1, 6.409204713369606: 1, 6.403529394530735: 1, 6.402749654183564: 1, 6.395904899
9618588: 1, 6.39538485386375: 1, 6.393982832989672: 1, 6.392965559482354: 1, 6.383762171967167: 1,
6.379271692446058: 1, 6.377271923373212: 1, 6.374526243482281: 1, 6.367724863221917: 1,
6.365638238436131: 1, 6.363798371918286: 1, 6.360110049629526: 1, 6.356441822925419: 1,
6.349672418281060 6: 1, 6.348313057054024: 1, 6.346753268859157: 1, 6.344943658506798: 1,
6.344793671776530 5: 1, 6.337165873013639: 1, 6.322008906107813: 1, 6.317941052448127: 1, 6.308604634
46488: 1, 6.303091764093152: 1, 6.302747518023063: 1, 6.301480959336766: 1, 6.295838909258175: 1, 6
.274552537173517: 1, 6.273268453675212: 1, 6.264312831711604: 1, 6.257795131603648: 1,
6.246794808154727: 1, 6.236929921403840 4: 1, 6.233247142320113 5: 1, 6.229072040600027: 1,
6.227702765574663: 1, 6.219470020048486: 1, 6.210310141242846: 1, 6.182129228103325: 1,
6.181969958187565: 1, 6.173064617963864: 1, 6.169967055655259: 1, 6.158968818568889: 1,
6.157649582647923: 1, 6.154420778818703: 1, 6.154393311173855: 1, 6.146639856998214: 1,
6.144837079848674: 1, 6.144494530156770 5: 1, 6.143255809525184: 1, 6.142089578945703: 1,
6.132650890885391: 1, 6.127242697098853: 1, 6.127196561298692 5: 1, 6.126157878288092: 1,
6.124180152466384: 1, 6.120374034668556: 1, 6.119499285219982: 1, 6.117806533469040 4: 1,
6.116030254243932: 1, 6.113710902912561: 1, 6.105692916355521: 1, 6.099743382943006: 1,
6.093250211931622: 1, 6.079952437941108: 1, 6.072078621761918: 1, 6.070847602909128: 1,
6.067564964468666: 1, 6.065477488839646: 1, 6.058056983546818: 1, 6.053168882708494: 1,
6.050108211979004: 1, 6.043607533132614: 1, 6.039875528283789: 1, 6.037793840878304 5: 1,
6.034183940379569: 1, 6.028179159176635: 1, 6.021650155236119: 1, 6.001632267323848: 1,
6.000775276861732: 1, 5.994479273088247: 1, 5.989924911029036 5: 1, 5.986197514037721 5: 1,
5.984898962749972 5: 1, 5.973944019532654: 1, 5.972845175587909: 1, 5.971719477624247: 1, 5.971554282
2844084: 1, 5.969591874607548 5: 1, 5.969120879237472: 1, 5.964791751952129: 1, 5.962626485174034 5:
1, 5.959414050373795: 1, 5.958414500484771: 1, 5.957084248378886: 1, 5.951532979791094: 1,
5.949872145085566: 1, 5.948375913668699: 1, 5.946295282020696: 1, 5.940206035611952: 1,
5.935197794818502: 1, 5.929851697197081: 1, 5.918939974019537: 1, 5.908273138287631 5: 1,
5.896402357156848: 1, 5.887350439135931: 1, 5.884804988908533: 1, 5.883185507681215: 1,
5.882645510971944: 1, 5.877562285787959: 1, 5.872312116296036: 1, 5.871103692249544: 1,
5.870265940636711 5: 1, 5.867746904113195: 1, 5.866240379568137: 1, 5.866144229496655: 1, 5.859858555
5981159: 1, 5.859188674785904: 1, 5.857952261944328 5: 1, 5.857900879879205: 1, 5.857470442907313:
1, 5.857265578886341: 1, 5.855724505385474: 1, 5.84693068899758: 1, 5.845484726459003: 1,
5.829458343864069 5: 1, 5.82703887171815: 1, 5.824148327867448: 1, 5.818977343777875: 1,
5.816193330434535: 1, 5.815318666134377: 1, 5.812038718804703: 1, 5.808570232000180: 1,
5.806391284195158: 1, 5.806315064359885: 1, 5.806110738820116: 1, 5.800018267473418: 1,
5.783926413383710: 1, 5.783637245139146: 1, 5.783082791518261: 1, 5.777220135714081 6: 1,
5.774267637774814: 1, 5.771604965342592 5: 1, 5.771048222534588: 1, 5.766674263805221: 1,
5.764792859868213: 1, 5.761631400084966: 1, 5.759436319805744: 1, 5.753607409287985: 1,
5.751579549473305 5: 1, 5.747068215568149: 1, 5.746259939321546: 1, 5.745766003397046: 1, 5.741554602
2365752: 1, 5.737245688024716: 1, 5.730190665327123: 1, 5.727719740363490: 1, 5.727383522147559: 1,
5.712791741441141: 1, 5.712393009111731: 1, 5.708883408524108: 1, 5.706774578990843: 1,
5.706678097169117: 1, 5.706384036882424: 1, 5.690832435016793: 1, 5.690572301317101: 1,
5.688759133738728: 1, 5.688175433348513: 1, 5.687632324467615: 1, 5.686173299351839: 1,
5.682954786256777 5: 1, 5.676990778068320: 1, 5.671946593031806: 1, 5.669714773503368: 1,
5.668411987604666: 1, 5.661420114097144: 1, 5.653752512997879: 1, 5.651708709834608: 1,
5.642844688364518 5: 1, 5.641685474920735 5: 1, 5.640839063091378: 1, 5.638592709671558: 1,
5.636250334255294: 1, 5.634272698488614: 1, 5.632799440534448: 1, 5.632298963375090: 1,
5.626731137608437: 1, 5.624785798924454: 1, 5.623741773355290: 1, 5.621243201483153: 1,
5.620571158663751: 1, 5.614569326195800 5: 1, 5.604626244801977: 1, 5.600541704828942 5: 1,
5.595831310314539: 1, 5.594808888938788: 1, 5.579085797804034: 1, 5.578506906620358: 1,
5.577641925284682: 1, 5.577445627208546: 1, 5.572727296708440 5: 1, 5.571376480443224: 1,
5.566966864811504: 1, 5.564444373252154 5: 1, 5.563008128028511: 1, 5.558475262741576: 1,
5.554218883843487: 1, 5.551886476465001: 1, 5.551225735216258: 1, 5.537632984266286: 1,
5.535148527766253: 1, 5.528178091202780 5: 1, 5.524768034178771: 1, 5.518583963630648: 1,
5.518150615550132: 1, 5.513172085238886: 1, 5.509814345396347: 1, 5.507350534027872: 1,
5.506668004140982: 1, 5.503651105699991: 1, 5.503293081445777 4: 1, 5.503094526885917: 1,
5.501782547797769: 1, 5.470709919769643: 1, 5.460361607205831: 1, 5.449370610157585: 1,
5.447986426021904: 1, 5.444910954881885: 1, 5.444441012515491: 1, 5.441444496269296 5: 1,
5.439964709287755: 1, 5.437488464107199: 1, 5.436096204387382: 1, 5.433474459211827: 1,
5.432259376409754: 1, 5.430474590517380 5: 1, 5.427503197835938 5: 1, 5.422528190886771: 1,
5.420287035738446: 1, 5.416392769668609: 1, 5.411254759357924 5: 1, 5.409073009445462: 1,
5.407485042361238: 1, 5.406757440938835 5: 1, 5.404823363860078 4: 1, 5.402021173187109 5: 1,
5.400246703810557 5: 1, 5.395220738700878: 1, 5.384765784486563: 1, 5.381028283395834 5: 1,
5.378596638881495: 1, 5.377395053325046 5: 1, 5.374665532766862: 1, 5.367723270496062: 1,
5.366667662846646: 1, 5.366220278123658: 1, 5.360196197520194: 1, 5.359028297357486: 1,
5.358975780945703: 1, 5.356944524966066 5: 1, 5.355152742136378: 1, 5.339696603651260: 1,

5.339691385910712: 1, 5.332944527064564: 1, 5.331556232378751: 1, 5.325362135057308: 1,
5.324750269739023: 1, 5.322536921315426: 1, 5.3215120430228176: 1, 5.317167262564026: 1,
5.312723697524871: 1, 5.301152991893218: 1, 5.296468819274334: 1, 5.294554575205645: 1,
5.291786599854245: 1, 5.289213266954681: 1, 5.279738894948403: 1, 5.277768656539985: 1,
5.272564531706311: 1, 5.2681488795949: 1, 5.2677986807193: 1, 5.2651312791730405: 1,
5.264885780561877: 1, 5.263569730463123: 1, 5.261855818248236: 1, 5.255366762756311: 1,
5.245376693707119: 1, 5.23828315923265: 1, 5.2338777705553525: 1, 5.2338568020719265: 1,
5.2271330078306715: 1, 5.222093976952847: 1, 5.218572948456999: 1, 5.217082558638831: 1, 5.20828296
9541784: 1, 5.204795843725673: 1, 5.199193999630937: 1, 5.199086510179174: 1, 5.198011577888501: 1,
5.181977476005176: 1, 5.176438232969643: 1, 5.1642084197248925: 1, 5.163339282337597: 1,
5.159196655383438: 1, 5.154199851621217: 1, 5.150040778522802: 1, 5.14880170578133: 1,
5.146687212469487: 1, 5.145524461695376: 1, 5.142115638077634: 1, 5.140876528136022: 1,
5.13439401972982: 1, 5.1322684271141785: 1, 5.132110991855535: 1, 5.131802479352444: 1,
5.126279796824176: 1, 5.125965681461976: 1, 5.125667600453308: 1, 5.123127433294794: 1,
5.121117191662843: 1, 5.118183445902968: 1, 5.116145430386011: 1, 5.114752448416472: 1,
5.11307958575913: 1, 5.1119375900726345: 1, 5.1097373124637295: 1, 5.101515415873524: 1,
5.0982480998546595: 1, 5.0917844031104424: 1, 5.088957682893288: 1, 5.085375195891081: 1,
5.084414708391372: 1, 5.077681630130626: 1, 5.075315467310417: 1, 5.0747197889357345: 1,
5.07254443283528: 1, 5.067264370100227: 1, 5.05831478669739: 1, 5.049903177270397: 1,
5.048498353106404: 1, 5.04607101734235: 1, 5.0438077331806515: 1, 5.0421737252772845: 1,
5.041709393481684: 1, 5.040923314129887: 1, 5.040448784908461: 1, 5.03911464846492: 1,
5.036222647938265: 1, 5.033003041524998: 1, 5.032310322926781: 1, 5.029668510358937: 1,
5.028582750469311: 1, 5.023759010208993: 1, 5.021638395249277: 1, 5.019795580448603: 1,
5.019576404258024: 1, 5.016067790773914: 1, 5.014709459114503: 1, 5.014340373230566: 1,
5.014320806081184: 1, 5.012087925284579: 1, 5.008672482700285: 1, 5.003166937697126: 1,
4.991918265320766: 1, 4.98145244409801: 1, 4.9760500840582536: 1, 4.9754297448753 12: 1,
4.961030977763306: 1, 4.959046739264154: 1, 4.9577186262779085: 1, 4.955155085715615: 1,
4.95494930021863: 1, 4.950735257020502: 1, 4.9406608025591625: 1, 4.9378301072241 82: 1,
4.931846341130514: 1, 4.931079132970633: 1, 4.930829288641913: 1, 4.9307067296202 83: 1,
4.929997081865287: 1, 4.929490028410926: 1, 4.919011424617784: 1, 4.9176197476031 22: 1,
4.917144001865098: 1, 4.914612646568 65: 1, 4.914178126802 88: 1, 4.9129357109797 68: 1,
4.909312760619118: 1, 4.908713518959329: 1, 4.9075413712710 59: 1, 4.9071494443182 58: 1,
4.907126559960653: 1, 4.9051919640955415: 1, 4.9050685950146 02: 1, 4.9039014575649205: 1,
4.903598053345924: 1, 4.901671935572441: 1, 4.899830250540395: 1, 4.8982068399856 67: 1,
4.897350118671699: 1, 4.896476242006021: 1, 4.888429265731888: 1, 4.8873299192585 79: 1,
4.886691860084317: 1, 4.886586438934677: 1, 4.883668658585875: 1, 4.8791027689301 32: 1,
4.8743398971880625: 1, 4.873212002848829: 1, 4.8726702635550865: 1, 4.8723808624076 88: 1,
4.86928591548617: 1, 4.865623064499438: 1, 4.862372445662085: 1, 4.8613756753124875: 1,
4.860530322760232: 1, 4.852993296356575: 1, 4.852109421526454: 1, 4.848022500117 62: 1,
4.8451836879233126: 1, 4.843801707222193: 1, 4.8366827788668525: 1, 4.8306972493864 02: 1,
4.829282690242312: 1, 4.815378830322836: 1, 4.815271326087672: 1, 4.8129226259877 22: 1,
4.812628148133769: 1, 4.812556659980711: 1, 4.812260477961264: 1, 4.8117620855870 81: 1,
4.807448257889878: 1, 4.805378501922653: 1, 4.80431860757574: 1, 4.8008393692126 54: 1,
4.798177176607188: 1, 4.797045389601555: 1, 4.791058211071733: 1, 4.7841539182693 16: 1,
4.777253713884985: 1, 4.775473694657911: 1, 4.773469351098314: 1, 4.7701913497688 25: 1,
4.768346024728895: 1, 4.768312178835777: 1, 4.766900459495974: 1, 4.7645869118863 78: 1,
4.763873712765624: 1, 4.762463545963428: 1, 4.761093668367931: 1, 4.7444730805588 21: 1,
4.744445873933351: 1, 4.737072649832358: 1, 4.735995008728488: 1, 4.7345086151641 72: 1,
4.733641274962405: 1, 4.731925826512423: 1, 4.730852099261028: 1, 4.7276999937701 38: 1,
4.727556814831059: 1, 4.72140258909048: 1, 4.7196741469737145: 1, 4.7186933556893385: 1,
4.717170727384271: 1, 4.717002403950102: 1, 4.714320947169594: 1, 4.7131698639599815: 1,
4.711570047384591: 1, 4.710817291605737: 1, 4.708610285263843: 1, 4.7082153255713 17: 1,
4.7050864115276045: 1, 4.704960069212 65: 1, 4.704503743973341: 1, 4.7043019438549 97: 1,
4.703901463954769: 1, 4.702151566650456: 1, 4.701180287956593: 1, 4.6988147885583 19: 1,
4.698410370796322: 1, 4.696510551496548: 1, 4.695769575957668: 1, 4.6920352779007 78: 1,
4.68371398799548: 1, 4.68205582323598: 1, 4.681875188089041: 1, 4.6813197286641 66: 1,
4.680750249095316: 1, 4.678319904402004: 1, 4.677965616651426: 1, 4.6701884748943 54: 1,
4.669632333663684: 1, 4.664138423947872: 1, 4.664081587334025: 1, 4.6636469818258 24: 1,
4.661275559454175: 1, 4.661073381422527: 1, 4.661043482747748: 1, 4.6589796637816 12: 1,
4.658659144767745: 1, 4.656150309842871: 1, 4.6508193260656805: 1, 4.6504099081156 69: 1,
4.648652355109798: 1, 4.647915777689 0805: 1, 4.646546934201654: 1, 4.6439245793715 15: 1,
4.643338848891038: 1, 4.642619058200528: 1, 4.6408277731858885: 1, 4.6363781681048994: 1,
4.634162246815843: 1, 4.630822076645737: 1, 4.627495794829751: 1, 4.622339254885 49: 1,
4.622043255493979: 1, 4.621738977422 59: 1, 4.6203604699216605: 1, 4.619483828674 6: 1,
4.615896120343859: 1, 4.606529054761288: 1, 4.605561552848126: 1, 4.6049936911596 75: 1,
4.600685447261607: 1, 4.5974883057655855: 1, 4.596447369610762: 1, 4.5964242055146 22: 1,
4.59529888046255: 1, 4.592282752514336: 1, 4.5912888048774 55: 1, 4.588257501043214 5: 1,
4.586072784624603: 1, 4.582186549116152: 1, 4.5794534969939376: 1, 4.5789339026783 06: 1,
4.577680792437914: 1, 4.576663048895189: 1, 4.575817143861765: 1, 4.5731233888961 94: 1,
4.567913727058988: 1, 4.560312982315965: 1, 4.558987306666596: 1, 4.5534706240212 08: 1,
4.548673088102177: 1, 4.5470223215029435: 1, 4.545695440925222: 1, 4.5425218816332 94: 1,
4.537528068704363: 1, 4.531299559677702: 1, 4.5312115436911995: 1, 4.5262573809557 45: 1,
4.521758276387219: 1, 4.521231170097496: 1, 4.514404202680861: 1, 4.5139060321433 44: 1,
4.511798635111701: 1, 4.510583618827445: 1, 4.51024600999702: 1, 4.5090840786310 64: 1,
4.504322450829477: 1, 4.503926157483514: 1, 4.500336175973048: 1, 4.4989798450907 75: 1,
4.497838838836569: 1, 4.496154874279455: 1, 4.4941991202174325: 1, 4.4928764141912 95: 1,
4.491896339727356: 1, 4.4895684130235205: 1, 4.486560453269054: 1, 4.4849969097120 41: 1,

