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
```

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

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

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

3.1.2. Reading Text Data

In [3]:
```
# note the seprator in this file
data_text =pd.read_csv("training/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[3]:

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

3.1.3. Preprocessing of text

In [4]:
```
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))


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

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

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

In [5]:
```
#text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_time, "seconds")
```

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 202.937976066 seconds
```

In [6]:
```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[6]:

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

In [7]:
```
result[result.isnull().any(axis=1)]
```

Out[7]:

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

In [8]:
```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

In [9]:
```
result[result['ID']==1109]
```

Out[9]:

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

3.1.4. Test, Train and Cross Validation Split

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

In [10]:
```
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

# split the data into test and train by maintaining same distribution of 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

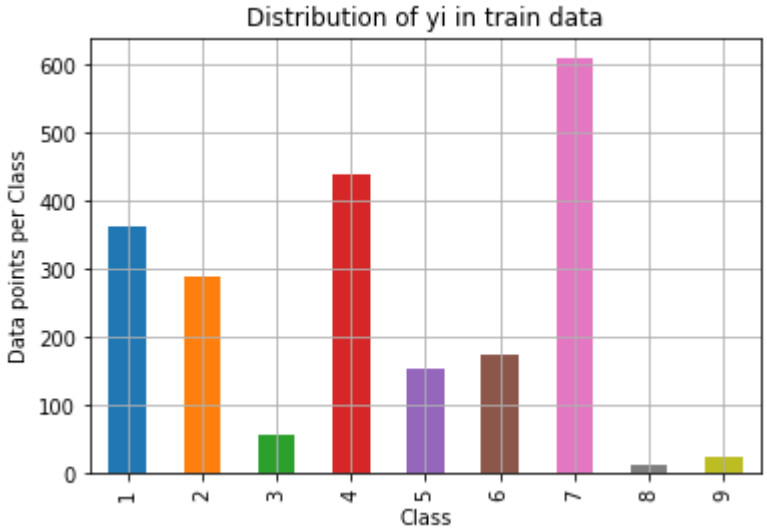
In [11]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

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

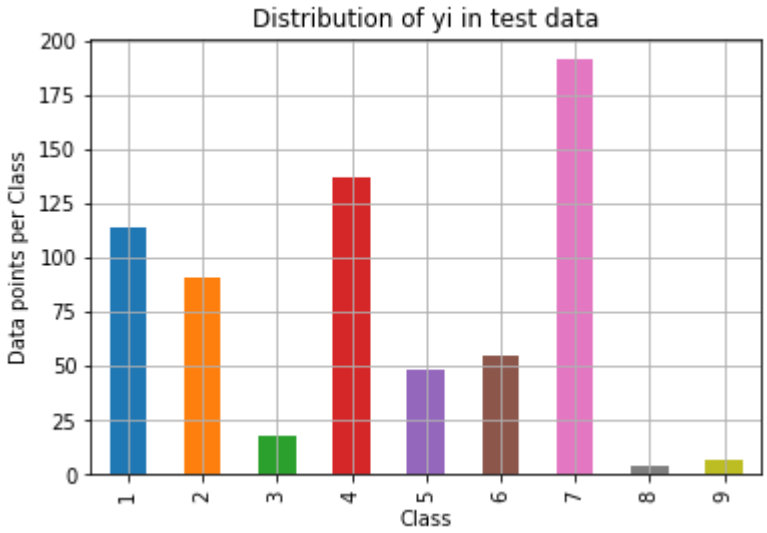
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [12]:
```
# it returns a dict, keys as class labels and values as the number of data 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
```

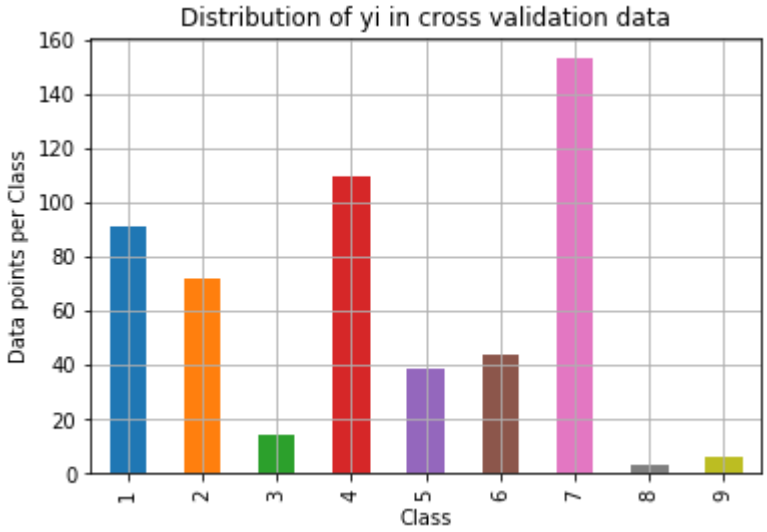
```
g order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.val
ues[i], '(', np.round((cv_class_distribution.values[i]/cv_df.shape[0]*100), 3
), '%)')
```




```
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

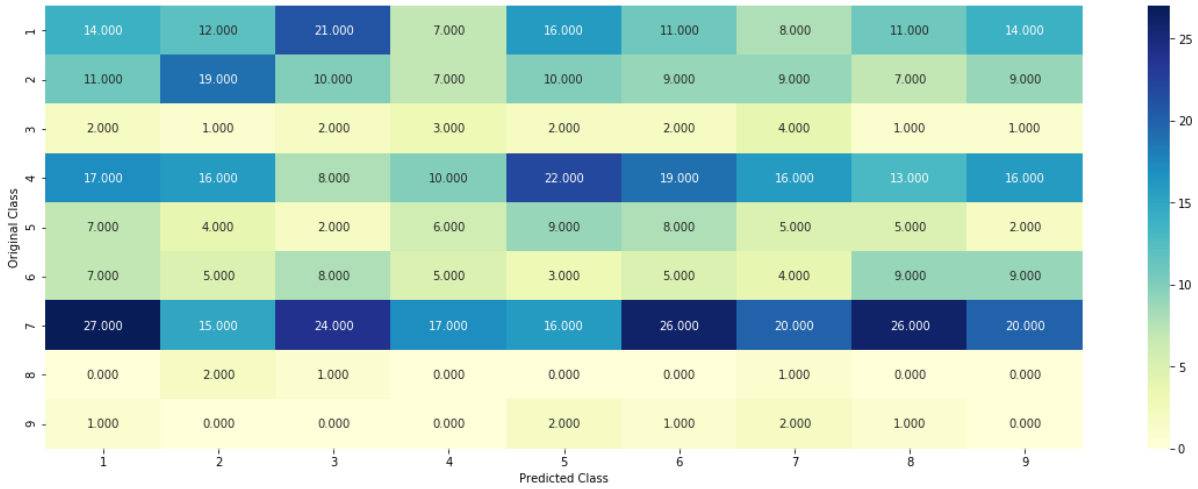
In a 'Random' Model, we generate the NINE class probabilites randomly such that they sum to 1.

In [13]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i 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 [14]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by 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 2.471068847276733
Log loss on Test Data using Random Model 2.447873317983623
-------------------- Confusion matrix --------------------
```




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


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

3.3 Univariate Analysis

In [15]:
```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feature in
 train data dataframe
# build a vector (1*9) , the first element = (number of times it occured in 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, 0.07284768211920 5295, 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?

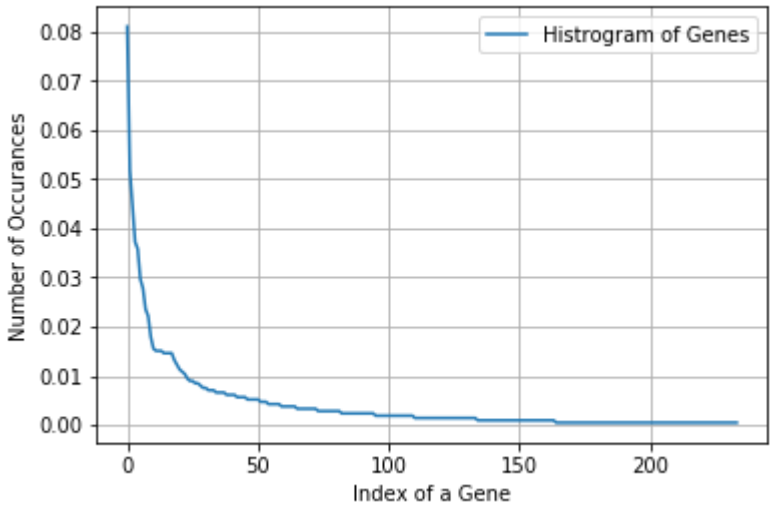
In [16]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 234
BRCA1     172
TP53      110
EGFR       95
BRCA2      79
PTEN       76
KIT        63
BRAF       59
ERBB2      50
ALK        47
PDGFRA     38
Name: Gene, dtype: int64
```

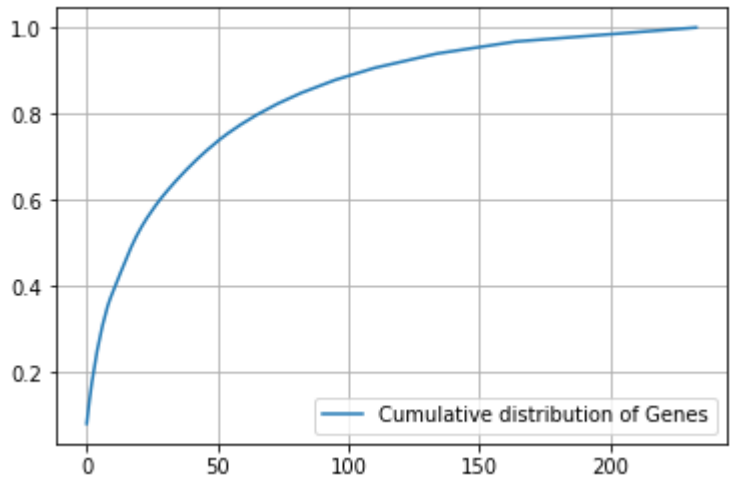
In [17]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes
 in the train data, and they are distibuted as follows",)
```

Ans: There are 234 different categories of genes in the train data, and they are distibuted as follows

In [18]:
```
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histrogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [19]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

In [20]:
```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", 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 [21]:
```
print("train_gene_feature_responseCoding is converted feature using respone co
ding method. The shape of gene feature:", train_gene_feature_responseCoding.sh
ape)
```

```
train_gene_feature_responseCoding is converted feature using respone coding m
ethod. The shape of gene feature: (2124, 9)
```

In [22]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
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 [23]:
```
train_df['Gene'].head()
```

Out[23]:
```
2604    BRCA1
351     EPCAM
1510      ALK
618     FBXW7
62      PTPRT
Name: Gene, dtype: object
```

In [24]:
```
gene_vectorizer.get_feature_names()
```

Out[24]:
```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl1',
 'asxl2',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'axin1',
 'axl',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk6',
 'cdkn1a',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctla4',
 'ctnnb1',
 'ddr2',
 'dicer1',
 'dnmt3a',
```

```
                'dnmt3b',
                'egfr',
                'eif1ax',
                'elf3',
                'ep300',
                'epas1',
                'epcam',
                'erbb2',
                'erbb3',
                'erbb4',
                'ercc2',
                'ercc3',
                'ercc4',
                'erg',
                'errfi1',
                'esr1',
                'etv1',
                'etv6',
                'ewsr1',
                'ezh2',
                'fam58a',
                'fanca',
                'fancc',
                'fat1',
                'fbxw7',
                'fgf3',
                'fgf4',
                'fgfr1',
                'fgfr2',
                'fgfr3',
                'fgfr4',
                'flt1',
                'flt3',
                'foxa1',
                'foxl2',
                'foxp1',
                'fubp1',
                'gata3',
                'gli1',
                'gnaq',
                'gnas',
                'h3f3a',
                'hla',
                'hnf1a',
                'hras',
                'idh1',
                'idh2',
                'igf1r',
                'ikbke',
                'inpp4b',
                'jak1',
                'jak2',
                'kdm5a',
                'kdm5c',
                'kdm6a',
                'kdr',
                'keap1',
```

```
'kit',
'klf4',
'kmt2c',
'knstrn',
'kras',
'lats1',
'lats2',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
'mapk1',
'mdm2',
'mdm4',
'med12',
'mef2b',
'met',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'myd88',
'myod1',
'ncor1',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',
'nkx2',
'notch1',
'notch2',
'npm1',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'nup93',
'pak1',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pik3r3',
'pim1',
'pms1',
'pms2',
'pole',
'ppm1d',
'ppp2r1a',
'ppp6c',
```

```
'prdm1',
'ptch1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rab35',
'rac1',
'rad21',
'rad50',
'rad51c',
'rad51d',
'raf1',
'rara',
'rasa1',
'rb1',
'rbm10',
'ret',
'rheb',
'rhoa',
'rit1',
'rnf43',
'ros1',
'rras2',
'runx1',
'rxra',
'rybp',
'sdhb',
'setd2',
'sf3b1',
'shoc2',
'shq1',
'smad2',
'smad3',
'smad4',
'smarca4',
'smarcb1',
'smo',
'sos1',
'sox9',
'spop',
'src',
'stag2',
'stat3',
'stk11',
'tcf7l2',
'tert',
'tet1',
'tet2',
'tgfbr1',
'tgfbr2',
'tmprss2',
'tp53',
'tp53bp1',
'tsc1',
'tsc2',
'u2af1',
```

```
 'vhl',
 'whsc1l1',
 'xpo1',
 'xrcc2',
 'yap1']
```

In [25]:
```
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, 233)
```

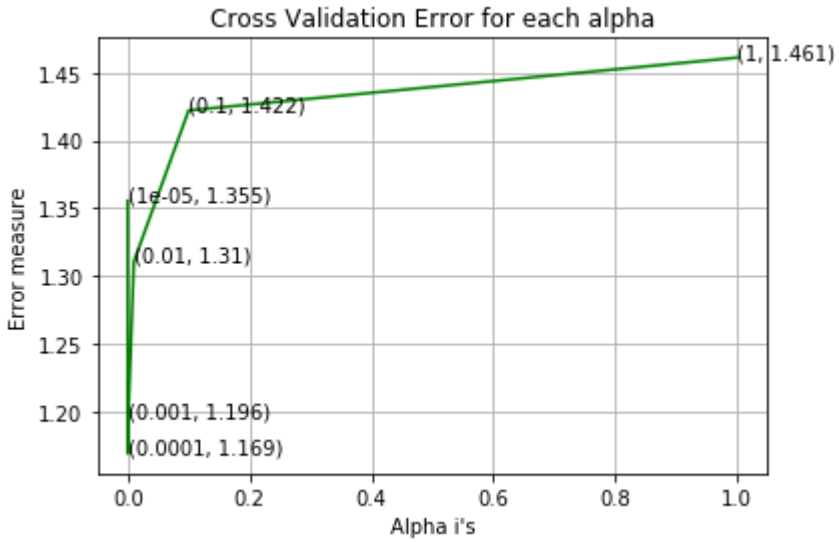
Q4. How good is this gene feature in predicting y_i?

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

In [26]:
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/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: 1.3553657105535337
For values of alpha =  0.0001 The log loss is: 1.1687417391482027
For values of alpha =  0.001 The log loss is: 1.1962959003267652
For values of alpha =  0.01 The log loss is: 1.3104675877365644
For values of alpha =  0.1 The log loss is: 1.4221261880499074
For values of alpha =  1 The log loss is: 1.4610918417637557
```




```
For values of best alpha =  0.0001 The train log loss is: 1.0525109632684866
For values of best alpha =  0.0001 The cross validation log loss is: 1.168741
7391482027
For values of best alpha =  0.0001 The test log loss is: 1.228532666375545
```

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

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

In [27]:
```
print("Q6. How many data points in Test and CV datasets are covered by the ",
unique_genes.shape[0], " genes in train dataset?")

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

print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(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  234  gen
es in train dataset?
Ans
1. In test data 647 out of 665 : 97.29323308270676
2. In cross validation data 515 out of  532 : 96.80451127819549
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

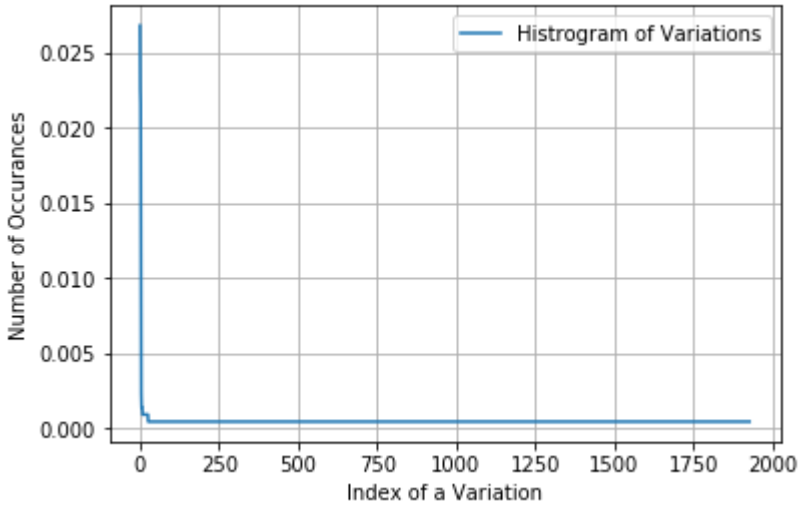
In [28]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1928
Truncating_Mutations    57
Amplification           48
Deletion                46
Fusions                 21
Overexpression           5
E17K                     3
Q61R                     3
G12V                     3
Q22K                     2
G12S                     2
Name: Variation, dtype: int64
```

In [29]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of v
ariations in the train data, and they are distibuted as follows",)
```

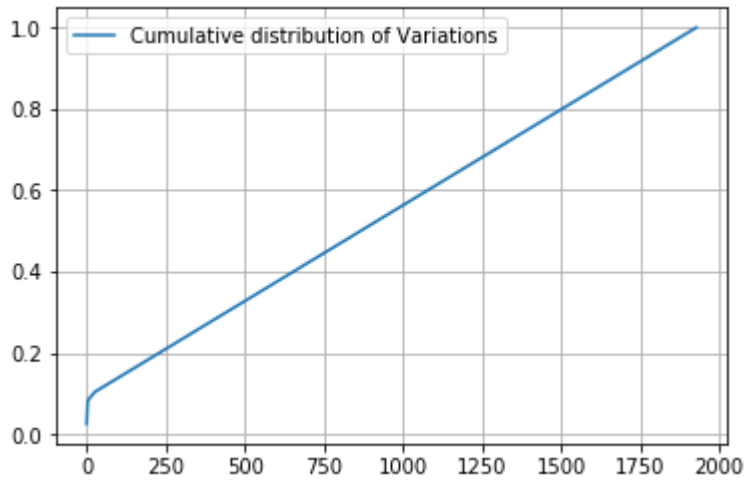
```
Ans: There are 1928 different categories of variations in the train data, and
they are distibuted as follows
```

In [30]:
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



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

```
[0.02683616 0.04943503 0.07109228 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [32]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_df))
```

In [33]:
```
print("train_variation_feature_responseCoding is a converted feature using the response coding method. The shape of Variation feature:", train_variation_feature_responseCoding.shape)
```

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

In [34]:
```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [35]:
```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Variation feature:", train_variation_feature_onehotCoding.shape)
```

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Variation feature: (2124, 1959)
```

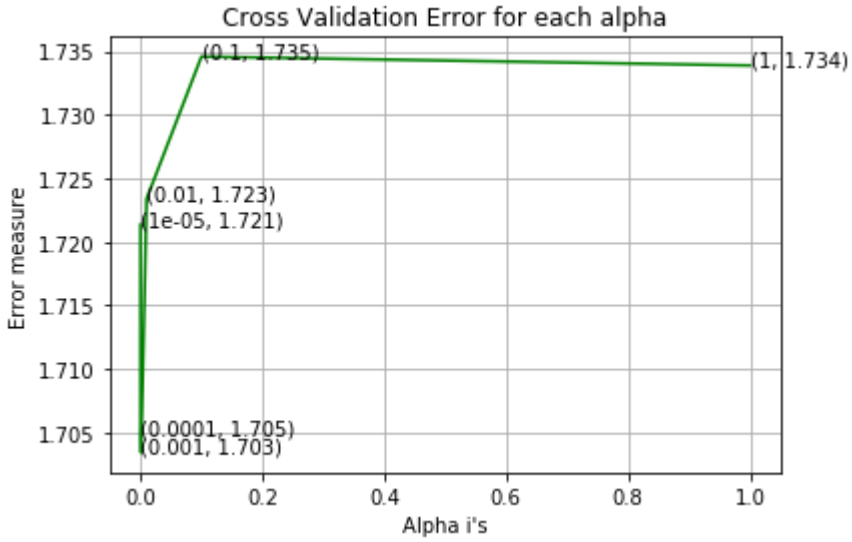
Q10. How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

In [36]:
```
alpha = [10 ** x for x in range(-5, 1)]

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

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

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


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

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

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

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


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

predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
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: 1.7213521513664671
For values of alpha =  0.0001 The log loss is: 1.7048178883123368
For values of alpha =  0.001 The log loss is: 1.7034224136478455
For values of alpha =  0.01 The log loss is: 1.723391487080406
For values of alpha =  0.1 The log loss is: 1.7345804965433111
For values of alpha =  1 The log loss is: 1.7338844740230923
```




```
For values of best alpha =  0.001 The train log loss is: 1.038909563435895
For values of best alpha =  0.001 The cross validation log loss is: 1.7034224
136478455
For values of best alpha =  0.001 The test log loss is: 1.7091261245846272
```

Q11. Is the Variation feature stable across all the data sets (Test, Train, Cross validation)?

Ans. Not sure! But lets be very sure using the below analysis.

In [37]:
```
print("Q12. How many data points are covered by total ", unique_variations.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  1928  genes in test and cross
validation data sets?
Ans
1. In test data 64 out of 665 : 9.624060150375941
2. In cross validation data 63 out of  532 : 11.842105263157894
```

3.2.3 Univariate Analysis on Text Feature

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

In [38]:
```
# cls_text is a data frame
# for every row in data fram consider the 'TEXT'
# split the words by space
# make a dict with those words
# increment its count whenever we see that word

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

In [39]:
```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_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
```

Tf-idf

In [40]:
```
# building a CountVectorizer with all the words that occured minimum 3 times i
n train data
text_vectorizer = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_features=5
000)
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 : 5000
```

In [41]:
```
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = train_df[train_df['Class']==i]
    # build a word dict based on the words in that class
    dict_list.append(extract_dictionary_paddle(cls_text))
    # append it to dict_list

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


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

In [42]:
```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
test_text_feature_responseCoding  = get_text_responsecoding(test_df)
cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)
```

In [43]:
```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.T/train
_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_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 [44]:
```
# 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 [45]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] ,
reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

