Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

1. https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25

2. https://www.youtube.com/watch?v=UwbuW7oK8rk
3. https://www.youtube.com/watch?v=qxXRKVompI8

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

ID,Text
0||Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity revealed. Previous work has shown that CDK10 silencing increases ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The precise mechanisms by which CDK10 modulates ETS2 activity, and more generally the functions of CDK10, remain elusive. Here we demonstrate that CDK10 is a cyclin-dependent kinase by identifying cyclin M as an activating cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR syndrome, a human developmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malformations. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and in conferring tamoxifen resistance to breast cancer cells. CDK10/cyclin M phosphorylates ETS2 in vitro, and in cells it positively controls ETS2 degradation by the proteasome. ETS2 protein levels are increased in cells derived from a STAR patient, and this increase is attributable to decreased cyclin M levels. Altogether, our results reveal an additional regulatory mechanism for ETS2, which plays key roles in cancer and development. They also shed light on the molecular mechanisms underlying STAR syndrome.Cyclin-dependent kinases (CDKs) play a pivotal role in the control of a number of fundamental cellular processes (1). The human genome contains 21 genes encoding proteins that can be considered as members of the CDK family owing to their sequence similarity with bona fide CDKs, those known to be activated by cyclins (2). Although discovered almost 20 y

ago (3, 4), CDK10 remains one of the two CDKs without an identified cyclin partner. This knowledge gap has largely impeded the exploration of its biological functions. CDK10 can act as a positive cell cycle regulator in some cells (5, 6) or as a tumor suppressor in others (7, 8). CDK10 interacts with the ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity through an unknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK pathway, and induces resistance of MCF7 cells to tamoxifen (6). ...

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
```

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

from mlxtend.classifier import StackingClassifier

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

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:
```
data = pd.read_csv('training_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

	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 t
he 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_tim
e, "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 : 182.57717181207636 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

	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]
result.TEXT[1]
```

Out[9]:
```
'abstract background non small cell lung cancer nsclc heterogeneous gro
up disorders number genetic proteomic alterations c cbl e3 ubiquitin li
gase adaptor molecule important normal homeostasis cancer determined ge
netic variations c cbl relationship receptor tyrosine kinases egfr met
functionality nsclc methods findings using archival formalin fixed para
ffin embedded ffpe extracted genomic dna show c cbl mutations occur som
atic fashion lung cancers c cbl mutations mutually exclusive met egfr m
utations however independent p53 kras mutations normal tumor pairwise a
nalysis significant loss heterozygosity loh c cbl locus 22 n 8 37 none
samples revealed mutation remaining copy c cbl c cbl loh also positivel
y correlated egfr met mutations observed samples using select c cbl som
atic mutations s80n h94y q249e w802 obtained caucasian taiwanese africa
n american samples respectively transfected nsclc cell lines increased
cell viability cell motility conclusions taking overall mutation rate c
cbl combination somatic missense mutation loh clear c cbl highly mutate
d lung cancers may play essential role lung tumorigenesis metastasis go
introduction us alone year approximately 219 400 people diagnosed lung
cancers 145 000 succumb disease 1 number roughly equivalent combined mo
rtality rates cancers breast prostate colon liver kidney melanoma 1 add
ition prognosis usually poor five year survival rate less 15 also signi
ficant ethnic differences lung cancer outcome worse blacks compared whi
```

tes gender differences also striking women significantly better prognos
is compared men number genetic alterations occur lung cancer example ns