4.483989364306241: 1, 4.479024163541899: 1, 4.472509097950202: 1, 4.471940429504053: 1,
4.463678863146975: 1, 4.462417062279139: 1, 4.46185131069246: 1, 4.460899129443088: 1,
4.458319135051776: 1, 4.451952288354699: 1, 4.450143563841159: 1, 4.44870867228119: 1,
4.447976275867133: 1, 4.445619488693897: 1, 4.444448606179204: 1, 4.437165724115978: 1,
4.436551980935215: 1, 4.426283384813851: 1, 4.4262376875231295: 1, 4.423141948233394: 1,
4.4186638418844995: 1, 4.417723541894881: 1, 4.416071816245196: 1, 4.413615283817534: 1,
4.409774220014961: 1, 4.409163289086134: 1, 4.40463138751417: 1, 4.402815996625527: 1,
4.402657637674164: 1, 4.400099475678767: 1, 4.397180983817298: 1, 4.392977407213467: 1,
4.3916640999697405: 1, 4.391006821918677: 1, 4.38795475023046: 1, 4.3841770152276341: 1,
4.38403336346622: 1, 4.382766219552983: 1, 4.382519239568688: 1, 4.381115285512439: 1,
4.38093441917693: 1, 4.379089091677991: 1, 4.377551208449716: 1, 4.375139292091964: 1,
4.372214421257319: 1, 4.370232846722088: 1, 4.359780857969083: 1, 4.35619421987138461: 1,
4.353678894832186: 1, 4.352777633817832: 1, 4.3482345933624496: 1, 4.344310230566271: 1,
4.341947921465521: 1, 4.3418483267047625: 1, 4.3367812236113386: 1, 4.336688467691605: 1,
4.336185544546384: 1, 4.336116746850263: 1, 4.334174119842233: 1, 4.330445556401219: 1,
4.330039020955906: 1, 4.326893428258702: 1, 4.322886022836183: 1, 4.320179574324598: 1,
4.316959694556077: 1, 4.313712219008208: 1, 4.313098735448069: 1, 4.307346743928534: 1,
4.304607592771137: 1, 4.297931452327304: 1, 4.294787707354331: 1, 4.292833206403733: 1,
4.287628452558168: 1, 4.28742662325162: 1, 4.2867415462124181: 1, 4.286089303682499: 1,
4.277735914860132: 1, 4.269078676652155: 1, 4.268456345848884: 1, 4.266832321554204: 1,
4.260516596508663: 1, 4.2603940171078385: 1, 4.2580970025850835: 1, 4.25659548775314: 1,
4.254057022787321: 1, 4.252153322775894: 1, 4.2511289185853425: 1, 4.250648255909654: 1,
4.249980927371811: 1, 4.247826912142825: 1, 4.2461618987633365: 1, 4.241621097381413: 1,
4.237994240979409: 1, 4.237984830098239: 1, 4.231801188177408: 1, 4.225121375664211: 1,
4.225001527128224: 1, 4.220626833839379: 1, 4.209873996510858: 1, 4.209350662809046: 1,
4.206016457957396: 1, 4.205939882006699: 1, 4.205114365712886: 1, 4.200525578813167: 1,
4.1999692028666: 1, 4.199480509769506: 1, 4.198970493679805: 1, 4.191950410985973: 1, 4.19169886579
0254: 1, 4.191682048692628: 1, 4.190642095550866: 1, 4.188776461601256: 1, 4.187241478736631: 1, 4.
186820385166985: 1, 4.186504501383255: 1, 4.179829649173208: 1, 4.179759787427794: 1, 4.17820844080
996: 1, 4.176190193835651: 1, 4.174060319619655: 1, 4.172612266084921: 1, 4.1724632250679: 1,
4.172376399459833: 1, 4.169179863072531: 1, 4.1681739528684681: 1, 4.167532349910755: 1,
4.167492446069257: 1, 4.166860114665993: 1, 4.1645322743630211: 1, 4.158532546549348: 1,
4.156612913185954: 1, 4.154107401992976: 1, 4.153914900850315: 1, 4.150877361085412: 1,
4.141292269274449: 1, 4.1390105725038255: 1, 4.135710491331994: 1, 4.132487689890348: 1,
4.129580674204995: 1, 4.1291314914154515: 1, 4.12891705460276: 1, 4.1230085812412841: 1,
4.121776877241423: 1, 4.121731201960772: 1, 4.117426622748339: 1, 4.1130938643760781: 1,
4.111099345479789: 1, 4.111019581048365: 1, 4.1077129319266845: 1, 4.1067465223990961: 1,
4.10671010733075: 1, 4.1058526847278864: 1, 4.1045452123596196: 1, 4.1036378807561281: 1,
4.103054967261154: 1, 4.1019508530455235: 1, 4.09983168803333: 1, 4.0954308036077361: 1,
4.095215010016664: 1, 4.092662118470895: 1, 4.09110729145: 1, 4.0904590215911565: 1,
4.087568163228935: 1, 4.087361601891977: 1, 4.080016465612647: 1, 4.0780974347210981: 1,
4.077569903381195: 1, 4.075317351974646: 1, 4.072320316434387: 1, 4.0720959356187721: 1,
4.069524638169391: 1, 4.069422320233304: 1, 4.068324956587289: 1, 4.0636616000175361: 1,
4.063262897646965: 1, 4.059086056809563: 1, 4.057965655383182: 1, 4.0478624905556621: 1,
4.046246563729037: 1, 4.042413257312822: 1, 4.039944860261477: 1, 4.0382479169152731: 1,
4.035437278995711: 1, 4.034778568545411: 1, 4.032785304123494: 1, 4.0325527952094555: 1,
4.031343123246181: 1, 4.030981455643915: 1, 4.0299018028415485: 1, 4.0246685200318671: 1,
4.021547014333152: 1, 4.017582612023062: 1, 4.012963366637967: 1, 4.0112261113762875: 1,
4.010785836894342: 1, 4.010039080015857: 1, 4.007177321443606: 1, 4.0068953662667531: 1,
4.006271113111563: 1, 4.005822791384095: 1, 4.003539267332393: 1, 4.0010871218118671: 1,
4.000215289938729: 1, 3.9931567383859: 1, 3.9907187331770495: 1, 3.9865168882350037: 1,
3.9837978347109853: 1, 3.981050053378222: 1, 3.978362445522496: 1, 3.977874134406148: 1, 3.97614362
17848547: 1, 3.976044331585522: 1, 3.973331119377355: 1, 3.9706683573349975: 1, 3.96901196009443:
1, 3.960044011449588: 1, 3.9593606542241035: 1, 3.958892500091543: 1, 3.9583344859531361: 1,
3.953120953053713: 1, 3.9528693889578115: 1, 3.952768851396277: 1, 3.9521971924131694: 1,
3.950852104913573: 1, 3.950131096007616: 1, 3.9481045939054975: 1, 3.944857871120973: 1,
3.941745918097084: 1, 3.939896753703729: 1, 3.936757372372553: 1, 3.9333830369913111: 1,
3.9323226612961055: 1, 3.9298442315004407: 1, 3.928148754903612: 1, 3.9242927961209237: 1,
3.9232351499779483: 1, 3.919658792239799: 1, 3.916485475105058: 1, 3.9159085455405571: 1, 3.91163330
82887618: 1, 3.911210815791307: 1, 3.9087133192871644: 1, 3.9065874594639673: 1,
3.901862764975665: 1, 3.8998678897024615: 1, 3.897264595564887: 1, 3.8938208286978786: 1,
3.881104129600298: 1, 3.880979685224987: 1, 3.878276458792138: 1, 3.8770053823335933: 1,
3.875657754053698: 1, 3.8727688107583997: 1, 3.871533369714571: 1, 3.8703865856286221: 1,
3.869684348108568: 1, 3.869245775199919: 1, 3.869158992654537: 1, 3.8668156944542231: 1,
3.865493179796143: 1, 3.8614959395780284: 1, 3.8604949777813946: 1, 3.8598171166229381: 1,
3.8594773481914175: 1, 3.8555899086681475: 1, 3.8553111387262136: 1, 3.8543667623865051: 1,
3.853721148325391: 1, 3.852968199239035: 1, 3.8497123094396093: 1, 3.8494003935633121: 1,
3.847183920730697: 1, 3.8462253567181905: 1, 3.8433344382594936: 1, 3.8392777258554784: 1,
3.838405815418881: 1, 3.8383738087534196: 1, 3.8354472454360256: 1, 3.8347494395581005: 1,
3.832872330329562: 1, 3.832189822242302: 1, 3.8321637693754917: 1, 3.8314608857801313: 1,
3.8274794946980832: 1, 3.818393356148403: 1, 3.811980997111826: 1, 3.8086142291848261: 1, 3.80738112
0956407: 1, 3.807225378034486: 1, 3.802315705922808: 1, 3.8004270014119945: 1, 3.7994704771485623:
1, 3.7987587513209315: 1, 3.7976941687446084: 1, 3.7927076219227893: 1, 3.7919806244421541: 1,
3.7865057736516903: 1, 3.781567019894376: 1, 3.7804030702048945: 1, 3.7798200439574052: 1,
3.7746289137435354: 1, 3.7736984710776427: 1, 3.7720618216859743: 1, 3.7703366047226368: 1,
3.7640585780298281: 1, 3.7639498706137151: 1, 3.7635720256074183: 1, 3.7617956109046475: 1,
3.7610368110251233: 1, 3.759384526864428: 1, 3.758647035884251: 1, 3.7575491626528721: 1, 3.75386309