In [46]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({3.5719233325108375: 23, 2.870132397059507: 13, 2.4117762788026513: 1
1, 1.8070595458221017: 10, 5.216936450760817: 8, 3.067245532892035: 8, 7.2264
52947165024: 6, 10.839679420747537: 5, 4.0692895637750555: 4, 2.6357621840466
634: 4, 2.168017194578289: 4, 1.8061371778453923: 4, 4.986504240917422: 3, 3.
5450979768183695: 3, 3.118421113689724: 3, 2.8481621535479715: 3, 2.354680444
625185: 3, 2.3467485884069768: 3, 2.3397839751082183: 3, 8.394514468081836:
2, 8.053531204104347: 2, 7.731084685764383: 2, 7.274615415733761: 2, 6.887898
659063268: 2, 6.777366951191471: 2, 6.673008091574861: 2, 6.607488318852: 2,
5.94651686394017: 2, 5.845337422328721: 2, 5.425868491729922: 2, 5.1892920704
35329: 2, 4.7647835939815195: 2, 4.013268748246839: 2, 3.986786110232097: 2,
3.9148685874015063: 2, 3.8034527253345307: 2, 3.5497066562500033: 2, 3.547965
9099234575: 2, 3.206679487609694: 2, 2.8019048823540484: 2, 2.678677928871234
6: 2, 2.6654204487653304: 2, 2.643689449882512: 2, 2.6321440953964075: 2, 2.5
65920226741367: 2, 2.3821415257270773: 2, 2.37677192637514: 2, 2.369837453972
7824: 2, 2.3563024825015813: 2, 2.3521510020802694: 2, 2.293611068720473: 2,
2.28640208592776: 2, 2.235879729861791: 2, 2.13863754240571: 2, 2.12571773682
30975: 2, 2.07942800353474: 2, 2.0486543474208276: 2, 1.9994193313775326: 2,
1.921066847117741: 2, 1.7074487915454537: 2, 1.7059083281230494: 2, 1.5606117
329605975: 2, 176.34491238767544: 1, 117.9085136034322: 1, 107.5886430132061
4: 1, 87.46164542725901: 1, 83.56211134352839: 1, 81.3709931776006: 1, 81.204
93004619001: 1, 78.4249569320506: 1, 78.08873242942393: 1, 77.89776568673362:
1, 74.24860049651066: 1, 71.98570923019096: 1, 67.67841369378955: 1, 66.43274
586519331: 1, 62.75535325334309: 1, 60.407428756568784: 1, 58.97036694206810
4: 1, 55.29427036841773: 1, 54.82421278218734: 1, 53.99454885255048: 1, 53.82
7659476199685: 1, 52.051338494185366: 1, 50.691459014087904: 1, 47.5710913382
1067: 1, 46.60786876669868: 1, 45.617363680200675: 1, 45.251347418692184: 1,
44.6905988607679: 1, 44.30346290834678: 1, 43.98175661178057: 1, 43.486878520
54303: 1, 43.41293260894629: 1, 43.11557052913189: 1, 43.11318074528229: 1, 4
3.07442389373769: 1, 42.443384771102124: 1, 41.709488554368704: 1, 40.7438134
235652: 1, 39.96883917205457: 1, 39.03903338678093: 1, 38.87161085959539: 1,
38.381527039440435: 1, 37.715728920149736: 1, 36.82088131328698: 1, 36.143286
11074154: 1, 35.28199640219446: 1, 35.25115218829195: 1, 34.06400152855661:
1, 33.3535428623908: 1, 32.39421732230367: 1, 32.15028193972094: 1, 31.535500
938473394: 1, 31.36246988322446: 1, 31.25374719711086: 1, 31.086486789767246:
1, 30.976658732038924: 1, 30.679920222113214: 1, 30.394394795120174: 1, 30.18
23559032671: 1, 29.39153312700934: 1, 29.347324760224716: 1, 29.2130904520744
4: 1, 29.120926509865154: 1, 28.80819264466182: 1, 28.77252843600485: 1, 28.7
34639827023223: 1, 28.706809936899166: 1, 28.451538972853943: 1, 28.420427868
49432: 1, 28.417289451829927: 1, 27.78548131090132: 1, 27.395734670445876: 1,
26.868055393064676: 1, 26.607335031205707: 1, 26.556952196821186: 1, 26.49992
7566081094: 1, 26.481923351158166: 1, 26.44332378970045: 1, 25.9797369440043
6: 1, 25.85817313988762: 1, 25.83258751012022: 1, 25.66868090977552: 1, 25.43
8312147003: 1, 25.422177305414777: 1, 25.279937779044577: 1, 25.0644244263865
23: 1, 25.036556424190522: 1, 25.021744384841856: 1, 24.80947419180002: 1, 2
4.608031716379546: 1, 24.21963973860481: 1, 24.21228488380607: 1, 24.14419518
9217978: 1, 24.026060377168182: 1, 24.00248365603775: 1, 23.88356404159339:
1, 23.779437451210512: 1, 23.17227534302357: 1, 23.055761118263497: 1, 22.905
633463617722: 1, 22.656762778340177: 1, 22.396368419484247: 1, 22.32429289701
1716: 1, 22.314952906559554: 1, 22.221968191805615: 1, 22.188486070642575: 1,
22.176301134738985: 1, 22.147410013350424: 1, 22.144739766456844: 1, 21.97936
5941666973: 1, 21.82394498208695: 1, 21.776017490826028: 1, 21.73263419737866
2: 1, 21.556982495901625: 1, 21.51384966329537: 1, 21.487160092224066: 1, 21.
386690002586274: 1, 21.385562449165345: 1, 21.22167551715612: 1, 21.162674991
15691: 1, 20.85364650484459: 1, 20.822512613565106: 1, 20.73285202743506: 1,
20.66697407126805: 1, 20.56747252128296: 1, 20.497537750840074: 1, 20.2941058
100006: 1, 20.26700454783719: 1, 20.236058172511665: 1, 20.109715032485187:
1, 19.996844782298545: 1, 19.935864930681046: 1, 19.914087393330917: 1, 19.88
```

091786598293: 1, 19.863654369173332: 1, 19.837916859288256: 1, 19.78112174742
8816: 1, 19.687960917308885: 1, 19.550068328262437: 1, 19.515036837643926: 1,
19.19881670858201: 1, 19.157001937570122: 1, 18.960192709448037: 1, 18.941732
944624466: 1, 18.634518923979876: 1, 18.57812173133096: 1, 18.51708833774622
3: 1, 18.482007720733467: 1, 18.469908730118107: 1, 18.298560956525098: 1, 1
8.20669919915299: 1, 18.154623751182125: 1, 18.06162013584651: 1, 18.05725941
83183: 1, 17.99599326671676: 1, 17.95168464684396: 1, 17.86190585125351: 1, 1
7.847005722456128: 1, 17.59807741254877: 1, 17.57952095026376: 1, 17.57052196
387717: 1, 17.515189068132308: 1, 17.45978317595441: 1, 17.425551263789394:
1, 17.380078691509315: 1, 17.283778257409956: 1, 17.275515440578594: 1, 17.22
5217949824284: 1, 17.031415310952802: 1, 16.982031451539985: 1, 16.9807081688
47336: 1, 16.953553650465146: 1, 16.91791650442677: 1, 16.861795383901057: 1,
16.852265865856406: 1, 16.8460247909185: 1, 16.842321880277943: 1, 16.8411282
3513848: 1, 16.80688673445558: 1, 16.800036694521506: 1, 16.79930906952077:
1, 16.761890419219917: 1, 16.747637289113914: 1, 16.619468191764025: 1, 16.61
832959713289: 1, 16.468987406771873: 1, 16.426678095808786: 1, 16.40915862964
3045: 1, 16.327374471381905: 1, 16.319197998125198: 1, 16.22428263049169: 1,
16.21087100778749: 1, 16.1386484572571: 1, 16.133066114296064: 1, 16.07900231
925313: 1, 16.06128095858148: 1, 16.04146280518396: 1, 16.0143206830725: 1, 1
6.00594177047896: 1, 15.994745432656183: 1, 15.913303938096925: 1, 15.8958647
58417378: 1, 15.786156441806819: 1, 15.762918585006465: 1, 15.70330309884612
6: 1, 15.684259459777753: 1, 15.5931794901748: 1, 15.549000354155314: 1, 15.5
30037665125159: 1, 15.475681985647578: 1, 15.42729254904794: 1, 15.4201602959
02285: 1, 15.35010771582652: 1, 15.348739944079625: 1, 15.339559558956328: 1,
15.319399515328708: 1, 15.265308159233296: 1, 15.169944138894477: 1, 15.15350
013619202: 1, 15.118908273739152: 1, 15.111259388011678: 1, 15.08265772381057
2: 1, 15.00540817436455: 1, 14.976968097218181: 1, 14.958009598384665: 1, 14.
92783190113381: 1, 14.89244421051559: 1, 14.879650126610048: 1, 14.8773006706
47328: 1, 14.876126212591368: 1, 14.86798025710482: 1, 14.836959382397568: 1,
14.803091685920215: 1, 14.701630125313292: 1, 14.694762767356172: 1, 14.65169
442720798: 1, 14.64629401007018: 1, 14.61449422188812: 1, 14.60836302852888:
1, 14.594031545780076: 1, 14.571062545304738: 1, 14.563121154232254: 1, 14.50
9956252278158: 1, 14.500855309170916: 1, 14.483513837194138: 1, 14.4729167322
9364: 1, 14.462943315099395: 1, 14.45732311241102: 1, 14.45700914819594: 1, 1
4.44109650949898: 1, 14.419449276038236: 1, 14.386937052036187: 1, 14.3232974
4051728: 1, 14.315594841316258: 1, 14.312407980151729: 1, 14.221496516932062:
1, 14.21297043134994: 1, 14.205171063567896: 1, 14.159767291526741: 1, 14.085
768519978174: 1, 14.061772877898663: 1, 14.04703317704751: 1, 14.028822923438
833: 1, 13.9796925343236: 1, 13.940963416875281: 1, 13.925659486930915: 1, 1
3.915390414230307: 1, 13.877202351084087: 1, 13.842825804464017: 1, 13.799418
742885434: 1, 13.752452423022374: 1, 13.725008234806102: 1, 13.71734686323518
4: 1, 13.707535536703443: 1, 13.704988989289493: 1, 13.703298907865872: 1, 1
3.697638256971068: 1, 13.67698579735868: 1, 13.65593341442733: 1, 13.65413356
1162705: 1, 13.653934757529944: 1, 13.63756844126438: 1, 13.632594010754067:
1, 13.604783224488557: 1, 13.582348782544955: 1, 13.578835081810482: 1, 13.56
1028360760243: 1, 13.516696618535958: 1, 13.47090207486841: 1, 13.45977164615
0507: 1, 13.448964251692876: 1, 13.373655646588036: 1, 13.331680579166639: 1,
13.324485943744415: 1, 13.158328196464472: 1, 13.137062240041868: 1, 13.12581
256742827: 1, 13.085115592879372: 1, 13.066438305022517: 1, 13.01951429753949
4: 1, 13.018478284832339: 1, 12.970470394674178: 1, 12.955343511244862: 1, 1
2.929385717483536: 1, 12.877874906445204: 1, 12.871998528065513: 1, 12.866828
660121179: 1, 12.858943851010926: 1, 12.841960412595357: 1, 12.84158508732074
6: 1, 12.807137454995857: 1, 12.780602431098576: 1, 12.711280432315197: 1, 1
2.701493933212314: 1, 12.695511635671762: 1, 12.690695918445986: 1, 12.679424
36564165: 1, 12.669460306892692: 1, 12.63795829066244: 1, 12.601044073471641:
1, 12.580638991082974: 1, 12.541699359133236: 1, 12.516478689376422: 1, 12.47
0922491693502: 1, 12.447121306668121: 1, 12.445091332214737: 1, 12.4261388432

67323: 1, 12.418365649795502: 1, 12.404894074534317: 1, 12.36073233752771: 1,
12.321992773184782: 1, 12.29588232236702: 1, 12.243922141973021: 1, 12.216222
99966201: 1, 12.204391589865205: 1, 12.191759029656286: 1, 12.17999261584251
3: 1, 12.14135906834964: 1, 12.134143240207045: 1, 12.126527579558752: 1, 12.
108722368941239: 1, 12.0998599315259: 1, 12.09624266898532: 1, 12.04932040986
412: 1, 12.033915152571831: 1, 12.01962599810711: 1, 12.01305746848673: 1, 1
1.958776377476045: 1, 11.93659699000231: 1, 11.905456633962705: 1, 11.9049858
98547071: 1, 11.893575923742855: 1, 11.85573925056179: 1, 11.854139062859963:
1, 11.84969717673642: 1, 11.828505334131801: 1, 11.806842083786817: 1, 11.799
43090800851: 1, 11.792006324440871: 1, 11.756938167783716: 1, 11.754082341056
405: 1, 11.735645369893918: 1, 11.706465369236263: 1, 11.67864845953425: 1, 1
1.64845746624222: 1, 11.592984794657864: 1, 11.577099379739082: 1, 11.5762938
28451107: 1, 11.575191128268772: 1, 11.549824330811752: 1, 11.54375994980952
5: 1, 11.511115993116842: 1, 11.481305497776109: 1, 11.433074626255236: 1, 1
1.432651460394503: 1, 11.33809932439345: 1, 11.336954670351776: 1, 11.3198625
25837504: 1, 11.295741858882044: 1, 11.294318712693002: 1, 11.28751869984410
1: 1, 11.273725257419311: 1, 11.262658737745095: 1, 11.261929297442158: 1, 1
1.250884029024439: 1, 11.249651704960996: 1, 11.22896604771353: 1, 11.2241275
8840295: 1, 11.208916681534214: 1, 11.184927078102806: 1, 11.17961309032952:
1, 11.169975966938228: 1, 11.161796496630242: 1, 11.161141721691042: 1, 11.15
783497378164: 1, 11.148804303892804: 1, 11.090555493422334: 1, 11.07214319305
1927: 1, 11.056171893613513: 1, 11.032356666211692: 1, 11.027922641638142: 1,
10.94884610870902: 1, 10.930100818994706: 1, 10.92239447865869: 1, 10.9196658
771601: 1, 10.88959299685274: 1, 10.884088810114608: 1, 10.880099879108547:
1, 10.870126535834105: 1, 10.869773058987311: 1, 10.788938061245307: 1, 10.78
3935144272997: 1, 10.781939188804937: 1, 10.780538282505653: 1, 10.7795875555
2441: 1, 10.763017317267776: 1, 10.759392658016605: 1, 10.757993511098018: 1,
10.737378134837032: 1, 10.728482060330634: 1, 10.728477553062993: 1, 10.69491
151505838: 1, 10.677592251217483: 1, 10.670681734395016: 1, 10.66606288279780
8: 1, 10.63616371562056: 1, 10.596556939121037: 1, 10.594117541167169: 1, 10.
573623756274777: 1, 10.554216454943926: 1, 10.55315573231163: 1, 10.504207750
105472: 1, 10.482890148950407: 1, 10.48019971116795: 1, 10.478110094939453:
1, 10.46098872754008: 1, 10.458127813826291: 1, 10.45812218260981: 1, 10.4174
22221625042: 1, 10.41653497563037: 1, 10.4148016720031: 1, 10.40189996142089
9: 1, 10.38046177629452: 1, 10.36221575930512: 1, 10.35527128784892: 1, 10.34
889692272471: 1, 10.347992540780837: 1, 10.347009784727083: 1, 10.33814222893
1716: 1, 10.331140947648658: 1, 10.32514455605427: 1, 10.324804783311686: 1,
10.316184724109968: 1, 10.309942276862605: 1, 10.30885866377888: 1, 10.307923
5567336: 1, 10.307748816637247: 1, 10.285190815158067: 1, 10.256469984909048:
1, 10.204249260893851: 1, 10.202988389985983: 1, 10.20040993206246: 1, 10.194
348717648742: 1, 10.19417478865427: 1, 10.164592412062804: 1, 10.144361132455
34: 1, 10.141935929793187: 1, 10.128980963944775: 1, 10.127485759423301: 1, 1
0.125187858092227: 1, 10.106209035939234: 1, 10.058013184498723: 1, 10.045114
833449006: 1, 10.034885263099804: 1, 10.008981613275264: 1, 9.99183637218152
4: 1, 9.984031507144154: 1, 9.983656519817478: 1, 9.96279287768526: 1, 9.9372
77338336775: 1, 9.933286007727393: 1, 9.92347488942397: 1, 9.907623653413433:
1, 9.903055985996494: 1, 9.874447899546894: 1, 9.863955703913884: 1, 9.863281
88466338: 1, 9.848884565831957: 1, 9.825582173652306: 1, 9.811122758740774:
1, 9.778668798103258: 1, 9.762768107194765: 1, 9.753565312803016: 1, 9.752811
448910286: 1, 9.734135217370056: 1, 9.732835323340417: 1, 9.729294175262728:
1, 9.703330420514037: 1, 9.678607015661223: 1, 9.676558635647998: 1, 9.671400
781192506: 1, 9.664853700782235: 1, 9.64717497522519: 1, 9.639726241057776:
1, 9.621997322005646: 1, 9.621260077816846: 1, 9.61603287639294: 1, 9.6157777
44162141: 1, 9.612280116796036: 1, 9.609977749112886: 1, 9.607464324834277:
1, 9.604218302973774: 1, 9.591351006884503: 1, 9.566577797692359: 1, 9.555574
287919299: 1, 9.535780663002049: 1, 9.520200919586378: 1, 9.49587524084884:
1, 9.490253625436901: 1, 9.458742842557166: 1, 9.440492907248853: 1, 9.438635

062812784: 1, 9.415423723599424: 1, 9.407569739646371: 1, 9.398461835459615:
1, 9.391641687615731: 1, 9.38793797736614: 1, 9.383416740935461: 1, 9.3696300
29647467: 1, 9.364840303813438: 1, 9.35771508365923: 1, 9.338743848765757: 1,
9.300898325962311: 1, 9.298570419453986: 1, 9.292900367015056: 1, 9.287100666
730595: 1, 9.281884772894614: 1, 9.278495272348511: 1, 9.278451782018047: 1,
9.269802833917263: 1, 9.262004092675902: 1, 9.231465547907419: 1, 9.212615974
720329: 1, 9.208388304688185: 1, 9.199927467261878: 1, 9.193536626426551: 1,
9.174467837924631: 1, 9.167342995053794: 1, 9.156674480605078: 1, 9.146320374
045555: 1, 9.131578539698625: 1, 9.128347337280381: 1, 9.115773053250475: 1,
9.112934099662953: 1, 9.071134209439348: 1, 9.057479910606899: 1, 9.048666978
459824: 1, 9.026266385729647: 1, 9.012121003866087: 1, 9.010429509767514: 1,
8.977734008703631: 1, 8.9718913777688: 1, 8.963210732619675: 1, 8.95447544788
381: 1, 8.93303601731467: 1, 8.929895353016674: 1, 8.924365758177405: 1, 8.89
3402793102455: 1, 8.889967980916966: 1, 8.873532684742196: 1, 8.8686750676148
96: 1, 8.86432318415482: 1, 8.848437468808552: 1, 8.837286469092493: 1, 8.837
089365381111: 1, 8.834162752020681: 1, 8.829831058817136: 1, 8.82047980697176
1: 1, 8.807650365041594: 1, 8.79879970222813: 1, 8.797288713069285: 1, 8.7933
06594350893: 1, 8.7921897825: 1, 8.780659298716834: 1, 8.775746475120128: 1,
8.768252030410034: 1, 8.739553366460758: 1, 8.73418484440909: 1, 8.7111899429
98121: 1, 8.705294397528522: 1, 8.665215633959821: 1, 8.642750026870548: 1,
8.637735960627298: 1, 8.625249346131456: 1, 8.608233818746777: 1, 8.598821625
765574: 1, 8.592386983052066: 1, 8.540025691196664: 1, 8.53241188433524: 1,
8.527361790399505: 1, 8.509741378739443: 1, 8.462402610458609: 1, 8.434093726
062628: 1, 8.427054285041473: 1, 8.425877355300504: 1, 8.422711592406078: 1,
8.413485644845167: 1, 8.410367868443739: 1, 8.404887232197593: 1, 8.394068940
895615: 1, 8.393898275433326: 1, 8.37818373263038: 1, 8.36926509558755: 1, 8.
36823288222233: 1, 8.364467182206383: 1, 8.360956719237542: 1, 8.354505350945
528: 1, 8.349805508538493: 1, 8.347187968443752: 1, 8.337871461434041: 1, 8.3
33262821965688: 1, 8.310863023092542: 1, 8.30835569151172: 1, 8.3036927480807
28: 1, 8.302270600459662: 1, 8.296915118175209: 1, 8.270775271851756: 1, 8.25
7760459337653: 1, 8.255090368730626: 1, 8.254590687180109: 1, 8.2453414141927
31: 1, 8.231835335353784: 1, 8.230057774098201: 1, 8.226247106725271: 1, 8.22
4514525085638: 1, 8.222804357504296: 1, 8.217766055621274: 1, 8.1938163781159
69: 1, 8.176815241889338: 1, 8.159547534383169: 1, 8.158986929697601: 1, 8.15
5832229152091: 1, 8.139882264569032: 1, 8.137946396418728: 1, 8.1366518345162
86: 1, 8.13188064702904: 1, 8.118912951835064: 1, 8.103072926825192: 1, 8.098
571040872503: 1, 8.093865776262023: 1, 8.09147430267935: 1, 8.07665536209481
9: 1, 8.068789461510239: 1, 8.05310239434475: 1, 8.044629144292056: 1, 8.0379
73450035029: 1, 8.03577322706778: 1, 8.024243136132192: 1, 8.015098383103592:
1, 8.013804120372523: 1, 8.01143640982525: 1, 8.008372041137529: 1, 8.0071810
28365489: 1, 7.983640027943877: 1, 7.974610218698649: 1, 7.968187902518896:
1, 7.95607941206842: 1, 7.946704836074053: 1, 7.937397204002931: 1, 7.9313077
3977904: 1, 7.920348603763132: 1, 7.920100701778077: 1, 7.890748853301732: 1,
7.88565859932581: 1, 7.8799553077178475: 1, 7.8727477115817175: 1, 7.86645905
5924309: 1, 7.853817072148141: 1, 7.849868339965525: 1, 7.8450953671668975:
1, 7.835774755404122: 1, 7.833286621467787: 1, 7.82346009512649: 1, 7.8213642
35310956: 1, 7.81924438197967: 1, 7.818414974642891: 1, 7.802828843943118: 1,
7.789174783657168: 1, 7.785440298915325: 1, 7.778309377608: 1, 7.771621887383
823: 1, 7.756423763829247: 1, 7.753395178857739 5: 1, 7.750510572974808: 1, 7.
747541206805559: 1, 7.736340641680655: 1, 7.733826247282813: 1, 7.73292565489
4455: 1, 7.704210251777623: 1, 7.687041759617091: 1, 7.6839609894292265: 1,
7.657732860989811: 1, 7.655560439271855: 1, 7.646528501021077: 1, 7.620568334
720252: 1, 7.61564674565298: 1, 7.610048932505413: 1, 7.6048675668285135: 1,
7.589679489862656: 1, 7.588639884925945: 1, 7.574231404333837: 1, 7.572524392
078687: 1, 7.570091054055979: 1, 7.538640036431658: 1, 7.534531006048766: 1,
7.534405403079244: 1, 7.532703943395705: 1, 7.530844114090731: 1, 7.525431075
546679: 1, 7.522282301736196: 1, 7.516003786381438: 1, 7.4997403045877835: 1,

7.489043417689902: 1, 7.483659778453195: 1, 7.483202128707229: 1, 7.475467875
961048: 1, 7.465808808426066: 1, 7.455555534757147: 1, 7.454623820689964: 1,
7.45424145849746: 1, 7.451207594172679: 1, 7.447756069354458: 1, 7.4451834181
61324: 1, 7.439517234356867: 1, 7.438095422083778: 1, 7.43720021619946: 1, 7.
435650711192156: 1, 7.435150582520152: 1, 7.433671037680537: 1, 7.43109744352
2721: 1, 7.426118935109595: 1, 7.4157644477909015: 1, 7.414123623024488: 1,
7.4135163362848795: 1, 7.405462722559985: 1, 7.396917939253185: 1, 7.38975017
5254799: 1, 7.386854814980005: 1, 7.379427257433485: 1, 7.349537178413652: 1,
7.34699062136292: 1, 7.34632139239041: 1, 7.345342540188189: 1, 7.34531281804
8243: 1, 7.3363017375351856: 1, 7.334333588326137: 1, 7.333744063460562: 1,
7.3295577374947145: 1, 7.32462386032966: 1, 7.315749875471744: 1, 7.315734042
372669: 1, 7.309169176311314: 1, 7.3035756888443375: 1, 7.295888724324: 1, 7.
279422990081774: 1, 7.277834989318718: 1, 7.26996545519193: 1, 7.266634651843
623: 1, 7.2657420874694285: 1, 7.265227543391312: 1, 7.2548464405535045: 1,
7.254040545710514: 1, 7.25195884364719: 1, 7.250645584225865: 1, 7.2385715262
45146: 1, 7.2385415572683245: 1, 7.225596520996678: 1, 7.2242724773372125: 1,
7.223160338720904: 1, 7.21844784799384: 1, 7.21789855088967: 1, 7.21111605918
8388: 1, 7.203631260463822: 1, 7.201337814220095: 1, 7.199813081433659: 1, 7.
196334837791678: 1, 7.189996245322334: 1, 7.188328607148156: 1, 7.17891770411
3717: 1, 7.176660521327448: 1, 7.173792840523503: 1, 7.166117295334991: 1, 7.
165893920821868: 1, 7.1546843780807095: 1, 7.1544372628388215: 1, 7.147273240
316896: 1, 7.1333798367153225: 1, 7.114300712469973: 1, 7.107819409662268: 1,
7.1040100256344525: 1, 7.087634701979964: 1, 7.085238318416251: 1, 7.07834005
5818692: 1, 7.071471971354926: 1, 7.0710537775616595: 1, 7.04768693516066: 1,
7.046992975854593: 1, 7.046200361757911: 1, 7.044902535497098: 1, 7.011122179
443234: 1, 6.995832744814691: 1, 6.9949332366985315: 1, 6.991379929441302: 1,
6.985242387709379: 1, 6.980189262148104: 1, 6.974608886700278: 1, 6.967345315
198388: 1, 6.965054166158064: 1, 6.959269186060345: 1, 6.957088538134727: 1,
6.95348677156546: 1, 6.953122911432042: 1, 6.952673537004035: 1, 6.9517323885
35946: 1, 6.948458814807734: 1, 6.940445334514639: 1, 6.940177500805482: 1,
6.9302349906939416: 1, 6.929932106379356: 1, 6.922188506210147: 1, 6.92044945
5284573: 1, 6.895492711966139: 1, 6.892209930566253: 1, 6.876206279428842: 1,
6.864684220133762: 1, 6.864382399860799: 1, 6.8600934072711475: 1, 6.85210308
372816: 1, 6.842063204345825: 1, 6.840133145562919: 1, 6.840055875684098: 1,
6.8346419656512385: 1, 6.832687522389299: 1, 6.824801546891815: 1, 6.81891146
8487335: 1, 6.8182643265158855: 1, 6.814329336147163: 1, 6.809465706307469:
1, 6.799602095386883: 1, 6.790663778421243: 1, 6.781044587962151: 1, 6.777798
977285573: 1, 6.763157740938493: 1, 6.76279434248843: 1, 6.761754844236478:
1, 6.758662885358046: 1, 6.757278585296382: 1, 6.752701089652637: 1, 6.742569
733928842: 1, 6.739188859816833: 1, 6.733306220481862: 1, 6.7325469150983555:
1, 6.726078122178468: 1, 6.711584878067522: 1, 6.706793715166904: 1, 6.699437
888063343: 1, 6.694560180838213: 1, 6.691456312104805: 1, 6.690639919975034:
1, 6.686145997767717: 1, 6.685114926516651: 1, 6.6765111981999326: 1, 6.67408
530850851: 1, 6.673771750500853: 1, 6.673458481171909: 1, 6.671768237394078:
1, 6.650612935091607: 1, 6.649095518378811: 1, 6.64701361767823: 1, 6.6431922
9468136: 1, 6.629485967629403: 1, 6.620650356079414: 1, 6.620029180428358: 1,
6.616532412486531: 1, 6.610457336342002: 1, 6.606615674970633: 1, 6.605845961
889398: 1, 6.605470940606997: 1, 6.605204997374264: 1, 6.600123120795025: 1,
6.5960721452276445: 1, 6.595267488881402: 1, 6.588636200259939: 1, 6.58614391
3984788: 1, 6.585457736855693: 1, 6.584411244206195: 1, 6.583335931579108: 1,
6.581907153390796: 1, 6.568218855851037: 1, 6.561014803591035: 1, 6.559127202
635514: 1, 6.554706774970606: 1, 6.550865568169322: 1, 6.550395663475484: 1,
6.544578708832675: 1, 6.539912352142364: 1, 6.537692134723019: 1, 6.537200571
672416: 1, 6.535176537024325: 1, 6.5319781862003286: 1, 6.5314331761800775:
1, 6.529941864431565: 1, 6.526585080068716: 1, 6.517879770576342: 1, 6.516188
9489508855: 1, 6.510520356494515: 1, 6.502446321387386: 1, 6.499281196495652:
1, 6.473916956154237: 1, 6.4698421448621: 1, 6.468086386878376: 1, 6.46124880

165743: 1, 6.455373186919611: 1, 6.44102375837227: 1, 6.440666261397331: 1,
6.439493966141169: 1, 6.43892467869658: 1, 6.4303497450675025: 1, 6.421161223
4142015: 1, 6.419179656800385: 1, 6.4130819016584475: 1, 6.406915403625017:
1, 6.406218151621363: 1, 6.400314208651538: 1, 6.392427006856368: 1, 6.392402
913300821: 1, 6.38648500196152: 1, 6.384946622116972: 1, 6.383768831416948:
1, 6.382769457240634: 1, 6.381158396730236: 1, 6.366398629584229: 1, 6.366254
348259082: 1, 6.363243761070787: 1, 6.355542919408978: 1, 6.354602168925607:
1, 6.352868481170254: 1, 6.350670532768012: 1, 6.343365190141653: 1, 6.339272
535915991: 1, 6.336022360407264: 1, 6.320803037613698: 1, 6.318906698548467:
1, 6.315996939917673: 1, 6.311183581313232: 1, 6.309955178335944: 1, 6.309441
563156058: 1, 6.308765680933838: 1, 6.300735285761324: 1, 6.29772804745985:
1, 6.296179172602322: 1, 6.292267135501518: 1, 6.290951297356713: 1, 6.289503
735146313: 1, 6.289049839191083: 1, 6.287233063423776: 1, 6.284957625698734:
1, 6.278545881681361: 1, 6.274462494591547: 1, 6.2731177421064706: 1, 6.26972
6807621484: 1, 6.266521810072739: 1, 6.264241477363005: 1, 6.257187535522934
5: 1, 6.2521216119130525: 1, 6.246817889124164: 1, 6.239762121603014: 1, 6.23
8637908852316: 1, 6.225425799733955: 1, 6.221110646375823: 1, 6.2152879118373
72: 1, 6.208495303393895: 1, 6.208160120526324: 1, 6.204329370106509: 1, 6.19
9051262766211: 1, 6.196159881377546: 1, 6.1952268329431295: 1, 6.190076923196
2185: 1, 6.18664254498155: 1, 6.182874395935738: 1, 6.178505754864548: 1, 6.1
75603887804001: 1, 6.174929481257693: 1, 6.174387339474762: 1, 6.173922901358
872: 1, 6.167778014123222: 1, 6.154439318385555: 1, 6.146369597086603: 1, 6.1
40523163300803: 1, 6.139531660252986: 1, 6.1395008380207186: 1, 6.13888675436
6038: 1, 6.13646678542361: 1, 6.1343805391681485: 1, 6.123487098917152: 1, 6.
118874288281946: 1, 6.115827258859581: 1, 6.115236768136035: 1, 6.11411471177
5271: 1, 6.1003678926369815: 1, 6.099088208118183: 1, 6.0974850280686805: 1,
6.084873529767829: 1, 6.082710848711831: 1, 6.081906739008731: 1, 6.078293760
3251534: 1, 6.0738816406493275: 1, 6.0712208005447685: 1, 6.0684924408076615:
1, 6.066968138109702: 1, 6.065615066001509: 1, 6.0484552749972895: 1, 6.04752
4231922154: 1, 6.045952750162185: 1, 6.043026285241438: 1, 6.04158090854639:
1, 6.036099668704948: 1, 6.034106720503804: 1, 6.0259419089277815: 1, 6.02219
4678439608: 1, 6.021669762580522: 1, 6.020150231155806: 1, 6.018698112239185:
1, 6.015142017067388: 1, 6.013738239773923: 1, 6.008026332917549: 1, 6.005246
746670771: 1, 6.001726475379677: 1, 5.998358358796301: 1, 5.988896760608023:
1, 5.986759058197938: 1, 5.982095283124514: 1, 5.968994139106221: 1, 5.968533
006498173: 1, 5.958944905097003: 1, 5.957610854459674: 1, 5.957549016799026:
1, 5.953924000231291: 1, 5.936643398946758: 1, 5.932304377595661: 1, 5.928467
421946977: 1, 5.926006392519049: 1, 5.925985938804911: 1, 5.925149955475638:
1, 5.923485007181017: 1, 5.913231655982814: 1, 5.909792237431517: 1, 5.908875
565380516: 1, 5.904783106454189: 1, 5.903766207394438: 1, 5.903403503532106:
1, 5.903303617801972: 1, 5.901057869428784: 1, 5.890281012728765: 1, 5.882762
343511277: 1, 5.878639881856611: 1, 5.878392525689204: 1, 5.874006431437509:
1, 5.869809259902728: 1, 5.868232675619844: 1, 5.867800509392383: 1, 5.865502
366924055: 1, 5.84920738876643: 1, 5.848501220666147: 1, 5.846749729085603:
1, 5.839523490587984: 1, 5.838465703624011: 1, 5.837654023712371: 1, 5.832603
485763532: 1, 5.829420693487745: 1, 5.8199759629401395: 1, 5.815808179333826
5: 1, 5.815427125841968: 1, 5.8043958937708915: 1, 5.803951281093527: 1, 5.79
6857837048451: 1, 5.796095632772236: 1, 5.795816422650517: 1, 5.7862228233017
27: 1, 5.785318986841852: 1, 5.7844020021343825: 1, 5.780048633489434: 1, 5.7
78122764586138: 1, 5.771521853616708: 1, 5.762907673055105: 1, 5.761809538443
252: 1, 5.7611011951629205: 1, 5.7582797119473685: 1, 5.745268981984933: 1,
5.745016372887236: 1, 5.742817348612226: 1, 5.742359533887401: 1, 5.741864175
697434: 1, 5.737288254743049: 1, 5.733307226031314: 1, 5.729611407742973: 1,
5.728018527219012: 1, 5.724850182859415: 1, 5.714634957230862: 1, 5.714109196
374299: 1, 5.707877839579637: 1, 5.703232404348717: 1, 5.696487230771939: 1,
5.692407833646717: 1, 5.690751727145975: 1, 5.68675818270872: 1, 5.6860919142
06901: 1, 5.684464712369001: 1, 5.682995465136592: 1, 5.681995733459919: 1,

5.678320603218654: 1, 5.674772700694385: 1, 5.671562630714681: 1, 5.671442568
304026: 1, 5.670313394033222: 1, 5.669169887404552: 1, 5.666124797307498: 1,
5.66522651152809: 1, 5.665155978130395: 1, 5.6630301142637895: 1, 5.660363200
635671: 1, 5.655297707971018: 1, 5.653077182322894: 1, 5.651407983160256: 1,
5.648280134814735: 1, 5.642559694146953: 1, 5.637822084648064: 1, 5.634915644
851437: 1, 5.628341951177317: 1, 5.628036934354697: 1, 5.625460007075839: 1,
5.617783415680448: 1, 5.61700486181163: 1, 5.616260866284926: 1, 5.6125857774
92212: 1, 5.6057592431579915: 1, 5.602631003221215: 1, 5.598919541336835: 1,
5.593438939036978: 1, 5.584805617171513: 1, 5.581026740563202: 1, 5.576940660
2876614: 1, 5.573169003672782: 1, 5.573156009812074: 1, 5.5659392104268095:
1, 5.565599480520799: 1, 5.553151884462012: 1, 5.547795336392624: 1, 5.541656
840552694: 1, 5.538399900499009: 1, 5.537863712724488: 1, 5.5353830489530305:
1, 5.534914600381302: 1, 5.5335926250635605: 1, 5.531679830660683: 1, 5.53131
1644198382: 1, 5.528967467942003: 1, 5.527573412093377: 1, 5.525450596876813:
1, 5.525210592854513: 1, 5.522676848508043: 1, 5.521996568295644: 1, 5.519388
973508073: 1, 5.518490817971223: 1, 5.5176410581329955: 1, 5.509170924794335:
1, 5.508930780418513: 1, 5.50730414532773: 1, 5.503459697610953: 1, 5.4928620
65872174: 1, 5.490714870985797: 1, 5.488475356281087: 1, 5.487173504274308:
1, 5.48648459493178: 1, 5.486305164707815: 1, 5.4834747563736075: 1, 5.482159
088616977: 1, 5.482102583555164: 1, 5.481412499829843: 1, 5.480486631160474:
1, 5.459856940519584: 1, 5.458016914150016: 1, 5.455563093999803: 1, 5.452832
525628786: 1, 5.452828915254575: 1, 5.452294044661136: 1, 5.448543084755401:
1, 5.445141835968416: 1, 5.443872209822557: 1, 5.442431488394724: 1, 5.442300
573454713: 1, 5.441041261926465: 1, 5.439664522725182: 1, 5.436749554633992:
1, 5.432193576612721: 1, 5.430616004069497: 1, 5.419986079598382: 1, 5.417106
676036736: 1, 5.412430653685672: 1, 5.410151790222825: 1, 5.403369052915276:
1, 5.403291831902201: 1, 5.395021799291056: 1, 5.385297657226725: 1, 5.385045
75800931: 1, 5.38252050851918: 1, 5.377046092332859: 1, 5.373990523443844: 1,
5.370358081715579: 1, 5.367938806727514: 1, 5.3650720376179: 1, 5.36460534914
4809: 1, 5.362774923435719: 1, 5.362551974157419: 1, 5.36013660399378: 1, 5.3
56783159699931: 1, 5.356040474621517: 1, 5.354048941791896: 1, 5.337710727205
77: 1, 5.33739793841048: 1, 5.329992810423856: 1, 5.327766354361265: 1, 5.326
342249781517: 1, 5.326188204303895: 1, 5.325970900459007: 1, 5.31306049558179
1: 1, 5.310263757465261: 1, 5.305481044797053: 1, 5.300274073304189: 1, 5.290
321057210249: 1, 5.286178247635597: 1, 5.283403427307616: 1, 5.28006147124070
95: 1, 5.279802626240829: 1, 5.275054094086049: 1, 5.270990552777189: 1, 5.26
8615022301102: 1, 5.2680793169483895: 1, 5.264008029779109: 1, 5.256620356958
065: 1, 5.255831370919808: 1, 5.253106735820492: 1, 5.246283082681692: 1, 5.2
46073887862078: 1, 5.230934435447782: 1, 5.228255563999244: 1, 5.227610133650
37: 1, 5.226355984902416: 1, 5.224778006351584: 1, 5.221290716597707: 1, 5.22
0308913866194: 1, 5.219572889669103: 1, 5.217396604806913: 1, 5.2040697082383
54: 1, 5.2008277213660214: 1, 5.200198279718415: 1, 5.199597777233765: 1, 5.1
92738363422409: 1, 5.189966003122697: 1, 5.189837267539828: 1, 5.188162689541
539: 1, 5.1874077093072595: 1, 5.18457885147363: 1, 5.182614667759963: 1, 5.1
78304644686706: 1, 5.176183792622736: 1, 5.176101359676091: 1, 5.175024326736
048: 1, 5.17221288597867: 1, 5.171659799877852: 1, 5.171435418539386: 1, 5.17
1208791019984: 1, 5.170586679876687: 1, 5.169691347019148: 1, 5.1688516581515
89: 1, 5.165448219098819: 1, 5.158088181356422: 1, 5.155671054592791: 1, 5.15
5371614798946: 1, 5.1511425794436: 1, 5.148557265406951: 1, 5.14695348898619
7: 1, 5.144833687303473: 1, 5.138683251787828: 1, 5.137204184188592: 1, 5.136
72958020539: 1, 5.1364824327841685: 1, 5.130481719874734: 1, 5.12931831128938
3: 1, 5.127508283279585: 1, 5.127378245812993: 1, 5.124320922272676: 1, 5.121
600671272781: 1, 5.120930505119319: 1, 5.118597929740117: 1, 5.11819685035674
9: 1, 5.112825171117649: 1, 5.102214251767377: 1, 5.102142748614428: 1, 5.093
135629314614: 1, 5.089014120261349: 1, 5.086382454575593: 1, 5.08525143720278
7: 1, 5.084339046273328: 1, 5.079986307754622: 1, 5.0782087904957365: 1, 5.07
813360288275: 1, 5.0769685385872485: 1, 5.0758696965366985: 1, 5.068026973869

969: 1, 5.066513917540595: 1, 5.060642977649037: 1, 5.059171636923679: 1, 5.0
534272913015155: 1, 5.052695975944092: 1, 5.0503788523342505: 1, 5.0489588693
67676: 1, 5.048649633988063: 1, 5.0433304938089485: 1, 5.043156891133524: 1,
5.041900486051247: 1, 5.040407327482474: 1, 5.037543453551812: 1, 5.035425305
196968: 1, 5.034601932583041: 1, 5.033764692872557: 1, 5.030531837248916: 1,
5.029417854782003: 1, 5.026921737265824: 1, 5.026113630431629: 1, 5.014508303
332805: 1, 5.0141994944171575: 1, 5.012649730352227: 1, 5.009556367958058: 1,
5.004753865323863: 1, 5.002018307969044: 1, 5.001525939931623: 1, 4.999605160
549759: 1, 4.997463522035032: 1, 4.99698972882577: 1, 4.9934420081768565: 1,
4.993153753271658: 1, 4.987689589816278: 1, 4.986588589893383: 1, 4.986455412
130371: 1, 4.985165633455892: 1, 4.981582194548199: 1, 4.978086838049666: 1,
4.9777533809782035: 1, 4.973761640628383: 1, 4.969165982000685: 1, 4.96657556
52523: 1, 4.965546068014742: 1, 4.964993294362422: 1, 4.9621201329297975: 1,
4.9617393525904365: 1, 4.95817903277206: 1, 4.957263361774005: 1, 4.955056377
577102: 1, 4.941622233660841: 1, 4.941212323485376: 1, 4.94092062587522: 1,
4.940710026577117: 1, 4.93764170486758: 1, 4.93540592020105: 1, 4.93382241507
5081: 1, 4.933174484286557: 1, 4.9326300481294645: 1, 4.930496821880117: 1,
4.927001195241066: 1, 4.924980049795287: 1, 4.924118298736195: 1, 4.923133242
