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
```

```
Using TensorFlow backend.
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:
516: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is depr
ecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:
517: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is depr
ecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:
518: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is depr
ecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:
519: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is depr
ecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:
520: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is depr
ecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtypes.py:
525: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is depr
ecated; in a future version of numpy, it will be understood as (type, (1,)) /
'(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
/usr/local/lib/python3.5/dist-packages/tensorboard/compat/tensorflow_stub/dty
pes.py:541: FutureWarning: Passing (type, 1) or '1type' as a synonym of type
is deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/usr/local/lib/python3.5/dist-packages/tensorboard/compat/tensorflow_stub/dty
pes.py:542: FutureWarning: Passing (type, 1) or '1type' as a synonym of type
is deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/usr/local/lib/python3.5/dist-packages/tensorboard/compat/tensorflow_stub/dty
pes.py:543: FutureWarning: Passing (type, 1) or '1type' as a synonym of type
is deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/usr/local/lib/python3.5/dist-packages/tensorboard/compat/tensorflow_stub/dty
pes.py:544: FutureWarning: Passing (type, 1) or '1type' as a synonym of type
is deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/usr/local/lib/python3.5/dist-packages/tensorboard/compat/tensorflow_stub/dty
pes.py:545: FutureWarning: Passing (type, 1) or '1type' as a synonym of type
is deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/usr/local/lib/python3.5/dist-packages/tensorboard/compat/tensorflow_stub/dty
```

```
pes.py:550: FutureWarning: Passing (type, 1) or '1type' as a synonym of type
is deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

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

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

Out[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_text",sep="\|\|",engine="python",names=["ID",
"TEXT"],skiprows=1)
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

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

Out[3]:

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

3.1.3. Preprocessing of text

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


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

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

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

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

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 35.505905999999996 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

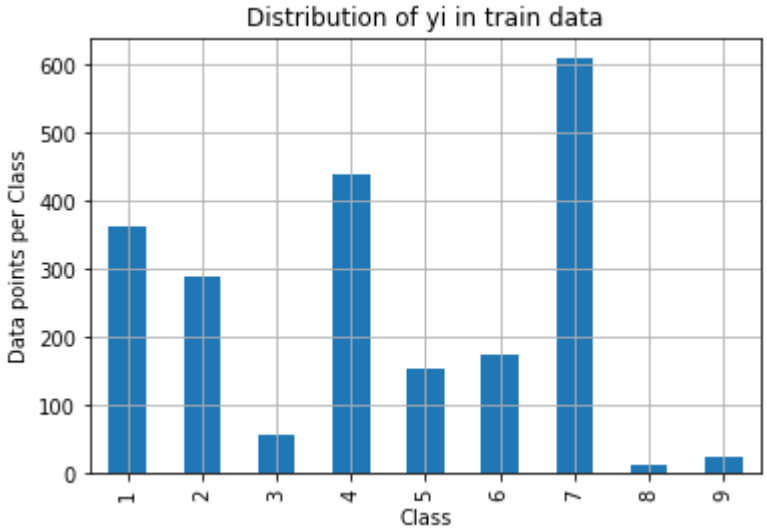
In [11]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
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
```

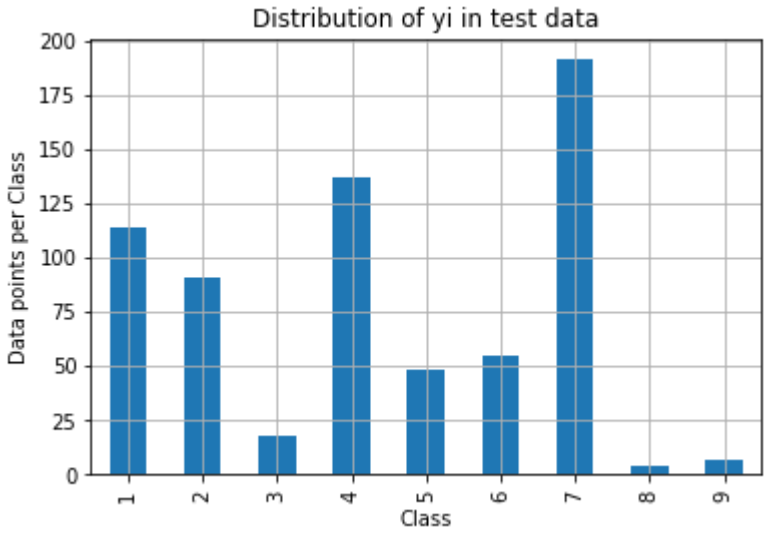
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

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().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

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
```

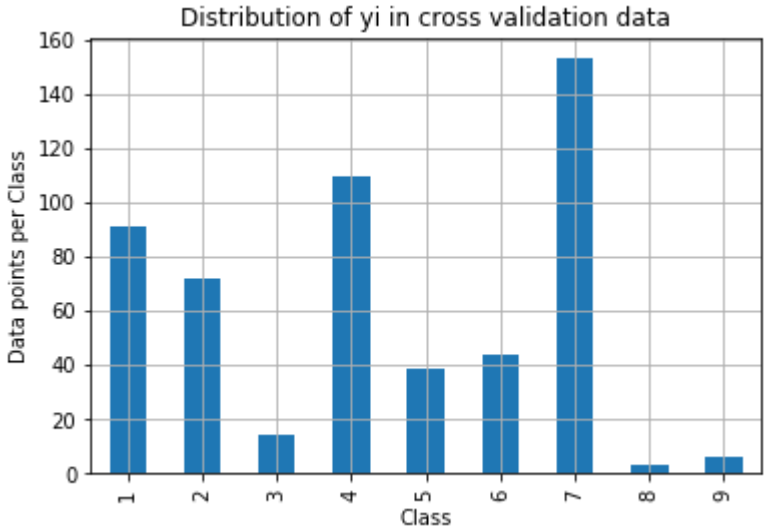
```
g order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.val
ues[i], '(', np.round((cv_class_distribution.values[i]/cv_df.shape[0]*100), 3
), '%)')
```




```
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

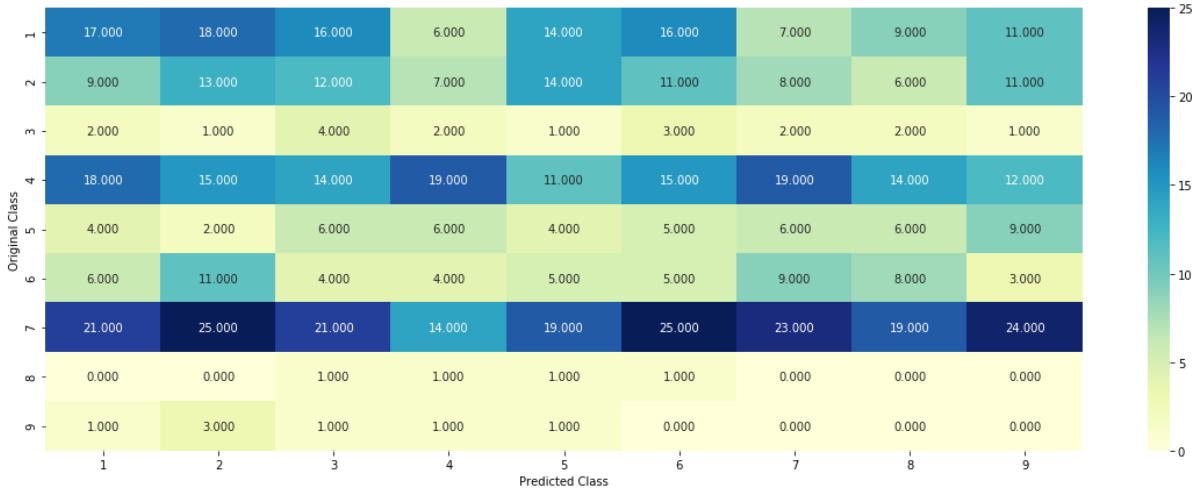
In a 'Random' Model, we generate the NINE class probabilites randomly such that they sum to 1.

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()
```

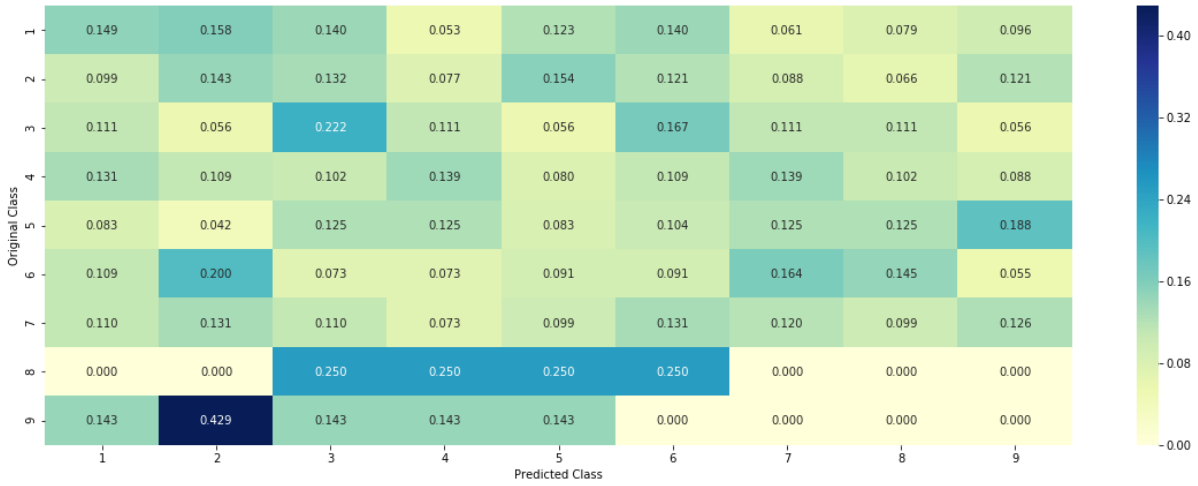
In [14]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by thei
r sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# 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 2.496007849418387
Log loss on Test Data using Random Model 2.438241757580394
-------------------- 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?

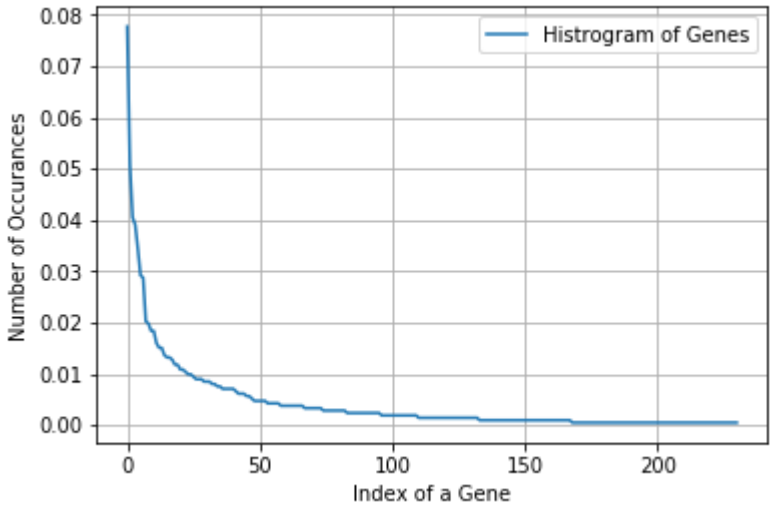
In [16]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 231
BRCA1     165
TP53      108
PTEN       86
EGFR       83
BRCA2      73
BRAF       62
KIT        61
ALK        43
ERBB2      42
PIK3CA     39
Name: Gene, dtype: int64
```

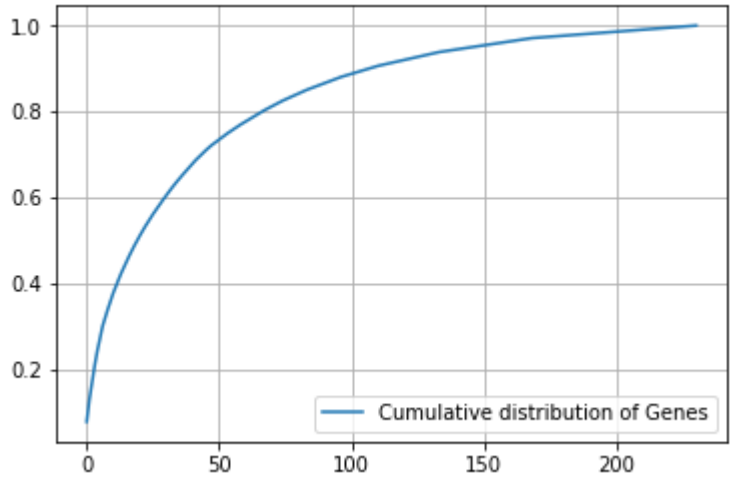
In [17]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes
 in the train data, and they are distibuted as follows",)
```

Ans: There are 231 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]:
```
2780     BRCA2
1410     FGFR3
2165      PTEN
1973    CTNNB1
2944       KDR
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',
 'asxl2',
 'atm',
 'atrx',
 'aurka',
 'aurkb',
 'axl',
 'b2m',
 'bap1',
 'bcl10',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk6',
 'cdkn1a',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cdkn2c',
 'cebpa',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctnnb1',
 'ddr2',
 'dicer1',
 'dnmt3a',
 'dnmt3b',
 'egfr',
 'eif1ax',
```

```
                'elf3',
                'ep300',
                'epas1',
                'epcam',
                'erbb2',
                'erbb3',
                'erbb4',
                'ercc2',
                'ercc3',
                'ercc4',
                'erg',
                'errfi1',
                'esr1',
                'etv1',
                'etv6',
                'ewsr1',
                'ezh2',
                'fanca',
                'fancc',
                'fat1',
                'fbxw7',
                'fgf19',
                'fgf4',
                'fgfr1',
                'fgfr2',
                'fgfr3',
                'fgfr4',
                'flt3',
                'foxa1',
                'foxl2',
                'foxp1',
                'fubp1',
                'gata3',
                'gli1',
                'gnaq',
                'gnas',
                'h3f3a',
                'hla',
                'hnf1a',
                'hras',
                'idh1',
                'idh2',
                'igf1r',
                'ikbke',
                'il7r',
                'jak1',
                'jak2',
                'kdm5c',
                'kdm6a',
                'kdr',
                'keap1',
                'kit',
                'klf4',
                'kmt2a',
                'kmt2b',
                'kmt2c',
                'kmt2d',
```

```
                'knstrn',
                'kras',
                'lats1',
                'lats2',
                'map2k1',
                'map2k2',
                'map2k4',
                'map3k1',
                'mapk1',
                'mdm2',
                'mdm4',
                'med12',
                'mef2b',
                'met',
                'mga',
                'mlh1',
                'mpl',
                'msh2',
                'msh6',
                'mtor',
                'myc',
                'mycn',
                'myd88',
                'myod1',
                'nf1',
                'nf2',
                'nfe2l2',
                'nfkbia',
                'nkx2',
                'notch1',
                'notch2',
                'npm1',
                'nras',
                'nsd1',
                'ntrk1',
                'ntrk2',
                'ntrk3',
                'nup93',
                'pak1',
                'pax8',
                'pbrm1',
                'pdgfra',
                'pdgfrb',
                'pik3ca',
                'pik3cb',
                'pik3cd',
                'pik3r1',
                'pik3r2',
                'pik3r3',
                'pim1',
                'pms1',
                'pms2',
                'pole',
                'ppp2r1a',
                'ppp6c',
                'prdm1',
                'ptch1',
```

```
                'pten',
                'ptpn11',
                'ptprd',
                'ptprt',
                'rac1',
                'rad21',
                'rad50',
                'rad51c',
                'rad54l',
                'raf1',
                'rara',
                'rasa1',
                'rb1',
                'rbm10',
                'ret',
                'rheb',
                'rhoa',
                'rictor',
                'rit1',
                'rnf43',
                'ros1',
                'rras2',
                'runx1',
                'rxra',
                'rybp',
                'sdhc',
                'setd2',
                'sf3b1',
                'shoc2',
                'smad2',
                'smad3',
                'smad4',
                'smarca4',
                'smarcb1',
                'smo',
                'sos1',
                'sox9',
                'spop',
                'src',
                'stat3',
                'stk11',
                'tcf3',
                'tert',
                'tet1',
                'tet2',
                'tgfbr1',
                'tgfbr2',
                'tmprss2',
                'tp53',
                'tp53bp1',
                'tsc1',
                'tsc2',
                'u2af1',
                'vhl',
                'whsc1',
                '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, 230)
```

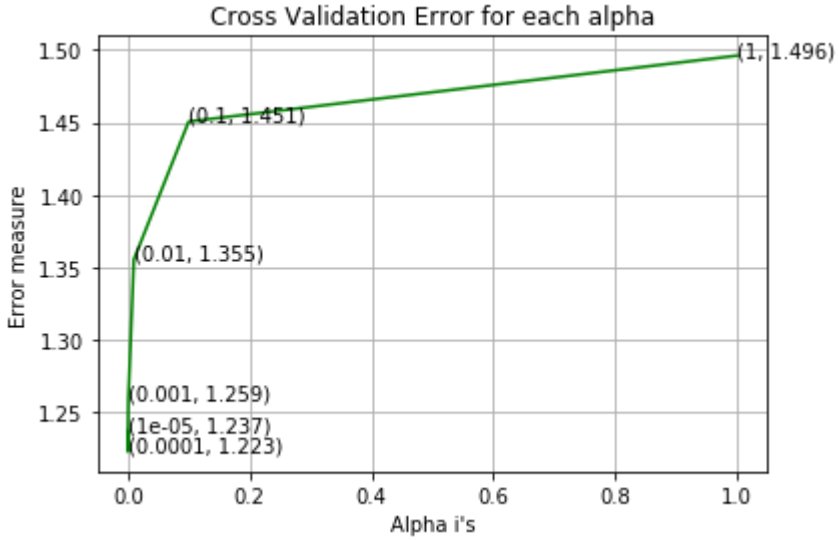
Q4. How good is this gene feature in predicting y_i?

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: 1.2368745507768673
For values of alpha =  0.0001 The log loss is: 1.2227028150130115
For values of alpha =  0.001 The log loss is: 1.258731994752348
For values of alpha =  0.01 The log loss is: 1.3552632355688468
For values of alpha =  0.1 The log loss is: 1.4505687516968373
For values of alpha =  1 The log loss is: 1.4960803362116422
```




```
For values of best alpha =  0.0001 The train log loss is: 0.9944723276058333
For values of best alpha =  0.0001 The cross validation log loss is: 1.222702
8150130115
For values of best alpha =  0.0001 The test log loss is: 1.146286856683591
```

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  231  gen
es in train dataset?
Ans
1. In test data 649 out of 665 : 97.59398496240601
2. In cross validation data 511 out of  532 : 96.05263157894737
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

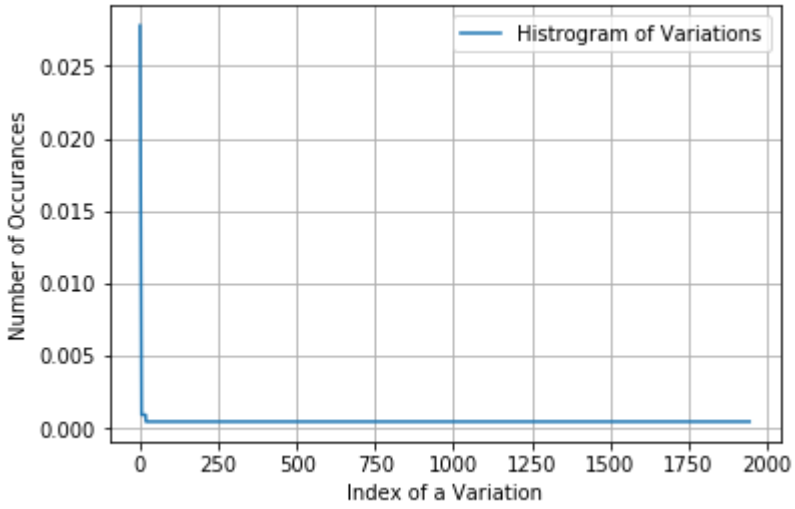
In [28]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1945
Truncating_Mutations    59
Amplification           47
Deletion                38
Fusions                 22
G12V                     4
G12S                     2
Q61R                     2
M1R                      2
A146T                    2
Q61L                     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",)
```

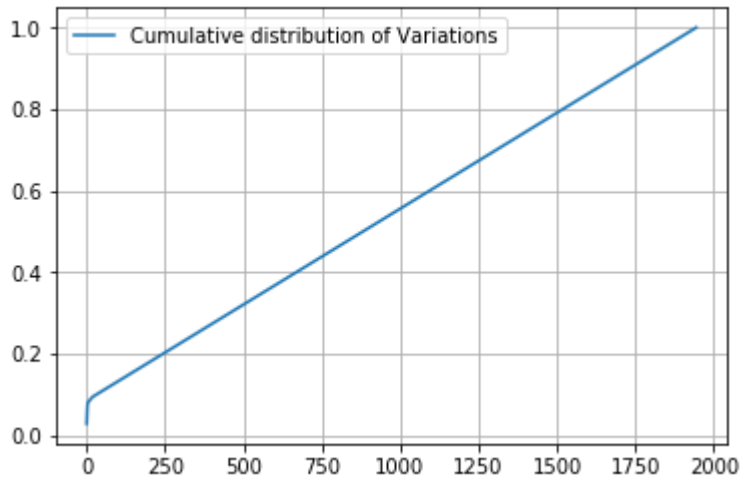
```
Ans: There are 1945 different categories of variations in the train data, and
they are distibuted as follows
```

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



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

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



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

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

In [33]:
```
print("train_variation_feature_responseCoding is a converted feature using the response coding 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)
```

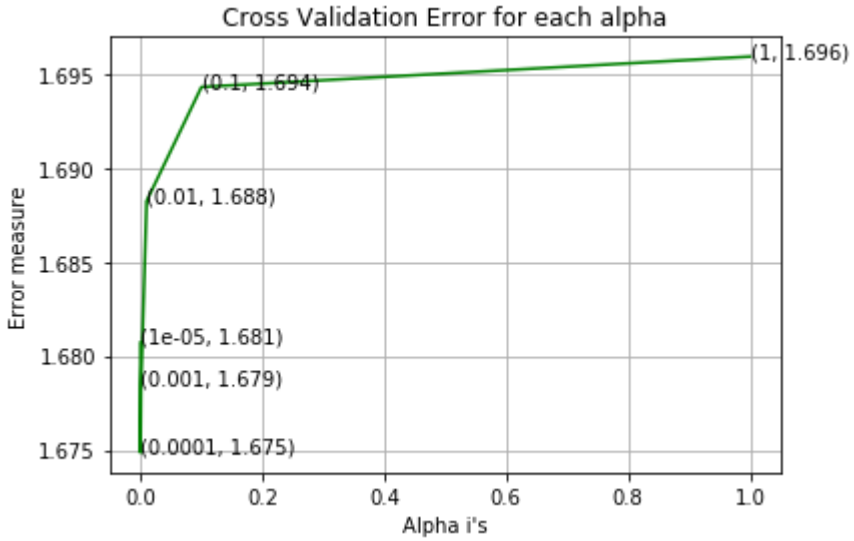
```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Variation feature: (2124, 1975)
```

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)]

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

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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.68070138569254
For values of alpha =  0.0001 The log loss is: 1.6748495691271983
For values of alpha =  0.001 The log loss is: 1.6785183675026936
For values of alpha =  0.01 The log loss is: 1.68823786018618
For values of alpha =  0.1 The log loss is: 1.6943130405613573
For values of alpha =  1 The log loss is: 1.6959299685638511
```




```
For values of best alpha =  0.0001 The train log loss is: 0.7118918480630402
For values of best alpha =  0.0001 The cross validation log loss is: 1.674849
5691271983
For values of best alpha =  0.0001 The test log loss is: 1.7125260608661488
```

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  1945  genes in test and cross
validation data sets?
Ans
1. In test data 74 out of 665 : 11.12781954887218
2. In cross validation data 67 out of  532 : 12.593984962406015
```

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
```

In [40]:
```
# building a CountVectorizer with all the words that occured minimum 3 times i
n train data
text_vectorizer = TfidfVectorizer(ngram_range = (1,4), min_df=10, max_features
=5000)
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({3.625887574715097: 23, 2.8556275619721285: 13, 2.4101451567991883: 1
1, 2.0909803914775082: 10, 1.7543119811153052: 9, 3.0625614793318667: 8, 7.02
0317971149534: 6, 3.432151319880382: 5, 2.6797412944153822: 5, 10.53047695672
4305: 5, 3.0129597036100724: 5, 3.7966991080670827: 4, 1.6677416466194461: 4,
2.2053259255070397: 4, 2.1838202638611737: 4, 2.639128943843483: 4, 2.3694391
75050214: 3, 2.9744098442591005: 3, 5.231726732350539: 3, 5.429752356305473:
3, 2.8149930548240953: 3, 2.471158644473419: 3, 2.3265958569896434: 3, 2.6767
130363629645: 2, 1.7049951645002475: 2, 2.450746618160825: 2, 2.6768045874369
393: 2, 2.3085229547102784: 2, 2.3505788854552674: 2, 4.063019339048954: 2,
8.255276359363751: 2, 2.0141533805532728: 2, 1.9571619956699764: 2, 2.9738319
37170001: 2, 2.6050926559280554: 2, 1.9282680856225125: 2, 5.122427721077339:
2, 7.580765854595517: 2, 2.247628495507132: 2, 7.1018044872958965: 2, 8.20547
258353874: 2, 5.306402992535871: 2, 1.4991125638913017: 2, 6.124371760911243:
2, 2.2664915816224895: 2, 5.44037704975803: 2, 1.6637309869090477: 2, 2.09478
0583253976: 2, 5.439409233603112: 2, 2.2909283032475813: 2, 1.815284047762711
7: 2, 3.1454757354907343: 2, 6.5959721039776165: 2, 2.7938508098668704: 2, 5.
739462017397702: 2, 3.3110670252304932: 2, 1.99123030508651: 2, 2.70156234179
8442: 2, 4.482148059370317: 2, 1.910847513811183: 2, 4.197942746913135: 2, 2.
3273221963209565: 2, 6.408797653175256: 2, 5.954535804311997: 2, 1.6703244457
752309: 2, 3.064851685482067: 1, 1.4257329061923372: 1, 2.2738510924729667:
1, 3.7579014096813808: 1, 4.057396875371889: 1, 5.311301705484784: 1, 6.13640
3180160812: 1, 1.7852222385798093: 1, 8.09828450753458: 1, 9.5819366843496:
1, 10.367194941127671: 1, 11.471495526778448: 1, 12.669232889896332: 1, 2.039
186899824971: 1, 14.520818241446406: 1, 15.546875131776403: 1, 16.17752966143
4242: 1, 17.503557604788675: 1, 18.79932176302024: 1, 19.620815329314944: 1,
20.600037383612065: 1, 21.34334176052012: 1, 22.403821680897646: 1, 23.837135
029595522: 1, 24.90074726197688: 1, 4.779968802429968: 1, 26.991178009663997:
1, 27.638918353233098: 1, 28.59591797917044: 1, 29.10975405133936: 1, 30.7921
83988482492: 1, 5.430704931870105: 1, 3.3402351639235226: 1, 33.5785312952492
35: 1, 2.1592444705437113: 1, 4.3939768808525255: 1, 2.6030180104138734: 1, 3
7.676698527252604: 1, 38.74115628010403: 1, 3.4270784206674305: 1, 40.4481022
8202023: 1, 41.975313734157275: 1, 42.47412851025969: 1, 7.951078656779675:
1, 44.98971436707321: 1, 45.500728598960194: 1, 2.576392282199437: 1, 5.94778
0640151891: 1, 34.437166067217746: 1, 8.460615158050084: 1, 10.02516658628928
4: 1, 51.309799916963925: 1, 52.76763994713095: 1, 53.597519097780676: 1, 54.
67142508887417: 1, 9.597645115507067: 1, 1.552528004986486: 1, 58.66059896366
036: 1, 59.52592300419476: 1, 3.4608633090148806: 1, 10.264120887648698: 1, 6
2.63745944982637: 1, 63.47519192031814: 1, 3.4057583087198675: 1, 2.668742489
607098: 1, 11.507063914283172: 1, 2.753339648006059: 1, 71.95667055851285: 1,
12.272345116449472: 1, 2.9836026616780624: 1, 75.87060761651377: 1, 76.044400
94642851: 1, 1.7366476157496507: 1, 13.31732983915435: 1, 2.44685150012885:
1, 2.600465257103771: 1, 82.11774841612217: 1, 1.5648432964033396: 1, 14.2850
65580229052: 1, 13.772495759934355: 1, 7.383851295722023: 1, 15.1516887057998
68: 1, 3.440596868588455: 1, 7.014303196154409: 1, 3.616003466852476: 1, 16.6
0677832341702: 1, 2.2528540583445253: 1, 1.8605717741593957: 1, 3.00978312665
7114: 1, 102.86092412602808: 1, 17.357900673220197: 1, 1.9657465897115605: 1,
18.46879248442477: 1, 12.913317000300363: 1, 3.0019477058969075: 1, 1.9635750
448149134: 1, 19.23798849865438: 1, 1.4960892655714855: 1, 7.223659803989397:
1, 1.9950087495458544: 1, 2.285089107527914: 1, 20.107215350433755: 1, 6.0508
89216505822: 1, 21.898359777028638: 1, 2.9654857542982365: 1, 4.3981008960557
8: 1, 21.45490995253742: 1, 3.0221797357723124: 1, 2.2080344336448556: 1, 6.2
46088818072248: 1, 3.314009478308067: 1, 22.64233293451436: 1, 4.848612602932
56: 1, 5.944994747620511: 1, 23.71927448969221: 1, 10.719055028066359: 1, 24.
46966306971319: 1, 5.399803988299229: 1, 1.648493431207372: 1, 1.552438601234
9098: 1, 2.673243509441671: 1, 25.899607807358283: 1, 7.709165008693559: 1,
4.223074661633391: 1, 3.240630542501658: 1, 5.337963412824024: 1, 5.700082765
79069: 1, 3.2840310332548492: 1, 1.8404566911089195: 1, 2.192641420777791: 1,