4583696: 1, 3.7534450995722843: 1, 3.7530467097455813: 1, 3.7526781384490464: 1,
3.7470254075580183: 1, 3.745890991225129: 1, 3.743934162556463: 1, 3.7436637507454513: 1,
3.743539072935134: 1, 3.738587779551163: 1, 3.73683504275808: 1, 3.7326881444736357: 1,
3.7299190341345754: 1, 3.727454387163091: 1, 3.72675936562977: 1, 3.726343265180801: 1,
3.721510215530393: 1, 3.720165017751658: 1, 3.7144809331083746: 1, 3.7129731629439195: 1,
3.7122221233255006: 1, 3.709653121083112: 1, 3.705247183169233: 1, 3.7038723883950717: 1,
3.6965620776697214: 1, 3.6960011869180565: 1, 3.6954606701385337: 1, 3.6949862036778724: 1,
3.6948161430115594: 1, 3.689784713878447: 1, 3.6852683530962507: 1, 3.6850896742596698: 1,
3.6825940252025244: 1, 3.6825700333563374: 1, 3.681691074605588: 1, 3.678220990165061: 1,
3.6757616732914404: 1, 3.675556546200086: 1, 3.6723356551103374: 1, 3.6713273331755873: 1,
3.670349055167623: 1, 3.6656828237960486: 1, 3.657286048457233: 1, 3.6568466012746343: 1,
3.6556637549082955: 1, 3.654890448625863: 1, 3.6538426971429137: 1, 3.6496823759627297: 1,
3.6477233969897895: 1, 3.646629048518728: 1, 3.640145424071912: 1, 3.638578044218652: 1, 3.6384564753835575: 1, 3.6380338483063497: 1, 3.6368033536867954: 1, 3.636428439486957: 1,
3.6349837324190104: 1, 3.6347633036192404: 1, 3.6343197257958013: 1, 3.630806371536332: 1,
3.6288404978626043: 1, 3.6283327781708072: 1, 3.627553887270l8: 1, 3.6268692917471017: 1,
3.6261433588306855: 1, 3.623383640692412: 1, 3.622713417987693: 1, 3.62233424543838003: 1,
3.6223021670620152: 1, 3.6196872424961155: 1, 3.618354559147122: 1, 3.6178852889932775: 1,
3.61467946227271: 1, 3.614340266898042: 1, 3.612101944724553: 1, 3.6089447008880904: 1,
3.6083504109755045: 1, 3.605854558936667: 1, 3.6056646160895167: 1, 3.6050994229362905: 1,
3.5993466646295373: 1, 3.5982527227208525: 1, 3.5979484359631564: 1, 3.5975463605476548: 1,
3.597045195289256: 1, 3.595776096745552: 1, 3.594208763010964: 1, 3.5922873098315296: 1,
3.5884770012761993: 1, 3.5827385052846803: 1, 3.582072501039512: 1, 3.582028494534832: 1,
3.5807325123020166: 1, 3.5762271075407677: 1, 3.576150825816691: 1, 3.575855622475611: 1,
3.5704008723083644: 1, 3.569439217563551: 1, 3.5652313654297436: 1, 3.5647118811349787: 1,
3.5646977620106353: 1, 3.5618962922291884: 1, 3.5606744294612898: 1, 3.560156611913135: 1,
3.5600605205384244: 1, 3.554059176807567: 1, 3.5488273670908894: 1, 3.5481389695045085: 1,
3.537352176453563: 1, 3.5371520637646006: 1, 3.5369101364937534: 1, 3.5341801443414704: 1,
3.5327635177624246: 1, 3.5316807005268998: 1, 3.5314405221087823: 1, 3.5304049133487605: 1,
3.526957497523514: 1, 3.5243433412218366: 1, 3.5217473072272063: 1, 3.5174865262716843: 1,
3.512967302796967: 1, 3.5112213920218465: 1, 3.510795526279925: 1, 3.5102257924282907: 1,
3.5101646124227845: 1, 3.5069205536839068: 1, 3.5066132743954865: 1, 3.4992652331511462: 1,
3.497530130627782: 1, 3.4961387352591355: 1, 3.4920991782535786: 1, 3.490332407571988: 1,
3.488359906942961: 1, 3.4871751761231087: 1, 3.485735870673054: 1, 3.4857314093926406: 1,
3.484882182638263: 1, 3.484299255554187: 1, 3.480396498280325: 1, 3.47496520513418351: 1,
3.474434342530658: 1, 3.4732205115219l8: 1, 3.4703334693652663: 1, 3.4671759209240234: 1,
3.4664015290152044: 1, 3.4660123719809994: 1, 3.4656123478239786: 1, 3.4582080053414392: 1,
3.4575156964480596: 1, 3.4565765788708154: 1, 3.454387808931459: 1, 3.451847604325299: 1,
3.44978698162077: 1, 3.4481862633950637: 1, 3.4449570226092674: 1, 3.4405587136898688: 1,
3.4384058331488525: 1, 3.4380768031401288: 1, 3.4351482607287678: 1, 3.4297611629813356: 1,
3.4293485933633083: 1, 3.4280335084269518: 1, 3.426024434876305: 1, 3.42110549891367l8: 1,
3.4187067269478986: 1, 3.4182444389961786: 1, 3.4107852355158808: 1, 3.410597248826941: 1,
3.4097177700170707: 1, 3.4072512886659356: 1, 3.4047887055637114: 1, 3.4009836443672303: 1,
3.3994546918162634: 1, 3.3987173953610212: 1, 3.3968868574099353: 1, 3.3961368335846176: 1,
3.395996402684319: 1, 3.3943158362584573: 1, 3.381987864883934: 1, 3.379799784101035: 1,
3.3794863122358922: 1, 3.3794191848188673: 1, 3.3788850155716554: 1, 3.377902127779429: 1,
3.3770204871136444: 1, 3.373070021624296: 1, 3.3718438262957915: 1, 3.370989804753043: 1,
3.3699527365875137: 1, 3.368148330454528: 1, 3.3658084954013274: 1, 3.362337589230008: 1,
3.3586074103810057: 1, 3.3530331584978708: 1, 3.3526591235321406: 1, 3.351142667250509: 1,
3.3486900829709847: 1, 3.3451151093705134: 1, 3.3437006990121168: 1, 3.3427211871628115: 1,
3.3325753701601744: 1, 3.3306629650632726: 1, 3.329955526573291: 1, 3.3287747076992793: 1,
3.324975282087017: 1, 3.324389399648731: 1, 3.320392565873806: 1, 3.320361857508173: 1,
3.3165026701452684: 1, 3.313507824693782: 1, 3.312121147314934: 1, 3.302728638148871: 1, 3.29800654786248: 1, 3.2972655317193067: 1, 3.296819749685588: 1, 3.2923183185788893: 1, 3.2869219083792123: 1, 3.285624388872015: 1, 3.28546324767722: 1, 3.2798269968647675: 1, 3.2755590043872687: 1,
3.2747453539812335: 1, 3.2718320612165446: 1, 3.2700932622990586: 1, 3.2652278689600207: 1,
3.264764308597342: 1, 3.2617739810690174: 1, 3.261474019733623: 1, 3.260765365907239: 1,
3.260532001739784: 1, 3.26027929129943O5: 1, 3.259144289845601: 1, 3.2529829347603814: 1,
3.2518540575377943: 1, 3.2413881264004925: 1, 3.2405971971879604: 1, 3.239169314802O2: 1,
3.238723832772746: 1, 3.2374517944943206: 1, 3.2352974810494497: 1, 3.2335804317594143: 1,
3.2313048928304124: 1, 3.2304063000756957: 1, 3.2262777894392523: 1, 3.2229127979079295: 1,
3.220981278731481: 1, 3.2190938474521076: 1, 3.218479319467151: 1, 3.217305782783398: 1,
3.217177268350238: 1, 3.2147681222798234: 1, 3.2139380681869736: 1, 3.21063481141765l5: 1,
3.209653907760345: 1, 3.204183850942991: 1, 3.2003801919375614: 1, 3.198488284294326: 1,
3.1957482688079604: 1, 3.192263270124098: 1, 3.1919469516069965: 1, 3.1855741095946555: 1,
3.1841639784868745: 1, 3.1826567253860705: 1, 3.17944911784852: 1, 3.1773465089309125: 1,
3.1743533189305375: 1, 3.1707930792090577: 1, 3.167085505078798: 1, 3.165497368352879: 1,
3.161952618174779: 1, 3.160291244777555: 1, 3.1597595092284005: 1, 3.1577291434228627: 1,
3.1554706592849984: 1, 3.152440117346223: 1, 3.1509626529831714: 1, 3.148000548600834: 1,
3.1475077679682233: 1, 3.1431611324998734: 1, 3.1394838991655223: 1, 3.136907880258449: 1,
3.1357634375213306: 1, 3.135525829716675: 1, 3.1353335986915556: 1, 3.133462178098411: 1,
3.1312395701565627: 1, 3.1308972646004185: 1, 3.130666010264747: 1, 3.1304797202547223: 1,
3.1272480631525306: 1, 3.1241586253342786: 1, 3.1183201941913126: 1, 3.118250444249866: 1,
3.117873876223365: 1, 3.1171199761453012: 1, 3.1118500690013917: 1, 3.1110222505742464: 1,
3.109804185199657: 1, 3.107852402811704: 1, 3.10714652964897: 1, 3.1016888529414643: 1,
3.0992410438247817: 1, 3.0974375141324266: 1, 3.096073785807393: 1, 3.0935514228069643: 1,
3.0932821424790764: 1, 3.0903822267438223: 1, 3.0899836972024204: 1, 3.0861851254747847: 1,

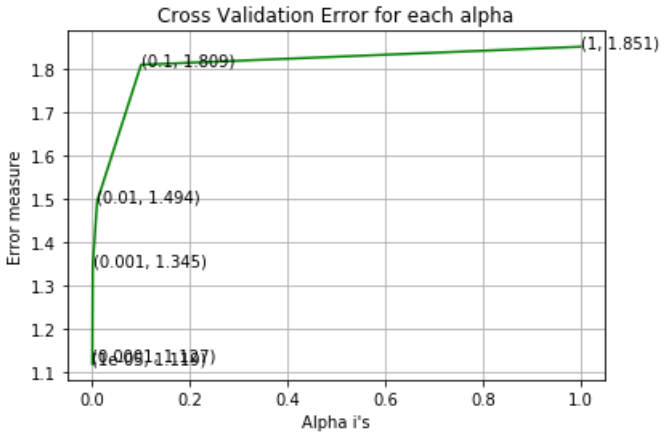
```
3.0861523554916652: 1, 3.083717665814368: 1, 3.0804443308484717: 1, 3.0800421986670847: 1,
3.0797212710247392: 1, 3.075690228202681: 1, 3.074567367277718: 1, 3.0737358350323345: 1,
3.0709448036302422: 1, 3.0649118270563593: 1, 3.062932362486373: 1, 3.062706118965942: 1,
3.0616366627806544: 1, 3.056512187602062: 1, 3.0537195388765452: 1, 3.0520140877778847: 1,
3.049323773706987: 1, 3.0493120297844327: 1, 3.046669991120177: 1, 3.0461389876460823: 1,
3.0417854137815636: 1, 3.0417798115943246: 1, 3.040922789240592: 1, 3.0399482562790077: 1,
3.0398422428847245: 1, 3.037420026169071: 1, 3.034811661963554: 1, 3.030146189342527: 1, 3.02957691
0681174: 1, 3.0286253968438652: 1, 3.026300683944909: 1, 3.0262707646548126: 1,
3.0240541309918965: 1, 3.023638232286665: 1, 3.0156963254325633: 1, 3.012279665822892: 1,
3.0115249924016347: 1, 3.011023411531579: 1, 3.0048263660295653: 1, 3.0004589631407073: 1,
2.9980309392241438: 1, 2.9966973728588773: 1, 2.994061726741822: 1, 2.9934868849816927: 1,
2.992933689256852: 1, 2.9907680540535835: 1, 2.984379285483132: 1, 2.9822520538096087: 1,
2.978154992825708: 1, 2.974029295664955: 1, 2.960386506796161: 1, 2.9601240299535916: 1,
2.957637477801147: 1, 2.9567983212105435: 1, 2.9557307201041154: 1, 2.9521399503233603: 1,
2.950947851845439: 1, 2.9488240578223692: 1, 2.9476161690607396: 1, 2.946269374971374: 1,
2.945630714056111: 1, 2.9436034706110075: 1, 2.940674018101056: 1, 2.939917667813112: 1,
2.933055785970827: 1, 2.926833817532184: 1, 2.9255166081935524: 1, 2.9216119039677055: 1,
2.9210070975962816: 1, 2.9191676025900675: 1, 2.9189576491765807: 1, 2.917439034998208: 1,
2.9150798918278547: 1, 2.908104613511234: 1, 2.9079823437013905: 1, 2.9027441797924469: 1,
2.9013861637079112: 1, 2.901140647441067: 1, 2.89805886044232: 1, 2.8976619226013938: 1,
2.8975684843325227: 1, 2.896542800959738: 1, 2.895893678106499: 1, 2.8945533087424313: 1,
2.8912776033087453: 1, 2.8902233510455324: 1, 2.889615536932021: 1, 2.889339519502321: 1,
2.8890592280166865: 1, 2.8872310841703186: 1, 2.8790307618057893: 1, 2.8775146730179615: 1,
2.8768799060685066: 1, 2.875757841569975: 1, 2.8750386868079527: 1, 2.8741530557202855: 1,
2.867761908573341: 1, 2.867452754120391: 1, 2.8653986137015686: 1, 2.8644242963995725: 1,
2.860795058904656: 1, 2.8571022819768084: 1, 2.855161532002004: 1, 2.8540518579889174: 1,
2.853834149454351: 1, 2.85238378609457: 1, 2.849103767311126: 1, 2.8449434869261196: 1,
2.842032919014637: 1, 2.840610129928097: 1, 2.839434760743826: 1, 2.839134897277836: 1,
2.8330452600760245: 1, 2.8322332141324864: 1, 2.8312539155123737: 1, 2.8208762781087406: 1,
2.820750619192734: 1, 2.8145130319569933: 1, 2.8135186260861613: 1, 2.8133447371222737: 1,
2.811939074829869: 1, 2.80485110903064: 1, 2.8035465518083256: 1, 2.8013224357884488: 1,
2.7987768205515264: 1, 2.797536481499931: 1, 2.7959395791984614: 1, 2.794431111871909: 1,
2.7942143376633566: 1, 2.7935984565697782: 1, 2.7928879603751753: 1, 2.7927960549606166: 1,
2.787895646935108: 1, 2.7870370992854827: 1, 2.7841051220666073: 1, 2.7798721477292516: 1,
2.776746215794436: 1, 2.770883943679697: 1, 2.7692532790394857: 1, 2.767855959812523: 1,
2.763790861642315: 1, 2.762700413279549: 1, 2.76250781119632: 1, 2.7618773212736296: 1,
2.761777843645369: 1, 2.7573587603516794: 1, 2.7541098316548753: 1, 2.753408419122978: 1,
2.7530390966685516: 1, 2.7474834273105793: 1, 2.7440293736845676: 1, 2.7372249249669376: 1,
2.731487571261661: 1, 2.7213711924110737: 1, 2.721369900498962: 1, 2.7195806177807844: 1,
2.7181443806253474: 1, 2.7148329867791845: 1, 2.7144326510120442: 1, 2.712308410242466: 1,
2.70962370277463: 1, 2.7059698410742477: 1, 2.6971656324112394: 1, 2.693999971464855: 1,
2.6918343175843664: 1, 2.688428580995178: 1, 2.687407106665554: 1, 2.6858229153636746: 1,
2.685666214716061: 1, 2.682754252522804: 1, 2.675871862489729: 1, 2.6751272840136506: 1,
2.673010185893701: 1, 2.667121972488666: 1, 2.666965723225849: 1, 2.6629213981568927: 1,
2.662218410186827: 1, 2.6617844783632445: 1, 2.654531968311135: 1, 2.6542478778670437: 1,
2.650473578060973: 1, 2.645862315056915: 1, 2.6403638650041135: 1, 2.632103628145681: 1,
2.6281826550521283: 1, 2.6273943287929997: 1, 2.6263429930175883: 1, 2.6245754608139675: 1,
2.623719849413574: 1, 2.6205481271766757: 1, 2.6117431042588053: 1, 2.6058497923366812: 1,
2.605656103234748: 1, 2.6040616903894347: 1, 2.6020854205776507: 1, 2.6006882615300118: 1,
2.5996671741146224: 1, 2.598796757372083: 1, 2.5957428086135765: 1, 2.594219632532045: 1,
2.5906588198175786: 1, 2.589349107605132: 1, 2.5831808869625115: 1, 2.580192470378162: 1,
2.5798068584637837: 1, 2.575207122021456: 1, 2.5670836985804333: 1, 2.566331496937709: 1,
2.566013101674256: 1, 2.5620719128382015: 1, 2.561588625722796: 1, 2.5603522792443236: 1,
2.5582221682047805: 1, 2.5521224869909935: 1, 2.5505556817730306: 1, 2.5463594660849163: 1,
2.5438506372064875: 1, 2.5425113432877366: 1, 2.539883402210862: 1, 2.53626310310611: 1,
2.5353253411981513: 1, 2.5323697027878938: 1, 2.5320704596474544: 1, 2.5295239282488358: 1,
2.5287760083371222: 1, 2.5284445426923057: 1, 2.5159334867487537: 1, 2.5006719484995: 1,
2.4909875473705623: 1, 2.467436978173619: 1, 2.465069907662842: 1, 2.4547396976798455: 1,
2.4541714292999135: 1, 2.4499879824901716: 1, 2.4403331494024028: 1, 2.4343917486503766: 1,
2.4234565807293413: 1, 2.4132337951310596: 1, 2.4111702443716845: 1, 2.393367709517562: 1,
2.3886582085915875: 1, 2.3850844915931977: 1, 2.371142047013985: 1, 2.3341136900811286: 1,
2.3282715546314554: 1, 2.2850192260585036: 1, 2.275598046794772: 1, 2.2365788317271322: 1,
2.2099372630222156: 1, 2.2054678163180603: 1, 2.134950590499209: 1})
```

In [50]:

```
# 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: 1.1189206318954048
For values of alpha =  0.0001 The log loss is: 1.1273473428032597
For values of alpha =  0.001 The log loss is: 1.3453162512481818
For values of alpha =  0.01 The log loss is: 1.4940339625273966
For values of alpha =  0.1 The log loss is: 1.8091403807417321
For values of alpha =  1 The log loss is: 1.850661663084213
```



```
For values of best alpha =  1e-05 The train log loss is: 0.7765333595900742
For values of best alpha =  1e-05 The cross validation log loss is: 1.1189206318954048
For values of best alpha =  1e-05 The test log loss is: 1.1626741498989979
```

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

Ans. Yes, it seems like!