866891: 1, 4.918230155866968: 1, 4.913341933963131: 1, 4.905293892820944: 1,
4.9007714516497884: 1, 4.893134452271082: 1, 4.891995703818504: 1, 4.88743096
8493908: 1, 4.885303146275631: 1, 4.884957021730542: 1, 4.883042703776913: 1,
4.878587802026344: 1, 4.874338029227746: 1, 4.873832918910743: 1, 4.873280686
933535: 1, 4.872024764019156: 1, 4.868299804433748: 1, 4.857870209410595: 1,
4.857791881557244: 1, 4.851000515583134: 1, 4.8492629186690355: 1, 4.84408903
5146269: 1, 4.840847371308519: 1, 4.8406150267643895: 1, 4.840202686960207:
1, 4.8368953991361625: 1, 4.835490825859953: 1, 4.833184643571825: 1, 4.83160
8681390676: 1, 4.828480586840948: 1, 4.8277502102229075: 1, 4.82739520193805
7: 1, 4.82291195180808: 1, 4.820024894124508: 1, 4.819922415905043: 1, 4.8197
25858301548: 1, 4.818229314554614: 1, 4.816790697601102: 1, 4.81337176891529
2: 1, 4.809756567453107: 1, 4.80867383814137: 1, 4.807603402264336: 1, 4.8067
623549876535: 1, 4.806374504659761: 1, 4.804359478236349: 1, 4.80216862597333
1: 1, 4.80173184948253: 1, 4.795719740023173: 1, 4.795479458376605: 1, 4.7950
21545208223: 1, 4.794617600532216: 1, 4.7946024324931145: 1, 4.79112023764214
3: 1, 4.789424220901266: 1, 4.77988031693299: 1, 4.778176277555871: 1, 4.7773
80731011005: 1, 4.775485394583881: 1, 4.772777996026796: 1, 4.770530265007967
5: 1, 4.770384385008422: 1, 4.770102399681072: 1, 4.765548876845922: 1, 4.765
37571378577: 1, 4.764357624779968: 1, 4.763571122261553: 1, 4.75538469973486
2: 1, 4.748560168620053: 1, 4.744528111864964: 1, 4.744097690568479: 1, 4.742
343904103674: 1, 4.739448103480621: 1, 4.736502567296855: 1, 4.73586061647314
7: 1, 4.729176693288644: 1, 4.7288678731388725: 1, 4.726208147100145: 1, 4.72
44143874115005: 1, 4.7177408819038895: 1, 4.717640188551039: 1, 4.71705064922
8022: 1, 4.716786078076501: 1, 4.715002218831641: 1, 4.713722635078321: 1, 4.
711115500725767: 1, 4.70588580498386: 1, 4.704699055405047: 1, 4.697910488830
115: 1, 4.695940248827233: 1, 4.685080463865606: 1, 4.682604099208886: 1, 4.6
80356438653123: 1, 4.679924915238479: 1, 4.675982847238895: 1, 4.671904396802
962: 1, 4.669705211163666: 1, 4.669370460382122: 1, 4.668092826066577: 1, 4.6
67236089307677: 1, 4.66665031145275: 1, 4.662659150851304: 1, 4.6623650529994
32: 1, 4.660935828582985: 1, 4.6589384591616625: 1, 4.657450000129365: 1, 4.6
502728516692935: 1, 4.6436639594045905: 1, 4.642295073785905: 1, 4.6372066874
405125: 1, 4.632960595937568: 1, 4.632182249484057: 1, 4.629672498124932: 1,
4.627704965015392: 1, 4.62531056852339: 1, 4.625044466704427: 1, 4.6246332966
12987: 1, 4.62421138257972: 1, 4.6227972959588115: 1, 4.621310378185856: 1,
4.619381881786463: 1, 4.617408358350504: 1, 4.615777948451423: 1, 4.611263333
241131: 1, 4.60995660927027: 1, 4.609452190574741: 1, 4.60816303064001: 1, 4.
604489433196407: 1, 4.60212208176599: 1, 4.599484513696541: 1, 4.597624939646
9795: 1, 4.596176245912769: 1, 4.595439417645223: 1, 4.595428882270077: 1, 4.
594378165079783: 1, 4.592920566742552: 1, 4.592314583619834: 1, 4.59127606512

53: 1, 4.583124117010127: 1, 4.570675918289247: 1, 4.57015274505814: 1, 4.566
709601792031: 1, 4.562029639680596: 1, 4.558966641884142: 1, 4.55633264414600
2: 1, 4.555130218306311: 1, 4.554821590264491: 1, 4.552668125712361: 1, 4.551
898549410656: 1, 4.551437606135269: 1, 4.546190107677923: 1, 4.54533842581638
9: 1, 4.544931167721888: 1, 4.543367999712644: 1, 4.543088784475368: 1, 4.540
167722507717: 1, 4.538482783428014: 1, 4.530659816549603: 1, 4.52714079365370
5: 1, 4.5251570878750575: 1, 4.524328548502583: 1, 4.5219096275952495: 1, 4.5
170104624382335: 1, 4.510936015980615: 1, 4.5071852469183185: 1, 4.5023885657
432885: 1, 4.502238988977838: 1, 4.497655906422237: 1, 4.497377476237154: 1,
4.497267653310948: 1, 4.4922771175074345: 1, 4.492088231879423: 1, 4.48770757
5265972: 1, 4.487565163189595: 1, 4.487364609248117: 1, 4.486318452000696: 1,
4.485802001050486: 1, 4.484991083328694: 1, 4.477922824996966: 1, 4.476544930
301677: 1, 4.472434949796257: 1, 4.471000438504249: 1, 4.466929385333499: 1,
4.464133661014904: 1, 4.46250232586096: 1, 4.453318513138981: 1, 4.4530894328
873964: 1, 4.452848504904694: 1, 4.45181755250286: 1, 4.451322246293616: 1,
4.450940976808292: 1, 4.449027540411314: 1, 4.448335601167764: 1, 4.448115909
834187: 1, 4.4475034731219445: 1, 4.445427774806077: 1, 4.443643239940175: 1,
4.442842238313728: 1, 4.441919841202091: 1, 4.441381693299498: 1, 4.438279230
676292: 1, 4.437064347442489: 1, 4.436529336808983: 1, 4.436277399171596: 1,
4.4356227360342375: 1, 4.435437575417772: 1, 4.426806838337929: 1, 4.42529782
9544068: 1, 4.421659129011802: 1, 4.420833518711186: 1, 4.4195927236976225:
1, 4.417962583761597: 1, 4.416538891707516: 1, 4.416357271687274: 1, 4.410159
3231529765: 1, 4.408368365683247: 1, 4.406046854137066: 1, 4.405112207144918:
1, 4.404842842915566: 1, 4.402452142520733: 1, 4.396019268371731: 1, 4.394239
022091352: 1, 4.394108588137325: 1, 4.391838728232267: 1, 4.3913959668881715:
1, 4.386903142687073: 1, 4.386106473547367: 1, 4.381377881373001: 1, 4.380522
384791478: 1, 4.38025434072404: 1, 4.379368610439866: 1, 4.376797567768485:
1, 4.372752786144968: 1, 4.3711516325390125: 1, 4.370217762879238: 1, 4.36903
4558503325: 1, 4.367486192166827: 1, 4.36083716760889: 1, 4.36058383082849:
1, 4.355801675566557: 1, 4.355230348178777: 1, 4.353271277260706: 1, 4.352839
393530154: 1, 4.346431694175009: 1, 4.344985068105563: 1, 4.344964154758223:
1, 4.344540496642199: 1, 4.344231091768459: 1, 4.338880406508014: 1, 4.336841
146500812: 1, 4.334895028349505: 1, 4.330292506653719: 1, 4.32456646083775:
1, 4.323595031401306: 1, 4.3229171471332055: 1, 4.321276172689234: 1, 4.32039
0472939229: 1, 4.311931883761854: 1, 4.308454618583407: 1, 4.308147140506289:
1, 4.307117523968253: 1, 4.303485324504089: 1, 4.301579275817228: 1, 4.301554
034022785: 1, 4.299650210363782: 1, 4.298953475931368: 1, 4.2973542764694495:
1, 4.296813111607984: 1, 4.296706856390512: 1, 4.292551695267064: 1, 4.292406
281710414: 1, 4.291476840297857: 1, 4.289016286575809: 1, 4.288722254783146:
1, 4.288360388093887: 1, 4.28693618496168: 1, 4.286599641436231: 1, 4.2856700
94369309: 1, 4.285356314143469: 1, 4.281372656842987: 1, 4.281306109239309:
1, 4.2783656601891025: 1, 4.2754108377612035: 1, 4.274331614145997: 1, 4.2728
80415172772: 1, 4.272365143328885: 1, 4.266726850391314: 1, 4.263946568588643
5: 1, 4.263738029571679: 1, 4.261717673216022: 1, 4.2590229312207315: 1, 4.25
7780515593382: 1, 4.256673276519642: 1, 4.253304695849948: 1, 4.2471884387517
35: 1, 4.246854148714547: 1, 4.244720865806438: 1, 4.244161965113231: 1, 4.24
4054192948692: 1, 4.243947206027827: 1, 4.2436839446456345: 1, 4.241535901883
33: 1, 4.239329096027751: 1, 4.237837706797284: 1, 4.237593601117136: 1, 4.23
6038329104904: 1, 4.233847066160102: 1, 4.228645474916917: 1, 4.2262606262194
81: 1, 4.223825261699573: 1, 4.2225637566470064: 1, 4.2224042177362655: 1, 4.
222241804109873: 1, 4.2199936349503195: 1, 4.218351714137975: 1, 4.2181013184
79783: 1, 4.217213239814814: 1, 4.216380631601942: 1, 4.214812886710508: 1,
4.214778800508218: 1, 4.212509613007285: 1, 4.209707354656219: 1, 4.209545143
619574: 1, 4.208250733146553: 1, 4.20527298132889: 1, 4.205244702524224: 1,
4.204620772624564: 1, 4.203882340256214: 1, 4.203070878251354: 1, 4.202465573
065834: 1, 4.2020014681024564: 1, 4.197772596857085: 1, 4.192581081732387: 1,
4.1898606750366865: 1, 4.189683877531032: 1, 4.1892537682684: 1, 4.1883623118

8634: 1, 4.187644014288872: 1, 4.184647616745885: 1, 4.182780958533313: 1, 4.
181715763224877: 1, 4.1786623825248075: 1, 4.1774033348975195: 1, 4.177399565
338208: 1, 4.177270436804917: 1, 4.1757362284688: 1, 4.1721045254228235: 1,
4.171426289748511: 1, 4.169140106865022: 1, 4.168691914657681: 1, 4.165877158
949549: 1, 4.165560797181955: 1, 4.164233971667076: 1, 4.1607839749534286: 1,
4.157759640850156: 1, 4.1577351207175015: 1, 4.156145640233748: 1, 4.15444490
0047216: 1, 4.151704729489672: 1, 4.149395584542194: 1, 4.1469418511698635:
1, 4.145743510026597: 1, 4.145356213333105: 1, 4.1441036620008: 1, 4.14271980
2749306: 1, 4.142376844057339: 1, 4.142138355810479: 1, 4.141442536048591: 1,
4.138456187449283: 1, 4.137018130914529: 1, 4.136119917164326: 1, 4.131946712
3919775: 1, 4.131935940249696: 1, 4.1313641130546594: 1, 4.127734249052269:
1, 4.127032266215833: 1, 4.126927297316035: 1, 4.125577154303145: 1, 4.123295
715255735: 1, 4.121763587166746: 1, 4.121581621866593: 1, 4.120510925419951:
1, 4.119097893619122: 1, 4.11793593775354: 1, 4.117107301933155: 1, 4.1152045
32666031: 1, 4.114467508485191: 1, 4.114041765070037: 1, 4.113677730533677:
1, 4.1133796266313425: 1, 4.111814849275471: 1, 4.1116874640772405: 1, 4.1107
58209265883: 1, 4.107271205158767: 1, 4.105974438618064: 1, 4.10553216363426
7: 1, 4.105340883861273: 1, 4.101691903963956: 1, 4.100554314590878: 1, 4.097
352052339666: 1, 4.0955512066488176: 1, 4.09494808591048: 1, 4.09442814735807
5: 1, 4.0940113259352575: 1, 4.093646486238842: 1, 4.093630272283933: 1, 4.09
2562495344942: 1, 4.090205830046106: 1, 4.089760608972303: 1, 4.0892534728784
51: 1, 4.08726676186626: 1, 4.087028672392644: 1, 4.0847212565358095: 1, 4.08
1686626337606: 1, 4.081605124508838: 1, 4.080404079009891: 1, 4.0785332549587
96: 1, 4.077015945782129: 1, 4.0769919154864995: 1, 4.076320900039112: 1, 4.0
75222611953705: 1, 4.074569103778705: 1, 4.072191197215857: 1, 4.070331968634
3375: 1, 4.069312248459469: 1, 4.068681949894429: 1, 4.0686328892240615: 1,
4.066659272472504: 1, 4.065919300063251: 1, 4.058836220730147: 1, 4.057929876
4529925: 1, 4.057139213567289: 1, 4.053858059442416: 1, 4.05319448012971: 1,
4.049720096977308: 1, 4.048492926785763: 1, 4.047364032969243: 1, 4.047263854
350644: 1, 4.046911754236252: 1, 4.045988998531548: 1, 4.04379844675502: 1,
4.043508316549219: 1, 4.041197687842369: 1, 4.039136413379458: 1, 4.038717982
6018045: 1, 4.035606314585228: 1, 4.033492687257926: 1, 4.03101620479559: 1,
4.030888843736879: 1, 4.030611842389772: 1, 4.029953388976614: 1, 4.029569823
414776: 1, 4.027899372659142: 1, 4.02751398873664: 1, 4.024072062795729: 1,
4.0204761813064405: 1, 4.018855161917928: 1, 4.0176084254295805: 1, 4.0150671
63921068: 1, 4.0107164240391615: 1, 4.0105547096746905: 1, 4.00995615032328:
1, 4.006837800583148: 1, 4.003655121346403: 1, 4.003147697219858: 1, 4.000919
43149058: 1, 3.9988599645573455: 1, 3.9959997753815037: 1, 3.992196067004726
5: 1, 3.9917340762030067: 1, 3.990468677870574: 1, 3.990394652037328: 1, 3.98
86965377342998: 1, 3.98792860505994: 1, 3.9879135231217777: 1, 3.987296598212
2133: 1, 3.985588326111395: 1, 3.985411150283441: 1, 3.985362123257546: 1, 3.
9840951147275647: 1, 3.9811866761012915: 1, 3.980742205334173: 1, 3.972753083
3131484: 1, 3.9717840600072214: 1, 3.9711464137005414: 1, 3.970660750633057:
1, 3.970434062853694: 1, 3.9683958744188375: 1, 3.9590278594243027: 1, 3.9582
931155338987: 1, 3.9566161680660024: 1, 3.954739157322774: 1, 3.9543576193885
537: 1, 3.9541646783420483: 1, 3.951903124666316: 1, 3.950487687262209: 1, 3.
9503707100442544: 1, 3.94987070023044: 1, 3.948804100106201: 1, 3.94875817088
81736: 1, 3.9462349648114268: 1, 3.942959449266555: 1, 3.942800461580562: 1,
3.940911459805586: 1, 3.939487934856059: 1, 3.938513606705208: 1, 3.937155003
2466587: 1, 3.936376585291516: 1, 3.934990193009524: 1, 3.932963647864643: 1,
3.9309380563858762: 1, 3.9305170498421464: 1, 3.930386212374188: 1, 3.9278853
232185953: 1, 3.9273506959840883: 1, 3.927273762108888: 1, 3.927252652204260
6: 1, 3.9267742884678145: 1, 3.9262636253414627: 1, 3.923266875697313: 1, 3.9
227529359361597: 1, 3.922322370295716: 1, 3.9217564262100573: 1, 3.9215611425
622416: 1, 3.9193894271234186: 1, 3.91710416086812: 1, 3.916388563820235: 1,
3.915941784265045: 1, 3.914838268161026: 1, 3.9125633818514154: 1, 3.91240924
4102399: 1, 3.9113136569857025: 1, 3.911311350600873: 1, 3.9099977417905936:

1, 3.909882073565051: 1, 3.908666051494722: 1, 3.905982495485077: 1, 3.904090
9351649384: 1, 3.9039059931197233: 1, 3.902558536664298: 1, 3.898652711911313
2: 1, 3.89856707497196: 1, 3.8983736267460585: 1, 3.8964494380244963: 1, 3.89
30378478157337: 1, 3.892562268076449: 1, 3.892430691328965: 1, 3.892320634931
066: 1, 3.8922322095127786: 1, 3.8898225365229804: 1, 3.8886605630816593: 1,
3.8881576108714353: 1, 3.885809514811007: 1, 3.885566175410352: 1, 3.88493930
10792577: 1, 3.884732057356655: 1, 3.883537022126647: 1, 3.8826598039333757:
1, 3.8807997343799: 1, 3.8768122456923244: 1, 3.8763584183323854: 1, 3.872994
546411792: 1, 3.8728763646351014: 1, 3.8676250084629817: 1, 3.86507182060092
4: 1, 3.863974414279139: 1, 3.8625729632685437: 1, 3.8616319566334614: 1, 3.8
613249476515312: 1, 3.8612940406881986: 1, 3.860530644229804: 1, 3.8602163323
9984: 1, 3.859153060067117: 1, 3.8563154193282267: 1, 3.8549377359203256: 1,
3.854142750656403: 1, 3.8530039562478082: 1, 3.8512369753819207: 1, 3.8508725
076370687: 1, 3.8502020798640078: 1, 3.8499644351545133: 1, 3.84888294998211
2: 1, 3.848297210248276: 1, 3.845711751838472: 1, 3.8437594650035614: 1, 3.84
1433477289187: 1, 3.8405691556075103: 1, 3.8397390756919973: 1, 3.83752337733
50415: 1, 3.835789662767118: 1, 3.8337679639797932: 1, 3.833433748225058: 1,
3.8306805515819335: 1, 3.826366730068398: 1, 3.824800571325812: 1, 3.82334081
05150675: 1, 3.8230038317074864: 1, 3.822166291480281: 1, 3.821399811713908:
1, 3.8206306199198297: 1, 3.820444002668998: 1, 3.819076628100997: 1, 3.81888
5283766418: 1, 3.816767510198759: 1, 3.8145547574464778: 1, 3.81441906151063
5: 1, 3.8140868197251345: 1, 3.8118550992117264: 1, 3.8112536705080733: 1, 3.
8106714819748766: 1, 3.8089941314488636: 1, 3.808122220137284: 1, 3.805901277
4471213: 1, 3.8029979225552784: 1, 3.8024559992671616: 1, 3.800066332875799:
1, 3.797929533220122: 1, 3.7964626217395674: 1, 3.796285088333692: 1, 3.79515
8122146315: 1, 3.792325436912676: 1, 3.7911372098507936: 1, 3.790515676573265
5: 1, 3.7893821738036766: 1, 3.786376002946523: 1, 3.7837115914004285: 1, 3.7
836013049884887: 1, 3.778884468843853: 1, 3.7767175567228506: 1, 3.7754754745
945025: 1, 3.7743859742414476: 1, 3.773374919430692: 1, 3.7714127971839853:
1, 3.7696704254831372: 1, 3.7661973254113663: 1, 3.7643305353689414: 1, 3.763
9838242627337: 1, 3.7606184828481704: 1, 3.7534191664934142: 1, 3.75306479349
25327: 1, 3.752393167747737: 1, 3.7478601659173396: 1, 3.7464834653772: 1, 3.
7456817302855607: 1, 3.7454261052432356: 1, 3.7452383881278712: 1, 3.74446729
83355436: 1, 3.7444372322554638: 1, 3.743030171987425: 1, 3.741708386417582:
1, 3.739335119779381: 1, 3.7380353222605827: 1, 3.7373395432729852: 1, 3.7372
93273122802: 1, 3.7330490647144177: 1, 3.7320524126612993: 1, 3.7317825698847
096: 1, 3.73148778961245: 1, 3.7303484872800796: 1, 3.7293596438668075: 1, 3.
728422096038097: 1, 3.7245448737487417: 1, 3.7185698480609815: 1, 3.714534773
857922: 1, 3.7138175892075167: 1, 3.712360771515992: 1, 3.7091967843360654:
1, 3.7082228386106566: 1, 3.7059720677504804: 1, 3.7040475643236186: 1, 3.702
378147969279: 1, 3.7020251144721334: 1, 3.7019710628614626: 1, 3.701312231407
065: 1, 3.6993350993157974: 1, 3.6988188907841124: 1, 3.6987474503470015: 1,
3.694636088623576: 1, 3.690520843166505: 1, 3.6894809520153276: 1, 3.68054257
7729424: 1, 3.679695421243088: 1, 3.6791082992635786: 1, 3.6770864419221803:
1, 3.676652226010043: 1, 3.67456311183866: 1, 3.6716590619942786: 1, 3.671474
05117815: 1, 3.6708976051611066: 1, 3.6692022151079455: 1, 3.667951774349109
2: 1, 3.6678601538971685: 1, 3.6661739596505307: 1, 3.6624036656122216: 1, 3.
6623715506141137: 1, 3.6617185720409333: 1, 3.661473455034146: 1, 3.661065790
1440025: 1, 3.660802010134634: 1, 3.660055451671785: 1, 3.65832666880561: 1,
3.6566228859807746: 1, 3.6531903532715666: 1, 3.6497996713633296: 1, 3.648880
006705141: 1, 3.648811652621654: 1, 3.6485568164230764: 1, 3.648183503717785
7: 1, 3.647164614631626: 1, 3.647039622272446: 1, 3.6454544285991846: 1, 3.64
52838444949287: 1, 3.6451802181606783: 1, 3.6446428383206917: 1, 3.6441568876
26683: 1, 3.644098529414188: 1, 3.64319198174087: 1, 3.6427413669475563: 1,
3.6399510987168817: 1, 3.639925620089906: 1, 3.638152928139466: 1, 3.63476125
30574303: 1, 3.63450514862231: 1, 3.6329030279560808: 1, 3.6248135929034295:
1, 3.6243932478124052: 1, 3.623149644827918: 1, 3.6230786588745336: 1, 3.6216

126201121024: 1, 3.6212166916465853: 1, 3.6209390089822153: 1, 3.620370178272
154: 1, 3.618331937634458: 1, 3.6158100942720157: 1, 3.6148955679033934: 1,
3.613658198173988: 1, 3.612539121527069: 1, 3.6110390674620643: 1, 3.61095300
75907634: 1, 3.608917933041071: 1, 3.6069315761018075: 1, 3.606257175151612:
1, 3.605715665724341: 1, 3.6038784802651596: 1, 3.6015956647183653: 1, 3.6011
795875605115: 1, 3.600564253531525: 1, 3.6005158824334007: 1, 3.5996640005764
05: 1, 3.5979445231131297: 1, 3.592001142440879: 1, 3.5913705974938472: 1, 3.
590219151320341: 1, 3.589271441138754: 1, 3.5892092166687655: 1, 3.5884885767
31628: 1, 3.5882386249644656: 1, 3.5882123266148396: 1, 3.5862987672594313:
1, 3.5850987435250166: 1, 3.5843811111891815: 1, 3.5837711219487045: 1, 3.582
4370901025153: 1, 3.5818179329472706: 1, 3.581777540878648: 1, 3.581306772016
9878: 1, 3.57953344546295: 1, 3.5775301095288925: 1, 3.5772759235653404: 1,
3.5756567872190033: 1, 3.574176198044599: 1, 3.5727705167724566: 1, 3.5715893
108768912: 1, 3.5709960690076743: 1, 3.570885974295522: 1, 3.570343552104848
3: 1, 3.570046204457708: 1, 3.5697478314035425: 1, 3.569551441624139: 1, 3.56
79037263435176: 1, 3.5677870568248076: 1, 3.5660550670380657: 1, 3.5654766069
531516: 1, 3.564310931502631: 1, 3.563847983661039: 1, 3.5619044383350156: 1,
3.5601869196826685: 1, 3.559750047445979: 1, 3.5596081820913423: 1, 3.5586186
7737047: 1, 3.554280848991224: 1, 3.5531072581027328: 1, 3.551774537260333:
1, 3.5497845209638723: 1, 3.548762731206151: 1, 3.548744357771772: 1, 3.54851
8116689316: 1, 3.548013513940624: 1, 3.547968570427789: 1, 3.547372596206661
6: 1, 3.547088336303552: 1, 3.5457893788491517: 1, 3.544262608999902: 1, 3.54
4161632667061: 1, 3.542786003587618: 1, 3.541237357578447: 1, 3.5393762264775
97: 1, 3.5375072645295473: 1, 3.536459521302296: 1, 3.536263175499393: 1, 3.5
336590633023333: 1, 3.5304200484355697: 1, 3.530307890191761: 1, 3.5299784725
18965: 1, 3.5297972029795255: 1, 3.527537470439139: 1, 3.527407523086587: 1,
3.5271775458366377: 1, 3.5262219196360807: 1, 3.5260442882084266: 1, 3.524533
80448328: 1, 3.5233734373982943: 1, 3.52288835643108: 1, 3.5218409602784404:
1, 3.518752813694259: 1, 3.5181189545798137: 1, 3.5176492016146206: 1, 3.5159
224457040756: 1, 3.515755102362661: 1, 3.513254204713535: 1, 3.51289626386199
6: 1, 3.5124049302537332: 1, 3.5111457479136323: 1, 3.5089426206297425: 1, 3.
507991708018388: 1, 3.5067730151856082: 1, 3.506029266079483: 1, 3.5058594903
0804: 1, 3.505406968751974: 1, 3.5053552519842244: 1, 3.504460214569461: 1,
3.503976518221479: 1, 3.5034254708702997: 1, 3.502540277590486: 1, 3.50204042
9257707: 1, 3.5017585832822604: 1, 3.4988493556760054: 1, 3.4980015863301306:
1, 3.4949480037600047: 1, 3.494743119099147: 1, 3.4925151587520316: 1, 3.4920
464784283425: 1, 3.4909047420835653: 1, 3.490086584021651: 1, 3.4889684716492
04: 1, 3.4881873353699158: 1, 3.4879083909554516: 1, 3.4827616863177417: 1,
3.481617061521227: 1, 3.4789933089621803: 1, 3.475674304338054: 1, 3.47546214
0804547: 1, 3.474979148546568: 1, 3.474550625367871: 1, 3.4703180297298744:
1, 3.4698226082948143: 1, 3.4677236067823696: 1, 3.466768575329738: 1, 3.4661
64985841972: 1, 3.461952675576106: 1, 3.461374950325600 6: 1, 3.46021215898635
24: 1, 3.459834319718322: 1, 3.4596780864712673: 1, 3.458306078356276: 1, 3.4
580878603984537: 1, 3.455806039568953: 1, 3.4556033511747266: 1, 3.4548440319
811715: 1, 3.4532650184596596: 1, 3.4527699513642185: 1, 3.4525370785623988:
1, 3.446371942784952: 1, 3.4461349930983998: 1, 3.443560229092086: 1, 3.44125
7438979597: 1, 3.440513324501166: 1, 3.438686709482148: 1, 3.436602146559966
5: 1, 3.435892952660317: 1, 3.43584896448743: 1, 3.433552461451802: 1, 3.4308
622209736424: 1, 3.4283374337034465: 1, 3.426539612408529: 1, 3.4256205462900
073: 1, 3.42290315641846: 1, 3.4219980659718563: 1, 3.4205179339499074: 1, 3.
420179695417198: 1, 3.419871571483843: 1, 3.4182555116965707: 1, 3.4173572907
363705: 1, 3.416731779797853: 1, 3.414703192800811: 1, 3.4129147486537423: 1,
3.411541816701858: 1, 3.411400710676222: 1, 3.4089372116621943: 1, 3.40744235
5775152: 1, 3.407308259803736: 1, 3.4062735983300856: 1, 3.406088295024526:
1, 3.4054581931610257: 1, 3.4047031451150085: 1, 3.4011692086992174: 1, 3.399
4678074165745: 1, 3.3977925138029783: 1, 3.3955728808358714: 1, 3.39310190290
7151: 1, 3.3920751888962246: 1, 3.391894136361878: 1, 3.3915374449912115: 1,

3.3898562479420096: 1, 3.389476622409793: 1, 3.389355573123474: 1, 3.38893682
55747505: 1, 3.3854565596631496: 1, 3.3808855010812775: 1, 3.380791530194635:
1, 3.379243232487059: 1, 3.3789688004708025: 1, 3.3765817465963686: 1, 3.3756
78504054423: 1, 3.3723216158814187: 1, 3.371408570300678: 1, 3.37124311353485
15: 1, 3.37054363331582: 1, 3.368882959785936: 1, 3.3688723327887904: 1, 3.36
5782382788377: 1, 3.3652578518580674: 1, 3.365160118209476: 1, 3.365078669038
0246: 1, 3.364900697593116: 1, 3.3645571894041897: 1, 3.364554198217011: 1,
3.3639835374074742: 1, 3.3631916574191596: 1, 3.3615858656819197: 1, 3.360332
1047614583: 1, 3.359126117086831: 1, 3.3586042655002846: 1, 3.358154516768442
6: 1, 3.3558508893962475: 1, 3.355454512996914: 1, 3.353347925911618: 1, 3.35
29033847906327: 1, 3.3527083981046975: 1, 3.352699633770058: 1, 3.35058980689
9998: 1, 3.348245102556307: 1, 3.34814468407828: 1, 3.3476808871233468: 1, 3.
3465838120836477: 1, 3.3463066921182074: 1, 3.345325654987345: 1, 3.344701718
6530372: 1, 3.343913142879969: 1, 3.343229838347942: 1, 3.3416259755817506:
1, 3.3407584497540532: 1, 3.3405582833465504: 1, 3.3396242536152436: 1, 3.339
4228979780904: 1, 3.3368339511805702: 1, 3.336368454869574: 1, 3.335893551811
8344: 1, 3.3356428953924864: 1, 3.3306443304774795: 1, 3.3296813285539217: 1,
3.3296339680758624: 1, 3.328371206895849: 1, 3.3270331812085354: 1, 3.3270262
303324665: 1, 3.326824307713834: 1, 3.3264507023248044: 1, 3.324811489091284
3: 1, 3.3244635929925135: 1, 3.322984490175533: 1, 3.3208032819025646: 1, 3.3
206102012059273: 1, 3.319428209962834: 1, 3.3193365873771383: 1, 3.3177159176
152253: 1, 3.317496628717847: 1, 3.317042752831584: 1, 3.3140805912411997: 1,
3.313306741871569: 1, 3.309804783071171: 1, 3.3097514018217535: 1, 3.30968033
6776694: 1, 3.305983744056652: 1, 3.305024021942041: 1, 3.3048612193397293:
1, 3.3045299493319167: 1, 3.3018548979013636: 1, 3.3014270103670262: 1, 3.300
7040697363785: 1, 3.299370410392905: 1, 3.2975966069564135: 1, 3.296146984833
444: 1, 3.2957342151972395: 1, 3.2945487804991553: 1, 3.294291454656505: 1,
3.29195974180559: 1, 3.2911901925093456: 1, 3.29074719136008: 1, 3.2905787701
431617: 1, 3.290323817690757: 1, 3.2901222091732913: 1, 3.290077281391436: 1,
3.288142409985758: 1, 3.288069332591204: 1, 3.2880111328530797: 1, 3.28672226
29781305: 1, 3.286404061521976: 1, 3.285504371764336: 1, 3.2852538836141907:
1, 3.283056635332356: 1, 3.2822533155295197: 1, 3.2814967572157814: 1, 3.2798
445851865354: 1, 3.279460528174203: 1, 3.278610060359498: 1, 3.27859079293096
12: 1, 3.275835009235769: 1, 3.275555333821871: 1, 3.2753985956360743: 1, 3.2
746666966075946: 1, 3.268343700993981: 1, 3.267329728077249: 1, 3.26518270982
55075: 1, 3.2651117187521153: 1, 3.2648291449783065: 1, 3.263679774287636: 1,
3.2622358186080986: 1, 3.2617285523689765: 1, 3.2598172747061214: 1, 3.259566
877417747: 1, 3.25848683360259: 1, 3.2562539924823692: 1, 3.2554740133017566:
1, 3.2541561618113874: 1, 3.2538599015142604: 1, 3.253531990512151: 1, 3.2525
161732761325: 1, 3.252124039655094: 1, 3.2505141369576926: 1, 3.2499578424767
05: 1, 3.248648567892235: 1, 3.248503738163055: 1, 3.246399319979474: 1, 3.24
33687087072403: 1, 3.241110386436025: 1, 3.238859307417068: 1, 3.234731888547
3113: 1, 3.2340803351038194: 1, 3.2321340644383216: 1, 3.2315660528576835: 1,
3.2311466437144185: 1, 3.2307382089790164: 1, 3.2299285811145566: 1, 3.229770
888527039: 1, 3.229221936976164: 1, 3.228847688144594: 1, 3.22686237734487:
1, 3.225503861809552: 1, 3.224130372600413: 1, 3.223489594808947: 1, 3.221147
4026775977: 1, 3.220683010437355: 1, 3.2203939092998284: 1, 3.219879706043932
4: 1, 3.2195911411596634: 1, 3.217851487549831: 1, 3.2173156264870695: 1, 3.2
171672366296056: 1, 3.2162358126863384: 1, 3.2143798405572035: 1, 3.211489995
287541: 1, 3.210764110610659: 1, 3.2070082305834933: 1, 3.20654518578303: 1,
3.2062982943819334: 1, 3.2051083998684504: 1, 3.2012711437059513: 1, 3.199701
722940532: 1, 3.1994426193789876: 1, 3.1947707372695042: 1, 3.19235308886456
6: 1, 3.191340711915893: 1, 3.1879506711647316: 1, 3.186220577553465: 1, 3.18
5477750705471: 1, 3.184946814969137: 1, 3.18446329511366: 1, 3.18414714006033
1: 1, 3.183675505676135: 1, 3.183468563394527: 1, 3.182963781068622: 1, 3.182
0990233756445: 1, 3.181454738578036: 1, 3.179889562983152: 1, 3.1792320272762
113: 1, 3.1779255626176965: 1, 3.1760970796773607: 1, 3.175564219717438: 1,

3.172074578034958: 1, 3.171988144928786: 1, 3.170912540547573: 1, 3.162675608
968086: 1, 3.162499161389235: 1, 3.1608451506778454: 1, 3.1606142908128394:
1, 3.159908270884584: 1, 3.1594008901980697: 1, 3.1590694500484227: 1, 3.1580
293144052387: 1, 3.1576357743647727: 1, 3.1563873132855282: 1, 3.155671000192
046: 1, 3.1536971650253824: 1, 3.1528870978101775: 1, 3.152856156584778: 1,
3.152365381950786: 1, 3.1479719239203394: 1, 3.1478780785917504: 1, 3.1453727
806688434: 1, 3.1450381523808364: 1, 3.1445510857792742: 1, 3.142987205677870
5: 1, 3.14189732476816: 1, 3.141873406196252: 1, 3.1377381650884217: 1, 3.137
718387693465: 1, 3.1367231412748335: 1, 3.136711849387889: 1, 3.1365259597176
59: 1, 3.1352928619543263: 1, 3.134678335410646: 1, 3.1346222475306256: 1, 3.
134347957312495: 1, 3.133540970134794: 1, 3.1316194945309888: 1, 3.1311560499
04918: 1, 3.1304850739158105: 1, 3.1299681694349433: 1, 3.129505167966477: 1,
3.125677173622961: 1, 3.1252334527453183: 1, 3.1251307853913124: 1, 3.1246611
296288114: 1, 3.1244658682649007: 1, 3.124089823496191: 1, 3.123122485509303
4: 1, 3.1222774459629785: 1, 3.120721244739566: 1, 3.1206990894300177: 1, 3.1
204257919202356: 1, 3.1188796452447054: 1, 3.118298741902003: 1, 3.1175793437
65868: 1, 3.1174933791051846: 1, 3.117430056832864: 1, 3.114150826171266: 1,
3.1126322732712737: 1, 3.111392566491143: 1, 3.1112482146256464: 1, 3.1094724
114669714: 1, 3.109234382380126: 1, 3.10854491543284: 1, 3.108474496257982:
1, 3.107469521092138: 1, 3.105964893231871: 1, 3.1057153450860877: 1, 3.10548
0512702686: 1, 3.1054244487931584: 1, 3.105385813776185: 1, 3.10415882389756
6: 1, 3.10150962082176: 1, 3.1009277400522537: 1, 3.100485733063243: 1, 3.099
5909235460206: 1, 3.098391880748403: 1, 3.0978717870014765: 1, 3.097372213640
5364: 1, 3.0970510895829384: 1, 3.096316749401354: 1, 3.0959740910068265: 1,
3.093769758847907: 1, 3.0930776684718655: 1, 3.0920502153108878: 1, 3.0910444
37845272: 1, 3.0899069848571714: 1, 3.0890338261611876: 1, 3.084133328860392
3: 1, 3.0828962337592603: 1, 3.082767330534696: 1, 3.0817336870815066: 1, 3.0
791975045098563: 1, 3.0790638071263645: 1, 3.079044512619929: 1, 3.0786304682
495578: 1, 3.0777139158462705: 1, 3.0776921899722467: 1, 3.0770995718343372:
1, 3.07681064265501: 1, 3.075214778059387: 1, 3.074990284168287: 1, 3.0749845
264878246: 1, 3.0747401130871896: 1, 3.0730877064304623: 1, 3.071966996530157
3: 1, 3.0705329409784374: 1, 3.069741871819278: 1, 3.0696237696314874: 1, 3.0
694203972140834: 1, 3.068847897394299: 1, 3.0685061947800776: 1, 3.0684024772
88421: 1, 3.0683277070702446: 1, 3.067476920480704: 1, 3.0671710372804704: 1,
3.0663611344612267: 1, 3.065161123980513: 1, 3.0638780556597753: 1, 3.0635483
791303613: 1, 3.063325706205218: 1, 3.0629216633306355: 1, 3.062024237423180
8: 1, 3.0618459443927244: 1, 3.061605239232748: 1, 3.0606381418963085: 1, 3.0
59808022002755: 1, 3.0591307008277178: 1, 3.0582468136131205: 1, 3.0561427952
741473: 1, 3.055114490611363: 1, 3.0547675846875237: 1, 3.054704780562804: 1,
3.0538521901799314: 1, 3.0535500667374307: 1, 3.048715044780589: 1, 3.0454955
226447917: 1, 3.0454114237183996: 1, 3.043286835421276: 1, 3.043045477894013
5: 1, 3.039304497113852: 1, 3.038481843394443: 1, 3.03775386771554: 1, 3.0377
344005545295: 1, 3.035871037653111: 1, 3.0354887942516693: 1, 3.0350338861835
104: 1, 3.034880796469203: 1, 3.034010367367006: 1, 3.0330610758190435: 1, 3.
0320892380713: 1, 3.028464111325622: 1, 3.0276477602560896: 1, 3.027334132655
7475: 1, 3.0256842953807257: 1, 3.024962646130696: 1, 3.023755451948727: 1,
3.0231715495976776: 1, 3.019465785932438: 1, 3.019278573120539: 1, 3.01845745
1984462: 1, 3.018390083693978: 1, 3.016426917315078: 1, 3.0161994687900564:
1, 3.01373803042418: 1, 3.012927318280179: 1, 3.0126980798466163: 1, 3.012682
16457621: 1, 3.0108475504301633: 1, 3.0104718614106907: 1, 3.008894401393174
4: 1, 3.0075059288643664: 1, 3.006707752052764: 1, 3.0045515310151885: 1, 3.0
04505222902979: 1, 3.001536314109964: 1, 3.0013106287960283: 1, 3.00062260252
77285: 1, 3.0003331004528997: 1, 2.999206696975039: 1, 2.99872608898689: 1,
2.99840145285686: 1, 2.997385569597654: 1, 2.997233682620307: 1, 2.9971080190
664616: 1, 2.9968734263058496: 1, 2.990432225556929: 1, 2.990418663935056: 1,
2.9895641050468815: 1, 2.9890988306326394: 1, 2.98839307042479: 1, 2.98808560
71158255: 1, 2.986126922268396: 1, 2.9858253717762393: 1, 2.9854038752429677:

1, 2.984986139471904: 1, 2.9848564582069765: 1, 2.983249697764082: 1, 2.98315
2220988603: 1, 2.982338887190084: 1, 2.982288183090497: 1, 2.98185525201215:
1, 2.9814993332444195: 1, 2.979977888526672: 1, 2.9798643069028885: 1, 2.9785
211333077437: 1, 2.977624944574051: 1, 2.976503642276651: 1, 2.97605708154245
67: 1, 2.9750692764246884: 1, 2.9747932458271475: 1, 2.974534164043489: 1, 2.
9743803526402317: 1, 2.9737102438043257: 1, 2.9699541965274983: 1, 2.96941240
34780938: 1, 2.968320636261603: 1, 2.9669228144694673: 1, 2.9662883091601877:
1, 2.964963583150371: 1, 2.964462602077258: 1, 2.963758025355985: 1, 2.962541
5741434273: 1, 2.9615645615586006: 1, 2.960934573519716: 1, 2.96036889731377
9: 1, 2.959929990113009: 1, 2.9593812870401464: 1, 2.9577621561182963: 1, 2.9
575943752666167: 1, 2.9558985994811846: 1, 2.955381163282544: 1, 2.9552606482
685215: 1, 2.9546484685756513: 1, 2.9535404323655325: 1, 2.950940535731669:
1, 2.9499673339762778: 1, 2.948781181905707: 1, 2.9473771855462587: 1, 2.9453
29828076603: 1, 2.9441956343311713: 1, 2.9437513394077195: 1, 2.9430056562674
456: 1, 2.941951380305765: 1, 2.9419094317812107: 1, 2.9396033169345825: 1,
2.937867374881575: 1, 2.937284023193474: 1, 2.935459889217788: 1, 2.934706824
9004735: 1, 2.933469061276302: 1, 2.9311785229309852: 1, 2.929113458391628:
1, 2.928914464286277: 1, 2.928402970403928: 1, 2.9279753224981886: 1, 2.92693
92163257226: 1, 2.9249916208342635: 1, 2.9234396563523153: 1, 2.9218765586379
507: 1, 2.9195102667452235: 1, 2.918482241894868: 1, 2.917647984970033: 1, 2.
9174538183023553: 1, 2.916355431779406: 1, 2.916290563649819: 1, 2.9149110281
397776: 1, 2.9140577888380994: 1, 2.9117289182886097: 1, 2.909934021113669:
1, 2.9078994323250593: 1, 2.9056513831275743: 1, 2.90465558855279: 1, 2.90454
91930805034: 1, 2.9017594510888225: 1, 2.9011026252720993: 1, 2.9001830497109
324: 1, 2.8992099580520874: 1, 2.897075235111418: 1, 2.8965286067873497: 1,
2.8959182460394555: 1, 2.8950777810489376: 1, 2.8949803372059204: 1, 2.894873
6170872955: 1, 2.8948128629290437: 1, 2.894761040627998: 1, 2.89271024596143
9: 1, 2.8920452072462397: 1, 2.8915716788437034: 1, 2.8908619247274436: 1, 2.
889613105735967: 1, 2.8879803217573388: 1, 2.88651244422981: 1, 2.88531659665
86354: 1, 2.8839758631769867: 1, 2.878284106118619: 1, 2.8773418674344935: 1,
2.8769431625585913: 1, 2.8749921764062076: 1, 2.8742333139879785: 1, 2.872036
021886482: 1, 2.870681614313796: 1, 2.870249689927023: 1, 2.868972369668757:
1, 2.865329767656798: 1, 2.8645424014728653: 1, 2.863244492218289: 1, 2.86306
2845379296: 1, 2.8629466225826885: 1, 2.862765308234694: 1, 2.862401206491852
7: 1, 2.8609311491235716: 1, 2.858907583124747: 1, 2.8576651459248548: 1, 2.8
55395713741857: 1, 2.855299942757068: 1, 2.855130113313624: 1, 2.854395133296
731: 1, 2.854168568263203: 1, 2.8540390736437655: 1, 2.8538300351626296: 1,
2.853724432905531: 1, 2.852957922455186: 1, 2.85187429793087: 1, 2.8506506052
