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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
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
```

```
D:\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:144: FutureWarnin
g: The sklearn.metrics.classification module is  deprecated in version 0.22 a
nd will be removed in version 0.24. The corresponding classes / functions sho
uld instead be imported from sklearn.metrics. Anything that cannot be importe
d from sklearn.metrics is now part of the private API.
  warnings.warn(message, FutureWarning)
D:\Anaconda3\lib\site-packages\sklearn\externals\six.py:31: FutureWarning: Th
e module is deprecated in version 0.21 and will be removed in version 0.23 si
nce we've dropped support for Python 2.7. Please rely on the official version
of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", FutureWarning)
D:\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:144: FutureWarnin
g: The sklearn.neighbors.base module is  deprecated in version 0.22 and will
be removed in version 0.24. The corresponding classes / functions should inst
ead be imported from sklearn.neighbors. Anything that cannot be imported from
sklearn.neighbors is now part of the private API.
  warnings.warn(message, FutureWarning)
D:\Anaconda3\lib\site-packages\sklearn\externals\joblib\__init__.py:15: Futur
eWarning: sklearn.externals.joblib is deprecated in 0.21 and will be removed
in 0.23. Please import this functionality directly from joblib, which can be
installed with: pip install joblib. If this warning is raised when loading pi
ckled models, you may need to re-serialize those models with scikit-learn 0.2
1+.
  warnings.warn(msg, category=FutureWarning)
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:
```
data = pd.read_csv('training/training_variants')
print('Number of data points : ', data.shape[0])
print('Number of features : ', data.shape[1])
print('Features : ', data.columns.values)
data.head()
```

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

Out[2]:

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

```
In [3]: data.shape
```

```
Out[3]: (3321, 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 [4]: # note the seprator in this file
data_text =pd.read_csv("training/training_text",sep="\|\|",engine="python",nam
es=["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[4]:

	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 [5]:
```
# 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 [6]:
```
#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, "sec
onds")
```

```
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 : 220.5662871749492 seconds
```

In [7]:
```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[7]:

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

In [8]:
```
result[result.isnull().any(axis=1)]
```

Out[8]:

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

In [9]:
```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Varia
tion']
```

In [10]:
```
result[result['ID']==1109]
```

Out[10]:

	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 [11]:
```
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 outpu
t 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 dis
tribution 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

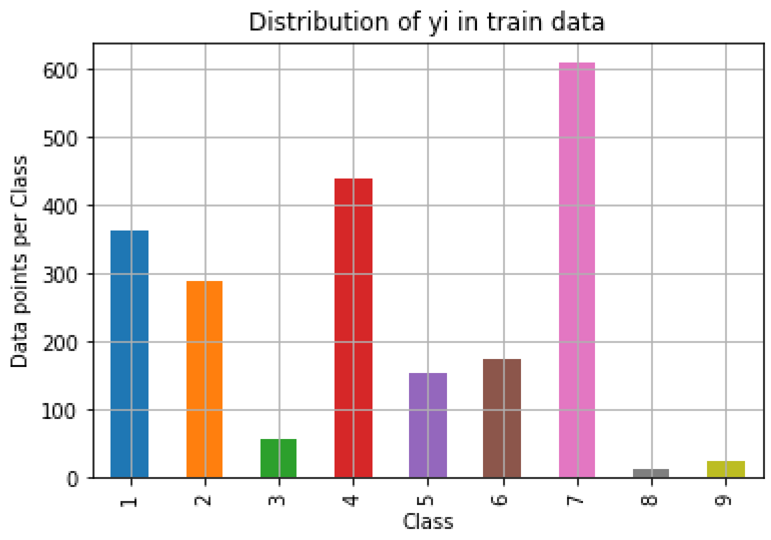
In [12]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
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
```

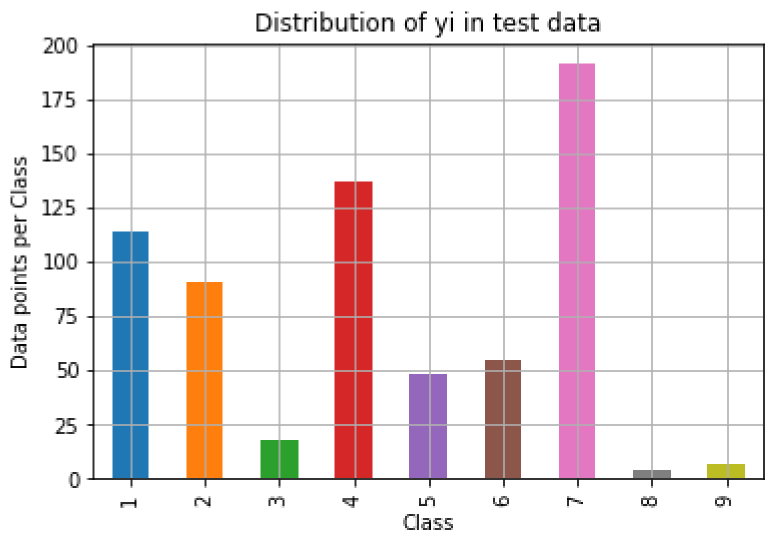
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [13]:
```
# it returns a dict, keys as class labels and values as the number of data poi
nts 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.args
ort.html
# -(train_class_distribution.values): the minus sign will give us in decreasin
g order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distribution.
values[i], '(', np.round((train_class_distribution.values[i]/train_df.shape[0]
*100), 3), '%)')


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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.args
ort.html
# -(train_class_distribution.values): the minus sign will give us in decreasin
g 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.v
alues[i], '(', np.round((test_class_distribution.values[i]/test_df.shape[0]*10
0), 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.args
ort.html
# -(train_class_distribution.values): the minus sign will give us in decreasin
```

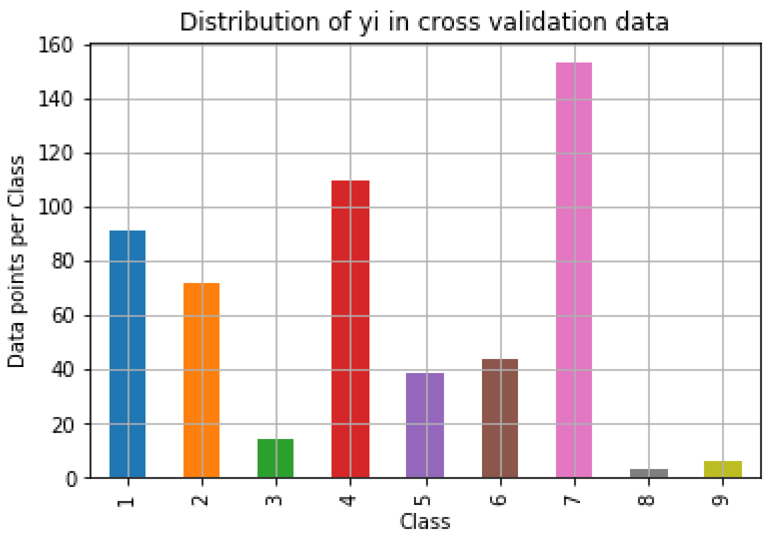
```
g order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.val
ues[i], '(', np.round((cv_class_distribution.values[i]/cv_df.shape[0]*100), 3
), '%)')
```



```
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 %)
```

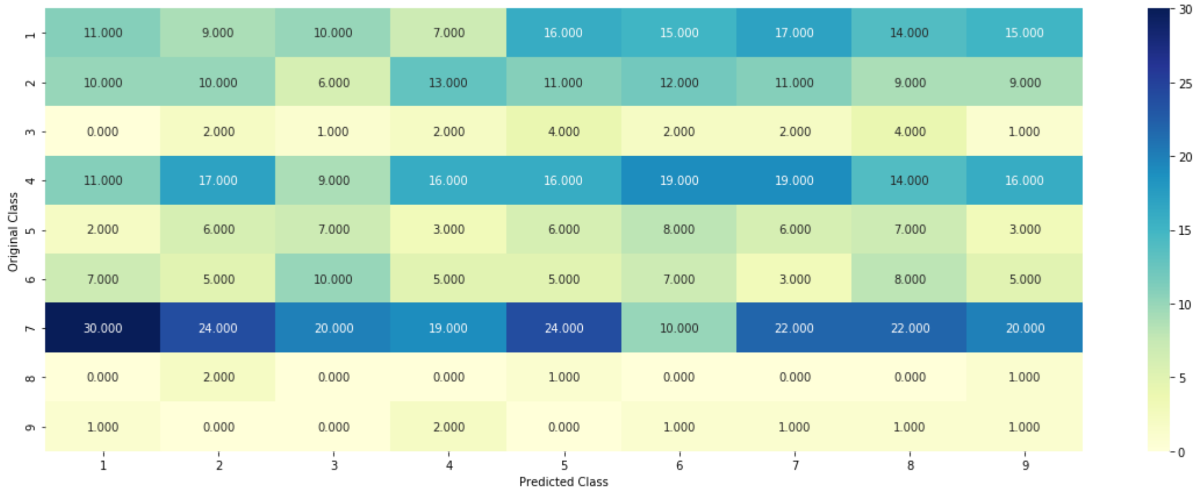
3.2 Prediction using a 'Random' Model

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

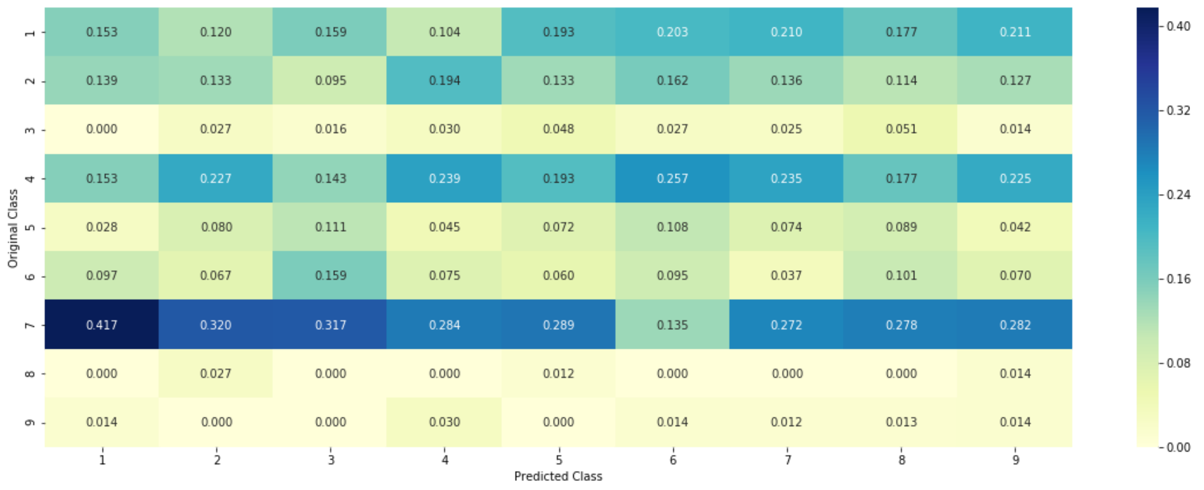
In [14]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i a
re predicted class j

    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in th
at 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 th
at 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, y
ticklabels=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, y
ticklabels=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, y
ticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

In [15]:
```
# 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 thei
r 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_predicte
d_y, eps=1e-15))

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

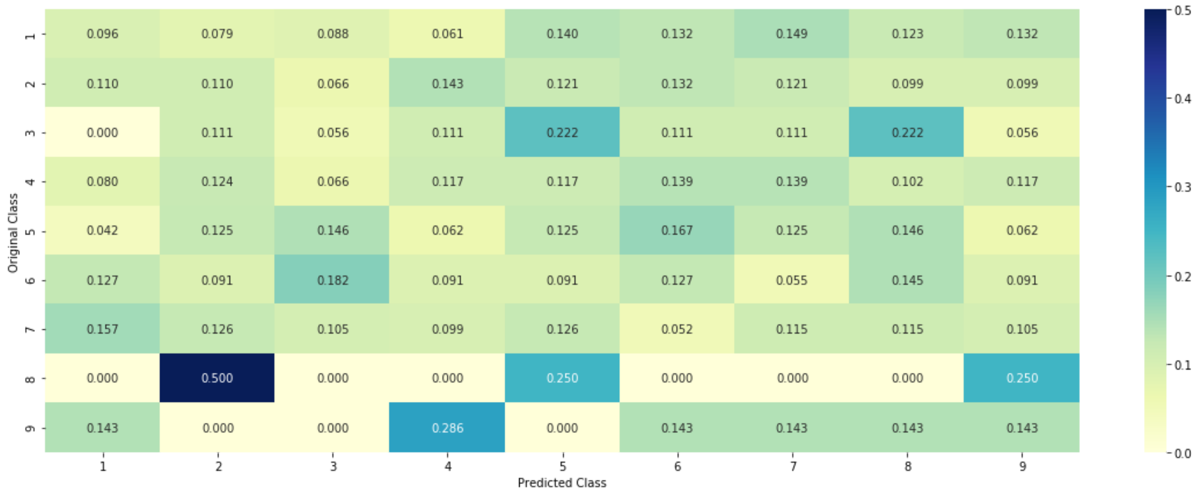
```
Log loss on Cross Validation Data using Random Model 2.5062271926893938
Log loss on Test Data using Random Model 2.4869298665993456
-------------------- Confusion matrix --------------------
```



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



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



3.3 Univariate Analysis

In [16]:
```
# 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 cl
ass1 + 10*alpha / number of time it occurred in total data+90*alpha)
# gv_dict is like a look up table, for every gene it store a (1*9) representat
ion of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

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

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

    # denominator will contain the number of time that particular feature occu
```

```
red 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 occured 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, 0.0681818181818
18177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.
03787878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.0612244897959
18366, 0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.0510
20408163265307, 0.051020408163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.06818
1818181818177, 0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.05
6818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606
060608, 0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.06060
6060606060608, 0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937
106917, 0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069
182389937106917, 0.062893081761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.07284768211920
5295, 0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.2715
2317880794702, 0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.0733333333333
33334, 0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.299
99999999999999, 0.066666666666666666, 0.066666666666666666],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

    # gv_fea: Gene_variation feature, it will contain the feature for each fea
ture value in the data
    gv_fea = []
    # for every feature values in the given data frame we will check if it is
 there in the train data 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?

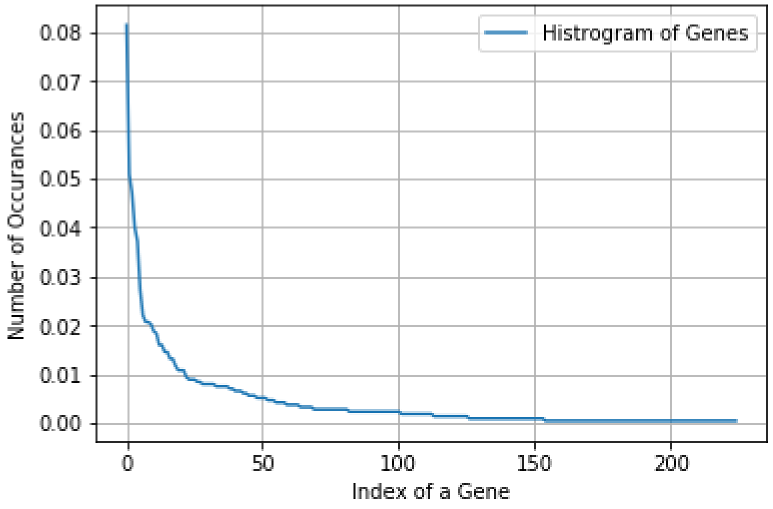
In [17]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 225
BRCA1     173
TP53      108
EGFR      100
PTEN       85
BRCA2      79
KIT        58
BRAF       47
PDGFRA     44
ERBB2      44
ALK        43
Name: Gene, dtype: int64
```

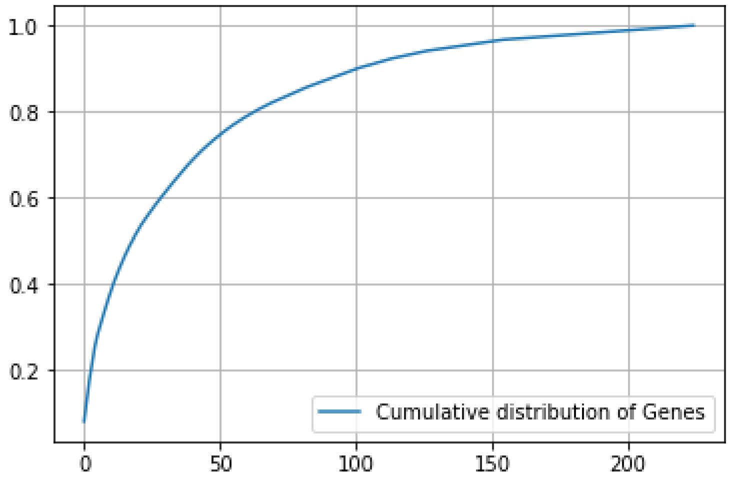
In [18]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes
 in the train data, and they are distibuted as follows",)
```

```
Ans: There are 225 different categories of genes in the train data, and they
are distibuted as follows
```

In [19]:
```
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 [20]:
```
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 [21]:
```
#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", tra
in_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 [22]:
```
print("train_gene_feature_responseCoding is converted feature using respone co
ding method. The shape of gene feature:", train_gene_feature_responseCoding.sh
ape)
print("test_gene_feature_responseCoding is converted feature using respone cod
ing method. The shape of test feature:", test_gene_feature_responseCoding.shap
e)
print("cv_gene_feature_responseCoding is converted feature using respone codin
g method. The shape of cv feature:", cv_gene_feature_responseCoding.shape)
```

```
train_gene_feature_responseCoding is converted feature using respone coding m
ethod. The shape of gene feature: (2124, 9)
test_gene_feature_responseCoding is converted feature using respone coding me
thod. The shape of test feature: (665, 9)
cv_gene_feature_responseCoding is converted feature using respone coding meth
od. The shape of cv feature: (532, 9)
```

In [23]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gen
e'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [24]:
```
train_df['Gene'].head()
```

Out[24]:
```
2998      KIT
2848    BRCA2
1799       AR
383      TP53
2722     BRAF
Name: Gene, dtype: object
```

In [25]:
```
gene_vectorizer.get_feature_names()
```

```
Out[25]: ['abl1',
          'acvr1',
          'ago2',
          'akt1',
          'akt2',
          'akt3',
          'alk',
          'apc',
          'ar',
          'araf',
          'arid1a',
          'arid1b',
          'arid2',
          'arid5b',
          'asxl1',
          'asxl2',
          'atm',
          'atr',
          'atrx',
          'aurka',
          'aurkb',
          'axin1',
          'b2m',
          'bap1',
          'bard1',
          'bcl10',
          'bcl2l11',
          'bcor',
          'braf',
          'brca1',
          'brca2',
          'brd4',
          'brip1',
          'btk',
          'card11',
          'carm1',
          'casp8',
          'cbl',
          'ccnd1',
          'ccnd2',
          'ccnd3',
          'cdh1',
          'cdk12',
          'cdk4',
          'cdk6',
          'cdk8',
          'cdkn1a',
          'cdkn1b',
          'cdkn2a',
          'cdkn2b',
          'cdkn2c',
          'chek2',
          'cic',
          'crebbp',
          'ctcf',
          'ctnnb1',
          'ddr2',
          'dicer1',
          'dnmt3a',
          'dnmt3b',
          'egfr',
          'eif1ax',
          'elf3',
          'ep300',
          'epas1',
          'epcam',
          'erbb2',
          'erbb3',
          'erbb4',
          'ercc2',
          'ercc4',
          'erg',
          'esr1',
          'etv1',
          'etv6',
          'ewsr1',
          'ezh2',
          'fanca',
          'fancc',
          'fat1',
          'fbxw7',
          'fgf19',
          'fgf3',
          'fgfr1',
          'fgfr2',
          'fgfr3',
          'fgfr4',
          'flt3',
          'foxa1',
          'foxp1',
          'gata3',
          'gli1',
          'gna11',
          'gnas',
          'h3f3a',
          'hist1h1c',
          'hla',
          'hnf1a',
          'hras',
          'idh1',
          'idh2',
          'igf1r',
          'il7r',
          'inpp4b',
          'jak1',
          'jak2',
          'kdm5a',
          'kdm5c',
          'kdm6a',
          'kdr',
          'keap1',
          'kit',
          'kmt2a',
          'kmt2b',
          'kmt2c',
          'kmt2d',
          'knstrn',
          'kras',
          'lats2',
          'map2k1',
          'map2k2',
          'map2k4',
          'map3k1',
          'mapk1',
          'med12',
          'mef2b',
          'met',
          'mga',
          'mlh1',
          'mpl',
          'msh2',
          'msh6',
          'mtor',
          'myc',
          'mycn',
          'myd88',
          'ncor1',
          'nf1',
          'nf2',
          'nfe2l2',
          'nfkbia',
          'nkx2',
          'notch1',
          'notch2',
          'npm1',
          'nras',
          'nsd1',
          'ntrk1',
          'ntrk2',
          'nup93',
          'pak1',
          'pax8',
          'pdgfra',
          'pdgfrb',
          'pik3ca',
          'pik3cb',
          'pik3cd',
          'pik3r1',
          'pik3r2',
          'pim1',
          'pms1',
          'pms2',
          'pole',
          'ppm1d',
          'ppp2r1a',
          'ppp6c',
          'prdm1',
          'ptch1',
          'pten',
          'ptpn11',
          'ptprd',
          'ptprt',
          'rab35',
          'rac1',
          'rad21',
          'rad50',
          'rad51b',
          'rad51c',
          'rad51d',
          'rad54l',
          'raf1',
          'rasa1',
          'rb1',
          'rbm10',
          'ret',
          'rheb',
          'rhoa',
          'rit1',
          'ros1',
          'runx1',
          'rxra',
          'sdhc',
          'sf3b1',
          'shq1',
          'smad2',
          'smad3',
          'smad4',
          'smarca4',
          'smarcb1',
          'smo',
          'sos1',
          'sox9',
          'spop',
          'src',
          'srsf2',
          'stag2',
          'stat3',
          'stk11',
          'tert',
          'tet2',
          'tgfbr1',
          'tgfbr2',
          'tmprss2',
          'tp53',
          'tp53bp1',
          'tsc1',
          'tsc2',
          'u2af1',
          'vhl',
          'whsc1',
          'whsc1l1',
          'xpo1',
          'xrcc2',
          'yap1']
```

In [26]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot enco
ding method. The shape of gene feature:", train_gene_feature_onehotCoding.shap
e)
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding m
ethod. The shape of gene feature: (2124, 224)
```

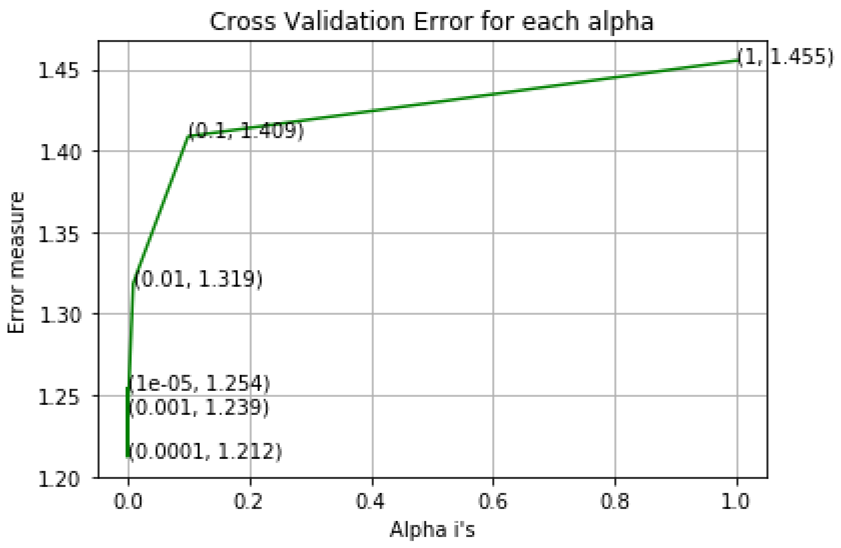
Q4. How good is this gene feature in predicting y_i?

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 [27]:
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

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

# some of methods
# fit(X, y[, coef_init, intercept_init, …])     Fit linear model with Stochast
ic 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_, e
ps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predi
ct_y, labels=clf.classes_, eps=1e-15))

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


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

predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss i
s:",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_loss(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 i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.2540111267947311
For values of alpha =  0.0001 The log loss is: 1.2120950763061806
For values of alpha =  0.001 The log loss is: 1.239000589887041
For values of alpha =  0.01 The log loss is: 1.3186440894440317
For values of alpha =  0.1 The log loss is: 1.408721964712307
For values of alpha =  1 The log loss is: 1.4553525134680059
```



```
For values of best alpha =  0.0001 The train log loss is: 0.9858999925424323
For values of best alpha =  0.0001 The cross validation log loss is: 1.212095
0763061806
For values of best alpha =  0.0001 The test log loss is: 1.1945034241197496
```

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 [28]:
```
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], ":",(te
st_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":"
,(cv_coverage/cv_df.shape[0])*100)
```

```
Q6. How many data points in Test and CV datasets are covered by the  225  gen
es in train dataset?
Ans
1. In test data 638 out of 665 : 95.93984962406014
2. In cross validation data 507 out of  532 : 95.30075187969925
```

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?

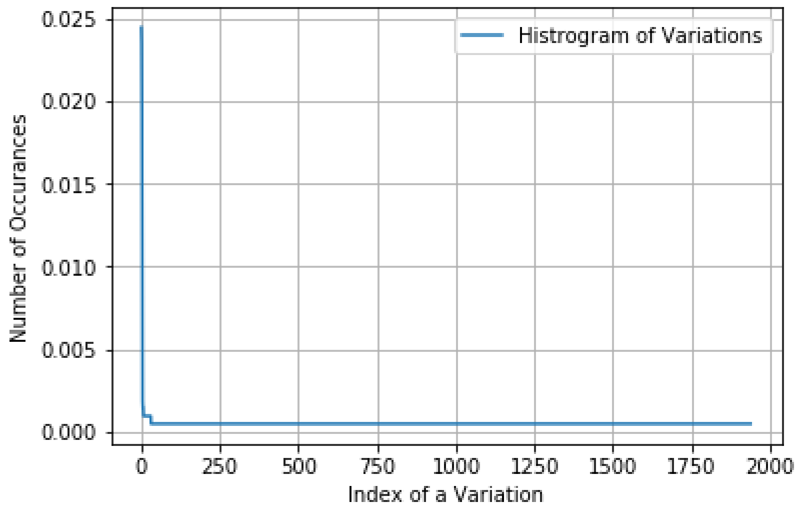
In [29]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1939
Truncating_Mutations    52
Deletion                50
Amplification           41
Fusions                 15
Overexpression           4
Q61L                     3
Q61R                     3
Y64A                     2
R170W                    2
E330K                    2
Name: Variation, dtype: int64
```

In [30]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of v
ariations in the train data, and they are distibuted as follows",)
```

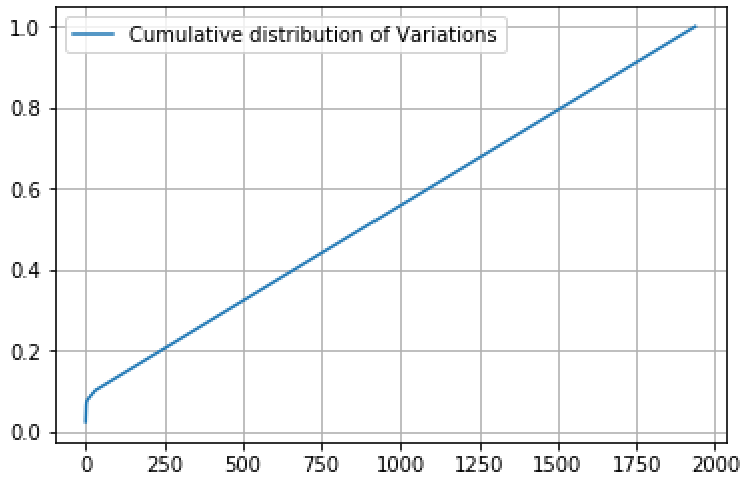
```
Ans: There are 1939 different categories of variations in the train data, and
they are distibuted as follows
```

In [31]:
```
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 [32]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.02448211 0.0480226  0.0673258  ... 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 [33]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Varia
tion", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variat
ion", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variatio
n", cv_df))
```

In [34]:
```
print("train_variation_feature_responseCoding is a converted feature using the
response coding method. The shape of Variation feature:", train_variation_feat
ure_responseCoding.shape)
```

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

In [35]:
```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(trai
n_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df[
'Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Vari
ation'])
```

In [36]:
```
print("train_variation_feature_onehotEncoded is converted feature using the on
ne-hot encoding method. The shape of Variation feature:", train_variation_feat
ure_onehotCoding.shape)
```

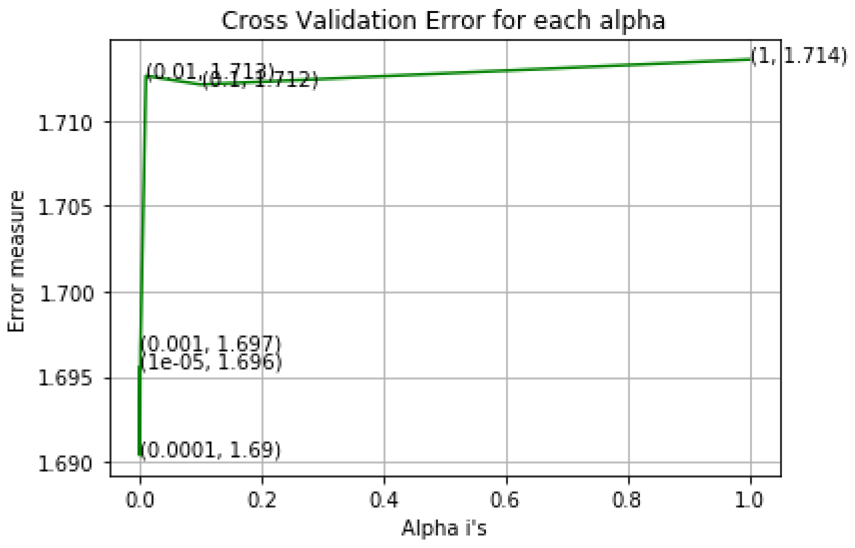
```
train_variation_feature_onehotEncoded is converted feature using the onne-hot
encoding method. The shape of Variation feature: (2124, 1973)
```

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

Let's build a model just like the earlier!