27.395175926526864: 1, 1.627169205496445: 1, 28.257030268515553: 1, 4.4140766
457754195: 1, 29.371994550986145: 1, 176.66516707240038: 1, 10.95227507299796
2: 1, 8.996308826233575: 1, 1.5300328499145215: 1, 30.67920231027361: 1, 8.73
931132879203: 1, 23.58106615209362: 1, 2.4483014548789788: 1, 6.0518737426824
6: 1, 31.691072009287755: 1, 4.500032983260383: 1, 3.387970888591496: 1, 2.78
86392454803417: 1, 3.5195619817106527: 1, 1.3363571462277002: 1, 3.2730571299
944575: 1, 33.10028452509492: 1, 1.4693294782548505: 1, 2.790306021582961: 1,
34.94623980847397: 1, 2.3146781286906886: 1, 11.643072686773392: 1, 1.8700163
355564878: 1, 5.220613720524941: 1, 2.023786553397789: 1, 36.32977447617351:
1, 4.270899844110305: 1, 4.840266663483663: 1, 37.429888065189786: 1, 3.47583
94604896257: 1, 3.0922036883056307: 1, 2.7483565984399854: 1, 1.5667275610384
79: 1, 16.683717700882195: 1, 2.4457021509375485: 1, 2.8742294917991864: 1,
2.34935122725445: 1, 2.426980941674569: 1, 5.7672231366896805: 1, 40.75706475
4691875: 1, 1.97571894400524: 1, 8.747160276968089: 1, 6.931163216157814: 1,
3.1596467454433665: 1, 1.83777159547793: 1, 42.490988027139444: 1, 4.14985768
2535982: 1, 3.542049198869821: 1, 4.041460380180102: 1, 2.471617558790686: 1,
5.9640398765818885: 1, 1.6722319252686222: 1, 44.95698081198181: 1, 1.9553592
962955926: 1, 3.0750601726124764: 1, 45.18564240098003: 1, 4.010504149117691:
1, 6.232429692041801: 1, 9.570038069715793: 1, 1.9131899366589522: 1, 7.74266
365270836: 1, 5.124091555007602: 1, 5.927335126054347: 1, 17.26784142850943:
1, 3.8818704824707906: 1, 2.0630596142548354: 1, 2.9544621882297153: 1, 2.595
778908850232: 1, 3.3087881649958906: 1, 2.17132386831217: 1, 1.86632978545314
15: 1, 3.3541895398379022: 1, 3.8731515141649218: 1, 5.904625378633924: 1, 1
3.02891804844341: 1, 3.6203949545407217: 1, 5.895514968387761: 1, 3.469433342
245832: 1, 6.260858873071813: 1, 2.8975512586440226: 1, 4.239959681457836: 1,
19.32169620928516: 1, 5.059615243252266: 1, 2.2984688479380146: 1, 1.76084775
540031: 1, 54.95139608056999: 1, 2.924032180139248: 1, 4.455656371587809: 1,
2.2104354879265884: 1, 1.7687529535190596: 1, 3.0129906063270777: 1, 4.082498
130607446: 1, 1.811983771239899: 1, 13.698810092831124: 1, 2.409269605646519
3: 1, 7.313271928586689: 1, 12.054807530698975: 1, 5.20955278848061: 1, 1.793
444298290081: 1, 3.0534580851398267: 1, 1.9317389351026626: 1, 2.872149927700
991: 1, 1.6811207899755276: 1, 4.86308408203421: 1, 2.630721848499205: 1, 2.7
369087450703486: 1, 3.162474587592626: 1, 12.62603006894733: 1, 7.62389879516
9287: 1, 2.821312296111336: 1, 3.6777579647602336: 1, 2.830907033927123: 1,
8.236974188847514: 1, 3.432670657730464: 1, 5.104970516929662: 1, 6.443120760
754769: 1, 2.1691193930930908: 1, 2.303637356105988: 1, 2.35108442049306: 1,
4.656780716759844: 1, 4.911979578987715: 1, 3.8719917605265306: 1, 3.60930614
3054212: 1, 2.2566794642792205: 1, 1.618453192602572: 1, 1.8803067664237907:
1, 3.2239864702106775: 1, 3.1985486560006278: 1, 2.9178462842563175: 1, 67.24
724976376093: 1, 9.830845101324828: 1, 5.708011722487928: 1, 7.56029630129529
8: 1, 22.075099096523783: 1, 5.434485061660215: 1, 3.6263067093671264: 1, 2.1
53490673421234: 1, 2.268184175081368: 1, 3.2256562157675193: 1, 19.2029462614
25765: 1, 2.3379954780899417: 1, 4.84476777547612: 1, 1.432352345551413: 1,
2.2109989624830684: 1, 14.872506938738638: 1, 2.754281252144182: 1, 7.5959260
52002301: 1, 2.39074998186459: 1, 4.203415711406276: 1, 10.602639740748943:
1, 2.1527540088099393: 1, 1.7658810593821015: 1, 6.65957875485447: 1, 3.39425
6956509419: 1, 6.685735743543436: 1, 6.0492515600894885: 1, 4.24624420809465
6: 1, 2.0403523709583604: 1, 3.7013870664578454: 1, 2.206277405451152: 1, 75.
42218443704743: 1, 1.7695178937273455: 1, 26.56409506577708: 1, 15.9528844822
08788: 1, 2.3894880800877583: 1, 3.958587483739257: 1, 2.4891061245809163: 1,
1.7222284341578902: 1, 5.025101483294041: 1, 3.2482670210944384: 1, 7.2185838
962026345: 1, 2.237613538169149: 1, 3.2690213635938377: 1, 5.851819202662289:
1, 9.592652214405629: 1, 2.180488274371862: 1, 3.896483878204668: 1, 80.43065
398961494: 1, 2.884146065248308: 1, 2.752412646115428: 1, 1.655576936355323:
1, 2.520457375774235: 1, 4.224922098601949: 1, 2.466611420404034: 1, 4.583469
746951443: 1, 4.115892658283764: 1, 1.9937976041224235: 1, 3.878614810345220
5: 1, 2.8778783258604674: 1, 2.1963530762518064: 1, 8.387053938333342: 1, 2.3

66533330254305: 1, 6.1563083107371925: 1, 1.9461124363076523: 1, 4.1737704186
23581: 1, 3.207500318067715: 1, 3.489416969952265: 1, 1.6211182025018191: 1,
2.323210860409694: 1, 4.801839668171857: 1, 1.763196286083804: 1, 3.039423982
5405998: 1, 9.76460976934505: 1, 1.7333917978523288: 1, 3.0765308089038577:
1, 5.00480689232711: 1, 13.827115240597893: 1, 2.155728944373737: 1, 88.61306
24186441: 1, 9.527790911399851: 1, 6.444620031777867: 1, 10.42082913045972:
1, 7.605353608074499: 1, 2.164616879462307: 1, 3.9035816491642206: 1, 3.54537
98355279575: 1, 5.790823881191164: 1, 1.7812339274420144: 1, 3.08193306046047
78: 1, 1.9719737719917316: 1, 2.509827659632243: 1, 3.6480875388108855: 1, 1
4.010599598205319: 1, 7.98985746381353: 1, 3.135665445533833: 1, 3.9087614166
418057: 1, 4.934763433572184: 1, 10.941746952847215: 1, 1.7219265800215668:
1, 2.258992512569401: 1, 4.184227469959198: 1, 6.336386607440437: 1, 2.070220
0400703186: 1, 4.113632255441366: 1, 3.490817693995701: 1, 2.750100493907105:
1, 1.528610888553239: 1, 2.145272605294336: 1, 1.8483025569968468: 1, 2.54355
99693569633: 1, 28.24716880641267: 1, 3.2270035204909457: 1, 2.50077820038484
38: 1, 2.4907827608872175: 1, 2.11175143118901: 1, 7.57993068817189: 1, 3.226
8940801029053: 1, 2.69883286550381: 1, 2.140147132019486: 1, 5.23932354956736
6: 1, 20.88219016083997: 1, 4.44270302506435: 1, 3.5302754290060037: 1, 2.214
9331032768846: 1, 4.214939014047152: 1, 1.7231116904255908: 1, 3.272856445360
7323: 1, 3.5604764904981394: 1, 3.1465706116663945: 1, 3.4823391351700264: 1,
5.807100308089118: 1, 29.064471079777146: 1, 12.045328337524897: 1, 4.3121111
53179527: 1, 1.6020427951674148: 1, 5.730122577148186: 1, 8.144132719980467:
1, 4.95543341254961: 1, 2.2647061549251988: 1, 2.698776490201957: 1, 6.230431
71994238: 1, 3.250219649459763: 1, 21.299628777169687: 1, 4.457758745135036:
1, 4.043372495228603: 1, 4.465155710890899: 1, 3.8824562193494008: 1, 3.91230
6626352418: 1, 2.1457278021866353: 1, 2.59794000895381: 1, 5.300573442530148:
1, 4.550259802787657: 1, 13.675362780831282: 1, 4.523686230851392: 1, 5.60335
229688869: 1, 2.637716172388793: 1, 3.1173285703420768: 1, 3.643225532486094
5: 1, 3.1765370010198075: 1, 3.339252010085098: 1, 2.7737919426110493: 1, 1.8
559918207253165: 1, 3.7972805658744466: 1, 2.7174522071754676: 1, 6.019647340
56883: 1, 14.731020719345342: 1, 5.662732913056473: 1, 8.586593240834762: 1,
2.954983298283294: 1, 10.466163728646853: 1, 1.428669001602798: 1, 1.95775446
78725423: 1, 3.201944372276053: 1, 2.6342319284992053: 1, 6.7115582394415085:
1, 3.2099551532997173: 1, 4.043524334834495: 1, 4.049558634213732: 1, 3.38201
5083332764: 1, 3.171374153691866: 1, 11.997472295344572: 1, 2.87398036549246
6: 1, 1.948952187726642: 1, 3.529815010512083: 1, 1.9227671905891557: 1, 15.1
63192118075589: 1, 2.0665223291878476: 1, 2.641093979348129: 1, 4.44017965586
06: 1, 11.14517857655415: 1, 2.1449455240259083: 1, 5.787222347310947: 1, 1.7
555843936648283: 1, 3.659884070247997: 1, 3.5680268173108756: 1, 1.8634775837
481456: 1, 5.13439970330223: 1, 2.156861681297082: 1, 3.760294890302257: 1,
2.0252352635185766: 1, 1.6950664629178602: 1, 3.7925650724348423: 1, 3.890156
631928478: 1, 2.5925836489853844: 1, 2.1409105702958717: 1, 3.41638692665825
6: 1, 2.221031242676781: 1, 17.369668278663294: 1, 12.853347848034337: 1, 3.0
11114559565647: 1, 2.8479342600127526: 1, 5.025264607581465: 1, 2.89789140571
8907: 1, 6.596581726632715: 1, 6.493749067992504: 1, 2.215316609415191: 1, 4.
65562171612647: 1, 3.64858181132494: 1, 2.3947120383446174: 1, 4.238435416143
2665: 1, 1.7550679187500964: 1, 4.92202764313513: 1, 4.043378812547263: 1, 1.
8828878017650879: 1, 4.458077492603946: 1, 13.270942141757365: 1, 9.965584487
09073: 1, 7.233747708566349: 1, 2.9178542283188804: 1, 5.090236467770305: 1,
6.932175434511235: 1, 3.163530030060836: 1, 2.4111530327438504: 1, 1.88239507
82471108: 1, 2.7648295612465055: 1, 2.5495960124630686: 1, 1.734377051712686
2: 1, 17.00422528685424: 1, 1.96040628764337: 1, 4.9036646934343615: 1, 3.438
9107259269927: 1, 2.1640567123092507: 1, 1.8351762166547838: 1, 2.68844499564
98844: 1, 4.150812363224643: 1, 3.4499926526622136: 1, 6.05024566168664: 1, 1
0.55829310536015: 1, 4.372350223646843: 1, 3.057498931841774: 1, 3.1175167057
414654: 1, 2.4187196333450736: 1, 7.578572016138513: 1, 2.7656803497484206:
1, 3.1935534156510137: 1, 1.4414173294398036: 1, 28.620671957059358: 1, 4.818

270207429332: 1, 11.220154413622716: 1, 1.7504609087706906: 1, 2.955758445040
932: 1, 1.7644544895032648: 1, 2.2366455062378012: 1, 2.0768174892448936: 1,
7.46760415907273: 1, 4.32164272683572: 1, 2.3762620435751027: 1, 5.5384302725
91607: 1, 3.007216710962821: 1, 2.48577892923041: 1, 3.622564948558317: 1, 2.
0774866142207142: 1, 2.7873830376786204: 1, 1.8839238828563163: 1, 3.83867765
7637731: 1, 4.032069617630492: 1, 12.165968447953567: 1, 2.5551101860784273:
1, 1.925952992558192: 1, 8.107062658835284: 1, 1.9259433074533463: 1, 6.44025
9473435229: 1, 21.09660332997237: 1, 4.626088738165446: 1, 1.734029337896947
4: 1, 2.5760299468682435: 1, 1.9835305475367995: 1, 3.2958256201291247: 1, 2.
7725719264659525: 1, 13.425373397259484: 1, 1.6852661682069827: 1, 2.83683967
9299731: 1, 1.9929804805129536: 1, 1.8562537073969432: 1, 9.529641700635715:
1, 1.9688209171889477: 1, 5.418280334792154: 1, 2.8633913996586724: 1, 6.0056
014444981525: 1, 3.3317967340505676: 1, 2.4262598519825196: 1, 4.996664631775
92: 1, 2.182509761706424: 1, 31.282834334689824: 1, 14.320911636135868: 1, 5.
163497282843976: 1, 2.4923576004823365: 1, 3.579010056910936: 1, 10.161178025
855415: 1, 10.201276952609387: 1, 1.8049269403211652: 1, 3.4684547245403228:
1, 6.093438638500071: 1, 6.301032850152279: 1, 4.467065123994257: 1, 3.702413
8569603173: 1, 2.2444989181386417: 1, 2.7748773640357864: 1, 1.81373676999707
68: 1, 4.567391865073963: 1, 1.465062887921946: 1, 6.313886106634727: 1, 2.99
88979140201186: 1, 2.1554737159941917: 1, 2.600874648223908: 1, 11.1051719542
38367: 1, 1.9257142927141317: 1, 1.8322094066031944: 1, 1.9777242417849823:
1, 3.368198454066425: 1, 2.7751450988198108: 1, 1.4272364809394888: 1, 7.8060
36399758693: 1, 1.683919510945212: 1, 5.403167649844886: 1, 11.56111584572474
8: 1, 3.1259246197595427: 1, 2.4871012023914276: 1, 1.802061810783882: 1, 16.
36180940248574: 1, 8.891543781742655: 1, 3.5562030143237746: 1, 3.64656963883
54205: 1, 2.2077034473901884: 1, 4.789234078695525: 1, 2.204410337366998: 1,
3.379590321378695: 1, 3.683125008495796: 1, 1.5014737428897873: 1, 4.50357784
2356237: 1, 2.6370356627976412: 1, 9.649432210604527: 1, 4.384898375764818:
1, 1.7208665319427654: 1, 3.9264054124781027: 1, 1.5394477703444704: 1, 2.486
6195158126385: 1, 3.9845054527417934: 1, 1.9221274269387858: 1, 1.66930018555
17238: 1, 3.6756872467995536: 1, 3.8166432058255046: 1, 4.273852051353496: 1,
4.775696923778309: 1, 3.5669690721550005: 1, 4.674457527200706: 1, 6.79750864
321445: 1, 2.4089045623657825: 1, 3.812199704544183: 1, 5.714591661911653: 1,
12.330152149222432: 1, 3.52934466273821: 1, 2.2292006185002573: 1, 1.57382291
88541624: 1, 17.754185386280376: 1, 4.021381101013013: 1, 2.7730951683079317:
1, 2.018558288490439: 1, 2.8856149741347545: 1, 2.141222680372381: 1, 5.39810
0349881575: 1, 2.59626884260378: 1, 10.030752586853861: 1, 1.851251452841580
9: 1, 4.245535168848818: 1, 6.536806024150691: 1, 3.5431121764270417: 1, 3.69
78487289504165: 1, 2.770219408638789: 1, 4.043913106892503: 1, 3.018099866174
8926: 1, 15.00093253197387: 1, 2.976566437338434: 1, 3.277212246904492: 1, 1.
6963955946375084: 1, 8.382422579356497: 1, 15.693607218313455: 1, 4.422864782
5371: 1, 6.590385145824322: 1, 4.345469310788172: 1, 11.366568593180556: 1,
3.6684556424545742: 1, 1.5834526503726358: 1, 1.9046323750155536: 1, 2.061698
457355042: 1, 5.234463707616967: 1, 2.389181539845711: 1, 5.856820997088217:
1, 2.78787192365148: 1, 1.9672144026390734: 1, 2.4421666115170924: 1, 17.7149
50929234156: 1, 1.8706910904812608: 1, 3.490628638370741: 1, 9.7657393382862
7: 1, 2.541485840741716: 1, 2.2144233734055523: 1, 8.469882748938419: 1, 1.72
64838547361887: 1, 3.3471737502029644: 1, 8.630904264124991: 1, 2.39325958535
65646: 1, 2.6757962427892115: 1, 4.252802412541328: 1, 3.7221317997086065: 1,
2.7238067716750303: 1, 1.5439260259937264: 1, 3.1611166805272317: 1, 2.113320
9145269225: 1, 2.0900718202774295: 1, 13.71288410358967: 1, 4.46001486284834
9: 1, 2.3217420154284567: 1, 8.086835949045934: 1, 3.473547269166082: 1, 7.10
3429457212781: 1, 9.627054563701234: 1, 2.7709091013015614: 1, 22.62016408420
1072: 1, 1.2895960788387426: 1, 5.238924470250495: 1, 3.092753191239423: 1,
2.5210995279993345: 1, 3.8685326440476766: 1, 2.685352622179822: 1, 5.4268432
50273559: 1, 2.7884236965246285: 1, 1.3371522172382135: 1, 2.599765515472113:
1, 5.002956283349716: 1, 1.637927240824706: 1, 3.210238492391014: 1, 3.541325

0493308737: 1, 8.275640315207497: 1, 1.504928115073289: 1, 2.650269785969609
6: 1, 3.6240548945672466: 1, 3.6093767942954122: 1, 2.207627564555344: 1, 3.5
08340540777573: 1, 2.1511385955066333: 1, 10.604810565947414: 1, 4.8464115999
58814: 1, 2.342232694159401: 1, 3.9722302421654248: 1, 5.155100435040504: 1,
2.062709261891671: 1, 5.52471613344991: 1, 5.30578236091304: 1, 2.70357983529
736: 1, 1.7243589383977378: 1, 1.943944471467343: 1, 4.2653582763025195: 1,
2.6127356091679155: 1, 1.8192613959118622: 1, 5.1897739619905705: 1, 1.629025
3958418954: 1, 17.36128582357955: 1, 2.93891945723825: 1, 7.112858831678878:
1, 1.832440593550134: 1, 3.541196093967156: 1, 5.667574230417203: 1, 7.603917
91910776: 1, 5.381234387789683: 1, 3.4282111154069685: 1, 2.3441062195335456:
1, 2.0458604375566085: 1, 1.8917277480558878: 1, 16.92455203585224: 1, 2.0525
9115872674: 1, 2.001202361016838: 1, 1.605125028077445: 1, 19.39897947176955
4: 1, 3.9572259006217587: 1, 1.9512958596228345: 1, 8.18213358059576: 1, 2.35
15122816205496: 1, 6.695226721967473: 1, 1.5187120470901163: 1, 4.15568255670
634: 1, 2.165973253438289: 1, 3.2327471910296475: 1, 2.5548255991064535: 1,
2.072164544892157: 1, 1.7247997574989884: 1, 2.6223769858815418: 1, 4.4842594
73857869: 1, 5.855563926339721: 1, 13.145651044813588: 1, 4.30869552036675:
1, 3.5611628150819077: 1, 4.8181016278718936: 1, 1.9592818996578705: 1, 7.916
956159887304: 1, 2.887766903575502: 1, 2.2391664896202044: 1, 2.2193354426770
42: 1, 4.489005654596911: 1, 5.868489934805682: 1, 3.367304819666703: 1, 3.19
7513281371772: 1, 2.4877837288594153: 1, 1.8515698020149962: 1, 2.34764238874
5755: 1, 1.8835499177905397: 1, 1.6765811887661735: 1, 6.080894418345967: 1,
9.364578697903514: 1, 14.485142372239734: 1, 3.0535432800288227: 1, 4.5795907
69974673: 1, 10.488314933941766: 1, 1.8285761417575126: 1, 9.316858941798024:
1, 3.087036999905375: 1, 9.46235764103554: 1, 6.553723295349657: 1, 6.4257471
056556135: 1, 2.4695881547815666: 1, 4.500734738665304: 1, 4.083568418702913:
1, 6.0888864648934655: 1, 3.0005112253959285: 1, 1.7574527897598655: 1, 2.521
0374292719457: 1, 6.9173978821446225: 1, 1.8171799443469365: 1, 2.72225956845
79263: 1, 2.773161410638052: 1, 1.785725444084275: 1, 2.1661478813968547: 1,
11.688080002229995: 1, 2.6544542357851553: 1, 3.116393671881362: 1, 1.9285869
048089281: 1, 7.738356363687801: 1, 4.451987930920681: 1, 5.045081991127478:
1, 9.452366457164882: 1, 1.7635337141415135: 1, 3.1143750232646603: 1, 2.9033
195120029416: 1, 2.593924046492827: 1, 1.5645103349211995: 1, 9.4775537914392
77: 1, 3.136371752691649: 1, 3.8702206434329085: 1, 3.0806220320363398: 1, 8.
6736470935182: 1, 12.199546738725566: 1, 2.050996754220711: 1, 6.929526653008
412: 1, 21.786739662165143: 1, 4.747646833356516: 1, 3.1283526120196723: 1,
2.5169867756752256: 1, 1.9023936077805863: 1, 2.9542164740314125: 1, 6.342737
644143019: 1, 9.612519923797224: 1, 7.261820479320323: 1, 2.872198255964981:
1, 2.22895129791349: 1, 5.37662711588111: 1, 2.1855172769537408: 1, 3.0319964
40271276: 1, 116.39598512176664: 1, 2.8983697225064495: 1, 4.922832504805191:
1, 1.6494671017818714: 1, 2.078081362395264: 1, 1.4636305666245175: 1, 8.2128
54037704556: 1, 2.489838828651229: 1, 5.783205861592096: 1, 10.48470471881104
7: 1, 1.96929767458588: 1, 7.620613698094983: 1, 3.86460496722713: 1, 5.29863
6534717749: 1, 6.752979348842036: 1, 4.166675944747252: 1, 3.189932193482936
6: 1, 11.276261267346989: 1, 2.5517471107608767: 1, 5.868259834055716: 1, 16.
3615011192407: 1, 4.938176663906241: 1, 2.2404007338362404: 1, 11.15461704462
0311: 1, 5.693857273802708: 1, 1.5999171779937515: 1, 3.2335172845507953: 1,
7.521613199413554: 1, 3.0268521963784116: 1, 4.9637335298844345: 1, 2.0110311
11193399: 1, 4.565379353268459: 1, 2.4730404323464: 1, 2.3818802966713513: 1,
2.420685324212283: 1, 21.040947217732363: 1, 2.64638317656353: 1, 5.308414831
28477: 1, 11.404861509830317: 1, 6.887065898985984: 1, 3.8457574398854795: 1,
12.331212776410796: 1, 1.7946972247834434: 1, 4.184948473210082: 1, 1.4909561
703849257: 1, 3.341439987966561: 1, 3.786717023374162: 1, 6.22928682925063:
1, 7.333108276856422: 1, 3.338167385531873: 1, 1.9847101561054252: 1, 4.13794
8224585349: 1, 4.204313327699325: 1, 3.7560450196968334: 1, 1.780943631099817
2: 1, 2.8625648421035157: 1, 1.9096821511476494: 1, 4.193885127152676: 1, 3.5
879007326901893: 1, 2.087313110189597: 1, 1.7005011090577211: 1, 1.9307465321

94799: 1, 1.775490577828769: 1, 2.882060730696922: 1, 1.7780512436080587: 1,
2.091713345656521: 1, 21.835002806679913: 1, 7.398598991666445: 1, 2.76385676
442215: 1, 5.796019921885792: 1, 4.410347412886483: 1, 2.069003422725767: 1,
3.8090333577418596: 1, 9.52659510946924: 1, 2.7332687868613372: 1, 2.72192625
9458948: 1, 4.607095086306187: 1, 2.3426760400395863: 1, 12.043226370016633:
1, 14.211048671493716: 1, 3.0833864269398186: 1, 4.046226614826277: 1, 3.1947
939403952357: 1, 1.672387167759702: 1, 13.727907899161831: 1, 3.292129722326
9: 1, 6.392919020134078: 1, 1.553730548322366: 1, 6.574096343213836: 1, 4.567
590990532572: 1, 3.686591286995046: 1, 15.94716627273446: 1, 2.58900669687331
94: 1, 2.427153522946579: 1, 1.6569210027427563: 1, 4.290874846742901: 1, 15.
477316184457452: 1, 2.0466550408375066: 1, 3.9424733296520706: 1, 7.172148641
705187: 1, 1.7101029193092896: 1, 4.156087267914341: 1, 3.46993798103826: 1,
1.3966514253580236: 1, 2.9661269800240353: 1, 1.7369700612906205: 1, 5.461799
245222628: 1, 9.722240899551931: 1, 1.9220753199094502: 1, 10.16628998850341:
1, 3.7863534058021955: 1, 2.0348772614997115: 1, 2.1491620961230002: 1, 4.728
656122083545: 1, 1.6691050397793896: 1, 4.122369958839861: 1, 1.8324532949530
854: 1, 1.5025083418228584: 1, 2.9769509334324202: 1, 3.370081578223591: 1,
2.6015684813390454: 1, 3.0161764206120485: 1, 3.7499289956216963: 1, 4.022057
347417078: 1, 6.177111613542944: 1, 6.344384664694949: 1, 4.250632166841591:
1, 2.5272884207175483: 1, 3.453019040653253: 1, 2.475860988379588: 1, 1.45889
55404542445: 1, 16.01044396539419: 1, 1.6714251859683011: 1, 2.68427218721232
7: 1, 2.5196322817011914: 1, 3.977956821849422: 1, 2.2305455187716494: 1, 2.6
197703575818916: 1, 1.6477769868166519: 1, 9.699219326274394: 1, 3.6692212055
767848: 1, 7.669511782430013: 1, 3.8244381633244435: 1, 2.652715999386593: 1,
2.0396065483386465: 1, 5.661862701874178: 1, 3.722005285377149: 1, 2.05943444
99574446: 1, 3.182778689593339: 1, 3.1895114592881977: 1, 4.207357113297111:
1, 2.0090450132034845: 1, 1.5678394108809122: 1, 1.679122644099289: 1, 14.967
757155662884: 1, 4.074710976431761: 1, 3.1226381190986165: 1, 5.2589495676525
6: 1, 1.3565847211715811: 1, 6.332753581308019: 1, 6.0239019717869375: 1, 2.9
14418254425729: 1, 4.443333886783841: 1, 3.251534630948066: 1, 3.832650531264
4343: 1, 2.999587302270948: 1, 1.585120217377432: 1, 2.972587750309825: 1, 1
5.160798586982327: 1, 9.781251791495707: 1, 2.784695260227584: 1, 7.631519748
180617: 1, 11.754858870429436: 1, 5.44637882967618: 1, 1.7528414151896168: 1,
4.974314603880159: 1, 3.6477449240164597: 1, 7.483650062480272: 1, 2.08085822
75481973: 1, 6.10533861860679: 1, 5.007710390243994: 1, 3.8994405189759487:
1, 1.5979437018284244: 1, 4.422790702258158: 1, 7.390763809328934: 1, 2.35035
42632770924: 1, 2.4744191058678773: 1, 1.9368575940735253: 1, 2.2863255626177
024: 1, 16.027229014329254: 1, 2.458582884332292: 1, 2.3246526978160778: 1,
1.8072385951392003: 1, 12.837661803981879: 1, 2.738117765833424: 1, 1.5532385
977504461: 1, 39.98045575423321: 1, 2.0144304098921935: 1, 6.243466307254004:
1, 26.312817747361308: 1, 4.268397389508914: 1, 3.559900065811855: 1, 1.81338
17414068842: 1, 2.432408618880653: 1, 2.241579837538975: 1, 1.378003475438386
5: 1, 5.27250730936532: 1, 1.7446855237576178: 1, 2.219769257311656: 1, 2.782
7609656212524: 1, 3.0605480322925644: 1, 3.8217589745761438: 1, 2.80820742919
31627: 1, 3.7934412658744994: 1, 1.9932509333938346: 1, 7.12020345937193: 1,
2.7393186931872604: 1, 10.544922518356216: 1, 5.3289534739345985: 1, 11.95962
0894979826: 1, 3.865647905816867: 1, 2.5161186387640653: 1, 2.41186161747361
5: 1, 2.3189130605908557: 1, 7.020324302134006: 1, 2.0346340011737696: 1, 5.2
02927095641213: 1, 3.523774013870825: 1, 3.6289971665822143: 1, 2.48469292479
9746: 1, 2.977489661107438: 1, 10.302854084036902: 1, 1.702508767543389: 1,
5.029762914499078: 1, 6.7591021384177115: 1, 2.891312711441053: 1, 2.00144733
2473085: 1, 2.8104342153863735: 1, 4.277028250666002: 1, 3.5534734174923375:
1, 2.9537479882731326: 1, 5.471847689642625: 1, 1.84363869654845: 1, 4.611313
015995171: 1, 1.938118102588151: 1, 1.8832848013993635: 1, 3.90293045271935:
1, 9.86685551991663: 1, 2.388447340670766: 1, 45.124726786492154: 1, 2.439362
0982549096: 1, 1.9200767650778006: 1, 11.471034774902158: 1, 3.37471979554678
2: 1, 1.7764946907102057: 1, 2.6267679592792885: 1, 2.5214297872751725: 1, 7.