40165: 1, 2.850284333417922: 1, 2.8482629872406897: 1, 2.8465484056528436: 1,
2.845982917786289: 1, 2.845965613332903: 1, 2.84513871448343: 1, 2.8448366378
52339: 1, 2.844700142430424: 1, 2.844550301760421: 1, 2.8434118575297496: 1,
2.8431207034190646: 1, 2.841671489919765: 1, 2.841551076740976: 1, 2.84101601
2636784: 1, 2.839479487962679: 1, 2.8380296183000793: 1, 2.8380213679902733:
1, 2.8374366721188733: 1, 2.835279685219884: 1, 2.831857226524934: 1, 2.82952
63819222463: 1, 2.8282157614116032: 1, 2.826379007435803: 1, 2.82465142919686
45: 1, 2.8229904641917063: 1, 2.82288056183075: 1, 2.821512372385685: 1, 2.82
0296251434422: 1, 2.8181478208033415: 1, 2.8181108757856164: 1, 2.81755560056
58144: 1, 2.8167788394393254: 1, 2.8156396745458325: 1, 2.815638316208307: 1,
2.814385125205789: 1, 2.8139387737742307: 1, 2.8137468823063965: 1, 2.8130014
339587004: 1, 2.8118684472024316: 1, 2.810310034959467: 1, 2.810030001200716
7: 1, 2.8098125822477624: 1, 2.808628390917974: 1, 2.805928921173537: 1, 2.80
5079551267314: 1, 2.8036348174335175: 1, 2.8022978067068185: 1, 2.80215237779
58617: 1, 2.800976416978804: 1, 2.8003803710190542: 1, 2.8001806408543195: 1,
2.7994799425749495: 1, 2.798602319018746: 1, 2.7984549421815315: 1, 2.7979598
033691007: 1, 2.7973322651115793: 1, 2.796135241672007: 1, 2.795173846103226:
1, 2.7947176722209592: 1, 2.793236408529089: 1, 2.792875292454691: 1, 2.79216
5545549298: 1, 2.792156150628967: 1, 2.7909117029898294: 1, 2.790759793120366

3: 1, 2.7905934141375397: 1, 2.789761472130946: 1, 2.7893514798158625: 1, 2.7
889728421370688: 1, 2.7889461905451447: 1, 2.7888179651289233: 1, 2.787956641
3741457: 1, 2.7876152154937524: 1, 2.787233818741567: 1, 2.7867911600250825:
1, 2.786680319925138: 1, 2.786100675945473: 1, 2.785311128688366: 1, 2.78516
860568111: 1, 2.7847993424773683: 1, 2.7845847593354938: 1, 2.783273066291332
3: 1, 2.783110214115978: 1, 2.7826056815217926: 1, 2.7818646854660978: 1, 2.7
816705463833626: 1, 2.781470369334639: 1, 2.781293887623248: 1, 2.77970389573
8316: 1, 2.776898104626581: 1, 2.77606847084382: 1, 2.775699545334138: 1, 2.7
7536601739048: 1, 2.772436826378181: 1, 2.772311058674942: 1, 2.7719743766178
62: 1, 2.7678947721590155: 1, 2.7678533642957723: 1, 2.767408643647669: 1, 2.
7654351994941773: 1, 2.762849460988423: 1, 2.7609254746384013: 1, 2.760786218
6360896: 1, 2.7604719789692305: 1, 2.759219191016965: 1, 2.7589807932752066:
1, 2.7587637933224847: 1, 2.758411362157984: 1, 2.758102496117883: 1, 2.7566
423371787456: 1, 2.7556843320536712: 1, 2.7556146200269183: 1, 2.755340681899
314: 1, 2.7551111951108997: 1, 2.7541374680641075: 1, 2.753602564516128: 1,
2.7533162946704453: 1, 2.7526944147353007: 1, 2.752447960903112: 1, 2.751587
1460437213: 1, 2.7512058199308367: 1, 2.7509431190342375: 1, 2.74914413662129
2: 1, 2.7490007375169796: 1, 2.7475110726904313: 1, 2.7474325896454754: 1, 2.
745545781088269: 1, 2.745426199716602: 1, 2.7446679103043374: 1, 2.7436003989
22966: 1, 2.742265009311602: 1, 2.739499385663144: 1, 2.7392567945720194: 1,
2.7388961251169035: 1, 2.7383632472015402: 1, 2.737145546608984: 1, 2.737009
5619975485: 1, 2.736790954910234: 1, 2.7364209506880357: 1, 2.735357595833827
6: 1, 2.7353236546538247: 1, 2.7341376412569804: 1, 2.734045424701191: 1, 2.7
327269303628294: 1, 2.7315720744847902: 1, 2.731228377350036: 1, 2.7310340018
12907: 1, 2.7295305906737286: 1, 2.7293083070792097: 1, 2.7289933561172: 1,
2.7284755688629363: 1, 2.728406066217852: 1, 2.7282894230932886: 1, 2.728127
513649882: 1, 2.7272358731459603: 1, 2.725797038855683: 1, 2.723700527643940
5: 1, 2.7229009161911146: 1, 2.72269659718364: 1, 2.72107464194883: 1, 2.7208
12613493764: 1, 2.7200759282647162: 1, 2.7198641588847527: 1, 2.7191305121548
384: 1, 2.7185131074227904: 1, 2.7181294738116475: 1, 2.718021363620058: 1,
2.717722037592698: 1, 2.7176243177803197: 1, 2.7144568641623184: 1, 2.713980
0120373594: 1, 2.713592050270702: 1, 2.7130496900053616: 1, 2.71074384731823
9: 1, 2.710430656739331: 1, 2.709172294729122: 1, 2.709028021457894: 1, 2.708
5084478606696: 1, 2.7080139062433592: 1, 2.7069147537495155: 1, 2.70685804039
53773: 1, 2.7066598067633465: 1, 2.7063265895266997: 1, 2.706258268471376: 1,
2.7052592592951457: 1, 2.702939996967194: 1, 2.7028585743513642: 1, 2.7008193
374492655: 1, 2.6992560976579782: 1, 2.699015777131989: 1, 2.69825951213656:
1, 2.696952852984239: 1, 2.696626791005511: 1, 2.6963793259627806: 1, 2.6954
699322131694: 1, 2.6946060505772222: 1, 2.693082133445691: 1, 2.6921340276936
16: 1, 2.6919830264009663: 1, 2.6916079486106472: 1, 2.690785962654963: 1, 2.
687219632212665: 1, 2.687128476095228: 1, 2.686944243991179: 1, 2.68577583447
7386: 1, 2.6857465358287507: 1, 2.6847332627311697: 1, 2.6846293462685837: 1,
2.6845513951335014: 1, 2.683620882776957: 1, 2.6829621656712526: 1, 2.6815159
74642991: 1, 2.6807095370171847: 1, 2.680451189621116: 1, 2.6787184361115797:
1, 2.678354579138953: 1, 2.6764792534344624: 1, 2.6749847232028086: 1, 2.6746
814931976766: 1, 2.674540452121608: 1, 2.6736861983559344: 1, 2.6734317130374
854: 1, 2.672700150659753: 1, 2.6725930286192803: 1, 2.67110001286389: 1, 2.6
71082155789609: 1, 2.670843197591813: 1, 2.670792750109488: 1, 2.669928566098
8052: 1, 2.669497096611165: 1, 2.668545846245317: 1, 2.66844550383642: 1, 2.6
67476206737312: 1, 2.6648525813235726: 1, 2.66379044354554: 1, 2.663774614213
2697: 1, 2.6605200173781074: 1, 2.659924844308026: 1, 2.659005878877723: 1,
2.6564130021487213: 1, 2.6557414601695486: 1, 2.6554056098202374: 1, 2.65527
1054426064: 1, 2.654867141674708: 1, 2.653921242744153: 1, 2.653130820944860
5: 1, 2.6527983243504596: 1, 2.65214671226363: 1, 2.651827477574014: 1, 2.651
5255910251243: 1, 2.6501389535476902: 1, 2.6497689118334184: 1, 2.64795908259
91073: 1, 2.6476979588163636: 1, 2.6456701803291653: 1, 2.6456567712569576:
1, 2.644941471848917: 1, 2.6439547984210594: 1, 2.6427563144608976: 1, 2.642

5644786549936: 1, 2.6401941990174893: 1, 2.6382683671802662: 1, 2.63662437565
27195: 1, 2.635911103376237: 1, 2.6353534614883563: 1, 2.6353331712812142: 1,
2.634887349302025: 1, 2.6341805829749005: 1, 2.63347912233766: 1, 2.633413401
727093: 1, 2.6330275265251224: 1, 2.631172161571599: 1, 2.631082952220771: 1,
2.6307559122839583: 1, 2.629768470690665: 1, 2.629674228612818: 1, 2.62967105
3596896: 1, 2.629543906412375: 1, 2.6294926460943655: 1, 2.6294264801521274:
1, 2.62900901898329: 1, 2.628578151793672: 1, 2.627910573145743: 1, 2.626866
101187837: 1, 2.6268028250582143: 1, 2.626793241144326: 1, 2.626528263044150
4: 1, 2.6261472312418688: 1, 2.6249575706243915: 1, 2.6245797236035906: 1, 2.
622161648527487: 1, 2.62160544475091: 1, 2.6212760079724315: 1, 2.61930814991
79236: 1, 2.618809632938903: 1, 2.6172335612557376: 1, 2.616879632068264: 1,
2.6163714934918696: 1, 2.61614799220098: 1, 2.6153360975553173: 1, 2.6145201
835786955: 1, 2.6142022487116163: 1, 2.6140186992872025: 1, 2.61378616370466
9: 1, 2.6129712378942345: 1, 2.6118870896046156: 1, 2.6117334928578186: 1, 2.
6114016723182503: 1, 2.6109115001064533: 1, 2.6104056127176225: 1, 2.60852325
75815653: 1, 2.6059169392662604: 1, 2.6053361447452197: 1, 2.605207588228628:
1, 2.604128235825989: 1, 2.6039894716997263: 1, 2.603396752068775: 1, 2.60297
4172797063: 1, 2.602940439862121: 1, 2.600248424964497: 1, 2.599837227604725
2: 1, 2.5996551928579943: 1, 2.5995307108100394: 1, 2.5981558862012566: 1, 2.
596769363033813: 1, 2.5955806381779882: 1, 2.5941416184892616: 1, 2.593927924
0979127: 1, 2.593797417983296: 1, 2.593452061006338: 1, 2.5929607762039506:
1, 2.5925087515989182: 1, 2.591760251010664: 1, 2.5917471681893107: 1, 2.590
4919641013313: 1, 2.590461928479638: 1, 2.590323271214801: 1, 2.5898499014295
866: 1, 2.5881824544050502: 1, 2.5877875577735265: 1, 2.5874788663675754: 1,
2.5851886979145853: 1, 2.5851096362385126: 1, 2.584841309854407: 1, 2.584114
0716203435: 1, 2.5837953874050514: 1, 2.5831479835213904: 1, 2.58282612555141
44: 1, 2.582490155887853: 1, 2.582122814502457: 1, 2.5820375991146367: 1, 2.5
81922034151392: 1, 2.580859410363804: 1, 2.5794368584173815: 1, 2.57847169084
8224: 1, 2.5783243743012036: 1, 2.578286754446223: 1, 2.578236212675303: 1,
2.578075075617438: 1, 2.577649587943626: 1, 2.577107890976937: 1, 2.57671747
8567122: 1, 2.5765732981002714: 1, 2.576053394281894: 1, 2.574659607097805:
1, 2.5742974059680375: 1, 2.5739915957748023: 1, 2.5734293150401792: 1, 2.57
30096552021453: 1, 2.5724369812388344: 1, 2.5699687384590075: 1, 2.5698679699
117424: 1, 2.5696301832053763: 1, 2.569550432069374: 1, 2.568826805736648: 1,
2.5685213191824032: 1, 2.5684393785276134: 1, 2.567858165418807: 1, 2.5669014
183934324: 1, 2.566218382839375: 1, 2.5640128673645997: 1, 2.561834479309926:
1, 2.561255744011503: 1, 2.5601629273249302: 1, 2.5590742754207083: 1, 2.5576
72748478581: 1, 2.5576711857926067: 1, 2.5557718035399835: 1, 2.5538261282608
694: 1, 2.5534270204920135: 1, 2.553393930679397: 1, 2.5531275191571305: 1,
2.552855124617994: 1, 2.5527925422623046: 1, 2.5520974040833573: 1, 2.551258
850880245: 1, 2.550386095918229: 1, 2.5499699881061195: 1, 2.549745974751599:
1, 2.548411901975422: 1, 2.548196281461383: 1, 2.547546658252544: 1, 2.544835
4624081144: 1, 2.5443752176467163: 1, 2.544254987593379: 1, 2.54364255390120
7: 1, 2.543127862323392: 1, 2.5422183111489574: 1, 2.541664413710092: 1, 2.54
15548463934923: 1, 2.54138644827782: 1, 2.5400433944531344: 1, 2.539759898413
1635: 1, 2.5355057225132898: 1, 2.5336721225022827: 1, 2.531533555613999: 1,
2.5302721130340897: 1, 2.5302245433449055: 1, 2.529973698543443: 1, 2.529968
933846741: 1, 2.529355019735676: 1, 2.5278924378213707: 1, 2.527634774904121
3: 1, 2.526375984091288: 1, 2.52611838158263: 1, 2.526073170017722: 1, 2.5260
724548777334: 1, 2.525566617322722: 1, 2.525553191775407: 1, 2.52551807762256
87: 1, 2.524588310740134: 1, 2.5235203658715717: 1, 2.5233755840792615: 1, 2.
522490626163991: 1, 2.5218644416690124: 1, 2.5190945095965445: 1, 2.518410820
568361: 1, 2.5179324943163772: 1, 2.5172886797774017: 1, 2.5172833270250674:
1, 2.517151649019318: 1, 2.5169338491710533: 1, 2.516626535515929: 1, 2.5156
544776503624: 1, 2.5154213193649335: 1, 2.512203317785948: 1, 2.5108318092692
77: 1, 2.5105579611956004: 1, 2.510254248001604: 1, 2.509839611846096: 1, 2.5
098283091632294: 1, 2.5097495104713 47: 1, 2.5096621014424714: 1, 2.5092882839

538193: 1, 2.508549137952046: 1, 2.5085171404615734: 1, 2.5084317858235954:
1, 2.5078331227822446: 1, 2.507370536330371: 1, 2.5056672033840552: 1, 2.505
388098304003: 1, 2.504216530960137: 1, 2.503968917391986: 1, 2.50371913416589
24: 1, 2.5036574990298552: 1, 2.5034802465739325: 1, 2.5031410310628504: 1,
2.503130557336935: 1, 2.5027000478169015: 1, 2.5011828852136864: 1, 2.501168
521760339: 1, 2.4985733217215413: 1, 2.4982581766590415: 1, 2.498065981779540
8: 1, 2.497902436611655: 1, 2.497175759055428: 1, 2.4945991410702177: 1, 2.49
44921751258855: 1, 2.4938510648767824: 1, 2.4933478701246523: 1, 2.4930647021
060346: 1, 2.492983040675783: 1, 2.4910388064783575: 1, 2.4909281761892443:
1, 2.4900677010443384: 1, 2.489998510639618: 1, 2.4899816075376333: 1, 2.489
616563934302: 1, 2.4887074092070764: 1, 2.48756090305087: 1, 2.48714150332290
5: 1, 2.4868859345613474: 1, 2.485974850082968: 1, 2.4858467557147113: 1, 2.4
855720060136575: 1, 2.484573285686229: 1, 2.483641048286471: 1, 2.48128380255
12822: 1, 2.480107876552189: 1, 2.4799522290877354: 1, 2.4795303090045224: 1,
2.4790655316786543: 1, 2.476219467879466: 1, 2.475728570562142: 1, 2.47481826
9944728: 1, 2.474302851726004: 1, 2.473462090213015: 1, 2.472420846002211: 1,
2.471820551019254: 1, 2.471091509505482: 1, 2.4696512482898076: 1, 2.46939724
77085593: 1, 2.469258913926566: 1, 2.4687291976340386: 1, 2.4685228816738856:
1, 2.467209060921833: 1, 2.4652478987063136: 1, 2.465047501636462: 1, 2.46474
79989843735: 1, 2.464112670172248: 1, 2.4628583842799006: 1, 2.46268263329827
16: 1, 2.4624118804074313: 1, 2.462036959463438: 1, 2.4614664314922816: 1, 2.
4612979602266103: 1, 2.461206625632844: 1, 2.4610744705644505: 1, 2.459975444
580674: 1, 2.459522499243023: 1, 2.459270790934594: 1, 2.459161431958588: 1,
2.458694861037044: 1, 2.4578963394211537: 1, 2.457447639709256: 1, 2.4562853
3507385: 1, 2.4555039625597224: 1, 2.4553952260923153: 1, 2.4549372020309117:
1, 2.454307830561327: 1, 2.4535407423746953: 1, 2.4528791340118627: 1, 2.4522
612308286167: 1, 2.4522542616636493: 1, 2.451374059875604: 1, 2.4512993034115
627: 1, 2.450797026074641: 1, 2.4499024664192626: 1, 2.4493587327446806: 1,
2.4487927141183894: 1, 2.4487819378947613: 1, 2.4457951676411294: 1, 2.44579
0874201916: 1, 2.4451962135950684: 1, 2.444919689886275: 1, 2.44241158066698
6: 1, 2.4422589258410294: 1, 2.440686915580938: 1, 2.4403221183315917: 1, 2.4
40293754998184: 1, 2.4401737224645137: 1, 2.4393057725915774: 1, 2.4377359290
96412: 1, 2.4366464649268855: 1, 2.4363264084909426: 1, 2.4360848155918102:
1, 2.434892240822629: 1, 2.434710799682433: 1, 2.4341117570951747: 1, 2.4337
2537899865: 1, 2.432977655925336: 1, 2.429061141020672: 1, 2.42879878591374:
1, 2.4286617900255134: 1, 2.4283978363764795: 1, 2.427734798980728: 1, 2.427
6062144662363: 1, 2.426835071637671: 1, 2.425645027444746: 1, 2.4211340366019
76: 1, 2.4207665009154598: 1, 2.419323161028156: 1, 2.418502017440504: 1, 2.4
180993181357273: 1, 2.4180518003971256: 1, 2.4179773274112395: 1, 2.416808735
352408: 1, 2.4161116873672355: 1, 2.415953516542531: 1, 2.415147171965957: 1,
2.4150073620296717: 1, 2.4144088051890265: 1, 2.413764773291928: 1, 2.4131862
383297946: 1, 2.4127558361226615: 1, 2.4117549886691965: 1, 2.410700230172900
6: 1, 2.409920674716357: 1, 2.4092114847820207: 1, 2.409135778457719: 1, 2.40
8742795236063: 1, 2.4078010604748474: 1, 2.4067157129735595: 1, 2.40617990394
08136: 1, 2.40586234731292: 1, 2.4056724120198494: 1, 2.40510083966702: 1, 2.
402611003721757: 1, 2.401404340619735: 1, 2.400556882102531: 1, 2.39880855920
8457: 1, 2.3980848220478235: 1, 2.3979096141763083: 1, 2.3961760895531965: 1,
2.395994262559479: 1, 2.395955566999084: 1, 2.395379482013102: 1, 2.394703492
534144: 1, 2.393952403216368: 1, 2.392985408085834: 1, 2.392187767137513: 1,
2.3920177987269313: 1, 2.391647058217876: 1, 2.39083743559909: 1, 2.38920075
9857551: 1, 2.389167748239947: 1, 2.388959679501647: 1, 2.3884040343520097:
1, 2.3858080337006498: 1, 2.385007979974458: 1, 2.384121125843706: 1, 2.3829
37916373085: 1, 2.3817867877502272: 1, 2.381357859632073: 1, 2.38078470614954
53: 1, 2.3802896804884175: 1, 2.378877511723123: 1, 2.3782623798250833: 1, 2.
3780454673478184: 1, 2.3770246001846704: 1, 2.3766762584318983: 1, 2.37658453
03757183: 1, 2.3765160679632507: 1, 2.374922192116731: 1, 2.3738271848834365:
1, 2.371017744575309: 1, 2.370503970083: 1, 2.3701308581205205: 1, 2.36924484

6291537: 1, 2.368456170851237: 1, 2.3661656932477166: 1, 2.365166435619033:
1, 2.364790990542149: 1, 2.364655771543894: 1, 2.363479074423729: 1, 2.36340
8877613089: 1, 2.3631130883776668: 1, 2.3619746793474894: 1, 2.36174898695914
5: 1, 2.3606336909895194: 1, 2.360566909474881: 1, 2.3595945166081673: 1, 2.3
559628285942282: 1, 2.3559132719707407: 1, 2.355338579527016: 1, 2.3528355739
663547: 1, 2.352141694948249: 1, 2.3505503753271815: 1, 2.350373584929534: 1,
2.3499952334953638: 1, 2.3484229756117183: 1, 2.3480036811958485: 1, 2.347150
5930394088: 1, 2.347119703353058: 1, 2.347004297781605: 1, 2.345452081110662:
1, 2.345056407951556: 1, 2.344535546173547: 1, 2.344370760601079: 1, 2.343426
8012428987: 1, 2.3426300380911815: 1, 2.341380699885293: 1, 2.341047006525206
3: 1, 2.3379852486298054: 1, 2.337935093696245: 1, 2.3379029510573415: 1, 2.3
371643711704944: 1, 2.336966495643085: 1, 2.3357589466456865: 1, 2.3347958584
423507: 1, 2.3328945653862934: 1, 2.3328512239838632: 1, 2.3327543964503743:
1, 2.331047366365612: 1, 2.33043141326728: 1, 2.3298266040653894: 1, 2.32967
0007559562: 1, 2.329117344582304: 1, 2.3280334046755895: 1, 2.326529752967492
6: 1, 2.326321255368327: 1, 2.3246883889139287: 1, 2.324629320376911: 1, 2.32
34611333724673: 1, 2.322007779523034: 1, 2.321949593423077: 1, 2.321499401051
257: 1, 2.321348458422624: 1, 2.3197917869245965: 1, 2.3194186474302945: 1,
2.3190023820303574: 1, 2.3189680002389554: 1, 2.3188449281064827: 1, 2.31843
33850023378: 1, 2.3183176417029836: 1, 2.318086903797399: 1, 2.31728560099118
35: 1, 2.3168093763092004: 1, 2.316027776783859: 1, 2.3151697704409004: 1, 2.
314281253167689: 1, 2.3131489940996444: 1, 2.3124415983473914: 1, 2.311975062
7989566: 1, 2.31159650762352: 1, 2.3107578686877264: 1, 2.309464168206702: 1,
2.3087686111388295: 1, 2.306901231130944: 1, 2.3068639891616147: 1, 2.3068096
129335327: 1, 2.3067132134609047: 1, 2.3026902801872944: 1, 2.30180972286161
9: 1, 2.301521912952288: 1, 2.299937894146309: 1, 2.2993155624644976: 1, 2.29
76784536387407: 1, 2.2970288479565517: 1, 2.2964771809850126: 1, 2.2956335951
337676: 1, 2.2950737763088624: 1, 2.2950538469708954: 1, 2.2949098533969448:
1, 2.294459419347378: 1, 2.2942060143490837: 1, 2.293965979517805: 1, 2.2927
0442558567: 1, 2.2920135539255613: 1, 2.2912270645254296: 1, 2.29082730378445
55: 1, 2.2883564123182207: 1, 2.288117547357632: 1, 2.286350570642871: 1, 2.2
85234578437262: 1, 2.2835865097897625: 1, 2.2814218723995285: 1, 2.2801566908
92821: 1, 2.2783386977054687: 1, 2.2781264652168947: 1, 2.2763104295947754:
1, 2.275075598888401: 1, 2.2750338126291143: 1, 2.2742887390507622: 1, 2.274
1123094436944: 1, 2.2739555351908916: 1, 2.273340221987815: 1, 2.273263095617
6036: 1, 2.2719852573713806: 1, 2.2691852714959495: 1, 2.2691584803984948: 1,
2.2684883027899816: 1, 2.268050036807694: 1, 2.2674601452474947: 1, 2.2653643
42303015: 1, 2.2650880966713745: 1, 2.264159414245271: 1, 2.2634263506354793:
1, 2.262795191809403: 1, 2.2625064245926425: 1, 2.262243105931839: 1, 2.26202
90177848696: 1, 2.2607970839262186: 1, 2.2604340487495445: 1, 2.2594360234198
81: 1, 2.259124700326988: 1, 2.258415153469092: 1, 2.2582559908499213: 1, 2.2
58049607840047: 1, 2.2580052827107853: 1, 2.2572490936818554: 1, 2.2568204988
840916: 1, 2.2563659551676185: 1, 2.2552109806518863: 1, 2.254720099912399:
1, 2.2536319863104937: 1, 2.2530777998172207: 1, 2.252850588296286: 1, 2.251
649630376099: 1, 2.2500770653176527: 1, 2.248982827985684: 1, 2.2486093348704
053: 1, 2.2469655010683462: 1, 2.245677011468938: 1, 2.2455926192795643: 1,
2.2439300042642354: 1, 2.2426972286562927: 1, 2.241597049732362: 1, 2.239623
4071807855: 1, 2.2390006382499306: 1, 2.2389198948314974: 1, 2.23778575414904
4: 1, 2.2376977831145046: 1, 2.23723748539169: 1, 2.2368793871063866: 1, 2.23
68751088693: 1, 2.2367827987822975: 1, 2.2354391294592073: 1, 2.2353871062758
475: 1, 2.2348318213598066: 1, 2.2339010616486314: 1, 2.233861208081591: 1,
2.232612836552347: 1, 2.2325742087411444: 1, 2.23246907721462: 1, 2.23196491
64751825: 1, 2.2319546897336875: 1, 2.2313500132050375: 1, 2.231296625846335:
1, 2.2309620439414597: 1, 2.2309232257363774: 1, 2.230564360188486: 1, 2.2305
22634985562: 1, 2.230296087655464: 1, 2.2302180362466126: 1, 2.23012357382106
56: 1, 2.2270732196734824: 1, 2.2270116049271755: 1, 2.2266146301720284: 1,
2.226593932759463: 1, 2.2264904094463303: 1, 2.2263560918067125: 1, 2.226248

2331524214: 1, 2.2252878503398814: 1, 2.2242905544925806: 1, 2.22351477989857
9: 1, 2.222886156765813: 1, 2.221512100577024: 1, 2.2208883842764204: 1, 2.22
0312752845377: 1, 2.2189523334272425: 1, 2.2182733280526956: 1, 2.21816990051
97753: 1, 2.2180187548656276: 1, 2.2175654646284078: 1, 2.21685017864616: 1,
2.216810828086883: 1, 2.2155697444196396: 1, 2.2144273886353347: 1, 2.214072
9377200326: 1, 2.213626376010362: 1, 2.2132132648910754: 1, 2.211793901340433
5: 1, 2.2116129281336763: 1, 2.2097975207741314: 1, 2.209520779359044: 1, 2.2
09216672755121: 1, 2.2075990329907573: 1, 2.205098437464816: 1, 2.20481960214
206: 1, 2.2043104610253663: 1, 2.2038453340008477: 1, 2.2029758549674594: 1,
2.201683982698021: 1, 2.2007154141621936: 1, 2.2005892386279826: 1, 2.200294
0988890582: 1, 2.198728228221952: 1, 2.198451364750855: 1, 2.198194310911536:
1, 2.197590540767905: 1, 2.1950677721019813: 1, 2.194490930003144: 1, 2.19433
53920961184: 1, 2.1927279425940127: 1, 2.1926375787359147: 1, 2.1924112864229
66: 1, 2.1918883057350325: 1, 2.1917467488695013: 1, 2.191285346230482: 1, 2.
1912846821437104: 1, 2.1905839577058317: 1, 2.189954130248665: 1, 2.186701920
9495635: 1, 2.186293444394163: 1, 2.1850451612096857: 1, 2.184929766762263:
1, 2.18485650000776: 1, 2.1847093355977645: 1, 2.1831876171078: 1, 2.1820841
945010527: 1, 2.182064184724785: 1, 2.181928851770526: 1, 2.1817206281820076:
1, 2.1817113368797876: 1, 2.1812540871986985: 1, 2.181253543363195: 1, 2.1811
593198591064: 1, 2.1808935335301958: 1, 2.179535090886202: 1, 2.1793595685376
89: 1, 2.1788195546016205: 1, 2.1773084482413894: 1, 2.176542527717826: 1, 2.
1760735912583034: 1, 2.174271307637705: 1, 2.172320808291971: 1, 2.1721526028
206317: 1, 2.1699011991070662: 1, 2.1696102041236243: 1, 2.1686757611454315:
1, 2.1685399934393272: 1, 2.1683781265490802: 1, 2.1682276515198238: 1, 2.16
7860058059151: 1, 2.16739305229021: 1, 2.1669142497080545: 1, 2.1661028060668
417: 1, 2.1658701149384525: 1, 2.165749260097217: 1, 2.165637539977086: 1, 2.
1648045715971116: 1, 2.1625786065764703: 1, 2.162387463188617: 1, 2.1622611252
259865: 1, 2.162068970139419: 1, 2.1614693228271893: 1, 2.1605092488483333:
1, 2.159810472694095: 1, 2.1596334466823777: 1, 2.159248594602619: 1, 2.1565
76769947093: 1, 2.156450842258564: 1, 2.1562435632160004: 1, 2.15576528416057
8: 1, 2.155544538833871: 1, 2.153950070498751: 1, 2.1536629722039646: 1, 2.15
3611952214934: 1, 2.152808000034268: 1, 2.152727465858637: 1, 2.1525808225367
02: 1, 2.1518896942491286: 1, 2.1516658357007112: 1, 2.1490372243963267: 1,
2.1485746786026985: 1, 2.147634675383611: 1, 2.1473610553225764: 1, 2.147154
5038840785: 1, 2.146487052990492: 1, 2.146352829556906: 1, 2.146189441629272:
1, 2.146110725732236: 1, 2.1445739713091996: 1, 2.1444943953821483: 1, 2.1443
434125185363: 1, 2.1436933288234536: 1, 2.1430321589388273: 1, 2.142865329161
2774: 1, 2.1417660476922977: 1, 2.1407762297164257: 1, 2.1406263216474755: 1,
2.140577167548156: 1, 2.1405247003168517: 1, 2.1396736127696667: 1, 2.1382635
33911778: 1, 2.1381483146469824: 1, 2.1355482961864825: 1, 2.134565030203158:
1, 2.1341502112754815: 1, 2.1331290070894067: 1, 2.132576932563131: 1, 2.1322
77679504181: 1, 2.132246128860255: 1, 2.1317377724110496: 1, 2.13162609676110
56: 1, 2.1300249669440223: 1, 2.1298875961125785: 1, 2.129181754465213: 1, 2.
128829794550605: 1, 2.1287979808483195: 1, 2.1283191937907198: 1, 2.128006714
8222326: 1, 2.1278090645379897: 1, 2.1277668884566343: 1, 2.1275929676686407:
1, 2.126770970977059: 1, 2.1263868264079693: 1, 2.1263051092124536: 1, 2.1229
250337802994: 1, 2.121541862900297: 1, 2.12125174643212: 1, 2.119849125442644
6: 1, 2.1195065010028906: 1, 2.1176811713231594: 1, 2.1171596234150747: 1, 2.
116527683274492: 1, 2.116426029667625: 1, 2.115571204400763: 1, 2.11459418693
83513: 1, 2.11419282326794: 1, 2.1140684263586333: 1, 2.114015051163443: 1,
2.113921201094029: 1, 2.113608036301838: 1, 2.113259778878747: 1, 2.11208484
76433207: 1, 2.1105268267482606: 1, 2.110034311411958: 1, 2.109519056373773:
1, 2.109412340873173: 1, 2.109334171178952: 1, 2.108857940745446: 1, 2.10835
63267586958: 1, 2.1082746867669915: 1, 2.1070058482658: 1, 2.106712959546032
3: 1, 2.1060849569792435: 1, 2.105743824289963: 1, 2.104973108951847: 1, 2.10
4420095852345: 1, 2.103809889255505: 1, 2.1034241150584188: 1, 2.102422670692
6648: 1, 2.101571056240637: 1, 2.1007320717944804: 1, 2.0996629101151996: 1,

2.099197403086834: 1, 2.098993342094193: 1, 2.0979408205970196: 1, 2.0962720
31908678: 1, 2.095868130512034: 1, 2.09543157625123: 1, 2.095049338536023: 1,
2.0944738470056987: 1, 2.094403885773088: 1, 2.0931146646970404: 1, 2.0921043
491300706: 1, 2.0914858570803005: 1, 2.0913186673242707: 1, 2.089863680058544
4: 1, 2.0893179595409226: 1, 2.0890455105106076: 1, 2.0860170085985814: 1, 2.
085889370571056: 1, 2.085467588068422: 1, 2.0854409817359993: 1, 2.0853036042
944: 1, 2.084026154744722: 1, 2.083896158202133: 1, 2.0814775762451827: 1, 2.
081202290645922: 1, 2.079968082959496: 1, 2.0796014550312836: 1, 2.0787914454
714254: 1, 2.078162263607473: 1, 2.0776029961065814: 1, 2.076644494717714: 1,
2.076429217715119: 1, 2.075836057785826: 1, 2.0747257799086842: 1, 2.07364015
64570697: 1, 2.071150954026274: 1, 2.0702751205361243: 1, 2.0700441430919936:
1, 2.06926409344589: 1, 2.068298832440469: 1, 2.067251492264893: 1, 2.0666005
12405013: 1, 2.0664850686438947: 1, 2.066425203935483: 1, 2.0658792008420703:
1, 2.065546850595764: 1, 2.0652356343253815: 1, 2.0637385736409493: 1, 2.0636
908571120207: 1, 2.063232146054546: 1, 2.0618578060409645: 1, 2.0617636054697
246: 1, 2.0615138261451245: 1, 2.061429229815901: 1, 2.060641044358223: 1, 2.
059541205373171: 1, 2.0578726857606764: 1, 2.0576310733032477: 1, 2.057613267
552275: 1, 2.057595400318793: 1, 2.0563733876595722: 1, 2.0557599887697955:
1, 2.0557348886820206: 1, 2.054050227650936: 1, 2.0537587473789367: 1, 2.053
4398546990666: 1, 2.053179460874313: 1, 2.0523148334659007: 1, 2.051429259834
448: 1, 2.0509003160778767: 1, 2.050591206734235: 1, 2.0501079219452825: 1,
2.050023109842717: 1, 2.0498553911116604: 1, 2.0485398782367588: 1, 2.046066
1099018633: 1, 2.04566562668559: 1, 2.0456655477649113: 1, 2.045065395078365
7: 1, 2.0435943591196506: 1, 2.0432677960963934: 1, 2.0432632192704467: 1, 2.
0431078892956056: 1, 2.042473106588619: 1, 2.042385410983558: 1, 2.0416383548
1138: 1, 2.0413947041825575: 1, 2.0408125690473202: 1, 2.0406513154832533: 1,
2.0402433851061548: 1, 2.0400961695952358: 1, 2.0397822567401382: 1, 2.037304
15245459: 1, 2.0364435872708793: 1, 2.0355761295796317: 1, 2.03377350462997:
1, 2.0335525330319304: 1, 2.0335406543053147: 1, 2.0327035231485864: 1, 2.03
19002324388706: 1, 2.031760547589357: 1, 2.0304192756271706: 1, 2.02999402409
96386: 1, 2.0298936556675478: 1, 2.029425023429708: 1, 2.028841060522711: 1,
2.028494219596149: 1, 2.027511376359664: 1, 2.0255762554700127: 1, 2.0247321
911540044: 1, 2.0245210901235544: 1, 2.024187046894208: 1, 2.024098968422449
7: 1, 2.0240735990054204: 1, 2.0239823714863197: 1, 2.023564039734413: 1, 2.0
226707306053036: 1, 2.021707329343273: 1, 2.021410178732674: 1, 2.02038267346
12798: 1, 2.020314234828095: 1, 2.0196010911798568: 1, 2.019279419683274: 1,
2.017090402720789: 1, 2.0163728103771543: 1, 2.0156778249293255: 1, 2.015614
3782502514: 1, 2.014237720136596: 1, 2.013180187480817: 1, 2.012665895960493
5: 1, 2.0122759339481906: 1, 2.0116820871339214: 1, 2.0105463192932245: 1, 2.
009703409198558: 1, 2.009688964011095: 1, 2.0088468797303696: 1, 2.0060831705
905753: 1, 2.005859130022385: 1, 2.005694877441465: 1, 2.0035781963546526: 1,
2.002956095768888: 1, 2.002944428654173: 1, 2.00275877596744: 1, 2.0025246107
731656: 1, 2.0020572529159244: 1, 2.001336188216692: 1, 2.0010761601670106:
1, 2.0008237144795644: 1, 2.0006552180339128: 1, 2.000086208866588: 1, 1.999
8864859471632: 1, 1.9995113391606016: 1, 1.9994843156334288: 1, 1.99943293196
54267: 1, 1.9991718612643068: 1, 1.9985690455017118: 1, 1.9983499863920506:
1, 1.9981599283105107: 1, 1.9977289563926686: 1, 1.997433103851463: 1, 1.994
7347151704384: 1, 1.992387036989254: 1, 1.99203600230695: 1, 1.99067693072226
38: 1, 1.9891627402814738: 1, 1.9881242364482468: 1, 1.986816858705243: 1, 1.
9862960463276373: 1, 1.9856692015191937: 1, 1.9841045386789344: 1, 1.98396986
76761048: 1, 1.983693981627989: 1, 1.9830388189230157: 1, 1.9816805606884742:
1, 1.9814109787958614: 1, 1.9808727398407444: 1, 1.9795944716904916: 1, 1.978
8289253876932: 1, 1.978369465020696: 1, 1.9776173377874273: 1, 1.977293862509
8038: 1, 1.9768391024690515: 1, 1.9767054162555733: 1, 1.9762160981804364: 1,
1.9757672187048514: 1, 1.9746933002939147: 1, 1.9729066869486984: 1, 1.972152
6411198484: 1, 1.9713117025469322: 1, 1.9712906045734728: 1, 1.97076144420225
65: 1, 1.9692805926113743: 1, 1.9691700149727023: 1, 1.9690201470578033: 1,

1.9686763316423008: 1, 1.9657041118651615: 1, 1.9655431808511814: 1, 1.96550
03898828378: 1, 1.9652356313871202: 1, 1.9642699323947141: 1, 1.9618309757081
092: 1, 1.9592962469124522: 1, 1.959255255416912: 1, 1.9592386276868194: 1,
1.95902824922407: 1, 1.9582555838764961: 1, 1.9582183006259584: 1, 1.9577538
546418436: 1, 1.9576276405256479: 1, 1.9572131972968543: 1, 1.95627322472124
7: 1, 1.956050485275134: 1, 1.9552659743564447: 1, 1.9534151920982663: 1, 1.9
532856239656955: 1, 1.9531239287395126: 1, 1.953053322692245: 1, 1.9526263356
731617: 1, 1.9519742160515676: 1, 1.9502447897100708: 1, 1.9502251673697841:
1, 1.950125412250924: 1, 1.9473743976990385: 1, 1.9470459040784969: 1, 1.946
3267454050972: 1, 1.9456847988965198: 1, 1.9452665237199684: 1, 1.94522608289
69143: 1, 1.944535095268645: 1, 1.943605621258318: 1, 1.9425692208585732: 1,
1.9415456330970944: 1, 1.9415268552102734: 1, 1.9412564293442125: 1, 1.93967
36369999017: 1, 1.939649184176965: 1, 1.938660931333991: 1, 1.938556893778934
1: 1, 1.9376251807004088: 1, 1.9371978812513742: 1, 1.9370069385477604: 1, 1.
9349643500070117: 1, 1.9343687328869423: 1, 1.9342913913050264: 1, 1.93367135
58029877: 1, 1.9333533880521547: 1, 1.931808185975562: 1, 1.9315655044545308:
1, 1.9310526172510412: 1, 1.9306330546322976: 1, 1.9294799472468875: 1, 1.928
379546185982: 1, 1.9275111278731483: 1, 1.9273708997002428: 1, 1.927326516902
2415: 1, 1.927087973003376: 1, 1.9265725132739115: 1, 1.9263633471957953: 1,
1.9259727183430766: 1, 1.9252905402358207: 1, 1.924164633236224: 1, 1.923176
1752291268: 1, 1.9223420686241575: 1, 1.9221028014659503: 1, 1.92203707306181
22: 1, 1.9216106715042793: 1, 1.9206756079364682: 1, 1.920384991792048: 1, 1.
9203520138181784: 1, 1.9189160636522022: 1, 1.918141052978912: 1, 1.918001905
3655493: 1, 1.9170138439638897: 1, 1.91699298730285: 1, 1.9169191729254602:
1, 1.915959538854817: 1, 1.9155397181603462: 1, 1.9153344570202926: 1, 1.915
036631284079: 1, 1.9136916718676857: 1, 1.9133598457844325: 1, 1.913324989813
8434: 1, 1.9128409517505682: 1, 1.9125836462993557: 1, 1.9124622057159217: 1,
1.9114021219240125: 1, 1.910839562902678: 1, 1.9105609522378577: 1, 1.9094874
16516926: 1, 1.9085133852705145: 1, 1.908202001814092: 1, 1.9081917417457368:
1, 1.9080065621201565: 1, 1.9077340954534623: 1, 1.9073429467710854: 1, 1.906
7580104553135: 1, 1.906392767021213: 1, 1.9057568485951548: 1, 1.905754754361
7495: 1, 1.9053853405507077: 1, 1.9053625678455717: 1, 1.9050162689433914: 1,
1.9050029308897283: 1, 1.9039902450565167: 1, 1.9028361353537666: 1, 1.902748
3275083594: 1, 1.9024150622413243: 1, 1.9021146630284294: 1, 1.90085971058594
57: 1, 1.900820442509248: 1, 1.900659369757913: 1, 1.8991477396193874: 1, 1.8
987256628959321: 1, 1.8985811657999425: 1, 1.898550025694208: 1, 1.8977181725
443766: 1, 1.8963306628029362: 1, 1.8962419482318855: 1, 1.8958701451221796:
1, 1.8954254758813915: 1, 1.8949148682597132: 1, 1.8944580681839487: 1, 1.89
39809341571294: 1, 1.8936591625510024: 1, 1.8923672932526125: 1, 1.8922364245
811019: 1, 1.8920956032250464: 1, 1.8917531746156586: 1, 1.8917379741631373:
1, 1.8907728559537875: 1, 1.89039068246262: 1, 1.890171849866281: 1, 1.88996
05067314764: 1, 1.8899423402889115: 1, 1.8898903681551982: 1, 1.8895662016483
699: 1, 1.8889683218472095: 1, 1.8888777070755678: 1, 1.888563751609186: 1,
1.8873247726353555: 1, 1.887110770651912: 1, 1.8855640891450773: 1, 1.884894
8663707468: 1, 1.8847690803262644: 1, 1.8841739838181573: 1, 1.88208258626939
33: 1, 1.8816808966679588: 1, 1.881143484109114: 1, 1.881024413592924: 1, 1.8
809145087743435: 1, 1.8807770434501498: 1, 1.8802498103202419: 1, 1.879905566
616684: 1, 1.8790511599283637: 1, 1.8775275658294037: 1, 1.8770215612095542:
1, 1.8764014590591829: 1, 1.8763003921001877: 1, 1.8758521091975486: 1, 1.87
5137129030615: 1, 1.8729800220407178: 1, 1.8722942190690655: 1, 1.87192831428
01035: 1, 1.8714505567489952: 1, 1.8703634990730054: 1, 1.87000684274862: 1,
1.8690880405687942: 1, 1.8682235083059935: 1, 1.867812558793672: 1, 1.866717
4412877303: 1, 1.866280490620581: 1, 1.866104500512291: 1, 1.865985049639117
7: 1, 1.8649643622369436: 1, 1.8636388859913713: 1, 1.8615992608243281: 1, 1.
8609679253679585: 1, 1.8599899195948097: 1, 1.8594766315813869: 1, 1.85912389
31404055: 1, 1.8590109565197388: 1, 1.858112336326802: 1, 1.8580170721949598:
1, 1.8578013853382087: 1, 1.8565695704404763: 1, 1.8563510211080538: 1, 1.855