In [37]:
```
alpha = [10 ** x for x in range(-5, 1)]

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

# some of methods
# fit(X, y[, coef_init, intercept_init, …])     Fit linear model with Stochast
ic 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_, e
ps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predi
ct_y, labels=clf.classes_, eps=1e-15))

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


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

predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss i
s:",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_loss(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 i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.6955477942667168
For values of alpha =  0.0001 The log loss is: 1.6903773117608554
For values of alpha =  0.001 The log loss is: 1.6965718919110084
For values of alpha =  0.01 The log loss is: 1.7126152285050742
For values of alpha =  0.1 The log loss is: 1.7121025941256351
For values of alpha =  1 The log loss is: 1.7135711277220442
```



```
For values of best alpha =  0.0001 The train log loss is: 0.6900581252091896
For values of best alpha =  0.0001 The cross validation log loss is: 1.690377
3117608554
For values of best alpha =  0.0001 The test log loss is: 1.7027076929439666
```

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 [38]:
```
print("Q12. How many data points are covered by total ", unique_variations.sha
pe[0], " genes in test 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'])))].s
hape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(te
st_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":"
,(cv_coverage/cv_df.shape[0])*100)
```

```
Q12. How many data points are covered by total  1939  genes in test and cross
validation data sets?
Ans
1. In test data 76 out of 665 : 11.428571428571429
2. In cross validation data 59 out of  532 : 11.090225563909774
```

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 [39]:
```
# 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 [40]:

```
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_di
ct.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(
row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

In [41]:

```
# building a CountVectorizer with all the words that occured minimum 3 times i
n train data
text_vectorizer = TfidfVectorizer(min_df=5, max_features=3000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEX
T'])
# 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 return
s (1*number of features) vector
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

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


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

```
Total number of unique words in train data : 3000
```

In [42]:
```
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 [43]:
```
#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 [44]:
```
# 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_te
xt_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_fea
ture_responseCoding.sum(axis=1)).T
```

In [45]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, a
xis=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, axi
s=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 [46]:
```
#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 [47]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({193.87177083533788: 1, 127.28309351290949: 1, 123.9516369205886: 1,
95.27634367108831: 1, 89.49135585896761: 1, 88.45930440253197: 1, 87.76525239
990544: 1, 82.71406689968998: 1, 82.30785911801122: 1, 80.93104160221537: 1,
78.18050812460085: 1, 76.53874053412723: 1, 73.26940924075917: 1, 73.11228364
820379: 1, 68.55171974419262: 1, 62.968953418572504: 1, 59.55562691755885: 1,
59.510612038449665: 1, 59.1312029163687: 1, 58.710685016121815: 1, 54.9615870
39993695: 1, 53.50305936525071: 1, 52.75427793736526: 1, 50.009559714074165:
1, 49.319185747759235: 1, 48.574687892104016: 1, 48.431909215092396: 1, 47.85
8275167061656: 1, 47.29853833934683: 1, 47.09753485344249: 1, 46.685476140911
18: 1, 46.633634991667: 1, 46.39696865552379: 1, 46.12490703291477: 1, 44.349
5023026693: 1, 43.75740487272057: 1, 42.39872058071737: 1, 42.04550958731254:
1, 41.852064292591606: 1, 41.69793644078394: 1, 41.236905830027645: 1, 38.768
32817970881: 1, 38.36146652266362: 1, 38.28310233781683: 1, 38.18379421696625
5: 1, 37.19653306267872: 1, 36.73446706081922: 1, 35.35206271696007: 1, 35.30
4245996714904: 1, 34.827951309150414: 1, 34.780114553426515: 1, 34.0698445848
9117: 1, 33.92291938153465: 1, 33.9041744517319: 1, 33.89854120904206: 1, 32.
942589686711024: 1, 32.737986458638595: 1, 32.7064445822106: 1, 32.6018996433
4747: 1, 32.34950914849501: 1, 32.15076175713733: 1, 31.84663263110182: 1, 3
1.816163172959843: 1, 31.537507132038073: 1, 31.465391384465065: 1, 31.404846
060824234: 1, 31.179353318734737: 1, 31.04579669688957: 1, 30.93285688129221:
1, 30.88685889973828: 1, 30.384306328324268: 1, 30.025024180905522: 1, 29.770
49631673016: 1, 29.05573376031132: 1, 28.64883117005774: 1, 28.06417342315510
4: 1, 28.010170657849454: 1, 27.970425044562244: 1, 27.820646515406946: 1, 2
7.803149904907784: 1, 27.182720099938887: 1, 27.149771188392343: 1, 26.727076
375650906: 1, 26.517874815548407: 1, 26.38186325191481: 1, 26.14252933000759:
1, 26.115156805921895: 1, 25.79741421789772: 1, 25.639982812347817: 1, 25.611
866816098853: 1, 25.584730069851364: 1, 25.536882700841442: 1, 25.32969356052
2813: 1, 25.29846431435189: 1, 25.29668307254763: 1, 25.236628476995605: 1, 2
4.55330129720298: 1, 24.494726724436862: 1, 24.341945191668792: 1, 24.1582531
1610082: 1, 24.14330860512465: 1, 24.12679440864085: 1, 24.086753080561554:
1, 23.82946234270497: 1, 23.821917601049005: 1, 23.712575922538772: 1, 23.545
73826737324: 1, 23.47870478753906: 1, 23.387292663136407: 1, 23.3775328814707
07: 1, 23.340693666195193: 1, 23.268320701511172: 1, 23.18263950738593: 1, 2
3.155641711161437: 1, 22.950608103298716: 1, 22.857600219823762: 1, 22.813696
116894942: 1, 22.77032918483672: 1, 22.675767248823774: 1, 22.64985415296104
4: 1, 22.645705311908795: 1, 22.586473868991188: 1, 22.36137565698606: 1, 22.
287709101263392: 1, 22.09056389506355: 1, 22.00807404739081: 1, 21.6787732558
21426: 1, 21.640104608349674: 1, 21.584191346543385: 1, 21.577724201250284:
1, 21.52389504676622: 1, 21.470735472247632: 1, 21.444506386428763: 1, 21.418
982053153037: 1, 21.395844854374257: 1, 21.359828044008662: 1, 21.27975089268
496: 1, 21.223390604842685: 1, 21.19860023804342: 1, 20.851213292892687: 1, 2
0.745877206617475: 1, 20.52753789104859: 1, 20.523727758778954: 1, 20.4891284
8544538: 1, 20.33910766828922: 1, 20.167470548830185: 1, 20.154769057058118:
1, 20.06852057701836: 1, 20.03566976446402: 1, 19.862797219185435: 1, 19.8451
9935472838: 1, 19.76158558931519: 1, 19.742694582458224: 1, 19.64712471458678
4: 1, 19.46151857851957: 1, 19.24653259411346: 1, 19.245469968784846: 1, 19.1
54599758222854: 1, 19.14895836489851: 1, 19.112079013412984: 1, 19.0459202834
70775: 1, 18.960873782331557: 1, 18.93364273704337: 1, 18.721558286760274: 1,
18.62943376782954: 1, 18.595130187655258: 1, 18.44946480649205: 1, 18.4111557
86653406: 1, 18.359536652868297: 1, 18.35848275111671: 1, 18.248329827819315:
1, 18.09490245683486: 1, 18.084199487822094: 1, 18.044819361193397: 1, 17.990
12332399031: 1, 17.949204085379215: 1, 17.934923217489583: 1, 17.884234630584
78: 1, 17.872702364320283: 1, 17.74758945302231: 1, 17.703948332579003: 1, 1
7.637613373092826: 1, 17.585836455695883: 1, 17.56794413494214: 1, 17.4886831
3947405: 1, 17.45907203010106: 1, 17.38980471286387: 1, 17.335336445756194:
1, 17.317338515621095: 1, 17.287297644762898: 1, 17.274407813691397: 1, 17.22
8776800765267: 1, 17.189025493079217: 1, 17.080053051256744: 1, 17.0269961198
```

```
17124: 1, 16.94736850576957: 1, 16.937238260697242: 1, 16.92707837740094: 1,
16.873267800376592: 1, 16.80757203255165: 1, 16.788755520065184: 1, 16.787797
607882883: 1, 16.695541756564026: 1, 16.65966348913304: 1, 16.60687298507159:
1, 16.552940487685934: 1, 16.534108090874952: 1, 16.52738883406207: 1, 16.514
14114554796: 1, 16.50707629020627: 1, 16.48939477083588: 1, 16.48200919698082
6: 1, 16.466728204503095: 1, 16.432979680189405: 1, 16.37123728324817: 1, 16.
35748968600242: 1, 16.356882471162866: 1, 16.32390347519769: 1, 16.3183575695
1897: 1, 16.30827208639079: 1, 16.2426531365701: 1, 16.18238930489689: 1, 16.
148121649469395: 1, 16.079738469387085: 1, 16.022740925202854: 1, 16.01587318
4019533: 1, 15.966435013976854: 1, 15.932389374362169: 1, 15.886452345572064:
1, 15.876943825690718: 1, 15.818012521863153: 1, 15.808666825687249: 1, 15.78
8610711518652: 1, 15.756239108108751: 1, 15.753064291926602: 1, 15.7366146959
04471: 1, 15.706950115728686: 1, 15.671470968163481: 1, 15.637836529658067:
1, 15.633863290073712: 1, 15.504612395888499: 1, 15.490847742885952: 1, 15.36
6983860938388: 1, 15.364041936815338: 1, 15.195036321084595: 1, 15.1352686469
84635: 1, 15.111805353268913: 1, 15.086259178431757: 1, 15.05986042962984: 1,
15.057219643825707: 1, 14.981801400176925: 1, 14.968607333865515: 1, 14.95551
0287433821: 1, 14.88673746281047: 1, 14.849640340755055: 1, 14.81806730121445
7: 1, 14.814590370005465: 1, 14.813778078036: 1, 14.79891583199587: 1, 14.741
78672116593: 1, 14.699812626959647: 1, 14.681173343597512: 1, 14.679841219396
309: 1, 14.676736822258105: 1, 14.67551602740085: 1, 14.632358952784674: 1, 1
4.624541234541802: 1, 14.601375387238047: 1, 14.559617619572581: 1, 14.538656
756363382: 1, 14.514724865542027: 1, 14.495833663987867: 1, 14.47468927776008
1: 1, 14.433408673537075: 1, 14.414712689965071: 1, 14.28274450565617: 1, 14.
281013533777983: 1, 14.270527603027809: 1, 14.267767736684812: 1, 14.19697593
5587318: 1, 14.1446419676323: 1, 14.107024521764437: 1, 14.084751984451666:
1, 14.02654160218381: 1, 13.997834976894534: 1, 13.989710388698853: 1, 13.963
819564692901: 1, 13.919518160940248: 1, 13.900835983967696: 1, 13.87159424444
936: 1, 13.867320797013845: 1, 13.847494447919305: 1, 13.836576683874544: 1,
13.83563676402577: 1, 13.790796118444796: 1, 13.778248957427495: 1, 13.757861
400653118: 1, 13.731206172860306: 1, 13.703502851275656: 1, 13.66779509519682
8: 1, 13.599589690769248: 1, 13.594575755486794: 1, 13.548688954171345: 1, 1
3.482762509439539: 1, 13.440569987481874: 1, 13.38641793126872: 1, 13.3425821
0857564: 1, 13.336525061689692: 1, 13.33123501097349: 1, 13.321021295857292:
1, 13.302693413029436: 1, 13.290895371379934: 1, 13.269384982443809: 1, 13.26
6528911387299: 1, 13.241690585971504: 1, 13.234784306573644: 1, 13.2306004751
87822: 1, 13.210330352129608: 1, 13.120336500904813: 1, 13.119645923457323:
1, 13.058778374336736: 1, 13.01087343266874: 1, 12.986081187714483: 1, 12.970
309528738015: 1, 12.94979826894049: 1, 12.946334178506508: 1, 12.936812569181
653: 1, 12.926583650548002: 1, 12.801458649053924: 1, 12.789286822856653: 1,
12.765275060577721: 1, 12.700110213923047: 1, 12.691891093991586: 1, 12.67033
4156557304: 1, 12.656930899620283: 1, 12.65454213276898: 1, 12.6406780907891
9: 1, 12.624107292431814: 1, 12.623569532534193: 1, 12.62355092909542: 1, 12.
610246544326051: 1, 12.588405928729927: 1, 12.588283338654781: 1, 12.57537224
3755382: 1, 12.562889003193655: 1, 12.554073495095567: 1, 12.546541999335133:
1, 12.53586937336321: 1, 12.535129086488594: 1, 12.513274014843175: 1, 12.495
926366646575: 1, 12.487044995277369: 1, 12.466053790877998: 1, 12.44949058149
0972: 1, 12.447726121498402: 1, 12.426755036746556: 1, 12.42291916401745: 1,
12.420311051234448: 1, 12.377034939798671: 1, 12.370828090927223: 1, 12.35661
7960977506: 1, 12.339517760604481: 1, 12.27494943293379: 1, 12.26116302446623
3: 1, 12.25476157672538: 1, 12.171768615519705: 1, 12.117801372725683: 1, 12.
102088381772747: 1, 12.08056384679612: 1, 12.029243415881265: 1, 12.022783636
139009: 1, 12.00654938462864: 1, 11.98564732502804: 1, 11.948078805000627: 1,
11.933705889217519: 1, 11.928296723947097: 1, 11.92624242779842: 1, 11.915946
632793808: 1, 11.897180859955334: 1, 11.869265637892836: 1, 11.86051917702209
3: 1, 11.859561703265424: 1, 11.831308034135919: 1, 11.810038321244033: 1, 1
1.805611633877767: 1, 11.79884399456977: 1, 11.790977444222223: 1, 11.7576270
```

```
4134127: 1, 11.754016871448947: 1, 11.735547316489319: 1, 11.735077016429472:
1, 11.723189394404564: 1, 11.675715424478197: 1, 11.66859802184892: 1, 11.646
642557405771: 1, 11.641236555971643: 1, 11.595304163210722: 1, 11.5866296931
46927: 1, 11.572960605158405: 1, 11.540349087800944: 1, 11.53118849975283: 1,
11.521949347056994: 1, 11.505186603019377: 1, 11.494956703870832: 1, 11.44198
830512959: 1, 11.438483693873028: 1, 11.432973388806673: 1, 11.41734688433126
6: 1, 11.393914347187769: 1, 11.369176165454775: 1, 11.33499683421711: 1, 11.
326361259257189: 1, 11.319706476217428: 1, 11.316454184238045: 1, 11.27110988
6472384: 1, 11.267952043215322: 1, 11.262800379816474: 1, 11.245905539953355:
1, 11.240573424634356: 1, 11.207957405076423: 1, 11.190394513111626: 1, 11.16
7412210801757: 1, 11.156050346372473: 1, 11.151640515796164: 1, 11.1495436412
97463: 1, 11.145455750614522: 1, 11.119502412521424: 1, 11.116547945967413:
1, 11.106911752047852: 1, 11.090971580383645: 1, 11.076486282231782: 1, 11.06
1304284527152: 1, 11.057849570436023: 1, 11.043292235219473: 1, 11.0202559552
8952: 1, 11.01860704754094: 1, 10.999120840808736: 1, 10.987889596304957: 1,
10.958303770159073: 1, 10.952874416730497: 1, 10.914567431384604: 1, 10.90590
9991022366: 1, 10.896341608578439: 1, 10.8945934931141: 1, 10.89297311010256
7: 1, 10.881822260316842: 1, 10.870525389936018: 1, 10.85936594052028: 1, 10.
821053850601357: 1, 10.818908333337935: 1, 10.809625350019266: 1, 10.75973675
1333937: 1, 10.734761028797177: 1, 10.731103387538472: 1, 10.714785044126108:
1, 10.68348802933575: 1, 10.665441499681657: 1, 10.661672298163642: 1, 10.639
944025515096: 1, 10.636324378533148: 1, 10.623802604833095: 1, 10.62133722817
4392: 1, 10.586469343851398: 1, 10.58229441093738: 1, 10.567278968171278: 1,
10.560816531890113: 1, 10.530583110636806: 1, 10.471882122633586: 1, 10.46235
628372567: 1, 10.459826244199402: 1, 10.446393346175403: 1, 10.43589661495077
8: 1, 10.423345042283165: 1, 10.422698826935408: 1, 10.42121626378798: 1, 10.
41814743214225: 1, 10.368883307543694: 1, 10.368175039343624: 1, 10.346714753
467202: 1, 10.331128244244729: 1, 10.329075385423813: 1, 10.32130378095868:
1, 10.309479215773884: 1, 10.290523067517707: 1, 10.283045284895316: 1, 10.28
2882971124398: 1, 10.257600836367361: 1, 10.248416371898685: 1, 10.2364273884
6748: 1, 10.226348908871353: 1, 10.186253314245414: 1, 10.102953708025419: 1,
10.09684000117035: 1, 10.092540605202526: 1, 10.091093325097786: 1, 10.080642
747374194: 1, 10.077602284452578: 1, 10.069412761155105: 1, 10.06005608877648
8: 1, 10.055326130917162: 1, 10.024440861879574: 1, 9.997334858812804: 1, 9.9
90668467604703: 1, 9.981321137239831: 1, 9.95552819815024: 1, 9.9351203762103
02: 1, 9.911611034097799: 1, 9.893531407682614: 1, 9.891936001124042: 1, 9.88
936108619283: 1, 9.885301376443284: 1, 9.882372727868347: 1, 9.86680842928358
7: 1, 9.859160429340353: 1, 9.837106209280229: 1, 9.825959420543791: 1, 9.811
2448088251: 1, 9.808739008734754: 1, 9.791393898739832: 1, 9.789755911269435:
1, 9.78357253021833: 1, 9.781346474448164: 1, 9.764952805505455: 1, 9.7603298
07925507: 1, 9.749388520617195: 1, 9.739157942229017: 1, 9.728258610784243:
1, 9.724285438268778: 1, 9.698281630043647: 1, 9.676639661095681: 1, 9.659435
616326526: 1, 9.65111566998718: 1, 9.628914668167855: 1, 9.62492077816739: 1,
9.608204622196718: 1, 9.606739990896445: 1, 9.59816570043222: 1, 9.5919433253
82342: 1, 9.587397265553093: 1, 9.577754950714763: 1, 9.525041199524047: 1,
9.492317166280438: 1, 9.487316487586234: 1, 9.469477587565166: 1, 9.459301369
48915: 1, 9.445113479385144: 1, 9.427217202621506: 1, 9.416762999849544: 1,
9.411160665122882: 1, 9.409122141936972: 1, 9.376349999449756: 1, 9.374219106
1312: 1, 9.371858026154063: 1, 9.361904995314088: 1, 9.315378950542085: 1, 9.
28143027474465: 1, 9.268961027050198: 1, 9.259062366777679: 1, 9.251023038174
425: 1, 9.217185599335274: 1, 9.20211545700276: 1, 9.19990619884491: 1, 9.187
043445571469: 1, 9.185866448438118: 1, 9.184519227201287: 1, 9.18409343467945
6: 1, 9.178086407084518: 1, 9.175428826678544: 1, 9.156026353119788: 1, 9.148
374535424894: 1, 9.143377390651194: 1, 9.100950231551693: 1, 9.09811644906484
1: 1, 9.044937083466209: 1, 9.040058643173396: 1, 9.02049783123291: 1, 9.0038
01261387624: 1, 8.979866547012845: 1, 8.978773525261989: 1, 8.97848772436285
4: 1, 8.953866282457849: 1, 8.947329022097959: 1, 8.942648688849626: 1, 8.932
```

```
77045188741: 1, 8.932665504204666: 1, 8.926632130574177: 1, 8.92568163486072
1: 1, 8.924023272885329: 1, 8.923405953754594: 1, 8.915137270185033: 1, 8.907
550727052037: 1, 8.902223625742442: 1, 8.896613872175251: 1, 8.88700961264129
7: 1, 8.878691966388047: 1, 8.871626292363993: 1, 8.858954799391903: 1, 8.850
377365589361: 1, 8.848617873776874: 1, 8.820384114231032: 1, 8.81998005070059
7: 1, 8.809504382352126: 1, 8.807751191632764: 1, 8.784875344464497: 1, 8.757
974713923772: 1, 8.750481087258304: 1, 8.706195295211645: 1, 8.6870661340181
4: 1, 8.683674529945973: 1, 8.663113496342492: 1, 8.645339556267366: 1, 8.645
037333408299: 1, 8.642361272631426: 1, 8.63836389644485: 1, 8.63682229107017
6: 1, 8.633291329525521: 1, 8.629131975806622: 1, 8.625284497535212: 1, 8.623
299797175422: 1, 8.622454269864612: 1, 8.539979973995539: 1, 8.53761132018872
7: 1, 8.526383357673287: 1, 8.499088471221468: 1, 8.497297171883503: 1, 8.490
587397631803: 1, 8.48005734668533: 1, 8.475042182622056: 1, 8.46057898753323
7: 1, 8.458639895810867: 1, 8.434965292924764: 1, 8.427282785581891: 1, 8.425
42285426942: 1, 8.422601676207332: 1, 8.415866259214049: 1, 8.41439689942029
6: 1, 8.404269383592961: 1, 8.39716354296222: 1, 8.394195124279003: 1, 8.3784
15984090099: 1, 8.368582925327331: 1, 8.36586746016592: 1, 8.36414558065524:
1, 8.350669684140657: 1, 8.3469949463602: 1, 8.329227002740948: 1, 8.32333461
1707082: 1, 8.318904020394863: 1, 8.300499176273261: 1, 8.290087299134651: 1,
8.287641160770574: 1, 8.286322190307471: 1, 8.263021818951207: 1, 8.259795117
795553: 1, 8.25312623233653: 1, 8.236973775947327: 1, 8.23673087441342: 1, 8.
199496202266419: 1, 8.197832515172157: 1, 8.19651904097332: 1, 8.192841185504
019: 1, 8.189442061573079: 1, 8.178796231610784: 1, 8.176141118068523: 1, 8.1
59208148532793: 1, 8.158089063196696: 1, 8.153971765921566: 1, 8.145521302184
696: 1, 8.126870171666573: 1, 8.10922630798033: 1, 8.104947599442662: 1, 8.10
0345653178847: 1, 8.086735768897473: 1, 8.078165937468501: 1, 8.0758008578393
92: 1, 8.072893355831859: 1, 8.054767234339854: 1, 8.052612531077932: 1, 8.04
4135541723708: 1, 8.039082020742235: 1, 8.032799949088858: 1, 8.0234890468528
23: 1, 8.013688903642468: 1, 8.012721842264762: 1, 8.009694980804325: 1, 8.00
5382963592792: 1, 7.996739400138342: 1, 7.990426092212076: 1, 7.9901365288587
69: 1, 7.980166408187319: 1, 7.9754681743928275: 1, 7.974646035199601: 1, 7.9
703083851853345: 1, 7.965779840409521: 1, 7.959536920143209: 1, 7.94781266533
9138: 1, 7.93668325716446: 1, 7.9341318952936435: 1, 7.917228985563086: 1, 7.
9165684536642065: 1, 7.908504131300155: 1, 7.897371299452761: 1, 7.8932922809
26125: 1, 7.880770532153867: 1, 7.879592339105393: 1, 7.8573833036054515: 1,
7.856030165834487: 1, 7.854266059964118: 1, 7.843427917138643: 1, 7.840871562
158454: 1, 7.829259123537958: 1, 7.822818378790573: 1, 7.820793042708781: 1,
7.820401656074233: 1, 7.820094868030294: 1, 7.812293830907494: 1, 7.799875600
796433: 1, 7.7894243135781345: 1, 7.785869458882815: 1, 7.780305071986679: 1,
7.77557314894385: 1, 7.775075380455164: 1, 7.768059638887945: 1, 7.7680427385
14337: 1, 7.7530045163888985: 1, 7.749639116811307: 1, 7.738893191330873: 1,
7.731819863607597: 1, 7.716676777107958: 1, 7.712596154745814: 1, 7.707583709
461364: 1, 7.697043934266297: 1, 7.691110107328914: 1, 7.683740599918559: 1,
7.678993481371007: 1, 7.676874435604698: 1, 7.675694163742105: 1, 7.672760653
271402: 1, 7.665966035569618: 1, 7.656657352204757: 1, 7.65252263203452: 1,
7.649732685565413: 1, 7.636024467193714: 1, 7.629917351312548: 1, 7.620955888
071794: 1, 7.6015447463936: 1, 7.596486162014144: 1, 7.5923967682835345: 1,
7.58549807879333: 1, 7.579979104607622: 1, 7.578546434116645: 1, 7.5765519885
06318: 1, 7.5757148767474565: 1, 7.573513570668859: 1, 7.567729308850912: 1,
7.567112828470178: 1, 7.565050322142518: 1, 7.561208835257127: 1, 7.552880012
4009665: 1, 7.537706768010431: 1, 7.535302001347377: 1, 7.531305564176751: 1,
7.52552794729805: 1, 7.524056530006512: 1, 7.5004912639074774: 1, 7.500368956
5411795: 1, 7.494353748037146: 1, 7.493771133591274: 1, 7.492623575283266: 1,
7.491914594380652: 1, 7.490465858403182: 1, 7.481605860894506: 1, 7.464852301
690328: 1, 7.457897105867307: 1, 7.453723167873284: 1, 7.4338257822203975: 1,
7.43205071248247: 1, 7.426882506110944: 1, 7.417724229312042: 1, 7.4156609775
6492: 1, 7.407429070769639: 1, 7.407069248447494: 1, 7.395224756054181: 1, 7.
```

```
385027498918969: 1, 7.3819470280007735: 1, 7.376295363422325: 1, 7.3724160693
59654: 1, 7.372168934412718: 1, 7.37056954351975: 1, 7.358697481142908: 1, 7.
3567605805970775: 1, 7.34524697777243: 1, 7.331273447482039: 1, 7.33089710486
8821: 1, 7.328834950109552: 1, 7.325348063947937: 1, 7.324513343679015: 1, 7.
3231926599574075: 1, 7.311743325620549: 1, 7.302397362611505: 1, 7.2984641220
27087: 1, 7.297428651857971: 1, 7.277788625821352: 1, 7.272069622891181: 1,
7.2665605508413345: 1, 7.2661654985144235: 1, 7.262079633582819: 1, 7.2434359
60393033: 1, 7.230768563286913: 1, 7.225318046248361: 1, 7.224601996062507:
1, 7.222931691983325: 1, 7.208993595561583: 1, 7.203028236280691: 1, 7.198497
556567738: 1, 7.189427289310809: 1, 7.188144049552843: 1, 7.185338183266622:
1, 7.18529817691394: 1, 7.1827591387949274: 1, 7.176924726934586: 1, 7.169701
16603311: 1, 7.168170338187: 1, 7.167533576490072: 1, 7.164591213711705: 1,
7.162829408899581: 1, 7.150537313265291: 1, 7.145527576123244: 1, 7.139860601
870029: 1, 7.125138716502345: 1, 7.1197326624881345: 1, 7.113935992326688: 1,
7.086680818516647: 1, 7.078379824106145: 1, 7.076529204930513: 1, 7.074931343
993786: 1, 7.074705862731387: 1, 7.068190397955633: 1, 7.062720125432995: 1,
7.062642331534362: 1, 7.05886435676685: 1, 7.053358305585349: 1, 7.0523900065
341945: 1, 7.051368268589938: 1, 7.046863796043614: 1, 7.038957163457341: 1,
7.033446692354523: 1, 7.0210283715620765: 1, 7.019799909346458: 1, 7.01216829
1908941: 1, 7.011694021876302: 1, 7.007719003018658: 1, 7.006023008567924: 1,
6.9959401902388745: 1, 6.98468284329739: 1, 6.9805903252373245: 1, 6.97933445
1339554: 1, 6.971098125877917: 1, 6.9554082288400725: 1, 6.954517803341356:
1, 6.94392890762758: 1, 6.940093676373766: 1, 6.938466620167119: 1, 6.9347370
483602555: 1, 6.9298749791095755: 1, 6.909704549436163: 1, 6.907845457791064:
1, 6.901959484344427: 1, 6.898311104203807: 1, 6.894764557956705: 1, 6.894307
824148758: 1, 6.887913513507078: 1, 6.885250412753548: 1, 6.878559909685794:
1, 6.8735074198405375: 1, 6.871892362838734: 1, 6.854175974631295: 1, 6.85320
8855738512: 1, 6.852695088904501: 1, 6.852479371045506: 1, 6.850529910927380
5: 1, 6.848901019670414: 1, 6.847977676642771: 1, 6.847698981305847: 1, 6.841
574098291193: 1, 6.834295879635943: 1, 6.831463850304009: 1, 6.82869023779485
75: 1, 6.828175471884745: 1, 6.821282177221524: 1, 6.820347352376933: 1, 6.81
6214959152492: 1, 6.810926934973605: 1, 6.803112095082661: 1, 6.8030972747034
02: 1, 6.794529956854853: 1, 6.7903911116017435: 1, 6.78232376328574: 1, 6.77
7583160605031: 1, 6.775915249339214: 1, 6.774530503634549: 1, 6.7735224203228
91: 1, 6.769666568038315: 1, 6.764643238276599: 1, 6.751309937839854: 1, 6.74
5173794883556: 1, 6.742013271024642: 1, 6.74093020068589: 1, 6.72985033008159
6: 1, 6.726340394284921: 1, 6.699354858786156: 1, 6.698770112285369: 1, 6.689
005374934036: 1, 6.68729382598378: 1, 6.684576240219602: 1, 6.68390670165736:
1, 6.681762667767702: 1, 6.673987329714448: 1, 6.669151504768222: 1, 6.666958
351142194: 1, 6.661267695476758: 1, 6.66124648141493: 1, 6.660812835520064:
1, 6.650407003089762: 1, 6.647890434153005: 1, 6.64734754942142: 1, 6.6443683
12676256: 1, 6.641620940430767: 1, 6.632954746664222: 1, 6.621418936891897:
1, 6.620526460474487: 1, 6.612205716945513: 1, 6.61173678726514: 1, 6.5972884
49353827: 1, 6.595988474359389: 1, 6.590359087750559: 1, 6.588224735913206:
1, 6.587163473580585: 1, 6.577552745013945: 1, 6.576888470622014: 1, 6.569511
4066511735: 1, 6.56718844356095: 1, 6.564037781687151: 1, 6.560358327961881:
1, 6.552803869793501: 1, 6.549844933179496: 1, 6.549624354492555: 1, 6.541580
297686458: 1, 6.535347223492063: 1, 6.53460794331718: 1, 6.532686506743115:
1, 6.531507557993811: 1, 6.529916853096697: 1, 6.529279616995534: 1, 6.525091
000664415: 1, 6.511909516968993: 1, 6.500566721564747: 1, 6.50044493232545:
1, 6.493574685751356: 1, 6.492877804080287: 1, 6.49109404820549: 1, 6.4859416
72926976: 1, 6.468077565327889: 1, 6.466009118249975: 1, 6.4657841287346205:
1, 6.453741683621443: 1, 6.451064988245463: 1, 6.446786839376672: 1, 6.443740
163383907: 1, 6.443328497894149: 1, 6.442063274999907: 1, 6.4322688890632165:
1, 6.430388450024545: 1, 6.414174728703053: 1, 6.412904031538668: 1, 6.411988
1816783995: 1, 6.409482842692491: 1, 6.406560257799249: 1, 6.398505130200458
5: 1, 6.397573811915197: 1, 6.391072525463014: 1, 6.388345099858011: 1, 6.387
```