In [51]:

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

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

In [52]:

```
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")
```

```
8.776 % of word of test data appeared in train data
9.698 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [53]:

```
#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 [54]:

```
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 [55]:

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

```
    text_count_vec = CountVectorizer(min_df=3)

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

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

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

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

Stacking the three types of features

In [56]:

```
# 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 [57]:

```
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, 4690)
(number of data points * number of features) in test data =  (665, 4690)
(number of data points * number of features) in cross validation data = (532, 4690)
```

In [58]:

```
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

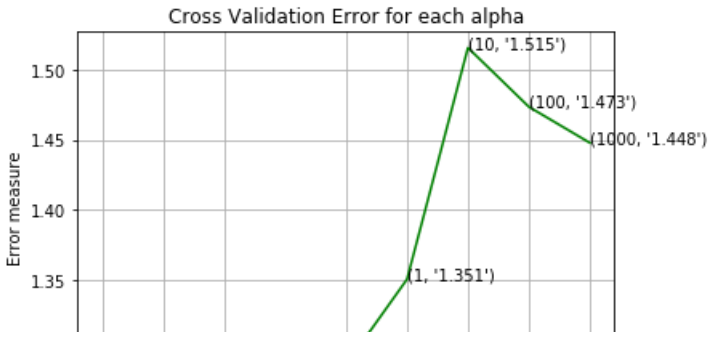
4.1.1.1. Hyper parameter tuning

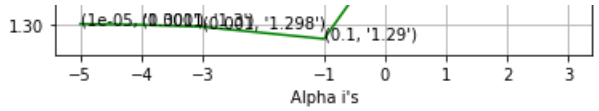
In [59]:

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

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


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

```
# 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 : 1.3005272634016205
for alpha = 0.0001
Log Loss : 1.2999068962051155
for alpha = 0.001
Log Loss : 1.2984863597893126
for alpha = 0.1
Log Loss : 1.2899091571643846
for alpha = 1
Log Loss : 1.350748172199648
for alpha = 10
Log Loss : 1.5154197061194394
for alpha = 100
Log Loss : 1.4733754035430038
for alpha = 1000
Log Loss : 1.4477901258489725
```





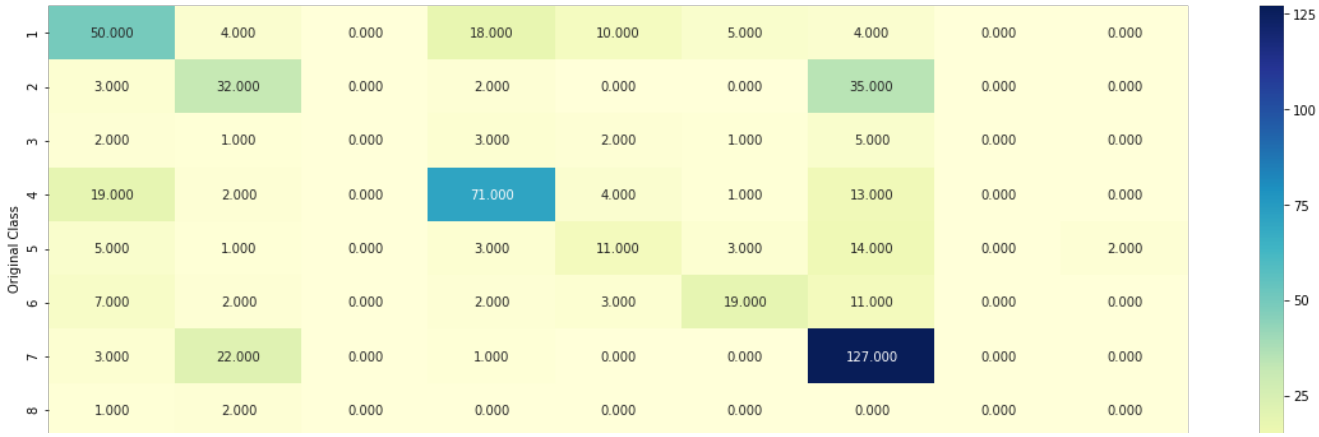

```
For values of best alpha =  0.1 The train log loss is: 0.7186341139931389
For values of best alpha =  0.1 The cross validation log loss is: 1.2899091571643846
For values of best alpha =  0.1 The test log loss is: 1.2865988601230922
```

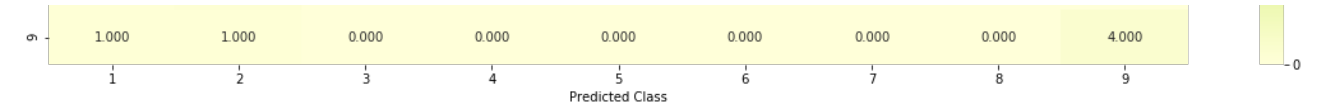
4.1.1.2. Testing the model with best hyper paramters

In [60]:

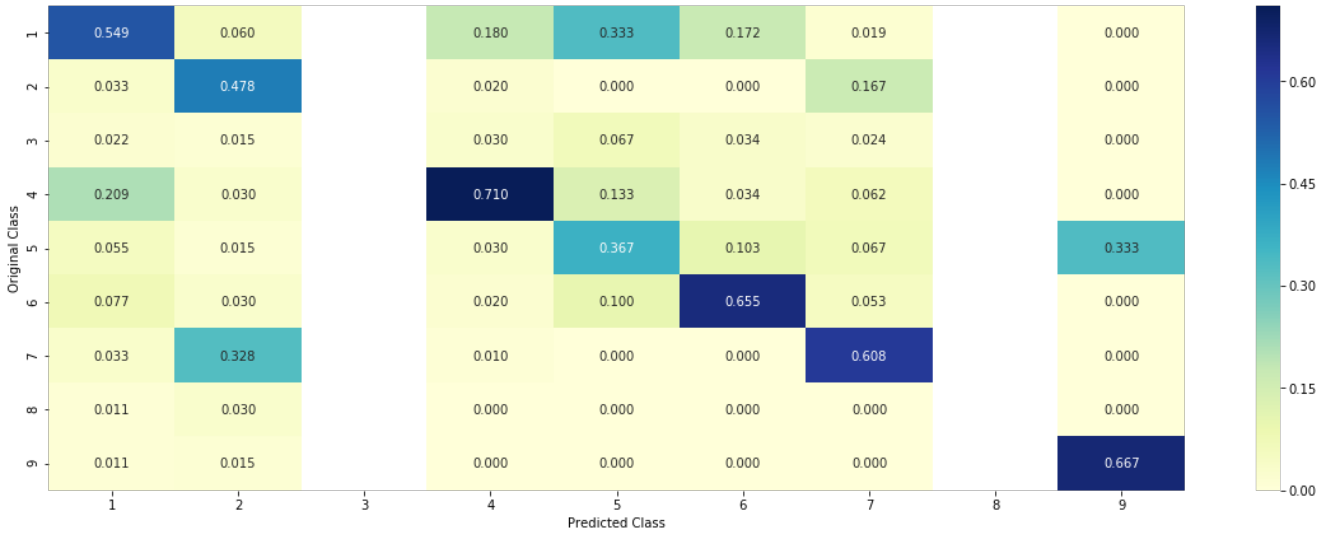
```
# find more about Multinomial Naive base function here http://scikit-
learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# 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 : 1.2899091571643846
Number of missclassified point : 0.40977443609022557
-------------------- Confusion matrix --------------------
```

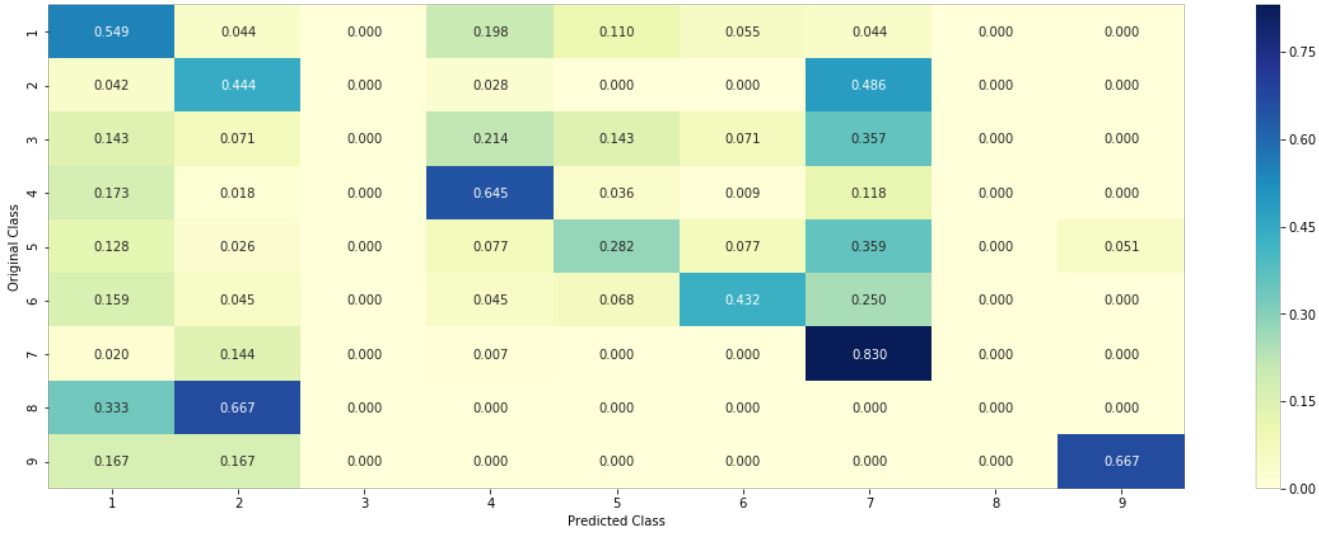





```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



4.1.1.3. Feature Importance, Correctly classified point

In [62]:

```
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: [[0.0494 0.0543 0.0152 0.0635 0.0376 0.0405 0.7315 0.005  0.003 ]]
Actual Class : 6
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
27 Text feature [2012] present in test data point [True]
30 Text feature [150] present in test data point [True]
Out of the top  100  features  2 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [63]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0772 0.0728 0.0201 0.1097 0.286  0.0538 0.3699 0.0066 0.0039]]
Actual Class : 5
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.2. K Nearest Neighbour Classification

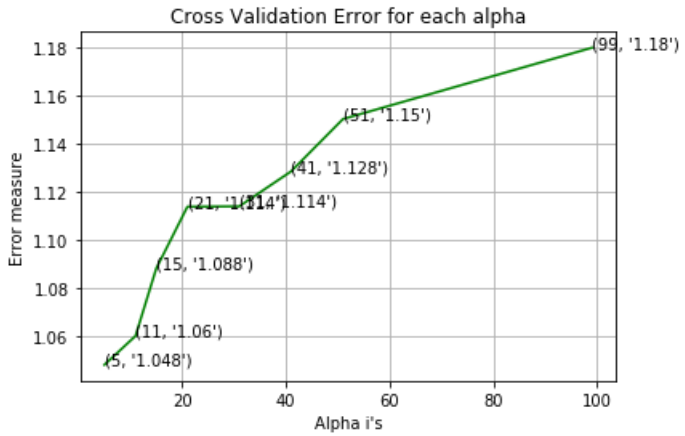
4.2.1. Hyper parameter tuning

In [64]:

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

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


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


alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
```

```
    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 : 1.047808670743755
for alpha = 11
Log Loss : 1.0598526387579437
for alpha = 15
Log Loss : 1.08797964618932
for alpha = 21
Log Loss : 1.1137413397767932
for alpha = 31
Log Loss : 1.1138987406874556
for alpha = 41
Log Loss : 1.1284829311564177
for alpha = 51
Log Loss : 1.1503086264492854
for alpha = 99
Log Loss : 1.180048740902868
```



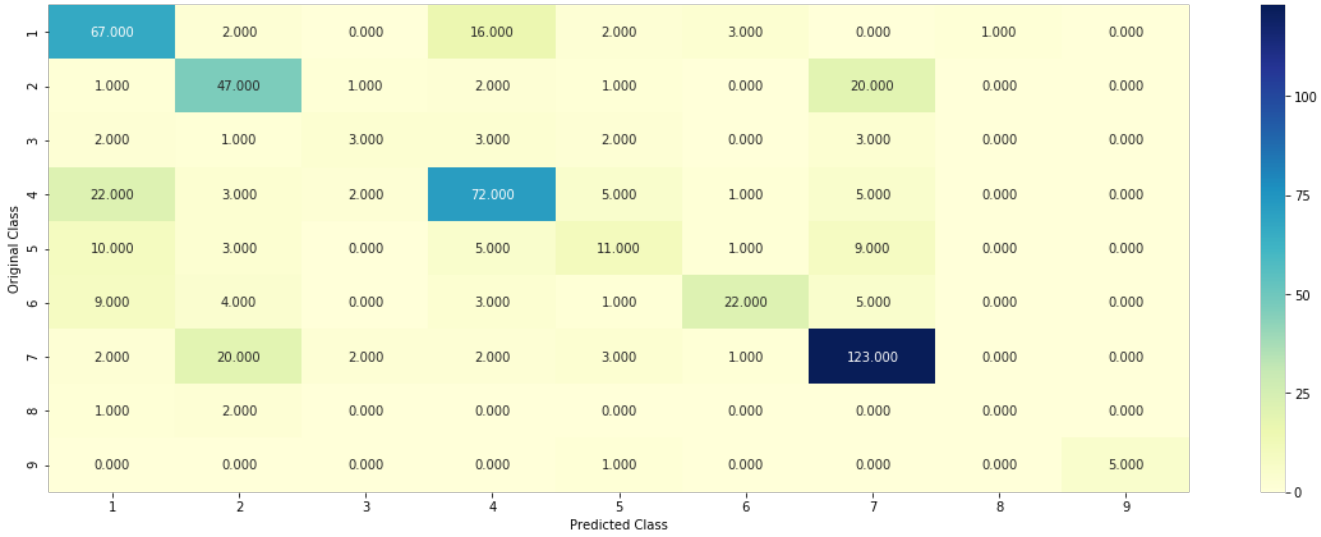

```
For values of best alpha =  5 The train log loss is: 0.49429829963986793
For values of best alpha =  5 The cross validation log loss is: 1.047808670743755
For values of best alpha =  5 The test log loss is: 1.1046606348428416
```

4.2.2. Testing the model with best hyper paramters

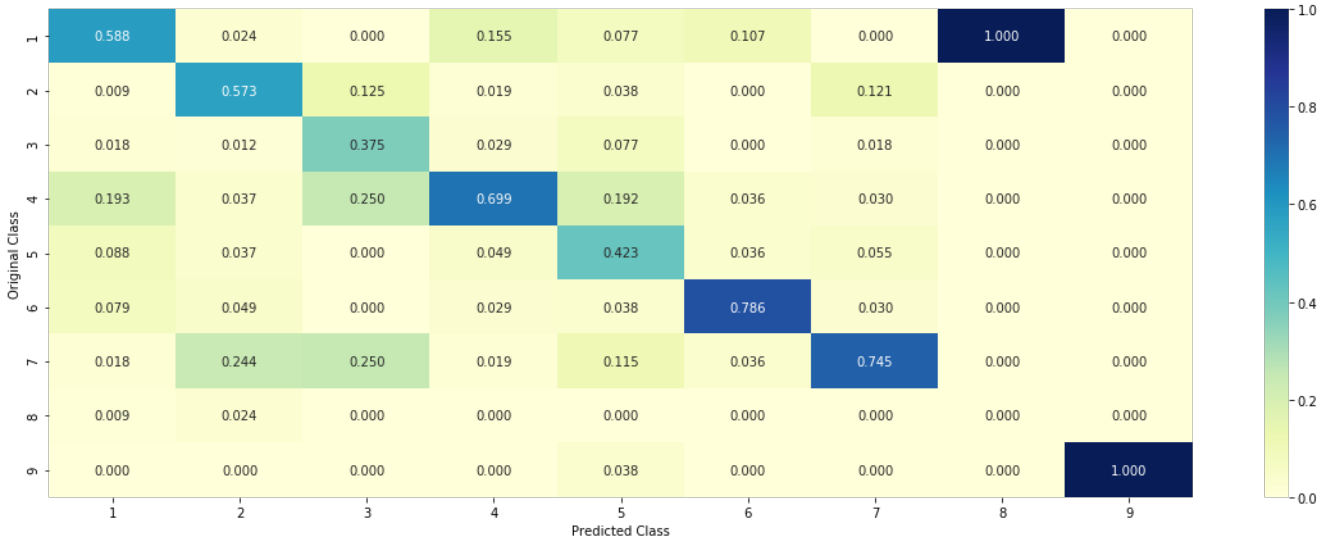
In [65]:

```
# 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)
```

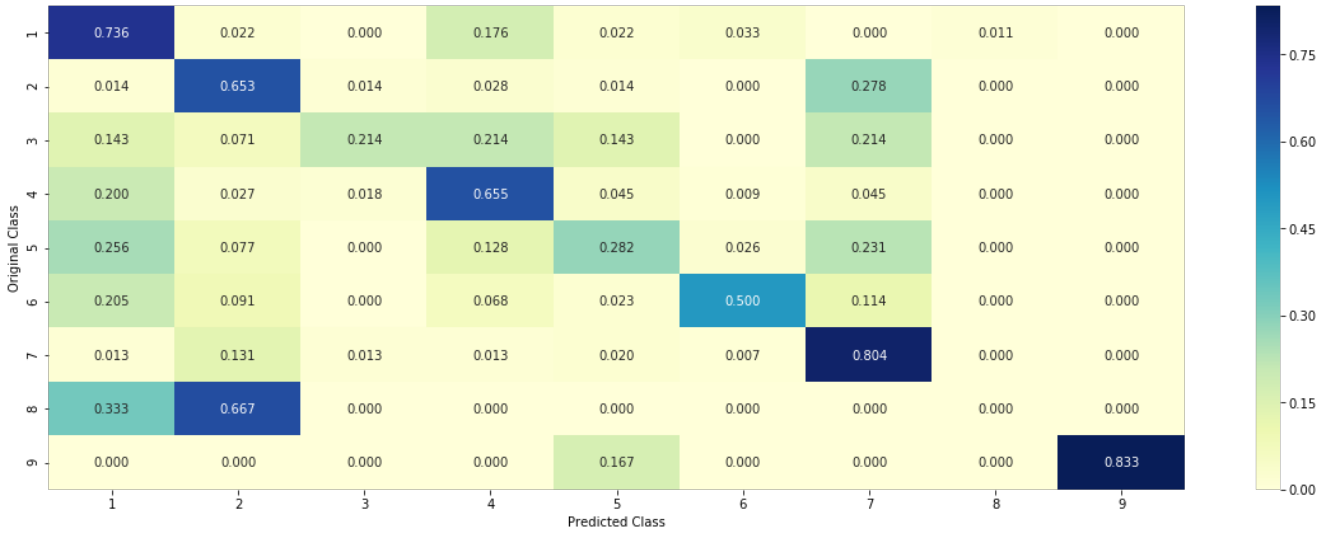
```
Log loss : 1.047808670743755
Number of mis-classified points : 0.34210526315789475
-------------------- Confusion matrix --------------------
```




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




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



4.2.3.Sample Query point -1

In [66]:

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

test_point_index = 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 : 3
Actual Class : 6
The  5  nearest neighbours of the test points belongs to classes [7 6 7 7 7]
Fequency of nearest points : Counter({7: 4, 6: 1})
```

4.2.4. Sample Query Point-2

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 = 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 : 8
Actual Class : 5
the k value for knn is 5 and the nearest neighbours of the test points belongs to classes [5 9 4 8
5]
Fequency of nearest points : Counter({5: 2, 9: 1, 4: 1, 8: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

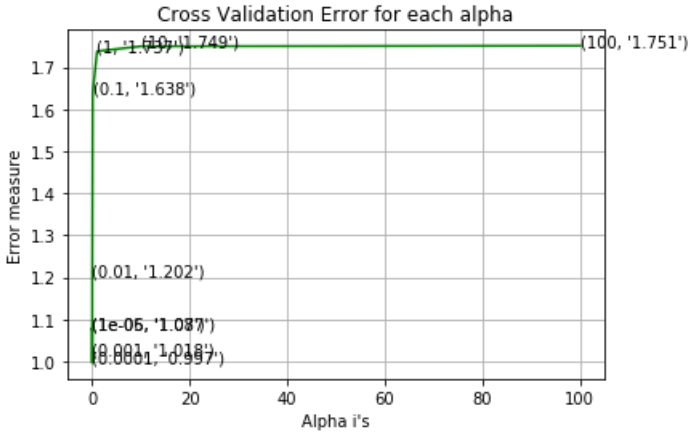
4.3.1.1. Hyper paramter tuning

In [68]:

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

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

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


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

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

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


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

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

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

```
for alpha = 1e-06
Log Loss : 1.0768538182431966
for alpha = 1e-05
Log Loss : 1.079668386972885
for alpha = 0.0001
Log Loss : 0.9968945652075092
for alpha = 0.001
Log Loss : 1.0176630491153176
for alpha = 0.01
Log Loss : 1.2021858911826744
for alpha = 0.1
Log Loss : 1.6383426678472262
for alpha = 1
Log Loss : 1.7374527537801951
for alpha = 10
Log Loss : 1.7493062782600493
for alpha = 100
Log Loss : 1.7506212799394831
```




```
For values of best alpha =  0.0001 The train log loss is: 0.4135949678702322
For values of best alpha =  0.0001 The cross validation log loss is: 0.9968945652075092
For values of best alpha =  0.0001 The test log loss is: 1.0568102900667553
```

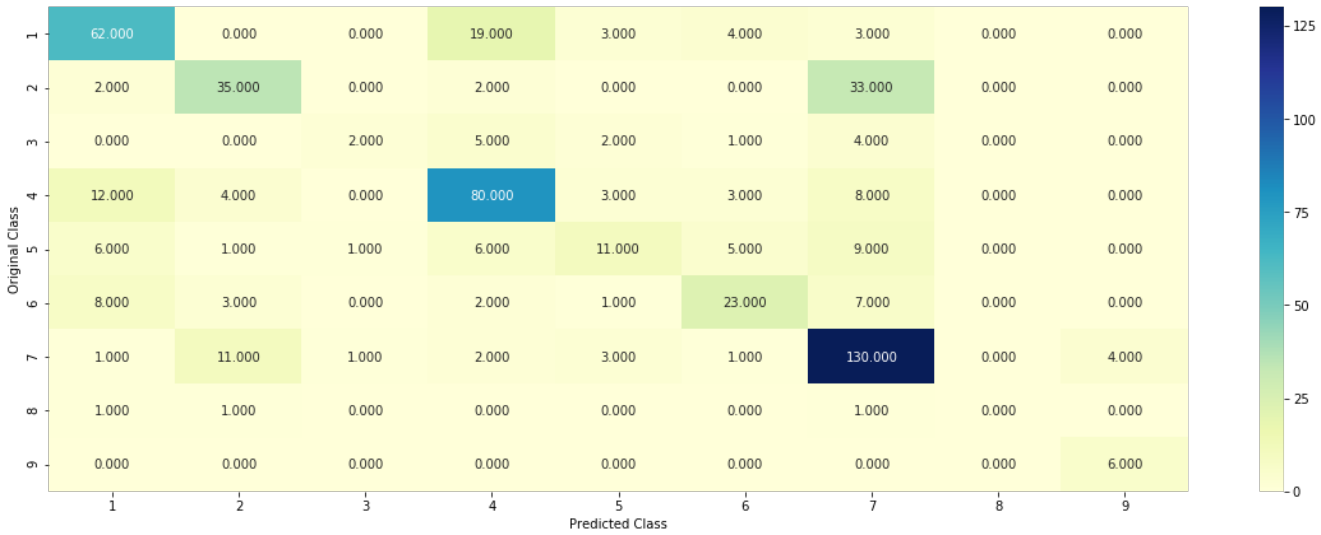
4.3.1.2. Testing the model with best hyper paramters

In [69]:

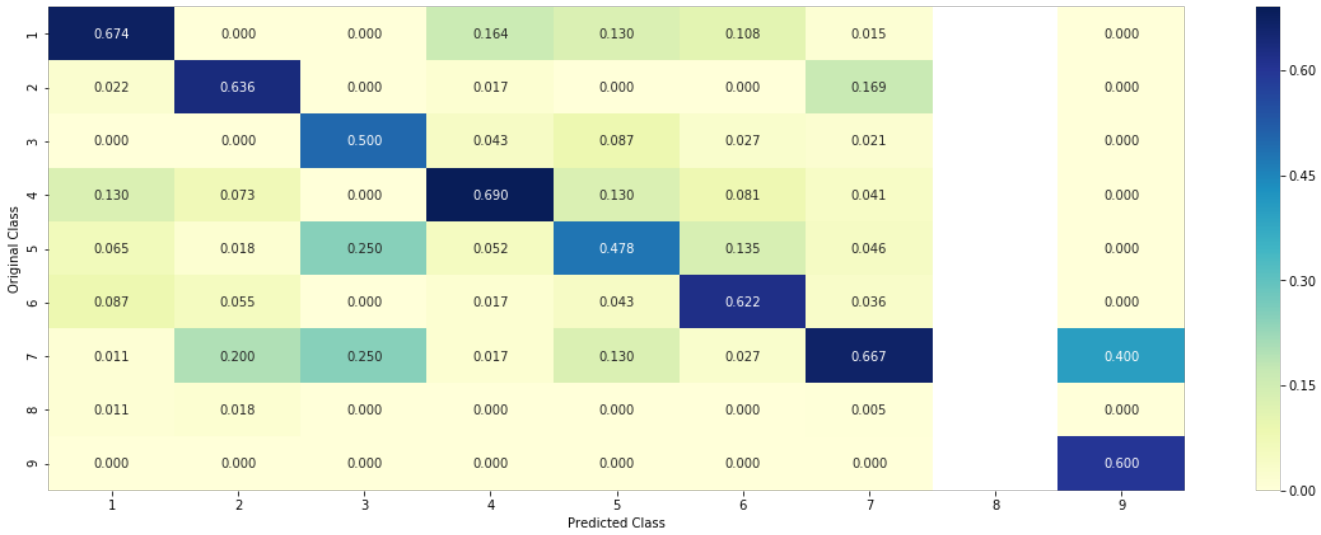
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,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

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

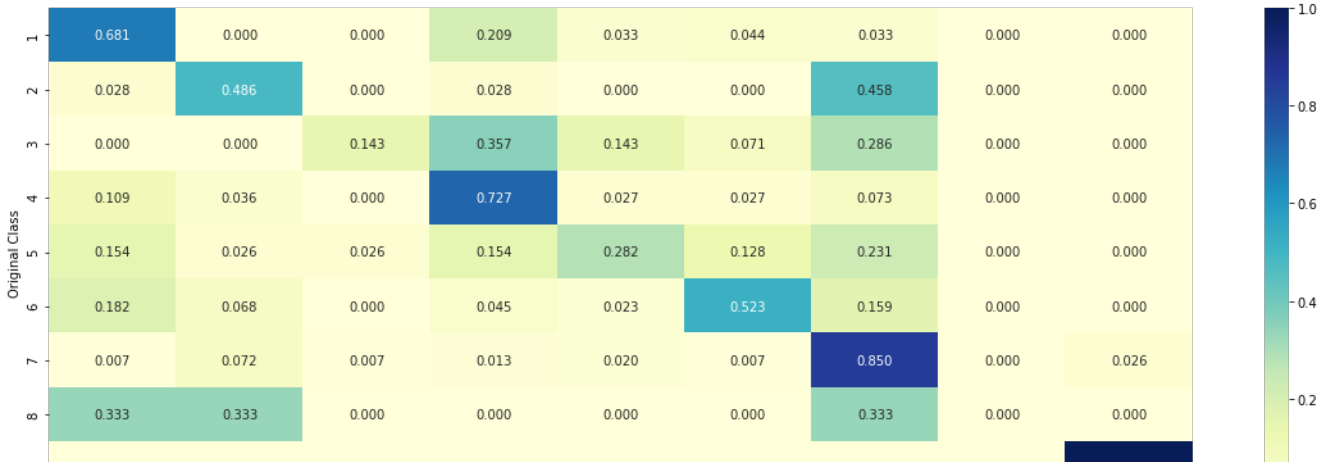
```
Log loss : 0.9968945652075092
Number of mis-classified points : 0.34398496240601506
-------------------- Confusion matrix --------------------
```

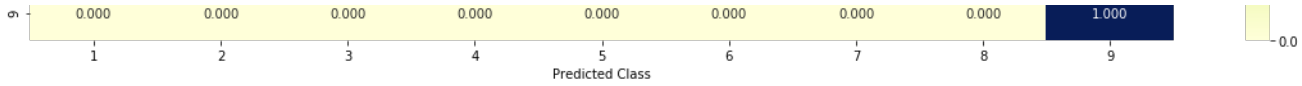



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





4.3.1.3. Feature Importance

In [70]:

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

4.3.1.3.1. Correctly Classified point

In [71]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', 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: [[2.400e-03 4.160e-02 5.000e-04 8.100e-03 5.200e-03 2.130e-02 9.181
e-01
  2.600e-03 2.000e-04]]
Actual Class : 6
--------------------------------------------------
33 Text feature [161] present in test data point [True]
120 Text feature [114] present in test data point [True]
129 Text feature [16] present in test data point [True]
173 Text feature [121] present in test data point [True]
174 Text feature [2012] present in test data point [True]
286 Text feature [100] present in test data point [True]
312 Text feature [152] present in test data point [True]
331 Text feature [124] present in test data point [True]
357 Text feature [128] present in test data point [True]
414 Text feature [177] present in test data point [True]
447 Text feature [1995] present in test data point [True]
453 Text feature [11] present in test data point [True]
472 Text feature [17] present in test data point [True]
Out of the top  500  features  13 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [72]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_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 : 5
Predicted Class Probabilities: [[1.020e-02 1.572e-01 3.000e-04 6.800e-03 6.596e-01 2.900e-03 1.578
e-01
  4.900e-03 2.000e-04]]
Actual Class : 5
--------------------------------------------------
171 Text feature [1b] present in test data point [True]
181 Text feature [24] present in test data point [True]
190 Text feature [12063] present in test data point [True]
199 Text feature [120] present in test data point [True]
338 Text feature [12] present in test data point [True]
371 Text feature [18] present in test data point [True]
396 Text feature [05] present in test data point [True]
397 Text feature [22] present in test data point [True]
415 Text feature [220] present in test data point [True]
450 Text feature [1a] present in test data point [True]
Out of the top  500  features  10 are present in query point
```

4.3.2. Without Class balancing

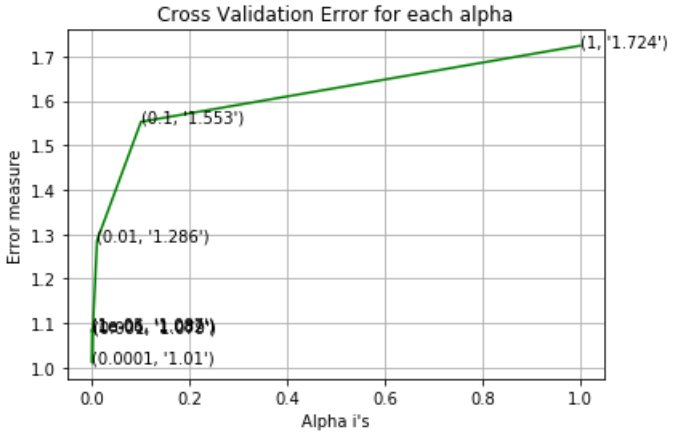
4.3.2.1. Hyper paramter tuning

In [73]:

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

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

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



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