```
2684757896714: 1, 1.8546080472249813: 1, 1.8534401714100848: 1, 1.85172051323
74743: 1, 1.8516775416876663: 1, 1.8515714709614428: 1, 1.8506048677121782:
 1, 1.850445062565258: 1, 1.8485613715358553: 1, 1.8481597453110732: 1, 1.848
0763301521415: 1, 1.8455695206373945: 1, 1.8445633954533407: 1, 1.84450377028
0843: 1, 1.8440085262146289: 1, 1.8435059860583998: 1, 1.843487628762069: 1,
 1.8418714997499828: 1, 1.841850286063551: 1, 1.838367031677153: 1, 1.8377838
88665482: 1, 1.8376306309518506: 1, 1.8367316510576799: 1, 1.836500802886502
6: 1, 1.834949429577712: 1, 1.834781726747892: 1, 1.8338462487588585: 1, 1.83
37245215279159: 1, 1.8334571410066918: 1, 1.8333858717235532: 1, 1.8330901531
340185: 1, 1.8324772054045277: 1, 1.8313693725154874: 1, 1.8310475533611843:
 1, 1.8299692209364253: 1, 1.8290389742720345: 1, 1.8285898740136894: 1, 1.82
85824181247035: 1, 1.8284686941757229: 1, 1.8279259529273528: 1, 1.8278579022
568056: 1, 1.8276011369518619: 1, 1.8274958256536808: 1, 1.8267641483344157:
 1, 1.8265427188331564: 1, 1.8259583028996067: 1, 1.8243269112945417: 1, 1.82
38463885431104: 1, 1.8236428937550095: 1, 1.8234127389225248: 1, 1.8231938563
016892: 1, 1.8227874190203048: 1, 1.8222528744804516: 1, 1.8222179884627925:
 1, 1.8220494359267214: 1, 1.8214728940577418: 1, 1.8205799668325813: 1, 1.82
04259953644168: 1, 1.8202261127715993: 1, 1.8196052761534054: 1, 1.8194356327
901922: 1, 1.8186898048468674: 1, 1.8181149734974766: 1, 1.81735759991988: 1,
1.8161253305924105: 1, 1.815845378488225: 1, 1.8156264553758479: 1, 1.8155029
5111885: 1, 1.8151958517876499: 1, 1.8150873020850238: 1, 1.8141126309360145:
1, 1.8133819151345134: 1, 1.812179868824929: 1, 1.8104348719817909: 1, 1.8085
835725474821: 1, 1.8074924139239243: 1, 1.805817171840297: 1, 1.8052746755376
958: 1, 1.8033811402030253: 1, 1.8021735602192293: 1, 1.8019732769248076: 1,
 1.801512550260943: 1, 1.8002678216513168: 1, 1.8002052393691037: 1, 1.800056
343078203: 1, 1.7998318589194096: 1, 1.7986481949304027: 1, 1.798344792689937
9: 1, 1.7982011798160584: 1, 1.7980809217464924: 1, 1.796603601449976: 1, 1.7
965585491988783: 1, 1.7965461522371837: 1, 1.7963482342132442: 1, 1.795884091
1740759: 1, 1.7948903315870564: 1, 1.794387207925586: 1, 1.7939691330356302:
 1, 1.79380640941425: 1, 1.7914874950203943: 1, 1.7914768614105612: 1, 1.7910
386985277262: 1, 1.7909816189282142: 1, 1.7899842090767175: 1, 1.789751578234
0485: 1, 1.789566115262258: 1, 1.788988566995809: 1, 1.7889516862866568: 1,
 1.7877914379759319: 1, 1.7860414354683183: 1, 1.7856403573736905: 1, 1.78534
44834736998: 1, 1.7851056640751932: 1, 1.7850848969051474: 1, 1.7847881985688
66: 1, 1.7847652342546017: 1, 1.784211558341392: 1, 1.783875587254927: 1, 1.7
837372446937099: 1, 1.7834935223095572: 1, 1.7834575824310885: 1, 1.782165841
7862705: 1, 1.7807789248090695: 1, 1.7802386313974379: 1, 1.7801213257822326:
1, 1.779982750197744: 1, 1.7798341441710697: 1, 1.779787453362186: 1, 1.77827
26310299997: 1, 1.775850302354053: 1, 1.774977113188567: 1, 1.774460531086019
3: 1, 1.7741526788845667: 1, 1.7739399873111306: 1, 1.7735334365517414: 1, 1.
7724153340094628: 1, 1.7722218494956545: 1, 1.771997678028631: 1, 1.771976071
3645136: 1, 1.7713822017040421: 1, 1.7700855423884054: 1, 1.7697355694637875:
1, 1.7696298536640227: 1, 1.7695728441070315: 1, 1.7694642226658928: 1, 1.769
443316026344: 1, 1.766904936642682: 1, 1.7655187882670804: 1, 1.7654967954800
08: 1, 1.7654213105562544: 1, 1.7649843337900537: 1, 1.764336448119524: 1, 1.
7642102786155092: 1, 1.7636354360545996: 1, 1.7585423906205946: 1, 1.75780750
48040145: 1, 1.7575314273890708: 1, 1.7571407856512713: 1, 1.754807892768738
5: 1, 1.7544279294316891: 1, 1.7540719979775823: 1, 1.7535515588648416: 1, 1.
7532161892582212: 1, 1.7528044782228767: 1, 1.752633623108324: 1, 1.75188487
9585553: 1, 1.7518160000910605: 1, 1.750346331878386: 1, 1.7494478092299632:
 1, 1.7473936661778846: 1, 1.7463959975970065: 1, 1.7449659818521448: 1, 1.74
4815521282718: 1, 1.7436764976234054: 1, 1.7432193476872153: 1, 1.74308894655
39945: 1, 1.7414613359847184: 1, 1.7404608084858457: 1, 1.7384726066308696:
 1, 1.737745892125787: 1, 1.7375302147281981: 1, 1.737247712438745: 1, 1.7365
48313792752: 1, 1.7362649568720538: 1, 1.7357326070582806: 1, 1.7356874494925
703: 1, 1.7355881175222638: 1, 1.7350271231358259: 1, 1.7342585552895453: 1,
 1.7332983101330144: 1, 1.7330700423180339: 1, 1.7326354043660976: 1, 1.73111
```

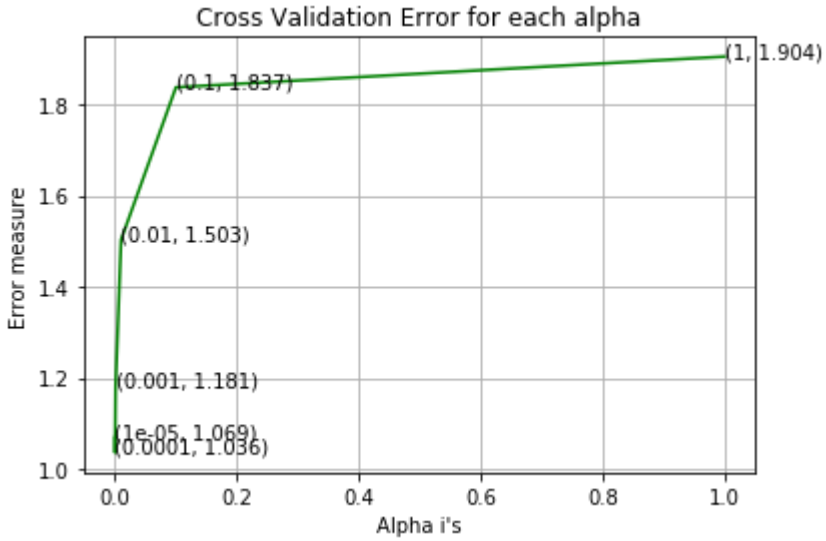
75073435516: 1, 1.7309458287248847: 1, 1.7308160837936937: 1, 1.7303738373397
146: 1, 1.7301534859872543: 1, 1.7295120433739946: 1, 1.7291376931087037: 1,
1.7291178131862637: 1, 1.7279779070103147: 1, 1.7277300032423737: 1, 1.72753
89113886443: 1, 1.7268058215605102: 1, 1.7265509656081646: 1, 1.7244888250381
22: 1, 1.7229089185049615: 1, 1.721899815102905: 1, 1.7200170762346483: 1, 1.
7197050197300476: 1, 1.7197013696174384: 1, 1.7186656664632851: 1, 1.71830724
2698414: 1, 1.718259919732148: 1, 1.7178408405626695: 1, 1.7164988459760386:
1, 1.7128208085488332: 1, 1.7122934041566882: 1, 1.71099556569979: 1, 1.7108
841063340958: 1, 1.7099440204723655: 1, 1.7097792890534893: 1, 1.709642906958
7034: 1, 1.7093981536978418: 1, 1.7073326513929274: 1, 1.707132420360462: 1,
1.707093719806416: 1, 1.7070415344724985: 1, 1.7066795711168217: 1, 1.705802
9463127984: 1, 1.70573256745827: 1, 1.7053555510155056: 1, 1.704688395224572
6: 1, 1.7046061165238229: 1, 1.7034505792025407: 1, 1.6996647519767918: 1, 1.
6995113570781932: 1, 1.6990690817014604: 1, 1.6985777473082324: 1, 1.69855596
59473364: 1, 1.6977212105054489: 1, 1.696994723528094: 1, 1.696682884644789:
1, 1.6958579673485705: 1, 1.6942845487290479: 1, 1.6940431732486345: 1, 1.69
3439411386499: 1, 1.6931053738429387: 1, 1.6930499965037025: 1, 1.69217670512
474: 1, 1.6913097259994823: 1, 1.6912718065858958: 1, 1.6893213695655376: 1,
1.6888364391715418: 1, 1.6884621043148427: 1, 1.688151039257078: 1, 1.687562
2338193725: 1, 1.6858940652331598: 1, 1.6857189118457252: 1, 1.68417165105986
32: 1, 1.6839562174583735: 1, 1.6838866860021344: 1, 1.683370895841538: 1, 1.
6821040930918825: 1, 1.6816636899359358: 1, 1.681661080415694: 1, 1.680884319
5050414: 1, 1.6807312131330296: 1, 1.6794000291509419: 1, 1.676905298298222:
1, 1.6747420049240216: 1, 1.6746887506593942: 1, 1.6745854538735558: 1, 1.67
30034184856282: 1, 1.6720962464863705: 1, 1.6713556508527625: 1, 1.6712360343
843589: 1, 1.6701079914059533: 1, 1.6694201191115596: 1, 1.6692755061558526:
1, 1.66884097248144: 1, 1.6688294668819226: 1, 1.66882449169769: 1, 1.666953
619568901: 1, 1.666519306586901: 1, 1.6651489313036278: 1, 1.664380342888950
5: 1, 1.6643613375431734: 1, 1.6609227564879858: 1, 1.6604608060241541: 1, 1.
6593279901213223: 1, 1.658201949744692: 1, 1.6561676438124915: 1, 1.652677582
643481: 1, 1.6516952792007675: 1, 1.651063309871674: 1, 1.6493393461044206:
1, 1.6485979286313241: 1, 1.6483488407636897: 1, 1.6478838032281211: 1, 1.64
77951596870026: 1, 1.6476330804423682: 1, 1.6469338937660525: 1, 1.6466727210
332555: 1, 1.6464360296588543: 1, 1.6462998054255298: 1, 1.64572875802465: 1,
1.645354909986617: 1, 1.6437714225575155: 1, 1.6427676200559773: 1, 1.6426975
408871363: 1, 1.6425749937942398: 1, 1.6422232654285946: 1, 1.641349837152638
5: 1, 1.6410793391712661: 1, 1.6410421405530515: 1, 1.6393289904679846: 1, 1.
637841148515865: 1, 1.6351530857484824: 1, 1.6350795212266773: 1, 1.635010168
0304974: 1, 1.634939790796282: 1, 1.6346604817664443: 1, 1.634213649678681:
1, 1.6340705020381643: 1, 1.634031983364873: 1, 1.633672730371482: 1, 1.6335
186630886132: 1, 1.6315940622092098: 1, 1.6293449806020324: 1, 1.626316890754
141: 1, 1.6250375978565879: 1, 1.6239917417486613: 1, 1.6239846755034861: 1,
1.6236887735934913: 1, 1.6236230478488758: 1, 1.6234888519095143: 1, 1.62256
14728382476: 1, 1.6209365119602308: 1, 1.6193447924209503: 1, 1.6168613578389
903: 1, 1.6167637384874587: 1, 1.616396338517596: 1, 1.6163304188292813: 1,
1.616200135901714: 1, 1.6154914406879468: 1, 1.6149751201198574: 1, 1.614824
9027261252: 1, 1.6142131302946094: 1, 1.6138408663230104: 1, 1.61297248168091
9: 1, 1.6116207177521304: 1, 1.6115174514956458: 1, 1.608112515429976: 1, 1.6
08043750567257: 1, 1.6076278209277228: 1, 1.6065732046760712: 1, 1.6054292866
900985: 1, 1.6039052836075645: 1, 1.603530572971433: 1, 1.6024722292879965:
1, 1.6016387916288937: 1, 1.5997784807656519: 1, 1.59968150123973: 1, 1.5992
726065561753: 1, 1.5981111645595487: 1, 1.597170995026282: 1, 1.5952530928135
324: 1, 1.5950900809021598: 1, 1.594712037009399: 1, 1.594365023859119: 1, 1.
5943604440433115: 1, 1.5939736305612333: 1, 1.5933207953688986: 1, 1.59190472
20482232: 1, 1.5918352479120883: 1, 1.5915517255337561: 1, 1.591165857101327
4: 1, 1.5908819122878186: 1, 1.590133631346384: 1, 1.588778063597838: 1, 1.58
83873222191756: 1, 1.5870307209606844: 1, 1.5869008425087952: 1, 1.5866833775

694085: 1, 1.5837965635635816: 1, 1.5832364378931987: 1, 1.582344271063895:
1, 1.5815955046233405: 1, 1.5813659933291127: 1, 1.5811655250542953: 1, 1.58
10224678378808: 1, 1.5801251104203793: 1, 1.5796362152188648: 1, 1.5785817606
518928: 1, 1.578307963784805: 1, 1.5775925595841407: 1, 1.576442324785501: 1,
1.5759771838876597: 1, 1.5755687271163363: 1, 1.5751912936816244: 1, 1.575120
2594734057: 1, 1.5743040877766137: 1, 1.5710268854411982: 1, 1.56910596020431
4: 1, 1.5690213819409091: 1, 1.5671857433838627: 1, 1.56597280198833: 1, 1.56
58604764348827: 1, 1.5637205846010436: 1, 1.5633050021902766: 1, 1.5612512161
960281: 1, 1.5609207557102405: 1, 1.5607712511881333: 1, 1.5606420047724234:
1, 1.5605947814982057: 1, 1.5604930867338438: 1, 1.559208472242492: 1, 1.558
3747843377898: 1, 1.5571018223395334: 1, 1.5564679432336823: 1, 1.55460956407
39772: 1, 1.5545797472443867: 1, 1.5542389329529238: 1, 1.5541512599390073:
1, 1.5536299232670514: 1, 1.5526390542069526: 1, 1.5517745287931668: 1, 1.55
02075682057137: 1, 1.5500689654205062: 1, 1.5470441595570472: 1, 1.5465619915
583704: 1, 1.5461351063090256: 1, 1.5443366538528298: 1, 1.5435146549345182:
1, 1.539324760229491: 1, 1.5391251859745372: 1, 1.536898578942029: 1, 1.5354
98210173569: 1, 1.5354451421064488: 1, 1.5338983540081723: 1, 1.5337311476704
993: 1, 1.5336764210507807: 1, 1.5333820132475153: 1, 1.5332050182120676: 1,
1.532570545801181: 1, 1.5325677397054382: 1, 1.5325414029455606: 1, 1.532032
5495326959: 1, 1.5290958312398253: 1, 1.5262340794119185: 1, 1.52621719197092
7: 1, 1.5255110869701607: 1, 1.5252055097778965: 1, 1.5194472497848: 1, 1.518
6479211015587: 1, 1.5183290737894064: 1, 1.5179595883294281: 1, 1.51725920354
86812: 1, 1.5144016032998466: 1, 1.5142636566926697: 1, 1.5132884482992612:
1, 1.5101836870677077: 1, 1.5101445592481773: 1, 1.5094851510219047: 1, 1.50
92994125948866: 1, 1.5051348995932539: 1, 1.501925511567903: 1, 1.49514806578
34646: 1, 1.4936666183775718: 1, 1.4930994476367263: 1, 1.492033196242118: 1,
1.483959353178052: 1, 1.4831548706485154: 1, 1.4830675907092283: 1, 1.4812460
978692028: 1, 1.479751610278316: 1, 1.4790615018530213: 1, 1.478994419942351:
1, 1.4777053552347652: 1, 1.4772866130966313: 1, 1.4744893286703655: 1, 1.473
3891860238426: 1, 1.4700159224244354: 1, 1.4693098265462623: 1, 1.46879911406
2497: 1, 1.4685088110440043: 1, 1.4650088491165878: 1, 1.4639796646702774: 1,
1.4632404316201337: 1, 1.4613993492440498: 1, 1.4607014969211813: 1, 1.457672
6041106605: 1, 1.45720859280953: 1, 1.4567384379464394: 1, 1.455970803329804
4: 1, 1.4524143738397275: 1, 1.4441726362918956: 1, 1.4410808321474662: 1, 1.
4408381287197563: 1, 1.4394062255409084: 1, 1.4360424999274075: 1, 1.43553620
38322226: 1, 1.4312526135058359: 1, 1.4309109076146775: 1, 1.430194262840330
1: 1, 1.4276102061024094: 1, 1.4257613937954718: 1, 1.4250512935132662: 1, 1.
4233213358233507: 1, 1.4207406004191436: 1, 1.4200859910173922: 1, 1.41956670
46211395: 1, 1.41908729513401: 1, 1.4141426936947716: 1, 1.408707553415686:
1, 1.4059775677541448: 1, 1.403966601663742: 1, 1.403074014879048: 1, 1.4022
912611455145: 1, 1.401507419188379: 1, 1.401418958892676: 1, 1.40058535138009
79: 1, 1.3958870617406083: 1, 1.3931851850607764: 1, 1.3904203266718276: 1,
1.3866798939387597: 1, 1.380086688376824: 1, 1.3728179238788474: 1, 1.360279
0071658943: 1, 1.3327942657572887: 1, 1.3286109299121251: 1, 1.32474634579275
66: 1, 1.323577812447117: 1, 1.312385767938065: 1, 1.312023685075471: 1, 1.30
68525050739952: 1, 1.3015195377399449: 1, 1.279625028642576: 1, 1.25073793489
83294: 1, 1.2470911585497801: 1, 1.214336416730665: 1, 1.2135849428581753: 1,
1.2129808601389094: 1, 1.20153378941137: 1, 0.9723268204134087: 1})

In [47]:
```
# Train a Logistic regression+Calibration model using text features whicha re
 on-hot encoded
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/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: 1.0692508210702205
For values of alpha =  0.0001 The log loss is: 1.036023162657831
For values of alpha =  0.001 The log loss is: 1.1808892242686757
For values of alpha =  0.01 The log loss is: 1.5027570652747648
For values of alpha =  0.1 The log loss is: 1.8370265451543233
For values of alpha =  1 The log loss is: 1.9044055619061886
```