```
515652544683: 1, 6.37657002806621: 1, 6.375968690930346: 1, 6.37143688374543
4: 1, 6.369663986862744: 1, 6.36427765591762: 1, 6.357485637959014: 1, 6.3558
146841532155: 1, 6.353569166626952: 1, 6.350446525381036: 1, 6.34360864583478
6: 1, 6.343515696610905: 1, 6.342846271921501: 1, 6.330114316805535: 1, 6.318
510838774051: 1, 6.31076425869817: 1, 6.309377207073936: 1, 6.30783290334857
7: 1, 6.300637022746812: 1, 6.3002771534142115: 1, 6.297436266528405: 1, 6.29
7384824091763: 1, 6.296747839498341: 1, 6.293186404242105: 1, 6.2908182915794
58: 1, 6.288218211761155: 1, 6.285955798483476: 1, 6.284656131636122: 1, 6.28
4594835555546: 1, 6.278449528248002: 1, 6.277489834896366: 1, 6.2747265348684
77: 1, 6.269938232948384: 1, 6.266859971245541: 1, 6.263543935437717: 1, 6.25
5404014295078: 1, 6.240498955475491: 1, 6.2350549927997125: 1, 6.230784291799
234: 1, 6.223093291494692: 1, 6.215807478288429: 1, 6.2151788412262645: 1, 6.
210754851449287: 1, 6.207188072930951: 1, 6.198035626402154: 1, 6.18612763195
2373: 1, 6.1767855743079325: 1, 6.173506632341786: 1, 6.171269851182426: 1,
6.1710740393201515: 1, 6.169799839533524: 1, 6.155573851923314: 1, 6.14885926
3596764: 1, 6.143640313807616: 1, 6.14312992322315: 1, 6.1320208784665695: 1,
6.128467130836381: 1, 6.120705207179262: 1, 6.114179154816723: 1, 6.109773197
1247455: 1, 6.103340374163565: 1, 6.102012947412282: 1, 6.090922503329984: 1,
6.087949730722574: 1, 6.068762702964481: 1, 6.066073412103794: 1, 6.055142427
860265: 1, 6.05315939727707: 1, 6.04898133909134: 1, 6.0465059003174435: 1,
6.04408740068659: 1, 6.033234923616095: 1, 6.032738671834175: 1, 6.0326402380
230135: 1, 6.031184360703106: 1, 6.030987914541547: 1, 6.027566910436541: 1,
6.026188322247555: 1, 6.020403034597123: 1, 6.020384053285587: 1, 6.014339933
300137: 1, 6.012582291479741: 1, 6.005418856249745: 1, 5.997918988459353: 1,
5.995801759277665: 1, 5.9937885587966155: 1, 5.981621599436273: 1, 5.96330113
3973628: 1, 5.963281235464185: 1, 5.957850874438121: 1, 5.955831943136923: 1,
5.9510724189885815: 1, 5.940128428202666: 1, 5.93221525427665: 1, 5.931288025
659112: 1, 5.916638237477272: 1, 5.905862372888252: 1, 5.9052447376936605: 1,
5.903366434790971: 1, 5.902945796042184: 1, 5.898689280883562: 1, 5.894167980
711571: 1, 5.892900188189126: 1, 5.892185013133433: 1, 5.888131343212597: 1,
5.881935727955796: 1, 5.8796701738692265: 1, 5.879339079862091: 1, 5.87903135
796333: 1, 5.878674793317592: 1, 5.873186054052349: 1, 5.872417267999007: 1,
5.871716351528401: 1, 5.871545392294013: 1, 5.8689512786304: 1, 5.86724635678
7675: 1, 5.866252754207397: 1, 5.865140510010267: 1, 5.864697550153495: 1, 5.
863277516457579: 1, 5.827541000869115: 1, 5.820460543945737: 1, 5.82033843151
8924: 1, 5.816609915855818: 1, 5.812473013029323: 1, 5.810921940532548: 1, 5.
808729284544381: 1, 5.807362529109448: 1, 5.806761023161498: 1, 5.79948965658
5405: 1, 5.795160049823191: 1, 5.794873374933489: 1, 5.791661119087586: 1, 5.
789793164596802: 1, 5.786540646054724: 1, 5.7795738186452805: 1, 5.7780465826
80203: 1, 5.777067251507555: 1, 5.776276099677748: 1, 5.764876577577711: 1,
5.7628945516756875: 1, 5.7618090485124664: 1, 5.761457710635057: 1, 5.7534226
95057535: 1, 5.746870806364432: 1, 5.7459469253980515: 1, 5.744517156137161:
1, 5.735374105623882: 1, 5.734578031617064: 1, 5.731857274609318: 1, 5.724203
101725122: 1, 5.714956597172376: 1, 5.705436191221437: 1, 5.698972084352073:
1, 5.6970273401037606: 1, 5.696021917535018: 1, 5.688796139031355: 1, 5.68513
5249899602: 1, 5.680966883163246: 1, 5.677249182544254: 1, 5.675707089953009
5: 1, 5.675437518885716: 1, 5.6729280624336775: 1, 5.671534607999155: 1, 5.67
1378729201887: 1, 5.667817324345495: 1, 5.666110706895406: 1, 5.6644599933718
91: 1, 5.662361778936175: 1, 5.659338077513867: 1, 5.654949639289688: 1, 5.65
1152739083522: 1, 5.64648153451257: 1, 5.6456408578683925: 1, 5.6455768785600
9: 1, 5.642387656601188: 1, 5.63619658016733: 1, 5.635630846393857: 1, 5.6352
21340681548: 1, 5.632936918052099: 1, 5.632661427044247: 1, 5.62721997978953
5: 1, 5.622790841749682: 1, 5.6161493563275435: 1, 5.612023659247025: 1, 5.61
0394600653861: 1, 5.609700829386011: 1, 5.6090255083173215: 1, 5.587642432069
077: 1, 5.571540304302698: 1, 5.567584133209463: 1, 5.566548771163745: 1, 5.5
64980508358852: 1, 5.560596427956347: 1, 5.559207369665225: 1, 5.556331395146
322: 1, 5.554376597003859: 1, 5.550081273028058: 1, 5.547193415108779: 1, 5.5
```

```
44473416644668: 1, 5.544400878271412: 1, 5.539652790386198: 1, 5.539450604546
589: 1, 5.53906053588975: 1, 5.538087433354151: 1, 5.537680979174492: 1, 5.53
2216497137921: 1, 5.531759673215449: 1, 5.528803121680563: 1, 5.5277626039973
34: 1, 5.527232643350295: 1, 5.525317823436987: 1, 5.522496152248665: 1, 5.51
8442951850567: 1, 5.517501004378755: 1, 5.514789032844336: 1, 5.5131360501271
96: 1, 5.509570762589798: 1, 5.509265929259796: 1, 5.502619362626053: 1, 5.49
9502931485873: 1, 5.4978204026275455: 1, 5.487503438158661: 1, 5.487107851631
773: 1, 5.480856450179883: 1, 5.478234173310868: 1, 5.475965353710287: 1, 5.4
69877361339219: 1, 5.4654541369438485: 1, 5.459118148306031: 1, 5.45771256904
6439: 1, 5.453377406042187: 1, 5.45265396168987: 1, 5.452589109415177: 1, 5.4
4790746061192: 1, 5.437268425032982: 1, 5.43609944128947: 1, 5.43468453469821
4: 1, 5.433649988177073: 1, 5.432823166833012: 1, 5.428995793962675: 1, 5.413
745194587122: 1, 5.400927968954027: 1, 5.390493147575962: 1, 5.3887833288183
9: 1, 5.387665791210405: 1, 5.386656895780422: 1, 5.384585840743472: 1, 5.379
293969002137: 1, 5.379227250341898: 1, 5.377500046998303: 1, 5.37459629689579
45: 1, 5.372585182011097: 1, 5.36171407454631: 1, 5.360579884511451: 1, 5.356
999404743565: 1, 5.354303291990656: 1, 5.34736561514534: 1, 5.34491478550216
7: 1, 5.336295728431571: 1, 5.335382053706493: 1, 5.3345367155924: 1, 5.33403
1698366675: 1, 5.332838832753911: 1, 5.315871978330009: 1, 5.312350240322388:
1, 5.312010451487504: 1, 5.303384849510536: 1, 5.299901531116548: 1, 5.297343
954089837: 1, 5.295348187415909: 1, 5.294550623383171: 1, 5.281256613628355:
1, 5.270653988803258: 1, 5.269199531168359: 1, 5.26789345517102: 1, 5.2673319
2622349: 1, 5.2553448253517425: 1, 5.250369616477234: 1, 5.2412893680704915:
1, 5.240603797175084: 1, 5.233052453649941: 1, 5.221022291338704: 1, 5.220154
985900155: 1, 5.219897921881736: 1, 5.2162402136173975: 1, 5.214937517261312:
1, 5.204758033966555: 1, 5.1891626176706716: 1, 5.180176874937202: 1, 5.17236
5194737352: 1, 5.171020225891639: 1, 5.165988858022691: 1, 5.162722796942465:
1, 5.159868194901314: 1, 5.157529211895454: 1, 5.1565084727591755: 1, 5.15532
26885405135: 1, 5.151675186659846: 1, 5.146450657743205: 1, 5.145528617761840
5: 1, 5.141127076156626: 1, 5.141083200602227: 1, 5.139781710817536: 1, 5.139
688834414076: 1, 5.139089077388967: 1, 5.12076479255567: 1, 5.11966449958473
1: 1, 5.117674234694734: 1, 5.113910118748087: 1, 5.112808779978584: 1, 5.107
0355533490055: 1, 5.10651074469701: 1, 5.105062322330148: 1, 5.10453083101992
7: 1, 5.103455441002889: 1, 5.101075269399237: 1, 5.089123378628886: 1, 5.081
763476040172: 1, 5.079457295905482: 1, 5.066154778580886: 1, 5.06421850429222
9: 1, 5.06366481361975: 1, 5.06154508401369: 1, 5.0613224879358425: 1, 5.0605
84710646138: 1, 5.060365959813561: 1, 5.059024069555945: 1, 5.05877452699054
6: 1, 5.058772407670942: 1, 5.056090353860968: 1, 5.054845978773364: 1, 5.054
052081722371: 1, 5.053574214627069: 1, 5.049689577204261: 1, 5.04406760545964
6: 1, 5.041720867945053: 1, 5.039799441326703: 1, 5.038179638534551: 1, 5.037
468254064151: 1, 5.036286852397473: 1, 5.034551516911696: 1, 5.02984164572060
5: 1, 5.027057478379977: 1, 5.025851467456503: 1, 5.022272845592937: 1, 5.021
8516493604985: 1, 5.012002877192234: 1, 5.010956962729517: 1, 5.0107518102712
45: 1, 5.00590894558583: 1, 4.9998222430877926: 1, 4.999406118296209: 1, 4.99
5274238269705: 1, 4.9941250324549955: 1, 4.992969202769172: 1, 4.989890927562
367: 1, 4.989126588358104: 1, 4.984636568144023: 1, 4.984079904521168: 1, 4.9
83935748109963: 1, 4.978945608054387: 1, 4.975931593549244: 1, 4.968872880977
529: 1, 4.9639864256453: 1, 4.95925440625599: 1, 4.957418109879136: 1, 4.9567
30058278727: 1, 4.9542932862439955: 1, 4.953568555696982: 1, 4.95204608123638
85: 1, 4.951984056520089: 1, 4.949504597714414: 1, 4.94910384983532: 1, 4.945
831319069116: 1, 4.943708963757511: 1, 4.941152134276553: 1, 4.93923935140912
1: 1, 4.9388249940126245: 1, 4.937250774777199: 1, 4.931420742741158: 1, 4.93
0290984876541: 1, 4.929176597274912: 1, 4.928042710020301: 1, 4.9274257103494
52: 1, 4.925761391897712: 1, 4.9255414855112765: 1, 4.92074932726269: 1, 4.90
8602937587921: 1, 4.904722999980031: 1, 4.900898019101675: 1, 4.8962825397782
22: 1, 4.894873208167134: 1, 4.893468547768292: 1, 4.893332566814321: 1, 4.89
3173927356759: 1, 4.888128684295156: 1, 4.887885758590525: 1, 4.8712108072082
```

```
45: 1, 4.870150995738257: 1, 4.868437250277618: 1, 4.866216945042803: 1, 4.86
1911814116857: 1, 4.850780277688517: 1, 4.846602844979642: 1, 4.8457685246290
69: 1, 4.84110901210518: 1, 4.83842483970687: 1, 4.835153024863251: 1, 4.8319
66902879494: 1, 4.831011547335977: 1, 4.830668597664459: 1, 4.82776079409738
9: 1, 4.826534614513615: 1, 4.820418311378226: 1, 4.819486121638456: 1, 4.807
738556319653: 1, 4.805542801357546: 1, 4.798551866414923: 1, 4.79767470990930
3: 1, 4.794986809490969: 1, 4.794974892649809: 1, 4.7936351392475665: 1, 4.79
35809865411665: 1, 4.790193386397884: 1, 4.786995777893809: 1, 4.784895085905
547: 1, 4.78131600944506: 1, 4.78057216521009: 1, 4.77886162310863: 1, 4.7777
30047984047: 1, 4.775928975997887: 1, 4.773960745590681: 1, 4.76989590968276
3: 1, 4.768799069609183: 1, 4.768740861658539: 1, 4.763246649261218: 1, 4.758
978427375315: 1, 4.758886718904766: 1, 4.758790172143131: 1, 4.75535007021487
4: 1, 4.753758011774955: 1, 4.752901979869622: 1, 4.751985522616786: 1, 4.750
864484318578: 1, 4.750441425506989: 1, 4.750025940768779: 1, 4.74955087741366
1: 1, 4.747549134997196: 1, 4.745896619794648: 1, 4.74490589784158: 1, 4.7439
96725907298: 1, 4.742068360598096: 1, 4.734088430709834: 1, 4.73237113948665
6: 1, 4.724991393527957: 1, 4.722059816260948: 1, 4.719471377196999: 1, 4.719
21056632071: 1, 4.716612576973192: 1, 4.7146343724807025: 1, 4.70780419457291
1: 1, 4.700727612460848: 1, 4.699806380791066: 1, 4.698088117333121: 1, 4.697
066267356256: 1, 4.694727580635782: 1, 4.6908608935358185: 1, 4.6889172214117
05: 1, 4.684448165392596: 1, 4.683993547107189: 1, 4.683195100921038: 1, 4.67
688630368336: 1, 4.676074727808154: 1, 4.6667690290744135: 1, 4.6660499745421
44: 1, 4.6643728803387905: 1, 4.6636018650296975: 1, 4.662999024776231: 1, 4.
660291378441485: 1, 4.656367404850576: 1, 4.655781775232488: 1, 4.65506222957
4373: 1, 4.654190640633929: 1, 4.652876582467517: 1, 4.652251904177874: 1, 4.
651815025177954: 1, 4.65049644745777: 1, 4.649656907377585: 1, 4.648692646634
543: 1, 4.643686514270899: 1, 4.643426342905488: 1, 4.639180587445792: 1, 4.6
36916423645925: 1, 4.635182232581991: 1, 4.634658667262613: 1, 4.632843204464
626: 1, 4.630912328541166: 1, 4.627496862653686: 1, 4.6221014042579345: 1, 4.
621160723504042: 1, 4.6175081829657785: 1, 4.615707565490562: 1, 4.6087905281
81029: 1, 4.604406814830227: 1, 4.601368391728523: 1, 4.601264007948587: 1,
4.594061836410239: 1, 4.593074808042741: 1, 4.592251279947166: 1, 4.592123187
189329: 1, 4.58758786259062: 1, 4.5870436540547335: 1, 4.576212700212289: 1,
4.574359636581015: 1, 4.573174071672549: 1, 4.572036921777054: 1, 4.562208607
62768: 1, 4.560482067246486: 1, 4.559202617535509: 1, 4.555936003869667: 1,
4.55316169195895: 1, 4.552843496455748: 1, 4.547750875700927: 1, 4.5446086750
443415: 1, 4.54315633232467: 1, 4.543068596622529: 1, 4.53547706173976: 1, 4.
531471694244074: 1, 4.5277120951760645: 1, 4.527248973111972: 1, 4.5271620984
20333: 1, 4.527022629318957: 1, 4.526647729140743: 1, 4.52443748405229: 1, 4.
516951871091057: 1, 4.516123744762207: 1, 4.515119274114703: 1, 4.51254632346
7512: 1, 4.509080180114841: 1, 4.507898185017586: 1, 4.505865589680778: 1, 4.
504159446291289: 1, 4.500856078366969: 1, 4.499623843406334: 1, 4.49929027328
3084: 1, 4.49454630256541: 1, 4.4918585991235656: 1, 4.488065699819345: 1, 4.
486953891225096: 1, 4.486870239712206: 1, 4.484912608124847: 1, 4.48464797173
8969: 1, 4.48057148810334: 1, 4.478889244701549: 1, 4.478459049914942: 1, 4.4
78106815507935: 1, 4.476808794056319: 1, 4.476785735302247: 1, 4.473496685495
861: 1, 4.472834280704315: 1, 4.470158120372596: 1, 4.470098144695698: 1, 4.4
68862361518404: 1, 4.46699466352917: 1, 4.464904814342533: 1, 4.4639567986005
02: 1, 4.461882873957076: 1, 4.459943768152855: 1, 4.458357176196144: 1, 4.45
6170081203698: 1, 4.456115829012881: 1, 4.4512874449838815: 1, 4.449297052006
84: 1, 4.440534709574476: 1, 4.438105018468018: 1, 4.437542067664914: 1, 4.43
4013330283239: 1, 4.4298895375527065: 1, 4.4297671091770106: 1, 4.42931186841
8367: 1, 4.422440788205055: 1, 4.42127309269398: 1, 4.420917140346345: 1, 4.4
20784551633854: 1, 4.418835131408685: 1, 4.414328514727713: 1, 4.413474450207
572: 1, 4.413229130028185: 1, 4.4117481879257605: 1, 4.411440727279674: 1, 4.
405400760570337: 1, 4.40240933436706: 1, 4.3997023058762466: 1, 4.39774149355
1306: 1, 4.39669283557897: 1, 4.394956171498758: 1, 4.389043065971306: 1, 4.3
```

```
87704972370426: 1, 4.38263125453914: 1, 4.382300204154238: 1, 4.3767700689760
28: 1, 4.366082004734845: 1, 4.365421938280668: 1, 4.360675151832565: 1, 4.35
6627292289897: 1, 4.353017732674497: 1, 4.348551223841266: 1, 4.3478644931977
16: 1, 4.346360719468213: 1, 4.346109263623332: 1, 4.341347354677878: 1, 4.33
5766451094374: 1, 4.332849634571223: 1, 4.331979591661529: 1, 4.3301894343662
36: 1, 4.3296647158202095: 1, 4.329034667127301: 1, 4.3267978450449975: 1, 4.
321467903974046: 1, 4.319409168910403: 1, 4.318389284653249: 1, 4.31749397988
6841: 1, 4.316283809859334: 1, 4.308700033575829: 1, 4.308037052873442: 1, 4.
30485222254125: 1, 4.302819548758459: 1, 4.301785101886079: 1, 4.300221573735
576: 1, 4.293850303824495: 1, 4.2925658674443845: 1, 4.292339294438951: 1, 4.
291570801294596: 1, 4.2876353316794935: 1, 4.28527299690806: 1, 4.28318845140
7836: 1, 4.279475280185159: 1, 4.279216676365495: 1, 4.279084020979221: 1, 4.
275346166749218: 1, 4.274419760930619: 1, 4.272218921615013: 1, 4.26845192034
0591: 1, 4.267009818898967: 1, 4.26294783156286: 1, 4.2608420147522565: 1, 4.
260763386165405: 1, 4.257160230396238: 1, 4.256162400088198: 1, 4.25613768875
3975: 1, 4.255156647041396: 1, 4.253393731997442: 1, 4.25178998767272: 1, 4.2
443107941057505: 1, 4.242404599181007: 1, 4.238525534887753: 1, 4.23630533566
7456: 1, 4.235355845532031: 1, 4.234188725864385: 1, 4.233541352537479: 1, 4.
23330156640022: 1, 4.232084744584502: 1, 4.228357728312868: 1, 4.227266322522
207: 1, 4.22724426636135: 1, 4.226979108590222: 1, 4.226732186326534: 1, 4.22
512759170856: 1, 4.221147685309821: 1, 4.220940744278355: 1, 4.22082177744649
8: 1, 4.218819284022429: 1, 4.215956541517889: 1, 4.215618223236915: 1, 4.214
217204720615: 1, 4.211668757205356: 1, 4.209612951289684: 1, 4.20861585924912
25: 1, 4.205561825133731: 1, 4.203162848826229: 1, 4.1951476837506965: 1, 4.1
93202822274016: 1, 4.192027825142278: 1, 4.190787153581125: 1, 4.190690570675
403: 1, 4.18931420419578: 1, 4.186565690829107: 1, 4.181475371867971: 1, 4.17
7975857459942: 1, 4.177061633744532: 1, 4.174641363358141: 1, 4.1733012815772
71: 1, 4.172670748374196: 1, 4.170462082541809: 1, 4.170367835352791: 1, 4.16
9874778813473: 1, 4.166415934837324: 1, 4.166330700489613: 1, 4.1657585657039
17: 1, 4.165687832175082: 1, 4.165405097721872: 1, 4.158923338119645: 1, 4.15
6319457839494: 1, 4.1542752967030605: 1, 4.151509729894343: 1, 4.149644594371
968: 1, 4.14561420215654: 1, 4.14516541260324: 1, 4.144713456168646: 1, 4.142
440843057854: 1, 4.139506089918387: 1, 4.138748357228107: 1, 4.13755882273408
7: 1, 4.137141815395755: 1, 4.134476123275039: 1, 4.133902702786823: 1, 4.128
067921256107: 1, 4.126602104319434: 1, 4.121795655327156: 1, 4.11690163616694
8: 1, 4.115425590303513: 1, 4.111319784162429: 1, 4.110509890372155: 1, 4.108
932919916527: 1, 4.108387113275039: 1, 4.105087599662516: 1, 4.10256462300490
8: 1, 4.101307691591375: 1, 4.095273026614111: 1, 4.094620214327885: 1, 4.092
8492142561765: 1, 4.091819273459392: 1, 4.0906413456693205: 1, 4.087571180708
53: 1, 4.086319359590117: 1, 4.085702053218294: 1, 4.0854533822745305: 1, 4.0
85325849980406: 1, 4.084741567849183: 1, 4.082160997998085: 1, 4.079610998470
426: 1, 4.079186251231372: 1, 4.078300367972983: 1, 4.07777853668905: 1, 4.07
2894846073083: 1, 4.070338479334589: 1, 4.066972117254546: 1, 4.0667504517610
7: 1, 4.065894750531521: 1, 4.064296593474717: 1, 4.061835600203901: 1, 4.059
175516635944: 1, 4.058938124695932: 1, 4.058778241223024: 1, 4.0587038009675
3: 1, 4.056378358539442: 1, 4.049106557974373: 1, 4.048818506140668: 1, 4.045
014097095545: 1, 4.043669631632621: 1, 4.041979436744445: 1, 4.04179135444109
5: 1, 4.041105374179748: 1, 4.039879378016334: 1, 4.038183011448725: 1, 4.036
672439258585: 1, 4.036055401154339: 1, 4.034947990226329: 1, 4.03080333347998
6: 1, 4.027791128669757: 1, 4.026732568995539: 1, 4.026375227116831: 1, 4.024
809351312656: 1, 4.022176005415525: 1, 4.0210720969896725: 1, 4.0206659800196
2: 1, 4.020553995109056: 1, 4.019661674496704: 1, 4.018097874673741: 1, 4.016
229737974829: 1, 4.015849822482197: 1, 4.010143258774422: 1, 4.00693526528394
1: 1, 4.00610676303657: 1, 4.006023308266325: 1, 3.9987007842420352: 1, 3.998
665326977374: 1, 3.998378487620518: 1, 3.9945123627939636: 1, 3.9940515992091
26: 1, 3.9897718375446436: 1, 3.9861722587796753: 1, 3.9823933813790235: 1,
3.9822030588921793: 1, 3.981248161101295: 1, 3.9803139622435335: 1, 3.9781889
```

```
458755764: 1, 3.974626788768033: 1, 3.9737087129588406: 1, 3.969381897149635
3: 1, 3.969307084738411: 1, 3.9690265069745885: 1, 3.962771060457636: 1, 3.95
9603467363326: 1, 3.955868226887048: 1, 3.955470561614359: 1, 3.9553612900009
987: 1, 3.9546163345605807: 1, 3.951690626787508: 1, 3.951278626975126: 1, 3.
950908025785735: 1, 3.950845463996645: 1, 3.94785250797679: 1, 3.942357747944
2384: 1, 3.942170081223109: 1, 3.9393735861447237: 1, 3.9379178033391815: 1,
3.934756826586977: 1, 3.9344863262255254: 1, 3.93274527610205: 1, 3.931642288
456989: 1, 3.9261135020132896: 1, 3.924246222130553: 1, 3.923122107674109: 1,
3.920811708082094: 1, 3.9200523730034926: 1, 3.9171139639454555: 1, 3.9170374
73604068: 1, 3.9158561037546784: 1, 3.9140477481925013: 1, 3.913776528476961
7: 1, 3.906821448215659: 1, 3.905090363565722: 1, 3.9050543648745473: 1, 3.90
16863446754155: 1, 3.9015440772585466: 1, 3.892909862562512: 1, 3.88696612747
78498: 1, 3.8864923132062934: 1, 3.8836653536696852: 1, 3.8806198451374194:
1, 3.8746322050372575: 1, 3.8732131645955987: 1, 3.8702231136841774: 1, 3.866
0877286314608: 1, 3.8658990821640282: 1, 3.8619297587103656: 1, 3.86035256581
6864: 1, 3.858695535940779: 1, 3.8572802107730233: 1, 3.8549262410964436: 1,
3.8544422991361436: 1, 3.8533016794889616: 1, 3.8516475651791966: 1, 3.846369
1682670818: 1, 3.8440479389994984: 1, 3.8433829731437315: 1, 3.83758718446652
44: 1, 3.835634719722884: 1, 3.834620437036863: 1, 3.833861282154509: 1, 3.82
72144320517754: 1, 3.824640087083814: 1, 3.8228754600800077: 1, 3.82279253233
40517: 1, 3.822786751629589: 1, 3.82010262748874: 1, 3.8192593688067826: 1,
3.8137546719821627: 1, 3.8075480338421035: 1, 3.8030597813462905: 1, 3.802861
4638710287: 1, 3.802471866960158: 1, 3.802232258086435: 1, 3.800599806283066
6: 1, 3.7987289227494303: 1, 3.7959443039111305: 1, 3.794749625631295: 1, 3.7
93791768973213: 1, 3.793019551091195: 1, 3.792350406136747: 1, 3.786984904908
487: 1, 3.7844344708648925: 1, 3.78221680717954: 1, 3.7822131590657784: 1, 3.
781100199072917: 1, 3.7787532688050276: 1, 3.77443921189116: 1, 3.76841106663
49504: 1, 3.7650550205476407: 1, 3.7621844919247045: 1, 3.761428759323274: 1,
3.7593223155440416: 1, 3.7592211555885755: 1, 3.756624481643063: 1, 3.7552341
861026393: 1, 3.754974867491729: 1, 3.751385597611516: 1, 3.751194939842891:
1, 3.7511279983942076: 1, 3.7510651113376463: 1, 3.749979784806412: 1, 3.7498
93719127949: 1, 3.7438231913071784: 1, 3.7415032289517405: 1, 3.7412646410862
602: 1, 3.7385938659455347: 1, 3.737237164824365: 1, 3.7298330636326966: 1,
3.7288825362088236: 1, 3.7284219703185957: 1, 3.7282575617374083: 1, 3.727870
5847669715: 1, 3.723838719303063: 1, 3.719892835400177: 1, 3.718673360372446
4: 1, 3.7186628238987915: 1, 3.7180883810098804: 1, 3.715854128035774: 1, 3.7
156382382657047: 1, 3.7148105758703176: 1, 3.713491022720891: 1, 3.7123151761
93821: 1, 3.710608092255818: 1, 3.7099118386915224: 1, 3.7072075560610065: 1,
3.7049101568287597: 1, 3.703993778548759: 1, 3.7000253654148505: 1, 3.6981730
783346642: 1, 3.697053299150337: 1, 3.694922035162427: 1, 3.694619106570001:
1, 3.6930326032449305: 1, 3.691039277524557: 1, 3.6888382753627016: 1, 3.6857
94057393566: 1, 3.685049159629853: 1, 3.683287401896022: 1, 3.68171894951416
5: 1, 3.681313572711852: 1, 3.6800548694235062: 1, 3.6800249178898783: 1, 3.6
79634745347264: 1, 3.6788817217545153: 1, 3.677111096683095: 1, 3.67709762052
6659: 1, 3.674905090452072: 1, 3.6748853686410192: 1, 3.66685773982135: 1, 3.
664603269763988: 1, 3.6641960432522467: 1, 3.6640988504836445: 1, 3.662906074
3784447: 1, 3.6627692058295875: 1, 3.6619511465681764: 1, 3.6584304882911813:
1, 3.658297306895233: 1, 3.656346302290703: 1, 3.6563234487907543: 1, 3.64865
93153578566: 1, 3.6482824019107993: 1, 3.645649471748178: 1, 3.64365659653065
24: 1, 3.642148301965277: 1, 3.636865852602486: 1, 3.6360703320594245: 1, 3.6
31748532694951: 1, 3.62756017277359: 1, 3.6257611164535795: 1, 3.618764902768
234: 1, 3.616810301614538: 1, 3.6159203153988257: 1, 3.615625615120139: 1, 3.
615373568109403: 1, 3.611348414525137: 1, 3.611274163928122: 1, 3.61120343462
5296: 1, 3.607244837577702: 1, 3.6062470592530858: 1, 3.6046822588488427: 1,
3.6040013467472267: 1, 3.603511130403283: 1, 3.601480884363173: 1, 3.6014258
651632427: 1, 3.6013974650530396: 1, 3.598818703540057: 1, 3.597842198932131:
1, 3.5961602782886892: 1, 3.595195565312551: 1, 3.5924430720756644: 1, 3.5904
```