clc mutations kras p53 egfr met identified many pathways especially rec
eptor tyrosine kinases rtks controlled c cbl cbl casitas b lineage lymp
homa mammalian gene located human chromosome 11q23 3 2 involved cell si
gnaling protein ubiquitination 3 cbl proteins belong ring finger class
ubiquitin ligases e3 three homologues c cbl cbl b cbl 3 4 c cbl cbl b g
enes ubiquitously expressed highest levels hematopoietic tissues 5 c cb
l consists four regions encoding functionally distinct protein domains
n terminal tyrosine kinase binding tkb domain linker region catalytic r
ing finger domain proline rich region c terminal ubiquitin associated u
ba domain also overlaps leucine zipper lz domain 3 tkb ring finger doma
ins essential ligand induced ubiquitination rtks 6 7 8 9 ring finger do
main required recruitment e2 ubiquitin conjugating enzymes tkb domain i
ncludes four helix bundle 4h calcium biding ef hand modified sh2 domain
binds phosphotyrosine residues 3 10 11 12 addition proline rich region
c cbl associate sh3 domain grb2 indirectly recruit c cbl rtks via grb2
adaptor protein 7 13 14 c cbl also binds egfr acts e3 targets egfr ubiq
uitination degradation furthermore cbl desensitizes egf signaling oppos
es cellular proliferation induced egf 15 egf activation also appears ac
tivate tyrosine kinase src phosphorylates c cbl turn activates ubiquiti
nation degradation egfr 16 17 18 recent study shows defective endocytos
is egfr characterized deletion mutant point mutation l858r whereby asso
ciation c cbl subsequent ubiquitination impaired 19 recently first huma
n c cbl mutations reported acute myeloid leukemia aml patients 20 mutat
ion r420q inhibits fms like tyrosine kinase 3 flt3 internalization ubiq
uitination 20 e3 activity important oncogenesis c cbl dual separate fun
ction signal transduction molecule previously shown c cbl important bin
ding crkl bcr abl hematopoietic cells also bind modulate functions cyto
skeleton binding proteins like talin paxillin tkb domain important bind
ing number molecules function signal transduction given critical role c
bl normal homeostasis cancer hypothesized might mutated lung cancers st
udy report novel c cbl somatic mutations s80n h94y q249e w802 caucasian
taiwanese african american lung cancer patients respectively expressing
mutations nsclc cell lines lead increased proliferation cell motility s
how c cbl mutations occur without met egfr mutations mutually exclusive
loh c cbl locus additionally c cbl loh associated either met egfr mutat
ions thus hypothesize c cbl mutations might contribute oncogenic potent
ial met egfr lung cancer go methods ethics statement written consent re

search human subjects obtained institutional review board university ch
icago covers research performed laboratory following contact informatio
n institutional review board university chicago mcgiffert hall 5751 woo
dlawn ave 2nd floor chicago il 60637 written informed consents received
patients whose tissue samples used study tissue samples lung cancer tis
sue paired adjacent normal lung tissues obtained 50 caucasian 29 africa
n americans 40 taiwanese nsclc patients recruited university chicago ho
spital chicago usa caucasian african american patients taipei veterans
general hospital taiwan taiwanese patients obtaining appropriate instit
utional review board permission informed consent patients 119 samples 7
7 men 38 women 4 unknown age diagnosis ranging 47 90 years terms tumor
types 53 adenocarcinoma 32 squamous cell carcinoma 34 large cell carcin
oma 49 stage 14 stage ii 34 stage iii 13 stage iv table s1 cell culture
human non small cell lung carcinoma cells a549 h358 maintained dmem rpm
i 1640 respectively human embryonic kidney 293t cells cultured dmem med
ia supplemented 10 fetal bovine serum 100 units ml penicillin 100 g ml
streptomycin invitrogen carlsbad ca cells cultured 37 c humidified incu
bator containing 5 co2 c cbl gene mutational analysis exons 2 16 c cbl
gene individually amplified polymerase chain reaction pcr primers liste
d table s2 pcr conditions 1 cycle 95 c 5 minutes 35 cycles 94 c 30 seco
nds 58 c 30 seconds 72 c 2 minutes one cycle 72 c 10 minutes pcr produc
ts treated exosap usb corporation cleveland oh sequenced big dye termin
ator chemistry applied biosystems foster city ca sequencing performed f
orward coding strand confirmation c cbl alterations performed sequencin
g reverse strand well chromatograms analyzed mutations using mutation s
urveyor v2 61 softgenetics state college pa plasmid constructs site dir