```
For values of best alpha =  0.0001 The train log loss is: 0.7076491678021458
For values of best alpha =  0.0001 The cross validation log loss is: 1.036023
162657831
For values of best alpha =  0.0001 The test log loss is: 1.111851002846273
```

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

Ans. Yes, it seems like!

In [48]:
```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_features=5
000)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

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

In [49]:
```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train
 data")
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in
train data")
```

```
91.94 % of word of test data appeared in train data
90.08 % of word of Cross Validation appeared in train data
```

Feature Engineering

In [50]:
```
# Collecting all the genes and variations data into a single list
gene_variation = []

for gene in result['Gene'].values:
    gene_variation.append(gene)

for variation in result['Variation'].values:
    gene_variation.append(variation)
```

In [51]:
```
tfidfVectorizer = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_features=5
000)
text2 = tfidfVectorizer.fit_transform(gene_variation)
gene_variation_features = tfidfVectorizer.get_feature_names()

train_text = tfidfVectorizer.transform(train_df['TEXT'])
test_text = tfidfVectorizer.transform(test_df['TEXT'])
cv_text = tfidfVectorizer.transform(cv_df['TEXT'])
```

4. Machine Learning Models

In [52]:
```
#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 [53]:
```
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 [54]:
```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_feature
s=5000)

    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 [55]:
```
# 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 [56]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_onehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 7192)
(number of data points * number of features) in test data =  (665, 7192)
(number of data points * number of features) in cross validation data = (532, 7192)
```

In [57]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_responseCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_responseCoding.shape)
```

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

4.1. Base Line Model

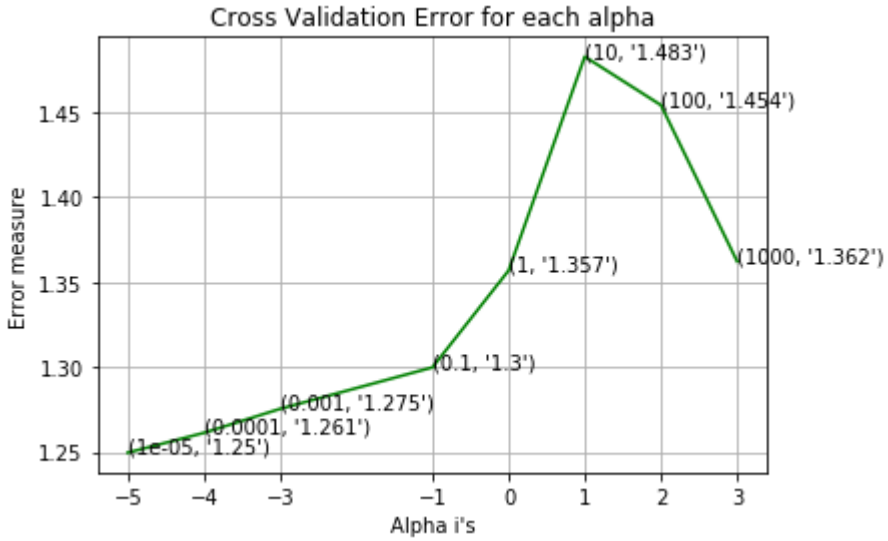
4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [66]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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


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

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

```
plt.grid()
plt.xticks(np.log10(alpha))
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


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


predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss 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 : 1.2496455459864657
for alpha = 0.0001
Log Loss : 1.2612996728968546
for alpha = 0.001
Log Loss : 1.2754214616966109
for alpha = 0.1
Log Loss : 1.2998371064157221
for alpha = 1
Log Loss : 1.3568881665129566
for alpha = 10
Log Loss : 1.4828352809927494
for alpha = 100
Log Loss : 1.4542631685566174
for alpha = 1000
Log Loss : 1.362282414959461
```



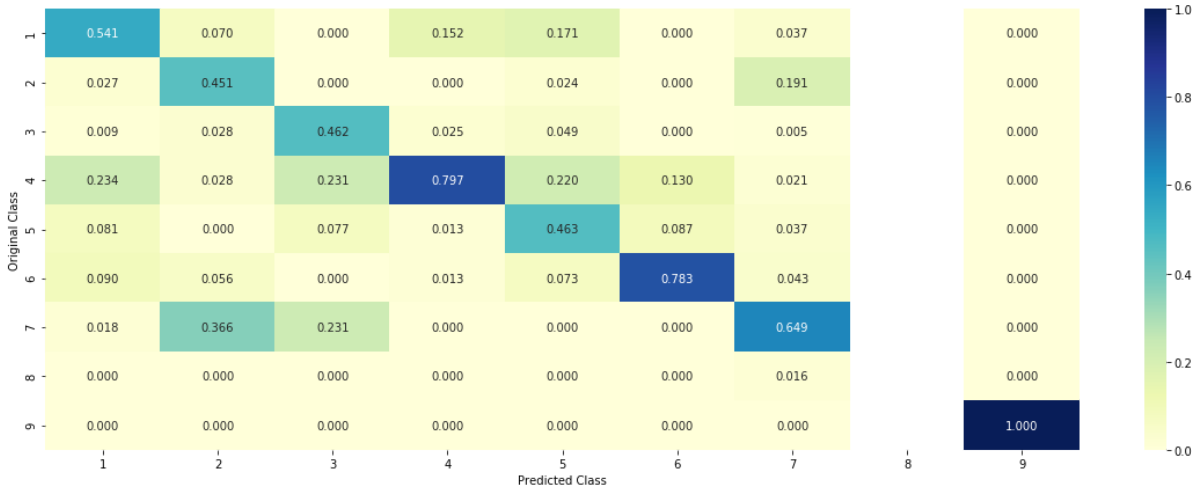

```
For values of best alpha =  1e-05 The train log loss is: 0.8322754981534775
For values of best alpha =  1e-05 The cross validation log loss is: 1.2496455
459864657
For values of best alpha =  1e-05 The test log loss is: 1.2555540268889347
```

4.1.1.2. Testing the model with best hyper paramters

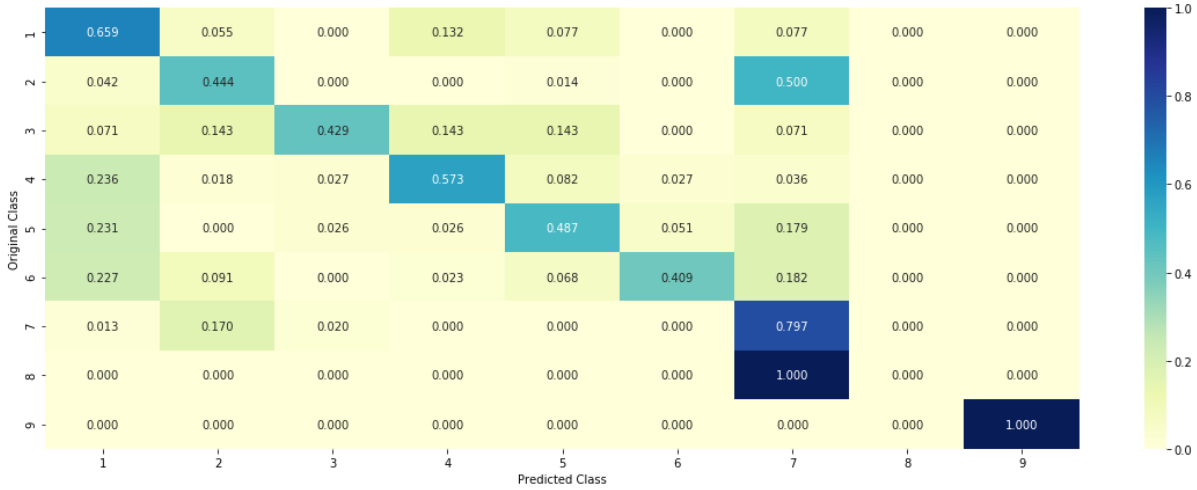
In [67]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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

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

```
Log Loss : 1.2496455459864657
Number of missclassified point : 0.38721804511278196
-------------------- Confusion matrix --------------------
```


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




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



4.1.1.3. Feature Importance, Correctly classified point

In [68]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0829 0.5905 0.0118 0.1042 0.0484 0.0483 0.
1014 0.0065 0.0061]]
Actual Class : 2
--------------------------------------------------
10 Text feature [treatment] present in test data point [True]
14 Text feature [patients] present in test data point [True]
15 Text feature [clinical] present in test data point [True]
16 Text feature [therapy] present in test data point [True]
17 Text feature [response] present in test data point [True]
18 Text feature [molecular] present in test data point [True]
19 Text feature [time] present in test data point [True]
24 Text feature [study] present in test data point [True]
25 Text feature [recently] present in test data point [True]
26 Text feature [confirmed] present in test data point [True]
27 Text feature [including] present in test data point [True]
28 Text feature [therapeutic] present in test data point [True]
29 Text feature [achieved] present in test data point [True]
30 Text feature [months] present in test data point [True]
31 Text feature [first] present in test data point [True]
32 Text feature [however] present in test data point [True]
33 Text feature [inhibitor] present in test data point [True]
34 Text feature [11] present in test data point [True]
36 Text feature [advanced] present in test data point [True]
37 Text feature [also] present in test data point [True]
38 Text feature [another] present in test data point [True]
39 Text feature [13] present in test data point [True]
40 Text feature [primary] present in test data point [True]
41 Text feature [kinase] present in test data point [True]
42 Text feature [may] present in test data point [True]
43 Text feature [mutation] present in test data point [True]
44 Text feature [10] present in test data point [True]
45 Text feature [treated] present in test data point [True]
46 Text feature [15] present in test data point [True]
47 Text feature [respectively] present in test data point [True]
48 Text feature [using] present in test data point [True]
50 Text feature [18] present in test data point [True]
51 Text feature [different] present in test data point [True]
52 Text feature [initial] present in test data point [True]
53 Text feature [common] present in test data point [True]
54 Text feature [small] present in test data point [True]
55 Text feature [12] present in test data point [True]
56 Text feature [mutations] present in test data point [True]
57 Text feature [higher] present in test data point [True]
58 Text feature [reported] present in test data point [True]
59 Text feature [observed] present in test data point [True]
60 Text feature [sequencing] present in test data point [True]
61 Text feature [patient] present in test data point [True]
62 Text feature [performed] present in test data point [True]
63 Text feature [cases] present in test data point [True]
64 Text feature [still] present in test data point [True]
65 Text feature [harbor] present in test data point [True]
66 Text feature [identified] present in test data point [True]
67 Text feature [analysis] present in test data point [True]
68 Text feature [case] present in test data point [True]
69 Text feature [approved] present in test data point [True]
70 Text feature [gene] present in test data point [True]
```