```
978413737982: 1, 3.5882490931514623: 1, 3.587540302305619: 1, 3.5867669583594
22: 1, 3.5785722117992775: 1, 3.5748234906154446: 1, 3.574549743519945: 1, 3.
573981561308301: 1, 3.566806955058511: 1, 3.56657278749464: 1, 3.564607057726
1537: 1, 3.5637681324675454: 1, 3.5635906550320824: 1, 3.558260635651991: 1,
3.5552162218186796: 1, 3.5550485437207784: 1, 3.5523784134225243: 1, 3.548205
1108618826: 1, 3.546271376024326: 1, 3.5435464141329747: 1, 3.542001078047879
7: 1, 3.5409409144296204: 1, 3.53965022679303: 1, 3.5380270308725486: 1, 3.53
7341446986883: 1, 3.5371714281496107: 1, 3.536506542575079: 1, 3.535586263707
5353: 1, 3.5333993517714557: 1, 3.5322592063369065: 1, 3.53066254471795: 1,
3.5270426895167093: 1, 3.5265001715073256: 1, 3.5260889014005636: 1, 3.525843
195811141: 1, 3.525164878816009: 1, 3.52293754696992: 1, 3.520131346547001:
1, 3.519429992752532: 1, 3.517319293883311: 1, 3.515463218414912: 1, 3.510918
070530162: 1, 3.5093687239798745: 1, 3.5092588392161637: 1, 3.508070382646068
6: 1, 3.5080038711912773: 1, 3.5073427544141547: 1, 3.5060896424888393: 1, 3.
5060795726731975: 1, 3.5046347220285066: 1, 3.502704578884599: 1, 3.501774172
1414: 1, 3.501105697740837: 1, 3.4978653775130883: 1, 3.4948083062013318: 1,
3.4933949499040575: 1, 3.4909970831677177: 1, 3.4902785549092608: 1, 3.482960
3502360413: 1, 3.480239620742044: 1, 3.4793435429979946: 1, 3.476611361674562
4: 1, 3.4757860457762058: 1, 3.4718375440212115: 1, 3.471790205405706: 1, 3.4
70894582132835: 1, 3.4702591778257523: 1, 3.4667315276404396: 1, 3.4649434686
29618: 1, 3.463642733578678: 1, 3.4619738592275118: 1, 3.4618035134360174: 1,
3.4606419675673585: 1, 3.460254721741273: 1, 3.46012776233547: 1, 3.459339726
9886843: 1, 3.4591245717558734: 1, 3.455770431742814: 1, 3.4539183094621144:
1, 3.4528635569029893: 1, 3.4527142711182415: 1, 3.4526908200942783: 1, 3.452
2030365181853: 1, 3.4519081657190878: 1, 3.449842380435351: 1, 3.449356329458
241: 1, 3.448411596789727: 1, 3.4465381872371696: 1, 3.4440826856852973: 1,
3.441762753257424: 1, 3.44058530112013: 1, 3.440270077925744: 1, 3.4393745187
245477: 1, 3.4387245722019086: 1, 3.4369173334044034: 1, 3.432662774363207:
1, 3.4320426023294424: 1, 3.430462570333492: 1, 3.4293700133203235: 1, 3.4256
63733216876: 1, 3.4252286430547336: 1, 3.4224630256346007: 1, 3.4221574023273
71: 1, 3.4221529479061616: 1, 3.4197365087409826: 1, 3.415870533040233: 1, 3.
414282261552821: 1, 3.4104146581530017: 1, 3.405703004781899: 1, 3.4037731950
209964: 1, 3.40359655451417: 1, 3.401226511471915: 1, 3.39902482452444: 1, 3.
3988975747268997: 1, 3.3980275106056936: 1, 3.396993587833546: 1, 3.396877720
1468754: 1, 3.3945275193858673: 1, 3.3942172906911914: 1, 3.3922947548565436:
1, 3.39058287500259: 1, 3.3891589384555036: 1, 3.3875877329980506: 1, 3.38690
7731657596: 1, 3.3866477705779348: 1, 3.3828350752399716: 1, 3.38170325528811
37: 1, 3.373495349584724: 1, 3.37324684415292: 1, 3.3732407555247743: 1, 3.37
2058998878573: 1, 3.3719970485782866: 1, 3.369950955940868: 1, 3.367568342843
2266: 1, 3.3647994448180816: 1, 3.3596823767237898: 1, 3.358586662710163: 1,
3.3565814025503564: 1, 3.351620828116798: 1, 3.3513409118734696: 1, 3.3500771
12705513: 1, 3.3495382909778373: 1, 3.349142694059623: 1, 3.3476965352316714:
1, 3.346961104314669: 1, 3.3467407335425072: 1, 3.346450202086028: 1, 3.34580
40224526164: 1, 3.3453852659603673: 1, 3.343134429298677: 1, 3.33862766393500
54: 1, 3.332087924203746: 1, 3.3310843869989895: 1, 3.323633333089102: 1, 3.3
235238521449357: 1, 3.32321034686937: 1, 3.3213846511052263: 1, 3.32072536610
1866: 1, 3.320568292026711: 1, 3.3190617657557113: 1, 3.315703052737846: 1,
3.315502991656965: 1, 3.3092793923428148: 1, 3.309266203434792: 1, 3.30916323
38507884: 1, 3.30639698564936: 1, 3.3060559781074925: 1, 3.306047519197358:
1, 3.303657584502134: 1, 3.3034927922249553: 1, 3.30197571565142: 1, 3.301393
3151391237: 1, 3.299856397700352: 1, 3.299304236365207: 1, 3.299300110738183
6: 1, 3.295103179665053: 1, 3.2915734799888416: 1, 3.2903670072063447: 1, 3.2
90051887185097: 1, 3.2867795763039482: 1, 3.286397208364785: 1, 3.27944840071
79503: 1, 3.275214248313328: 1, 3.2747632596180534: 1, 3.269944199710947: 1,
3.2687728117993133: 1, 3.2638490926282513: 1, 3.2638269192386677: 1, 3.263538
5346271706: 1, 3.2611501971839227: 1, 3.2602973955609302: 1, 3.25906257137900
33: 1, 3.2573763404386415: 1, 3.256646541586142: 1, 3.2522731068545827: 1, 3.
```

```
2507603713237736: 1, 3.243020054976453: 1, 3.240734765327281: 1, 3.2391135936
154773: 1, 3.2383368086692053: 1, 3.237471768030247: 1, 3.2355740586182966:
1, 3.233314361396216: 1, 3.2318921108416205: 1, 3.2294843076922146: 1, 3.2285
186266247097: 1, 3.227979718072568: 1, 3.2253845275730826: 1, 3.2242590306043
47: 1, 3.216792511072312: 1, 3.2147126234208643: 1, 3.213662817462621: 1, 3.2
127657158047422: 1, 3.211268212393672: 1, 3.2090930373130075: 1, 3.2085941427
810605: 1, 3.2080512192852946: 1, 3.207215991122145: 1, 3.206586798234558: 1,
3.2059643204853043: 1, 3.2050032251655414: 1, 3.203408983002889: 1, 3.1989615
20591315: 1, 3.1963479243739643: 1, 3.1931675853452273: 1, 3.192299991472078
4: 1, 3.189767386312025: 1, 3.189667334430986: 1, 3.188552032389875: 1, 3.188
0516060947697: 1, 3.1872857610766836: 1, 3.1864831392733426: 1, 3.18425224602
95286: 1, 3.184052109437738: 1, 3.1835035393886355: 1, 3.1828680661014617: 1,
3.1816917273002647: 1, 3.1804721454306417: 1, 3.1802678288712807: 1, 3.179900
6245043033: 1, 3.179073597931619: 1, 3.176532729088297: 1, 3.173326431462191:
1, 3.170679392660777: 1, 3.169013423662789: 1, 3.164920433924728: 1, 3.163545
4658074917: 1, 3.1629355462682907: 1, 3.1620816244925742: 1, 3.15728265280414
58: 1, 3.1569599397642465: 1, 3.154743475517321: 1, 3.152004929838707: 1, 3.1
510991409010267: 1, 3.1502410672862373: 1, 3.1487371492956227: 1, 3.148689759
1316305: 1, 3.1443523321457745: 1, 3.1414157744237503: 1, 3.1394780096463424:
1, 3.136691169664281: 1, 3.1363047747382944: 1, 3.135121408405816: 1, 3.13398
19999542953: 1, 3.131853212956354: 1, 3.1291437029519176: 1, 3.12759709242146
84: 1, 3.1248026907705397: 1, 3.1227417473620416: 1, 3.119700548594342: 1, 3.
1186714576268804: 1, 3.1176475535371835: 1, 3.1171322313934744: 1, 3.11584098
49252453: 1, 3.1151610936404412: 1, 3.1142465033681845: 1, 3.111174298276123
5: 1, 3.1105385796201035: 1, 3.1100731245608664: 1, 3.1067896546208336: 1, 3.
104337483732121: 1, 3.1017019810570914: 1, 3.1015910162799063: 1, 3.100608512
8497247: 1, 3.1003536612471883: 1, 3.0987924868763934: 1, 3.097270577976514:
1, 3.09593039215193: 1, 3.0909883782868217: 1, 3.0893030476905246: 1, 3.08913
9892555264: 1, 3.0877044672894947: 1, 3.086060974290635: 1, 3.085807163403102
6: 1, 3.0839315468456863: 1, 3.0837294233164627: 1, 3.0787521417707056: 1, 3.
0782509214485416: 1, 3.077482961310642: 1, 3.073906927194865: 1, 3.0725419157
66331: 1, 3.072292595705374: 1, 3.069360016391759: 1, 3.0667313426104252: 1,
3.0632518711521466: 1, 3.0593204881722893: 1, 3.057398530952036: 1, 3.0520869
723216477: 1, 3.05197091865621: 1, 3.0501173725195283: 1, 3.049696350721832:
1, 3.0475662398060632: 1, 3.046898324091203: 1, 3.0460252521651094: 1, 3.0440
462940619817: 1, 3.0427411348364886: 1, 3.04077763417554: 1, 3.03937384913007
7: 1, 3.0360643455846907: 1, 3.035743479814871: 1, 3.034868811964755: 1, 3.03
26757266970796: 1, 3.0320885578948924: 1, 3.030294370142614: 1, 3.02955842413
89366: 1, 3.028708748088951: 1, 3.0280846193206874: 1, 3.027854941806662: 1,
3.0276423551952156: 1, 3.0229324180048094: 1, 3.0164615115987226: 1, 3.015366
907056022: 1, 3.014286198598259: 1, 3.013666567042025: 1, 3.009999174167348:
1, 3.009120249664613: 1, 3.0077269495772057: 1, 3.007484415219563: 1, 3.00734
8730377347: 1, 3.0060285344585482: 1, 3.00398453448953: 1, 2.998846169040937
6: 1, 2.9967841395238737: 1, 2.9949407709076286: 1, 2.993243266674515: 1, 2.9
928309639430597: 1, 2.9902407391761185: 1, 2.9901229381438212: 1, 2.990075609
3914715: 1, 2.989803244022582: 1, 2.9888377424417962: 1, 2.987359675668571:
1, 2.985563687689467: 1, 2.9853828335064856: 1, 2.9845293849726278: 1, 2.9819
883615744462: 1, 2.9818193892465716: 1, 2.9802611193921367: 1, 2.974241690663
377: 1, 2.971353621893581: 1, 2.9712589632673425: 1, 2.970634425289743: 1, 2.
969415785433234: 1, 2.969220774662364: 1, 2.968670124120835: 1, 2.96786391120
1126: 1, 2.9668389736819205: 1, 2.9662439774523044: 1, 2.9644671243994813: 1,
2.962615481804976: 1, 2.9603704777037447: 1, 2.960109499316348: 1, 2.95849705
07691564: 1, 2.957889551506404: 1, 2.9551844330759205: 1, 2.954685546459267:
1, 2.954410507320464: 1, 2.9535105571890297: 1, 2.9519852254022387: 1, 2.9505
793883585456: 1, 2.950144431468807: 1, 2.945737511174325: 1, 2.9455982854048
8: 1, 2.942084524275321: 1, 2.940854105999663: 1, 2.9399708140223444: 1, 2.93
9086092466388: 1, 2.937596856042709: 1, 2.937094135956869: 1, 2.9362597556463
```

```
057: 1, 2.9340767379194017: 1, 2.9340400450513715: 1, 2.9336669324466684: 1,
2.933371989026639: 1, 2.9331882083587386: 1, 2.9329359830233734: 1, 2.9323977
968374146: 1, 2.931652853960488: 1, 2.926611526819323: 1, 2.9256798321314683:
1, 2.925229060566934: 1, 2.924992110587143: 1, 2.9233260198437643: 1, 2.92301
4243373859: 1, 2.9215383368794114: 1, 2.92126849632764: 1, 2.920737379189187
5: 1, 2.916057717489083: 1, 2.914394728520284: 1, 2.9142742024492976: 1, 2.90
76236101529926: 1, 2.9074907354412503: 1, 2.9060266156040337: 1, 2.9048319244
52731: 1, 2.904479564102875: 1, 2.8971678378666406: 1, 2.891779620550435: 1,
2.8891645728403397: 1, 2.8885170357466237: 1, 2.8882358490562843: 1, 2.887639
1291882815: 1, 2.887588254081636: 1, 2.887202003495418: 1, 2.886032628054872:
1, 2.8812826021763205: 1, 2.8796205020116643: 1, 2.8792937678873596: 1, 2.878
6892502153694: 1, 2.878032990079043: 1, 2.877425878030673: 1, 2.8759452272966
506: 1, 2.873129728663176: 1, 2.8730974153573223: 1, 2.8721550285526427: 1,
2.871637182011215: 1, 2.8669553862564556: 1, 2.8636555249226294: 1, 2.8622637
83558363: 1, 2.860122769198284: 1, 2.8593080783421274: 1, 2.8579308711969786:
1, 2.8558291669680007: 1, 2.855256462875683: 1, 2.8514989065378358: 1, 2.8478
507696449498: 1, 2.8431311689130854: 1, 2.840793028975837: 1, 2.840660989947
9: 1, 2.839894338767075: 1, 2.839713299437253: 1, 2.8397101567584184: 1, 2.83
96602795763326: 1, 2.8388109785354434: 1, 2.838235007862375: 1, 2.83793382932
66647: 1, 2.830115928609448: 1, 2.8266347945352845: 1, 2.8262330420571335: 1,
2.825878751665906: 1, 2.8236373434968467: 1, 2.823524417246526: 1, 2.82262385
03450456: 1, 2.8205624327222933: 1, 2.8202381858653665: 1, 2.819468312330465:
1, 2.819043574651522: 1, 2.8150923410319426: 1, 2.813796649047008: 1, 2.81341
94787086346: 1, 2.811162555721237: 1, 2.808793065755371: 1, 2.80770340085797
6: 1, 2.8071535259034404: 1, 2.8060723521443474: 1, 2.7999805530822006: 1, 2.
7987025090810054: 1, 2.7981157009388444: 1, 2.797210415719662: 1, 2.795169807
8386866: 1, 2.7941012761846986: 1, 2.7917534336574494: 1, 2.7912548155414076:
1, 2.7888242495084863: 1, 2.787843997721283: 1, 2.7871166061864545: 1, 2.7857
725382026874: 1, 2.7848789479932283: 1, 2.7812244567029794: 1, 2.780664494434
3764: 1, 2.778047628341239: 1, 2.7766288453816053: 1, 2.7761794202705037: 1,
2.7726553974790447: 1, 2.7717995670736735: 1, 2.771429100173963: 1, 2.7704929
84678984: 1, 2.7703667155453893: 1, 2.7695679108325124: 1, 2.769561906891298
4: 1, 2.7666134894253367: 1, 2.764575755741644: 1, 2.7644530550949593: 1, 2.7
626918843980834: 1, 2.761947536475873: 1, 2.759395540249808: 1, 2.75925641224
50236: 1, 2.7579436014644707: 1, 2.757624088959089: 1, 2.757447935768877: 1,
2.757111608656866: 1, 2.756130048338187: 1, 2.7534509584158964: 1, 2.75214845
78043434: 1, 2.751645223298127: 1, 2.748828033304803: 1, 2.748583203077956:
1, 2.7481943568838996: 1, 2.7448091478331054: 1, 2.7433357864839856: 1, 2.741
089604759948: 1, 2.740031821796593: 1, 2.7394401808783337: 1, 2.7390482902859
548: 1, 2.736965477267326: 1, 2.736393984886058: 1, 2.735691748858124: 1, 2.7
346702381155037: 1, 2.734021825761068: 1, 2.7338101782841733: 1, 2.7305763889
767856: 1, 2.725034564191521: 1, 2.7237962959189286: 1, 2.7202601651538307:
1, 2.719219281994074: 1, 2.7171033168600873: 1, 2.716947985656484: 1, 2.71667
49489991915: 1, 2.7163723877992996: 1, 2.715513961698774: 1, 2.71506175873888
7: 1, 2.7150196168217255: 1, 2.714087501897477: 1, 2.712219172816906: 1, 2.71
16374328844417: 1, 2.7098250471929433: 1, 2.70819069647033: 1, 2.707090362818
828: 1, 2.7066069149971277: 1, 2.7055481393363383: 1, 2.70511035825843: 1, 2.
703529370065931: 1, 2.700786819019665: 1, 2.699898076421519: 1, 2.69442038802
7162: 1, 2.6918985301091807: 1, 2.685471634546622: 1, 2.684795249205513: 1,
2.679350325213471: 1, 2.6782739842637837: 1, 2.675438175373467: 1, 2.67423087
5595458: 1, 2.667857434934005: 1, 2.6677807155363222: 1, 2.6676550575857045:
1, 2.667479907645604: 1, 2.6660449170571545: 1, 2.665480729775682: 1, 2.66488
49675525614: 1, 2.6648437784853916: 1, 2.6647042949285167: 1, 2.6617712566857
44: 1, 2.657752025130667: 1, 2.656794059336451: 1, 2.652852739204001: 1, 2.65
19645868369506: 1, 2.6511742770286952: 1, 2.6476437429888606: 1, 2.6476417754
00457: 1, 2.6432950451536654: 1, 2.641351778067206: 1, 2.641181308435179: 1,
2.6408419894231803: 1, 2.640239968097961: 1, 2.6398991096216684: 1, 2.6380850
```

```
276009222: 1, 2.634726441002518: 1, 2.6333490759375744: 1, 2.632261013135168
7: 1, 2.62807746666392: 1, 2.6217810637728505: 1, 2.6196337045580815: 1, 2.61
9617575837605: 1, 2.6194904417981757: 1, 2.619036371500831: 1, 2.618273009408
3984: 1, 2.6172056191878483: 1, 2.6168981530950117: 1, 2.6163427416502154: 1,
2.615657171991476: 1, 2.6141483772608796: 1, 2.613926669049378: 1, 2.61290500
36489843: 1, 2.6100086055300618: 1, 2.6091774261283454: 1, 2.607996863194966:
1, 2.6062857618330626: 1, 2.6044697530456102: 1, 2.603827337682411: 1, 2.6012
56110565748: 1, 2.601012055695202: 1, 2.600132133203487: 1, 2.598126585706028
3: 1, 2.5969565661337133: 1, 2.595340556665602: 1, 2.59279450374548: 1, 2.590
61472846246: 1, 2.587623026285294: 1, 2.587558293140676: 1, 2.58741374011548
8: 1, 2.5823122028751335: 1, 2.5819591033885527: 1, 2.57896002549166: 1, 2.57
79820230632433: 1, 2.5768772115689527: 1, 2.5750556020964384: 1, 2.5718007650
83635: 1, 2.5697435444084995: 1, 2.5688115696942515: 1, 2.5683398441967324:
1, 2.567808001200652: 1, 2.5664364914159155: 1, 2.560902319207104: 1, 2.56077
70797098337: 1, 2.55946381498254: 1, 2.5593697509966264: 1, 2.557862565665057
5: 1, 2.557095252881177: 1, 2.5565660091899662: 1, 2.5548550894642017: 1, 2.5
548452869081677: 1, 2.5500659681053572: 1, 2.548798553113119: 1, 2.5476156268
298618: 1, 2.5472035747293074: 1, 2.546616146301687: 1, 2.5457850615459745:
1, 2.54394667328417: 1, 2.5432570278577975: 1, 2.542951568547263: 1, 2.542240
800209723: 1, 2.541870956660213: 1, 2.540629850774468: 1, 2.538100060073469:
1, 2.537493894750212: 1, 2.5320520289290323: 1, 2.5307415232728685: 1, 2.5297
74341566591: 1, 2.526975118069478: 1, 2.5265586698293228: 1, 2.52645764195701
53: 1, 2.526426476072916: 1, 2.525069693550488: 1, 2.524275912467874: 1, 2.52
41339687016846: 1, 2.5223700049212963: 1, 2.5211207878096262: 1, 2.5210758399
73921: 1, 2.5210126844054517: 1, 2.5199821030607534: 1, 2.5179567135445806:
1, 2.5151671186862554: 1, 2.513495984353533: 1, 2.5130618680609893: 1, 2.5106
534249066654: 1, 2.5075507565115345: 1, 2.507201208554439: 1, 2.5035577451334
07: 1, 2.5034153628562903: 1, 2.5029022311126856: 1, 2.5028614037539376: 1,
2.501933136472444: 1, 2.4982833211322957: 1, 2.497616876794362: 1, 2.49664119
16272278: 1, 2.494848280806039: 1, 2.4900352917362376: 1, 2.4893830418049316:
1, 2.485635046594663: 1, 2.485482356111894: 1, 2.4823206128754944: 1, 2.48193
1745839503: 1, 2.4794680775350475: 1, 2.4787292806205223: 1, 2.47690352056280
4: 1, 2.476309506961825: 1, 2.475969323424238: 1, 2.4744315434802666: 1, 2.47
3307552590654: 1, 2.4728934013001425: 1, 2.4718778886352415: 1, 2.47134021951
4686: 1, 2.466665776525199: 1, 2.465016005673193: 1, 2.4650027724450263: 1,
2.463367651882188: 1, 2.4609844886666505: 1, 2.459127728647688: 1, 2.45888464
5833307: 1, 2.4575741959560244: 1, 2.4565616363386353: 1, 2.45360543993456:
1, 2.453163556056039: 1, 2.4526582996695616: 1, 2.452600961377953: 1, 2.45221
9998796772: 1, 2.4516405845438687: 1, 2.451051591129413: 1, 2.44839692730360
1: 1, 2.4480512652354087: 1, 2.447287666789577: 1, 2.4470352893102425: 1, 2.4
45983561925909: 1, 2.4445719736279887: 1, 2.4426052443119954: 1, 2.4399307845
964446: 1, 2.4387704098972587: 1, 2.43775655634505: 1, 2.437260780388144: 1,
2.436919874121273: 1, 2.4356698506825554: 1, 2.435007826357198: 1, 2.43429086
2052174: 1, 2.434288627569871: 1, 2.4330197442436985: 1, 2.4314660723908927:
1, 2.4307287461833935: 1, 2.4279568003975753: 1, 2.4239339394143884: 1, 2.423
2097588894494: 1, 2.4219773548330887: 1, 2.4218202005398664: 1, 2.42047604622
8221: 1, 2.419213857583288: 1, 2.418781849596723: 1, 2.4185015467161564: 1,
2.4162825958115786: 1, 2.416039412681658: 1, 2.4133762932820124: 1, 2.4132635
20051742: 1, 2.4131606322957295: 1, 2.4119270744554555: 1, 2.410765591963535
6: 1, 2.409418865822183: 1, 2.405817156723289: 1, 2.405470998051154: 1, 2.403
6552759272074: 1, 2.400376900083287: 1, 2.4001148873231952: 1, 2.399737172525
645: 1, 2.399510197401826: 1, 2.3968048671110878: 1, 2.396469124396718: 1, 2.
3952031903906756: 1, 2.395108780217643: 1, 2.393634469515029: 1, 2.3929899978
72574: 1, 2.392349724864673: 1, 2.392095596028094: 1, 2.3895172710199226: 1,
2.387911413355165: 1, 2.387869321926015: 1, 2.3854837552138553: 1, 2.38031770
31322704: 1, 2.3789497684971805: 1, 2.378674013277387: 1, 2.376950336225056:
1, 2.3752386363616855: 1, 2.373126194649974: 1, 2.372916593109907: 1, 2.37242
```

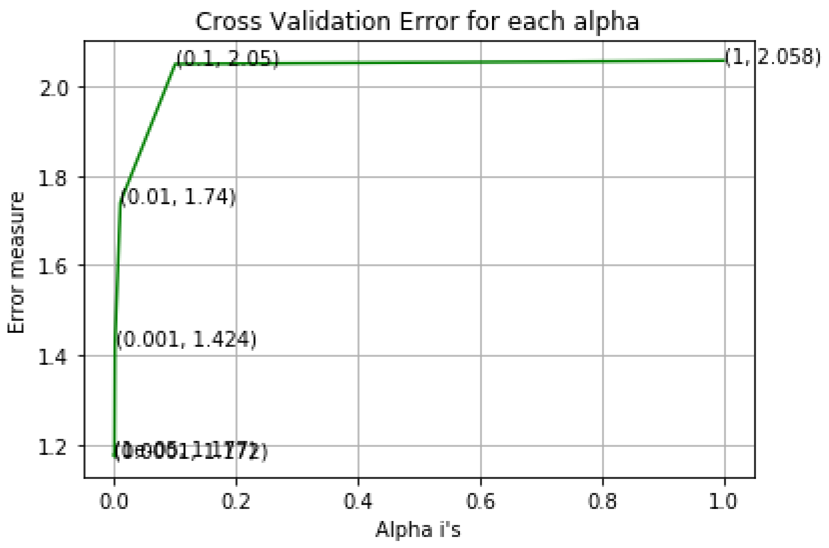
```
6053679023: 1, 2.3713037436749906: 1, 2.3677761202498337: 1, 2.36658593124413
44: 1, 2.36598749066205: 1, 2.3646140599547523: 1, 2.3633715951307313: 1, 2.3
631960021285514: 1, 2.362654355948742: 1, 2.36247498796904: 1, 2.362206361163
8646: 1, 2.3618664930198543: 1, 2.359597911324222: 1, 2.356838476581796: 1,
2.356363595341795: 1, 2.3544889813019654: 1, 2.3543111029576926: 1, 2.3472068
583679313: 1, 2.345615031017461: 1, 2.3455083682351536: 1, 2.345138771002341:
1, 2.3445668982697243: 1, 2.342928616898407: 1, 2.3402288508798166: 1, 2.3356
47961140897: 1, 2.333919296089879: 1, 2.3337595636306334: 1, 2.3326476550844
6: 1, 2.3320623078222034: 1, 2.3296964982995183: 1, 2.3283438908811096: 1, 2.
327769404584003: 1, 2.324486099957632: 1, 2.323382735330806: 1, 2.32298850599
96542: 1, 2.3213003856131427: 1, 2.320556124688797: 1, 2.317567944004838: 1,
2.3150107400414743: 1, 2.3146159338865746: 1, 2.314435151596901: 1, 2.3139139
811172975: 1, 2.3102424639333656: 1, 2.310110334767819: 1, 2.309745001074183:
1, 2.3093693093481886: 1, 2.3089536464215685: 1, 2.3060187607408564: 1, 2.305
2736220065504: 1, 2.30480499498071: 1, 2.302839298916829: 1, 2.30149241767834
36: 1, 2.294003590608884: 1, 2.293437755869696: 1, 2.291892140304166: 1, 2.29
00321566488926: 1, 2.2898785363168983: 1, 2.289304168957208: 1, 2.28764635419
2466: 1, 2.2856722584083697: 1, 2.284977374980259: 1, 2.2846284229776272: 1,
2.283444704437413: 1, 2.281849925580031: 1, 2.280283117039155: 1, 2.272469555
5630277: 1, 2.272062523446709: 1, 2.2687175398898347: 1, 2.2649384355732964:
1, 2.261528770514529: 1, 2.2598439789633242: 1, 2.257943302663145: 1, 2.25702
50598290427: 1, 2.2548919683798245: 1, 2.250836212504296: 1, 2.25082707931438
8: 1, 2.250668851026069: 1, 2.2480156025872584: 1, 2.2460488971971393: 1, 2.2
433326755140564: 1, 2.2409945689689397: 1, 2.2401941631791304: 1, 2.240182847
5270946: 1, 2.239212591814302: 1, 2.2381739850944946: 1, 2.2381625192258587:
1, 2.2368755378575518: 1, 2.235444992502204: 1, 2.233799087762679: 1, 2.22854
86213855554: 1, 2.2281421910720045: 1, 2.2277255507591005: 1, 2.2272334895120
287: 1, 2.2261837245367624: 1, 2.22461647435861: 1, 2.2243611638792675: 1, 2.
222238739731972: 1, 2.21963505753031: 1, 2.219605867180967: 1, 2.216077727344
2425: 1, 2.2120709739354183: 1, 2.210654630630541: 1, 2.210267994650048: 1,
2.209461643510598: 1, 2.208691189963: 1, 2.208360815242066: 1, 2.208258038532
747: 1, 2.207392755061802: 1, 2.204371923416117: 1, 2.201731195933883: 1, 2.2
017253821023646: 1, 2.200566024826006: 1, 2.200449054094146: 1, 2.19808979249
2036: 1, 2.196590333547758: 1, 2.192191141199017: 1, 2.189817824103931: 1, 2.
187627270928355: 1, 2.184772089268833: 1, 2.1839492966692773: 1, 2.1816653506
35501: 1, 2.1786951295898827: 1, 2.1772846142771694: 1, 2.1737954722987514:
1, 2.170089395704325: 1, 2.169978441511822: 1, 2.1690372462253262: 1, 2.16665
86750412415: 1, 2.1641412493790617: 1, 2.163611773889955: 1, 2.16232355905495
37: 1, 2.1620618075030897: 1, 2.1522318427676903: 1, 2.150859912160984: 1, 2.
148306100187284: 1, 2.1457172512470777: 1, 2.1404092791540976: 1, 2.140197359
4564137: 1, 2.1396428649311483: 1, 2.1393767727430584: 1, 2.1385323735508366:
1, 2.137433401443583: 1, 2.1366847884962263: 1, 2.1341812424631823: 1, 2.1336
213138162425: 1, 2.1284359707481477: 1, 2.1260978339131777: 1, 2.125025729524
54: 1, 2.1247281713458817: 1, 2.1220565914362886: 1, 2.1212574678709974: 1,
2.1211589732431633: 1, 2.1203835725414404: 1, 2.1196741285915803: 1, 2.119233
574944751: 1, 2.117129777244511: 1, 2.1166073506405314: 1, 2.115959632082641:
1, 2.1089783628691494: 1, 2.1062933637418046: 1, 2.106114389502829: 1, 2.1003
966636194766: 1, 2.0997335490567166: 1, 2.0990724933558806: 1, 2.089012249626
9794: 1, 2.086558406235426: 1, 2.0854949441739428: 1, 2.083032380726135: 1,
2.080822390644407: 1, 2.0792319107315893: 1, 2.0784707334556924: 1, 2.0774644
18190011: 1, 2.072633357740014: 1, 2.0714418683417004: 1, 2.0684594159613128:
1, 2.053529441696101: 1, 2.0457268348301447: 1, 2.0417738812291475: 1, 2.0411
53129604513: 1, 2.0410173551142314: 1, 2.0408934711664126: 1, 2.0408803192869
014: 1, 2.040841284268154: 1, 2.0389804740642585: 1, 2.0354905717365095: 1,
 2.0352313032609044: 1, 2.03122570328685: 1, 2.0247752581293343: 1, 2.0224188
63402552: 1, 2.022203998132849: 1, 2.0215340911528497: 1, 2.0133647839557756:
1, 2.0132502297925585: 1, 2.013128500301446: 1, 2.0123043418854896: 1, 2.0088
```