ected mutagenesis wild type c cbl cdna insert subcloned paltermax expre
ssion vector using xhoi sali restriction enzyme sites promega madison w
i using parental plasmid paltermax c cbl tkb domain double mutation s80
n h94y point mutation q249e c terminal point mutation w802 c cbl create
d using following primers 5 gctggcgctaaagaataacccaccttatatcttagac 3 5 c
taccagatacctaccagtatctccgtactatcttgtc 3 double mutation s80n h94y 5 ctt
tacccgactctttgagccctggtcctctttgc 3 q249e 5 cagctcctcctttggctgattgtctctg
gatggtgatc 3 w802 along complementary primers using quickchange site di
rected mutagenesis xl kit stratagene la jolla ca according manufacturer
instructions constructs confirmed point mutations standard dna sequenci
ng strands loss heterozygosity loh analysis five microsatellites chromo
some 11 3 11q within 200 kb downstream c cbl gene 2 control markers 11p
selected analysis table s3 established microsatellite markers respectiv

e primer sequences selected geneloc database http genecards weizmann ac
il geneloc index shtml weizmann institute science rehovot israel primer
s custom designed forward primer fluorescently labeled 5 end fam pet ne
d vic applied biosystems primer annealing temperatures duplex scores ev
aluated nist primer tools http yellow nist gov 8444 dnaanalysis primert
oolspage national institute standards technology gaithersburg md primer
s verified performing pcr control dna isolated tk6 cells resolving prod
ucts agarose gels bands visualized uv transilluminator genomic dna extr
acted tumor samples paired normal lung tissue primers grouped multiplex
combinations shown table s4 marker d11s929 served internal control chec
k consistency pcrs peaks capillary electrophoresis multiplex pcrs carri
ed volume 10 l contained 1 l genomic dna 20 50 ng 0 5 primer 1 0 total
primer pair 400 dntps 1x pcr buffer containing mgcl2 0 2 u taq dna poly
merase pcr performed abi geneamp 9700 pcr system following conditions 5
min 94 c 30 cycles 30 sec 94 c 1 min 60 c 1 min 72 c 5 min 72 c pcr pro
ducts separated capillary electrophoresis abi 3130xl dna analyzer chrom
atograms analyzed peak scanner 1 0 genemapper 3 7 software applied bios
ystems allelic alterations area peaks produced dna pcr products quantif
ied allele ratio allelic areas calculated tumor paired normal dna sampl
e qloh allelic ratio tumor peaks divided allelic ratio paired normal sa
mple 0 5 2 0 c cbl least one 11q marker least two separate experiments
sample considered allelic imbalance interpreted loh samples evaluated l
east two separate experiments samples showing prospective loh c cbl rep
eated third time included new control marker bax locus data shown chrom
osome 19 verify integrity sample dna transfection c cbl constructs a549
cell line transfected using fugene hd roche nutley nj reagent according
manufacturer instructions eight g plasmid dna containing either insert
empty vector wild type c cbl s80n h94y c cbl q249e c cbl w802 cbl used
transfection 6 well culture plate cells harvested 48 h transfection ana
lyzed expression c cbl knockdown c cbl knockdown performed using lentiv
iral transduction using mission lentiviral transduction particles sigma
aldrich st louis mo per manufacturer instructions briefly 1 105 h358 ce
lls well seeded 6 well plates infected following day c cbl lentiviral s
hrna constructs generate stable c cbl knockdown cell lines cells select
ed 2 days 1 g ml puromycin c cbl levels determined using whole cell lys
ates immunoblotting anti cbl antibody santa cruz biotechnologies santa
cruz ca cell viability assay cells transfected described transfection a
ssay forty eight hours transfection viability cells assessed using tryp
an blue exclusion wound healing assay a549 cells seeded 6 well plates c

ultured 48 h 100 confluent medium changed cells transfected described t
ransfection assay twelve hours transfection straight scratch made acros
s cell layer using 1 ml pipette tip cells gently washed 1 pbs remove ce
llular debris media replaced photographs taken wound region every 12 h
48 h western blot analysis forty eight hours transfection cells collect
ed washed twice 1x pbs lysed ice cold lysis buffer 0 5m tris hcl ph 7 4
1 5 nacl 2 5 deoxycholic acid 10 mm edta 10 np 40 0 5 mm dtt 1 mm pheny
lmethylsulfonyl fluoride 5 g ml leupeptin 10 g ml aprotinin 5 minutes l
ysate centrifuged 13 000 rpm 20 minutes 4 c protein content supernatant
measured total cell lysates 50 g well separated sds page electrophoresi
s gels transferred onto nitrocellulose membranes whatman piscataway nj
membranes blocked 5 non fat dry milk phosphate buffered saline containi