```
71 Text feature [line] present in test data point [True]
72 Text feature [one] present in test data point [True]
73 Text feature [second] present in test data point [True]
74 Text feature [longer] present in test data point [True]
75 Text feature [detection] present in test data point [True]
77 Text feature [studies] present in test data point [True]
78 Text feature [report] present in test data point [True]
79 Text feature [similar] present in test data point [True]
80 Text feature [overall] present in test data point [True]
81 Text feature [17] present in test data point [True]
82 Text feature [disease] present in test data point [True]
83 Text feature [imatinib] present in test data point [True]
84 Text feature [due] present in test data point [True]
86 Text feature [table] present in test data point [True]
88 Text feature [found] present in test data point [True]
89 Text feature [novel] present in test data point [True]
90 Text feature [detected] present in test data point [True]
91 Text feature [demonstrated] present in test data point [True]
92 Text feature [two] present in test data point [True]
93 Text feature [harboring] present in test data point [True]
94 Text feature [phase] present in test data point [True]
95 Text feature [suggests] present in test data point [True]
97 Text feature [number] present in test data point [True]
98 Text feature [best] present in test data point [True]
99 Text feature [shown] present in test data point [True]
Out of the top  100  features  77 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [69]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0823 0.4222 0.0118 0.1036 0.0481 0.048  0.
2715 0.0065 0.006 ]]
Actual Class : 2
--------------------------------------------------
10 Text feature [treatment] present in test data point [True]
14 Text feature [patients] present in test data point [True]
15 Text feature [clinical] present in test data point [True]
16 Text feature [therapy] present in test data point [True]
17 Text feature [response] present in test data point [True]
18 Text feature [molecular] present in test data point [True]
19 Text feature [time] present in test data point [True]
24 Text feature [study] present in test data point [True]
25 Text feature [recently] present in test data point [True]
26 Text feature [confirmed] present in test data point [True]
27 Text feature [including] present in test data point [True]
28 Text feature [therapeutic] present in test data point [True]
29 Text feature [achieved] present in test data point [True]
30 Text feature [months] present in test data point [True]
31 Text feature [first] present in test data point [True]
32 Text feature [however] present in test data point [True]
34 Text feature [11] present in test data point [True]
35 Text feature [progression] present in test data point [True]
36 Text feature [advanced] present in test data point [True]
37 Text feature [also] present in test data point [True]
39 Text feature [13] present in test data point [True]
41 Text feature [kinase] present in test data point [True]
42 Text feature [may] present in test data point [True]
43 Text feature [mutation] present in test data point [True]
44 Text feature [10] present in test data point [True]
45 Text feature [treated] present in test data point [True]
46 Text feature [15] present in test data point [True]
47 Text feature [respectively] present in test data point [True]
48 Text feature [using] present in test data point [True]
49 Text feature [median] present in test data point [True]
50 Text feature [18] present in test data point [True]
51 Text feature [different] present in test data point [True]
52 Text feature [initial] present in test data point [True]
53 Text feature [common] present in test data point [True]
54 Text feature [small] present in test data point [True]
55 Text feature [12] present in test data point [True]
56 Text feature [mutations] present in test data point [True]
57 Text feature [higher] present in test data point [True]
58 Text feature [reported] present in test data point [True]
59 Text feature [observed] present in test data point [True]
60 Text feature [sequencing] present in test data point [True]
61 Text feature [patient] present in test data point [True]
63 Text feature [cases] present in test data point [True]
65 Text feature [harbor] present in test data point [True]
66 Text feature [identified] present in test data point [True]
67 Text feature [analysis] present in test data point [True]
69 Text feature [approved] present in test data point [True]
71 Text feature [line] present in test data point [True]
72 Text feature [one] present in test data point [True]
73 Text feature [second] present in test data point [True]
75 Text feature [detection] present in test data point [True]
77 Text feature [studies] present in test data point [True]
```

```
79 Text feature [similar] present in test data point [True]
81 Text feature [17] present in test data point [True]
82 Text feature [disease] present in test data point [True]
84 Text feature [due] present in test data point [True]
86 Text feature [table] present in test data point [True]
88 Text feature [found] present in test data point [True]
89 Text feature [novel] present in test data point [True]
90 Text feature [detected] present in test data point [True]
91 Text feature [demonstrated] present in test data point [True]
92 Text feature [two] present in test data point [True]
93 Text feature [harboring] present in test data point [True]
97 Text feature [number] present in test data point [True]
99 Text feature [shown] present in test data point [True]
Out of the top  100  features  65 are present in query point
```

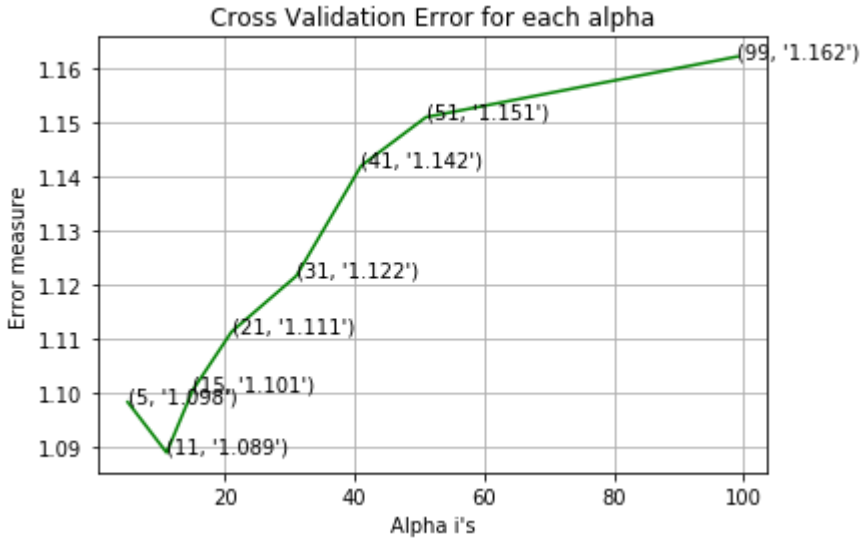
4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [70]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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


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


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

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

```
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


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

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss 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 : 1.0983838320439394
for alpha = 11
Log Loss : 1.089033101229865
for alpha = 15
Log Loss : 1.1005412406892965
for alpha = 21
Log Loss : 1.1113571618668758
for alpha = 31
Log Loss : 1.1216954074262648
for alpha = 41
Log Loss : 1.1420559663007332
for alpha = 51
Log Loss : 1.1510534638333667
for alpha = 99
Log Loss : 1.1622421734687933
```



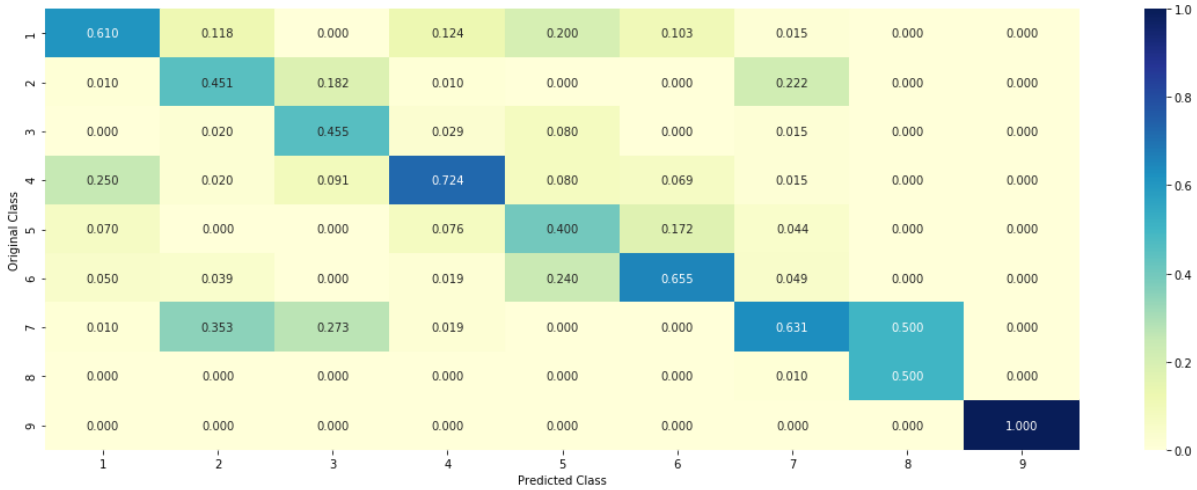

```
For values of best alpha =  11 The train log loss is: 0.6530475872513292
For values of best alpha =  11 The cross validation log loss is: 1.0890331012
29865
For values of best alpha =  11 The test log loss is: 1.0369567667460287
```

4.2.2. Testing the model with best hyper paramters

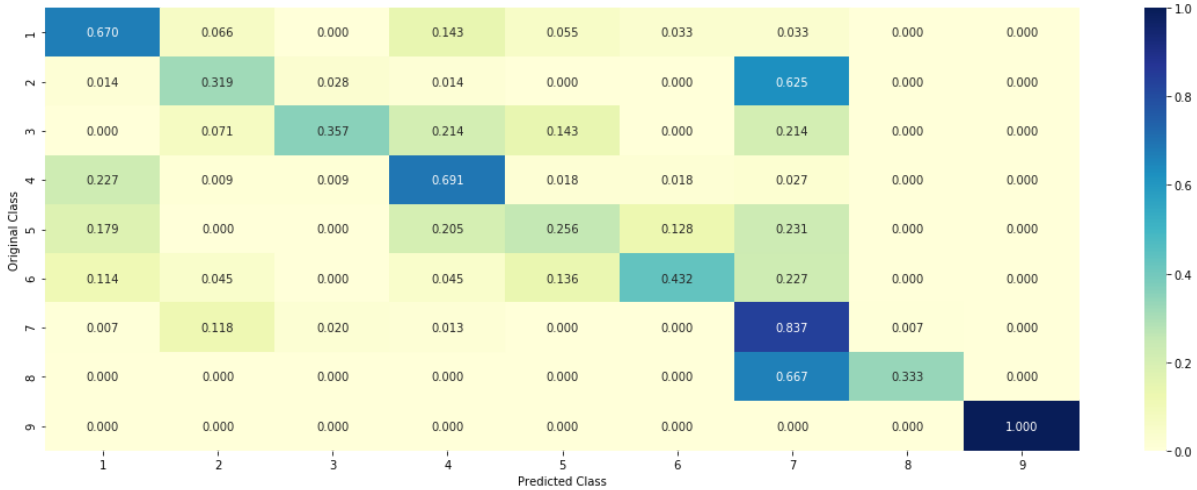
In [71]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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

```
Log loss : 1.089033101229865
Number of mis-classified points : 0.3815789473684211
-------------------- Confusion matrix --------------------
```


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




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



4.2.3.Sample Query point -1

In [72]:
```
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 : 7
Actual Class : 2
The  11  nearest neighbours of the test points belongs to classes [2 2 2 2 2
2 2 2 2 2 2]
Fequency of nearest points : Counter({2: 11})
```

4.2.4. Sample Query Point-2

In [73]:
```
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 11 and the nearest neighbours of the test points belon
gs to classes [2 2 7 7 2 2 7 6 7 8 2]
Fequency of nearest points : Counter({2: 5, 7: 4, 6: 1, 8: 1})
```

4.3. Logistic Regression

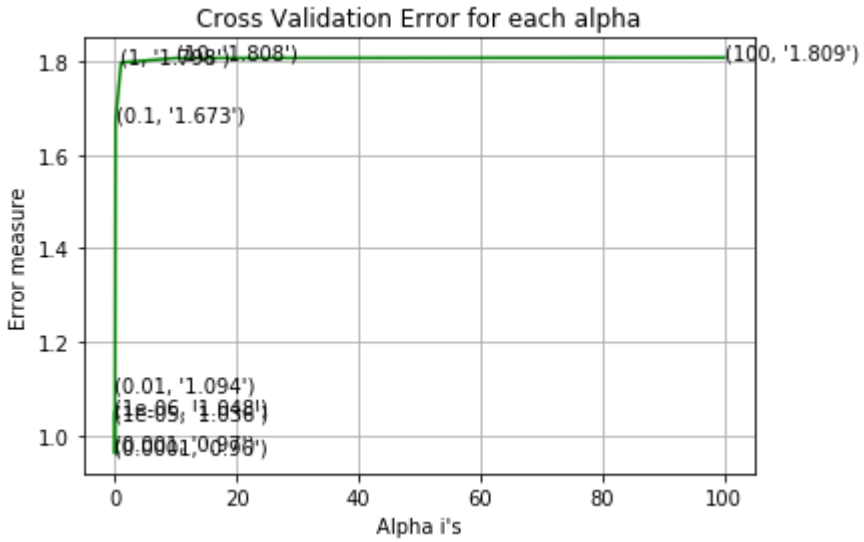
4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [58]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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

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

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
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-06
Log Loss : 1.048211064931988
for alpha = 1e-05
Log Loss : 1.0356783957112583
for alpha = 0.0001
Log Loss : 0.9603738637348174
for alpha = 0.001
Log Loss : 0.9695332170062932
for alpha = 0.01
Log Loss : 1.0942043697287114
for alpha = 0.1
Log Loss : 1.6728190300480688
for alpha = 1
Log Loss : 1.7981229152116998
for alpha = 10
Log Loss : 1.8077071232453779
for alpha = 100
Log Loss : 1.808702711080987
```




```
For values of best alpha =  0.0001 The train log loss is: 0.4436165864672593
For values of best alpha =  0.0001 The cross validation log loss is: 0.960373
8637348174
For values of best alpha =  0.0001 The test log loss is: 1.0245635011510865
```

4.3.1.2. Testing the model with best hyper paramters

In [59]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

```
Log loss : 0.9603738637348174
Number of mis-classified points : 0.30451127819548873
-------------------- Confusion matrix --------------------
```


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


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

4.3.1.3. Feature Importance

In [60]:
```
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], ye
s_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query poin
t")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0]," cl
ass:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or
Not']))
```

4.3.1.3.1. Correctly Classified point

In [61]:
```
# 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: [[0.0482 0.3028 0.0162 0.1923 0.0486 0.0092 0.
3791 0.0022 0.0014]]
Actual Class : 7
--------------------------------------------------
24 Text feature [rarely] present in test data point [True]
75 Text feature [stronger] present in test data point [True]
82 Text feature [phospho] present in test data point [True]
90 Text feature [intrinsic] present in test data point [True]
93 Text feature [downstream] present in test data point [True]
106 Text feature [thyroid] present in test data point [True]
110 Text feature [codon] present in test data point [True]
113 Text feature [constitutive] present in test data point [True]
143 Text feature [activation] present in test data point [True]
149 Text feature [tk] present in test data point [True]
169 Text feature [transformed] present in test data point [True]
197 Text feature [extracellular] present in test data point [True]
207 Text feature [serine] present in test data point [True]
210 Text feature [sos] present in test data point [True]
234 Text feature [3e] present in test data point [True]
235 Text feature [immune] present in test data point [True]
237 Text feature [erk] present in test data point [True]
242 Text feature [cysteine] present in test data point [True]
245 Text feature [resected] present in test data point [True]
268 Text feature [mixed] present in test data point [True]
273 Text feature [incorporation] present in test data point [True]
287 Text feature [leading] present in test data point [True]
307 Text feature [constitutively] present in test data point [True]
310 Text feature [account] present in test data point [True]
326 Text feature [oncogenes] present in test data point [True]
329 Text feature [neoplastic] present in test data point [True]
343 Text feature [oncogene] present in test data point [True]
361 Text feature [respect] present in test data point [True]
362 Text feature [clone] present in test data point [True]
372 Text feature [metastases] present in test data point [True]
377 Text feature [receptors] present in test data point [True]
393 Text feature [phosphorylation] present in test data point [True]
396 Text feature [3b] present in test data point [True]
417 Text feature [serum] present in test data point [True]
418 Text feature [mechanisms] present in test data point [True]
436 Text feature [signaling] present in test data point [True]
468 Text feature [carcinoma] present in test data point [True]
471 Text feature [month] present in test data point [True]
474 Text feature [modest] present in test data point [True]
Out of the top  500  features  39 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [62]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[3.600e-03 7.120e-02 1.400e-03 4.400e-03 1.10
0e-03 1.200e-03 9.166e-01
  5.000e-04 0.000e+00]]
Actual Class : 7
--------------------------------------------------
54 Text feature [technology] present in test data point [True]
70 Text feature [nude] present in test data point [True]
82 Text feature [phospho] present in test data point [True]
90 Text feature [intrinsic] present in test data point [True]
93 Text feature [downstream] present in test data point [True]
104 Text feature [activated] present in test data point [True]
108 Text feature [undergo] present in test data point [True]
113 Text feature [constitutive] present in test data point [True]
143 Text feature [activation] present in test data point [True]
169 Text feature [transformed] present in test data point [True]
171 Text feature [doses] present in test data point [True]
175 Text feature [murine] present in test data point [True]
177 Text feature [neu] present in test data point [True]
183 Text feature [cross] present in test data point [True]
197 Text feature [extracellular] present in test data point [True]
204 Text feature [spontaneous] present in test data point [True]
207 Text feature [serine] present in test data point [True]
234 Text feature [3e] present in test data point [True]
239 Text feature [virus] present in test data point [True]
268 Text feature [mixed] present in test data point [True]
286 Text feature [promoting] present in test data point [True]
287 Text feature [leading] present in test data point [True]
295 Text feature [threonine] present in test data point [True]
308 Text feature [adenocarcinomas] present in test data point [True]
310 Text feature [account] present in test data point [True]
322 Text feature [specimens] present in test data point [True]
323 Text feature [remain] present in test data point [True]
326 Text feature [oncogenes] present in test data point [True]
330 Text feature [therapeutics] present in test data point [True]
343 Text feature [oncogene] present in test data point [True]
347 Text feature [lipid] present in test data point [True]
348 Text feature [ba] present in test data point [True]
356 Text feature [parameters] present in test data point [True]
362 Text feature [clone] present in test data point [True]
372 Text feature [metastases] present in test data point [True]
377 Text feature [receptors] present in test data point [True]
390 Text feature [f3] present in test data point [True]
392 Text feature [achieve] present in test data point [True]
393 Text feature [phosphorylation] present in test data point [True]
396 Text feature [3b] present in test data point [True]
417 Text feature [serum] present in test data point [True]
418 Text feature [mechanisms] present in test data point [True]
421 Text feature [pkb] present in test data point [True]
430 Text feature [week] present in test data point [True]
432 Text feature [institute] present in test data point [True]
436 Text feature [signaling] present in test data point [True]
438 Text feature [term] present in test data point [True]
440 Text feature [around] present in test data point [True]
464 Text feature [overexpression] present in test data point [True]
468 Text feature [carcinoma] present in test data point [True]
474 Text feature [modest] present in test data point [True]
```

```
476 Text feature [isoforms] present in test data point [True]
487 Text feature [long] present in test data point [True]
489 Text feature [equally] present in test data point [True]
490 Text feature [prolonged] present in test data point [True]
495 Text feature [adenocarcinoma] present in test data point [True]
Out of the top  500  features  56 are present in query point
```

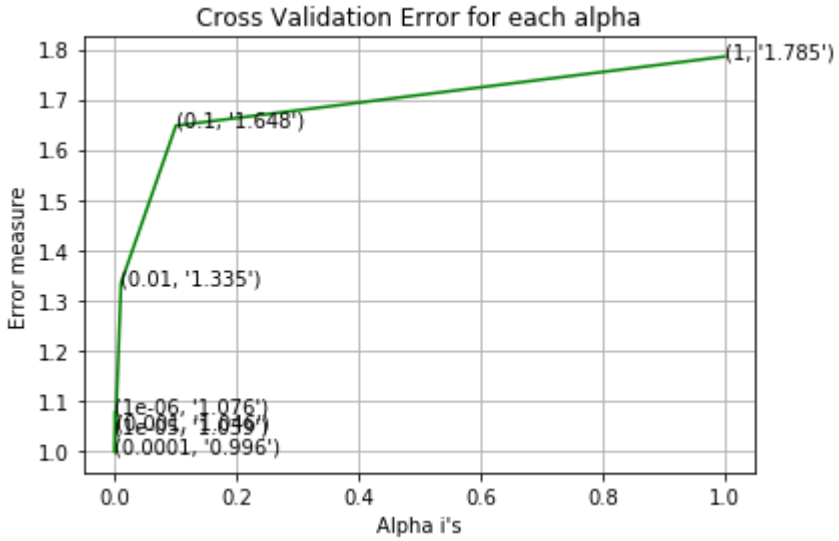
4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

In [63]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



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

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

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

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
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-06
Log Loss : 1.0763004536983516
for alpha = 1e-05
Log Loss : 1.0389537250944532
for alpha = 0.0001
Log Loss : 0.9964450451567329
for alpha = 0.001
Log Loss : 1.045584164163437
for alpha = 0.01
Log Loss : 1.3345531751080701
for alpha = 0.1
Log Loss : 1.6476341824848544
for alpha = 1
Log Loss : 1.7851619229421918
```



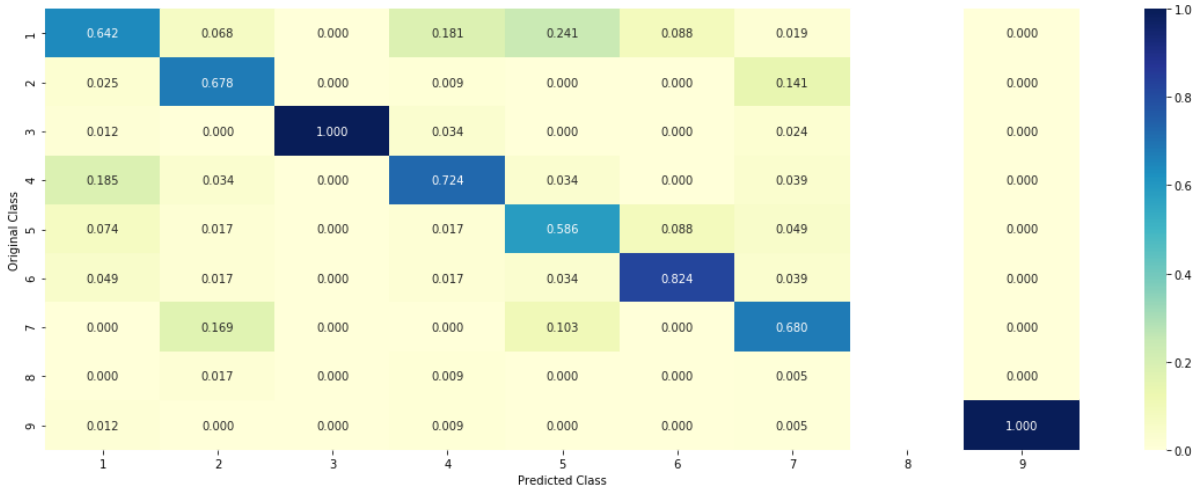

```
For values of best alpha =  0.0001 The train log loss is: 0.4452507519321114
For values of best alpha =  0.0001 The cross validation log loss is: 0.996445
0451567329
For values of best alpha =  0.0001 The test log loss is: 1.0431193065915563
```

4.3.2.2. Testing model with best hyper parameters

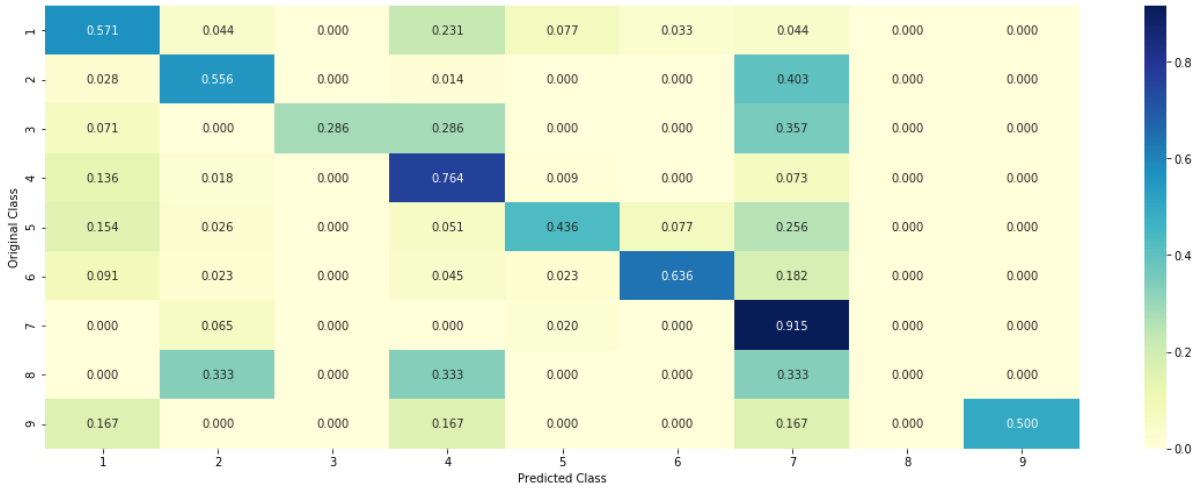
In [64]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

```
Log loss : 0.9964450451567329
Number of mis-classified points : 0.3082706766917293
-------------------- Confusion matrix --------------------
```


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




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



4.3.2.3. Feature Importance, Correctly Classified point

In [65]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_
state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0572 0.293  0.01   0.2228 0.0478 0.0084 0.
3562 0.004  0.0006]]
Actual Class : 7
--------------------------------------------------
43 Text feature [rarely] present in test data point [True]
87 Text feature [downstream] present in test data point [True]
121 Text feature [intrinsic] present in test data point [True]
132 Text feature [thyroid] present in test data point [True]
145 Text feature [stronger] present in test data point [True]
150 Text feature [codon] present in test data point [True]
154 Text feature [constitutive] present in test data point [True]
161 Text feature [phospho] present in test data point [True]
176 Text feature [resected] present in test data point [True]
181 Text feature [immune] present in test data point [True]
193 Text feature [tk] present in test data point [True]
226 Text feature [oncogene] present in test data point [True]
233 Text feature [phosphorylation] present in test data point [True]
251 Text feature [3e] present in test data point [True]
257 Text feature [erk] present in test data point [True]
258 Text feature [cysteine] present in test data point [True]
260 Text feature [activation] present in test data point [True]
271 Text feature [incorporation] present in test data point [True]
299 Text feature [transformed] present in test data point [True]
304 Text feature [constitutively] present in test data point [True]
316 Text feature [oncogenes] present in test data point [True]
323 Text feature [sos] present in test data point [True]
327 Text feature [mixed] present in test data point [True]
337 Text feature [serine] present in test data point [True]
346 Text feature [account] present in test data point [True]
365 Text feature [extracellular] present in test data point [True]
370 Text feature [leading] present in test data point [True]
386 Text feature [neoplastic] present in test data point [True]
421 Text feature [receptors] present in test data point [True]
432 Text feature [respect] present in test data point [True]
441 Text feature [carcinoma] present in test data point [True]
442 Text feature [clone] present in test data point [True]
449 Text feature [metastases] present in test data point [True]
460 Text feature [3d] present in test data point [True]
487 Text feature [signaling] present in test data point [True]
Out of the top  500  features  35 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [66]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index],
no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[4.600e-03 7.560e-02 4.000e-04 6.200e-03 7.00
0e-04 8.000e-04 9.109e-01
  8.000e-04 0.000e+00]]
Actual Class : 7
--------------------------------------------------
73 Text feature [nude] present in test data point [True]
85 Text feature [technology] present in test data point [True]
87 Text feature [downstream] present in test data point [True]
111 Text feature [undergo] present in test data point [True]
121 Text feature [intrinsic] present in test data point [True]
148 Text feature [spontaneous] present in test data point [True]
154 Text feature [constitutive] present in test data point [True]
161 Text feature [phospho] present in test data point [True]
185 Text feature [activated] present in test data point [True]
190 Text feature [promoting] present in test data point [True]
209 Text feature [cross] present in test data point [True]
216 Text feature [doses] present in test data point [True]
226 Text feature [oncogene] present in test data point [True]
233 Text feature [phosphorylation] present in test data point [True]
251 Text feature [3e] present in test data point [True]
260 Text feature [activation] present in test data point [True]
291 Text feature [murine] present in test data point [True]
292 Text feature [threonine] present in test data point [True]
298 Text feature [virus] present in test data point [True]
299 Text feature [transformed] present in test data point [True]
308 Text feature [around] present in test data point [True]
316 Text feature [oncogenes] present in test data point [True]
318 Text feature [neu] present in test data point [True]
327 Text feature [mixed] present in test data point [True]
337 Text feature [serine] present in test data point [True]
339 Text feature [parameters] present in test data point [True]
346 Text feature [account] present in test data point [True]
362 Text feature [achieve] present in test data point [True]
365 Text feature [extracellular] present in test data point [True]
370 Text feature [leading] present in test data point [True]
378 Text feature [adenocarcinomas] present in test data point [True]
381 Text feature [term] present in test data point [True]
392 Text feature [lipid] present in test data point [True]
394 Text feature [ba] present in test data point [True]
415 Text feature [regions] present in test data point [True]
421 Text feature [receptors] present in test data point [True]
428 Text feature [histologic] present in test data point [True]
441 Text feature [carcinoma] present in test data point [True]
442 Text feature [clone] present in test data point [True]
449 Text feature [metastases] present in test data point [True]
453 Text feature [f3] present in test data point [True]
460 Text feature [3d] present in test data point [True]
469 Text feature [long] present in test data point [True]
472 Text feature [therapeutics] present in test data point [True]
475 Text feature [isoforms] present in test data point [True]
480 Text feature [pkb] present in test data point [True]
484 Text feature [invasive] present in test data point [True]
487 Text feature [signaling] present in test data point [True]
Out of the top  500  features  48 are present in query point
```