39027477347272: 1, 1.5793254236911969: 1, 5.803876568058821: 1, 3.30323794125
2901: 1, 2.133396572986185: 1, 3.116556357426955: 1, 9.69363751626821: 1, 2.0
032040872981125: 1, 8.56432151540026: 1, 4.748016244439916: 1, 4.768300777819
4835: 1, 2.570306495796036: 1, 4.729516280853569: 1, 12.62683402686175: 1, 1
3.797597484255917: 1, 1.7258459548265064: 1, 1.8171675222755372: 1, 12.045036
413199707: 1, 6.562769569403794: 1, 2.602081501589138: 1, 21.128380678381763:
1, 2.2776971628808207: 1, 4.522137773131956: 1, 3.5804879127412588: 1, 2.6203
688311066715: 1, 1.7963588807819797: 1, 3.2749172705375442: 1, 4.134940099204
4485: 1, 1.4184977260746279: 1, 47.12340186246502: 1, 2.095619398633388: 1,
2.475987296165923: 1, 2.116623403736072: 1, 4.038862772855395: 1, 4.312087781
8696055: 1, 9.04791951557072: 1, 7.689981770393241: 1, 4.020870906771038: 1,
4.554703939429446: 1, 2.2504755331646824: 1, 1.8114812100682571: 1, 2.4610499
243779542: 1, 3.864397554843632: 1, 2.850201560961293: 1, 2.131179892546027:
1, 4.624535270720024: 1, 4.068414705775247: 1, 14.074118366318679: 1, 2.11077
8756103793: 1, 2.0910317027083454: 1, 2.6439026924686586: 1, 3.76415693210368
03: 1, 10.55810387619672: 1, 1.993721129021351: 1, 2.4605834381481535: 1, 15.
324671635585766: 1, 6.693307737357511: 1, 2.4968965177795823: 1, 3.1415296699
320105: 1, 4.4022552577892755: 1, 3.3006092717205653: 1, 3.264293278154921:
1, 3.2821627653677155: 1, 2.227321495686992: 1, 1.9978974222680663: 1, 2.2311
19397465844: 1, 7.161017854869024: 1, 3.4375065286247994: 1, 11.4227129631814
48: 1, 3.2556458939478508: 1, 1.7193824798811892: 1, 6.477720097166207: 1, 3.
739372505723717: 1, 3.6778661320822255: 1, 3.8703079153143682: 1, 3.626504145
478121: 1, 2.552213954013348: 1, 2.123933148150608: 1, 3.1821018157061456: 1,
2.1888516210466933: 1, 1.7010301141005906: 1, 1.6307438765802251: 1, 2.445304
0214150366: 1, 2.77872171937534: 1, 1.9700109443132476: 1, 5.10644090878969:
1, 3.3595092524065744: 1, 2.804602789826884: 1, 1.4534459993294455: 1, 2.0774
702413933044: 1, 8.668635647714634: 1, 10.265315161016188: 1, 2.3671116919594
035: 1, 6.494451031716122: 1, 4.728503222584008: 1, 2.977117852455048: 1, 3.0
28419836707427: 1, 1.7184617774390185: 1, 2.506579425077999: 1, 3.08373996520
03485: 1, 4.160125342882226: 1, 5.046332993569085: 1, 6.180477365252387: 1,
7.356327989002332: 1, 13.84782825851662: 1, 2.328019125347438: 1, 5.293942643
7070285: 1, 2.9154616972248877: 1, 1.406735101226842: 1, 6.280958904919421:
1, 4.252514947297208: 1, 3.2141500709048354: 1, 1.8659550547228176: 1, 2.1003
22600899315: 1, 3.122191251155125: 1, 7.51599296601884: 1, 11.89223559324961
2: 1, 2.4478269051656767: 1, 2.0467071165894812: 1, 3.0010599346227016: 1, 1.
5040991774608212: 1, 2.4233508213471846: 1, 1.8421053346919927: 1, 1.90178700
99005734: 1, 3.5261354705316017: 1, 2.3754356548831037: 1, 3.963107756802617
4: 1, 6.710086528305598: 1, 3.2883552610809144: 1, 3.144436842309253: 1, 27.8
68606925606645: 1, 5.606800621731426: 1, 3.5734612949864357: 1, 2.60957805527
39036: 1, 3.263352173299799: 1, 3.559243327570792: 1, 3.2528753878073036: 1,
1.6133781730843555: 1, 3.6345794116248387: 1, 4.525417077250579: 1, 3.3575949
09963154: 1, 4.063756667658187: 1, 2.1990482702168777: 1, 1.6351360419257472:
1, 4.72956664583183: 1, 3.2735221773703653: 1, 11.314173250331375: 1, 3.64132
2145877802: 1, 1.439963667198598: 1, 2.4702545678209784: 1, 7.20194460761800
1: 1, 1.909472967799804: 1, 3.5720826041836347: 1, 2.114028308066493: 1, 2.12
54069865997853: 1, 1.7487695208905913: 1, 9.559809772756466: 1, 2.66337092688
2018: 1, 1.5066601895578566: 1, 9.06645560742078: 1, 2.779125522968993: 1, 1
2.94170633753313: 1, 4.321306801437981: 1, 3.861731686656249: 1, 8.7111997088
51966: 1, 1.481970798256371: 1, 3.1839034048838113: 1, 2.116695045264599: 1,
6.047119034306859: 1, 1.8030659806730056: 1, 4.518340290034331: 1, 2.41601447
7969691: 1, 3.9273131594405304: 1, 2.9438961988122125: 1, 2.046746537479091:
1, 1.9932266034091994: 1, 2.1339040308710486: 1, 4.469025795627665: 1, 3.5275
46781961323: 1, 3.588695252892382: 1, 2.780786101876175: 1, 2.067263726655797
4: 1, 4.097434534401811: 1, 2.963933932607563: 1, 2.2991785181695588: 1, 9.72
7732697539095: 1, 4.008121877369759: 1, 7.437269053560811: 1, 3.0923927107928
33: 1, 22.740651504541482: 1, 5.957623227169421: 1, 7.003580014056512: 1, 1.5
407048574406468: 1, 2.0867364137414857: 1, 2.049070826993899: 1, 1.8917337764

571887: 1, 5.606052947758894: 1, 2.75747671385737: 1, 2.5012096407601554: 1,
3.5460376363023505: 1, 1.9137948339104005: 1, 10.825928640889368: 1, 10.26840
6348356079: 1, 1.417094095292031: 1, 2.7385900056843346: 1, 7.81573437621747
7: 1, 3.92370816846724: 1, 4.9027850010162135: 1, 1.8610781918509924: 1, 3.35
7802158938854: 1, 2.457993999064259: 1, 1.9808553304210954: 1, 3.085420843976
3817: 1, 2.1555484348621103: 1, 15.660764420853436: 1, 2.2968491615575592: 1,
2.617272693425107: 1, 2.0024183047525526: 1, 2.736872016139714: 1, 1.83164426
39588026: 1, 8.628026900042931: 1, 11.283792713023585: 1, 1.8739799459588777:
1, 75.83478980036784: 1, 1.9911330754527001: 1, 1.6678164559903788: 1, 3.5136
924987624703: 1, 7.357152261737866: 1, 3.213597595258183: 1, 3.41952657450133
35: 1, 1.6193388093747418: 1, 4.925151496135598: 1, 3.2337522354734194: 1, 2.
2346535738417783: 1, 2.0193687504506124: 1, 1.617153929867615: 1, 4.459937226
231052: 1, 1.8755071703146784: 1, 3.1668103984293823: 1, 1.9201766558214877:
1, 1.910561688782953: 1, 2.6149808356744333: 1, 12.337284804807757: 1, 1.8645
94957146971: 1, 4.239317125895132: 1, 4.411711013902875: 1, 6.40701932291162
4: 1, 4.075958014317168: 1, 2.0125918714130018: 1, 4.491058437718069: 1, 3.33
12045551897844: 1, 3.0657587664734955: 1, 2.026457952611824: 1, 1.81910816879
7928: 1, 9.5189251627598: 1, 1.6849805240674063: 1, 3.5250662178898993: 1, 1.
9479174591682051: 1, 13.613158967288909: 1, 3.6659199484441194: 1, 13.0963055
59701799: 1, 3.079003460782044: 1, 1.4469673029474455: 1, 9.358447299803366:
1, 3.973261118602461: 1, 2.720224088673112: 1, 3.609502653511779: 1, 2.424124
2497585556: 1, 1.5328603253827098: 1, 5.88550369534906: 1, 2.399233186267857
8: 1, 3.1029210391451443: 1, 3.05945568655338: 1, 2.024233019978409: 1, 18.80
870445239034: 1, 1.4953264719272603: 1, 3.9380419178247354: 1, 3.148455412779
5215: 1, 2.251836841774816: 1, 6.002736105716455: 1, 9.259692211574336: 1, 6.
888971819938666: 1, 2.9670056558048565: 1, 10.96899891184276: 1, 1.9218615644
786292: 1, 1.5533968118819315: 1, 2.7912357308374163: 1, 3.64975790721348: 1,
6.571240509233606: 1, 2.65988247449392: 1, 9.357745455959599: 1, 4.3732534682
72739: 1, 5.921567978340925: 1, 19.90682282020421: 1, 2.653941878494858: 1,
2.4950245917794156: 1, 1.9476334250711949: 1, 1.8124702372452401: 1, 2.784066
5298251523: 1, 4.506355444998978: 1, 7.740944357270771: 1, 1.701686318790464:
1, 2.089963384205197: 1, 2.4765556813751477: 1, 2.4932246417375716: 1, 2.4233
584195635163: 1, 4.544070188699761: 1, 1.6463499718590426: 1, 4.0251645094763
395: 1, 7.206506655086029: 1, 2.3136197104039757: 1, 4.741782154801557: 1, 5.
400258251730608: 1, 2.190154240509234: 1, 2.0852856403718185: 1, 1.8788457729
35823: 1, 2.9853983128268435: 1, 3.1716113883280115: 1, 2.9067876188555464:
1, 2.133646906862551: 1, 2.346112527900089: 1, 3.731933769840387: 1, 2.745778
6759412905: 1, 6.073447538279317: 1, 10.047178845927803: 1, 1.582480242778605
8: 1, 6.157067345379101: 1, 2.9134617709328747: 1, 6.713068155793482: 1, 4.87
00209415461035: 1, 3.2772283556172566: 1, 4.6777413178411145: 1, 3.4448863643
188514: 1, 1.6460562125932139: 1, 2.302313274202537: 1, 2.9651339795097122:
1, 1.731201499854142: 1, 2.566411701623326: 1, 3.7274358274331143: 1, 5.55813
71645349575: 1, 2.0554042476165786: 1, 13.30791641528279: 1, 4.506173065026965
65: 1, 3.353246103488702: 1, 3.995440167703497: 1, 1.938553764831773: 1, 9.27
0849650793737: 1, 3.3550303984638052: 1, 2.8453057242256157: 1, 2.99053273613
12626: 1, 7.376731310332363: 1, 2.2595046845883378: 1, 5.1233418733553755: 1,
2.7517583886915222: 1, 3.289422487158821: 1, 2.5996209352554827: 1, 2.1484780
247882376: 1, 1.6227848803612084: 1, 1.5275249213978324: 1, 3.55409221633290
6: 1, 1.9862720058984864: 1, 2.6899687093816302: 1, 2.0286518168069274: 1, 3.
4362859594687545: 1, 10.87415397576902: 1, 1.7142096642362985: 1, 4.788178322
967639: 1, 3.656968761501144: 1, 6.337759685367278: 1, 2.774675710286055: 1,
4.985981169845876: 1, 4.835736501136122: 1, 2.7627384724756094: 1, 1.93155593
09329218: 1, 2.068103771022405: 1, 2.5240227745270665: 1, 1.96273889353341:
1, 5.909621291772195: 1, 2.390216348386805: 1, 3.249122438801328: 1, 1.998221
1578838196: 1, 4.1925054527562065: 1, 1.8816272725937035: 1, 5.09327944320577
7: 1, 2.8515393870311923: 1, 7.618832107379408: 1, 6.818053630370198: 1, 5.05
7490690749977: 1, 1.541535567018155: 1, 3.636343740658651: 1, 2.1081759775170

7: 1, 2.005069179078595: 1, 7.638338783701838: 1, 1.3378793067132397: 1, 3.26
53452072590343: 1, 3.9769335795443124: 1, 4.068540953916335: 1, 2.32038503450
71264: 1, 6.141733712252976: 1, 4.350675401393854: 1, 2.5193053969266566: 1,
3.9358238880846024: 1, 1.7496733339529478: 1, 8.59758290091398: 1, 1.99726150
25135176: 1, 11.712425775373594: 1, 6.4069717119708445: 1, 3.618823710668649:
1, 3.2855272104730546: 1, 1.9420639551471635: 1, 2.607031606978331: 1, 1.9294
71919319219: 1, 6.231458113000359: 1, 2.2072829605597226: 1, 13.7748613889426
98: 1, 6.332656985886861: 1, 2.6755205811087523: 1, 26.44199712589457: 1, 9.3
1583775509299: 1, 3.0439049409545853: 1, 42.88340881237113: 1, 7.089003344086
994: 1, 2.4687583319773756: 1, 3.045272839908716: 1, 5.841586644727192: 1, 2.
7624648947345998: 1, 7.460605146827971: 1, 3.3375593455338963: 1, 1.675330624
8575996: 1, 3.690976864411185: 1, 2.820646203058488: 1, 4.575629520600837: 1,
4.480515653752695: 1, 1.8885642470837896: 1, 3.9681372418939342: 1, 2.8871198
548599: 1, 10.752822681557209: 1, 2.653151740621285: 1, 1.5165405835285173:
1, 1.9168968056754474: 1, 2.8145310119162787: 1, 3.423614371354245: 1, 2.5767
04478698766: 1, 1.63198218255209: 1, 4.864422769857334: 1, 2.326625146099156:
1, 2.1137445558174286: 1, 2.7431405152594053: 1, 2.6167321003902053: 1, 3.190
580588068651: 1, 3.9183209086470288: 1, 15.180647436464854: 1, 1.797638237433
0078: 1, 1.8200813758920955: 1, 2.2973024049377404: 1, 8.187496323398992: 1,
6.462364521890761: 1, 1.897812659901512: 1, 2.4834284579344703: 1, 1.96189156
13620178: 1, 4.559984019337507: 1, 7.603398890052958: 1, 5.448275823600034:
1, 2.932206887110376: 1, 9.979063270847645: 1, 3.1237350169178333: 1, 3.49074
7197030636: 1, 3.036255594614365: 1, 2.086350659079422: 1, 10.38901395081523:
1, 5.045307425650874: 1, 2.6913892327819737: 1, 2.2300962587991298: 1, 8.5661
61947130142: 1, 1.6123041447377713: 1, 2.8669684284371884: 1, 4.5023786219971
17: 1, 3.562607693124001: 1, 2.539537568993617: 1, 2.8526091252897214: 1, 1.6
001794938982896: 1, 2.647582051618455: 1, 2.889943860364353: 1, 2.05245357845
46165: 1, 3.6188446511659436: 1, 2.388795440774246: 1, 3.5905179531343006: 1,
2.4562151744423937: 1, 3.5142894073681745: 1, 1.659906538744933: 1, 2.7239594
0307209: 1, 11.050521112900414: 1, 2.0997652440999466: 1, 6.188237139522068:
1, 2.379530504156782: 1, 6.259359300593995: 1, 5.55065363001669: 1, 4.2213870
8266998: 1, 1.8307482438892013: 1, 3.8557029765711173: 1, 1.7743548402488245:
1, 2.9034078557024343: 1, 1.7040590541347829: 1, 3.9710250362948347: 1, 4.684
008421533939: 1, 2.5099539845149303: 1, 14.914419803363362: 1, 1.887752120580
5725: 1, 6.421992598824635: 1, 3.350258824670695: 1, 2.0842711812286883: 1,
4.783263132164845: 1, 10.00610159966917: 1, 2.244312468640647: 1, 1.885746891
960759: 1, 3.4328768396103815: 1, 2.60054084861378: 1, 2.1036321655690156: 1,
2.5749875352354: 1, 2.5617256475711017: 1, 4.042274551540452: 1, 4.2691215084
94302: 1, 2.9788810026785852: 1, 3.761109100101075: 1, 2.967111657865611: 1,
2.0120516515890277: 1, 2.0004859798389667: 1, 1.7676483952311153: 1, 4.532946
944513191: 1, 2.3629558888934077: 1, 7.854031285900796: 1, 2.629659275004642:
1, 7.321173003372005: 1, 3.374181139231516: 1, 11.17314243718298: 1, 6.132721
232088581: 1, 5.328666083829833: 1, 2.612557756805462: 1, 2.3846269123945705:
1, 1.9140175148394512: 1, 2.352450407143129: 1, 7.83358200279887: 1, 5.689085
74748614: 1, 4.232473847136056: 1, 4.2492113541710985: 1, 3.2343025789667: 1,
1.9836600438745289: 1, 2.754264987582881: 1, 2.5694870850608793: 1, 2.3492738
603836707: 1, 4.014408224598319: 1, 3.1010797531097642: 1, 3.617893054425403:
1, 12.088673478515181: 1, 3.226556555412606: 1, 3.162254314116474: 1, 2.62268
0613495348: 1, 42.95617984805815: 1, 8.596036301328503: 1, 6.59834784636005:
1, 1.9093494854187048: 1, 4.338825919649745: 1, 2.6249805124035905: 1, 3.0356
378867193805: 1, 2.453599712105638: 1, 2.8523574409585803: 1, 2.7471874382780
15: 1, 2.219339010626872: 1, 2.3231773925633066: 1, 6.679387652033693: 1, 4.2
06894593580564: 1, 2.781830053966402: 1, 4.153156088069359: 1, 3.478348083736
535: 1, 2.468747109742374: 1, 43.59655643565727: 1, 9.654417262900132: 1, 3.7
05206253043031: 1, 7.065216391930883: 1, 9.573230982718679: 1, 3.148777440665
253: 1, 3.4401105903558546: 1, 1.6690290850708638: 1, 2.7417791754922485: 1,
1.8238342286518656: 1, 16.538823518628945: 1, 3.2961907208991574: 1, 2.824996

3721120923: 1, 1.8161423480170633: 1, 4.893363693340226: 1, 8.83621891898830
5: 1, 4.8828357560061315: 1, 5.857974325656711: 1, 14.456916680876736: 1, 1.6
81930672486427: 1, 27.195728135679275: 1, 10.369753787007916: 1, 12.626721489
585822: 1, 6.11586012316449: 1, 2.589591083446514: 1, 2.9806059459587364: 1,
2.892143298931239: 1, 4.641668163397621: 1, 3.385943997154992: 1, 2.789247851
6466035: 1, 1.760840962865768: 1, 3.4177812206177154: 1, 1.7503568877370872:
1, 2.0443313983986986: 1, 11.969008407626497: 1, 2.160671699572256: 1, 4.9752
650896024715: 1, 1.7959354641226042: 1, 4.798093116562781: 1, 2.1830982178807
026: 1, 5.988786099105941: 1, 7.642933343840392: 1, 5.417415440387894: 1, 20.
312634841841177: 1, 2.1721995490422135: 1, 3.0189568710385792: 1, 2.446870374
7047966: 1, 2.588580427210513: 1, 8.911419190614888: 1, 1.8469855840369123:
1, 4.735398998494495: 1, 2.3805321536054307: 1, 3.6234330336239: 1, 2.0936245
475422046: 1, 1.7135730528171917: 1, 3.5967294500480445: 1, 12.9366235483145
5: 1, 2.223690384096062: 1, 2.2130789600943395: 1, 1.600526609453593: 1, 2.88
6815199996808: 1, 8.31336558682373: 1, 5.705516219506326: 1, 3.44918080207820
6: 1, 6.297183141995087: 1, 5.543282492425951: 1, 1.6756313610903921: 1, 4.33
99954775386895: 1, 4.8892463691231605: 1, 2.8342580247559774: 1, 2.9407746580
917378: 1, 1.9124571144894964: 1, 5.520965499809: 1, 4.649407239355673: 1, 6.
251225768016287: 1, 3.0537995512025637: 1, 4.15401077570679: 1, 2.23174368688
7556: 1, 8.071696248260128: 1, 1.8403102951713186: 1, 3.623669180488726: 1,
3.646107351733286: 1, 2.510705864718991: 1, 2.4613582118011466: 1, 5.01889317
8916306: 1, 5.243285112276606: 1, 2.997125466825204: 1, 1.831251914071491: 1,
5.922618117522384: 1, 3.3302510126055553: 1, 4.662327308743319: 1, 2.02751094
0747914: 1, 1.86429668635712: 1, 2.2268319205188734: 1, 1.4530081854179577:
1, 5.150044637892661: 1, 2.601704010657499: 1, 3.4860210143905213: 1, 2.71829
5977910119: 1, 4.8929241105430465: 1, 10.26697277861687: 1, 7.35747964783368
2: 1, 2.8947837422007052: 1, 7.652645943913056: 1, 3.8103785654228806: 1, 2.1
268192024877295: 1, 6.811665733664807: 1, 2.934784377470855: 1, 1.87759711941
39003: 1, 5.235038940110222: 1, 4.163170798886182: 1, 3.067819477634101: 1,
1.850125500436155: 1, 2.287234088069722: 1, 4.383540786360049: 1, 1.503934288
6495448: 1, 3.783950864618804: 1, 2.9910643466492233: 1, 2.092345929355786:
1, 2.486648629665212: 1, 3.4096845223845462: 1, 13.523776074708573: 1, 2.4792
287988008788: 1, 2.15415956870503: 1, 2.6432457691421734: 1, 1.80255781382157
45: 1, 3.672697720033308: 1, 7.878850492405984: 1, 1.6422258275040018: 1, 2.7
72341007630594: 1, 5.962180852076796: 1, 1.8142191934005218: 1, 3.44854743234
64995: 1, 1.8526169206602012: 1, 2.800743777119989: 1, 1.990059995863347: 1,
4.171850153931126: 1, 2.883590405036158: 1, 3.3768099670059164: 1, 3.80220108
91936397: 1, 12.916696046626877: 1, 2.004223976091398: 1, 5.139099337028789:
1, 8.342110653942727: 1, 2.4164398340221616: 1, 2.601486068371122: 1, 6.59662
0140129727: 1, 13.480253256227599: 1, 4.1142582448594975: 1, 1.72318620012686
54: 1, 3.4856897365927595: 1, 1.9372805263224868: 1, 2.1500441582890075: 1,
3.772624978105257: 1, 1.737935325055005: 1, 6.926725288133086: 1, 1.440705507
7782187: 1, 1.897252638951347: 1, 4.044042517400201: 1, 4.492395491928287: 1,
13.267759086522725: 1, 3.6908760528830284: 1, 4.845261529711848: 1, 9.8932283
14596305: 1, 3.3978162309180124: 1, 1.6431815225063529: 1, 16.30128301177949:
1, 7.631742426795637: 1, 5.952209499407252: 1, 5.121477646202761: 1, 3.591411
7866596045: 1, 8.659029505312596: 1, 3.7708544253373324: 1, 4.05326568592402
2: 1, 2.5417525438927124: 1, 1.622058880917777: 1, 1.8898049256071234: 1, 4.1
60597810041018: 1, 1.59970895895516: 1, 14.20214194018312: 1, 2.0809850972653
834: 1, 1.9717525966307556: 1, 2.517012519749415: 1, 2.251830134754732: 1, 6.
0354313408650295: 1, 4.16314419519584: 1, 3.2143362545238707: 1, 3.9588289960
708063: 1, 2.0007106836564543: 1, 2.4298344852740428: 1, 3.2530103277973845:
1, 1.942965553518278: 1, 4.340362129955771: 1, 15.905493758060553: 1, 4.16283
7241146167: 1, 28.899989594857093: 1, 2.255618958878644: 1, 11.27550336899370
7: 1, 1.6822403459670043: 1, 1.7357113697039217: 1, 6.284818669306497: 1, 7.1
05335983862956: 1, 8.897086286862764: 1, 5.453678007785682: 1, 3.222342349021
73: 1, 2.954806269755594: 1, 2.088276367939681: 1, 1.8321783656387973: 1, 6.0

88900743951798: 1, 2.128651547088278: 1, 2.4760377374779647: 1, 2.38262825960
06636: 1, 3.421006536568366: 1, 2.206231987080983: 1, 2.1199843337129214: 1,
12.480765064532687: 1, 1.5071845164894222: 1, 9.537401665837724: 1, 2.2037808
735390505: 1, 1.5687839292823125: 1, 4.132966152923559: 1, 6.698024801554751:
1, 6.689504840975342: 1, 4.8505198756916315: 1, 3.4343755982706203: 1, 2.2048
87076209375: 1, 5.023144804873449: 1, 1.4524731919615461: 1, 2.21947200907368
1: 1, 2.50189905229746: 1, 2.5207749812895752: 1, 3.3954028715890736: 1, 30.4
65980158699526: 1, 4.267830950260638: 1, 3.316060700646671: 1, 2.035397507166
1323: 1, 21.293271277904655: 1, 2.1222207055351: 1, 7.344764361347969: 1, 2.2
175291535170976: 1, 6.185285772721554: 1, 3.989481948635541: 1, 6.14022015720
9652: 1, 3.016295254261977: 1, 2.539874826044705: 1, 2.2553236628432543: 1,
1.5090126347763988: 1, 4.979381554965637: 1, 1.5226509773920947: 1, 4.8208763
3638707: 1, 10.360430690226776: 1, 1.6624415922912297: 1, 1.7304504598824357:
1, 6.864265028553656: 1, 1.9351653294651447: 1, 6.864494049131551: 1, 5.15501
4436396833: 1, 4.784993844650885: 1, 3.1833274553614657: 1, 2.65548915973617
9: 1, 5.4719647559541: 1, 1.734389320402061: 1, 2.8897243567545687: 1, 7.6153
60217490725: 1, 3.433054429259914: 1, 6.31148688275035: 1, 11.21642707005905
3: 1, 4.030727279893893: 1, 75.02767377742727: 1, 3.9780556148146835: 1, 1.83
32388183315256: 1, 4.601327015048379: 1, 6.165664972041748: 1, 7.839785097973
319: 1, 1.8435891247367329: 1, 3.191851951701737: 1, 5.182818937769398: 1, 3.
231430301358783: 1, 4.720880284844657: 1, 3.8030367907560265: 1, 2.1114993606
69955: 1, 3.969166986676425: 1, 1.822587946699275: 1, 5.959610529677942: 1, 1
6.976656535308205: 1, 1.9580442770072242: 1, 2.4378192214507663: 1, 12.154194
587828028: 1, 1.9707235081718013: 1, 3.1851950500075166: 1, 8.62499339135233
4: 1, 6.939704213019909: 1, 4.196928067046928: 1, 2.0623131939004455: 1, 3.67
19721751232606: 1, 2.5613141822282044: 1, 2.4148618701429676: 1, 1.9010312489
149501: 1, 2.509712282719133: 1, 1.6413900792470189: 1, 4.972131921419455: 1,
4.6269420426878485: 1, 3.699998628777599: 1, 5.226722090391322: 1, 2.78470310
4626841: 1, 1.6262420078554987: 1, 3.5324149777923233: 1, 3.1275606492874273:
1, 7.6387125336489: 1, 2.482994040640171: 1, 5.181615116301417: 1, 1.33683688
22504062: 1, 3.8123479150592514: 1, 1.841972842994926: 1, 1.6858072514865405:
1, 4.1162783962943115: 1, 2.859898616086609: 1, 1.8582414893833572: 1, 14.005
007705864314: 1, 4.791800281448809: 1, 3.298876119851661: 1, 10.3737065172991
12: 1, 3.3096921017835332: 1, 1.8871158610904255: 1, 2.5516504384444887: 1,
3.8242182920720205: 1, 6.946602584450533: 1, 2.8431417944509505: 1, 3.2247678
74492327: 1, 5.168761557808404: 1, 4.185694743254987: 1, 3.6948673843991346:
1, 3.026051918945015: 1, 2.786388392462687: 1, 1.6332126899506716: 1, 2.97363
21916132606: 1, 2.7889277430044745: 1, 7.438301774365366: 1, 1.87177449785267
38: 1, 2.7849479631596714: 1, 1.553872653152236: 1, 4.072697147400948: 1, 3.4
452156788225348: 1, 5.304670217938781: 1, 20.822010922999805: 1, 4.0710318012
67271: 1, 3.0688730663115855: 1, 2.8411958981068506: 1, 1.9746166592927823:
1, 1.8876729835307926: 1, 4.308222779236809: 1, 2.076629070965801: 1, 2.36230
3851576785: 1, 4.7589760911367565: 1, 3.920343765541127: 1, 8.50771859153660
3: 1, 3.3675634760136104: 1, 2.612831537896915: 1, 6.028183325379461: 1, 3.49
1632681871505: 1, 2.094101227997449: 1, 25.200937550920084: 1, 4.826952410805
735: 1, 3.9981570199486964: 1, 5.468215890983851: 1, 2.0118587196048923: 1,
2.028317223333172: 1, 1.5673524448027556: 1, 1.6990365912777396: 1, 5.2386980
06817484: 1, 2.4941399041214813: 1, 1.9184643913692923: 1, 13.31644819935154
1: 1, 1.879155030838824: 1, 3.056414147166955: 1, 3.7125300261491754: 1, 1.89
9343849679587: 1, 2.810432517150507: 1, 8.38418565922694: 1, 2.61442209527053
3: 1, 2.545497231183496: 1, 3.1432735797079063: 1, 5.3324055602474125: 1, 4.8
87842855397767: 1, 3.5102041826722727: 1, 9.001652536886779: 1, 2.06123742740
8988: 1, 4.188942152239141: 1, 6.176185323562181: 1, 2.2701646854866016: 1,
2.5779562076815656: 1, 1.9329293742928928: 1, 4.344713544987992: 1, 7.8650516
39041634: 1, 2.1089303802803174: 1, 4.500543617890717: 1, 5.314884457692689:
1, 23.379441412121576: 1, 2.902168047331365: 1, 2.0191977743327034: 1, 4.1663
43836843898: 1, 3.2066468690922156: 1, 3.4455319817616945: 1, 2.2641367754110

23: 1, 2.4973874032986565: 1, 1.7498835837706879: 1, 5.501033262829183: 1, 3.
0071006761954497: 1, 3.1394445188766213: 1, 4.924739560423898: 1, 2.686603794
380926: 1, 3.7011646563383573: 1, 24.26232047985381: 1, 2.3906608758594583:
1, 7.4440777836866125: 1, 1.82252368891717: 1, 5.71528941086216: 1, 3.8714315
085864954: 1, 1.415272641509877: 1, 3.2022182121840412: 1, 2.850326556714057
4: 1, 4.026231651931985: 1, 1.9053346460799423: 1, 13.188551463060907: 1, 3.4
719078199492395: 1, 2.2387854081783995: 1, 2.0430568086458862: 1, 3.618235975
646787: 1, 5.430134038244304: 1, 2.9047760742301874: 1, 2.6668410381721124:
1, 1.623817096111572: 1, 2.504704387890617: 1, 8.751314972965066: 1, 3.102970
880250918: 1, 6.1981069461838745: 1, 5.854818637550686: 1, 4.035858201675913:
1, 3.6475856190883595: 1, 3.1848365175280464: 1, 1.5833319303863556: 1, 2.573
2091641651698: 1, 1.8695997435754843: 1, 1.9684354492675313: 1, 4.02713200163
0016: 1, 2.7129788576032037: 1, 6.232802372868596: 1, 2.900081690427856: 1,
5.833995643456605: 1, 6.196910094926237: 1, 3.869537962911673: 1, 1.605824345
2521064: 1, 3.882259158694874: 1, 7.614624361989049: 1, 2.037685295463695: 1,
2.5720995501432564: 1, 1.850775751775852: 1, 4.170605778495417: 1, 3.84286209
6037371: 1, 2.3596030396819385: 1, 2.514896534814191: 1, 1.9921082222771744:
1, 2.945278573629607: 1, 14.974997153558819: 1, 1.9161682072411246: 1, 19.473
43635530853: 1, 10.269387755055778: 1, 2.362126884859928: 1, 1.80523317236588
2: 1, 6.2415092163168335: 1, 1.570008212279329: 1, 4.568991153950939: 1, 2.13
52155554013543: 1, 3.58692427477208: 1, 4.0909393202607065: 1, 2.510748773067
057: 1, 1.374201436448708: 1, 1.9435842462866688: 1, 5.8894656962613725: 1,
3.4857942712312187: 1, 4.253224487468219: 1, 8.459266846842354: 1, 11.6555612
92905805: 1, 3.2582245568916273: 1, 1.9208726665217248: 1, 2.241855590684143
5: 1, 2.435334481241179: 1, 7.61846813114237: 1, 2.176778563655155: 1, 2.2604
43055815107: 1, 20.09720562970658: 1, 1.9684929482247189: 1, 2.53664519324547
17: 1, 19.97830583065469: 1, 2.3599855490740187: 1, 2.869752568666794: 1, 3.5
870218672375334: 1, 8.44924854062749: 1, 5.686588027683139: 1, 3.711636822405
4566: 1, 12.399819129595858: 1, 3.7422421839081816: 1, 1.908977329128666: 1,
1.4594316895898: 1, 2.81739779994074: 1, 8.53063663952826: 1, 7.0509148919428
58: 1, 17.98745251744967: 1, 3.515371554222939: 1, 2.3107968284609735: 1, 4.0
38479159143371: 1, 3.999149834443721: 1, 3.6097547655418367: 1, 2.60521625468
2312: 1, 1.9797199654601854: 1, 1.5150338806114623: 1, 3.161269054833258: 1,
13.297820092374241: 1, 4.858960049776935: 1, 1.812117048882859: 1, 1.74953783
95032888: 1, 3.076769198279206: 1, 5.717085657967633: 1, 7.427328426164528:
1, 3.450253354754804: 1, 1.7659089158250045: 1, 3.4274514530391813: 1, 3.9529
494819105304: 1, 2.3656300861015285: 1, 4.9496742987504: 1, 1.827618440856378
3: 1, 4.741141181754933: 1, 2.465306650934947: 1, 9.833989142685683: 1, 14.89
1099998901883: 1, 5.6139064112086166: 1, 6.955670695515072: 1, 2.993347498013
375: 1, 3.42855283845575: 1, 4.827054013472659: 1, 10.576433073527076: 1, 2.1
27047045314446: 1, 1.8362165820113612: 1, 6.4047472306085: 1, 4.4923221463452
325: 1, 2.0942129661583313: 1, 4.928207434362428: 1, 2.8480932384906: 1, 3.80
4873159526005: 1, 5.048635913587896: 1, 2.4845892123830087: 1, 4.643704514205
132: 1, 2.243929473134427: 1, 3.5706575941663377: 1, 4.2614384773778315: 1,
3.974782590721366: 1, 2.3225271098889575: 1, 6.627991782381249: 1, 2.52851761
38728676: 1, 11.58790089013357: 1, 2.8354194952001612: 1, 2.8650756310241885:
1, 7.651094187599988: 1, 1.8262557309732677: 1, 5.846156125417457: 1, 5.74868
0904562655: 1, 4.534873786424742: 1, 12.258399933662613: 1, 3.59771117348457
8: 1, 4.021877168885358: 1, 2.1228945062835813: 1, 14.073660569652539: 1, 1.8
577782087115748: 1, 16.444143262567355: 1, 2.6820606231735455: 1, 2.403005657
3109833: 1, 7.267673885005557: 1, 1.8198729458400644: 1, 12.6675569407327: 1,
4.622126839551702: 1, 6.302052186511282: 1, 5.495884017907718: 1, 2.482909612
3872496: 1, 1.7642821234036419: 1, 1.6396218090293722: 1, 6.09685494534356:
1, 1.8542596651916894: 1, 4.174628877421498: 1, 6.602120841486333: 1, 2.62448
2656777041: 1, 4.263501607208065: 1, 6.537984847255059: 1, 3.979873369692053:
1, 2.347710015131379: 1, 6.471942537746074: 1, 17.64794113676253: 1, 4.221661
991646648: 1, 5.577754427915154: 1, 3.2839077771053393: 1, 1.763855972754699