```
98323166728: 1, 2.0073925966659654: 1, 1.9939408925501025: 1, 1.9891994658748
817: 1, 1.9874570796107143: 1, 1.9858313968040349: 1, 1.9838609510747374: 1,
 1.9832756406804855: 1, 1.9820186215857074: 1, 1.9817988456458813: 1, 1.97201
85565009511: 1, 1.9712564701580726: 1, 1.970837733349584: 1, 1.96985452119162
1: 1, 1.9623579524081578: 1, 1.9603214146229953: 1, 1.9600082710647793: 1, 1.
9596199584915057: 1, 1.957040826359256: 1, 1.9503400753609499: 1, 1.949219317
24851: 1, 1.9482294399994653: 1, 1.9450926728121183: 1, 1.936452841194715: 1,
1.9310519926612: 1, 1.9299140654313762: 1, 1.9276482451878758: 1, 1.922687712
126038: 1, 1.9218977710737408: 1, 1.9092700755453378: 1, 1.903167042117796:
 1, 1.9022386982484216: 1, 1.8988855809123546: 1, 1.8939983774394789: 1, 1.88
92428609627265: 1, 1.8862503360627643: 1, 1.8857954153704994: 1, 1.8692370128
152802: 1, 1.8564897178711741: 1, 1.8553226022180658: 1, 1.8225693481602918:
 1, 1.8160734064926318: 1, 1.7980075715042103: 1, 1.7956231021204738: 1, 1.79
37441601385933: 1, 1.7861533009122217: 1, 1.7798096075942662: 1, 1.7752807162
111215: 1, 1.7740353552439494: 1, 1.7672727984958074: 1, 1.7378972804613269:
 1, 1.689913353837599: 1})
```

In [48]:
```
# 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/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, …])     Fit linear model with Stochast
ic 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_, e
ps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predi
ct_y, labels=clf.classes_, eps=1e-15))

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


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

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
```

```
print('For values of best alpha = ', alpha[best_alpha], "The train log loss i
s:",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_loss(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 i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.1766143372443885
For values of alpha =  0.0001 The log loss is: 1.1722123562885514
For values of alpha =  0.001 The log loss is: 1.4235495836242769
For values of alpha =  0.01 The log loss is: 1.740338484095111 5
For values of alpha =  0.1 The log loss is: 2.0500092667630945
For values of alpha =  1 The log loss is: 2.0578269884956337
```



```
For values of best alpha =  0.0001 The train log loss is: 0.6423746250903991
For values of best alpha =  0.0001 The cross validation log loss is: 1.172212
3562885514
For values of best alpha =  0.0001 The test log loss is: 1.1586735910532906
```

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

Ans. Yes, it seems like!

In [49]:
```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=5, max_features=3000)
    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 [50]:
```
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")
```

```
93.7 % of word of test data appeared in train data
92.733 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [51]:
```
#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 bel
ongs 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 [52]:
```
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 [53]:
```
# 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 [{}]".f
ormat(word,yes_no))
        elif (v < fea1_len+fea2_len):
            word = var_vec.get_feature_names()[v-(fea1_len)]
            yes_no = True if word == var else False
            if yes_no:
                word_present += 1
                print(i, "variation feature [{}] present in test data point [
{}]".format(word,yes_no))
        else:
            word = text_vec.get_feature_names()[v-(fea1_len+fea2_len)]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
                print(i, "Text feature [{}] present in test data point [{}]".f
ormat(word,yes_no))

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

Stacking the three types of features

In [54]:
```
# 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_va
riation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_varia
tion_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_f
eature_onehotCoding))

train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature
_onehotCoding)).tocsr()
train_y = np.array(list(train_df['Class']))

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

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


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,t
rain_variation_feature_responseCoding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,tes
t_variation_feature_responseCoding))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_vari
ation_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_fea
ture_responseCoding))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_r
esponseCoding))
```

In [55]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x
_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_o
nehotCoding.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, 5197)
(number of data points * number of features) in test data =  (665, 5197)
(number of data points * number of features) in cross validation data = (532,
5197)
```

In [56]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x
_responseCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_r
esponseCoding.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)
```

Feature Engineering on one-hot encoded features

In [57]:
```
train_x_onehotCodingFE=np.sqrt(train_x_onehotCoding)
test_x_onehotCodingFE=np.sqrt(test_x_onehotCoding)
cv_x_onehotCodingFE=np.sqrt(cv_x_onehotCoding)
```

4.1. Base Line Model

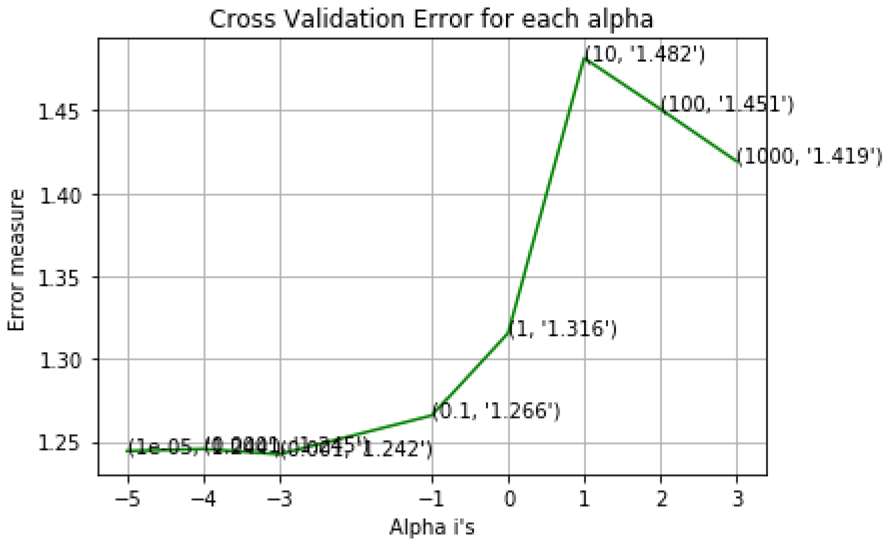
4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [58]:
```
# find more about Multinomial Naive base function here http://scikit-learn.or
g/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=Non
e)

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


alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes
_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probab
ility 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 i
s:",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_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-05
Log Loss : 1.2442427651450625
for alpha = 0.0001
Log Loss : 1.2454826518681834
for alpha = 0.001
Log Loss : 1.2423007584053438
for alpha = 0.1
Log Loss : 1.2656989120604816
for alpha = 1
Log Loss : 1.3155405657141117
for alpha = 10
Log Loss : 1.4817120025323702
for alpha = 100
Log Loss : 1.4509086811280916
for alpha = 1000
Log Loss : 1.4194057044076456
```

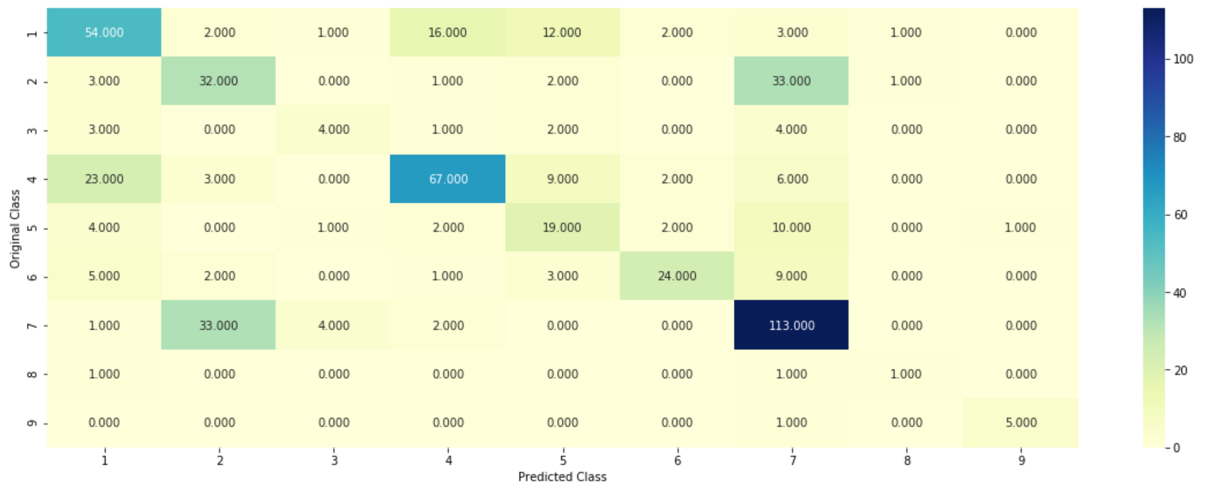


```
For values of best alpha =  0.001 The train log loss is: 0.5827010882734858
For values of best alpha =  0.001 The cross validation log loss is: 1.2423007
584053438
For values of best alpha =  0.001 The test log loss is: 1.2647146781626806
```

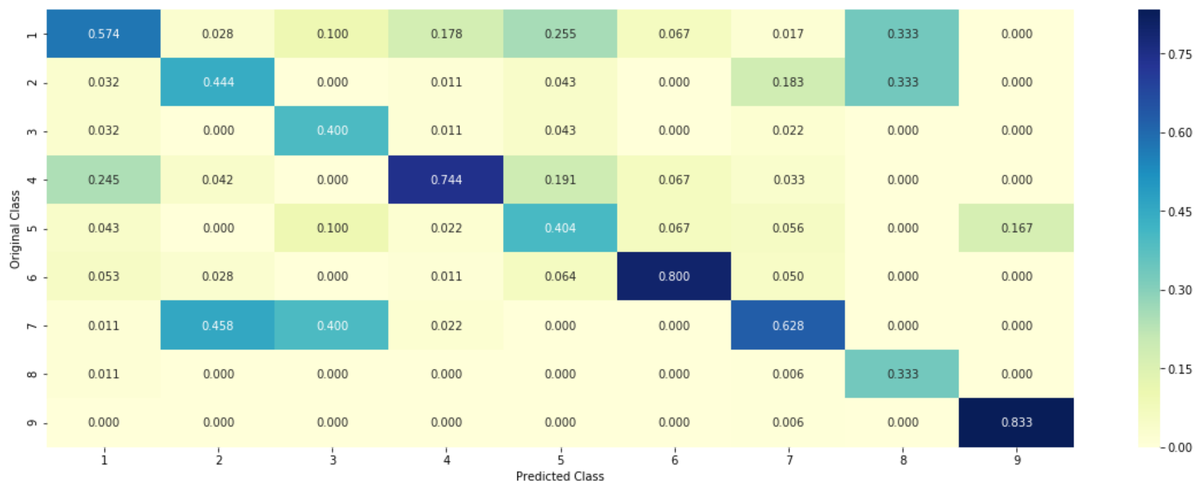
4.1.1.2. Testing the model with best hyper paramters

In [59]:
```
# find more about Multinomial Naive base function here http://scikit-learn.or
g/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=Non
e)

# 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/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', 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-probabilit
y 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()))
```

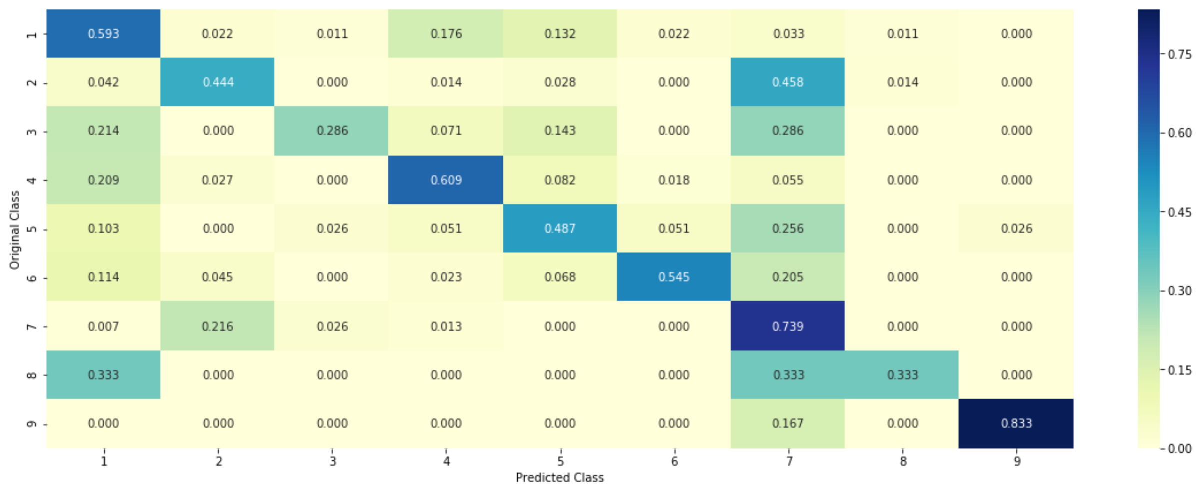
```
Log Loss : 1.2423007584053438
Number of missclassified point : 0.40037593984962405
-------------------- Confusion matrix --------------------
```



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



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



4.1.1.3. Feature Importance, Correctly classified point

In [60]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices=np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0811 0.0627 0.0133 0.0758 0.0384 0.0401 0.
679  0.0055 0.0042]]
Actual Class : 7
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [61]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

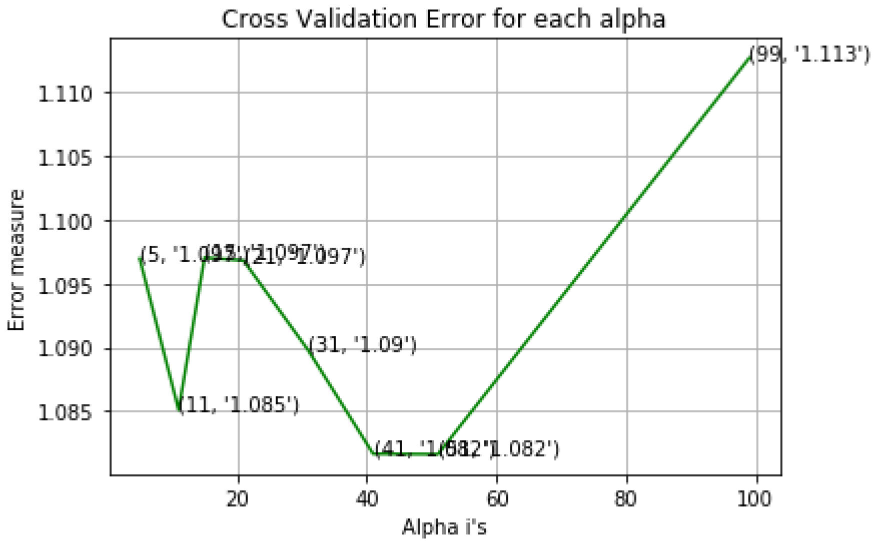
```
Predicted Class : 2
Predicted Class Probabilities: [[0.0718 0.6318 0.0143 0.0814 0.0412 0.0434 0.
1057 0.0059 0.0045]]
Actual Class : 2
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [62]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/m
odules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', lea
f_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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


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


alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_responseCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_responseCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes
_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probab
ility 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 i
s:",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_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 5
Log Loss : 1.0969303646472561
for alpha = 11
Log Loss : 1.0851016136985003
for alpha = 15
Log Loss : 1.0970109480905619
for alpha = 21
Log Loss : 1.0967950051940067
for alpha = 31
Log Loss : 1.0897571502715409
for alpha = 41
Log Loss : 1.0816610736908696
for alpha = 51
Log Loss : 1.081632841418107
for alpha = 99
Log Loss : 1.1126261578804915
```

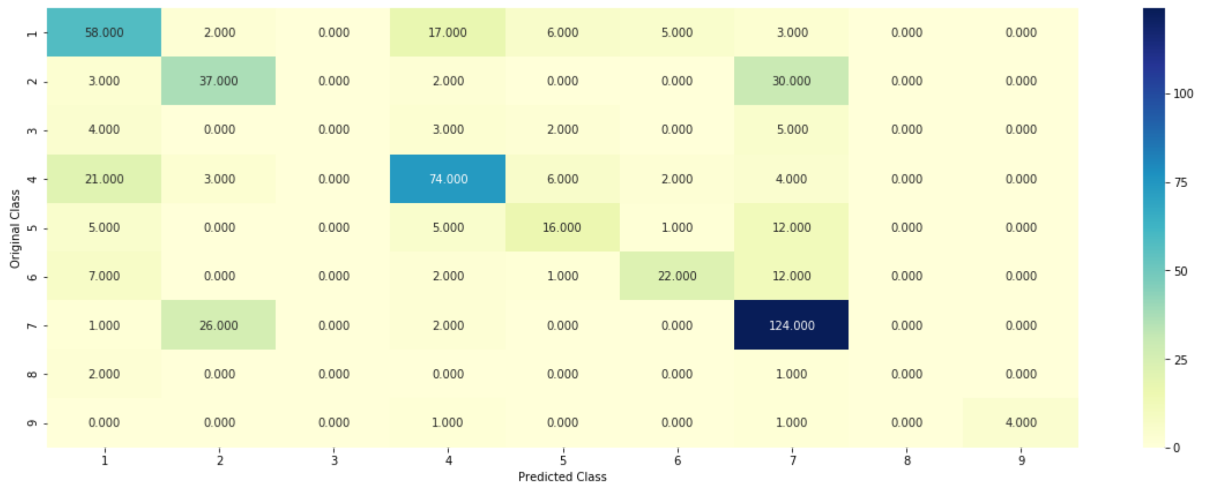


```
For values of best alpha =  51 The train log loss is: 0.8691792565078065
For values of best alpha =  51 The cross validation log loss is: 1.0816328414
18107
For values of best alpha =  51 The test log loss is: 1.0883792266791434
```

4.2.2. Testing the model with best hyper paramters

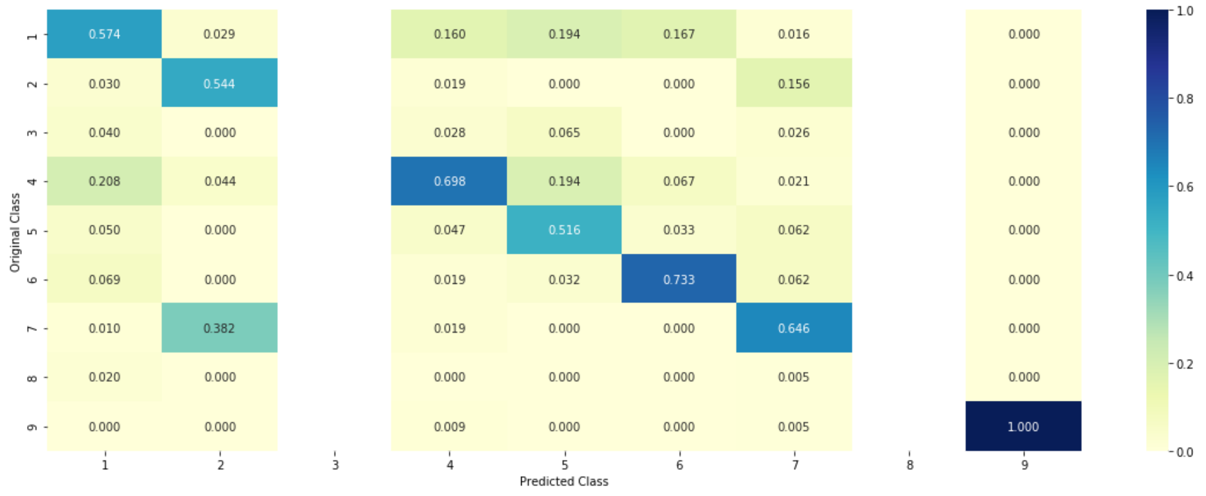
In [63]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/m
odules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', lea
f_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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

```
Log loss : 1.081632841418107
Number of mis-classified points : 0.37030075187969924
-------------------- Confusion matrix --------------------
```



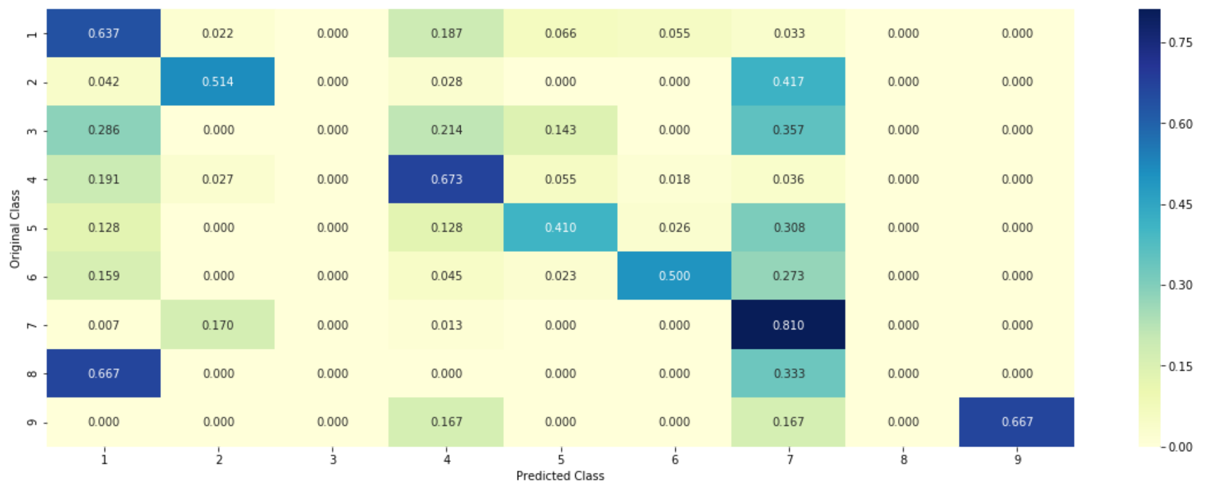
	1	2	3	4	5	6	7	8	9
1	58.000	2.000	0.000	17.000	6.000	5.000	3.000	0.000	0.000
2	3.000	37.000	0.000	2.000	0.000	0.000	30.000	0.000	0.000
3	4.000	0.000	0.000	3.000	2.000	0.000	5.000	0.000	0.000
4	21.000	3.000	0.000	74.000	6.000	2.000	4.000	0.000	0.000
5	5.000	0.000	0.000	5.000	16.000	1.000	12.000	0.000	0.000
6	7.000	0.000	0.000	2.000	1.000	22.000	12.000	0.000	0.000
7	1.000	26.000	0.000	2.000	0.000	0.000	124.000	0.000	0.000
8	2.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000
9	0.000	0.000	0.000	1.000	0.000	0.000	1.000	0.000	4.000

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



	1	2	3	4	5	6	7	8	9
1	0.574	0.029		0.160	0.194	0.167	0.016		0.000
2	0.030	0.544		0.019	0.000	0.000	0.156		0.000
3	0.040	0.000		0.028	0.065	0.000	0.026		0.000
4	0.208	0.044		0.698	0.194	0.067	0.021		0.000
5	0.050	0.000		0.047	0.516	0.033	0.062		0.000
6	0.069	0.000		0.019	0.032	0.733	0.062		0.000
7	0.010	0.382		0.019	0.000	0.000	0.646		0.000
8	0.020	0.000		0.000	0.000	0.000	0.005		0.000
9	0.000	0.000		0.009	0.000	0.000	0.005		1.000

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



	1	2	3	4	5	6	7	8	9
1	0.637	0.022	0.000	0.187	0.066	0.055	0.033	0.000	0.000
2	0.042	0.514	0.000	0.028	0.000	0.000	0.417	0.000	0.000
3	0.286	0.000	0.000	0.214	0.143	0.000	0.357	0.000	0.000
4	0.191	0.027	0.000	0.673	0.055	0.018	0.036	0.000	0.000
5	0.128	0.000	0.000	0.128	0.410	0.026	0.308	0.000	0.000
6	0.159	0.000	0.000	0.045	0.023	0.500	0.273	0.000	0.000
7	0.007	0.170	0.000	0.013	0.000	0.000	0.810	0.000	0.000
8	0.667	0.000	0.000	0.000	0.000	0.000	0.333	0.000	0.000
9	0.000	0.000	0.000	0.167	0.000	0.000	0.167	0.000	0.667

4.2.3.Sample Query point -1

In [64]:
```
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 : 1
Actual Class : 7
The  51  nearest neighbours of the test points belongs to classes [7 6 1 6 1
1 1 6 1 7 7 7 7 7 2 7 2 7 7 2 7 7 1 7 2 6 2 2 7 7 6 7 2 7 2 7 7
 2 2 7 7 4 7 7 7 7 7 7 7 7 7 1]
Fequency of nearest points : Counter({7: 28, 2: 10, 1: 7, 6: 5, 4: 1})
```

4.2.4. Sample Query Point-2

In [65]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 100

predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshap
e(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 o
f the test points belongs to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 2
Actual Class : 2
the k value for knn is 51 and the nearest neighbours of the test points belon
gs to classes [2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 7 2 2 2 2 2 2 2 2 2 2 2 2 2
2 7 7 7 2 7
 2 2 2 2 2 7 7 2 2 2 7 7 2 7]
Fequency of nearest points : Counter({2: 41, 7: 10})
```

4.3. Logistic Regression

With Count Vectorizer and Unigram and bigram

In [66]:
```
# one-hot encoding of Gene feature.
gene_vectorizer_lr = CountVectorizer()
train_gene_feature_onehotCoding_lr = gene_vectorizer_lr.fit_transform(train_df
['Gene'])
test_gene_feature_onehotCoding_lr = gene_vectorizer_lr.transform(test_df['Gen
e'])
cv_gene_feature_onehotCoding_lr = gene_vectorizer_lr.transform(cv_df['Gene'])


# one-hot encoding of variation feature.
variation_vectorizer_lr = CountVectorizer()
train_variation_feature_onehotCoding_lr = variation_vectorizer_lr.fit_transfor
m(train_df['Variation'])
test_variation_feature_onehotCoding_lr = variation_vectorizer_lr.transform(tes
t_df['Variation'])
cv_variation_feature_onehotCoding_lr = variation_vectorizer_lr.transform(cv_df
['Variation'])


text_vectorizer_lr = CountVectorizer(ngram_range=(1, 4))
train_text_feature_onehotCoding_lr = text_vectorizer_lr.fit_transform(train_df
['TEXT'])
# don't forget to normalize every feature
train_text_feature_onehotCoding_lr = normalize(train_text_feature_onehotCoding
_lr, axis=0)

test_text_feature_onehotCoding_lr = text_vectorizer_lr.transform(test_df['TEX
T'])
test_text_feature_onehotCoding_lr = normalize(test_text_feature_onehotCoding_l
r, axis=0)

cv_text_feature_onehotCoding_lr = text_vectorizer_lr.transform(cv_df['TEXT'])
# don't forget to normalize every feature
cv_text_feature_onehotCoding_lr = normalize(cv_text_feature_onehotCoding_lr, a
xis=0)
```

stacking all the features(gene,vartions,text of one-hot encoded)

In [67]:
```
train_gene_var_onehotCoding_lr = hstack((train_gene_feature_onehotCoding_lr,tr
ain_variation_feature_onehotCoding_lr))
test_gene_var_onehotCoding_lr = hstack((test_gene_feature_onehotCoding_lr,test
_variation_feature_onehotCoding_lr))
cv_gene_var_onehotCoding_lr = hstack((cv_gene_feature_onehotCoding_lr,cv_varia
tion_feature_onehotCoding_lr))

train_x_onehotCoding_lr = hstack((train_gene_var_onehotCoding_lr, train_text_f
eature_onehotCoding_lr)).tocsr()
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding_lr = hstack((test_gene_var_onehotCoding_lr, test_text_feat
ure_onehotCoding_lr)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_onehotCoding_lr = hstack((cv_gene_var_onehotCoding_lr, cv_text_feature_on
ehotCoding_lr)).tocsr()
cv_y = np.array(list(cv_df['Class']))
```

In [68]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x
_onehotCoding_lr.shape)
print("(number of data points * number of features) in test data = ", test_x_o
nehotCoding_lr.shape)
print("(number of data points * number of features) in cross validation data
 =", cv_x_onehotCoding_lr.shape)
```