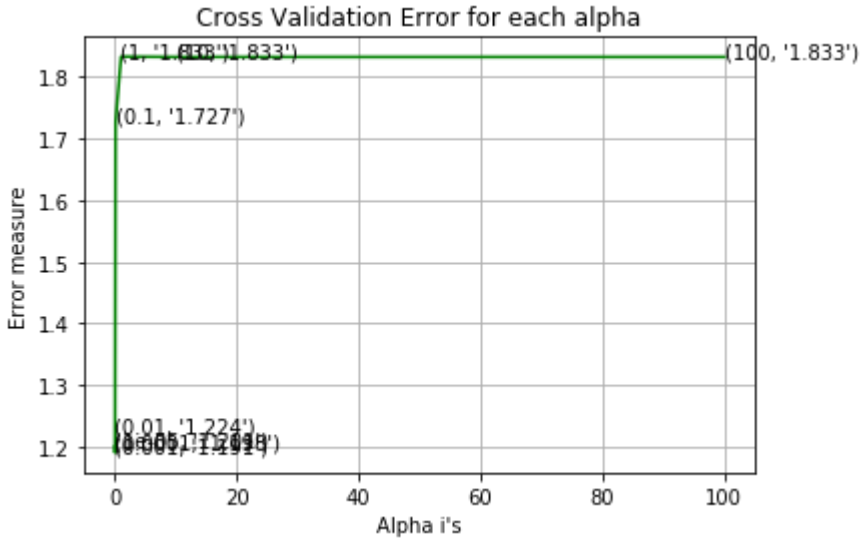
4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [74]:
```
# 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 : 1.2006400057947222
for C = 0.0001
Log Loss : 1.1976265162740343
for C = 0.001
Log Loss : 1.1905127542799265
for C = 0.01
Log Loss : 1.2244121047119916
for C = 0.1
Log Loss : 1.727319143570992
for C = 1
Log Loss : 1.833161892826495
for C = 10
Log Loss : 1.8325855109344091
for C = 100
Log Loss : 1.8325869583881378
```



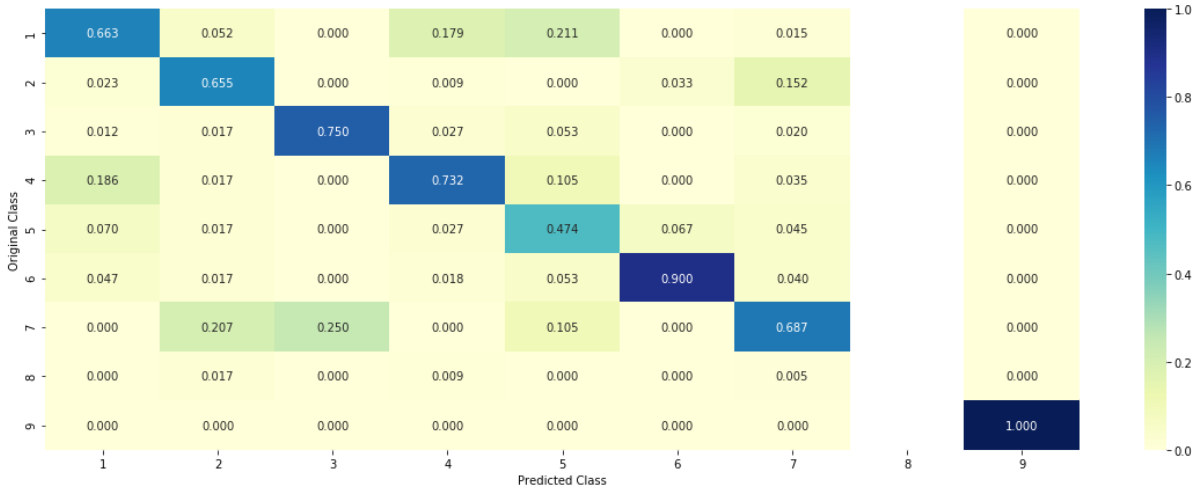

```
For values of best alpha =  0.001 The train log loss is: 0.6107393419667463
For values of best alpha =  0.001 The cross validation log loss is: 1.1905127
542799265
For values of best alpha =  0.001 The test log loss is: 1.0706345555042065
```

4.4.2. Testing model with best hyper parameters

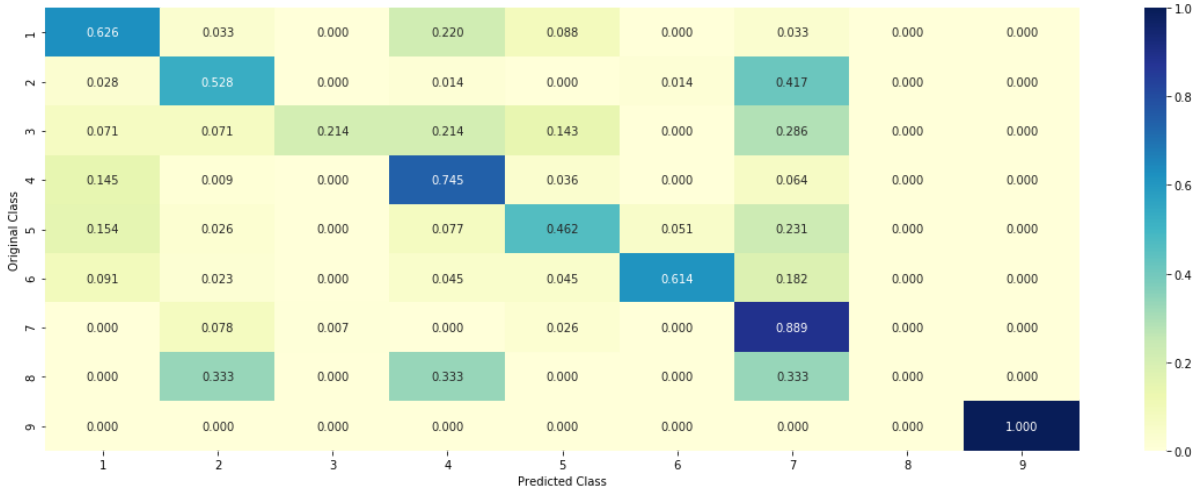
In [67]:
```
# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

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

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


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

```
Log loss : 0.9920336811540992
Number of mis-classified points : 0.3101503759398496
-------------------- 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 [76]:
```
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(-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: [[1.630e-02 8.739e-01 2.600e-03 1.070e-02 3.58
0e-02 1.130e-02 4.770e-02
  1.500e-03 3.000e-04]]
Actual Class : 2
--------------------------------------------------
128 Text feature [v559d] present in test data point [True]
157 Text feature [achieved] present in test data point [True]
250 Text feature [pronounced] present in test data point [True]
258 Text feature [concerning] present in test data point [True]
290 Text feature [existing] present in test data point [True]
393 Text feature [initial] present in test data point [True]
398 Text feature [limitation] present in test data point [True]
401 Text feature [custom] present in test data point [True]
425 Text feature [need] present in test data point [True]
Out of the top  500  features  9 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [77]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_
onehotCoding[test_point_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 : 2
Predicted Class Probabilities: [[2.050e-02 7.531e-01 2.200e-03 7.100e-03 2.97
0e-02 3.900e-03 1.820e-01
  1.200e-03 3.000e-04]]
Actual Class : 2
--------------------------------------------------
67 Text feature [white] present in test data point [True]
157 Text feature [achieved] present in test data point [True]
222 Text feature [011] present in test data point [True]
259 Text feature [head] present in test data point [True]
281 Text feature [nonsmokers] present in test data point [True]
372 Text feature [amplifications] present in test data point [True]
393 Text feature [initial] present in test data point [True]
394 Text feature [dye] present in test data point [True]
398 Text feature [limitation] present in test data point [True]
425 Text feature [need] present in test data point [True]
Out of the top  500  features  10 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [78]:
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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


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

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

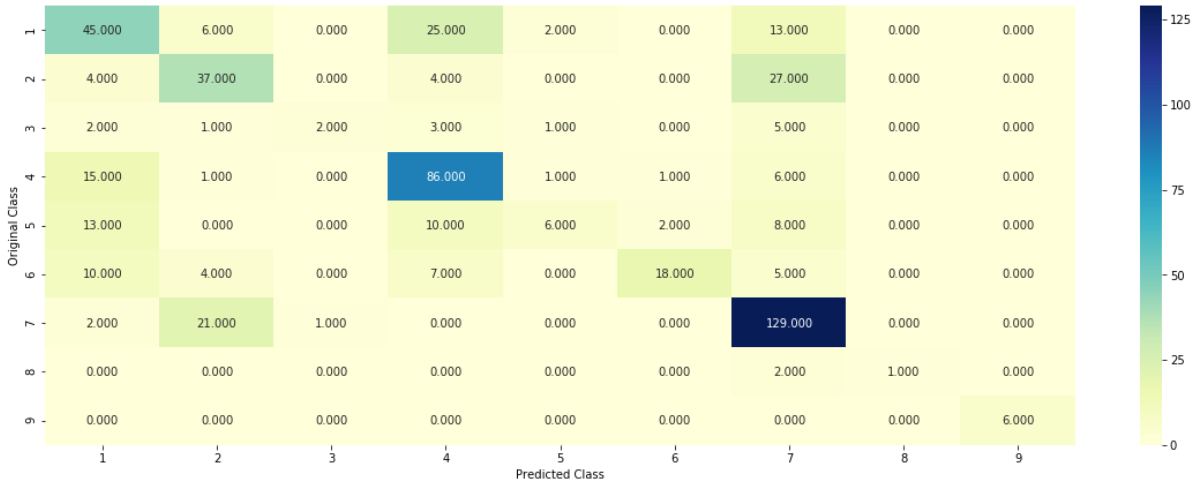
```
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

'''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 : 1.2511330299511971
for n_estimators = 100 and max depth =  10
Log Loss : 1.174955383374727
for n_estimators = 200 and max depth =  5
Log Loss : 1.2316622776586228
for n_estimators = 200 and max depth =  10
Log Loss : 1.1675602038501256
for n_estimators = 500 and max depth =  5
Log Loss : 1.2210081355698041
for n_estimators = 500 and max depth =  10
Log Loss : 1.1654310587887262
for n_estimators = 1000 and max depth =  5
Log Loss : 1.217685226870146
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1634485830763925
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2161640871063408
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1617506055793394
For values of best estimator =  2000 The train log loss is: 0.631557326931339
1
For values of best estimator =  2000 The cross validation log loss is: 1.1617
506055793392
For values of best estimator =  2000 The test log loss is: 1.1474745966884863
```

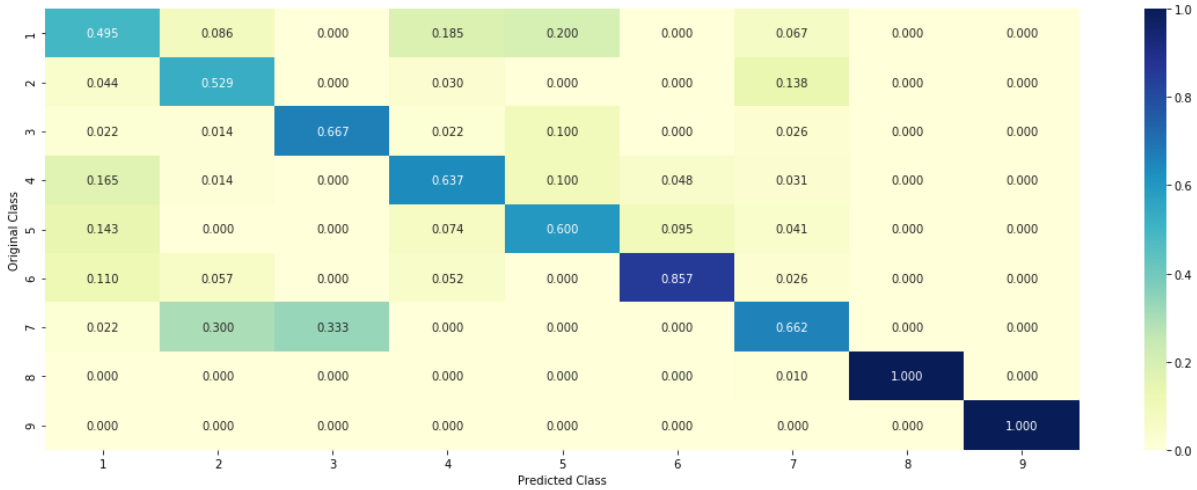
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [79]:
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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

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

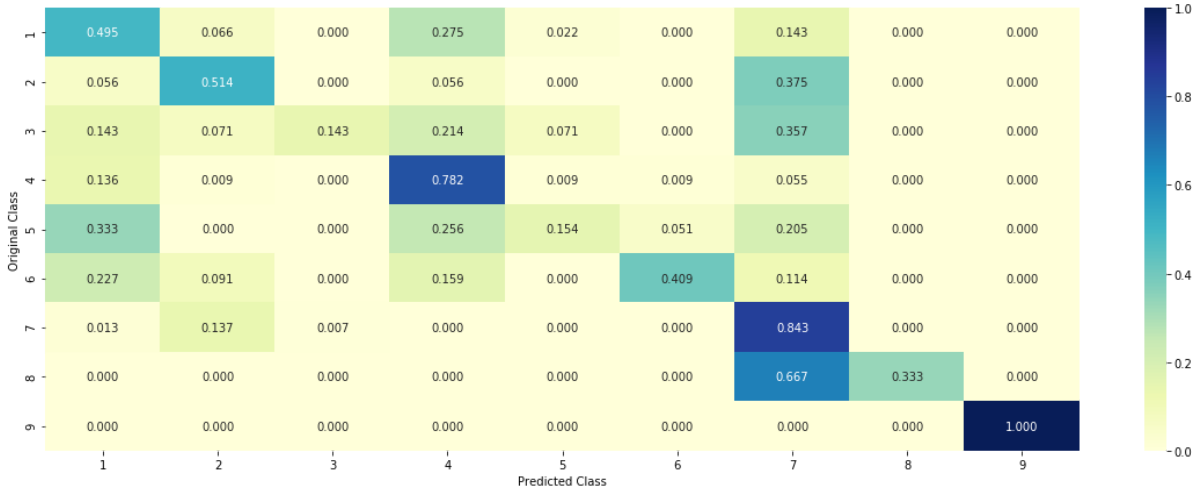
```
Log loss : 1.1617506055793392
Number of mis-classified points : 0.37969924812030076
-------------------- Confusion matrix --------------------
```




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




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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [80]:
```
# 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 : 2
Predicted Class Probabilities: [[0.0237 0.7333 0.0122 0.0178 0.0345 0.0315 0.
1407 0.0036 0.0027]]
Actual Class : 2
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [activation] present in test data point [True]
6 Text feature [tyrosine] present in test data point [True]
7 Text feature [inhibitors] present in test data point [True]
8 Text feature [inhibitor] present in test data point [True]
9 Text feature [phosphorylation] present in test data point [True]
10 Text feature [treatment] present in test data point [True]
11 Text feature [activated] present in test data point [True]
12 Text feature [oncogenic] present in test data point [True]
16 Text feature [function] present in test data point [True]
20 Text feature [therapy] present in test data point [True]
22 Text feature [inhibition] present in test data point [True]
25 Text feature [trials] present in test data point [True]
26 Text feature [cells] present in test data point [True]
28 Text feature [growth] present in test data point [True]
34 Text feature [treated] present in test data point [True]
35 Text feature [kinases] present in test data point [True]
36 Text feature [receptor] present in test data point [True]
38 Text feature [drug] present in test data point [True]
43 Text feature [therapeutic] present in test data point [True]
45 Text feature [loss] present in test data point [True]
46 Text feature [inhibited] present in test data point [True]
48 Text feature [variants] present in test data point [True]
52 Text feature [sensitivity] present in test data point [True]
53 Text feature [resistance] present in test data point [True]
57 Text feature [amplification] present in test data point [True]
58 Text feature [expressing] present in test data point [True]
62 Text feature [months] present in test data point [True]
65 Text feature [patients] present in test data point [True]
68 Text feature [efficacy] present in test data point [True]
71 Text feature [protein] present in test data point [True]
72 Text feature [harboring] present in test data point [True]
73 Text feature [ba] present in test data point [True]
75 Text feature [ic50] present in test data point [True]
78 Text feature [atp] present in test data point [True]
79 Text feature [proliferation] present in test data point [True]
80 Text feature [cell] present in test data point [True]
81 Text feature [respond] present in test data point [True]
84 Text feature [clinical] present in test data point [True]
86 Text feature [phosphorylated] present in test data point [True]
87 Text feature [sensitive] present in test data point [True]
91 Text feature [advanced] present in test data point [True]
92 Text feature [proteins] present in test data point [True]
99 Text feature [resistant] present in test data point [True]
Out of the top  100  features  45 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [81]:
```
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: [[0.0517 0.6563 0.0148 0.0372 0.0408 0.0389 0.
1509 0.0049 0.0046]]
Actuall Class : 2
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
6 Text feature [tyrosine] present in test data point [True]
7 Text feature [inhibitors] present in test data point [True]
10 Text feature [treatment] present in test data point [True]
20 Text feature [therapy] present in test data point [True]
24 Text feature [missense] present in test data point [True]
25 Text feature [trials] present in test data point [True]
26 Text feature [cells] present in test data point [True]
28 Text feature [growth] present in test data point [True]
34 Text feature [treated] present in test data point [True]
36 Text feature [receptor] present in test data point [True]
43 Text feature [therapeutic] present in test data point [True]
48 Text feature [variants] present in test data point [True]
52 Text feature [sensitivity] present in test data point [True]
53 Text feature [resistance] present in test data point [True]
55 Text feature [brca1] present in test data point [True]
57 Text feature [amplification] present in test data point [True]
62 Text feature [months] present in test data point [True]
65 Text feature [patients] present in test data point [True]
72 Text feature [harboring] present in test data point [True]
80 Text feature [cell] present in test data point [True]
81 Text feature [respond] present in test data point [True]
84 Text feature [clinical] present in test data point [True]
87 Text feature [sensitive] present in test data point [True]
91 Text feature [advanced] present in test data point [True]
Out of the top  100  features  25 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [82]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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


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

alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
```

```
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (features[i],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 : 2.1856324008803263
for n_estimators = 10 and max depth =  3
Log Loss : 1.675722165496704
for n_estimators = 10 and max depth =  5
Log Loss : 1.4305936721845143
for n_estimators = 10 and max depth =  10
Log Loss : 1.9203339982310377
for n_estimators = 50 and max depth =  2
Log Loss : 1.5584275922286557
for n_estimators = 50 and max depth =  3
Log Loss : 1.3820820173164214
for n_estimators = 50 and max depth =  5
Log Loss : 1.3416122197401608
for n_estimators = 50 and max depth =  10
Log Loss : 1.823859306404858
for n_estimators = 100 and max depth =  2
Log Loss : 1.4573493779161113
for n_estimators = 100 and max depth =  3
Log Loss : 1.385916513896538
for n_estimators = 100 and max depth =  5
Log Loss : 1.288384954248733
for n_estimators = 100 and max depth =  10
Log Loss : 1.7887781637975233
for n_estimators = 200 and max depth =  2
Log Loss : 1.5031617049927815
for n_estimators = 200 and max depth =  3
Log Loss : 1.3896602732775947
for n_estimators = 200 and max depth =  5
Log Loss : 1.3475104141872583
for n_estimators = 200 and max depth =  10
Log Loss : 1.767188199636615
for n_estimators = 500 and max depth =  2
Log Loss : 1.5582331278492856
for n_estimators = 500 and max depth =  3
Log Loss : 1.4392426166955374
for n_estimators = 500 and max depth =  5
Log Loss : 1.417474264108236
for n_estimators = 500 and max depth =  10
Log Loss : 1.8537213373188202
for n_estimators = 1000 and max depth =  2
Log Loss : 1.5534746246274356
for n_estimators = 1000 and max depth =  3
Log Loss : 1.4667867066879505
for n_estimators = 1000 and max depth =  5
Log Loss : 1.4283746756619198
for n_estimators = 1000 and max depth =  10
Log Loss : 1.826132362320969
For values of best alpha =  100 The train log loss is: 0.05542870268266396
For values of best alpha =  100 The cross validation log loss is: 1.288384986
803715
For values of best alpha =  100 The test log loss is: 1.2764222181349028
```

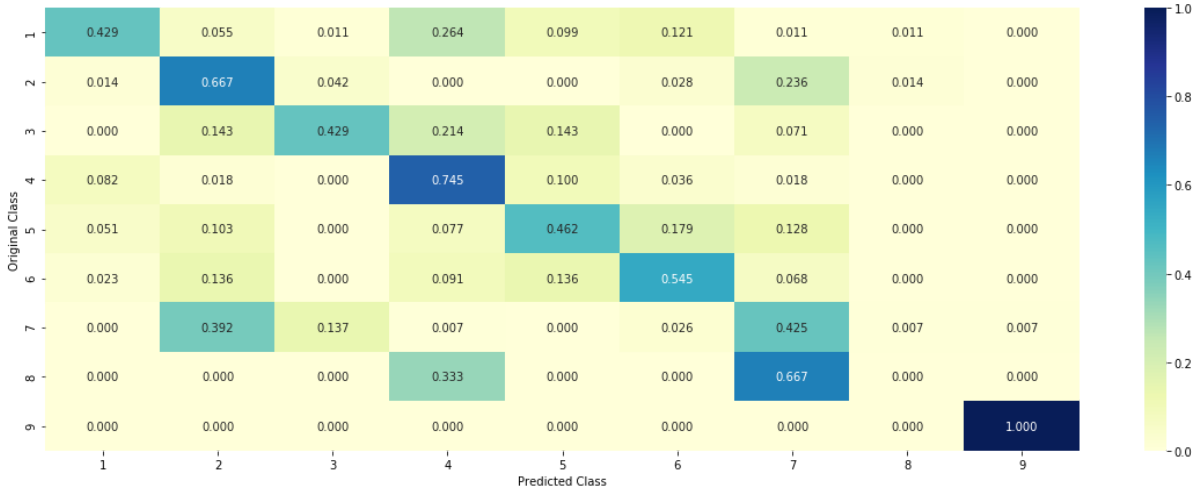
4.5.4. Testing model with best hyper parameters (Response Coding)

In [83]:
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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

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

```
Log loss : 1.2883849542487327
Number of mis-classified points : 0.45864661654135336
-------------------- 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 [84]:
```
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 : 2
Predicted Class Probabilities: [[0.0152 0.842  0.03   0.0193 0.0114 0.0165 0.
0269 0.0221 0.0166]]
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
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [85]:
```
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: [[0.0146 0.4786 0.1278 0.0174 0.0276 0.0347 0.
2621 0.0265 0.0107]]
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
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
```

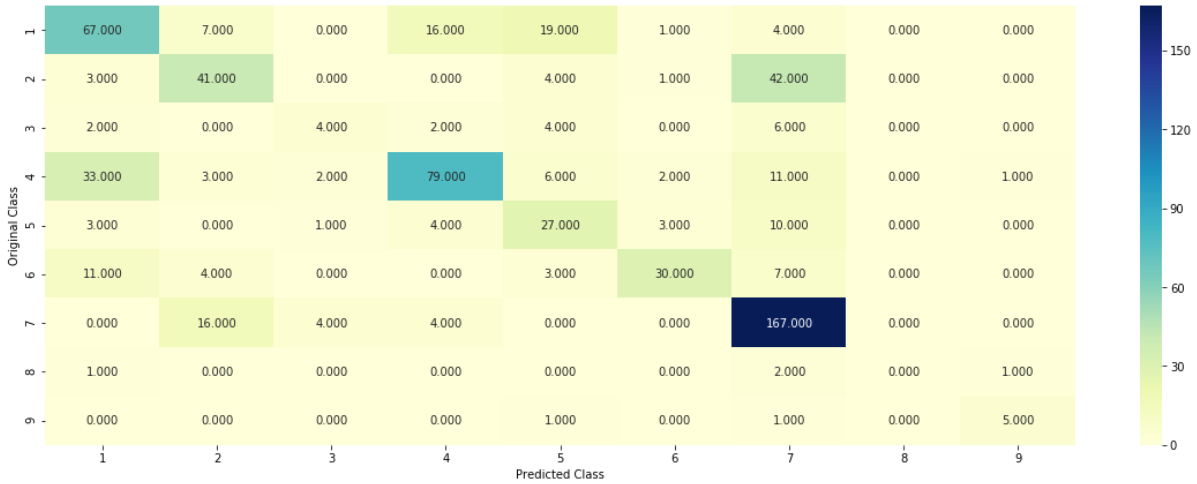
4.7 Stack the models

4.7.1 testing with hyper parameter tuning

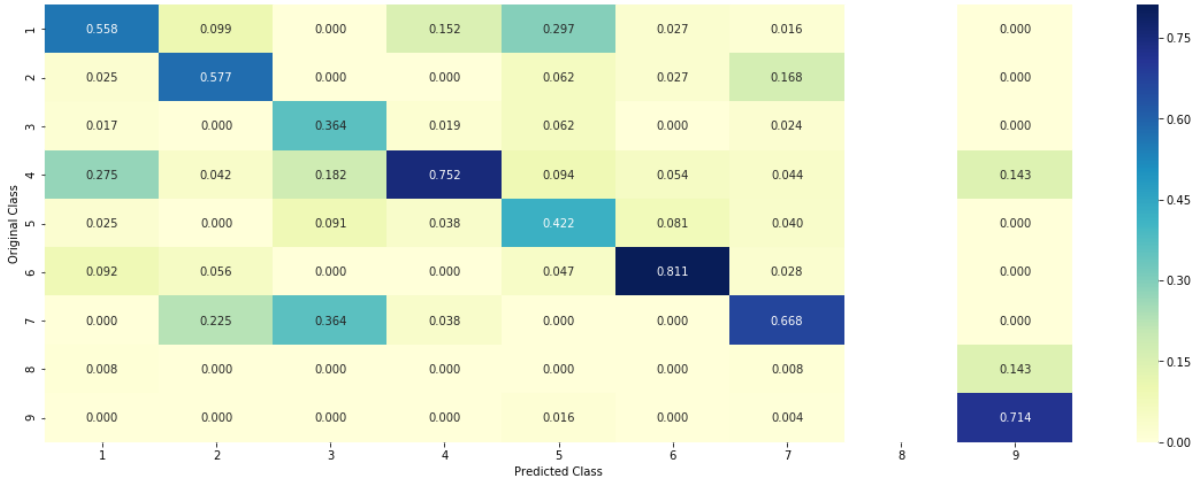
4.7.3 Maximum Voting classifier

In [88]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

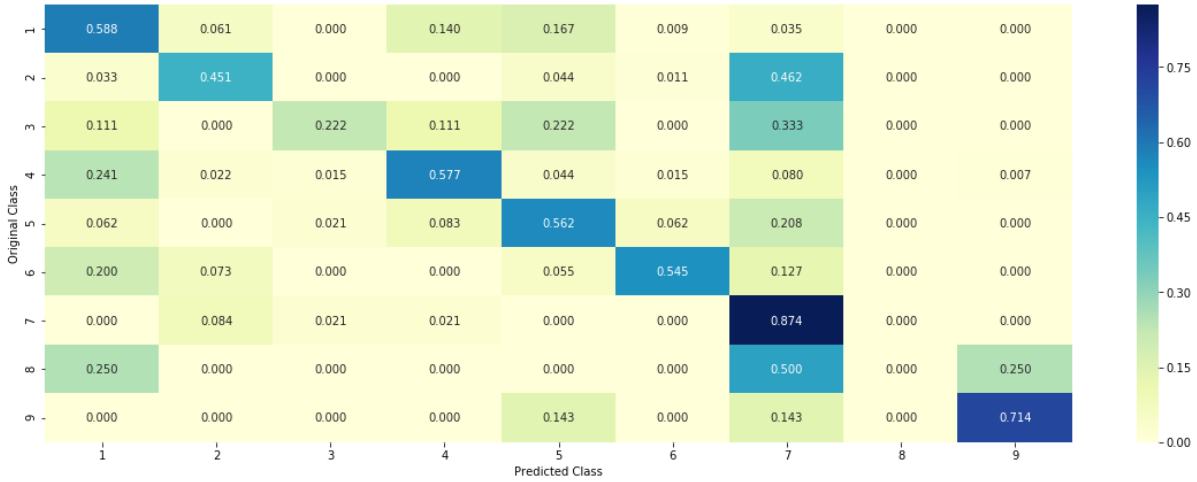
```
Log loss (train) on the VotingClassifier : 0.871591095102783
Log loss (CV) on the VotingClassifier : 1.182464003146664
Log loss (test) on the VotingClassifier : 1.1432909849320254
Number of missclassified point : 0.3684210526315789
-------------------- Confusion matrix --------------------
```




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




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



Conclusion

In [69]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names =["Models","Train","CV","Test","Misclassified(%)"]

x.add_row(["Naive Bayes (One hot coding)",0.83,1.24,1.25,0.38])
x.add_row(["KNN (Response)",0.66,1.08,1.08,0.38])
x.add_row(["LR(Class balanced) one hot coding",0.44,0.99,0.99,0.31])
x.add_row(["LR(Class unbalanced) one hot coding",0.54,1.19,0.99,0.37])
x.add_row(["Lr SVM one hot encoding",0.61,1.19,1.07,0.39])
x.add_row(["Random Forest one hot coding",0.63,1.16,1.14,0.37])
x.add_row(["Random Forest Response coding",0.55,1.28,1.27,0.45])
x.add_row(["Maximum Voting Classifier",0.87,1.18,1.14,0.36])

print(x)
print("\n")
```

```
+-------------------------------------+-------+------+------+----------------
--+
|                Models               | Train |  CV  | Test | Misclassified
(%) |
+-------------------------------------+-------+------+------+----------------
--+
|     Naive Bayes (One hot coding)    |  0.83 | 1.24 | 1.25 |       0.38
|
|            KNN (Response)           |  0.66 | 1.08 | 1.08 |       0.38
|
|  LR(Class balanced) one hot coding  |  0.44 | 0.99 | 0.99 |       0.31
|
| LR(Class unbalanced) one hot coding |  0.54 | 1.19 | 0.99 |       0.37
|
|       Lr SVM one hot encoding       |  0.61 | 1.19 | 1.07 |       0.39
|
|     Random Forest one hot coding    |  0.63 | 1.16 | 1.14 |       0.37
|
|    Random Forest Response coding    |  0.55 | 1.28 | 1.27 |       0.45
|
|      Maximum Voting Classifier      |  0.87 | 1.18 | 1.14 |       0.36
|
+-------------------------------------+-------+------+------+----------------
--+
```