6: 1, 2.9494866666634323: 1, 18.1670850861936: 1, 4.25673303402591: 1, 2.5690153484720044: 1, 6.751625767415894: 1, 2.271279990467608: 1, 9.153755461724295: 1, 2.6005391007719867: 1, 4.996935808628294: 1, 7.7012854510185145: 1, 5.075177370051654: 1, 5.188835021213487: 1, 1.7372034336309987: 1, 3.2380331908338613: 1, 19.77801803094863: 1, 3.2587068347438573: 1, 2.447723663722765: 1, 4.401238164750017: 1, 1.5752425962114636: 1, 5.951816452888703: 1, 1.5885650359157912: 1, 19.53134382393397: 1, 4.249489446187577: 1, 5.137186857786092: 1, 3.3744023724219336: 1, 1.872015683326776: 1, 5.115604459160058: 1, 2.950606028151674: 1, 5.409939820731453: 1, 6.127393331098512: 1, 4.66748478497056: 1, 2.314073902282753: 1, 4.933103784506552: 1, 2.0708600444528944: 1, 3.865975106396174: 1, 2.277544927165817: 1, 7.2559079415255825: 1, 1.912794144010118: 1, 3.04518093078501: 1, 2.282559681391224: 1, 2.4517801114915576: 1, 3.8879279624650898: 1, 1.5773730587333823: 1, 28.922565920355666: 1, 4.026888631043744: 1, 2.845485740980612: 1, 1.5694367463968149: 1, 2.681265191302349: 1, 2.826669587536947: 1, 2.071344376187209: 1, 5.724991248911119: 1, 3.302688391238629: 1, 2.9197464989144604: 1, 4.773852790994331: 1, 1.7122649206791012: 1, 3.788553441381333: 1, 3.5002065944818845: 1, 2.296596664758766: 1, 2.5414419716618903: 1, 2.4708919807012832: 1, 3.624424413351881: 1, 1.9982669235922803: 1, 2.1736007140002487: 1, 16.19145877014432: 1, 9.042755770693011: 1, 3.253186781121098: 1, 3.346263414310937: 1, 1.7453065162058115: 1, 2.546139410031963: 1, 4.134913354432282: 1, 2.371396852991632: 1, 3.6275918941844023: 1, 2.1843113932250136: 1, 2.630758219323436: 1, 1.6959648710735826: 1, 1.9909682102130333: 1, 6.652614150436892: 1, 3.271017991073681: 1, 2.7299302366183533: 1, 2.9435904310798504: 1, 2.2803094611279535: 1, 2.7026703794327678: 1, 6.098723788898403: 1, 2.3693206476989306: 1, 9.067941454294038: 1, 1.9653958594484922: 1, 2.2386806629351867: 1, 2.304192369126703: 1, 5.3813961121816725: 1, 3.577945958356392: 1, 2.9421219298008316: 1, 2.9879604094379704: 1, 2.188844475058741: 1, 2.3911572518622526: 1, 2.429333027605821: 1, 2.3530690220785173: 1, 1.625686142916335: 1, 2.878673165510142: 1, 2.493072896286488: 1, 3.4236881446335548: 1, 6.984553300378419: 1, 1.898556420374379: 1, 3.838566503431451 3: 1, 4.389471891989953: 1, 3.481024461389237: 1, 2.4217578660119994: 1, 1.9657216478138637: 1, 1.7552848985858567: 1, 15.819610296465207: 1, 1.8269230272395216: 1, 2.2516552671576613: 1, 13.500109827252752: 1, 2.4822502679324034: 1, 1.949675141677786: 1, 3.775418559110457: 1, 11.758112405748621: 1, 7.15819821142912: 1, 2.9648901800822287: 1, 4.602841600197154: 1, 5.13921588343521: 1, 4.714088751951482: 1, 1.3718637952275587: 1, 3.0831964994185324: 1, 2.6567896090612906: 1, 2.196617418734289: 1, 1.980738083604364: 1, 3.2095861990390966: 1, 2.462362986970756: 1, 3.2799800709313343: 1, 2.564601265944248: 1, 2.5663968765804137: 1, 2.9738944203749633: 1, 2.7578049605379986: 1, 2.1169478500335432: 1, 1.6447102913472371: 1, 8.925015063823306: 1, 4.411036675441694: 1, 6.209181244101612: 1, 1.463091943564376: 1, 2.620665703342818: 1, 1.9798695953113619: 1, 2.9732004335827806: 1, 3.3073349653891686: 1, 4.632316348241277: 1, 5.94924798396721: 1, 7.101352328401161: 1, 8.747411641452585: 1, 9.240529821760274: 1, 10.495906148407014: 1, 11.057251693406043: 1, 12.475401799397273: 1, 13.182286517065448: 1, 14.196108885588197: 1, 15.043422741264207: 1, 5.13198095888521: 1, 1.700516178674802: 1, 3.5126313678892975: 1, 9.0400511159234: 1, 2.2009077701722055: 1, 7.903363033872549: 1, 4.149038538276897: 1, 2.627459623735224: 1, 5.162684273662931: 1, 2.9617782526855705: 1, 5.43945791840605: 1, 3.9585758310303154: 1, 2.2109854390197583: 1, 2.9749962293851975: 1, 1.789116117719061: 1, 6.933965076133327: 1, 2.206910054567968: 1, 1.687923349914351: 1, 6.791812937674055: 1, 2.839350018882157: 1, 7.094599055990259: 1, 3.4956184587007098: 1, 3.9108324656095426: 1, 2.996477649383477: 1, 2.165098219682119: 1, 2.777713227226845: 1, 1.5579282981702758: 1, 3.784767110644221: 1, 2.091103413455378: 1, 6.883661916608672: 1, 13.167851248111276: 1, 4.383034520502519: 1, 3.2345374121481707: 1, 2.6169161415597513: 1, 2.8903766818114605: 1, 3.046053991884093: 1, 2.687236225135476: 1, 6.930630626580398: 1, 4.59513327985952: 1, 2.646360834128832: 1, 8.296311059624202: 1, 2.0198963074

67037: 1, 1.6793767401248807: 1, 2.5746951213036113: 1, 7.27668962835588: 1,
5.317109974061281: 1, 2.582678763919355: 1, 2.8054096987697448: 1, 3.72084857
9830033: 1, 1.9369125594919305: 1, 2.289300628729376: 1, 2.1102751173111467:
1, 2.2137531568740205: 1, 13.454153127185542: 1, 12.138753253889998: 1, 6.756
518529430509: 1, 1.7615579819048348: 1, 1.9101579424215214: 1, 2.792759122557
631: 1, 8.755160454345114: 1, 2.264247534598218: 1, 2.1756304287308166: 1, 6.
584750154816901: 1, 4.599474458133102: 1, 3.100556306761604: 1, 2.55248276524
21433: 1, 2.262753023377287: 1, 3.3154484160624387: 1, 1.8273127195756458: 1,
2.4879038547857677: 1, 2.0604354140890746: 1, 7.619065468121456: 1, 2.4069054
322047383: 1, 2.465882446376798: 1, 2.8584906966758132: 1, 9.97159571691992:
1, 5.095807619039139: 1, 2.8631842099394578: 1, 3.6935053051064526: 1, 3.1740
441559180086: 1, 5.692052171508692: 1, 5.641463667414833: 1, 2.69733652997943
27: 1, 3.9236648214127863: 1, 2.476892621575446: 1, 4.04222444743967: 1, 1.76
3000682786073: 1, 2.4523310583324314: 1, 5.067011192701428: 1, 2.586923624073
2327: 1, 9.917912193645954: 1, 10.775089725750659: 1, 2.1410535826044303: 1,
1.9231345779665665: 1, 1.801480919605375: 1, 1.5894903980711714: 1, 2.8691261
155422874: 1, 8.369350725549387: 1, 4.336130642460944: 1, 2.6450561047422747:
1, 4.825877805883125: 1, 5.864089610006709: 1, 4.316250855199263: 1, 3.240567
821667682: 1, 2.4097066100935667: 1, 1.872273331618063: 1, 6.108126896544963:
1, 1.9039855077639236: 1, 2.499577792749535: 1, 3.1623041371811724: 1, 4.9513
55820140233: 1, 5.533645492796866: 1, 3.4340385563661133: 1, 1.96322229508642
7: 1, 4.076444764640784: 1, 2.109123739940283: 1, 3.4488602393200956: 1, 5.28
87002464244475: 1, 3.288294181007304: 1, 3.780407702057232: 1, 2.377531452113
9728: 1, 1.5359799809141528: 1, 1.713370011777445: 1, 2.507247799489389: 1,
1.47412112300255: 1, 1.9874279039901055: 1, 2.1824783416589306: 1, 1.65266549
63596628: 1, 8.503495818351837: 1, 4.824704506738869: 1, 6.72466591146008: 1,
4.51134901818722: 1, 5.899089815442346: 1, 3.467545325071429: 1, 2.5027830820
805574: 1, 17.244052893980378: 1, 4.7521689133461456: 1, 4.799532004710791:
1, 1.833338313652155: 1, 1.6775868467633002: 1, 2.56627287851381: 1, 16.28410
7866392965: 1, 4.997305210151934: 1, 9.644678593396577: 1, 1.604757099431190
4: 1, 5.938829504924511: 1, 1.8526719559840608: 1, 1.8400454689033652: 1, 5.2
784113174404315: 1, 1.8664901876199345: 1, 3.8787529887904943: 1, 3.046219927
534824: 1, 2.607393418006754: 1, 1.7296198517598818: 1, 5.771826020365449: 1,
2.402246526299491: 1, 2.1031296364100123: 1, 1.6173297019890165: 1, 3.7417004
511428256: 1, 1.9977640898726643: 1, 28.318659513772523: 1, 6.562633840866114
5: 1, 3.8063496038502196: 1, 4.1589388668503515: 1, 3.043381083397921: 1, 16.
49172006965076: 1, 2.8003294457011556: 1, 10.82531002598606: 1, 1.79918102800
39755: 1, 5.424506082022846: 1, 2.025427600022481: 1, 4.121810921296079: 1, 1
1.649286463067654: 1, 1.9227566444329696: 1, 7.41029231411841: 1, 1.789387158
1958603: 1, 14.080452880424492: 1, 5.2676251374925895: 1, 3.624945439869522:
1, 5.256546225371468: 1, 4.125604487560789: 1, 3.0488913512086278: 1, 2.31279
8217023821: 1, 2.339744459012057: 1, 2.7213036586146124: 1, 1.879660121615761
3: 1, 5.187958010354958: 1, 4.445925849536657: 1, 7.530213985977378: 1, 12.66
6326822038116: 1, 3.68054527227273: 1, 6.558639246676507: 1, 1.99044060679752
49: 1, 8.694026201056603: 1, 6.818997032810219: 1, 21.1795085733317: 1, 4.870
97269110922: 1, 3.6775385778789524: 1, 1.5316487451754013: 1, 2.1833111790768
998: 1, 1.9650509292388612: 1, 3.3675910447925657: 1, 6.856472864210376: 1,
2.4411003665161517: 1, 3.309279733778414: 1, 2.788824784746297: 1, 4.14207856
2054752: 1, 2.1281670333738867: 1, 9.769995616996257: 1, 2.0963310098739028:
1, 7.535451294495945: 1, 5.6043008324387: 1, 4.384522681975727: 1, 5.82837338
1338147: 1, 3.017656197721185: 1, 2.356166833724648: 1, 1.8287112836743002:
1, 3.8850978724812424: 1, 2.3552280103466994: 1, 3.3656879906125545: 1, 10.03
5729635342081: 1, 2.545580837607405: 1, 1.3836111420003674: 1, 23.84543836483
675: 1, 1.8705550511808957: 1, 6.034999331363739: 1, 4.008123202387428: 1, 3.
8773475003267714: 1, 3.0742867064230146: 1, 2.8773918181295906: 1, 2.87461041
70567772: 1, 1.752165462524522: 1, 2.3908345021711446: 1, 2.0179457714788267:
1, 7.2993475007898585: 1, 3.3853715500408335: 1, 11.517497570748619: 1, 15.59

4665497289487: 1, 3.0787340391949916: 1, 1.614543129765244: 1, 3.414767557843
2715: 1, 4.170074579711965: 1, 2.24779909346396: 1, 3.796950041987171: 1, 5.6
62858044606314: 1, 3.63059561777124: 1, 4.248610195849911: 1, 2.2753857615606
194: 1, 1.3715293838748783: 1, 16.859547793700642: 1, 2.716738217637511: 1,
6.5451555960889936: 1, 2.790823923092098: 1, 2.6613366797411: 1, 2.6070509728
64233: 1, 25.64009882143423: 1, 8.538606546500947: 1, 2.3266467716361507: 1,
6.261633298699626: 1, 1.744230069360931: 1, 4.818030676729739: 1, 2.689354305
08102: 1, 3.07027720909432: 1, 1.6029915404819397: 1, 2.093888091917285: 1,
4.607558818134754: 1, 1.9495957912840036: 1, 9.76278857363748: 1, 1.526688170
589723: 1, 1.783430322872182: 1, 5.044320269743356: 1, 1.5908154012412257: 1,
7.1465870106214116: 1, 2.636113136020474: 1, 2.78297190565789: 1, 3.150599575
046975: 1, 1.5343846558693526: 1, 3.490791662137747: 1, 3.012706440271437: 1,
2.297030922091316: 1, 4.970313610600604: 1, 5.590141202031139: 1, 2.176905082
8430974: 1, 6.968143851159878: 1, 3.3217209542671133: 1, 2.8313422011124554:
1, 1.7850364180315594: 1, 2.507081289369029: 1, 2.375406508460571: 1, 2.74649
02717028905: 1, 4.395758716459336: 1, 4.016835421588339: 1, 2.16080053977799
8: 1, 1.7565067334413345: 1, 3.6774608086697307: 1, 6.669461361560769: 1, 2.4
65293691915001: 1, 4.695866476496014: 1, 19.564368348928667: 1, 2.10515800711
9211: 1, 1.7114500606329348: 1, 10.999526903449441: 1, 15.38444001163788: 1,
2.7861385790915976: 1, 2.547753573728634: 1, 3.9777521349212934: 1, 1.6282194
333303681: 1, 11.16967612039895: 1, 2.1369642837931866: 1, 1.783858804071535
7: 1, 2.3975920908470463: 1, 5.344801479691459: 1, 7.9074382897908375: 1, 3.1
971023296870804: 1, 4.425711152180727: 1, 2.678834394091847: 1, 5.29385936855
5679: 1, 2.1872629654060143: 1, 3.8988690693193533: 1, 2.1260944936080723: 1,
1.9165036033275635: 1, 2.9246097906823163: 1, 2.0224126203873274: 1, 6.515111
874378879: 1, 2.1792105208636077: 1, 6.46601126072597: 1, 3.752294370682869:
1, 1.8149965209880972: 1, 8.702447095965288: 1, 2.537766596601349: 1, 3.47910
2340258375: 1, 6.576381167174879: 1, 21.648151026600715: 1, 4.10581666841557
8: 1, 4.316272573605037: 1, 3.3908243260199034: 1, 2.653359941550442: 1, 4.24
0352170143445: 1, 5.511483045086973: 1, 2.4099079619102772: 1, 3.130599528981
4965: 1, 4.708842141711368: 1, 10.02782035923884: 1, 1.9562105118150919: 1,
2.604096721227609: 1, 1.5834936131158335: 1, 1.8031395117299611: 1, 5.1678204
06926899: 1, 1.3456543654206863: 1, 2.5647776462111045: 1, 3.133356463108017
7: 1, 5.853214537516301: 1, 2.1814021106463395: 1, 6.183739757700264: 1, 3.50
75552587120535: 1, 2.391927833903915: 1, 1.6867540944991308: 1, 4.89236722571
9789: 1, 2.382138386311348: 1, 2.2922711349575935: 1, 6.663674849513377: 1,
9.881915180877325: 1, 4.431567001707503: 1, 3.571616519782347: 1, 9.520669452
944714: 1, 2.657358974814608: 1, 1.6616941345338128: 1, 2.4465025854733353:
1, 15.381529745411118: 1, 2.24187150812699: 1, 2.5899336345546082: 1, 3.06029
36739869118: 1, 1.587008064933079: 1, 2.6707795168826225: 1, 14.9411354548650
35: 1, 4.5187608318994865: 1, 2.0991215117395474: 1, 2.45953637842025: 1, 1.9
721714016134484: 1, 5.152050079949341: 1, 3.3590806027519395: 1, 5.0181594848
98254: 1, 2.4329182583301: 1, 6.767664630491758: 1, 4.186567934041147: 1, 2.3
975424022924847: 1, 2.529544113575419: 1, 9.206014276897514: 1, 17.4925825845
51133: 1, 12.274050176498298: 1, 1.66947279129352: 1, 1.8964448056514371: 1,
8.278922531475196: 1, 1.9736270589499665: 1, 9.895109373657018: 1, 6.87592958
35762195: 1, 2.7236487872390223: 1, 4.421210490230417: 1, 2.2130704557440826:
1, 3.296783880338561: 1, 2.725487660426448: 1, 5.98222099257557: 1, 1.5865892
80280616: 1, 9.77815722953473: 1, 3.1221252683477334: 1, 7.759414768249878:
1, 4.002004855225609: 1, 3.4924625369994216: 1, 7.81520052451959: 1, 23.93453
6857627506: 1, 3.956305301458589: 1, 3.454640158758535: 1, 2.707567805107524
7: 1, 3.0105808029927728: 1, 7.6487452327453305: 1, 2.516103515710522: 1, 2.7
483523293401757: 1, 2.305333415739269: 1, 8.205555870106654: 1, 4.57659093479
0938: 1, 8.048987282012874: 1, 10.570434773088728: 1, 4.11461473066211: 1, 3.
9703121055755957: 1, 2.749238232746924: 1, 6.8830762978138385: 1, 4.831043894
19603: 1, 3.769812890085318: 1, 2.4086134610464223: 1, 4.867009613104083: 1,
3.753047459656766: 1, 2.4812349580701714: 1, 2.944399333792074: 1, 11.623662

044673916: 1, 11.984310679636629: 1, 1.7136556609338265: 1, 1.624324528961321
3: 1, 1.9879274329449674: 1, 6.345021690571139: 1, 7.369170655934633: 1, 1.80
95012268752078: 1, 5.300298531014174: 1, 20.62767889881292: 1, 3.096094488132
916: 1, 2.6088862804544815: 1, 2.2606087064890623: 1, 1.951501058923907: 1,
2.4538115117474453: 1, 4.680585145659189: 1, 12.352996823758254: 1, 3.469813
39386337: 1, 4.527241358545862: 1, 2.36836108551299: 1, 5.035539252784727: 1,
8.13179752303058: 1, 14.594476719474715: 1, 6.805572855846104: 1, 8.481658201
095703: 1, 4.92980447057057: 1, 3.1300844494854627: 1, 2.566187451275166: 1,
1.7987626614631558: 1, 2.2694821722325105: 1, 1.8030211479616138: 1, 2.59237
8893863728: 1, 1.9926844364959961: 1, 9.757976838220506: 1, 9.92505619901119:
1, 2.471097567225049: 1, 2.750420429443247: 1, 3.7615214923593316: 1, 3.15218
88233181214: 1, 22.550897112791464: 1, 2.0063241943646597: 1, 5.4774466475335
65: 1, 2.9415430428855927: 1, 4.195675817924911: 1, 3.9035375418644778: 1, 2.
4377935278351237: 1, 1.930907581078669: 1, 9.908765702174406: 1, 2.1748006428
38757: 1, 3.0840855952166524: 1, 14.853968626538013: 1, 9.079661236857834: 1,
3.9926156143826206: 1, 4.300585983283756: 1, 7.974076957933502: 1, 10.9952931
87591253: 1, 1.8487845765516495: 1, 1.5416287412682017: 1, 5.947335770251631:
1, 6.323077733620561: 1, 4.438622961118779: 1, 3.895717134820456: 1, 4.227055
3612210096: 1, 2.6097094309062197: 1, 1.9571104084765083: 1, 3.14722123319182
06: 1, 7.515959783435121: 1, 2.0426777488782046: 1, 4.1626154908713975: 1, 2.
222816804184349: 1, 11.552433573764185: 1, 7.165352057652921: 1, 2.5535710630
59525: 1, 1.5769893557147714: 1, 4.864975865812696: 1, 7.117951275365453: 1,
3.765679079984489: 1, 3.892808818490373: 1, 4.5198864058239: 1, 3.1704391714
09114: 1, 6.069849422811045: 1, 1.3721802394941411: 1, 2.268613223862045: 1,
2.3354122232812773: 1, 2.01267761461597: 1, 8.768561939887576: 1, 2.31538976
10844325: 1, 2.5627996802500252: 1, 1.8415452934329146: 1, 3.413131182051708
7: 1, 8.105312209208545: 1, 6.136055614734471: 1, 4.561704595425796: 1, 1.659
3323805453861: 1, 4.489677020703608: 1, 10.528053974954164: 1, 3.368981248900
7935: 1, 2.3292129273208695: 1, 3.0470652862435075: 1, 1.9284304537263064: 1,
2.8354550920534805: 1, 1.7277886305282804: 1, 4.006737745213602: 1, 1.4561715
848935193: 1, 2.596091070507797: 1, 2.4651730270144876: 1, 5.762201155204652:
1, 1.6348847067283143: 1, 7.320324567965952: 1, 1.8128407165901184: 1, 4.3991
9347544119: 1, 5.774521256601544: 1, 5.78747057484951: 1, 3.7945931142850347:
1, 4.341724447189269: 1, 2.7240024692016926: 1, 3.512254188677551: 1, 1.71356
1548051795: 1, 5.176399853577597: 1, 3.3461043704337854: 1, 10.74520511953455
7: 1, 2.9328271851251144: 1, 1.932911483387983: 1, 8.336425964885894: 1, 6.61
2828762514438: 1, 4.20162079844853: 1, 2.5570586328697633: 1, 3.664548313170
3: 1, 2.9747306812067364: 1, 3.039192167647017: 1, 1.9516955286118858: 1, 2.9
35617247873493: 1, 2.808701076779569: 1, 2.143588615768833: 1, 2.461794272941
974: 1, 11.990326966870338: 1, 2.591780357803972: 1, 6.385576111621354: 1, 1.
6151771743492425: 1, 7.280107271766225: 1, 1.8823137743746972: 1, 20.80644331
2088113: 1, 3.963198560888002: 1, 5.579531415508217: 1, 2.7524602146107644:
1, 1.5857828651475225: 1, 2.4640438225549954: 1, 2.9742409465578845: 1, 2.19
2780956771948: 1, 1.834611652963295: 1, 1.7448097477354463: 1, 8.688244001010
004: 1, 4.3433170331356905: 1, 7.526567726185959: 1, 6.932435141726628: 1, 3.
4744949885983205: 1, 3.6220951706297604: 1, 3.3324943126100504: 1, 4.01020015
6926956: 1, 3.792970009027143: 1, 2.173969932244072: 1, 2.848821588483262: 1,
1.5984792881364895: 1, 1.8681030124656508: 1, 1.770071388057198: 1, 4.5252075
04734405: 1, 2.3454243389013105: 1, 4.553592786739993: 1, 3.7169795226889084:
1, 3.603537353207308: 1, 1.9749170441119892: 1, 1.3531277079036401: 1, 3.9255
032173020954: 1, 6.01356953540588: 1, 3.8084962352153546: 1, 7.25188179522366
65: 1, 4.122560922120656: 1, 5.744687313484249: 1, 2.109165728828751: 1, 2.98
07166549428765: 1, 4.444343754759267: 1, 2.3918732907942952: 1, 1.14448249333
29152: 1, 1.846388735082387: 1, 2.3139981839281023: 1, 4.473255373487611: 1,
5.723963768442316: 1, 4.617879469032082: 1, 10.678963786218516: 1, 2.1191912
32141042: 1, 1.967367692806848: 1, 2.760450352282908: 1, 6.36179711925611: 1,
2.976027728270928: 1, 2.241119359502012: 1, 4.362711698148943: 1, 6.734285235

99008: 1, 3.266389827949: 1, 2.8190683858411183: 1, 2.157725347963792: 1, 3.2
08908392174366: 1, 2.2454058152384104: 1, 14.639321049764925: 1, 15.772411126
324368: 1, 2.0520540420999103: 1, 6.466258560048136: 1, 5.88044405954426: 1,
10.223679457946938: 1, 11.641940430338606: 1, 9.53165031161142: 1, 4.0937099
825178915: 1, 1.8919647858411988: 1, 13.02460256378232: 1, 3.211411691611582
7: 1, 7.28643075096268: 1, 2.691722060573532: 1, 1.8072233465329746: 1, 5.778
448659556276: 1, 2.6767180656334313: 1, 8.307737040829126: 1, 2.2138724915753
216: 1, 2.700845775617483: 1, 4.26969725188234: 1, 1.8536356099358668: 1, 3.1
25659063125101: 1, 2.1925756339991094: 1, 1.7713194898622076: 1, 2.3671000812
76292: 1, 2.032770885794564: 1, 3.1722493834813936: 1, 8.470529206488393: 1,
3.199551526309357: 1, 3.236804108567213: 1, 2.511741092042535: 1, 4.14483710
2650358: 1, 7.298667974929802: 1, 3.3940288719868357: 1, 2.0506742026919618:
1, 2.245473256972684: 1, 2.5994315328005353: 1, 2.867048007355335: 1, 3.9345
000015170553: 1, 2.8910305000471372: 1, 24.810923326741758: 1, 2.157978999434
436: 1, 9.34609403690276: 1, 2.9036703544756675: 1, 1.6958141917393197: 1, 2.
4706687923264887: 1, 2.9978580647428044: 1, 2.2533949977145147: 1, 2.72449812
42086745: 1, 2.510909092455495: 1, 4.612962250407596: 1, 1.7774695082660692:
1, 2.8990066741650637: 1, 14.903790733198845: 1, 4.012603750769505: 1, 4.605
264737382467: 1, 10.642421699836817: 1, 1.621860010780181: 1, 1.9158243909390
402: 1, 2.679748389652963: 1, 2.0279322048354995: 1, 2.2677016222393735: 1,
6.7676435227399265: 1, 6.391539390903249: 1, 3.4401442508786295: 1, 3.561475
640746023: 1, 4.44085809891057: 1, 4.252286100809872: 1, 3.9580441712995937:
1, 2.6543952406526223: 1, 7.025862479735541: 1, 9.998075533453417: 1, 2.6121
333513400917: 1, 4.412003659234538: 1, 2.7349225390232887: 1, 4.3015357005973
1: 1, 2.4554813913532203: 1, 2.300370235009997: 1, 7.38244039279944: 1, 5.340
653599086312: 1, 2.793082966567923: 1, 5.501072243044185: 1, 3.48895786713611
14: 1, 6.335776005180718: 1, 2.1284763786063547: 1, 1.9112194089146541: 1, 2.
1390361769962545: 1, 1.9655410548041186: 1, 2.865231346034462: 1, 2.342097848
674342: 1, 4.4516169838649295: 1, 3.603151622353628: 1, 7.513222432231284: 1,
2.0852183083679594: 1, 1.823311859273564: 1, 8.250602454239884: 1, 3.99037368
9387663: 1, 4.24743102922841: 1, 6.086334016700351: 1, 5.124348761232532: 1,
2.488052131731998: 1, 4.37695558228457: 1, 2.438144209846381: 1, 3.977432174
674267: 1, 2.598819507603385: 1, 2.129341373996933: 1, 1.8862368475618516: 1,
2.9962580711324716: 1, 2.1299758275883693: 1, 3.484741201958564: 1, 9.0612054
5550762: 1, 2.7766502493975667: 1, 2.811592676041234: 1, 7.085549647162128:
1, 22.979620953631933: 1, 2.5093992102340863: 1, 5.5659175251684765: 1, 3.60
75976235786396: 1, 3.7260629129891827: 1, 2.271429019719951: 1, 2.50000220476
0118: 1, 3.070528993620109: 1, 3.280796304482083: 1, 4.526045163878579: 1, 2.
78823949823138: 1, 2.9563464941170143: 1, 4.0600275127248775: 1, 10.117110009
991352: 1, 2.740260044089542: 1, 2.370011852921152: 1, 4.215118345643371: 1,
2.5781542922186054: 1, 3.1524989827989027: 1, 4.014600150499196: 1, 3.222927
0388204916: 1, 2.5366806767257994: 1, 3.2239570051204547: 1, 2.64723474391074
78: 1, 1.6664019189472614: 1, 5.7440792012524655: 1, 15.53411950647451: 1, 2.
8081805151029746: 1, 1.920918361906383: 1, 6.892913347214876: 1, 6.5280011389
71367: 1, 3.606547301951173: 1, 1.9832605332736843: 1, 3.789912905201838: 1,
2.623802753685542: 1, 2.001913621867776: 1, 2.2770431153005197: 1, 7.4565161
4811182: 1, 1.9576705227214382: 1, 5.82992856479611: 1, 4.829719738594082: 1,
2.5589783757964937: 1, 1.8666431376803474: 1, 3.0914200562676424: 1, 2.545170
9374207128: 1, 1.9582381537114975: 1, 2.09689171462053: 1, 16.06865939137738:
1, 10.182024681454598: 1, 2.631523512655339: 1, 3.07114930543985: 1, 1.856332
6105446014: 1, 6.047520874027673: 1, 12.668596786376684: 1, 8.14907144084249:
1, 1.832494713417525: 1, 2.060464818402813: 1, 2.430540170462611: 1, 2.812208
0817428303: 1, 6.996292178877164: 1, 2.1074444030552812: 1, 6.21774455346112
1: 1, 3.88013855084656: 1, 5.069512054891327: 1, 4.050513016165459: 1, 2.4591
400161971353: 1, 3.4261884952963926: 1, 1.330005261820008: 1, 2.2588072734841
01: 1, 2.3656872069952577: 1, 1.753675554293117: 1, 3.0363870902624965: 1, 2.
118376924695737: 1, 7.249864180585043: 1, 4.589002730020079: 1, 2.99400102185