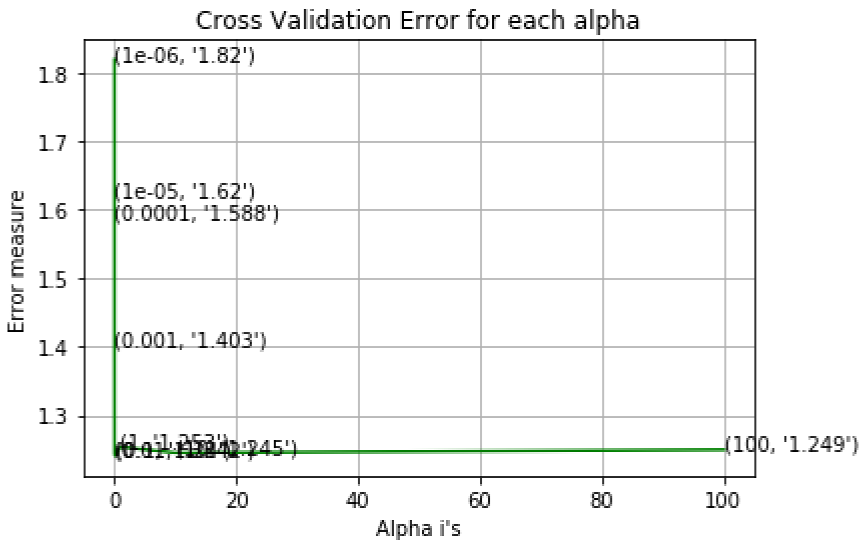
```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 1191560
9)
(number of data points * number of features) in test data =  (665, 11915609)
(number of data points * number of features) in cross validation data = (532,
11915609)
```

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [69]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=’hinge’, penalty=’l2’, alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate=’optimal’, eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=’sigm
oid’, 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_lr, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding_lr, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding_lr)
    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-probab
ility estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCoding_lr)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss i
s:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding_lr)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation
 log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding_lr)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.8201104916799518
for alpha = 1e-05
Log Loss : 1.6203771586952185
for alpha = 0.0001
Log Loss : 1.5875493680791593
for alpha = 0.001
Log Loss : 1.403376474810247
for alpha = 0.01
Log Loss : 1.2423272309940365
for alpha = 0.1
Log Loss : 1.2401531939506085
for alpha = 1
Log Loss : 1.2532748766119728
for alpha = 10
Log Loss : 1.244574367442384
for alpha = 100
Log Loss : 1.2491176828345538
```

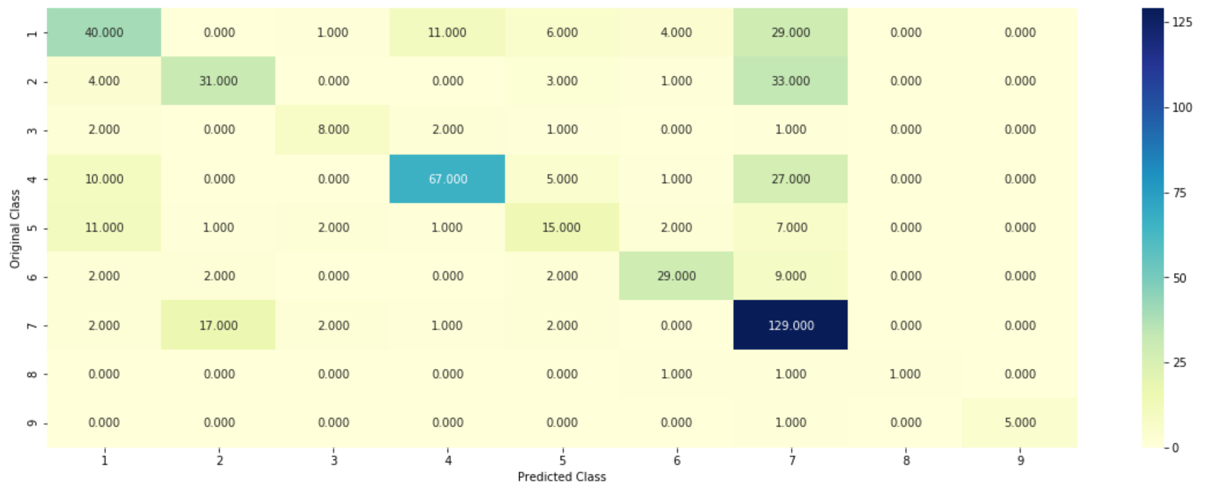


```
For values of best alpha =  0.1 The train log loss is: 0.5540673088073816
For values of best alpha =  0.1 The cross validation log loss is: 1.240153193
9506085
For values of best alpha =  0.1 The test log loss is: 1.2425756049800392
```

4.3.1.2. Testing the model with best hyper paramters

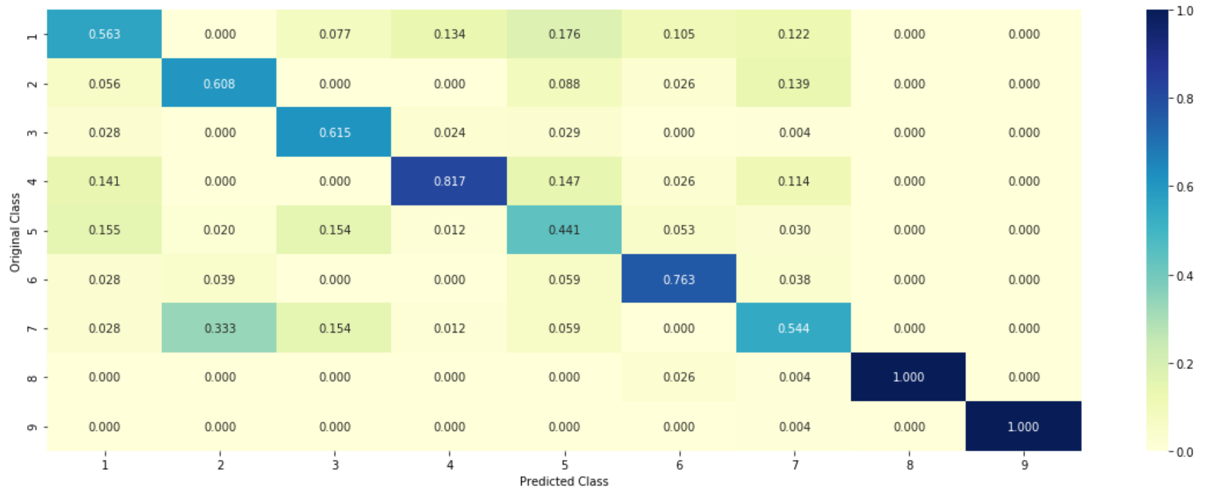
In [70]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

```
Log loss : 1.2401531939506085
Number of mis-classified points : 0.38909774436090923
-------------------- Confusion matrix --------------------
```



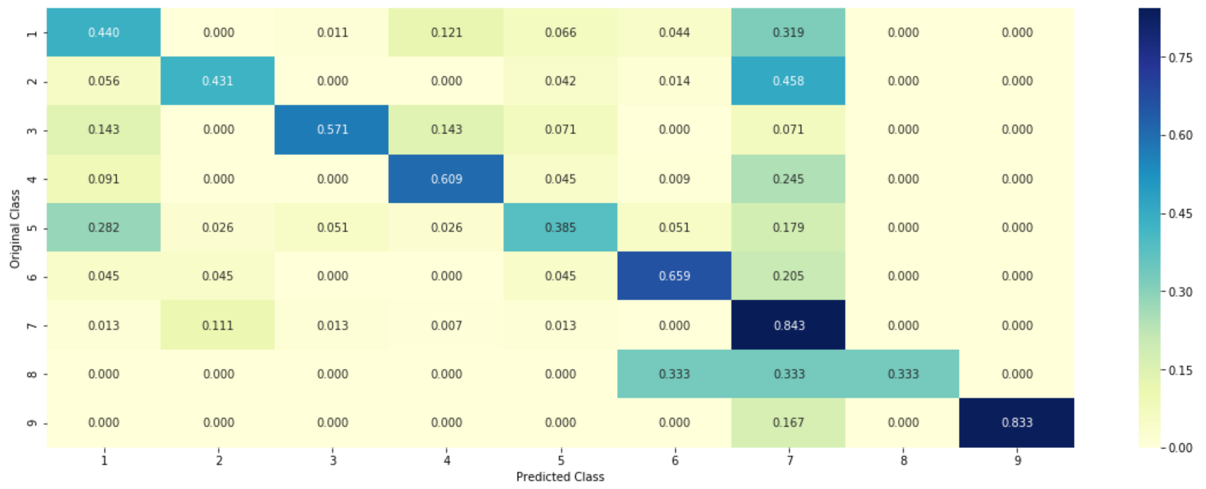
	1	2	3	4	5	6	7	8	9
1	40.000	0.000	1.000	11.000	6.000	4.000	29.000	0.000	0.000
2	4.000	31.000	0.000	0.000	3.000	1.000	33.000	0.000	0.000
3	2.000	0.000	8.000	2.000	1.000	0.000	1.000	0.000	0.000
4	10.000	0.000	0.000	67.000	5.000	1.000	27.000	0.000	0.000
5	11.000	1.000	2.000	1.000	15.000	2.000	7.000	0.000	0.000
6	2.000	2.000	0.000	0.000	2.000	29.000	9.000	0.000	0.000
7	2.000	17.000	2.000	1.000	2.000	0.000	129.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	1.000	1.000	1.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	5.000

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



	1	2	3	4	5	6	7	8	9
1	0.563	0.000	0.077	0.134	0.176	0.105	0.122	0.000	0.000
2	0.056	0.608	0.000	0.000	0.088	0.026	0.139	0.000	0.000
3	0.028	0.000	0.615	0.024	0.029	0.000	0.004	0.000	0.000
4	0.141	0.000	0.000	0.817	0.147	0.026	0.114	0.000	0.000
5	0.155	0.020	0.154	0.012	0.441	0.053	0.030	0.000	0.000
6	0.028	0.039	0.000	0.000	0.059	0.763	0.038	0.000	0.000
7	0.028	0.333	0.154	0.012	0.059	0.000	0.544	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.026	0.004	1.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.004	0.000	1.000

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



	1	2	3	4	5	6	7	8	9
1	0.440	0.000	0.011	0.121	0.066	0.044	0.319	0.000	0.000
2	0.056	0.431	0.000	0.000	0.042	0.014	0.458	0.000	0.000
3	0.143	0.000	0.571	0.143	0.071	0.000	0.071	0.000	0.000
4	0.091	0.000	0.000	0.609	0.045	0.009	0.245	0.000	0.000
5	0.282	0.026	0.051	0.026	0.385	0.051	0.179	0.000	0.000
6	0.045	0.045	0.000	0.000	0.045	0.659	0.205	0.000	0.000
7	0.013	0.111	0.013	0.007	0.013	0.000	0.843	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.333	0.333	0.333	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.167	0.000	0.833

4.3.1.3. Feature Importance

In [71]:
```
#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], y
es_no])
#        incresingorder_ind += 1
#    print(word_present, "most importent features are present in our query poi
nt")
#    print("-"*50)
#    print("The features that are most importent of the ",predicted_cls[0]," c
lass:")
#    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present o
r Not']))
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not

def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(ngram_range=(1, 4))

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

```
        else:
            word = text_vec.get_feature_names()[v-(fea1_len+fea2_len)]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
                print(i, "Text feature [{}] present in test data point [{}]".f
ormat(word,yes_no))

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

4.3.1.3.1. Correctly Classified point

In [72]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty=
'l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding_lr,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_lr[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding_lr[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
#indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
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_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1836 0.1433 0.0285 0.1797 0.077  0.0788 0.
2936 0.005  0.0103]]
Actual Class : 7
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [73]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_lr[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding_lr[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0122 0.9412 0.0022 0.0096 0.006  0.003  0.
0185 0.005  0.0024]]
Actual Class : 2
--------------------------------------------------
32 Text feature [wild] present in test data point [True]
33 Text feature [type] present in test data point [True]
36 Text feature [predicted] present in test data point [True]
38 Text feature [results] present in test data point [True]
41 Text feature [whereas] present in test data point [True]
43 Text feature [acid] present in test data point [True]
44 Text feature [used] present in test data point [True]
46 Text feature [amino] present in test data point [True]
47 Text feature [likely] present in test data point [True]
49 Text feature [determine] present in test data point [True]
50 Text feature [sequence] present in test data point [True]
54 Text feature [expected] present in test data point [True]
61 Text feature [data] present in test data point [True]
62 Text feature [thus] present in test data point [True]
63 Text feature [whether] present in test data point [True]
64 Text feature [although] present in test data point [True]
65 Text feature [shown] present in test data point [True]
66 Text feature [showed] present in test data point [True]
67 Text feature [previously] present in test data point [True]
68 Text feature [introduction] present in test data point [True]
70 Text feature [indicate] present in test data point [True]
72 Text feature [therefore] present in test data point [True]
73 Text feature [independent] present in test data point [True]
76 Text feature [containing] present in test data point [True]
77 Text feature [additional] present in test data point [True]
82 Text feature [also] present in test data point [True]
84 Text feature [mutant] present in test data point [True]
85 Text feature [using] present in test data point [True]
86 Text feature [addition] present in test data point [True]
87 Text feature [either] present in test data point [True]
88 Text feature [provide] present in test data point [True]
90 Text feature [well] present in test data point [True]
93 Text feature [indicated] present in test data point [True]
95 Text feature [result] present in test data point [True]
102 Text feature [found] present in test data point [True]
103 Text feature [changes] present in test data point [True]
105 Text feature [analysis] present in test data point [True]
112 Text feature [full] present in test data point [True]
113 Text feature [important] present in test data point [True]
115 Text feature [analyzed] present in test data point [True]
116 Text feature [critical] present in test data point [True]
117 Text feature [mutations] present in test data point [True]
118 Text feature [proteins] present in test data point [True]
119 Text feature [one] present in test data point [True]
121 Text feature [50] present in test data point [True]
124 Text feature [may] present in test data point [True]
125 Text feature [associated] present in test data point [True]
127 Text feature [domain] present in test data point [True]
130 Text feature [possible] present in test data point [True]
134 Text feature [resulting] present in test data point [True]
137 Text feature [assays] present in test data point [True]
138 Text feature [methods] present in test data point [True]
```

```
139 Text feature [directly] present in test data point [True]
141 Text feature [reduced] present in test data point [True]
146 Text feature [change] present in test data point [True]
147 Text feature [lower] present in test data point [True]
148 Text feature [figure] present in test data point [True]
151 Text feature [based] present in test data point [True]
152 Text feature [form] present in test data point [True]
153 Text feature [expressed] present in test data point [True]
156 Text feature [fact] present in test data point [True]
159 Text feature [cells] present in test data point [True]
160 Text feature [known] present in test data point [True]
161 Text feature [substitutions] present in test data point [True]
163 Text feature [note] present in test data point [True]
164 Text feature [highly] present in test data point [True]
166 Text feature [vitro] present in test data point [True]
168 Text feature [however] present in test data point [True]
170 Text feature [binding] present in test data point [True]
179 Text feature [levels] present in test data point [True]
181 Text feature [several] present in test data point [True]
184 Text feature [identified] present in test data point [True]
189 Text feature [performed] present in test data point [True]
193 Text feature [effects] present in test data point [True]
194 Text feature [mutation] present in test data point [True]
195 Text feature [represent] present in test data point [True]
196 Text feature [general] present in test data point [True]
197 Text feature [10] present in test data point [True]
198 Text feature [table] present in test data point [True]
199 Text feature [cancer] present in test data point [True]
201 Text feature [contribute] present in test data point [True]
204 Text feature [reported] present in test data point [True]
205 Text feature [members] present in test data point [True]
209 Text feature [indicates] present in test data point [True]
210 Text feature [residues] present in test data point [True]
213 Text feature [approximately] present in test data point [True]
214 Text feature [least] present in test data point [True]
217 Text feature [suggested] present in test data point [True]
221 Text feature [could] present in test data point [True]
227 Text feature [presence] present in test data point [True]
230 Text feature [30] present in test data point [True]
238 Text feature [would] present in test data point [True]
239 Text feature [different] present in test data point [True]
241 Text feature [genetic] present in test data point [True]
242 Text feature [cause] present in test data point [True]
245 Text feature [respectively] present in test data point [True]
247 Text feature [large] present in test data point [True]
248 Text feature [within] present in test data point [True]
249 Text feature [absence] present in test data point [True]
254 Text feature [derived] present in test data point [True]
255 Text feature [comparison] present in test data point [True]
259 Text feature [25] present in test data point [True]
265 Text feature [selected] present in test data point [True]
267 Text feature [predict] present in test data point [True]
270 Text feature [obtained] present in test data point [True]
274 Text feature [taken] present in test data point [True]
277 Text feature [26] present in test data point [True]
290 Text feature [available] present in test data point [True]
293 Text feature [possibility] present in test data point [True]
```

```
296 Text feature [confirmed] present in test data point [True]
301 Text feature [fold] present in test data point [True]
303 Text feature [studies] present in test data point [True]
304 Text feature [even] present in test data point [True]
305 Text feature [except] present in test data point [True]
306 Text feature [24] present in test data point [True]
310 Text feature [position] present in test data point [True]
311 Text feature [allows] present in test data point [True]
315 Text feature [present] present in test data point [True]
320 Text feature [identify] present in test data point [True]
321 Text feature [support] present in test data point [True]
322 Text feature [given] present in test data point [True]
324 Text feature [observation] present in test data point [True]
331 Text feature [approach] present in test data point [True]
332 Text feature [forms] present in test data point [True]
333 Text feature [cell] present in test data point [True]
336 Text feature [strong] present in test data point [True]
341 Text feature [yet] present in test data point [True]
342 Text feature [according] present in test data point [True]
345 Text feature [caused] present in test data point [True]
348 Text feature [relevant] present in test data point [True]
349 Text feature [specific] present in test data point [True]
350 Text feature [include] present in test data point [True]
352 Text feature [useful] present in test data point [True]
359 Text feature [frequently] present in test data point [True]
362 Text feature [significantly] present in test data point [True]
365 Text feature [similarly] present in test data point [True]
375 Text feature [recently] present in test data point [True]
378 Text feature [remaining] present in test data point [True]
384 Text feature [35] present in test data point [True]
385 Text feature [allow] present in test data point [True]
387 Text feature [low] present in test data point [True]
389 Text feature [grown] present in test data point [True]
391 Text feature [many] present in test data point [True]
397 Text feature [single] present in test data point [True]
400 Text feature [standard] present in test data point [True]
403 Text feature [model] present in test data point [True]
404 Text feature [assessment] present in test data point [True]
409 Text feature [little] present in test data point [True]
415 Text feature [proposed] present in test data point [True]
418 Text feature [resulted] present in test data point [True]
420 Text feature [including] present in test data point [True]
424 Text feature [detected] present in test data point [True]
425 Text feature [assessed] present in test data point [True]
435 Text feature [involved] present in test data point [True]
436 Text feature [less] present in test data point [True]
437 Text feature [basis] present in test data point [True]
441 Text feature [exception] present in test data point [True]
452 Text feature [structural] present in test data point [True]
453 Text feature [high] present in test data point [True]
458 Text feature [panel] present in test data point [True]
459 Text feature [impair] present in test data point [True]
463 Text feature [40] present in test data point [True]
466 Text feature [combined] present in test data point [True]
476 Text feature [difficult] present in test data point [True]
478 Text feature [31] present in test data point [True]
483 Text feature [published] present in test data point [True]
```

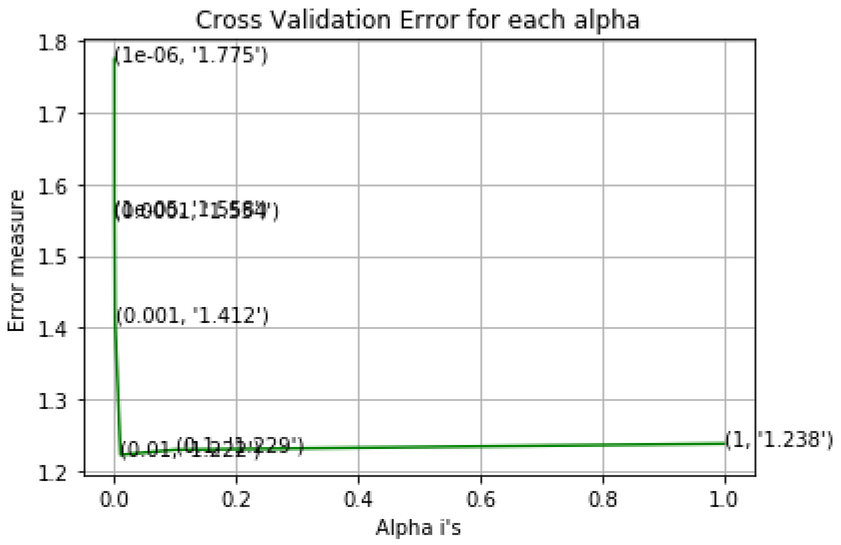
```
484 Text feature [comprehensive] present in test data point [True]
487 Text feature [abrogate] present in test data point [True]
489 Text feature [reduction] present in test data point [True]
496 Text feature [relatively] present in test data point [True]
497 Text feature [29] present in test data point [True]
499 Text feature [might] present in test data point [True]
Out of the top  500  features  172 are present in query point
```

4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

In [74]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=’hinge’, penalty=’l2’, alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate=’optimal’, eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=’sigm
oid’, 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_lr, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding_lr, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding_lr)
    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_lr, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding_lr, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding_lr)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss i
s:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding_lr)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation
 log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding_lr)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.774894857014709
for alpha = 1e-05
Log Loss : 1.5578290286106147
for alpha = 0.0001
Log Loss : 1.5544148761287524
for alpha = 0.001
Log Loss : 1.4115997798122784
for alpha = 0.01
Log Loss : 1.2223735845597772
for alpha = 0.1
Log Loss : 1.2294773325874306
for alpha = 1
Log Loss : 1.2379790995332929
```

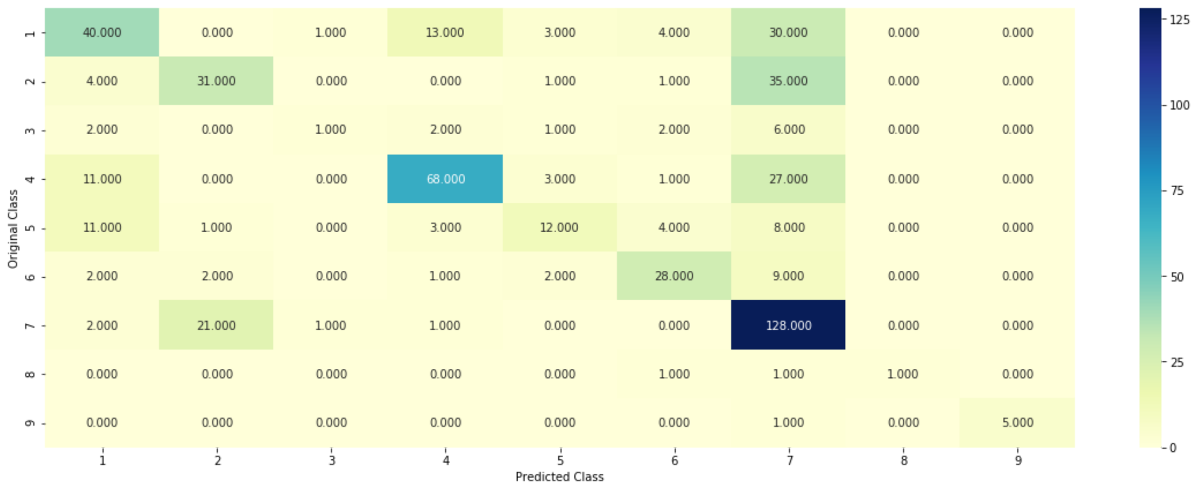


```
For values of best alpha =  0.01 The train log loss is: 0.6068200320716836
For values of best alpha =  0.01 The cross validation log loss is: 1.22237358
45597772
For values of best alpha =  0.01 The test log loss is: 1.235224292000013
```

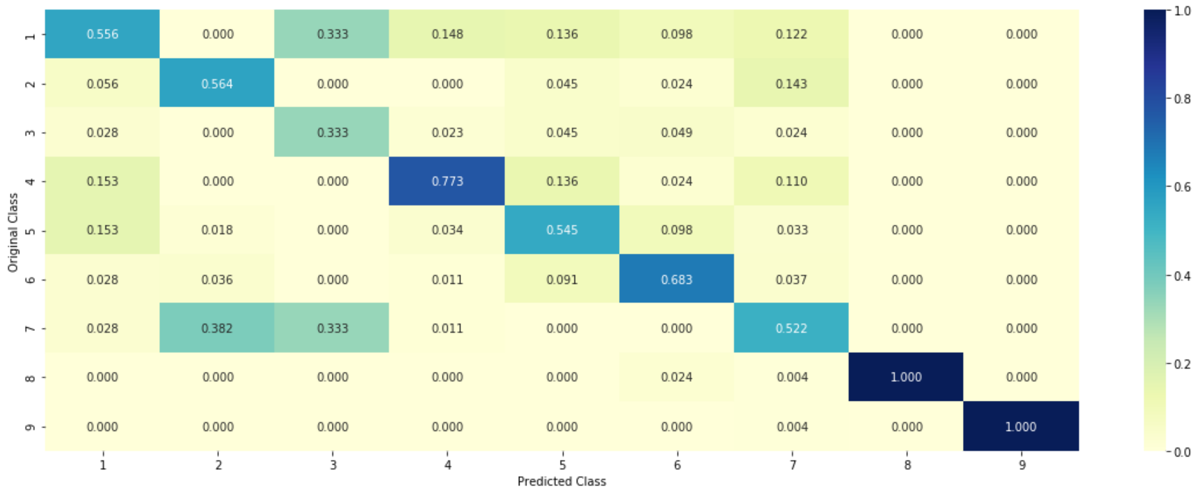
4.3.2.2. Testing model with best hyper parameters

In [75]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, …])     Fit linear model with Stochast
ic 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_lr, train_y, cv_x_oneho
tCoding_lr, cv_y, clf)
```

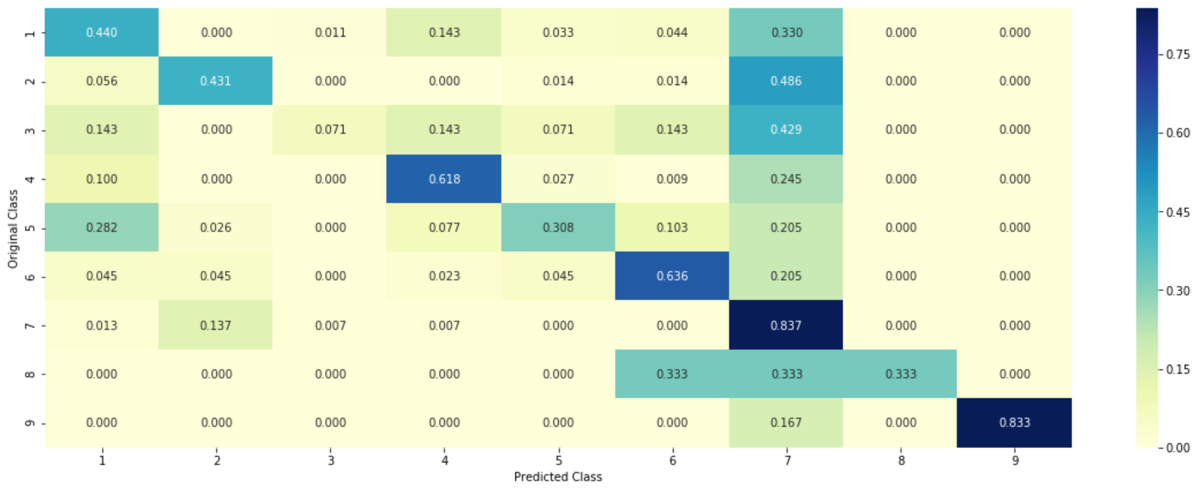
```
Log loss : 1.2223735845597772
Number of mis-classified points : 0.40977443609022557
-------------------- Confusion matrix --------------------
```



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



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



4.3.2.3. Feature Importance, Correctly Classified point

In [76]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_
state=42)
clf.fit(train_x_onehotCoding_lr,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_lr[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding_lr[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1797 0.1384 0.0365 0.1727 0.0877 0.0913 0.
2744 0.0055 0.0138]]
Actual Class : 7
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [77]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_lr[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding_lr[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0179 0.8893 0.0069 0.0126 0.0096 0.0048 0.
0507 0.006  0.0022]]
Actual Class : 2
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