ng tween 20 pbst 1x pbs 0 1 tween 20 1 h room temperature incubated app
ropriate primary antibody 4 c overnight membranes washed three times pb
st probed appropriate horseradish peroxidase hrp conjugated secondary a
ntibody 1 h room temperature membranes washed three times pbst bands vi
sualized using western blot chemiluminescence reagent biorad valencia c
a chemidoc gel documentation system biorad valencia ca antibodies obtai
ned santa cruz biotechnologies used following dilutions c cbl 1 5000 c
met 1 5000 egfr 1 5000 ubiquitin 1 1000 ha 1 5000 actin 1 10 000 flow c
ytometry cell cycle analysis carried flow cytometry approximately 2 106
cells grown media containing 10 fbs cells harvested trypsin edta treatm
ent washed 1x pbs three times fixed ice cold 70 ethanol 2 h cells washe
d cold pbs stained solution containing 25 g ml propidium iodide 200 g m
l rnase 0 1 triton x 100 30 minutes dark cell cycle analysis performed
using guava pca 96 flow cytometer guava technologies millipore billeric
a ubiquitin ligase activity 293t cells maintained culture dmem suppleme
nted 10 fbs 1 penicillin 100 units ml streptomycin 100 g ml transfected
0 2 g egfr pcdna3 2 g ha tagged c cbl constructs indicated using calciu
m phosphate according manufacturer protocol profection promega madison
wi twenty four hours post transfection cells starved overnight dmem sup
plemented 0 5 fbs treated without egf 100 ng ml 15 min cells collected
washed two times ice cold pbs containing 0 2 mm sodium orthovanadate ly
sed ice cold lysis buffer 10 mm tris hcl ph 7 5 150 mm nacl 5 mm edta 1
triton x100 10 glycerol 2 mm sodium orthovanadate protease inhibitors l
ysates cleared debris centrifugation 16 000 g 10 min 4 c egfr immunopre
cipitations performed 200 g cleared lysate using 250 ng rabbit anti egf
r protein g plus sepharose overnight 4 c precipitations washed 5 times
lysis buffer boiling laemmli buffer elutions immunoblotted anti ubiquit

in egfr twenty micrograms cleared lysate immunoblotted c cbl constructs using anti ha statistical analysis mutation rates different groups compared using fisher exact test continuous variables group comparisons performed using analysis variance anova followed sidak adjustment multiple comparisons experiments involving repeated measurements time analyzed using repeated measures anova greenhouse geisser adjustment degrees freedom analyses conducted using stata v10 1 software stata corporation college station tx go results c cbl gene mutations lung cancer investigate role c cbl lung cancer analyzed genomic dna tumor paired normal samples drawn multiple ethnicities lung tumor samples represented caucasians n 50 african americans n 29 taiwanese n 40 lung cancer patients designed 12 pairs primers sequence coding region c cbl gene spans exons 2 16 table s2 identified 8 unique somatic mutations c cbl exons among 8 different patients variation l620f known snp rs2227988 exon 11 also detected importantly eight novel non synonymous mutations confirmed sequencing strands c cbl genomic dna obtained lung tumor samples table 1 moreover none 8 mutations detected corresponding normal tissue indicating somatic mutations four synonymous single nucleotide variations snvs also identified used study table 1 table 1 c cbl mutation analysis 119 lung cancer patient tumor tissues three 8 novel non synonymous mutations located tkb tyrosine kinase binding domain s80n h94y q249e one ring finger domain v391i one proline rich region 72515 72517 del atg three c terminal region w802 r830k a848t c cbl protein figure 1a figure s1 figure 1b show model chromatograms representative samples figure 1 figure 1 c cbl mutations loh non small cell lung cancer 11q loh c cbl gene paired lung tumor normal lung tissue samples taiwanese patients n 37 investigated loh eight 21 6 showed loh c cbl locus chromosome 11 29 samples 78 4 revealed normal allelic contribution microsatellite markers figures 1c c cbl mutations different ethnic groups c cbl double mutant s80n h94y found patient overall mutation rate c cbl lung tumors 6 7 8 119 frequency c cbl mutation highest large cell carcinoma 14 7 5 34 patients followed squamous carcinoma 6 3 2 32 patients least observed adenocarcinoma ad 1 8 1 53 patients although rates statistically significant p 0 292 mutation rates 6 0 among caucasians 0 20 ad 0 10 sq 3 20 lc 13 8 african americans 1 10 ad 1 10 sq 2 9 lc 2 5 0 23 ad 1 12 sq 0 5 lc taiwanese population additionally two taiwanese patients lung cancer one squamous one adenocarcinoma known snp l620f ethnic differences statistically significant however power detect differences low mutations met egfr co associated c cbl alterations since east asians lung cancer higher frequency egfr met m