96043: 1, 3.7172274166547803: 1, 9.96132541706415: 1, 18.197862836555306: 1,
9.560262056788321: 1, 3.0551811122526225: 1, 2.2621905657307697: 1, 8.904661
902929124: 1, 3.1411152987986064: 1, 2.4549326528587887: 1, 4.55667136890902
3: 1, 5.653137265590166: 1, 2.5110559476014753: 1, 1.387202287326258: 1, 3.43
4188554172258: 1, 2.5048864201638064: 1, 1.540283737718788: 1, 3.232188810142
9126: 1, 1.9466675899786703: 1, 14.329078070885048: 1, 1.4258739302262837: 1,
5.30572159028005: 1, 1.7434219758676872: 1, 5.0026405316082085: 1, 3.79020852
56435716: 1, 10.193052903170116: 1, 2.3528379559870594: 1, 2.229811965551480
8: 1, 8.24929075146979: 1, 2.020709934969147: 1, 6.053738152582204: 1, 3.8367
76030565045: 1, 1.583073804091589: 1, 4.665161066135872: 1, 19.19370447856029
2: 1, 2.292586858923097: 1, 2.415653273387271: 1, 1.9702440630510343: 1, 4.44
83982603081875: 1, 3.3965777036957285: 1, 2.6827838194776046: 1, 2.3068999617
17847: 1, 2.5188441596131805: 1, 2.9541132319802297: 1, 2.1395544028355027:
1, 2.6075441234509036: 1, 11.314263308448137: 1, 2.4277728482876766: 1, 1.73
7713556095749: 1, 1.8150692328444296: 1, 3.803616293088377: 1, 4.604651241606
761: 1, 10.951692631475668: 1, 3.1144161095421268: 1, 1.9442542994977625: 1,
2.1435661231821475: 1, 2.8010574169536646: 1, 1.922981940866543: 1, 2.975747
3943946287: 1, 8.431900852086754: 1, 2.998142380113421: 1, 1.694873124387863
9: 1, 2.5432157999235656: 1, 3.9445531950882957: 1, 2.5490814901279952: 1, 3.
5813919420030174: 1, 8.447716237739845: 1, 2.1471819191294053: 1, 3.742363508
533839: 1, 1.7072090625312515: 1, 3.1990281328401893: 1, 4.194775396941701:
1, 2.1808262848067788: 1, 2.9820877202214655: 1, 1.3198584225472643: 1, 4.86
0148121767686: 1, 2.7061313379631535: 1, 1.4768324754658617: 1, 9.61009185306
4281: 1, 2.2464110466969123: 1, 7.584552110795264: 1, 4.680513827328422: 1,
5.342461627280805: 1, 2.4666713784749206: 1, 3.5073814420839193: 1, 5.320727
818628152: 1, 2.262275931596315: 1, 10.27587850906362: 1, 3.9551193860922518:
1, 7.108293154817793: 1, 2.853594686362071: 1, 14.465380990843602: 1, 5.29702
14174848556: 1, 4.1640461737978125: 1, 6.082748082053775: 1, 18.7216191829146
55: 1, 1.4644616654423805: 1, 2.2604683952789815: 1, 4.850205132697233: 1, 6.
455760552373896: 1, 2.140448279664974: 1, 4.641384779021814: 1, 3.06668564122
57686: 1, 2.6466622052339694: 1, 1.181855584992372: 1, 4.666292782128443: 1,
2.8928750353041432: 1, 9.970553787682674: 1, 2.09057090390195: 1, 3.12225990
2198898: 1, 2.060476865090681: 1, 3.957023787183709: 1, 1.5591148397567263:
1, 1.8142370839895892: 1, 24.415248515268743: 1, 3.4103817564076766: 1, 3.54
48981716525156: 1, 3.8548806778737763: 1, 4.069274241232721: 1, 5.43688349513
6748: 1, 2.2573099108228347: 1, 4.629398128038179: 1, 3.477127521999103: 1, 1
1.994624050326628: 1, 2.8983178273135715: 1, 1.3820793650966392: 1, 4.9035817
29674406: 1, 7.306493281111736: 1, 20.276020898891268: 1, 1.5211786691641644:
1, 5.458131219748779: 1, 4.28792934362826: 1, 2.0727568525618696: 1, 2.707983
7861834513: 1, 25.27877372679093: 1, 2.3434341057258523: 1, 8.76553365696380
4: 1, 1.6687871770726657: 1, 6.876445632850072: 1, 2.718122035156522: 1, 3.06
3641879122796: 1, 4.053201859219337: 1, 6.450218110385911: 1, 3.5398955662639
033: 1, 2.0090745928484024: 1, 2.692788973440056: 1, 2.732212096778096: 1, 5.
4924694082384224: 1, 12.762081391535034: 1, 6.980974049006546: 1, 3.476311470
0395086: 1, 4.978273556831747: 1, 1.827798795719563: 1, 2.9193064391074457:
1, 2.745741776654002: 1, 4.848809263822103: 1, 1.8238331208958234: 1, 5.4287
769191531225: 1, 2.267020518134479: 1, 5.542406032983616: 1, 6.44274160844716
2: 1, 2.070973000426187: 1, 3.366752607522268: 1, 5.666718525506267: 1, 2.404
552986834513: 1, 2.881650392138836: 1, 1.9229302360709788: 1, 2.8225653245029
57: 1, 3.8084016120407003: 1, 2.5734478528523956: 1, 14.30770428598756: 1, 1.
7303778587774128: 1, 3.697211024342982: 1, 27.815536969322537: 1, 10.16482158
1568013: 1, 5.356829828106629: 1, 1.900505404800452: 1, 3.0251630752482956:
1, 6.122029981245044: 1, 2.071060771887837: 1, 6.259632102091399: 1, 4.74714
6773888427: 1, 1.693526937538761: 1, 2.1427651008590183: 1, 3.521678547465100
5: 1, 2.2887956514450387: 1, 1.8418776167621869: 1, 5.657999000004704: 1, 3.7
836950787784245: 1, 1.8908688542358845: 1, 1.5196824662586388: 1, 3.355820396
1718553: 1, 3.17977168881712: 1, 3.1096666339842836: 1, 1.9699305191694447:

1, 5.084693840453464: 1, 1.8242765739431743: 1, 7.297526164189401: 1, 1.8992
928547747658: 1, 2.257417703892777: 1, 5.146479722199412: 1, 20.6969716173213
85: 1, 1.6354130846977633: 1, 3.484583614403359: 1, 3.5926444262795107: 1, 2.
28170569148767: 1, 3.969692137037357: 1, 1.6854927806812414: 1, 3.99090213067
4001: 1, 3.998465299768799: 1, 1.6773409428376684: 1, 5.393460029914606: 1,
2.2356182572963808: 1, 3.7998473245126463: 1, 10.573858455612134: 1, 11.7604
66869061071: 1, 2.57295141484396: 1, 1.9141334134667833: 1, 1.754258724912283
4: 1, 8.092745997951704: 1, 8.66137017737238: 1, 3.3990974758479062: 1, 6.199
128531497673: 1, 4.11030044378057: 1, 2.1356063793689035: 1, 3.63655721189047
34: 1, 2.3243658089483263: 1, 4.085373281113358: 1, 1.8279579515207707: 1, 2
9.155429486327808: 1, 2.211968621206408: 1, 2.512225461877746: 1, 13.03010159
3070036: 1, 4.019675851044592: 1, 3.694195123551894: 1, 9.245841113328163: 1,
2.1279377870645724: 1, 1.640286862857247: 1, 3.972912913520997: 1, 4.50311154
7810322: 1, 5.383550897979799: 1, 2.0553992365659983: 1, 3.80987622638568: 1,
2.812941192061826: 1, 1.34308821007866: 1, 4.562583722113907: 1, 4.4316272364
11617: 1, 7.138074477618883: 1, 10.99058559707097: 1, 2.2699097568169657: 1,
1.6932475641866473: 1, 3.7923914400309178: 1, 2.2520367891587463: 1, 18.3625
92462120823: 1, 6.945472143312681: 1, 2.742753799738578: 1, 4.09000475297287
2: 1, 5.158245117618968: 1, 4.975779874200751: 1, 3.373241912552148: 1, 3.440
43193879696: 1, 1.4422072834390316: 1, 2.49966629193065: 1, 2.17927313041478
2: 1, 5.024622538712915: 1, 7.269041577567982: 1, 6.6526248457178: 1, 3.17822
77786270674: 1, 1.8791994529530016: 1, 5.158324133779504: 1, 5.50887916777472
2: 1, 1.5090991852009095: 1, 24.12938340065644: 1, 4.518796078834541: 1, 3.17
5284064232411: 1, 2.7992575786394336: 1, 3.6568916848648003: 1, 5.43867633142
352: 1, 2.5211323723509986: 1, 3.388546286883722: 1, 4.03716893309083: 1, 2.9
43325468873239: 1, 2.0289002809407717: 1, 2.26908339502715: 1, 2.736021258962
1518: 1, 12.906202268457003: 1, 3.735483369667088: 1, 4.805183049124165: 1,
1.6777228200151477: 1, 1.3231979219090497: 1, 3.5599558592982823: 1, 6.66425
1831570849: 1, 4.802801701866057: 1, 3.166010575859444: 1, 1.859673370888130
8: 1, 2.995973896601714: 1, 3.1904354105749624: 1, 4.384431899021933: 1, 5.96
0984746534961: 1, 6.953257802521562: 1, 7.554844895520602: 1, 2.7933812220408
063: 1, 8.6892615692321: 1, 2.855113851068904: 1, 2.6872088921495862: 1, 6.50
81317536196: 1, 2.525066377701817: 1, 3.5133319724421903: 1, 1.59941647973937
46: 1, 7.218458567732822: 1, 3.7418927127495287: 1, 5.762293810669886: 1, 1.9
44802176659898: 1, 3.0698704920701076: 1, 3.1561379326671206: 1, 11.909264213
393676: 1, 2.6192119621549064: 1, 1.636580675577619: 1, 4.025774754874238: 1,
2.8414664373266665: 1, 2.5707787646898166: 1, 14.41583960884392: 1, 3.7369516
63658535: 1, 1.4236642132032635: 1, 7.134993882807038: 1, 8.488558157593545:
1, 3.554983633032148: 1, 10.90244671979914: 1, 1.960102224858119: 1, 3.37135
45664137663: 1, 1.796058517439741: 1, 3.053093793640524: 1, 2.32130730079850
1: 1, 6.406744508904209: 1, 4.3112417454759955: 1, 4.476561578692514: 1, 3.02
60918238214454: 1, 2.6026209412804917: 1, 8.056187227866184: 1, 1.92951396718
67223: 1, 21.30994744567634: 1, 2.4210769568637347: 1, 2.058972766086852: 1,
2.6524858334077632: 1, 3.203760511613745: 1, 1.5981120873079966: 1, 5.976829
248989707: 1, 7.517760422665871: 1, 2.8869801445787635: 1, 1.760685130785411
2: 1, 12.924890258440508: 1, 5.153415165011352: 1, 1.6556395440511467: 1, 1.7
09349435408481: 1, 3.018119157901509: 1, 2.310226818278807: 1, 7.049238283156
446: 1, 4.923082915788506: 1, 3.609970912469738: 1, 4.586009699285279: 1, 3.8
740994347940236: 1, 3.4168267659363694: 1, 1.6421496582159825: 1, 3.965664444
709155: 1, 2.6526712483081583: 1, 5.2485967997924545: 1, 1.8760732552706696:
1, 2.1222493083978304: 1, 3.4810756657233917: 1, 6.647612051121796: 1, 2.620
6340602936358: 1, 2.1471245272374024: 1, 2.1714822643366465: 1, 4.35251492850
5542: 1, 3.723939199726147: 1, 2.655991414477655: 1, 5.38553217990417: 1, 1.5
521343593965766: 1, 5.235157366595839: 1, 2.949752492279726: 1, 2.10481171640
5549: 1, 3.803664926826071: 1, 8.183685826275422: 1, 2.0171211952895125: 1,
2.7107829954933917: 1, 7.111070701945676: 1, 1.7529191819875949: 1, 4.950432
504378355: 1, 7.186807358402944: 1, 3.487189479088979: 1, 6.030827577996865:

1, 5.406121915469242: 1, 1.6684385984289276: 1, 5.156191874806873: 1, 3.2482145481804148: 1, 2.942716509766977: 1, 2.8759824361927158: 1, 1.8053977574623747: 1, 4.278527827347879: 1, 2.9738349571758964: 1, 4.487224966213965: 1, 1.5326769500247877: 1, 5.911528021359585: 1, 5.858504147564461: 1, 10.7533793125136: 1, 1.6819289329604195: 1, 3.9129095003920984: 1, 1.427827694607125: 1, 2.1791719213087277: 1, 2.3279572806951276: 1, 4.35059681272415: 1, 3.1030365753796914: 1, 2.4739581735945775: 1, 2.4095206485792193: 1, 1.8294187989921533: 1, 5.449897320512118: 1, 2.5707953232902816: 1, 2.0552462799554267: 1, 1.43026310103249 82: 1, 2.0086569562806695: 1, 5.837271899564471: 1, 9.869599442775915: 1, 4.5998466835932055: 1, 6.250576463230955: 1, 7.140300165517914: 1, 2.703030194328924: 1, 5.87003346434122: 1, 3.038324316704242: 1, 3.236269666659469: 1, 2.5993966712160086: 1, 2.6189166739576444: 1, 8.351183460347736: 1, 1.7686875299987461: 1, 2.125624801616067: 1, 3.7999028789169316: 1, 1.6709351110649604: 1, 2.3333507561214373: 1, 2.2329095451642775: 1, 4.1455256627614: 1, 8.550810924804722: 1, 6.715074075719666: 1, 2.746783949807983: 1, 3.0850080523888854: 1, 3.4594084240272065: 1, 4.478261237163311: 1, 4.019879076331737: 1, 3.7599534028856874: 1, 2.302604732884478: 1, 1.9394886099904087: 1, 5.878637255849181: 1, 6.031470653262391: 1, 9.558140396227106: 1, 2.171945869863351: 1, 2.73693501643617: 1, 2.1361897258663323: 1, 2.0892890662525363: 1, 9.146534967693214: 1, 6.69392824888874: 1, 2.2041975806529215: 1, 5.001277996067843: 1, 1.9784562599267128: 1, 1.8305176403201215: 1, 16.634865888646758: 1, 3.6314309177643365: 1, 2.8531263403825196: 1, 3.3279373106944816: 1, 3.506402366914975: 1, 3.345474510913386: 1, 4.535260994726254: 1, 4.135391499829585: 1, 4.863238877627109: 1, 10.758084309800523: 1, 1.8751881614615409: 1, 3.6790606310493046: 1, 2.1167214065376836: 1, 6.7655831037713785: 1, 3.5789109628048097: 1, 4.541545312812818: 1, 1.6037386836578638: 1, 3.7588557622627294: 1, 2.555970506235756: 1, 1.6729260506557506: 1, 2.1984048269222405: 1, 2.0543062642844006: 1, 6.232471256679323: 1, 2.7711553462872165: 1, 28.77832701677388: 1, 11.822276415674168: 1, 1.8151830526223207: 1, 7.804544405991408: 1, 1.7149179618337567: 1, 7.146648154043847: 1, 2.7604057719901722: 1, 4.508988272611435: 1, 8.669762262710101: 1, 3.5776058320137487: 1, 4.408899737916157: 1, 2.487815829279843: 1, 2.0545998813041813: 1, 1.7641168544111325: 1, 4.01253932057559: 1, 2.1419253201812247: 1, 3.0245154928005777: 1, 1.8496674211233708: 1, 1.9668384721200123: 1, 7.025344665606633: 1, 12.764966319651393: 1, 3.573688750119664: 1, 2.412453947635999: 1, 3.8946287058401: 1, 2.3662204019207: 1, 2.012551358560435: 1, 2.4461106562525208: 1, 5.876482770573793: 1, 21.89507908765105: 1, 2.774837363382283: 1, 4.765395920645776: 1, 3.894193144823593: 1, 2.5577748017697433: 1, 2.178061501839761: 1, 1.9707747633149428: 1, 2.9891239924015864: 1, 1.8155399640663585: 1, 2.0844636238154117: 1, 3.0570082754940913: 1, 9.239861970622515: 1, 3.2505863063515803: 1, 3.2373107318604433: 1, 2.6809576655936023: 1, 5.346037222251855: 1, 2.6905011781690247: 1, 3.1452342815760277: 1, 2.6696292350066577: 1, 1.6123923835795946: 1, 3.572207268723178: 1, 5.0566482465635945: 1, 2.9660169931628304: 1, 1.4964241639045728: 1, 10.91614182528857: 1, 11.150339872290898: 1, 3.3099825011144817: 1, 4.3043798491866: 1, 10.589154315385459: 1, 3.7930608962465806: 1, 1.5225276270830277: 1, 4.042288680555694: 1, 2.0621059518882183: 1, 1.6914674456693044: 1, 6.11477959868407: 1, 2.3305735178471214: 1, 6.0007211743902475: 1, 4.721638890350651: 1, 3.1820655978796273: 1, 3.088342208772081: 1, 2.8604248983135445: 1, 2.7898777111809325: 1, 3.7735016902897938: 1, 15.010543169862757: 1, 13.537633375410437: 1, 2.1990270497176385: 1, 4.4264747882255655: 1, 1.925471484501961: 1, 4.70823987531394: 1, 13.959419626305909: 1, 4.417739297614632: 1, 18.3192266386342: 1, 5.791849292431867: 1, 9.493363264447165: 1, 2.129513824669184: 1, 3.426537375350426: 1, 2.9514019292149483: 1, 2.100601706949084: 1, 15.528083717422149: 1, 16.04344437938377: 1, 16.575000961971387: 1, 2.780286276293311: 1, 2.8812822762074095: 1, 1.8220721516453786: 1, 8.675812233379904: 1, 3.2921358910146945: 1, 3.519394732831559: 1, 4.765008508213016: 1, 14.69623447072907: 1, 1.8632885172665579: 1, 2.1347001004463224: 1, 4.128320389812961:

1, 3.8626798275851173: 1, 1.9301294771317707: 1, 2.3355771074269915: 1, 1.564
4166131864448: 1, 4.315806157487578: 1, 8.214040921046287: 1, 2.5918717580671
61: 1, 6.889874096282919: 1, 2.6995520716954045: 1, 1.746733704514942: 1, 2.7
843835389569045: 1, 2.1273758515168337: 1, 2.134118087792988: 1, 9.1700712042
19514: 1, 2.6771224032043435: 1, 13.377429726017347: 1, 3.1767317663411605:
1, 5.287709993165299: 1, 2.021660304439333: 1, 2.321807255638083: 1, 3.40052
87104068094: 1, 4.528851295691017: 1, 2.3434416470276527: 1, 2.41248248306349
57: 1, 1.814554341077816: 1, 2.9602708068109655: 1, 2.0385562471372074: 1, 1
4.21231968715128: 1, 6.3070013456492156: 1, 1.7567789006672965: 1, 1.44951466
13182726: 1, 1.4797395421399295: 1, 1.998924952997376: 1, 5.248353156953867:
1, 1.6386081446261118: 1, 2.74033924843737: 1, 2.512763437653737: 1, 2.68890
8450836698: 1, 6.453799634165966: 1, 2.7779072966930376: 1, 4.65492866118388:
1, 4.741436438497579: 1, 4.216865391247982: 1, 19.952804972482767: 1, 2.49472
2301466361: 1, 2.266251247663355: 1, 4.037864839818105: 1, 3.774298238912396:
1, 4.085310314417008: 1, 15.047131151828554: 1, 2.899072210828563: 1, 4.45708
2132767123: 1, 11.245250384920357: 1, 2.8667449601043242: 1, 3.25407959482593
2: 1, 1.7548322620565606: 1, 2.7622464971161853: 1, 2.0264770612558993: 1, 2.
642583068153687: 1, 1.9893658918639365: 1, 2.6385515425810353: 1, 1.984966861
5294267: 1, 3.6137251585592836: 1, 2.3877793931106983: 1, 4.261995376714956:
1, 1.9892820031176734: 1, 9.344506356278202: 1, 2.105153075197044: 1, 2.4525
241305663723: 1, 2.5715653798764593: 1, 11.587693514154676: 1, 9.926205928769
004: 1, 12.258898523232094: 1, 2.8598870700864882: 1, 3.883314431159668: 1,
3.9732929735339373: 1, 1.9255949617696362: 1, 2.5774293095977936: 1, 7.66541
1680587651: 1, 8.326150625351238: 1, 1.8672486592139612: 1, 2.307516429272091
8: 1, 7.538913041705628: 1, 4.179346810604422: 1, 3.6317749490187756: 1, 3.01
3757203967437: 1, 2.5380069137631063: 1, 1.664969929734363: 1, 4.318244653954
532: 1, 2.6122053486642876: 1, 2.3659809060257535: 1, 2.296854246677625: 1,
3.2333866516707044: 1, 1.4219618216561585: 1, 5.438672715914238: 1, 5.295987
794854426: 1, 5.803261462833208: 1, 2.097061771658407: 1, 2.9867293359890783:
1, 4.004255328535251: 1, 7.861722440186134: 1, 3.4939126461219665: 1, 22.8208
59335276253: 1, 5.225022341606497: 1, 3.65008857698413: 1, 3.389408242679715:
1, 2.72701619627953: 1, 8.229762823375646: 1, 1.8797885407223962: 1, 2.734501
5434940594: 1, 1.7826887633115127: 1, 2.2186022468025004: 1, 14.4257561985540
16: 1, 3.7398505802726363: 1, 4.359625095499266: 1, 1.7156526017189384: 1, 2.
5229973937706: 1, 6.608831299584674: 1, 3.1664069148653726: 1, 2.619457984241
173: 1, 4.089707358414025: 1, 7.879584873185977: 1, 3.9698708665495785: 1, 5.
549546940378514: 1, 5.464635170255016: 1, 1.8798862039910866: 1, 4.6682141826
07648: 1, 2.7610039013325838: 1, 15.987416357464415: 1, 2.084729519183939: 1,
3.9192640749919616: 1, 3.662244612782777: 1, 2.423643733327753: 1, 2.49751757
50992697: 1, 1.9939339100701783: 1, 1.8702672955649762: 1, 7.349272282591393:
1, 5.649486510510465: 1, 7.412993603331431: 1, 7.151971359771085: 1, 3.883606
6343567186: 1, 2.0180818477302958: 1, 2.4650607544938166: 1, 1.73434063761138
06: 1, 10.569141981432782: 1, 18.62051291558272: 1, 3.1887063330746095: 1, 3.
2506649432142933: 1, 11.291766028565105: 1, 6.699775008109292: 1, 3.301402258
7275083: 1, 1.8950870204867822: 1, 1.5948307113140416: 1, 8.660877915872312:
1, 1.9651524130736546: 1, 4.565288813205145: 1, 4.314753044966077: 1, 2.5053
35510860414: 1, 3.854040262194085: 1, 2.6577232207328008: 1, 2.14764622352212
6: 1, 1.9211625697248098: 1, 3.840230020727685: 1, 2.0489279604448387: 1, 13.
671582006527933: 1, 1.6483282536044865: 1, 3.1200398776023155: 1, 4.150868516
824901: 1, 3.6218657805628216: 1, 1.5284987200079418: 1, 2.6764529856335364:
1, 7.377150673076735: 1, 2.588365065137743: 1, 3.953597972497118: 1, 5.52491
6836561636: 1, 3.9794372630856434: 1, 3.192257320607187: 1, 2.403185303469095
8: 1, 3.169847959099836: 1, 2.258892349385415: 1, 2.6244902304906037: 1, 1.91
38868043165824: 1, 3.3006275418349724: 1, 1.8607137007334458: 1, 2.5802875115
288377: 1, 2.761207540413327: 1, 1.8216544959435252: 1, 10.461330055674672:
1, 1.8689688779779738: 1, 4.1563588141154915: 1, 1.724366548991631: 1, 2.340
0005512856166: 1, 3.3569852543311893: 1, 2.669496437876057: 1, 4.363324745113

725: 1, 1.6251255989129652: 1, 3.891487813063303: 1, 3.178689677795475: 1, 2.
9606221774401824: 1, 1.6818187205675341: 1, 2.1156194666378365: 1, 5.90210106
5338085: 1, 3.262275878262796: 1, 19.500644717937316: 1, 1.6995704089574957:
1, 5.225597993001838: 1, 2.212006702441236: 1, 5.160139759409655: 1, 3.899098
6592175414: 1, 5.814937867496732: 1, 3.632858487322533: 1, 2.174273033038286
7: 1, 1.5864316131399336: 1, 2.7002213995221895: 1, 2.0351037695537126: 1, 4.
959398801586951: 1, 1.7700597182963045: 1, 2.8052092209590818: 1, 3.611089901
1616113: 1, 3.9629753873050007: 1, 1.5506664552231784: 1, 1.9242670400649875:
1, 4.414205033630679: 1, 3.680601784506788: 1, 2.177688173704004: 1, 6.504709
0294859045: 1, 1.4759425463844413: 1, 4.477506943872623: 1, 3.764100725915238
6: 1, 2.004663177795959: 1, 1.8249056430857542: 1, 1.6943008972399707: 1, 4.4
37564706717901: 1, 13.254501353906427: 1, 3.0059041389174777: 1, 2.7542898265
577542: 1, 1.5997896352192742: 1, 3.2362834073810416: 1, 1.635904392080503:
1, 1.7500150621947366: 1, 7.2724100152987985: 1, 6.210163863805771: 1, 3.335
8771775987996: 1, 5.201674512784148: 1, 1.7736373276401987: 1, 2.259789096550
529: 1, 3.5133899025708333: 1, 2.438814043817002: 1, 2.6868773034042697: 1,
1.9647614607643913: 1, 3.612515961647576: 1, 2.2925087640597392: 1, 14.96992
0930362356: 1, 3.525087253172701: 1, 3.021560062037343: 1, 2.220162192958840
7: 1, 1.8282510548066835: 1, 1.5347838630001418: 1, 5.518306166971162: 1, 23.
536497420808107: 1, 3.2530176363719785: 1, 6.275810950885312: 1, 2.6132653909
530834: 1, 4.217742270176967: 1, 3.7317429737319805: 1, 2.1045743954919898:
1, 2.3197156015782934: 1, 1.5727933851918707: 1, 2.8361471057587964: 1, 2.69
9308006448123: 1, 2.967965684715921: 1, 2.1734440415737812: 1, 9.324610339105
38: 1, 11.088681415528848: 1, 2.5391908927175306: 1, 4.118660173136943: 1, 3.
174546408682326: 1, 2.3941102576305857: 1, 3.947118101073202: 1, 7.5152743570
6576: 1, 5.806701800870241: 1, 2.3912038299646636: 1, 2.0392854372977873: 1,
2.1642124657034985: 1, 1.8152924735771157: 1, 2.4641267210926205: 1, 2.44510
7044682122: 1, 2.252078837908373: 1, 1.608429800814909: 1, 1.773276659249073
6: 1, 2.5485405448327647: 1, 1.95556893076565: 1, 2.6327561527514733: 1, 8.83
7302768160594: 1, 2.3964286169798137: 1, 1.9077904739444007: 1, 1.79728693940
71456: 1, 6.473401028500274: 1, 2.187987377728135: 1, 4.0014463308934705: 1,
3.2399743760130506: 1, 1.8930917050710323: 1, 2.1844241531904625: 1, 3.20249
34963332465: 1, 1.9427394173021209: 1, 8.887954125914762: 1, 1.92377321349978
85: 1, 2.3959907286313924: 1, 2.3385229645729178: 1, 2.918500494100741: 1, 7.
380761089163949: 1, 2.2132582536265164: 1, 6.665898054816912: 1, 2.1940880606
30473: 1, 9.369513400464353: 1, 5.260315009655959: 1, 2.452301230318205: 1,
2.4711108726420927: 1, 5.906802305167259: 1, 2.340075741749339: 1, 3.7935872
893331597: 1, 2.980574972738064: 1, 2.6557222210462466: 1, 2.209063990190682
3: 1, 9.791213920474405: 1, 3.30399736539893: 1, 4.521972631200207: 1, 3.7826
454302257537: 1, 10.084465904022203: 1, 9.631174978901274: 1, 1.6990865773479
455: 1, 2.9723792010076653: 1, 2.2253856651559283: 1, 3.5948540172743937: 1,
4.032780021158514: 1, 1.760130272340689: 1, 1.585283847003801: 1, 2.66133673
41465213: 1, 2.026924066586586: 1, 6.816547211794484: 1, 7.139875213750829:
1, 4.056466117136366: 1, 19.704138139507428: 1, 4.955043563901602: 1, 2.6141
129251112574: 1, 1.6334540135981261: 1, 2.1842980430036194: 1, 3.918305278399
42: 1, 2.2055757895330292: 1, 5.021235172442317: 1, 11.247467730674867: 1, 2.
753856029830703: 1, 2.0880985107837344: 1, 4.028533147204184: 1, 1.5464509046
702524: 1, 5.894934162808145: 1, 2.6026162222714135: 1, 5.2211880241301705:
1, 6.714752400572533: 1, 3.6048453119845902: 1, 2.036136582248825: 1, 2.6541
043323091036: 1, 1.4825060324867971: 1, 2.5243165261530125: 1, 4.754741021759
917: 1, 7.20552500275968: 1, 12.34490017543349: 1, 2.707257464510345: 1, 3.38
08511984413125: 1, 1.5839721898699406: 1, 2.4252117165079783: 1, 3.5485135137
78: 1, 4.4283472666803965: 1, 6.347259747107386: 1, 9.222067834986113: 1, 2.3
3001366797641: 1, 9.978300589563343: 1, 4.896437590378795: 1, 5.0163165854840
14: 1, 3.1869333359326366: 1, 4.469001117727998: 1, 2.769368665049499: 1, 2.1
50446225762333: 1, 3.8550797622550474: 1, 13.316009867647931: 1, 3.5047833387
797223: 1, 2.104599091389626: 1, 20.332058890867923: 1, 1.8471967576612955:

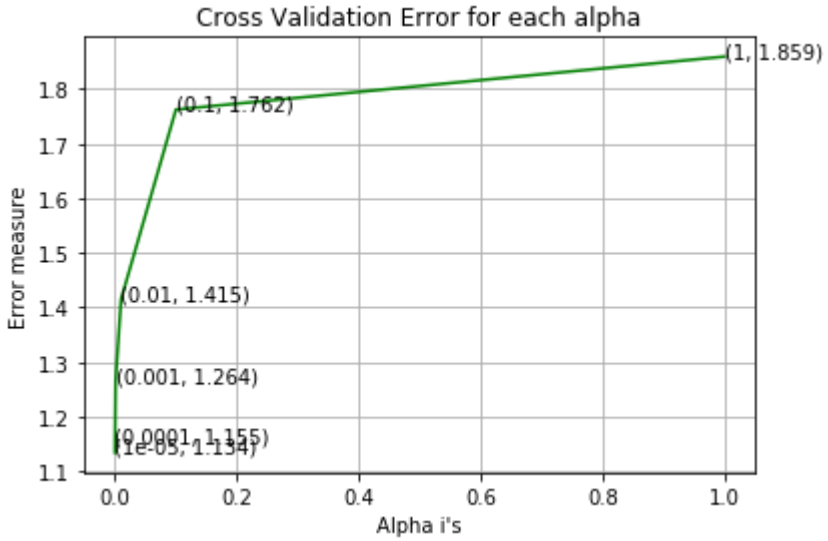
1, 5.647615964932417: 1, 2.8046297897072257: 1, 2.207405017828204: 1, 7.7526
01973793317: 1, 5.6622116094194865: 1, 3.738959845606858: 1, 19.5737347078162
86: 1, 2.25530883512245: 1, 1.6506845246296837: 1, 3.000133823744417: 1, 1.86
03102707914931: 1, 2.492014798042409: 1, 14.109849102683466: 1, 9.60045381355
3814: 1, 6.224529102696896: 1, 5.872622176907166: 1, 10.362001128012578: 1,
2.459723901288019: 1, 4.474404836487285: 1, 3.4487069816219407: 1, 4.3193955
38523893: 1, 3.5470052143359267: 1, 4.534520970267881: 1, 2.884669324325147:
1, 3.2705970395151818: 1, 1.6568134518353355: 1, 2.5463040616640713: 1, 2.17
01505543474564: 1, 15.078012555020319: 1, 3.0579329616968955: 1, 1.8114017293
713638: 1, 5.344251506741215: 1, 4.552142829013578: 1, 4.727409061470478: 1,
11.669759037248339: 1, 2.2586626428482006: 1, 1.8769061225605437: 1, 1.82997
24272813884: 1, 2.5794787546424285: 1, 7.615745097418866: 1, 1.31584055501180
5: 1, 3.2315151504652393: 1, 5.0603681682509185: 1, 2.120651835196477: 1, 2.4
30374071013341: 1, 5.488456473048657: 1, 3.1344009715260075: 1, 1.91957020095
96958: 1, 2.5036246553340953: 1, 8.055385394457923: 1, 1.4323460729347628: 1,
16.35274095261728: 1, 4.483472040056363: 1, 5.080182411540333: 1, 3.075711726
007688: 1, 12.227370811121075: 1, 2.0879319233687443: 1, 2.3711167720425577:
1, 2.8116516848753603: 1, 42.71281290261081: 1, 2.2128593810071995: 1, 3.593
536851476367: 1, 8.70362049218771: 1, 3.66459749155555: 1, 2.129139595800203
7: 1, 9.547466705203146: 1, 4.572183406966575: 1, 5.178098494648418: 1, 3.301
4484704938405: 1, 7.198997833423594: 1, 2.906823107396394: 1, 17.90544578435
9: 1, 5.541657880145091: 1, 2.7872941908144004: 1, 2.647742490747969: 1, 2.17
6367496167487: 1, 3.55891937554714: 1, 3.3937179834691085: 1, 3.7394050813926
24: 1, 1.7917210208588334: 1, 1.7698228341912527: 1, 5.904926932392731: 1, 2.
340365354887787: 1, 5.1677693895271535: 1, 6.378177836939746: 1, 3.8472343692
73269: 1, 3.305312467511982: 1, 2.4338934297571373: 1, 5.28361358207125: 1,
1.885104064407063: 1, 1.5529054094951886: 1, 2.0075244669974865: 1, 2.439227
532904264: 1, 5.627798473045731: 1, 14.356476696158763: 1, 2.605653469592362
5: 1, 4.942420025677597: 1, 4.018279179799866: 1, 2.7680262122993926: 1, 12.2
52372004066109: 1, 1.2717774476777814: 1, 4.697052417924623: 1, 2.25044570681
49626: 1, 4.45635562661695: 1, 6.418589112112822: 1, 6.604077004442603: 1, 6.
108903847764094: 1, 4.506239142014851: 1, 2.3696594283473935: 1, 3.0703725535
680237: 1, 2.376440990874915: 1, 2.39533746244717: 1, 1.6189356922949583: 1,
1.865678367271386: 1, 44.88535232568568: 1, 7.52768433515005: 1, 28.83728560
9038624: 1, 9.283055594704098: 1, 11.420794200047098: 1, 1.7557158245956666:
1, 1.920670107197466: 1, 1.6537926598328212: 1, 2.5075416222732416: 1, 2.072
8335911808924: 1, 7.296293530295076: 1, 2.244580280428102: 1, 5.7061620941685
08: 1, 11.07834497554939: 1, 2.650978295606738: 1, 3.7595680467661516: 1, 1.5
249795460647622: 1, 2.177007371803502: 1, 1.8344865841606428: 1, 1.6578919694
38471: 1, 2.6789887116518947: 1, 3.079364812734613: 1, 3.0854210540815696: 1,
5.867951526409956: 1, 3.218243564696907: 1, 9.913094107693158: 1, 12.90198231
6373264: 1, 2.0331556242018523: 1, 2.355564538626029: 1, 11.065782237962921:
1, 19.47419904350562: 1, 2.63672524884932: 1, 1.6801399847301515: 1, 2.37285
65505272816: 1, 2.561067050653983: 1, 1.9681883636835924: 1, 2.10777776356452
27: 1, 2.377644021235308: 1, 2.386140530840909: 1, 2.3241514711545754: 1, 1.8
37109598906371: 1, 2.23393923593511: 1, 30.048933664364746: 1, 3.483088644614
364: 1, 2.7916812660598826: 1, 5.561310893495972: 1, 2.8621700416967024: 1,
2.1539449007797726: 1, 5.093563155687522: 1, 1.8015095382267867: 1, 9.406496
113416573: 1, 3.0653888710896764: 1, 7.010843325905671: 1, 1.771398840578067
4: 1, 5.116384369323544: 1, 9.673437867323312: 1, 5.601285550869684: 1, 3.987
7693107792327: 1, 2.3445016319564806: 1, 5.733170067051655: 1, 1.977665257528
5336: 1, 2.7939785117217593: 1, 31.51162998853588: 1, 2.5016336161750123: 1,
2.6588508438240974: 1, 3.6521216433527046: 1, 5.4903759623437765: 1, 2.81878
5713942192: 1, 3.973890552567035: 1, 1.5347128442537747: 1, 2.07385722275281
3: 1, 6.527593548587323: 1, 1.7653190993136245: 1, 2.337081326267135: 1, 5.16
4483903868664: 1, 4.64456631055332: 1, 6.078865250069681: 1, 3.25938730975857
06: 1, 1.5977671020415922: 1, 2.3083459744491495: 1, 3.599117452159434: 1, 1.

609848304790623: 1, 2.8048815276281815: 1, 2.45233525855798: 1, 1.96238290075
87579: 1, 3.265857224555425: 1, 7.9237291197721165: 1, 28.68644531130852: 1,
2.4105819813982188: 1, 3.5683722217772895: 1, 3.9306610398095505: 1, 4.55738
64500834675: 1, 2.092887033845637: 1, 1.7118025804535817: 1, 5.75015208524387
3: 1, 4.023885386490707: 1, 2.067918701812241: 1, 3.1244528561235168: 1, 1.54
51823429784055: 1, 2.9564009612535274: 1, 4.420109777661301: 1, 1.57533121691
16986: 1, 16.15518947226495: 1, 1.9912154005706932: 1, 6.4200818596342835: 1,
2.495159763919917: 1, 1.8904524107419263: 1, 1.7603991061545232: 1, 6.1861387
683360345: 1, 2.279449743554598: 1, 8.32230418958169: 1, 1.7727069587578053:
1, 2.322097360711119: 1, 2.407515290872887: 1, 4.812172217736238: 1, 3.32792
1156157933: 1, 1.7948652409439807: 1, 2.0999118448902925: 1, 1.41765826484466
78: 1, 3.1839517344052433: 1, 3.5890164288493653: 1, 4.420840241802475: 1, 1.
9648695702126877: 1, 2.8822610225525764: 1, 1.7598124680885088: 1, 9.18983640
113878: 1, 1.7423951883437285: 1, 3.252306515659152: 1, 4.017416238500076: 1,
6.472111691901854: 1, 3.5304664747662726: 1, 1.9168075794017725: 1, 5.7075192
58108951: 1, 6.640484046806503: 1, 1.5258763129308668: 1, 3.4582018127618346:
1, 6.645399853343004: 1, 2.714674607159018: 1, 2.672109265761522: 1, 1.875052
714672158: 1, 1.9907137294172603: 1, 1.7234613371077576: 1, 9.36884615573743
1: 1, 14.593628830578597: 1, 3.0233699349008556: 1, 1.9466793531945155: 1, 2.
9554467580734523: 1, 1.6739102569746438: 1, 4.037906286236363: 1, 2.753767536
66365: 1, 4.789839771917571: 1, 6.3614749147104614: 1, 4.270797822135186: 1,
2.6950317890355624: 1, 3.8900544206237195: 1, 2.4809018071093494: 1, 2.16440
8513372443: 1, 1.847174868527326: 1, 2.899522360116662: 1, 4.098636265021706:
1, 11.801956122351601: 1, 2.016007385505492: 1, 1.5795624451450656: 1, 2.3516
190726847466: 1, 7.469385050489668: 1, 3.5055879263122214: 1, 1.5742388036406
065: 1, 10.657010977519846: 1, 5.1874256331131425: 1, 2.353113814072665: 1, 2
0.08708425040014: 1, 5.026041588127563: 1, 3.2440816403234223: 1, 2.663872577
5056997: 1, 2.017032316572703: 1, 4.057306578442556: 1, 1.6806243866994288:
1, 3.74741611265987: 1, 19.96416332918929: 1, 5.212862900139114: 1, 2.365559
628369199: 1, 4.3338196130502595: 1, 1.9077165075720555: 1, 8.09993778058036
7: 1, 3.9526074542708143: 1, 2.9243101508876417: 1, 6.5689092724622276: 1, 1.
8579734793463305: 1, 2.5098931634845423: 1, 4.682917264619839: 1, 3.784790783
282482: 1, 2.5537165389189846: 1, 5.598689880203177: 1, 1.7453810443632505:
1, 2.030337089106282: 1, 3.582701200977046: 1, 2.1253306571102266: 1, 3.1121
667934642: 1, 5.953211203286377: 1, 4.091777654352542: 1, 2.9871781358819276:
1, 2.460891649896471: 1, 1.9850632650334121: 1, 3.6812314901589596: 1, 20.323
851705935652: 1, 7.875737758524614: 1, 2.4405317181187334: 1, 5.4296542834359
73: 1, 4.018959848906479: 1, 2.588049003684938: 1, 3.7131636183162806: 1, 4.7
8692735701697: 1, 2.665306733282216: 1, 1.8882517991425152: 1, 1.641883466972
7433: 1, 14.428373782026132: 1, 1.8552593852705526: 1, 3.044835680428484: 1,
5.835896638933527: 1, 4.149831506528383: 1, 3.0986917626499704: 1, 2.5883855
5906016: 1, 1.9096142179811386: 1, 8.535015634069271: 1, 2.1443041786066375:
1, 2.230293642569527: 1, 2.620358936458949: 1, 4.21512615271442: 1, 4.639170
27993424: 1, 2.205196141849023: 1, 4.981810325530824: 1, 3.111031690695165:
1, 2.6636750528160693: 1, 3.879813768850867: 1, 1.3271959211831708: 1, 2.611
4188925844934: 1, 15.462853888438975: 1, 2.3596112954885853: 1, 4.10020850467
0244: 1, 2.844393125353212: 1, 2.6326462895372034: 1, 2.625804316523701: 1,
2.9968849420047867: 1, 1.9075651636701803: 1, 1.6841420950822583: 1, 2.63405
2667127738: 1, 7.156856355745724: 1, 5.035548593427898: 1, 2.997021446869795:
1, 3.679716284204996: 1, 4.320658683663556: 1, 3.149115377652601: 1, 3.166277
7835166116: 1, 2.795648221440184: 1, 2.439263570788011: 1, 3.065447987743697
8: 1, 5.004171454279867: 1, 2.341375737514138: 1, 5.766896725916767: 1, 2.635
621506585937: 1, 1.8825359771929684: 1, 8.448452939697397: 1, 6.3844936477959
875: 1, 6.144293549976674: 1, 12.411709038188215: 1, 4.376712764611733: 1, 3.
9988737139167605: 1, 2.438838026415847: 1, 2.3758288350446106: 1, 1.425516508
244518: 1, 17.00066248040797: 1, 6.894952611646706: 1, 3.038265935027754: 1,
1.8495096715033226: 1, 2.2594228083032517: 1, 2.8439426981450295: 1, 5.45925

1048660321: 1, 5.8925692395689016: 1, 4.626394207958027: 1, 1.750119039698162
7: 1, 3.790589582233995: 1, 2.209697996167242: 1, 5.726827296886194: 1, 2.001
554370680157: 1, 2.0561843348415696: 1, 3.3544917655769337: 1, 2.123339547625
5867: 1, 1.7432847774566482: 1, 1.8529430805050389: 1, 3.2602997170218675: 1,
4.550809866965061: 1, 14.497485650414424: 1, 8.897266501166184: 1, 4.11875537
4554043: 1, 4.711585615148915: 1, 2.0957243610488194: 1, 1.7162578605201697:
1, 1.9710751191512372: 1, 2.3641719656515994: 1, 4.862078340200976: 1, 1.701
7532281100896: 1, 3.6380233850713597: 1, 2.2306708835010305: 1, 1.83164562100
03733: 1, 2.3833754684339654: 1, 3.592804976606175: 1, 15.286178824096387: 1,
1.5845155931137467: 1, 2.1737194223045004: 1, 3.1742056720913587: 1, 1.604915
0273968171: 1, 1.5950408731220689: 1, 1.8653480051173748: 1, 11.0880197330201
26: 1, 3.0566947598292047: 1, 2.8557450390814725: 1, 5.052390050909767: 1, 1.
6810970006174955: 1, 2.0753490030456065: 1, 7.200850775886801: 1, 5.688402639
5591105: 1, 1.4192139045722754: 1, 3.044036702168338: 1, 2.0686563944009575:
1, 1.860633386517347: 1, 4.002947703121763: 1, 2.394074109661286: 1, 3.41152
78143666123: 1, 3.6224606223951725: 1, 2.9248144793954953: 1, 1.6959398966463
082: 1, 1.8261978856293812: 1, 2.4366402223526977: 1, 8.596556294602346: 1,
6.444080117485792: 1, 6.66642860942846: 1, 4.196525051181014: 1, 2.731128038
757441: 1, 4.742809155176811: 1, 6.014755752276228: 1, 3.4047088831026966: 1,
2.5328559976240372: 1, 3.8530074621370747: 1, 1.853359770860917: 1, 2.6838452
55392652: 1, 2.2346378838397087: 1, 1.632659168854976: 1, 3.5250179627953027:
1, 1.7051491060957942: 1, 3.1681216321235834: 1, 3.3226167799672823: 1, 10.74
57831922796: 1, 2.075468033759155: 1, 3.9670484907834638: 1, 2.05603429442755
07: 1, 7.443559422147298: 1, 22.27779782236654: 1, 2.7245065145946414: 1, 5.4
24785728098751: 1, 3.68675517607644: 1, 2.578440519893073: 1, 3.7653280275197
263: 1, 5.173127671163465: 1, 2.634463251631215: 1, 2.6541635990935477: 1, 1.
715117205551449: 1, 2.946306093773967: 1, 2.0248211585811013: 1, 1.5292685569
851385: 1, 4.3542129759105075: 1, 14.555677368631898: 1, 2.100374623256627:
1, 2.24947341404538: 1, 3.2103265465149975: 1, 2.441879777430048: 1, 10.3635
78965821484: 1, 1.771424408453653: 1, 2.4856558246355624: 1, 3.47767155827652
36: 1, 1.8584837601752628: 1, 6.211525057109721: 1, 4.783195337073581: 1, 2.3
072626927318898: 1, 2.3320161451100714: 1, 4.311569891322105: 1, 14.087647936
696603: 1, 3.213708474216529: 1, 2.07732589219472: 1, 2.096082839502938: 1,
1.5728971990486387: 1, 1.8820468724728416: 1, 6.304419450342875: 1, 2.678614
0288291294: 1, 2.1317883219675333: 1, 2.4863317577441895: 1, 3.42592836921476
44: 1, 11.056006335261708: 1, 3.8825032124551178: 1, 2.933824874528558: 1, 3.
5307233565969742: 1, 10.510391215836652: 1, 2.9940442551139546: 1, 2.93773739
61498487: 1, 5.352559493043929: 1, 5.869776973513862: 1, 3.1189764704691907:
1, 2.6213957727593695: 1, 50.159496797574526: 1, 1.6192671273922192: 1, 16.4
20957640549766: 1, 2.503373948504413: 1, 2.10732984961622: 1, 17.510066007254
864: 1, 12.681663160051352: 1, 2.486108343261036: 1, 1.981448072091282: 1, 3.
2308601401579105: 1, 4.831556549548304: 1, 4.323009319178284: 1, 6.2150436116
68929: 1, 8.333996801817156: 1, 5.974497467148506: 1, 4.678427302600197: 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)]

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: 1.1339680227335633
For values of alpha =  0.0001 The log loss is: 1.1548972359842409
For values of alpha =  0.001 The log loss is: 1.2642637719707213
For values of alpha =  0.01 The log loss is: 1.41484086363408
For values of alpha =  0.1 The log loss is: 1.762350379931959
For values of alpha =  1 The log loss is: 1.859327529546437
```




```
For values of best alpha =  1e-05 The train log loss is: 0.65114917839196
For values of best alpha =  1e-05 The cross validation log loss is: 1.1339680
227335633
For values of best alpha =  1e-05 The test log loss is: 1.1786260735640253
```

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(ngram_range = (1,4), min_df=10, max_features
=5000)
    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")
```

```
91.08 % of word of test data appeared in train data
90.84 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [50]:
```
#Data preparation for ML models.

#Misc. functionns for ML models


def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y, clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    pred_y = sig_clf.predict(test_x)

    # for calculating log_loss we willl provide the array of probabilities 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 [51]:
```
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

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

# building train, test and cross validation data sets
# a = [[1, 2],
#      [3, 4]]
# b = [[4, 5],
#      [6, 7]]
# hstack(a, b) = [[1, 2, 4, 5],
#                 [ 3, 4, 6, 7]]

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_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 [54]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_onehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 7205)
(number of data points * number of features) in test data =  (665, 7205)
(number of data points * number of features) in cross validation data = (532, 7205)
```

In [55]:
```
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)
```

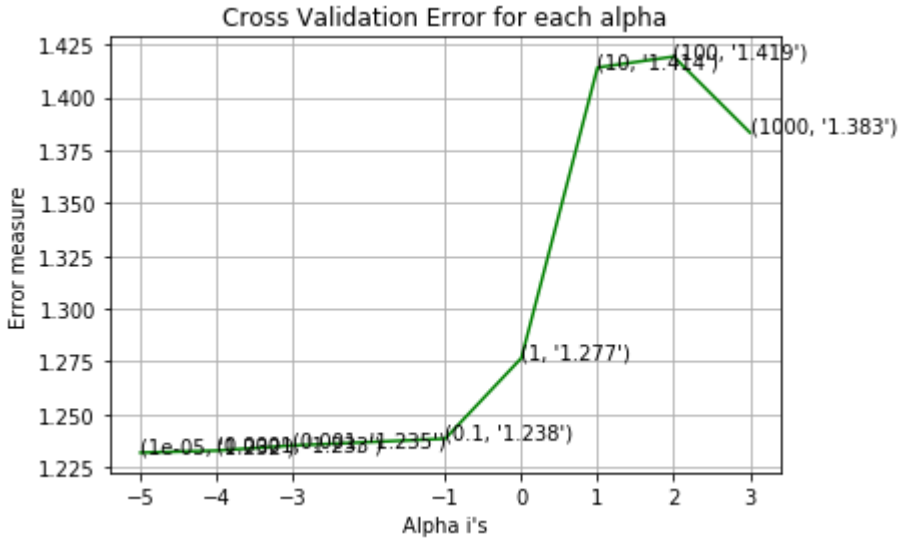
4.1. Base Line Model

4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [98]:
```
alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes
_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probab
ility estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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


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

```
for alpha = 1e-05
Log Loss : 1.2318129246635912
for alpha = 0.0001
Log Loss : 1.232785201993066
for alpha = 0.001
Log Loss : 1.235205053367683
for alpha = 0.1
Log Loss : 1.2384152725001947
for alpha = 1
Log Loss : 1.2765966711360812
for alpha = 10
Log Loss : 1.4141490525487528
for alpha = 100
Log Loss : 1.4193139479314296
for alpha = 1000
Log Loss : 1.3833884669045606
```




```
For values of best alpha =  1e-05 The train log loss is: 0.6749756701031507
For values of best alpha =  1e-05 The cross validation log loss is: 1.2318129
246635912
For values of best alpha =  1e-05 The test log loss is: 1.1754148928058274
```

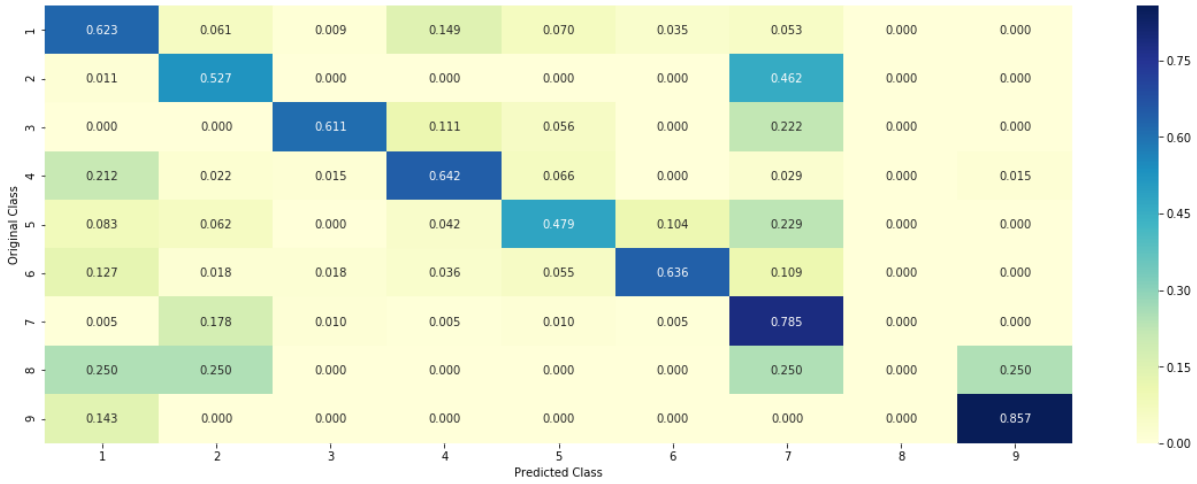
4.1.1.2. Testing the model with best hyper paramters

In [101]:
```
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(test_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probabilit
y estimates
print("Log Loss :",log_loss(test_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(te
st_x_onehotCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y, sig_clf.predict(test_x_onehotCoding.toarray()))
```

```
Log Loss : 1.1754148928058274
Number of missclassified point : 0.35037593984962406
-------------------- Confusion matrix --------------------
```


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


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



4.1.1.3. Feature Importance, Correctly classified point

In [58]:
```
test_point_index = 11
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 : 4
Predicted Class Probabilities: [[0.0695 0.0559 0.0164 0.6976 0.037  0.036  0.
0809 0.0042 0.0023]]
Actual Class : 4
--------------------------------------------------
11 Text feature [activity] present in test data point [True]
12 Text feature [protein] present in test data point [True]
13 Text feature [proteins] present in test data point [True]
18 Text feature [experiments] present in test data point [True]
19 Text feature [function] present in test data point [True]
20 Text feature [ability] present in test data point [True]
24 Text feature [results] present in test data point [True]
27 Text feature [acid] present in test data point [True]
28 Text feature [loss] present in test data point [True]
29 Text feature [functions] present in test data point [True]
31 Text feature [determined] present in test data point [True]
32 Text feature [shown] present in test data point [True]
34 Text feature [whether] present in test data point [True]
35 Text feature [suppressor] present in test data point [True]
36 Text feature [important] present in test data point [True]
37 Text feature [missense] present in test data point [True]
38 Text feature [described] present in test data point [True]
40 Text feature [bind] present in test data point [True]
41 Text feature [amino] present in test data point [True]
43 Text feature [also] present in test data point [True]
44 Text feature [type] present in test data point [True]
45 Text feature [mutations] present in test data point [True]
46 Text feature [two] present in test data point [True]
47 Text feature [indicate] present in test data point [True]
50 Text feature [functional] present in test data point [True]
51 Text feature [partially] present in test data point [True]
52 Text feature [retained] present in test data point [True]
53 Text feature [indicated] present in test data point [True]
56 Text feature [although] present in test data point [True]
57 Text feature [may] present in test data point [True]
59 Text feature [wild] present in test data point [True]
60 Text feature [levels] present in test data point [True]
62 Text feature [either] present in test data point [True]
67 Text feature [containing] present in test data point [True]
68 Text feature [purified] present in test data point [True]
69 Text feature [expressed] present in test data point [True]
70 Text feature [incubated] present in test data point [True]
72 Text feature [determine] present in test data point [True]
73 Text feature [analyzed] present in test data point [True]
75 Text feature [30] present in test data point [True]
77 Text feature [thus] present in test data point [True]
80 Text feature [vitro] present in test data point [True]
81 Text feature [buffer] present in test data point [True]
85 Text feature [tris] present in test data point [True]
86 Text feature [general] present in test data point [True]
88 Text feature [sds] present in test data point [True]
89 Text feature [terminal] present in test data point [True]
90 Text feature [previously] present in test data point [True]
91 Text feature [reduced] present in test data point [True]
92 Text feature [hcl] present in test data point [True]
93 Text feature [show] present in test data point [True]
96 Text feature [lack] present in test data point [True]
```

```
97 Text feature [lower] present in test data point [True]
98 Text feature [three] present in test data point [True]
99 Text feature [possible] present in test data point [True]
Out of the top  100  features  55 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [60]:
```
test_point_index = 5
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: [[0.094  0.4288 0.0222 0.0968 0.0499 0.0488 0.
2508 0.0057 0.0031]]
Actual Class : 6
--------------------------------------------------
13 Text feature [patients] present in test data point [True]
14 Text feature [treatment] present in test data point [True]
15 Text feature [clinical] present in test data point [True]
16 Text feature [time] present in test data point [True]
17 Text feature [response] present in test data point [True]
18 Text feature [first] present in test data point [True]
19 Text feature [molecular] present in test data point [True]
21 Text feature [study] present in test data point [True]
22 Text feature [therapy] present in test data point [True]
23 Text feature [recently] present in test data point [True]
24 Text feature [including] present in test data point [True]
25 Text feature [15] present in test data point [True]
26 Text feature [also] present in test data point [True]
27 Text feature [using] present in test data point [True]
28 Text feature [achieved] present in test data point [True]
29 Text feature [13] present in test data point [True]
30 Text feature [identified] present in test data point [True]
31 Text feature [10] present in test data point [True]
32 Text feature [common] present in test data point [True]
33 Text feature [therapeutic] present in test data point [True]
34 Text feature [initial] present in test data point [True]
35 Text feature [analysis] present in test data point [True]
36 Text feature [11] present in test data point [True]
37 Text feature [kinase] present in test data point [True]
38 Text feature [observed] present in test data point [True]
39 Text feature [may] present in test data point [True]
40 Text feature [harbor] present in test data point [True]
41 Text feature [18] present in test data point [True]
42 Text feature [12] present in test data point [True]
43 Text feature [reported] present in test data point [True]
44 Text feature [however] present in test data point [True]
45 Text feature [mutation] present in test data point [True]
46 Text feature [one] present in test data point [True]
47 Text feature [months] present in test data point [True]
48 Text feature [different] present in test data point [True]
49 Text feature [mutations] present in test data point [True]
50 Text feature [respectively] present in test data point [True]
51 Text feature [confirmed] present in test data point [True]
52 Text feature [gene] present in test data point [True]
53 Text feature [performed] present in test data point [True]
54 Text feature [identification] present in test data point [True]
55 Text feature [described] present in test data point [True]
56 Text feature [harboring] present in test data point [True]
57 Text feature [small] present in test data point [True]
58 Text feature [table] present in test data point [True]
59 Text feature [16] present in test data point [True]
60 Text feature [another] present in test data point [True]
61 Text feature [detection] present in test data point [True]
62 Text feature [previously] present in test data point [True]
63 Text feature [found] present in test data point [True]
64 Text feature [19] present in test data point [True]
65 Text feature [median] present in test data point [True]
```

```
66 Text feature [demonstrated] present in test data point [True]
67 Text feature [17] present in test data point [True]
68 Text feature [pcr] present in test data point [True]
69 Text feature [20] present in test data point [True]
70 Text feature [treated] present in test data point [True]
71 Text feature [although] present in test data point [True]
72 Text feature [present] present in test data point [True]
73 Text feature [novel] present in test data point [True]
74 Text feature [similar] present in test data point [True]
75 Text feature [well] present in test data point [True]
76 Text feature [higher] present in test data point [True]
77 Text feature [two] present in test data point [True]
78 Text feature [single] present in test data point [True]
79 Text feature [tissue] present in test data point [True]
80 Text feature [discussion] present in test data point [True]
81 Text feature [case] present in test data point [True]
82 Text feature [advanced] present in test data point [True]
83 Text feature [chronic] present in test data point [True]
84 Text feature [number] present in test data point [True]
85 Text feature [positive] present in test data point [True]
86 Text feature [complete] present in test data point [True]
87 Text feature [approved] present in test data point [True]
88 Text feature [studies] present in test data point [True]
89 Text feature [samples] present in test data point [True]
90 Text feature [62] present in test data point [True]
91 Text feature [due] present in test data point [True]
93 Text feature [overall] present in test data point [True]
94 Text feature [three] present in test data point [True]
95 Text feature [clinically] present in test data point [True]
96 Text feature [cases] present in test data point [True]
97 Text feature [total] present in test data point [True]
98 Text feature [sequencing] present in test data point [True]
Out of the top  100  features  84 are present in query point
```

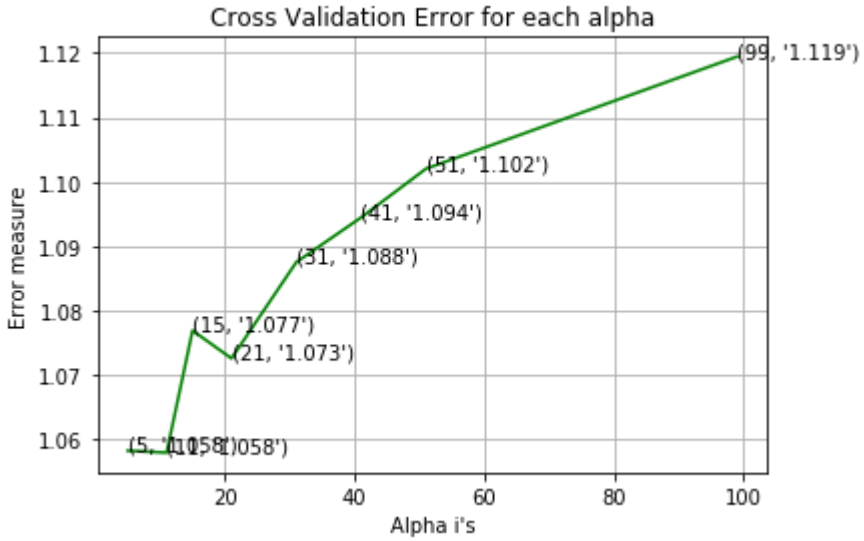
4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [102]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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