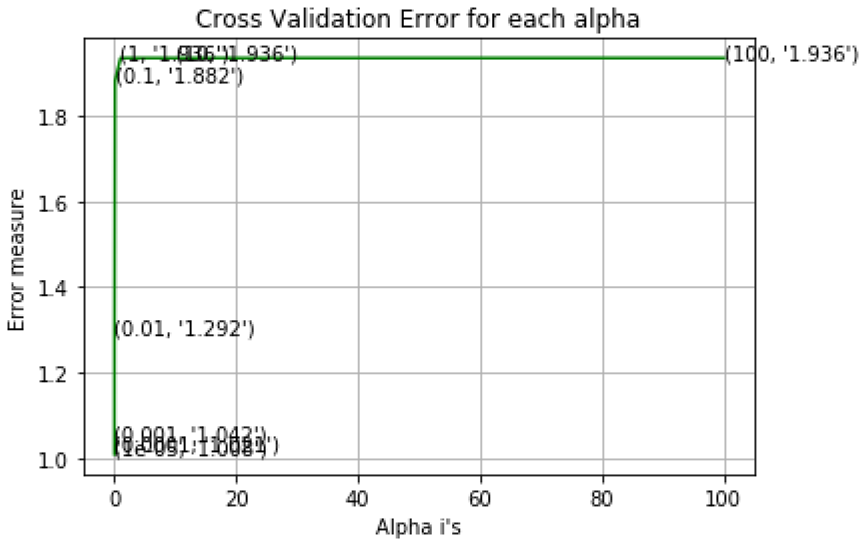
4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [78]:
```
# read more about support vector machines with linear kernals here http://scik
it-learn.org/stable/modules/generated/sklearn.svm.SVC.html

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

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given train
ing 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/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

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

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

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss i
s:",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_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for C = 1e-05
Log Loss : 1.0084816711655122
for C = 0.0001
Log Loss : 1.0211186485835775
for C = 0.001
Log Loss : 1.0424249526918998
for C = 0.01
Log Loss : 1.2923698826699872
for C = 0.1
Log Loss : 1.8817742008573548
for C = 1
Log Loss : 1.9360273420627463
for C = 10
Log Loss : 1.9360251971882068
for C = 100
Log Loss : 1.9360242795186076
```

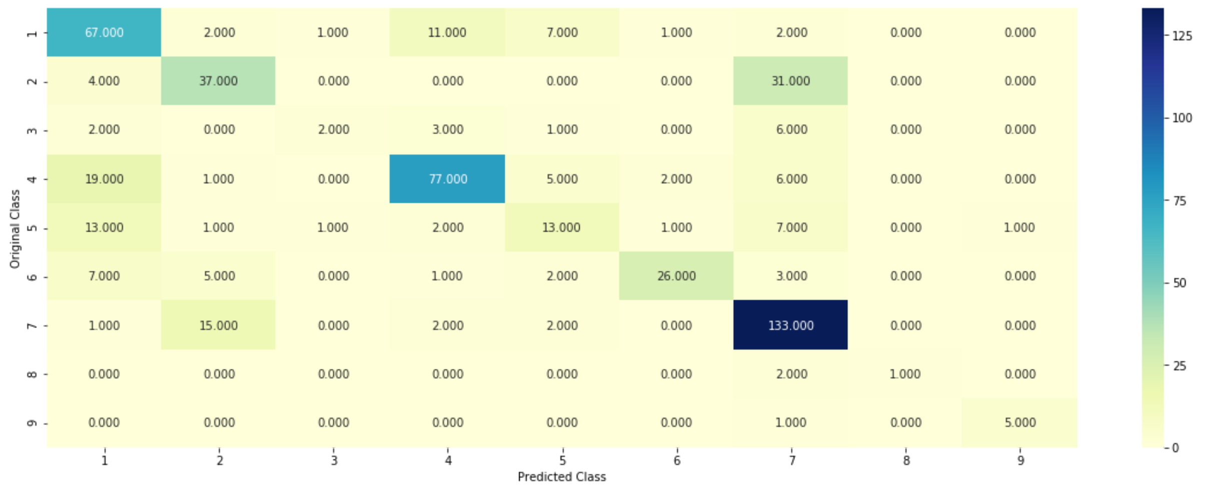


```
For values of best alpha =  1e-05 The train log loss is: 0.41723500769698235
For values of best alpha =  1e-05 The cross validation log loss is: 1.0084816
711655122
For values of best alpha =  1e-05 The test log loss is: 1.0867171998905403
```

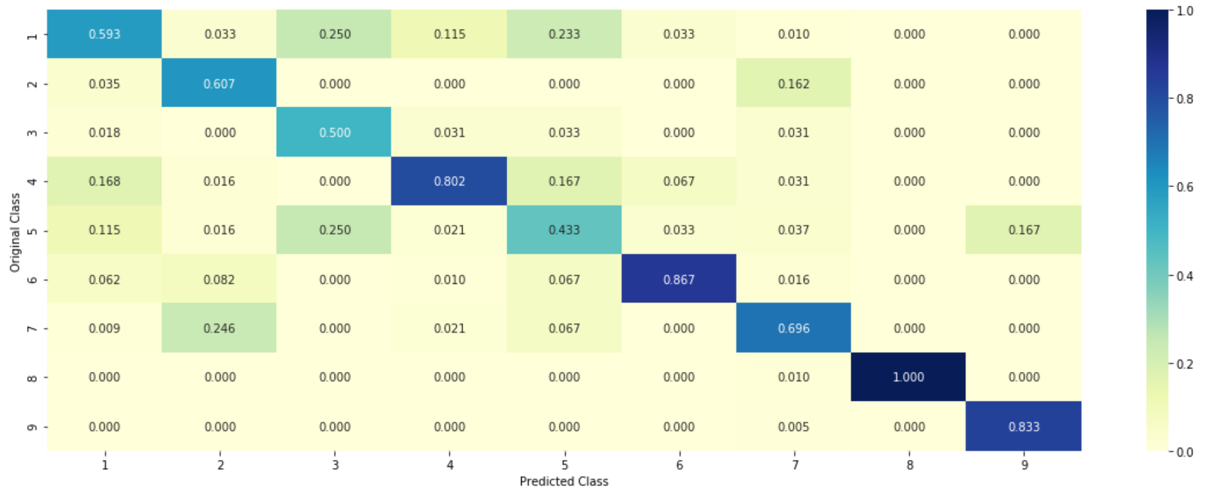
4.4.2. Testing model with best hyper parameters

In [79]:
```
# read more about support vector machines with linear kernals here http://scik
it-learn.org/stable/modules/generated/sklearn.svm.SVC.html

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

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given train
ing 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', rando
m_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCod
ing,cv_y, clf)
```

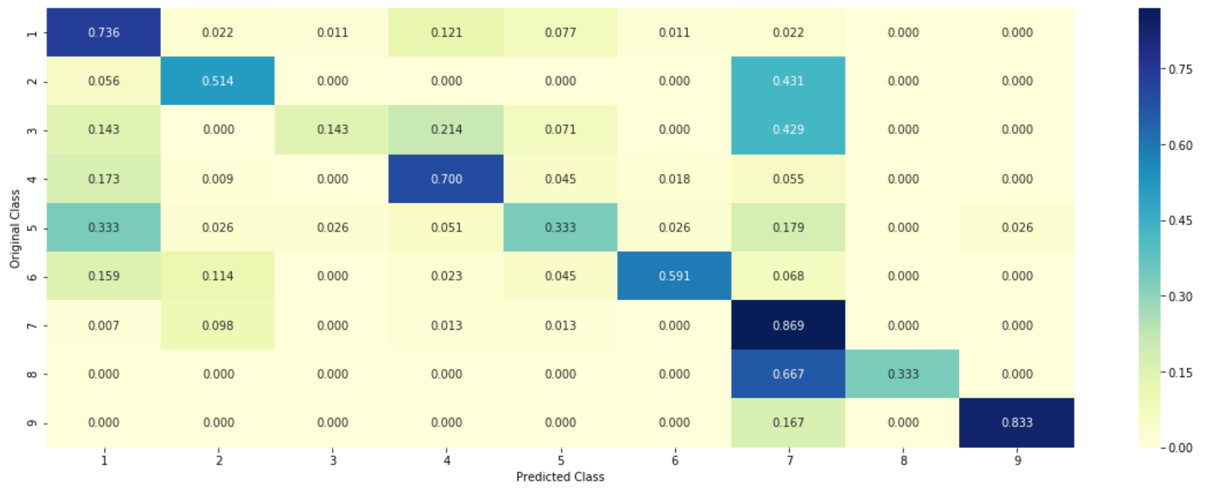
```
Log loss : 1.0084816711655122
Number of mis-classified points : 0.32142857142857145
-------------------- Confusion matrix --------------------
```



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



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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [80]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', rando
m_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(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1045 0.0585 0.0056 0.0072 0.0537 0.0231 0.
7386 0.0079 0.0009]]
Actual Class : 7
--------------------------------------------------
Out of the top  500  features  0 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(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[6.760e-02 8.167e-01 1.300e-03 3.200e-03 5.61
0e-02 1.910e-02 2.900e-02
  6.700e-03 3.000e-04]]
Actual Class : 2
--------------------------------------------------
Out of the top  500  features  0 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', m
ax_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_s
tate=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 train
ing data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X)     Perform classification on samples in X.

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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', 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_dep
th=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.cla
sses_, eps=1e-15))
```

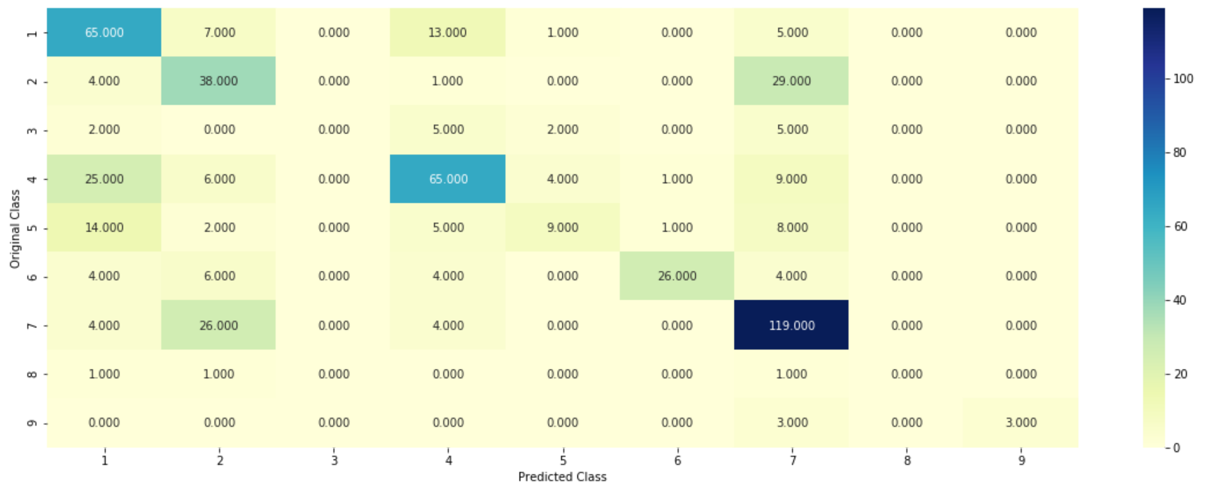
```
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

'''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],c
v_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=1
e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test l
og loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2281311521625655
for n_estimators = 100 and max depth =  10
Log Loss : 1.2016759360832534
for n_estimators = 200 and max depth =  5
Log Loss : 1.2175515228590474
for n_estimators = 200 and max depth =  10
Log Loss : 1.1925979376924802
for n_estimators = 500 and max depth =  5
Log Loss : 1.219568165490032
for n_estimators = 500 and max depth =  10
Log Loss : 1.1891083251834829
for n_estimators = 1000 and max depth =  5
Log Loss : 1.215188954131631
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1871364831297526
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2146143588782397
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1853708880800045
For values of best estimator =  2000 The train log loss is: 0.529806763228517
7
For values of best estimator =  2000 The cross validation log loss is: 1.1853
708880800042
For values of best estimator =  2000 The test log loss is: 1.18208984475796
```

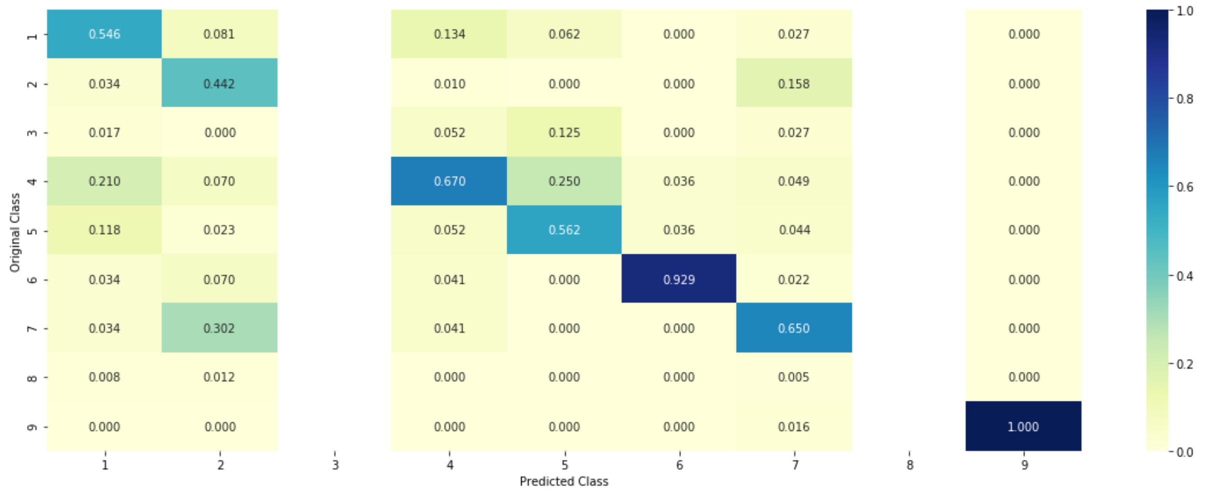
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [83]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=’gini’, m
ax_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=’auto’, max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_s
tate=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 train
ing 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_onehotCod
ing,cv_y, clf)
```

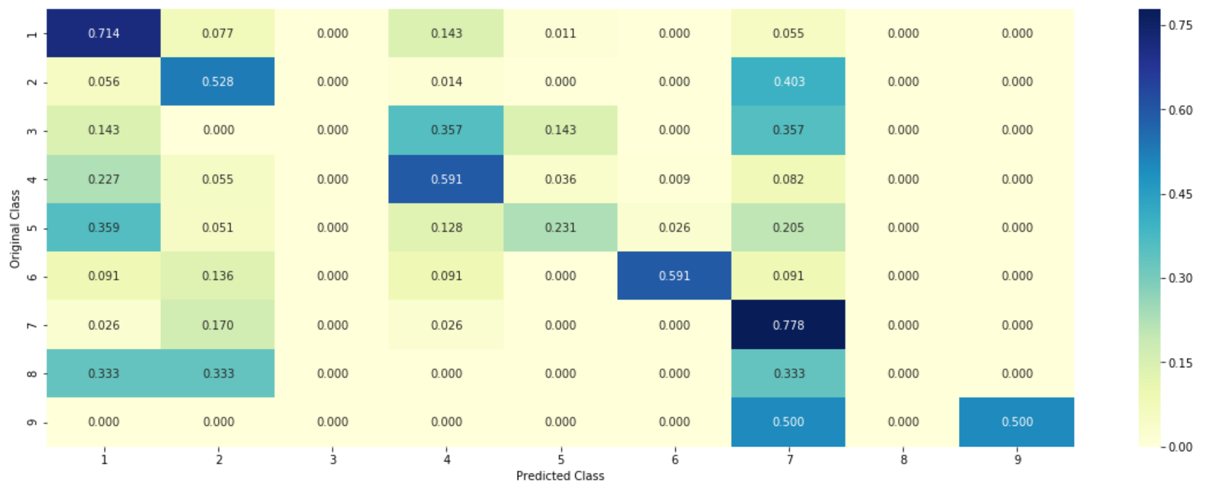
```
Log loss : 1.1853708880800045
Number of mis-classified points : 0.3890977443609023
-------------------- Confusion matrix --------------------
```



```
-------------------- 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_ind
ex],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_poin
t_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1483 0.2316 0.0227 0.1459 0.06   0.0604 0.
3133 0.0084 0.0094]]
Actual Class : 7
--------------------------------------------------
Out of the top  100  features  0 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_ind
ex],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_poin
t_index], no_feature)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0374 0.8232 0.0112 0.0237 0.0305 0.0276 0.
0379 0.0038 0.0047]]
Actuall Class : 2
--------------------------------------------------
Out of the top  100  features  0 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', m
ax_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_s
tate=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 train
ing data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X)     Perform classification on samples in X.

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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', 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_dep
th=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.cla
sses_, eps=1e-15))
```

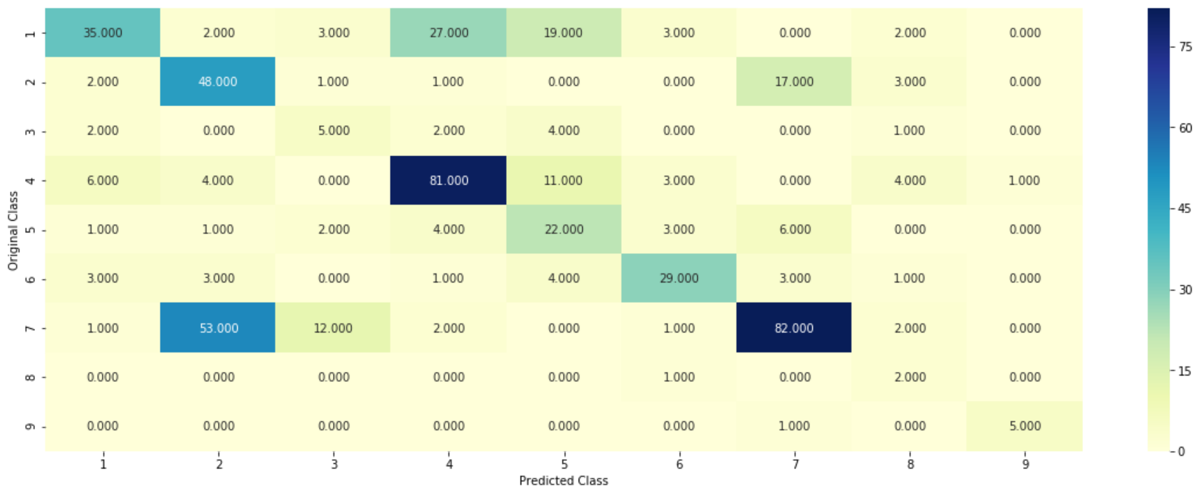
```
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (features[i],c
v_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 vali
dation 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 l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.061724756653649
for n_estimators = 10 and max depth =  3
Log Loss : 1.7910329172541843
for n_estimators = 10 and max depth =  5
Log Loss : 1.4895860644462515
for n_estimators = 10 and max depth =  10
Log Loss : 1.6284775889672949
for n_estimators = 50 and max depth =  2
Log Loss : 1.6817896209283998
for n_estimators = 50 and max depth =  3
Log Loss : 1.4416309015498987
for n_estimators = 50 and max depth =  5
Log Loss : 1.3743188677368006
for n_estimators = 50 and max depth =  10
Log Loss : 1.6418261536253445
for n_estimators = 100 and max depth =  2
Log Loss : 1.5642606651228206
for n_estimators = 100 and max depth =  3
Log Loss : 1.4474642777930014
for n_estimators = 100 and max depth =  5
Log Loss : 1.296283409073802
for n_estimators = 100 and max depth =  10
Log Loss : 1.707053409615415
for n_estimators = 200 and max depth =  2
Log Loss : 1.584006280908212
for n_estimators = 200 and max depth =  3
Log Loss : 1.4596795181637647
for n_estimators = 200 and max depth =  5
Log Loss : 1.3515525796758852
for n_estimators = 200 and max depth =  10
Log Loss : 1.67543766818155
for n_estimators = 500 and max depth =  2
Log Loss : 1.6037968514884746
for n_estimators = 500 and max depth =  3
Log Loss : 1.491675068311772
for n_estimators = 500 and max depth =  5
Log Loss : 1.3527340554950347
for n_estimators = 500 and max depth =  10
Log Loss : 1.6866252632477592
for n_estimators = 1000 and max depth =  2
Log Loss : 1.588058323618764
for n_estimators = 1000 and max depth =  3
Log Loss : 1.5009066626880303
for n_estimators = 1000 and max depth =  5
Log Loss : 1.354320213853695
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6762314802392537
For values of best alpha =  100 The train log loss is: 0.0690817567521354
For values of best alpha =  100 The cross validation log loss is: 1.296283409
0738018
For values of best alpha =  100 The test log loss is: 1.2929752726522183
```

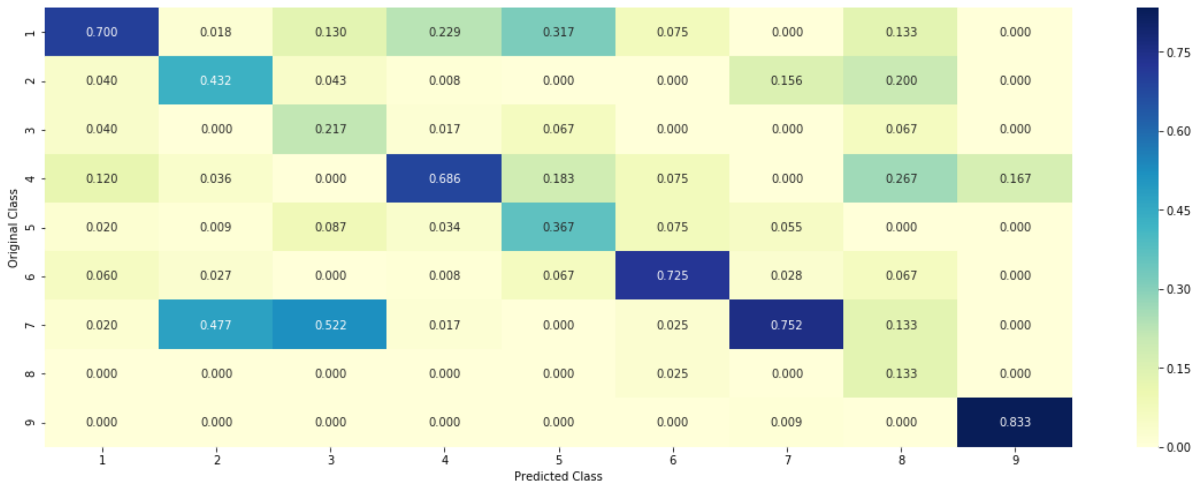
4.5.4. Testing model with best hyper parameters (Response Coding)

In [87]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', m
ax_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_s
tate=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 train
ing 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_estimat
ors=alpha[int(best_alpha/4)], criterion='gini', max_features='auto',random_sta
te=42)
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_respons
eCoding,cv_y, clf)
```

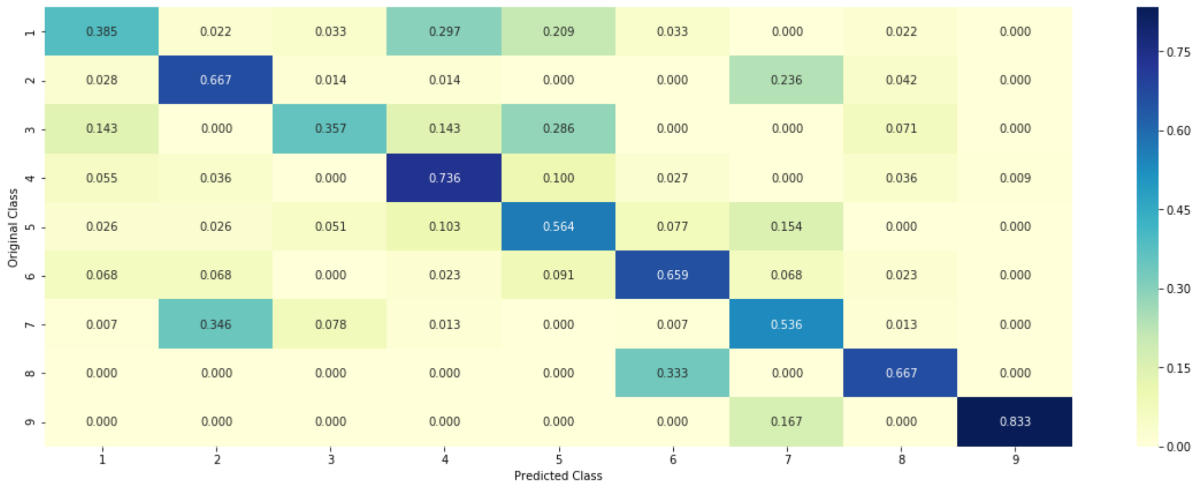
```
Log loss : 1.296283409073802
Number of mis-classified points : 0.4191729323308271
-------------------- Confusion matrix --------------------
```



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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0682 0.167  0.1092 0.062  0.096  0.1889 0.
1747 0.0791 0.0547]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [89]:
```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshap
e(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: [[0.0251 0.7726 0.0309 0.0272 0.017  0.035  0.
0373 0.0286 0.0263]]
Actual Class : 2
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [90]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



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

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given train
ing 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://scik
it-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.
html
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', m
ax_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_l
eaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_s
tate=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 train
ing data.
```

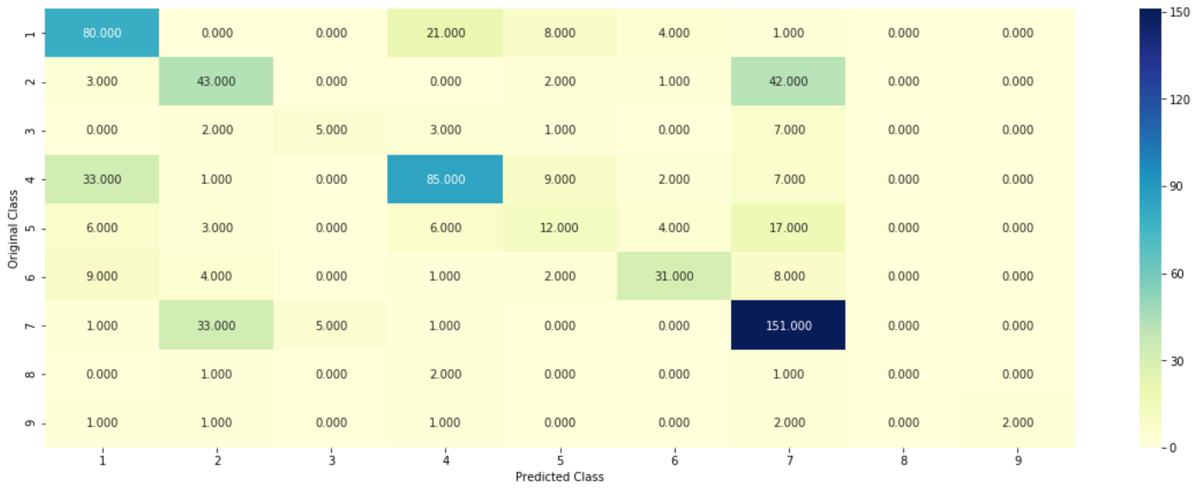
```
# predict(X)    Perform classification on samples in X.
# predict_proba (X)     Perform classification on samples in X.