```
# predict_proba(X) Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.0818342398141585
for alpha = 1e-05
Log Loss : 1.0869183241583666
for alpha = 0.0001
Log Loss : 1.010353613600408
for alpha = 0.001
Log Loss : 1.0790590293874829
for alpha = 0.01
Log Loss : 1.2860955548314605
for alpha = 0.1
Log Loss : 1.5529753770725612
for alpha = 1
Log Loss : 1.7241004925536587
```



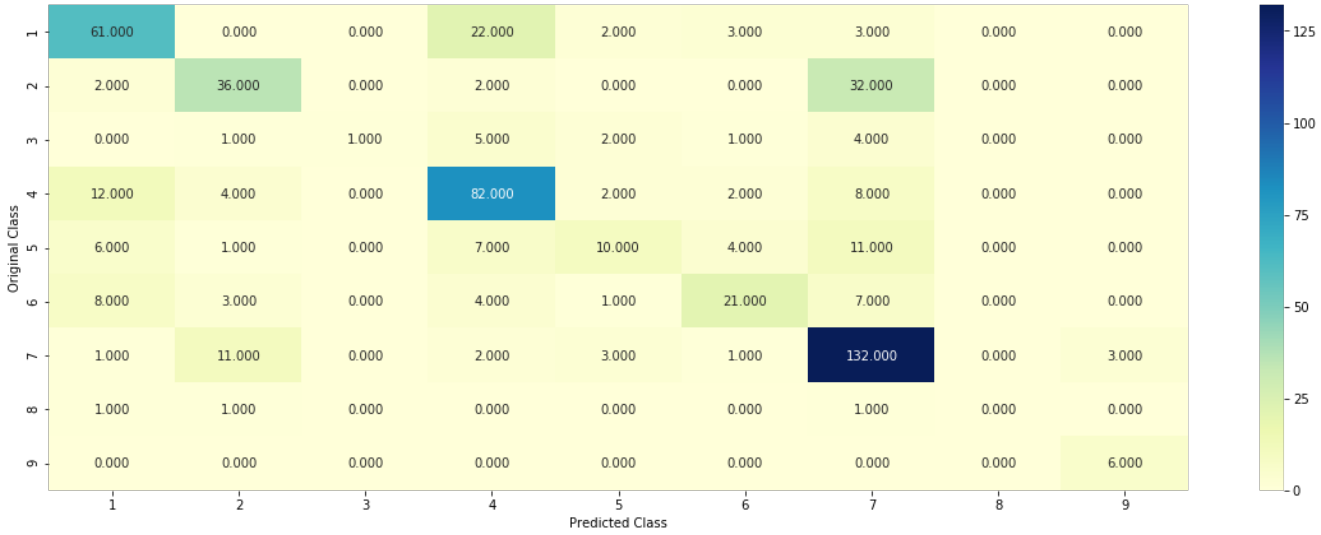
```
For values of best alpha =  0.0001 The train log loss is: 0.4171537263741621
For values of best alpha =  0.0001 The cross validation log loss is: 1.010353613600408
For values of best alpha =  0.0001 The test log loss is: 1.0876296352772683
```

4.3.2.2. Testing model with best hyper parameters

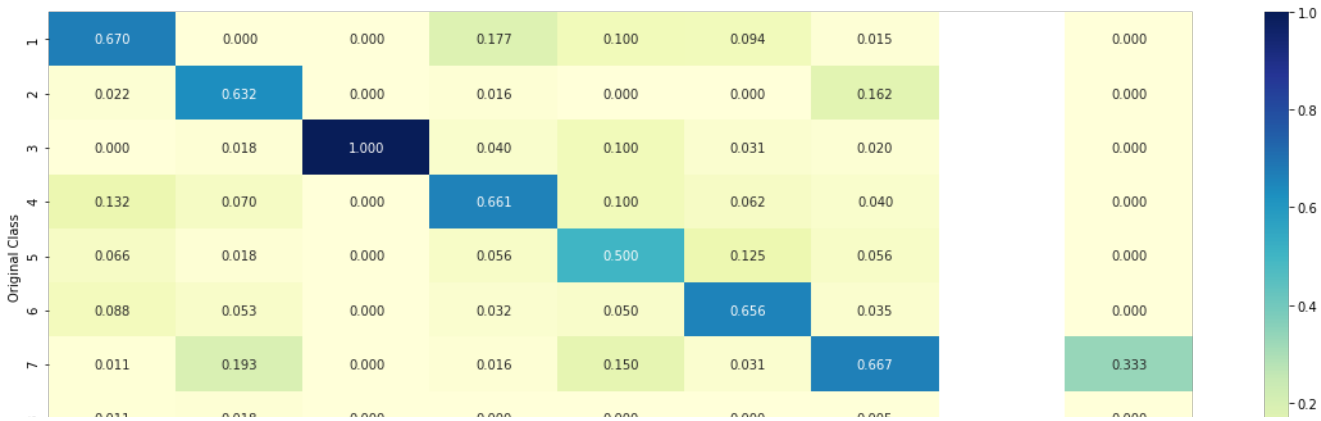
In [74]:

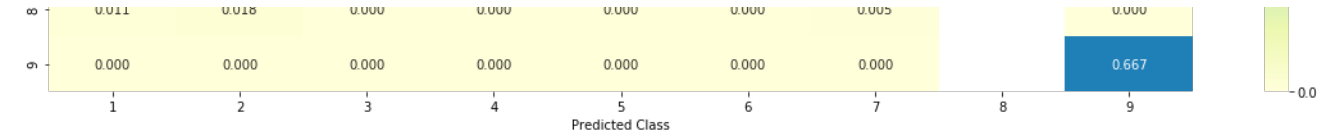
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# 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 : 1.010353613600408
Number of mis-classified points : 0.34398496240601506
-------------------- Confusion matrix --------------------
```

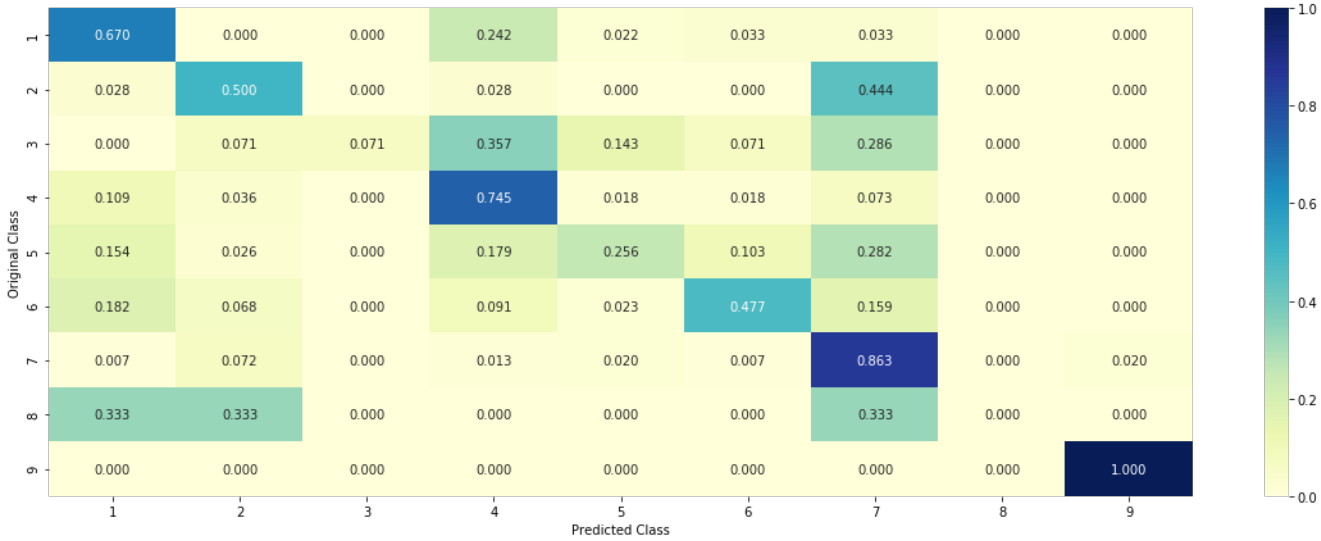



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```






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



4.3.2.3. Feature Importance, Correctly Classified point

In [75]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[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: [[3.000e-03 3.770e-02 4.000e-04 1.010e-02 5.200e-03 2.010e-02 9.192
e-01
  4.300e-03 0.000e+00]]
Actual Class : 6
--------------------------------------------------
47 Text feature [161] present in test data point [True]
161 Text feature [16] present in test data point [True]
162 Text feature [114] present in test data point [True]
187 Text feature [121] present in test data point [True]
287 Text feature [2012] present in test data point [True]
333 Text feature [17] present in test data point [True]
366 Text feature [100] present in test data point [True]
394 Text feature [128] present in test data point [True]
396 Text feature [1995] present in test data point [True]
455 Text feature [152] present in test data point [True]
475 Text feature [124] present in test data point [True]
Out of the top  500  features  11 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [76]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[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 : 5
Predicted Class Probabilities: [[1.030e-02 1.665e-01 1.000e-04 7.000e-03 5.816e-01 2.300e-03 2.232
e-01
  8.900e-03 0.000e+00]]
Actual Class : 5
--------------------------------------------------
174 Text feature [1b] present in test data point [True]
177 Text feature [24] present in test data point [True]
188 Text feature [12063] present in test data point [True]
209 Text feature [120] present in test data point [True]
382 Text feature [22] present in test data point [True]
385 Text feature [18] present in test data point [True]
397 Text feature [12] present in test data point [True]
422 Text feature [05] present in test data point [True]
473 Text feature [1a] present in test data point [True]
482 Text feature [220] present in test data point [True]
Out of the top  500  features  10 are present in query point
```

4.4. Linear Support Vector Machines

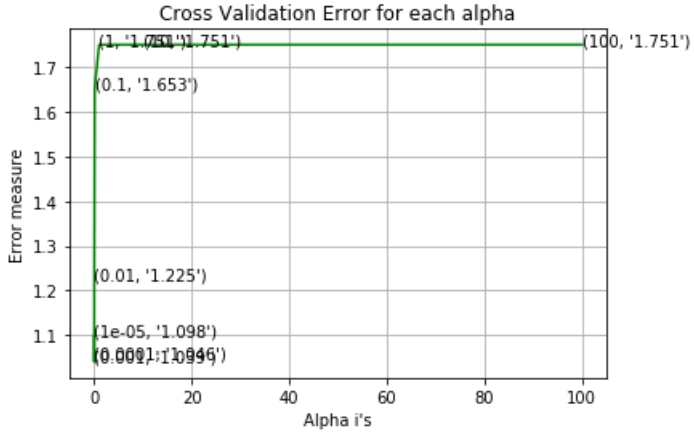
4.4.1. Hyper paramter tuning

In [77]:

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

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

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



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

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

```
alpha = [10** x for x in range(-5, 5)]
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 : 1.0980931410599077
for C = 0.0001
Log Loss : 1.0459757049246243
for C = 0.001
Log Loss : 1.0394823948724878
for C = 0.01
Log Loss : 1.224763213003777
for C = 0.1
Log Loss : 1.6530526795583167
for C = 1
Log Loss : 1.7508496540308447
for C = 10
Log Loss : 1.750849610373136
for C = 100
Log Loss : 1.7508497266195495
```



```
For values of best alpha =  0.001 The train log loss is: 0.5275488338698224
For values of best alpha =  0.001 The cross validation log loss is: 1.0394823948724878
For values of best alpha =  0.001 The test log loss is: 1.1031051230615974
```

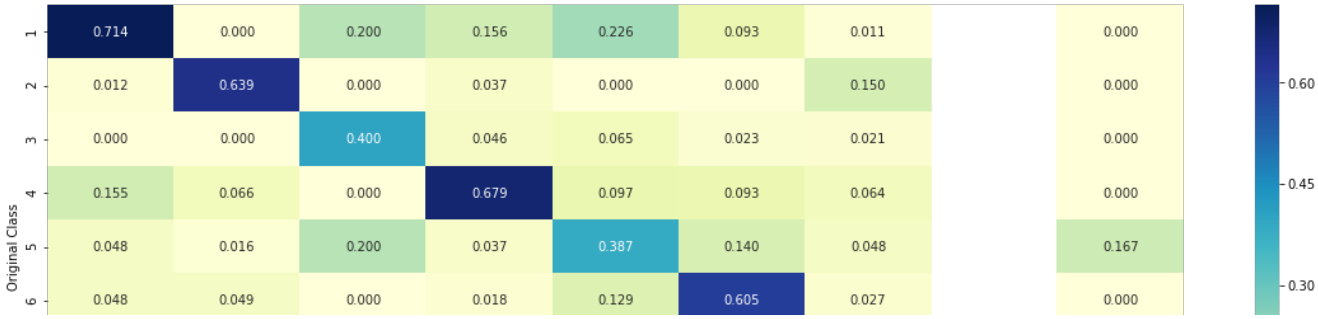
4.4.2. Testing model with best hyper parameters

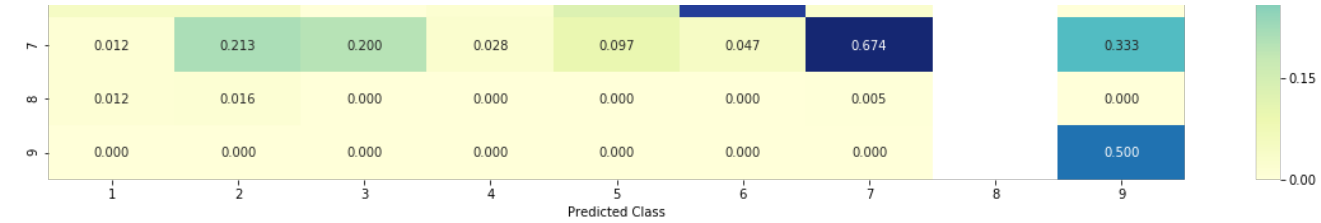
In [78]:

```
# 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)
```

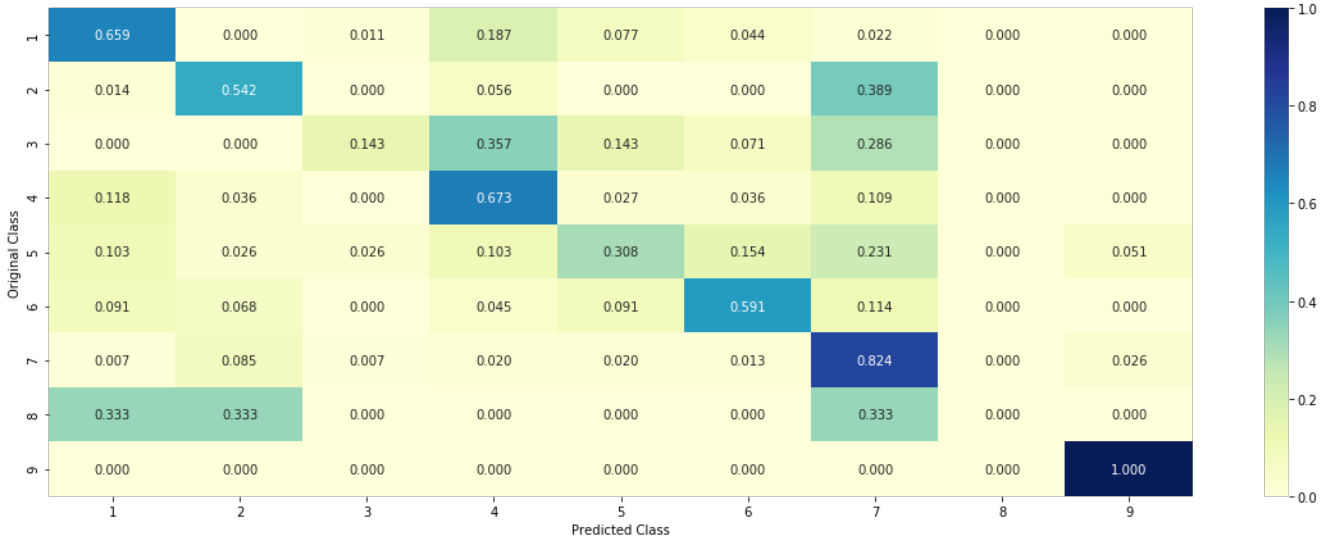
```
Log loss : 1.0394823948724878
Number of mis-classified points : 0.35150375939849626
-------------------- Confusion matrix --------------------
```


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






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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [79]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[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: [[1.370e-02 5.090e-02 1.500e-03 2.810e-02 1.620e-02 3.300e-02 8.538
e-01
  2.200e-03 6.000e-04]]
Actual Class : 6
--------------------------------------------------
38 Text feature [114] present in test data point [True]
42 Text feature [161] present in test data point [True]
63 Text feature [121] present in test data point [True]
65 Text feature [2012] present in test data point [True]
191 Text feature [16] present in test data point [True]
224 Text feature [152] present in test data point [True]
235 Text feature [124] present in test data point [True]
410 Text feature [128] present in test data point [True]
453 Text feature [000] present in test data point [True]
456 Text feature [126] present in test data point [True]
469 Text feature [1995] present in test data point [True]
479 Text feature [177] present in test data point [True]
```

```
479 Text feature [177] present in test data point [True]
480 Text feature [100] present in test data point [True]
488 Text feature [19] present in test data point [True]
Out of the top  500  features  14 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [81]:

```
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 : 5
Predicted Class Probabilities: [[0.031  0.0943 0.0019 0.0184 0.6092 0.0077 0.2254 0.0106 0.0014]]
Actual Class : 5
--------------------------------------------------
230 Text feature [12063] present in test data point [True]
237 Text feature [1b] present in test data point [True]
266 Text feature [120] present in test data point [True]
275 Text feature [24] present in test data point [True]
390 Text feature [18] present in test data point [True]
416 Text feature [220] present in test data point [True]
428 Text feature [036] present in test data point [True]
460 Text feature [12] present in test data point [True]
Out of the top  500  features  8 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [82]:

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

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

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

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


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

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

alpha = [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 : 1.2543748335369236
for n_estimators = 100 and max depth =  10
Log Loss : 1.2338632469523318
for n_estimators = 200 and max depth =  5
Log Loss : 1.2486577255617837
for n_estimators = 200 and max depth =  10
Log Loss : 1.217627808296658
for n_estimators = 500 and max depth =  5
Log Loss : 1.2352094742607689
for n_estimators = 500 and max depth =  10
Log Loss : 1.2115141599659864
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2329227223518533
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2072110644452425
for n_estimators = 2000 and max depth =  5
Log Loss : 1.232853003756967
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2041761853545683
```

```
For values of best estimator =  2000 The train log loss is: 0.5527081197855304
For values of best estimator =  2000 The cross validation log loss is: 1.2041761853545683
For values of best estimator =  2000 The test log loss is: 1.1744452460497228
```

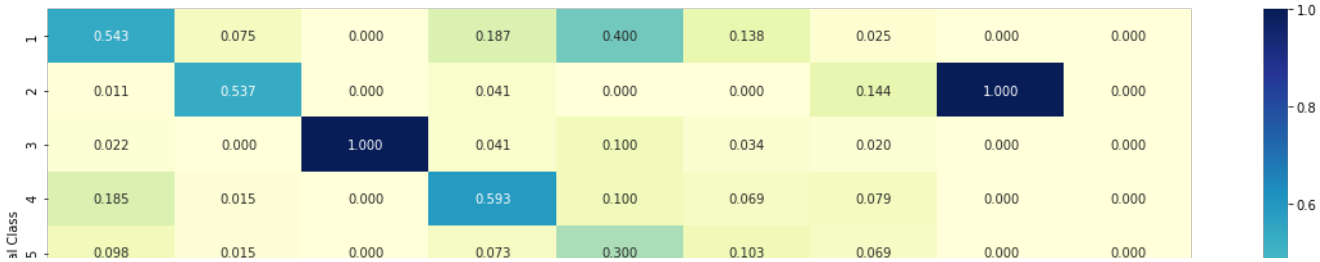
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

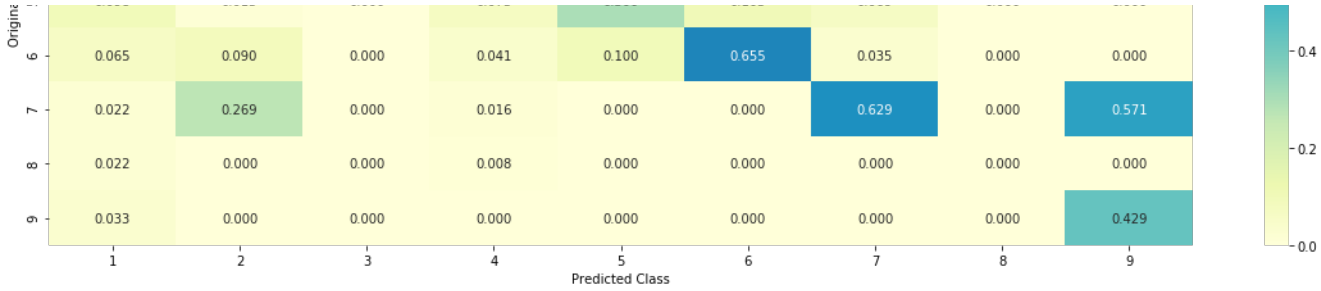
In [83]:

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

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

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

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

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

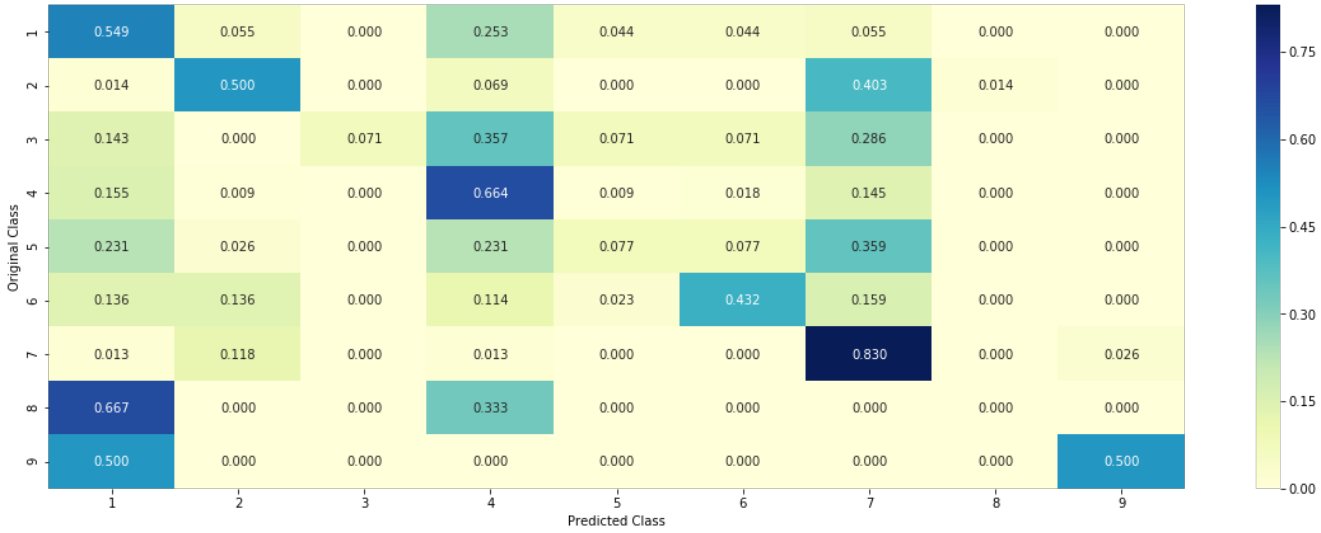
```
Log loss : 1.2041761853545683
Number of mis-classified points : 0.41353383458646614
-------------------- Confusion matrix --------------------
```


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





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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [84]:

```
# 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: [[0.0633 0.2177 0.0159 0.0534 0.0453 0.0519 0.5421 0.006  0.0044]]
Actual Class : 6
--------------------------------------------------
16 Text feature [2012] present in test data point [True]
22 Text feature [124] present in test data point [True]
56 Text feature [120] present in test data point [True]
79 Text feature [146] present in test data point [True]
Out of the top  100  features  4 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [85]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.129  0.2007 0.02   0.0819 0.2071 0.0539 0.2391 0.0612 0.0072]]
Actuall Class : 5
--------------------------------------------------
56 Text feature [120] present in test data point [True]
91 Text feature [23] present in test data point [True]
Out of the top  100  features  2 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [86]:

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

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

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

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


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

alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
```

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

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

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

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.2640705089121145
for n_estimators = 10 and max depth =  3
Log Loss : 1.8273269931631368
for n_estimators = 10 and max depth =  5
Log Loss : 1.9085878538636614
for n_estimators = 10 and max depth =  10
Log Loss : 2.1236820452476204
for n_estimators = 50 and max depth =  2
Log Loss : 1.9166484773764154
for n_estimators = 50 and max depth =  3
Log Loss : 1.6082270970525732
for n_estimators = 50 and max depth =  5
Log Loss : 1.623371291957262
for n_estimators = 50 and max depth =  10
Log Loss : 1.7510169709171128
for n_estimators = 100 and max depth =  2
Log Loss : 1.8117638154132403
for n_estimators = 100 and max depth =  3
Log Loss : 1.6426428043909738
for n_estimators = 100 and max depth =  5
Log Loss : 1.4355917851698359
for n_estimators = 100 and max depth =  10
Log Loss : 1.6630732153851104
for n_estimators = 200 and max depth =  2
Log Loss : 1.851431323237167
for n_estimators = 200 and max depth =  3
Log Loss : 1.678703678505206
for n_estimators = 200 and max depth =  5
Log Loss : 1.4730049864401593
for n_estimators = 200 and max depth =  10
Log Loss : 1.7426761946392184
for n_estimators = 500 and max depth =  2
Log Loss : 1.861306204156928
for n_estimators = 500 and max depth =  3
Log Loss : 1.7356599231922136
for n_estimators = 500 and max depth =  5
Log Loss : 1.5129184953164188
```

```
Log Loss : 1.5129104995104166
for n_estimators = 500 and max depth =  10
Log Loss : 1.7759439414156957
for n_estimators = 1000 and max depth =  2
Log Loss : 1.8528111804084293
for n_estimators = 1000 and max depth =  3
Log Loss : 1.7372400681044602
for n_estimators = 1000 and max depth =  5
Log Loss : 1.4870892348865992
for n_estimators = 1000 and max depth =  10
Log Loss : 1.7632280398428968
For values of best alpha =  100 The train log loss is: 0.05524550806960524
For values of best alpha =  100 The cross validation log loss is: 1.4355917851698359
For values of best alpha =  100 The test log loss is: 1.3974934385472335
```

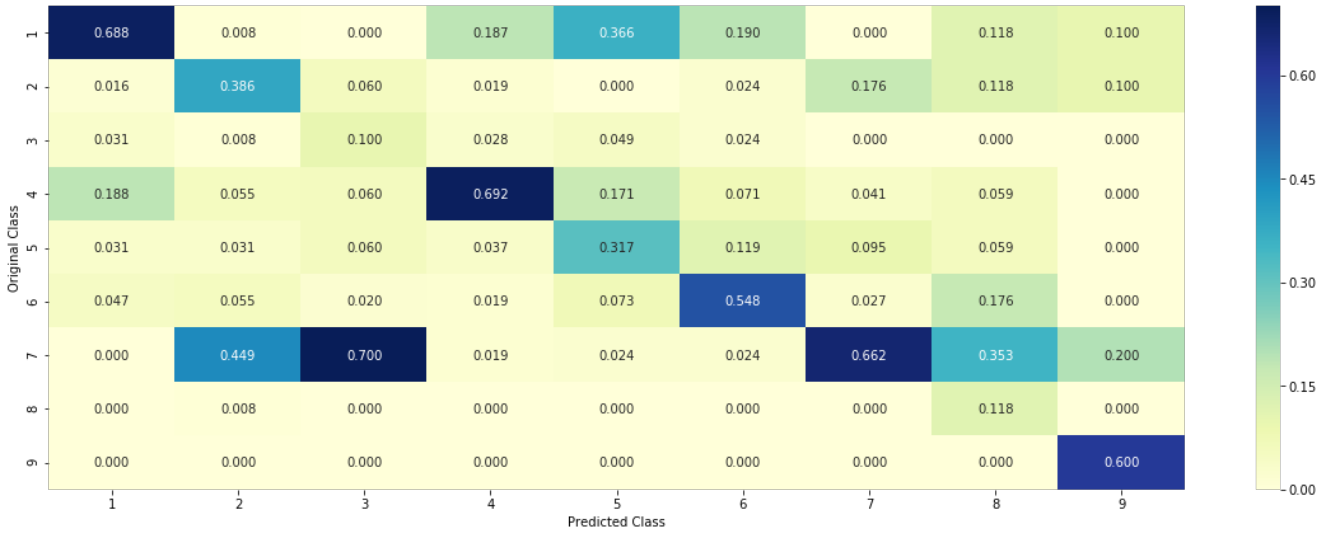
4.5.4. Testing model with best hyper parameters (Response Coding)

In [87]:

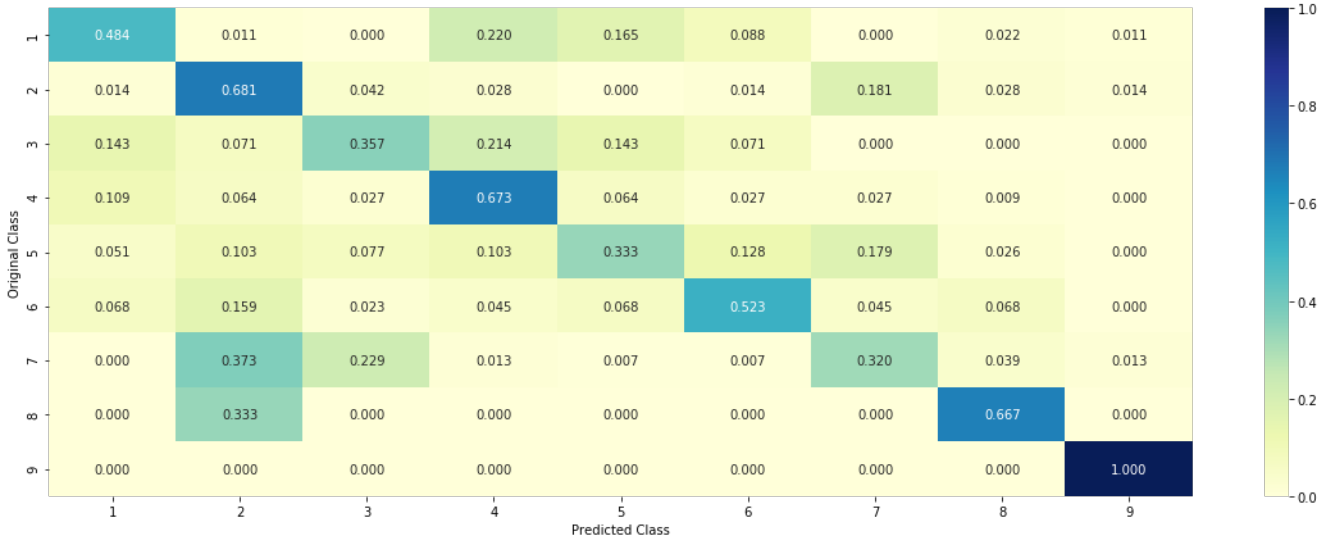
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None,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