utations lung tumors 21 22 also determined mutations egfr met taiwanese cohort samples compared results observed c cbl alterations loh mutations 37 samples tested find overlap c cbl mutations c cbl loh figure 2 three c cbl mutants including known l620f snp rs2227988 one samples met mutation n375s egfr mutation l858r among 8 samples loh c cbl locus 5 additional mutation met n375s 2 egfr exon 19 deletion twenty six samples neither c cbl mutation c cbl loh 3 patients c cbl mutation c cbl loh among 26 samples 9 met mutation 8 n375s 1 l211w 13 egfr mutation 7 exon 9 deletion 6 l858r 4 met egfr mutation thus rate met egfr mutations among patients loh c cbl locus 7 8 similar seen patients without c cbl mutation loh 22 26 patients p 0 99 4 patients identifiable mutation c cbl met egfr represented 10 8 37 patients analyzed taiwanese patient cohort conversely 89 2 taiwanese lung cancer patients identifiable mutation either c cbl met egfr combination three genes figure 2 additionally determined p53 kras mutations taiwanese cohorts two p53 1 kras mutation detected single kras mutation overlapped one p53 mutation patient also egfr exon 19 deletion c cbl mutation p53 mutation sample c cbl loh concurrent met n375s mutation thus taiwanese samples analyzed p53 kras mutations c cbl mutations mutually exclusive data shown figure 2 figure 2 c cbl mutations relationship met egfr mutations lung cancer cellular functions c cbl alterations context lung tumorigenesis e3 activity intact mutant c cbl proteins investigate whether different c cbl mutations affect e3 activity egfr chosen model substrate c cbl e3 function c cbl mutants tested enhanced ubiquitination activated egfr similar wild type c cbl protein result demonstrates catalytic activity c cbl mutants impaired egfr substrate figure 3a figure 3 figure 3 ubiquitination viability expression cell cycle analysis various c cbl mutants b effect lung cancer cell viability effect representative c cbl mutant three ethnic backgrounds lung cancer cell viability cell lines determined s80n h94y double mutation q249e w802 identified lung tumor samples obtained caucasian taiwanese african american respectively described methods c cbl wild type wt three mutants expressed cloning paltermax vector a549 cells cells express relatively low basal levels endogenous c cbl data shown transfection efficiency comparable different groups number cells transfected c cbl wild type construct 70 compared control cells transfected empty vector cells transfected s80n h94y q249e w802 c cbl mutant constructs resulted increased number viable cells 132 3 120 8 147 9 higher respectively relative empty vector control transfected cells significantly different wild type construct p 0 022 p 0 049 p 0 008 respectively figure 3b relative lev

els c cbl protein whole cell lysates prepared samples obtained parallel experiment determined c cbl protein levels samples representing untransfected empty vector transfected cells comparable representing c cbl wt three c cbl mutants comparable figure 3c c effect cell cycle investigate increases cell viability different c cbl mutants due increased cellular proliferation cell cycle analysis performed a549 cells transfected c cbl wt three different mutants s80n h94y q249e w802 empty vector transfectant used control forty eight hours transfection cell cycle analysis performed described materials methods significant change subg1 g1 phase cell cycle among different mutants compared wt construct p 0 64 p 0 40 p 0 28 respectively g2 phase cell cycle showed increase cell numbers three mutants s80n h94y q249e w802 compared wt difference statistically significant p 0 25 figure 3d effect cell motility investigate effect expression three c cbl mutants cell migration carried wound healing assay described materials methods closing scratch wound monitored 0 12 24 36 48 h figure 4a samples represented cells transfected mutants wound gap much smaller seen sample represented cells transfected c cbl wt p 0 001 also determined rate wound closure five groups 48 h wild type c cbl transfectants showed 61 1 open wound s80n h94y q249e w802 mutants showed 18 7 23 9 34 3 open wound respectively p 0 001 figure 4b figure 4 figure 4 c cbl mutations affect wound healing a549 cells e c cbl knockdown increases cell viability hypothesized loh seen samples could lead decreased expression c cbl thus tested effect c cbl knockdown lung cancer cells compared