# 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 : 1.0582459242255784
for alpha = 11
Log Loss : 1.0579427663391208
for alpha = 15
Log Loss : 1.076898953532393
for alpha = 21
Log Loss : 1.0725700152579816
for alpha = 31
Log Loss : 1.0875091676128672
for alpha = 41
Log Loss : 1.09444953427156
for alpha = 51
Log Loss : 1.102014333634802
for alpha = 99
Log Loss : 1.1194267408852046
```

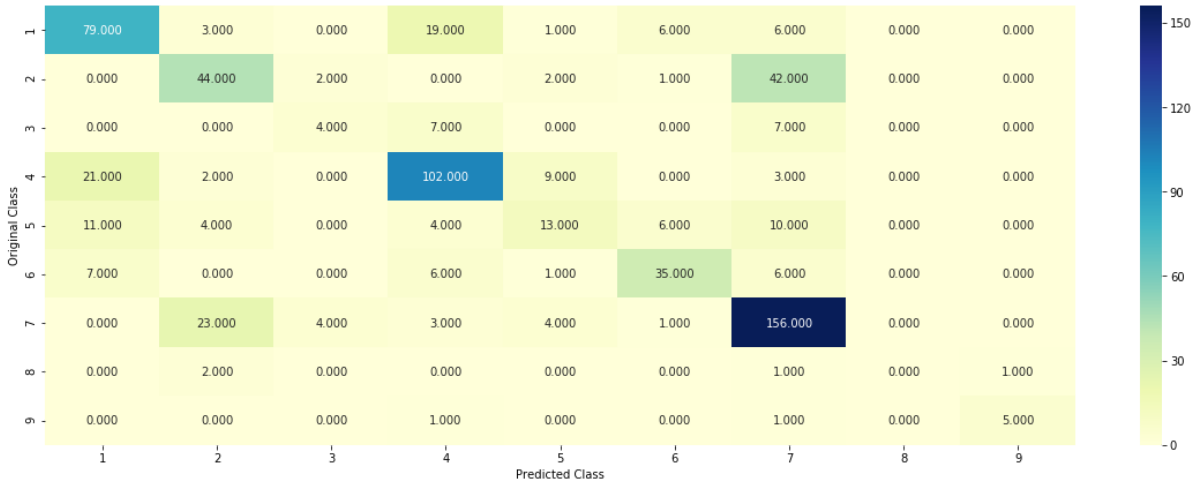



```
For values of best alpha =  11 The train log loss is: 0.6492554707867186
For values of best alpha =  11 The cross validation log loss is: 1.0579427663
391208
For values of best alpha =  11 The test log loss is: 0.9886585809459049
```

4.2.2. Testing the model with best hyper paramters

In [103]:
```
# 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, test_x_responseCoding, test_y, clf)
```

```
Log loss : 0.9886585809459049
Number of mis-classified points : 0.34135338345864663
-------------------- Confusion matrix --------------------
```




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


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

4.2.3.Sample Query point -1

In [63]:
```
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 : 7
The  11  nearest neighbours of the test points belongs to classes [7 7 7 7 7
7 7 7 7 2 2]
Fequency of nearest points : Counter({7: 9, 2: 2})
```

4.2.4. Sample Query Point-2

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

test_point_index = 5

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

4.3. Logistic Regression

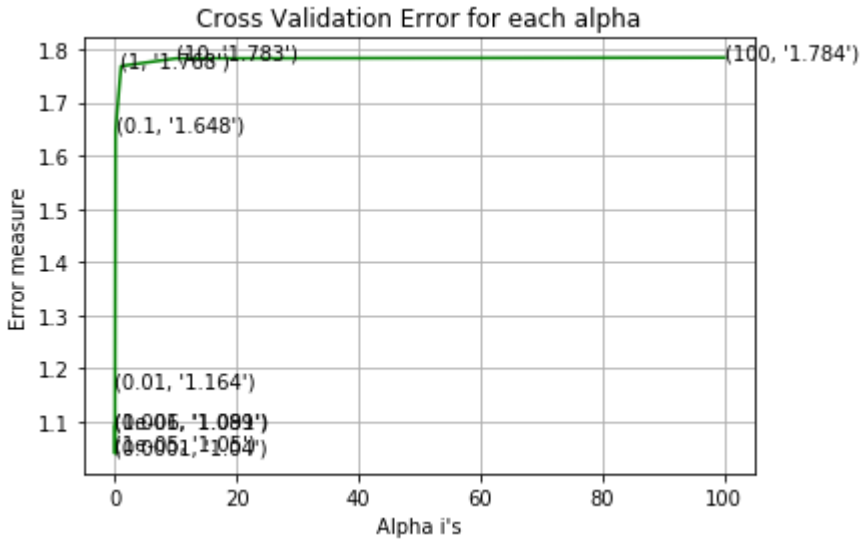
4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [104]:
```
# 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 : 1.0914491187749007
for alpha = 1e-05
Log Loss : 1.049706811580974
for alpha = 0.0001
Log Loss : 1.0396964861215026
for alpha = 0.001
Log Loss : 1.0893686813836045
for alpha = 0.01
Log Loss : 1.1637505157598274
for alpha = 0.1
Log Loss : 1.6476310666813305
for alpha = 1
Log Loss : 1.7681793697081507
for alpha = 10
Log Loss : 1.782553230134367
for alpha = 100
Log Loss : 1.7840827605565979
```

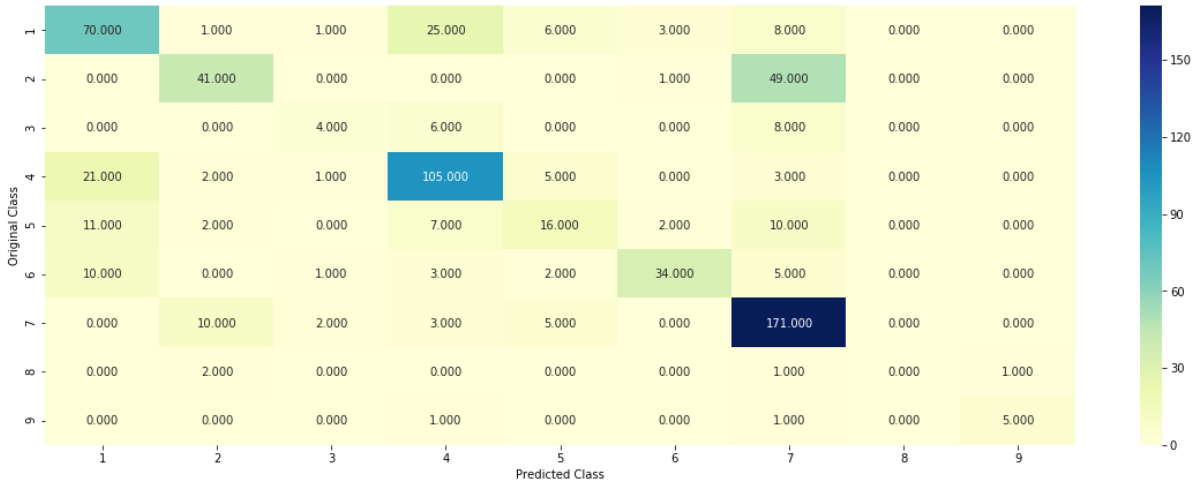



```
For values of best alpha =  0.0001 The train log loss is: 0.414428606953632
For values of best alpha =  0.0001 The cross validation log loss is: 1.039696
4861215026
For values of best alpha =  0.0001 The test log loss is: 0.9860566892998761
```

4.3.1.2. Testing the model with best hyper paramters

In [105]:
```
# 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, test_x_onehotCoding, test_y, clf)
```

```
Log loss : 0.9860566892998761
Number of mis-classified points : 0.3293233082706767
-------------------- Confusion matrix --------------------
```




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


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

4.3.1.3. Feature Importance

In [68]:
```
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 [70]:
```
# 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 = 11
no_feature = 300
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 : 4
Predicted Class Probabilities: [[1.54e-01 1.23e-02 6.60e-03 8.06e-01 9.50e-03
6.10e-03 3.90e-03 9.00e-04
  8.00e-04]]
Actual Class : 4
--------------------------------------------------
142 Text feature [inactivating] present in test data point [True]
147 Text feature [suppressor] present in test data point [True]
191 Text feature [inactivation] present in test data point [True]
218 Text feature [stabilization] present in test data point [True]
229 Text feature [bind] present in test data point [True]
237 Text feature [subcellular] present in test data point [True]
240 Text feature [spectra] present in test data point [True]
263 Text feature [consequence] present in test data point [True]
270 Text feature [asp] present in test data point [True]
295 Text feature [nonsense] present in test data point [True]
Out of the top  300  features  10 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [71]:
```
test_point_index = 5
no_feature = 300
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)
```

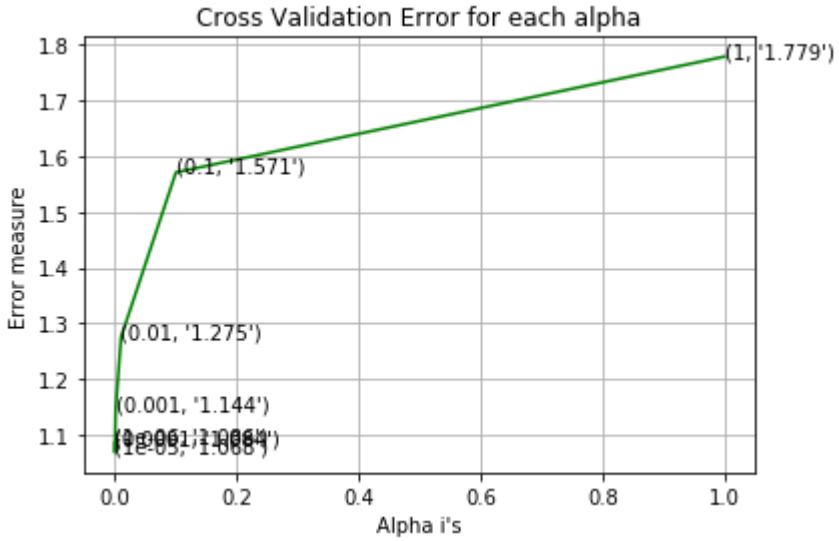
```
Predicted Class : 7
Predicted Class Probabilities: [[1.000e-03 4.321e-01 1.200e-03 4.000e-04 3.50
0e-03 1.920e-02 5.419e-01
  4.000e-04 2.000e-04]]
Actual Class : 6
--------------------------------------------------
53 Text feature [cross] present in test data point [True]
56 Text feature [nude] present in test data point [True]
62 Text feature [downstream] present in test data point [True]
65 Text feature [adenocarcinomas] present in test data point [True]
88 Text feature [activated] present in test data point [True]
98 Text feature [doses] present in test data point [True]
112 Text feature [overexpression] present in test data point [True]
129 Text feature [enhance] present in test data point [True]
130 Text feature [constitutive] present in test data point [True]
131 Text feature [nf] present in test data point [True]
132 Text feature [virus] present in test data point [True]
138 Text feature [untreated] present in test data point [True]
139 Text feature [rarely] present in test data point [True]
148 Text feature [activation] present in test data point [True]
156 Text feature [receptors] present in test data point [True]
161 Text feature [transformed] present in test data point [True]
162 Text feature [remain] present in test data point [True]
167 Text feature [ligand] present in test data point [True]
185 Text feature [lipid] present in test data point [True]
196 Text feature [enhanced] present in test data point [True]
201 Text feature [intrinsic] present in test data point [True]
202 Text feature [3t3] present in test data point [True]
216 Text feature [independently] present in test data point [True]
244 Text feature [pediatric] present in test data point [True]
247 Text feature [overcome] present in test data point [True]
254 Text feature [term] present in test data point [True]
259 Text feature [lymphomas] present in test data point [True]
279 Text feature [91] present in test data point [True]
280 Text feature [month] present in test data point [True]
288 Text feature [heterogeneous] present in test data point [True]
295 Text feature [inhibitor] present in test data point [True]
297 Text feature [extent] present in test data point [True]
Out of the top  300  features  32 are present in query point
```

4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

In [106]:
```
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 : 1.0857713172441734
for alpha = 1e-05
Log Loss : 1.0681720993905723
for alpha = 0.0001
Log Loss : 1.084079097351972
for alpha = 0.001
Log Loss : 1.143514987529129
for alpha = 0.01
Log Loss : 1.2754641684278205
for alpha = 0.1
Log Loss : 1.5711867818815803
for alpha = 1
Log Loss : 1.7790023232702894
```

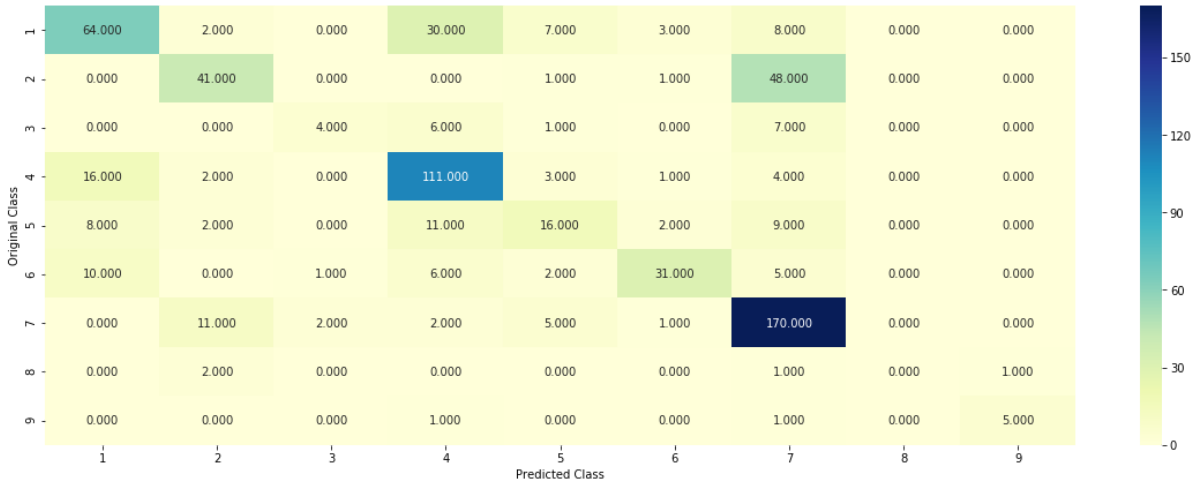



```
For values of best alpha =  1e-05 The train log loss is: 0.46574530399776204
For values of best alpha =  1e-05 The cross validation log loss is: 1.0681720
993905723
For values of best alpha =  1e-05 The test log loss is: 1.0843585331668242
```

4.3.2.2. Testing model with best hyper parameters

In [107]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_
state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, test_x_onehot
Coding, test_y, clf)
```

```
Log loss : 1.0843585331668242
Number of mis-classified points : 0.33533834586466166
-------------------- Confusion matrix --------------------
```




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


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

4.3.2.3. Feature Importance, Correctly Classified point

In [74]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_
state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 11
no_feature = 300
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 : 4
Predicted Class Probabilities: [[1.241e-01 2.390e-02 1.120e-02 7.668e-01 3.64
0e-02 2.450e-02 1.080e-02
  1.700e-03 6.000e-04]]
Actual Class : 4
--------------------------------------------------
80 Text feature [asp] present in test data point [True]
275 Text feature [iii] present in test data point [True]
283 Text feature [spectra] present in test data point [True]
Out of the top  300  features  3 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [75]:
```
test_point_index = 5
no_feature = 300
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)
```

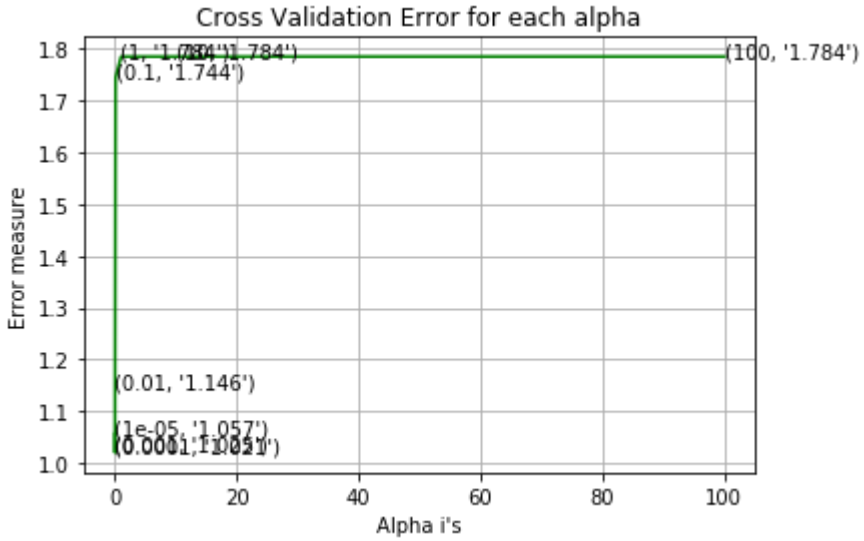
```
Predicted Class : 7
Predicted Class Probabilities: [[2.800e-03 3.743e-01 3.800e-03 1.100e-03 1.12
0e-02 1.940e-02 5.861e-01
  1.300e-03 1.000e-04]]
Actual Class : 6
--------------------------------------------------
54 Text feature [adenocarcinomas] present in test data point [True]
56 Text feature [cross] present in test data point [True]
81 Text feature [remain] present in test data point [True]
84 Text feature [nude] present in test data point [True]
87 Text feature [term] present in test data point [True]
101 Text feature [downstream] present in test data point [True]
112 Text feature [untreated] present in test data point [True]
114 Text feature [overexpression] present in test data point [True]
118 Text feature [independently] present in test data point [True]
144 Text feature [advanced] present in test data point [True]
162 Text feature [rarely] present in test data point [True]
167 Text feature [91] present in test data point [True]
174 Text feature [us] present in test data point [True]
179 Text feature [lipid] present in test data point [True]
183 Text feature [lesions] present in test data point [True]
184 Text feature [metastases] present in test data point [True]
216 Text feature [inhibitor] present in test data point [True]
224 Text feature [heterogeneous] present in test data point [True]
227 Text feature [86] present in test data point [True]
241 Text feature [intrinsic] present in test data point [True]
Out of the top  300  features  20 are present in query point
```

4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [108]:
```
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 : 1.0570465653544852
for C = 0.0001
Log Loss : 1.0205552821213693
for C = 0.001
Log Loss : 1.024986214599474
for C = 0.01
Log Loss : 1.1457297182312518
for C = 0.1
Log Loss : 1.7443270987369555
for C = 1
Log Loss : 1.7844082347530408
for C = 10
Log Loss : 1.7844082284792935
for C = 100
Log Loss : 1.7844082176585756
```

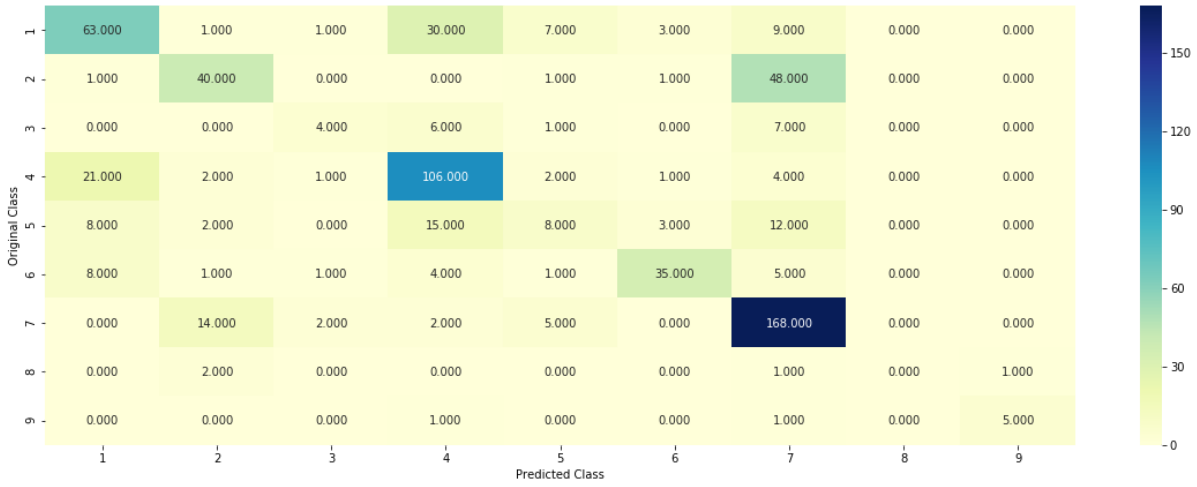



```
For values of best alpha =  0.0001 The train log loss is: 0.4000688840166313
For values of best alpha =  0.0001 The cross validation log loss is: 1.020555
2821213693
For values of best alpha =  0.0001 The test log loss is: 1.0451559188685706
```

4.4.2. Testing model with best hyper parameters

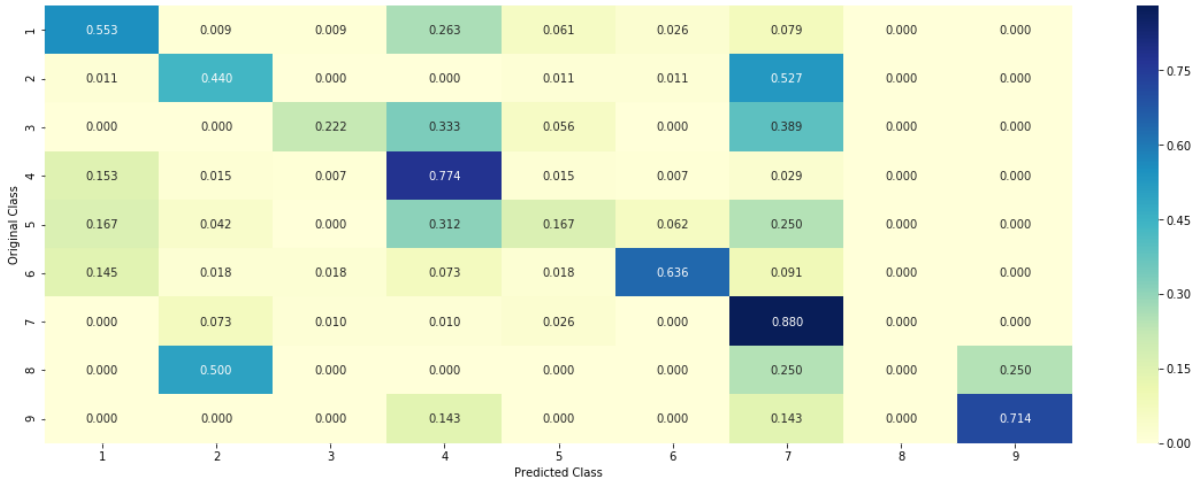
In [109]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', rando
m_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,test_x_onehotC
oding,test_y, clf)
```

```
Log loss : 1.0451559188685706
Number of mis-classified points : 0.3548872180451128
-------------------- 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 [78]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', rando
m_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 11
# test_point_index = 100
no_feature = 300
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 : 4
Predicted Class Probabilities: [[0.1269 0.0377 0.0099 0.7484 0.0472 0.0163 0.
0113 0.0011 0.0012]]
Actual Class : 4
--------------------------------------------------
203 Text feature [stabilization] present in test data point [True]
208 Text feature [inactivating] present in test data point [True]
213 Text feature [bind] present in test data point [True]
215 Text feature [asp] present in test data point [True]
216 Text feature [glu] present in test data point [True]
220 Text feature [consequence] present in test data point [True]
Out of the top  300  features  6 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [79]:
```
test_point_index = 5
no_feature = 300
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: [[1.600e-03 4.272e-01 2.600e-03 1.900e-03 1.88
0e-02 2.730e-02 5.195e-01
  6.000e-04 5.000e-04]]
Actual Class : 6
--------------------------------------------------
120 Text feature [overexpression] present in test data point [True]
121 Text feature [nude] present in test data point [True]
126 Text feature [independently] present in test data point [True]
127 Text feature [heterogeneous] present in test data point [True]
128 Text feature [remain] present in test data point [True]
129 Text feature [downstream] present in test data point [True]
131 Text feature [cross] present in test data point [True]
219 Text feature [adenocarcinomas] present in test data point [True]
222 Text feature [untreated] present in test data point [True]
231 Text feature [treating] present in test data point [True]
233 Text feature [doses] present in test data point [True]
236 Text feature [3t3] present in test data point [True]
239 Text feature [virus] present in test data point [True]
242 Text feature [enhance] present in test data point [True]
253 Text feature [cancers] present in test data point [True]
255 Text feature [activated] present in test data point [True]
261 Text feature [protocol] present in test data point [True]
263 Text feature [overcome] present in test data point [True]
267 Text feature [ligands] present in test data point [True]
270 Text feature [enhanced] present in test data point [True]
272 Text feature [rarely] present in test data point [True]
274 Text feature [91] present in test data point [True]
279 Text feature [intrinsic] present in test data point [True]
282 Text feature [parameters] present in test data point [True]
Out of the top  300  features  24 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

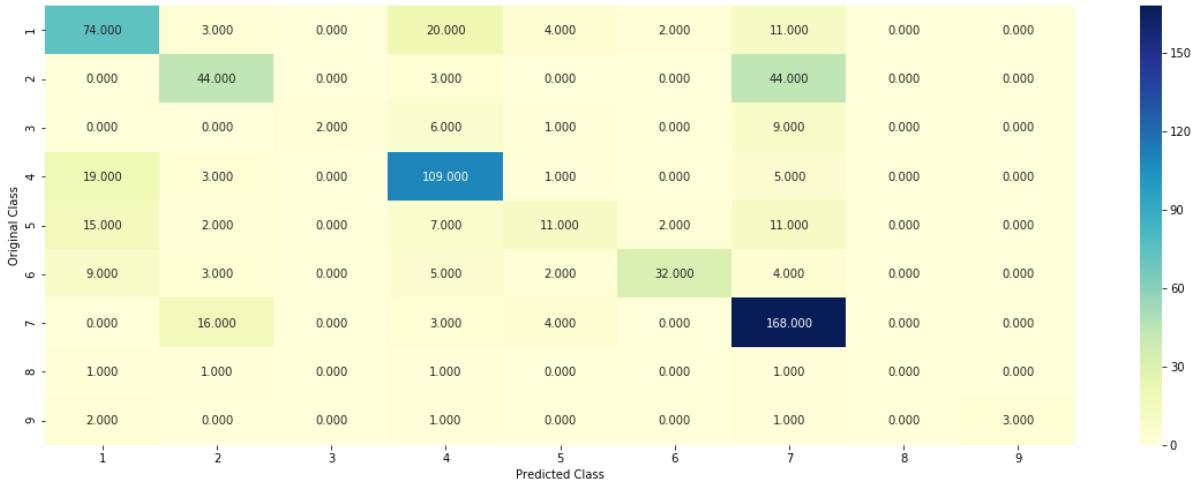
In [110]:
```
alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_dep
th=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_onehotCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.cla
sses_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2150018049530225
for n_estimators = 100 and max depth =  10
Log Loss : 1.1760807589148536
for n_estimators = 200 and max depth =  5
Log Loss : 1.1963578774840096
for n_estimators = 200 and max depth =  10
Log Loss : 1.1595625527864588
for n_estimators = 500 and max depth =  5
Log Loss : 1.1890810814033
for n_estimators = 500 and max depth =  10
Log Loss : 1.161460992507832
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1864263879386292
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1629487705124686
for n_estimators = 2000 and max depth =  5
Log Loss : 1.184558232717106
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1602360970350871
For values of best estimator =  200 The train log loss is: 0.5923182491087172
For values of best estimator =  200 The cross validation log loss is: 1.15956
25527864586
For values of best estimator =  200 The test log loss is: 1.1074529416264431
```

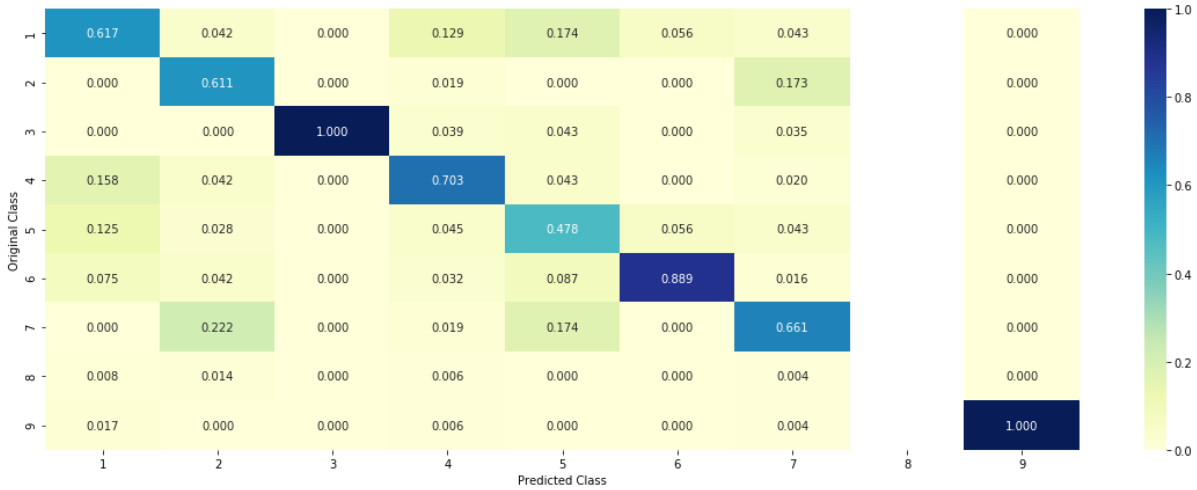
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [111]:
```
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,test_x_onehotC
oding,test_y, clf)
```