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

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

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

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

```
Log loss : 1.4355917851698359
Number of mis-classified points : 0.5018796992481203
-------------------- 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 [88]:

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


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

```
    else:
        print("Text is important feature")
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0208 0.3785 0.122  0.0204 0.0327 0.0462 0.3092 0.0544 0.0157]]
Actual Class : 6
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [89]:

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

```
Predicted Class : 9
Predicted Class Probabilities: [[0.0436 0.0357 0.0748 0.0701 0.193  0.075  0.0114 0.1722 0.3242]]
Actual Class : 5
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
```

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

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [91]:

```
from mlxtend.classifier import StackingClassifier

# 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-forest-and-their-construction-2/
# --------------------------------


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

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


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

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehotCoding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding))))
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sclf.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: 1.01
Support vector machines : Log Loss: 1.75
Naive Bayes : Log Loss: 1.30
-------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.177
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.031
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.507
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.203
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.427
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.832
```

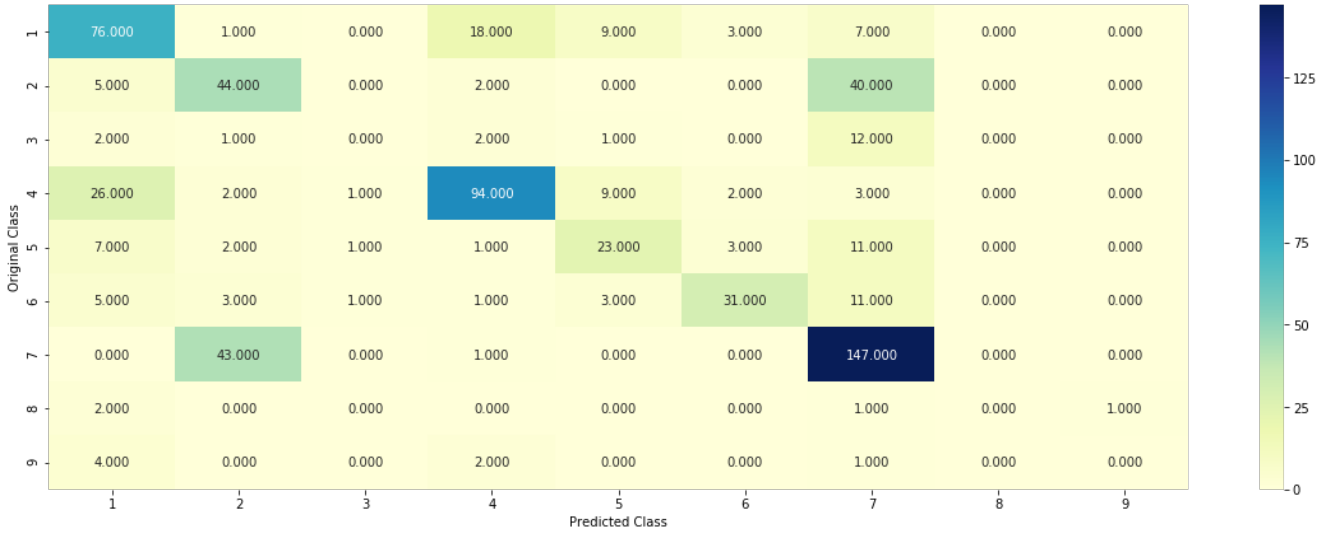
4.7.2 testing the model with the best hyper parameters

In [92]:

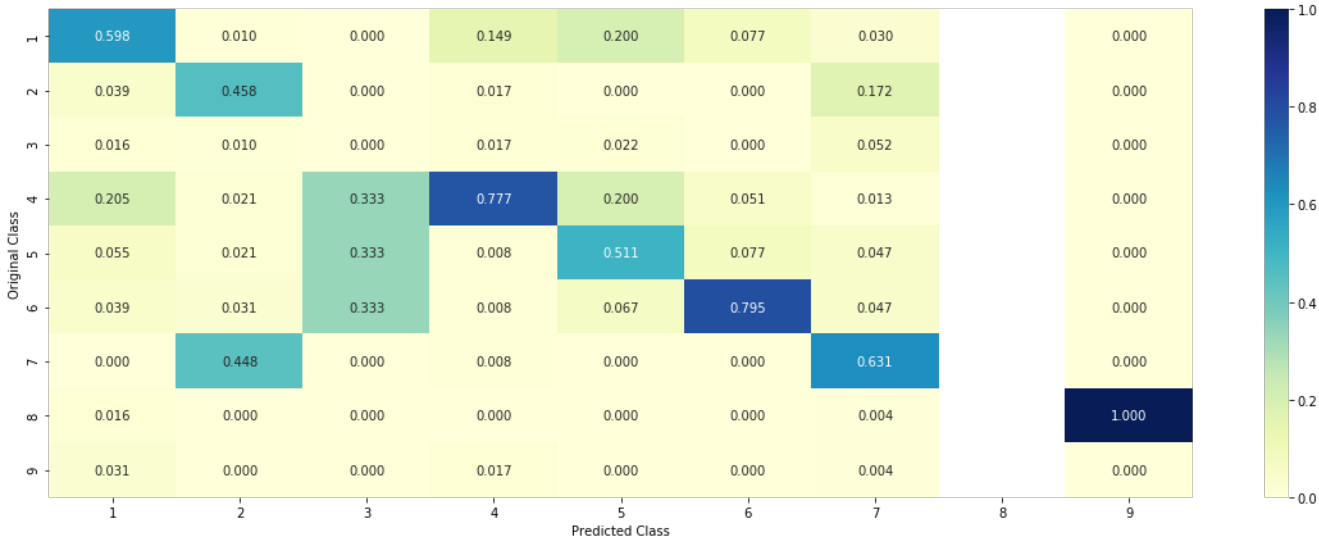
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

log_error = log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
```

```
print("Log loss (CV) on the stacking classifier :",log_error)

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

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

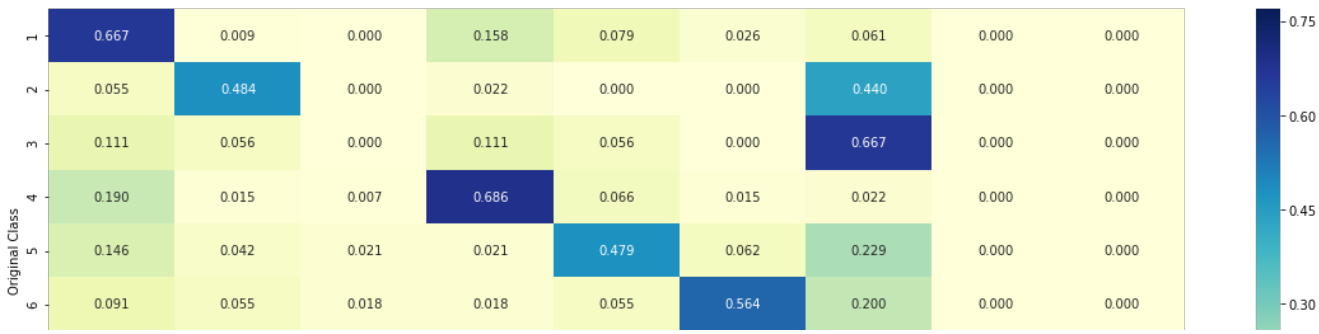
```
Log loss (train) on the stacking classifier : 0.5942577828272247
Log loss (CV) on the stacking classifier : 1.203152690196588
Log loss (test) on the stacking classifier : 1.1717977981796974
Number of missclassified point : 0.37593984962406013
-------------------- Confusion matrix --------------------
```

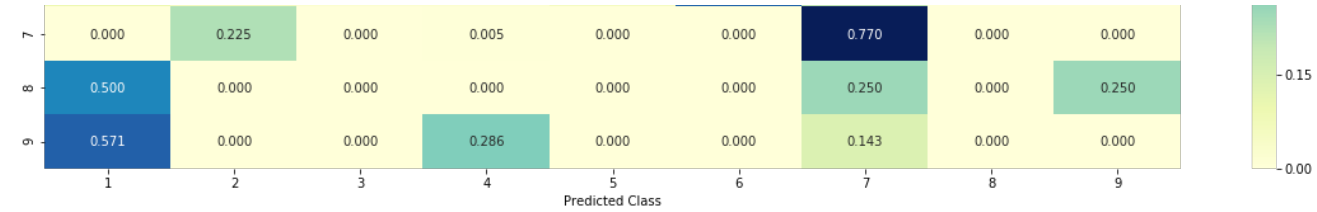



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





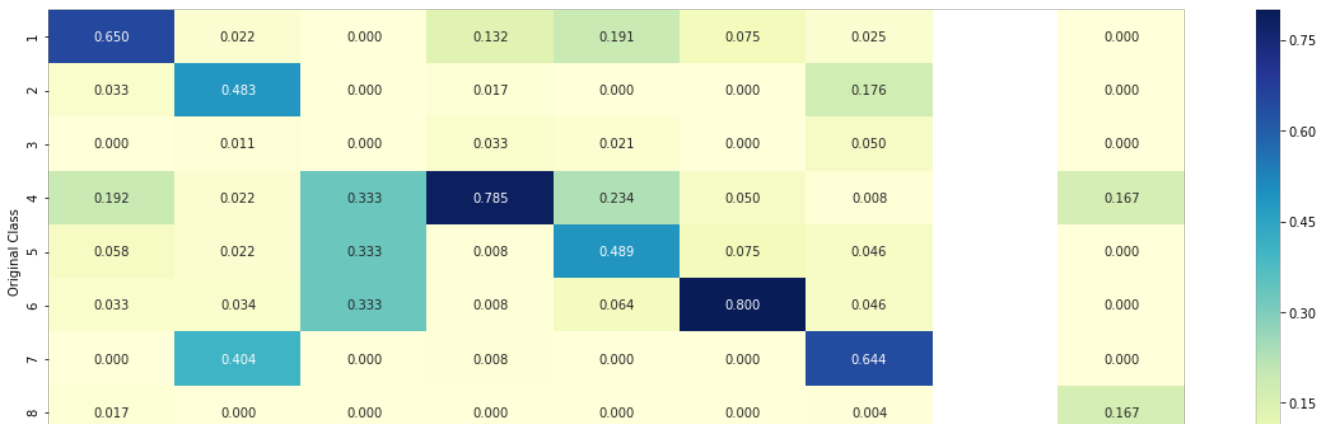
4.7.3 Maximum Voting classifier

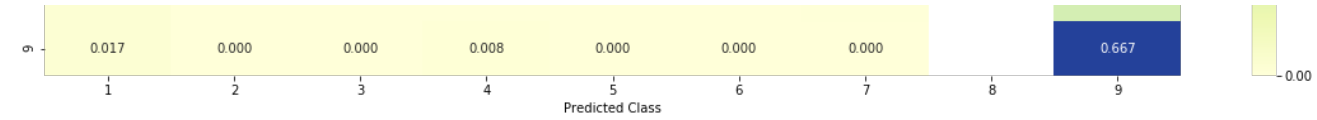
In [93]:

```
#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))
```

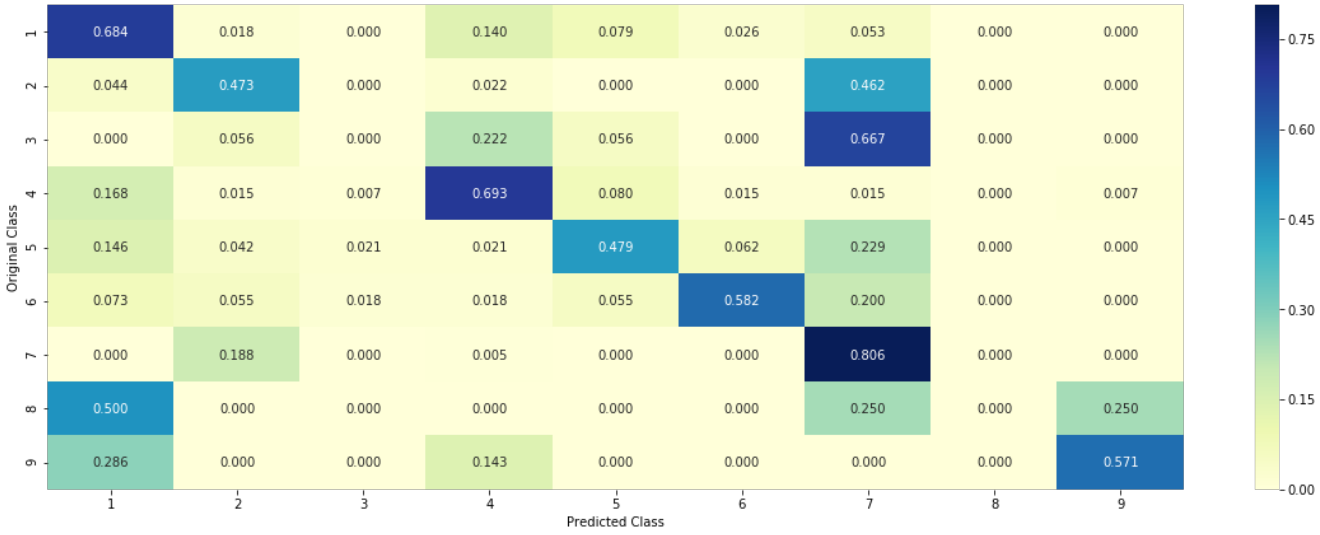
```
Log loss (train) on the VotingClassifier : 0.8443919004795133
Log loss (CV) on the VotingClassifier : 1.186979411702499
Log loss (test) on the VotingClassifier : 1.1925158267905624
Number of missclassified point : 0.3548872180451128
-------------------- Confusion matrix --------------------
```


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





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



Summary & Conclusion

In [94]:

```
from IPython.display import Image
img = 'C:/Users/Ghost/Downloads/AAIC Assignment/Tfidf PC.png'
Image(img, width=50000, height=50000)
```

Out[94]:

Tf-Idf Vectorizer With "2500" Most Important Words

Classifier	Train Loss	CV Loss	Test Loss	% Miss Class
Naïve Bayes	0.71	1.29	1.286	40.9 %
KNN	0.49	1.04	1.104	34.2 %
Logistic Reg. (Balanced)	0.413	0.996	1.05	34.3 %
Logistic Reg. (W/O Balanced)	0.417	1.010	1.08	34.39 %
Linear SVM	0.527	1.039	1.103	35.15 %
Random Forest (One Hot Encoding)	0.552	1.204	1.174	41.3 %
Random Forest (Response Coding)	0.05	1.43	1.39	50.1 %
Stacking	0.59	1.203	1.17	37.5 %
Max Voting	0.844	1.186	1.192	35.4 %

1] Used Tfidf Vectorizer with 2500 Most imp. Features of Text

2] Perfromance of KNN, Log Regression have the least % of Miss CLassified Class where For BOW only Logistic Regression with Class balance had Best Perfromance.

3] Logistic Regression with Class Balanced have the least Loss on Cross Validation and Test Data that is 0.996 on CV data and 1.05 for Test Data.

4] Random Forest with Response coding Features have the overall bad performance with more than 50% miss classified data points

and the Loss difference between Train and CV/Test loss is significant which leading to overfitting of the model.

5] Three Classifier have least % of Miss classified data with Tfidf with 2500 Important features.