# 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='bala
nced', 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='balanc
ed', 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.pred
ict_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.p
redict_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: 1.09
Support vector machines : Log Loss: 1.94
Naive Bayes : Log Loss: 1.24
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.817
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.715
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.320
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.243
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.598
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.977
```

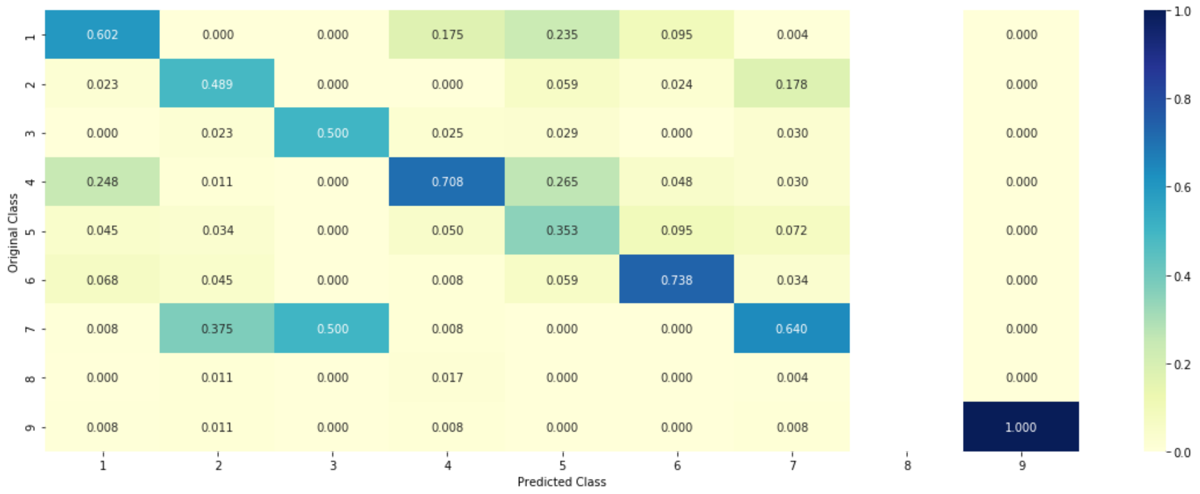
4.7.2 testing the model with the best hyper parameters

In [91]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_cla
ssifier=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_onehotCodin
g))
```

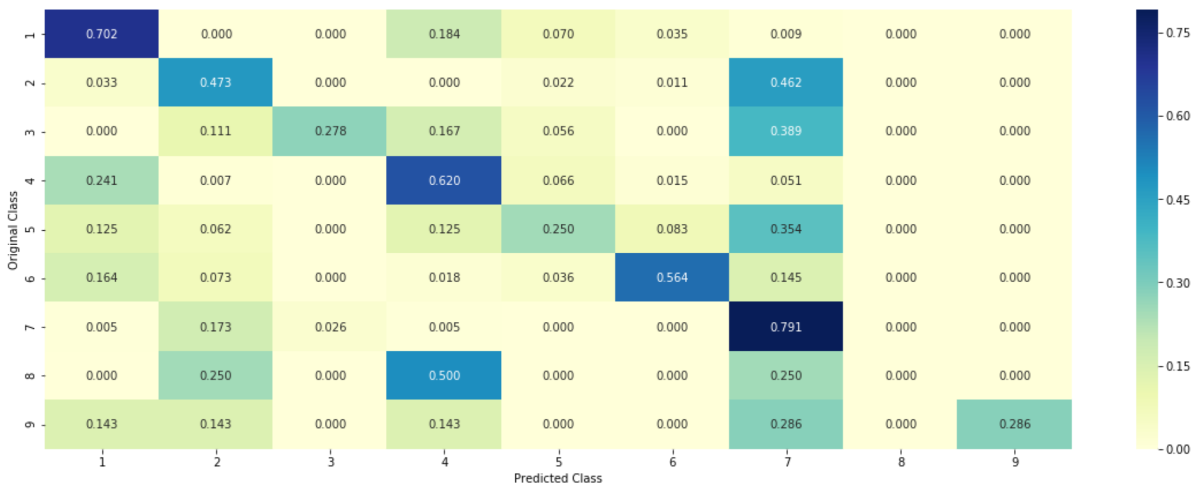
```
Log loss (train) on the stacking classifier : 0.4597113477737592
Log loss (CV) on the stacking classifier : 1.2434078280255831
Log loss (test) on the stacking classifier : 1.2759942326172031
Number of missclassified point : 0.3849624060150376
-------------------- Confusion matrix --------------------
```



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



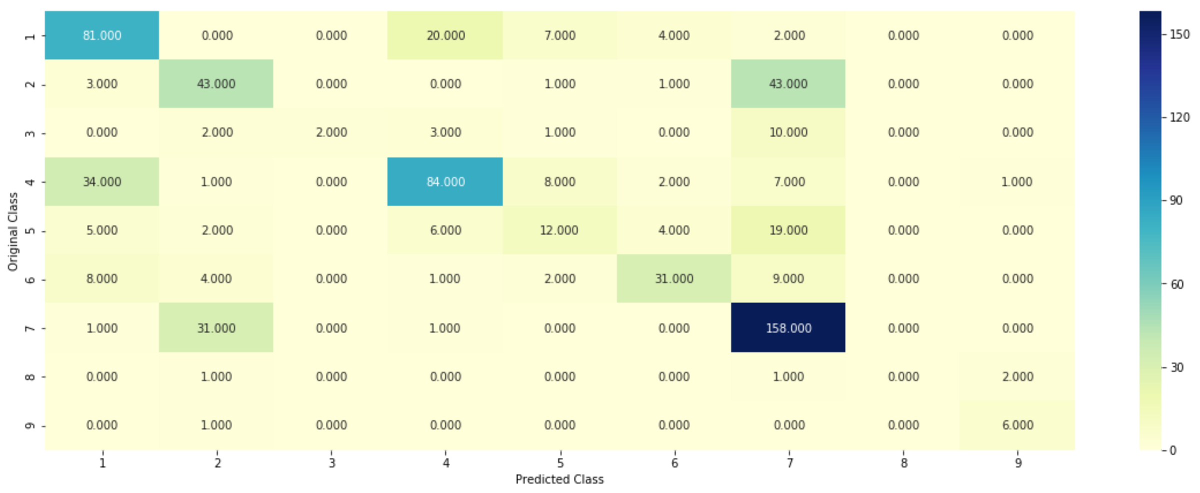
```
-------------------- Recall matrix (Row sum=1) --------------------
```



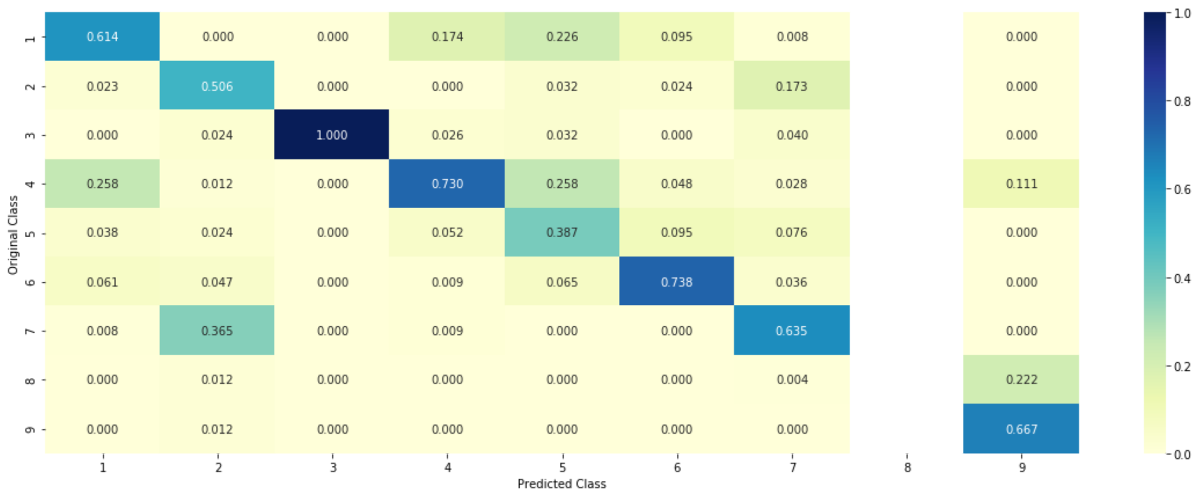
4.7.3 Maximum Voting classifier

In [92]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.Votin
gClassifier.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.pre
dict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_p
roba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predi
ct_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_onehotCodin
g))
```

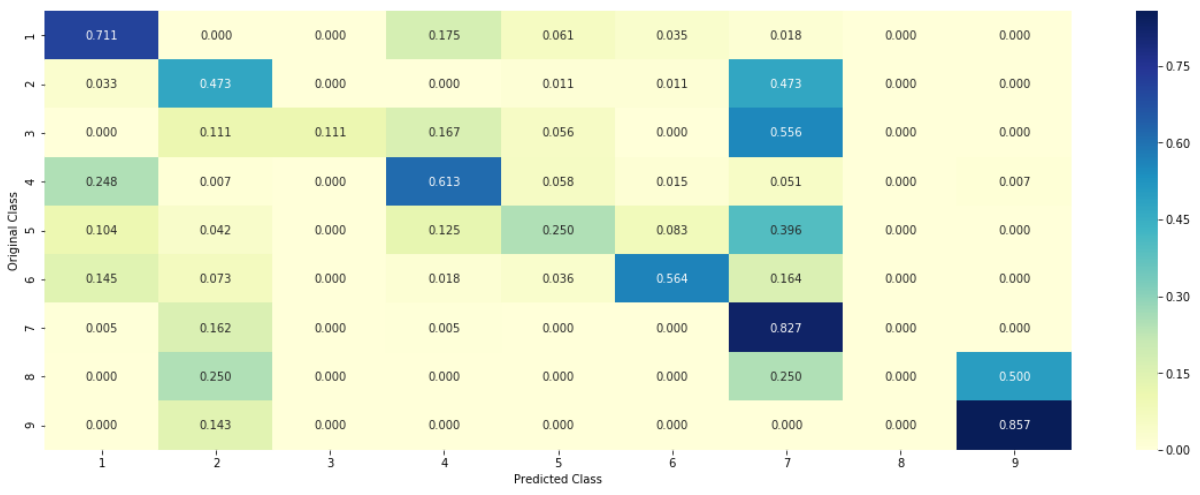
```
Log loss (train) on the VotingClassifier : 0.8295219395267811
Log loss (CV) on the VotingClassifier : 1.2063955879095085
Log loss (test) on the VotingClassifier : 1.224902534260774
Number of missclassified point : 0.37293233082706767
-------------------- Confusion matrix --------------------
```



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



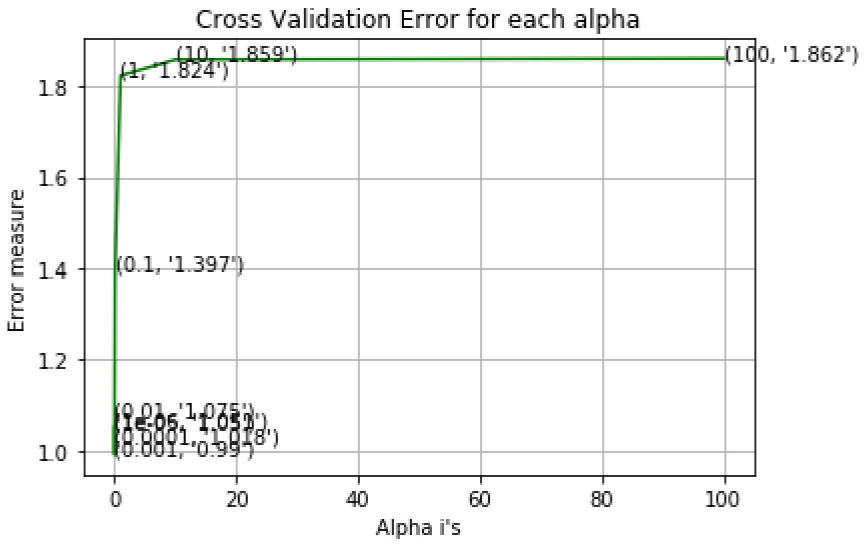
```
-------------------- Recall matrix (Row sum=1) --------------------
```



Logistic Regression with feature engineering

In [93]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stabl
e/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigm
oid', 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_onehotCodingFE, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCodingFE, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCodingFE)
    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-probab
ility estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCodingFE)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss i
s:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCodingFE)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation
 log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCodingFE)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.051003134377265
for alpha = 1e-05
Log Loss : 1.0497192969426299
for alpha = 0.0001
Log Loss : 1.0177458951615523
for alpha = 0.001
Log Loss : 0.9901953437928124
for alpha = 0.01
Log Loss : 1.0748573283596188
for alpha = 0.1
Log Loss : 1.3966976110157225
for alpha = 1
Log Loss : 1.8240950106014056
for alpha = 10
Log Loss : 1.8594408627142196
for alpha = 100
Log Loss : 1.8618215887417076
```

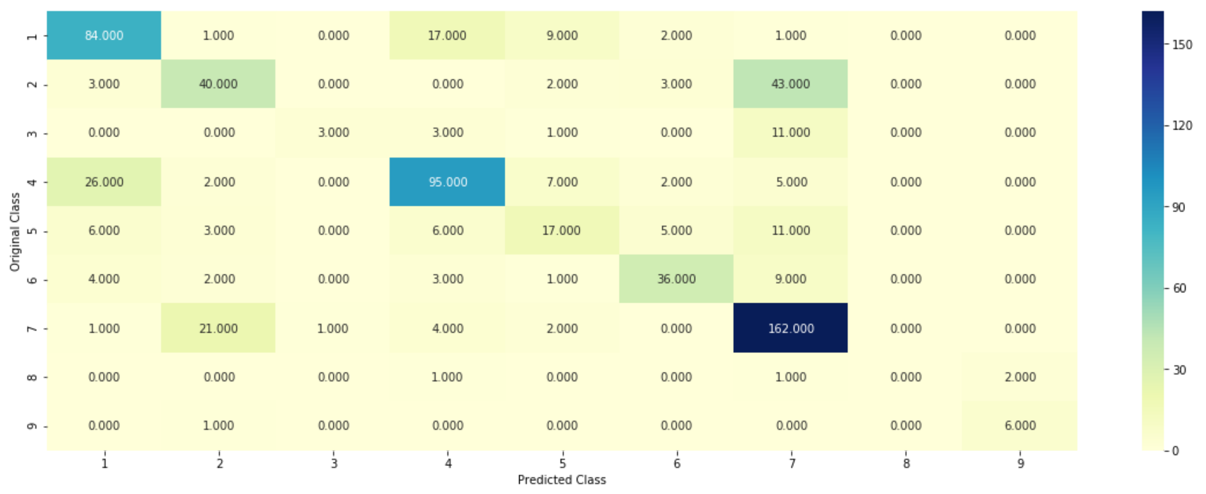


```
For values of best alpha =  0.001 The train log loss is: 0.5233614846923389
For values of best alpha =  0.001 The cross validation log loss is: 0.9901953
437928124
For values of best alpha =  0.001 The test log loss is: 0.9919153581128131
```

Testing model with best hyper parameters

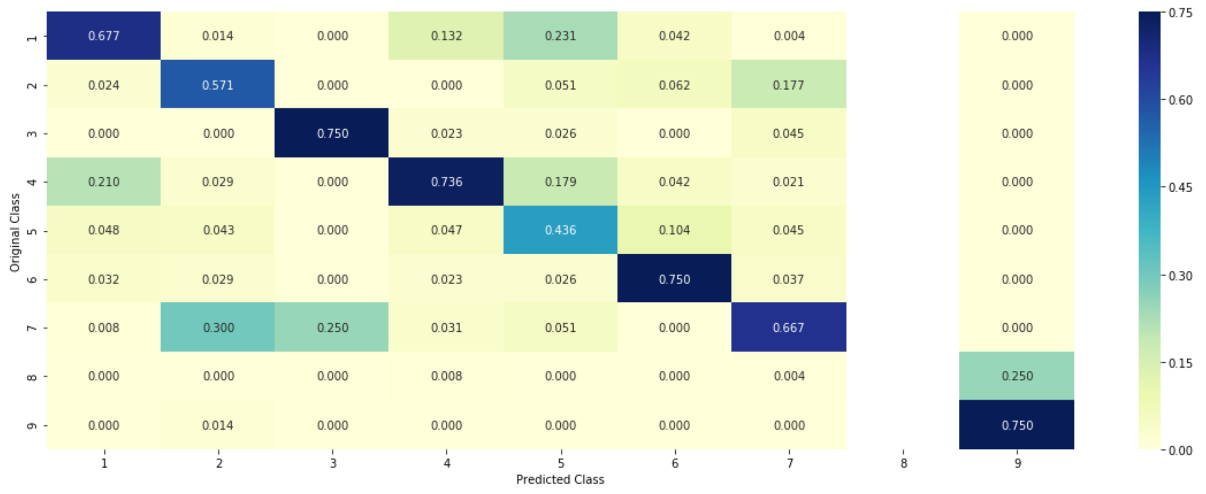
In [94]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_
rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

```
Log loss : 0.9919153581128131
Number of mis-classified points : 0.33383458646616543
-------------------- Confusion matrix --------------------
```



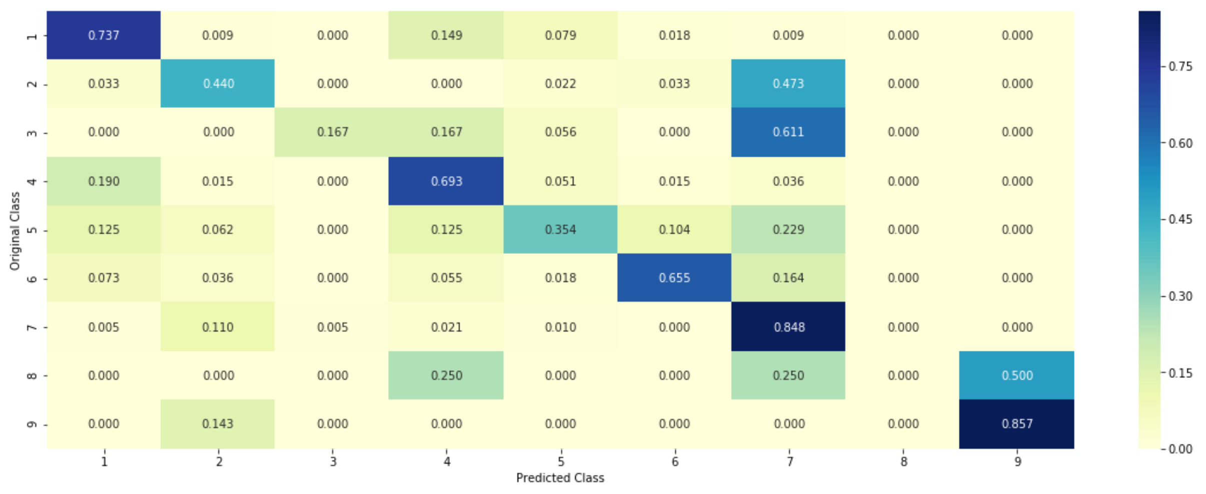
	1	2	3	4	5	6	7	8	9
1	84.000	1.000	0.000	17.000	9.000	2.000	1.000	0.000	0.000
2	3.000	40.000	0.000	0.000	2.000	3.000	43.000	0.000	0.000
3	0.000	0.000	3.000	3.000	1.000	0.000	11.000	0.000	0.000
4	26.000	2.000	0.000	95.000	7.000	2.000	5.000	0.000	0.000
5	6.000	3.000	0.000	6.000	17.000	5.000	11.000	0.000	0.000
6	4.000	2.000	0.000	3.000	1.000	36.000	9.000	0.000	0.000
7	1.000	21.000	1.000	4.000	2.000	0.000	162.000	0.000	0.000
8	0.000	0.000	0.000	1.000	0.000	0.000	1.000	0.000	2.000
9	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	6.000

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



	1	2	3	4	5	6	7	8	9
1	0.677	0.014	0.000	0.132	0.231	0.042	0.004		0.000
2	0.024	0.571	0.000	0.000	0.051	0.062	0.177		0.000
3	0.000	0.000	0.750	0.023	0.026	0.000	0.045		0.000
4	0.210	0.029	0.000	0.736	0.179	0.042	0.021		0.000
5	0.048	0.043	0.000	0.047	0.436	0.104	0.045		0.000
6	0.032	0.029	0.000	0.023	0.026	0.750	0.037		0.000
7	0.008	0.300	0.250	0.031	0.051	0.000	0.667		0.000
8	0.000	0.000	0.000	0.008	0.000	0.000	0.004		0.250
9	0.000	0.014	0.000	0.000	0.000	0.000	0.000		0.750

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



	1	2	3	4	5	6	7	8	9
1	0.737	0.009	0.000	0.149	0.079	0.018	0.009	0.000	0.000
2	0.033	0.440	0.000	0.000	0.022	0.033	0.473	0.000	0.000
3	0.000	0.000	0.167	0.167	0.056	0.000	0.611	0.000	0.000
4	0.190	0.015	0.000	0.693	0.051	0.015	0.036	0.000	0.000
5	0.125	0.062	0.000	0.125	0.354	0.104	0.229	0.000	0.000
6	0.073	0.036	0.000	0.055	0.018	0.655	0.164	0.000	0.000
7	0.005	0.110	0.005	0.021	0.010	0.000	0.848	0.000	0.000
8	0.000	0.000	0.000	0.250	0.000	0.000	0.250	0.000	0.500
9	0.000	0.143	0.000	0.000	0.000	0.000	0.000	0.000	0.857

Correctly Classified point

In [95]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty=
'l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_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_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1672 0.1572 0.0312 0.1138 0.1086 0.1003 0.
2854 0.0065 0.0296]]
Actual Class : 7
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

Incorrectly Classified point

In [96]:
```
test_point_index = 50
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_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.108  0.2367 0.0246 0.1145 0.0527 0.1108 0.
3173 0.0062 0.0291]]
Actual Class : 7
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

Conclusion

ngram_range(1,3), min_df=3, max_features=1000 used and below is the output

In [103]:
```
result = pd.DataFrame(columns = ["Model", "Train Log-loss", "CV Log-loss", "Te
st Log-loss", "No of Missclassified point", "Remarks"])
result = result.append(pd.DataFrame([["Naive Bayes", 0.693, 1.150, 1.247, "36.
09%", "Best Fit"]], columns = ["Model", "Train Log-loss", "CV Log-loss", "Test
Log-loss", "No of Missclassified point", "Remarks"]))
result = result.append(pd.DataFrame([["KNN", 0.600, 0.990, 1.101, "33.08%", "B
est Fit"]], columns = ["Model", "Train Log-loss", "CV Log-loss", "Test Log-los
s", "No of Missclassified point", "Remarks"]))
result = result.append(pd.DataFrame([["LR - Class Balancing", 0.648, 1.136, 1.
172, "38.72%", "Best Fit"]], columns = ["Model", "Train Log-loss", "CV Log-los
s", "Test Log-loss", "No of Missclassified point", "Remarks"]))
result = result.append(pd.DataFrame([["LR -Without Class Balancing", 0.557, 1.
165, 1.181, "40.03%", "Over Fit"]], columns = ["Model", "Train Log-loss", "CV
 Log-loss", "Test Log-loss", "No of Missclassified point", "Remarks"]))
result = result.append(pd.DataFrame([["LR -SVM", 0.774, 1.106, 1.180, "33.08%"
, "Best Fit"]], columns = ["Model", "Train Log-loss", "CV Log-loss", "Test Log
-loss", "No of Missclassified point", "Remarks"]))
result = result.append(pd.DataFrame([["RF -With OHE", 0.862, 1.186, 1.208, "4
0.22%", "Over Fit"]], columns = ["Model", "Train Log-loss", "CV Log-loss", "Te
st Log-loss", "No of Missclassified point", "Remarks"]))
result = result.append(pd.DataFrame([["RF -With RC", 0.066, 1.295, 1.339, "46.
80%", "Over Fit"]], columns = ["Model", "Train Log-loss", "CV Log-loss", "Test
Log-loss", "No of Missclassified point", "Remarks"]))
result = result.append(pd.DataFrame([["Stacking Models", 0.686, 1.102, 1.254,
"42.25%", "Over Fit"]], columns = ["Model", "Train Log-loss", "CV Log-loss",
"Test Log-loss", "No of Missclassified point", "Remarks"]))
result = result.append(pd.DataFrame([["Max Voting Classifier", 0.914, 1.160,
1.235, "40.30%", "Over Fit"]], columns = ["Model", "Train Log-loss", "CV Log-l
oss", "Test Log-loss", "No of Missclassified point", "Remarks"]))
result = result.append(pd.DataFrame([["LR with FE", 0.640, 0.975, 1.005, "36.3
9%", "Best Fit"]], columns = ["Model", "Train Log-loss", "CV Log-loss", "Test
 Log-loss", "No of Missclassified point", "Remarks"]))
result.reset_index(drop = True, inplace = True)
result
```

Out[103]:

	Model	Train Log-loss	CV Log-loss	Test Log-loss	No of Missclassified point	Remarks
0	Naive Bayes	0.693	1.150	1.247	36.09%	Best Fit
1	KNN	0.600	0.990	1.101	33.08%	Best Fit
2	LR - Class Balancing	0.648	1.136	1.172	38.72%	Best Fit
3	LR -Without Class Balancing	0.557	1.165	1.181	40.03%	Over Fit
4	LR -SVM	0.774	1.106	1.180	33.08%	Best Fit
5	RF -With OHE	0.862	1.186	1.208	40.22%	Over Fit
6	RF -With RC	0.066	1.295	1.339	46.80%	Over Fit
7	Stacking Models	0.686	1.102	1.254	42.25%	Over Fit
8	Max Voting Classifier	0.914	1.160	1.235	40.30%	Over Fit
9	LR with FE	0.640	0.975	1.005	36.39%	Best Fit

In order to reduce test log loss to below 1, made changes as below

ngram_range(1,4), min_df=5, max_features=3000 we got minimum test log-loss is 0.991 and minimum misclassified points as 33.38% for LR with FE

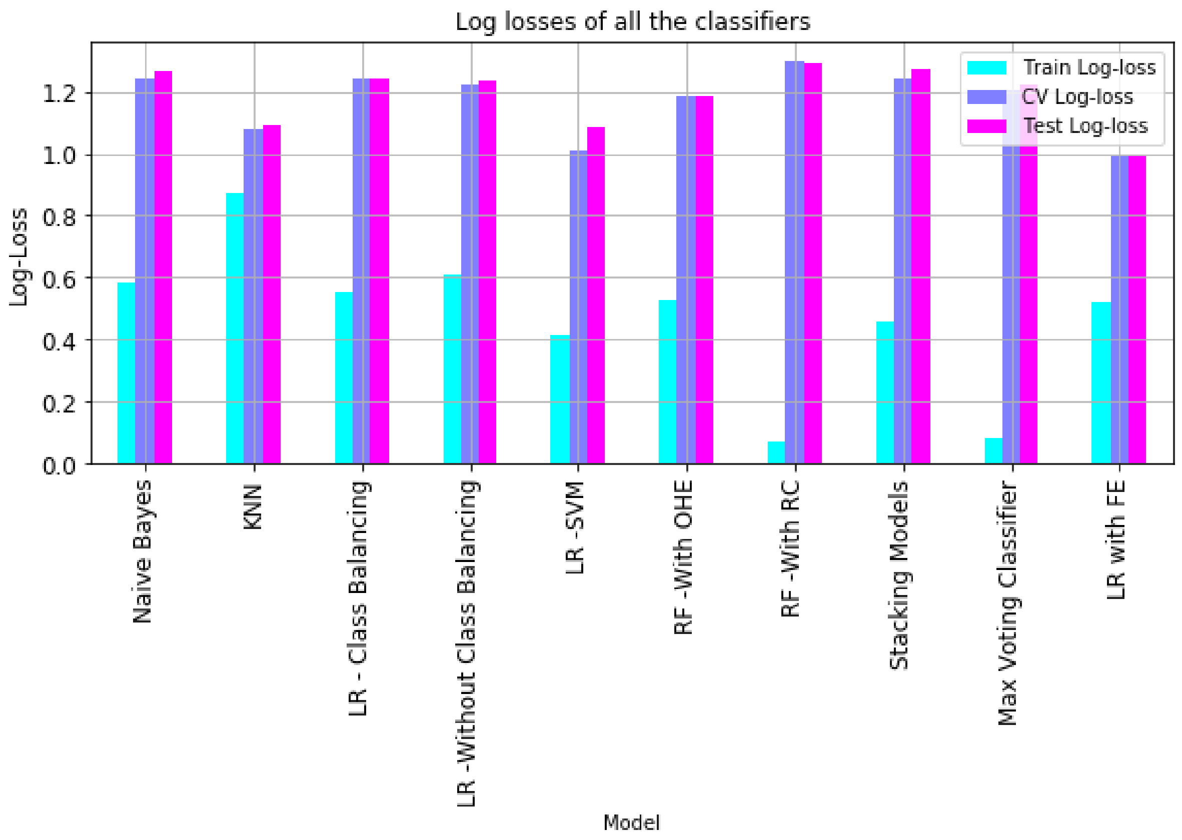
In [104]:
```
result = pd.DataFrame(columns = ["Model", "Train Log-loss", "CV Log-loss", "Te
st Log-loss", "No of Missclassified point", "Remarks"])
result = result.append(pd.DataFrame([["Naive Bayes", 0.582, 1.242, 1.264, "40.
03%", "Over Fit"]], columns = ["Model", "Train Log-loss", "CV Log-loss", "Test
Log-loss", "No of Missclassified point", "Remarks"]))
result = result.append(pd.DataFrame([["KNN", 0.869, 1.081, 1.088, "37.03%", "B
est Fit"]], columns = ["Model", "Train Log-loss", "CV Log-loss", "Test Log-los
s", "No of Missclassified point", "Remarks"]))
result = result.append(pd.DataFrame([["LR - Class Balancing", 0.554, 1.240, 1.
242, "38.90%", "Best Fit"]], columns = ["Model", "Train Log-loss", "CV Log-los
s", "Test Log-loss", "No of Missclassified point", "Remarks"]))
result = result.append(pd.DataFrame([["LR -Without Class Balancing", 0.606, 1.
222, 1.235, "40.97%", "Over Fit"]], columns = ["Model", "Train Log-loss", "CV
 Log-loss", "Test Log-loss", "No of Missclassified point", "Remarks"]))
result = result.append(pd.DataFrame([["LR -SVM", 0.417, 1.008, 1.086, "32.14%"
, "Best Fit"]], columns = ["Model", "Train Log-loss", "CV Log-loss", "Test Log
-loss", "No of Missclassified point", "Remarks"]))
result = result.append(pd.DataFrame([["RF -With OHE", 0.529, 1.185, 1.182, "3
8.90%", "Best Fit"]], columns = ["Model", "Train Log-loss", "CV Log-loss", "Te
st Log-loss", "No of Missclassified point", "Remarks"]))
result = result.append(pd.DataFrame([["RF -With RC", 0.069, 1.296, 1.292, "41.
91%", "Over Fit"]], columns = ["Model", "Train Log-loss", "CV Log-loss", "Test
Log-loss", "No of Missclassified point", "Remarks"]))
result = result.append(pd.DataFrame([["Stacking Models", 0.459, 1.243, 1.275,
"38.49%", "Over Fit"]], columns = ["Model", "Train Log-loss", "CV Log-loss",
"Test Log-loss", "No of Missclassified point", "Remarks"]))
result = result.append(pd.DataFrame([["Max Voting Classifier", 0.082, 1.206,
1.224, "37.29%", "Best Fit"]], columns = ["Model", "Train Log-loss", "CV Log-l
oss", "Test Log-loss", "No of Missclassified point", "Remarks"]))
result = result.append(pd.DataFrame([["LR with FE", 0.523, 0.990, 0.991, "33.3
8%", "Best Fit"]], columns = ["Model", "Train Log-loss", "CV Log-loss", "Test
 Log-loss", "No of Missclassified point", "Remarks"]))
result.reset_index(drop = True, inplace = True)
result
```

Out[104]:

	Model	Train Log-loss	CV Log-loss	Test Log-loss	No of Missclassified point	Remarks
0	Naive Bayes	0.582	1.242	1.264	40.03%	Over Fit
1	KNN	0.869	1.081	1.088	37.03%	Best Fit
2	LR - Class Balancing	0.554	1.240	1.242	38.90%	Best Fit
3	LR -Without Class Balancing	0.606	1.222	1.235	40.97%	Over Fit
4	LR -SVM	0.417	1.008	1.086	32.14%	Best Fit
5	RF -With OHE	0.529	1.185	1.182	38.90%	Best Fit
6	RF -With RC	0.069	1.296	1.292	41.91%	Over Fit
7	Stacking Models	0.459	1.243	1.275	38.49%	Over Fit
8	Max Voting Classifier	0.082	1.206	1.224	37.29%	Best Fit
9	LR with FE	0.523	0.990	0.991	33.38%	Best Fit

In [101]:
```
bar_result = result.drop(["No of Missclassified point","Remarks"], axis = 1)
```

In [102]:
```
bar_result.plot(x = "Model", kind = "bar", figsize = (10, 4), grid = True, fon
tsize = 12, colormap="cool")
plt.title("Log losses of all the classifiers", fontsize = 12)
plt.ylabel("Log-Loss", fontsize = 12)
plt.show()
```



Summary

1. Applied all the models with tf-idf feature
2. Maximum 1000 words used
3. NB - OHE is applied
4. KNN - Response coding is applied
5. LR with count vectorizer and unigram and bigram and OHE is applied, with class balancing
6. LR with count vectorizer and unigram and bigram and OHE is applied, without class balancing
7. Lr SVM with class balancing with OHE is applied
8. RF with OHE
9. RF with response coding
10. stacking the models with OHE
11. Maximum voting classifier with OHE
12. LR with FE