```
Log loss : 1.1074529416264431
Number of mis-classified points : 0.33383458646616543
-------------------- 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 [82]:
```
# 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 = 11
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 : 4
Predicted Class Probabilities: [[0.1392 0.0461 0.0183 0.6725 0.0357 0.0303 0.
0494 0.0036 0.0047]]
Actual Class : 4
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
3 Text feature [activated] present in test data point [True]
8 Text feature [phosphorylation] present in test data point [True]
10 Text feature [inhibitors] present in test data point [True]
11 Text feature [suppressor] present in test data point [True]
13 Text feature [missense] present in test data point [True]
14 Text feature [activation] present in test data point [True]
17 Text feature [function] present in test data point [True]
26 Text feature [activating] present in test data point [True]
27 Text feature [protein] present in test data point [True]
28 Text feature [loss] present in test data point [True]
29 Text feature [growth] present in test data point [True]
31 Text feature [nonsense] present in test data point [True]
38 Text feature [cells] present in test data point [True]
40 Text feature [functional] present in test data point [True]
43 Text feature [inhibition] present in test data point [True]
54 Text feature [proteins] present in test data point [True]
61 Text feature [cell] present in test data point [True]
85 Text feature [functions] present in test data point [True]
87 Text feature [activity] present in test data point [True]
96 Text feature [11] present in test data point [True]
97 Text feature [inhibited] present in test data point [True]
98 Text feature [lines] present in test data point [True]
99 Text feature [defective] present in test data point [True]
Out of the top  100  features  24 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [83]:
```
test_point_index = 5
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 : 7
Predicted Class Probabilities: [[0.0673 0.1727 0.0187 0.0361 0.0371 0.0421 0.
6162 0.0041 0.0057]]
Actuall Class : 6
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
3 Text feature [activated] present in test data point [True]
6 Text feature [tyrosine] present in test data point [True]
7 Text feature [constitutive] present in test data point [True]
8 Text feature [phosphorylation] present in test data point [True]
10 Text feature [inhibitors] present in test data point [True]
11 Text feature [suppressor] present in test data point [True]
12 Text feature [treatment] present in test data point [True]
13 Text feature [missense] present in test data point [True]
14 Text feature [activation] present in test data point [True]
15 Text feature [pten] present in test data point [True]
16 Text feature [inhibitor] present in test data point [True]
17 Text feature [function] present in test data point [True]
18 Text feature [erk] present in test data point [True]
21 Text feature [oncogenic] present in test data point [True]
22 Text feature [treated] present in test data point [True]
23 Text feature [activate] present in test data point [True]
24 Text feature [ic50] present in test data point [True]
26 Text feature [activating] present in test data point [True]
27 Text feature [protein] present in test data point [True]
28 Text feature [loss] present in test data point [True]
29 Text feature [growth] present in test data point [True]
32 Text feature [signaling] present in test data point [True]
34 Text feature [months] present in test data point [True]
36 Text feature [clinical] present in test data point [True]
38 Text feature [cells] present in test data point [True]
40 Text feature [functional] present in test data point [True]
43 Text feature [inhibition] present in test data point [True]
49 Text feature [response] present in test data point [True]
50 Text feature [receptor] present in test data point [True]
54 Text feature [proteins] present in test data point [True]
56 Text feature [therapeutic] present in test data point [True]
60 Text feature [akt] present in test data point [True]
61 Text feature [cell] present in test data point [True]
64 Text feature [serum] present in test data point [True]
66 Text feature [advanced] present in test data point [True]
69 Text feature [potential] present in test data point [True]
71 Text feature [mek] present in test data point [True]
75 Text feature [amplification] present in test data point [True]
77 Text feature [repair] present in test data point [True]
81 Text feature [downstream] present in test data point [True]
82 Text feature [drug] present in test data point [True]
85 Text feature [functions] present in test data point [True]
87 Text feature [activity] present in test data point [True]
90 Text feature [transforming] present in test data point [True]
95 Text feature [therapy] present in test data point [True]
96 Text feature [11] present in test data point [True]
97 Text feature [inhibited] present in test data point [True]
98 Text feature [lines] present in test data point [True]
Out of the top  100  features  49 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

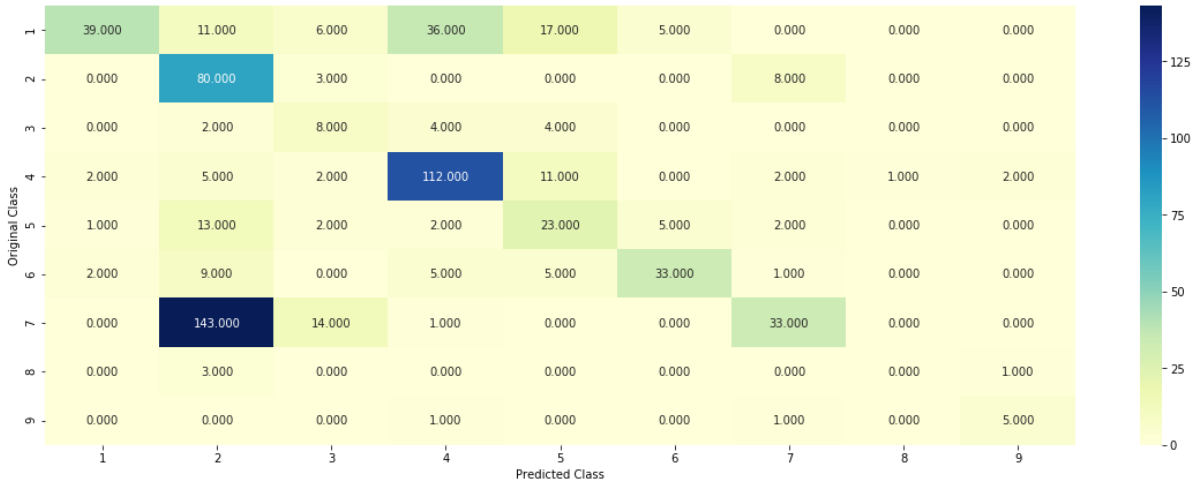
In [112]:
```
alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_dep
th=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.cla
sses_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (features[i],c
v_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

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

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

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.2225779734772373
for n_estimators = 10 and max depth =  3
Log Loss : 1.7205643127841401
for n_estimators = 10 and max depth =  5
Log Loss : 1.4313599952830258
for n_estimators = 10 and max depth =  10
Log Loss : 1.865315410801488
for n_estimators = 50 and max depth =  2
Log Loss : 1.7375381419207148
for n_estimators = 50 and max depth =  3
Log Loss : 1.516699182666602
for n_estimators = 50 and max depth =  5
Log Loss : 1.3321330206327053
for n_estimators = 50 and max depth =  10
Log Loss : 1.6803327946441324
for n_estimators = 100 and max depth =  2
Log Loss : 1.6406706721132633
for n_estimators = 100 and max depth =  3
Log Loss : 1.5708347788634307
for n_estimators = 100 and max depth =  5
Log Loss : 1.3422163832378395
for n_estimators = 100 and max depth =  10
Log Loss : 1.6488195514062143
for n_estimators = 200 and max depth =  2
Log Loss : 1.6307137336295923
for n_estimators = 200 and max depth =  3
Log Loss : 1.5231411902051564
for n_estimators = 200 and max depth =  5
Log Loss : 1.3780173129537336
for n_estimators = 200 and max depth =  10
Log Loss : 1.6362521928538065
for n_estimators = 500 and max depth =  2
Log Loss : 1.6896906921532182
for n_estimators = 500 and max depth =  3
Log Loss : 1.5740439497618555
for n_estimators = 500 and max depth =  5
Log Loss : 1.4000628388519205
for n_estimators = 500 and max depth =  10
Log Loss : 1.6472567423500535
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6568070311872165
for n_estimators = 1000 and max depth =  3
Log Loss : 1.5869373610621338
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3829784042888842
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6206387569050444
For values of best alpha =  50 The train log loss is: 0.05489681179369581
For values of best alpha =  50 The cross validation log loss is: 1.3321330206
327051
For values of best alpha =  50 The test log loss is: 1.3411921240569868
```

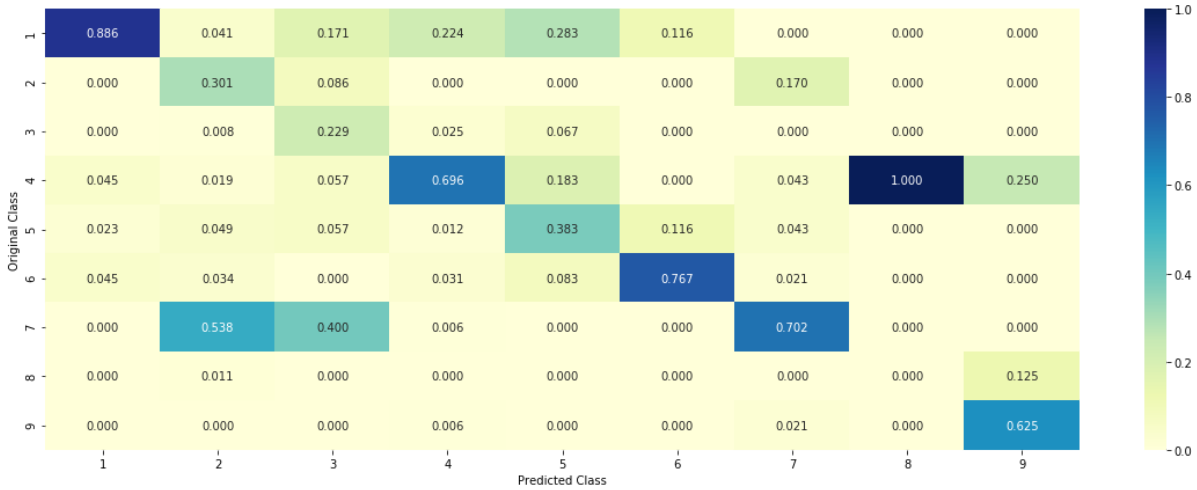
4.5.4. Testing model with best hyper parameters (Response Coding)

In [113]:
```
clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimat
ors=alpha[int(best_alpha/4)], criterion='gini', max_features='auto',random_sta
te=42)
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,test_x_respo
nseCoding,test_y, clf)
```

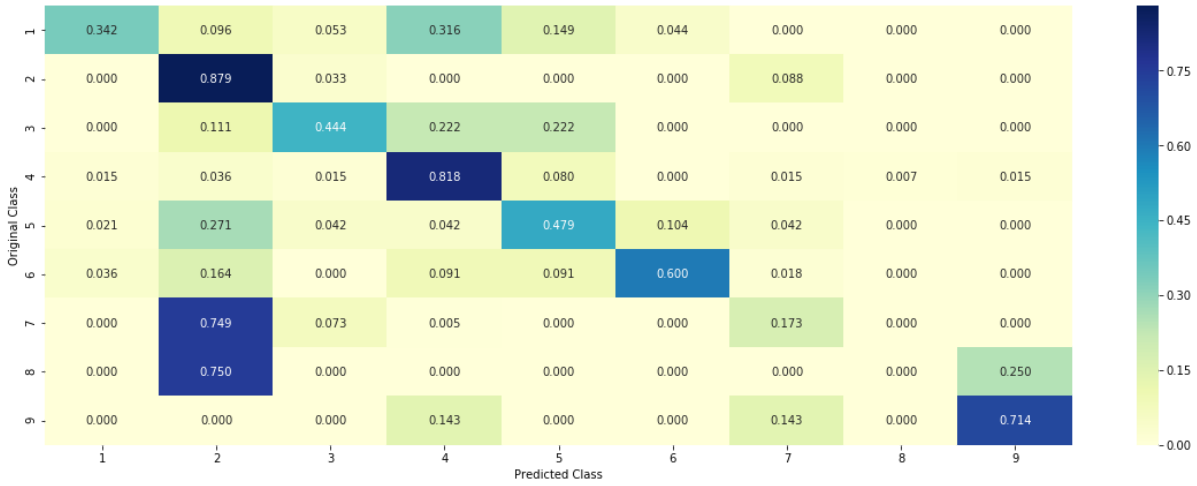
```
Log loss : 1.3411921240569866
Number of mis-classified points : 0.4992481203007519
-------------------- Confusion matrix --------------------
```




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




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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [86]:
```
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 = 11
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 : 4
Predicted Class Probabilities: [[0.0235 0.0235 0.1762 0.698  0.0241 0.0223 0.
0038 0.0127 0.0159]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [87]:
```
test_point_index = 5
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: [[0.0015 0.5003 0.0012 0.0025 0.0008 0.0093 0.
4817 0.0016 0.0012]]
Actual Class : 6
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Variation 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

In [88]:
```
clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='bala
nced', random_state=0)
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

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


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

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

```
Logistic Regression :  Log Loss: 1.09
Support vector machines : Log Loss: 1.78
Naive Bayes : Log Loss: 1.24
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.177
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.028
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.484
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.128
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.259
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.529
```

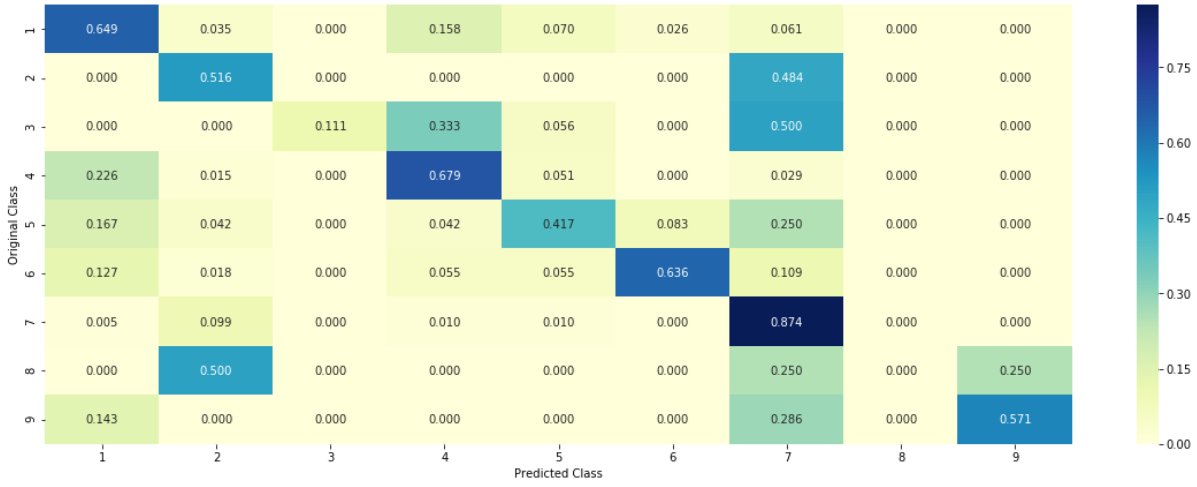
4.7.2 testing the model with the best hyper parameters

In [89]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_cla
ssifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

log_error = log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
print("Log loss (CV) on the stacking classifier :",log_error)

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

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

```
Log loss (train) on the stacking classifier : 0.6505369477610267
Log loss (CV) on the stacking classifier : 1.1281110077906034
Log loss (test) on the stacking classifier : 1.0854815320633013
Number of missclassified point : 0.33533834586466166
-------------------- Confusion matrix --------------------
```


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


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

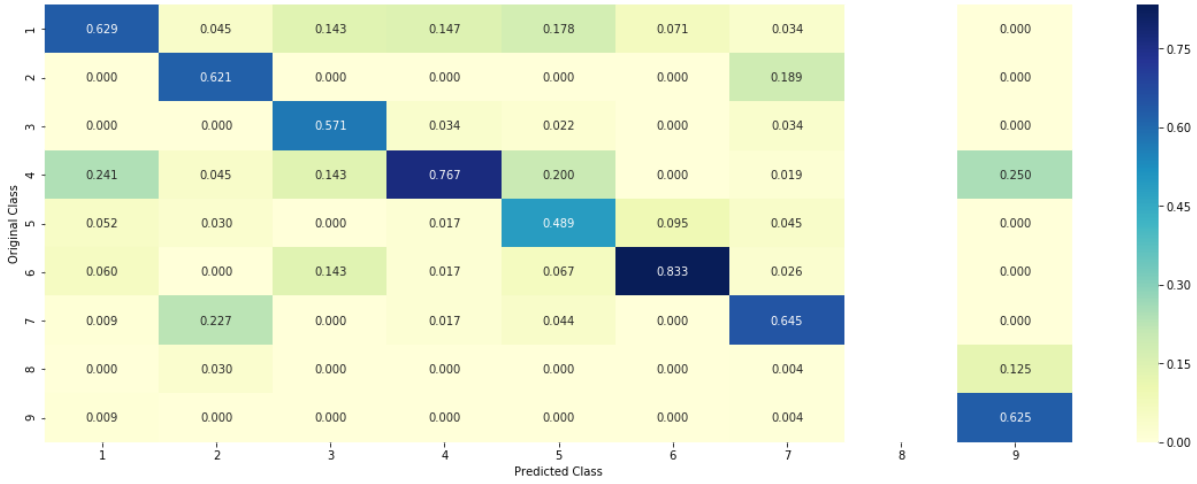


4.7.3 Maximum Voting classifier

In [90]:
```
#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 : 0.8707642892936034
Log loss (CV) on the VotingClassifier : 1.1697311290457197
Log loss (test) on the VotingClassifier : 1.1224887055347923
Number of missclassified point : 0.3383458646616541
-------------------- Confusion matrix --------------------
```


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




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

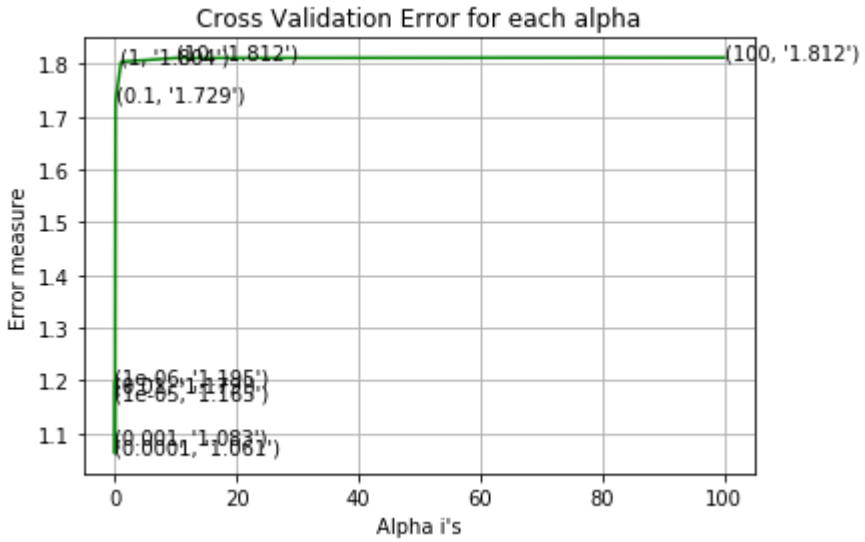
Logistic Regression

In [114]:
```
text_vectorizer = CountVectorizer(ngram_range=(1,1),min_df=5, max_features=100
00)
train_text_feature_unigram = text_vectorizer.fit_transform(train_df['TEXT'])

train_text_feature_unigram = normalize(train_text_feature_unigram, axis=0)

# we use the same vectorizer that was trained on train data
test_text_feature_unigram = text_vectorizer.transform(test_df['TEXT'])
# don't forget to normalize every feature
test_text_feature_unigram = normalize(test_text_feature_unigram, axis=0)

# we use the same vectorizer that was trained on train data
cv_text_feature_unigram = text_vectorizer.transform(cv_df['TEXT'])
# don't forget to normalize every feature
cv_text_feature_unigram = normalize(cv_text_feature_unigram, axis=0)

train_gene_var_unigram = hstack((train_gene_feature_onehotCoding,train_variati
on_feature_onehotCoding))
test_gene_var_unigram = hstack((test_gene_feature_onehotCoding,test_variation_
feature_onehotCoding))
cv_gene_var_unigram = hstack((cv_gene_feature_onehotCoding,cv_variation_featur
e_onehotCoding))

train_x_unigram = hstack((train_gene_var_unigram, train_text_feature_unigram))
.tocsr()
train_y = np.array(list(train_df['Class']))

test_x_unigram = hstack((test_gene_var_unigram, test_text_feature_unigram)).to
csr()
test_y = np.array(list(test_df['Class']))

cv_x_unigram = hstack((cv_gene_var_unigram, cv_text_feature_unigram)).tocsr()
cv_y = np.array(list(cv_df['Class']))
```

In [117]:
```
text_vectorizer = CountVectorizer(ngram_range=(1,2),min_df=5, max_features=100
00)
train_text_feature_bigram = text_vectorizer.fit_transform(train_df['TEXT'])

train_text_feature_bigram = normalize(train_text_feature_bigram, axis=0)

# we use the same vectorizer that was trained on train data
test_text_feature_bigram = text_vectorizer.transform(test_df['TEXT'])
# don't forget to normalize every feature
test_text_feature_bigram = normalize(test_text_feature_bigram, axis=0)

# we use the same vectorizer that was trained on train data
cv_text_feature_bigram = text_vectorizer.transform(cv_df['TEXT'])
# don't forget to normalize every feature
cv_text_feature_bigram = normalize(cv_text_feature_bigram, axis=0)

train_gene_var_bigram = hstack((train_gene_feature_onehotCoding,train_variatio
n_feature_onehotCoding))
test_gene_var_bigram = hstack((test_gene_feature_onehotCoding,test_variation_f
eature_onehotCoding))
cv_gene_var_bigram = hstack((cv_gene_feature_onehotCoding,cv_variation_feature
_onehotCoding))

train_x_bigram = hstack((train_gene_var_bigram, train_text_feature_bigram)).to
csr()
train_y = np.array(list(train_df['Class']))

test_x_bigram = hstack((test_gene_var_bigram, test_text_feature_bigram)).tocsr
()
test_y = np.array(list(test_df['Class']))

cv_x_bigram = hstack((cv_gene_var_bigram, cv_text_feature_bigram)).tocsr()
cv_y = np.array(list(cv_df['Class']))
```

In [118]:
```
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_unigram, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_unigram, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_unigram)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes
_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probab
ility estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

predict_y = sig_clf.predict_proba(train_x_unigram)
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_unigram)
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_unigram)
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))
```

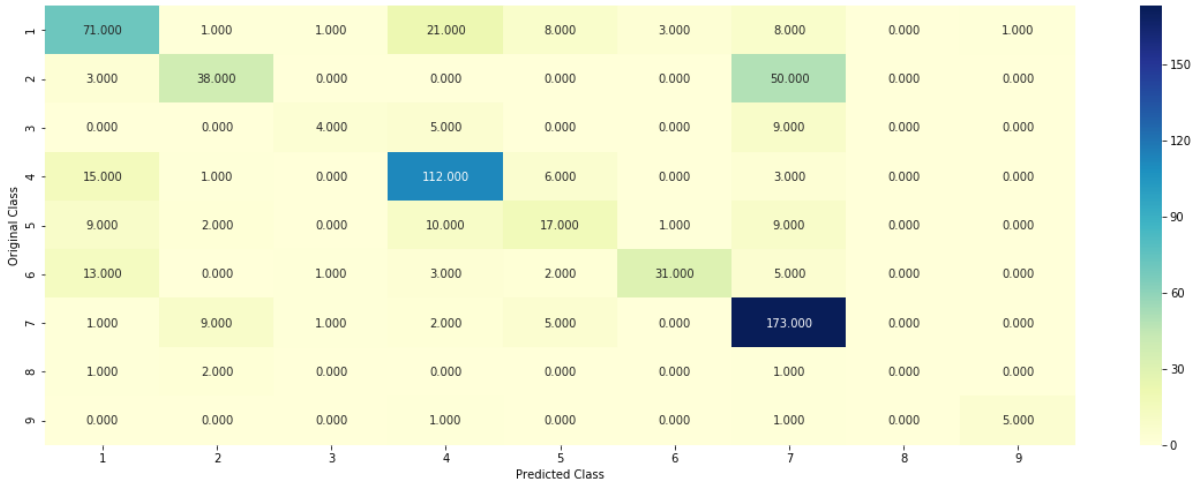
```
for alpha = 1e-06
Log Loss : 1.1949383643558178
for alpha = 1e-05
Log Loss : 1.165489778569634
for alpha = 0.0001
Log Loss : 1.0610850670679761
for alpha = 0.001
Log Loss : 1.0828092303937074
for alpha = 0.01
Log Loss : 1.1786977642016547
for alpha = 0.1
Log Loss : 1.7293653389622106
for alpha = 1
Log Loss : 1.8043583970906669
for alpha = 10
Log Loss : 1.8116113961676308
for alpha = 100
Log Loss : 1.8123467625178002
```




```
For values of best alpha =  0.0001 The train log loss is: 0.44632183198180686
For values of best alpha =  0.0001 The cross validation log loss is: 1.061085
0670679761
For values of best alpha =  0.0001 The test log loss is: 0.9744348565967921
```

In [119]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty=
'l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_unigram, train_y, test_x_unigram, te
st_y, clf)
```

```
Log loss : 0.9744348565967921
Number of mis-classified points : 0.3218045112781955
-------------------- Confusion matrix --------------------
```

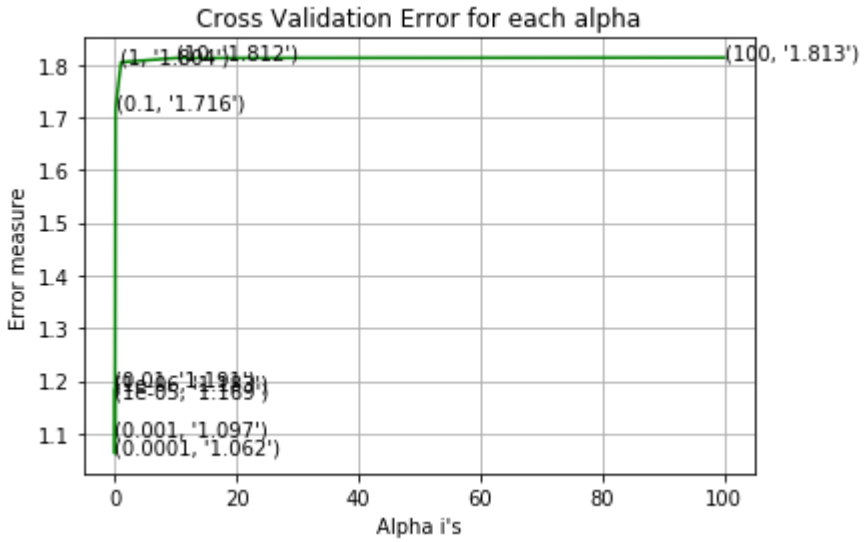



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```


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

In [120]:
```
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_bigram, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_bigram, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_bigram)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes
_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probab
ility estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

predict_y = sig_clf.predict_proba(train_x_bigram)
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_bigram)
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_bigram)
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))
```

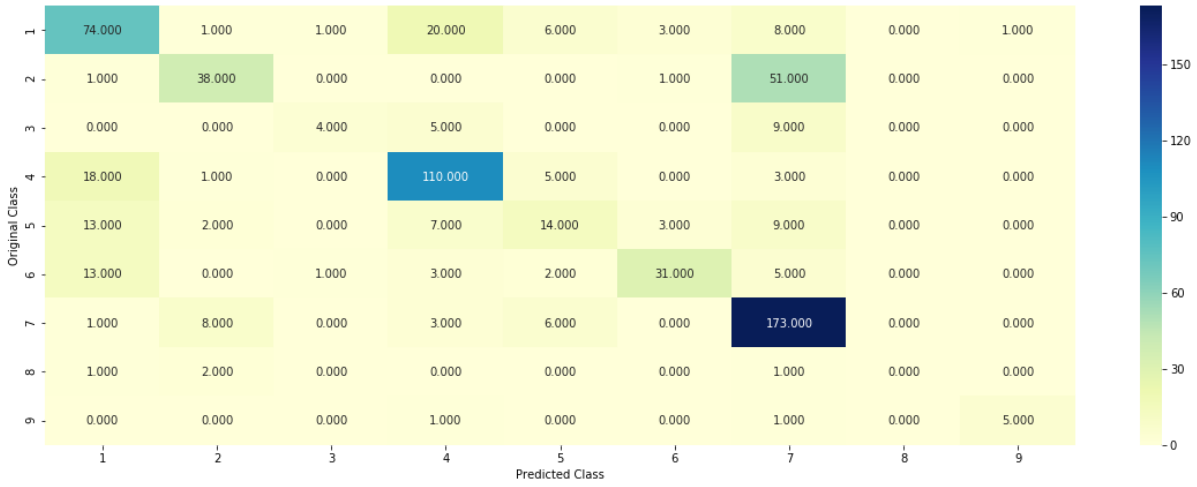
```
for alpha = 1e-06
Log Loss : 1.182867276805894
for alpha = 1e-05
Log Loss : 1.1690440227982835
for alpha = 0.0001
Log Loss : 1.0619137518668915
for alpha = 0.001
Log Loss : 1.0966656377534192
for alpha = 0.01
Log Loss : 1.19131272270934
for alpha = 0.1
Log Loss : 1.716401732973238
for alpha = 1
Log Loss : 1.8043443836645345
for alpha = 10
Log Loss : 1.8121772333911041
for alpha = 100
Log Loss : 1.8129913612599509
```




```
For values of best alpha =  0.0001 The train log loss is: 0.44273374532206616
For values of best alpha =  0.0001 The cross validation log loss is: 1.061913
7518668915
For values of best alpha =  0.0001 The test log loss is: 0.9905961612433127
```

In [121]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty=
'l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_bigram, train_y, test_x_bigram, test
_y, clf)
```

```
Log loss : 0.9905961612433127
Number of mis-classified points : 0.324812030075188
-------------------- Confusion matrix --------------------
```




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


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

In [3]:
```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable
x = PrettyTable()
columns = ["Model","Train log-loss","CV log-loss","Test log-loss"]
x.add_column(columns[0],["Naive Bayes","KNN","Logistic Regression(Class balanc
ing)","Logistic Regression(no Class balancing)",\
                         "Linear SVM","Random Forest(1-hot encoding)","Random
 Forest(Response Coding)","Maximum Voting",\
                         "Logistic Regression(uni-gram)","Logistic Regression
(bi-gram)"])
x.add_column(columns[1],["0.675","0.649","0.414","0.465","0.400","0.592","0.05
4","0.870","0.446","0.442"])
x.add_column(columns[2],["1.232","1.058","1.039","1.068","1.020","1.159","1.33
2","1.169","1.061","1.061"])
x.add_column(columns[3],["1.175","0.988","0.986","1.084","1.045","1.107","1.34
1","1.122","0.974","0.990"])
print(x)
```

```
+----------------------------------------+---------------+-------------+---
------------+
|                 Model                  | Train log-loss | CV log-loss | Te
st log-loss |
+----------------------------------------+---------------+-------------+---
------------+
|              Naive Bayes               |     0.675      |    1.232    |
1.175     |
|                  KNN                   |     0.649      |    1.058    |
0.988     |
|   Logistic Regression(Class balancing) |     0.414      |    1.039    |
0.986     |
| Logistic Regression(no Class balancing) |     0.465      |    1.068    |
1.084     |
|               Linear SVM               |     0.400      |    1.020    |
1.045     |
|      Random Forest(1-hot encoding)     |     0.592      |    1.159    |
1.107     |
|      Random Forest(Response Coding)    |     0.054      |    1.332    |
1.341     |
|             Maximum Voting             |     0.870      |    1.169    |
1.122     |
|      Logistic Regression(uni-gram)     |     0.446      |    1.061    |
0.974     |
|       Logistic Regression(bi-gram)     |     0.442      |    1.061    |
0.990     |
+----------------------------------------+---------------+-------------+---
------------+
```

Conclusion:

1.After trying bi,3,4,5-grams TfidfVectorization for text feature, 4-grams model is concluded as the best due to its low log-loss.

2.Under 4-grams, KNN, LR, Linear SVM has low log-loss and these models doesn't seem to overfit either.

3.By using uni,bi-gram for Logistic Regression, we were able to get the lowest log-loss without the risk of overfitting.

5. Assignments

1. Apply All the models with tf-idf features (Replace CountVectorizer with tfidfVectorizer and run the same cells)
2. Instead of using all the words in the dataset, use only the top 1000 words based of tf-idf values
3. Apply Logistic regression with CountVectorizer Features, including both unigrams and bigrams
4. Try any of the feature engineering techniques discussed in the course to reduce the CV and test log-loss to a value less than 1.0