a549 h358 lung cancer cells express relatively high levels endogenous c cbl data shown c cbl expression knocked using lentiviral construct expressed c cbl specific shrna compared results transduced scrambled shrna results shown figure 5 identified several clones revealed varying degrees c cbl knockdown showing different sets c cbl lentiviral shrna knockdown efficiency figure 5a clones tested clone 27 chosen experiments equal amount cells seeded 6 well plate cell proliferation measured various times results depicted figure 5b expected number cells increased time dependent fashion 100 190 relative scrambled shrna control span 48 h p 0 0002 figure 5b cell cycle phases h358 cells knocked c cbl shrna looked compared scrambled shrna discernable differences two constructs different phases cell cycle data shown figure 5 figure 5 knockdown c cbl using shrna increases cell proliferation go discussion results demonstrate c cbl somatically mutated loh lung cancers significantly contribute enhanced cell viability motility also high prevalence loh respect c cbl lung tumors harbored met egfr mutation present study demonstrat

ed occurrence c cbl mutations lung cancer patients especially different ancestral variations mutations c cbl recently reported juvenile myelomonocytic leukemia myeloid malignancies aml study mutation r420q located junction ring finger linker region inhibited fms like tyrosine kinase 3 flt3 internalization ubiquitination 20 thus contributing gain function rtk addition mutations h398y c384r l380p mapped ring finger domain linker region c cbl required e3 activity 23 24 25 26 27 additionally homozygous mutations ring finger domain c cbl gene described result acquired uniparental disomy upd 26 important note results indicate loh 11q23 locus mutually exclusive missense mutations c cbl somatic mutations heterozygous mutations aml led abrogation e3 activity leading prolonged rtk activation addition mutants located linker region surrounding ring finger domain exhibited enhanced akt signaling response cytokine stimulation 26 addition shown nh3t3 cells neither mutations ring finger linker region causes transformation however certain mutations perturbs ubiquitination others affect receptor recycling prolong kinase activity 28 report c cbl mutations mapped ring finger domain also tkb domain proline rich domain c terminal region none mapped linker region reported aml studies described 23 24 25 26 29 addition 8 mutants detected found different ethnic backgrounds example s80n h94y q249e w802 detected caucasians taiwanese african americans respectively results point difference lung cancer cancers also genetic polymorphism among different races cancer interestingly large disparity african american ethnic populations lung cancer 30 previously shown low frequency egfr met mutation african americans compared taiwanese caucasians 31 study number african american samples analyzed relatively fewer found 3 mutations unique ethnicity would behoove us study genetic alterations occur determine targeted therapeutics african americans results provide evidence importance c cbl tumorigenesis potential signaling prediction based aml data would v391i ring finger domain mutation would affect e3 activity also important determine binding partners c cbl tkb domain proline rich domain mutations previously shown tkb domain bind growth factor receptors important determine cross binding mutants met egfr would also important future look fluorescence situ hybridization copy number changes c cbl lung cancer c cbl plays important role regulating rtk mediated signaling k63 poly ubiquitination subsequent downregulation rtks followed lysosomal degredation 3 mono ubiquitination ubiquitinated k63 linked chains substrates c cbl may lead enhancement biological biochemical functions reviewed hermann et al 2007 32 mutations analyzed studies point fact e3 activity c cbl egfr intac

t egfr levels various mutants remain figure s2 multiple kinases rtks no
n rtks could acted upon c cbl including erbs pdgfr fms met c kit vegfr
flt 1 ron fgfr ir well syk fyn lck fgr lyn c abl 3 lung cancers relevan
t substrates c cbl terms degradation signal transduction yet identified
observation c cbl somatic mutations especially s80n h94y q249e w802 sho
wed increased cell viability cell motility agreement physiological role
cbl regulation apoptosis differentiation identified drosophila signific
ant 33 previously shown activating c cbl mutation downregulates egfr si
gnaling decreases cellular proliferation migration breast cancer cell l
ines 34 although role c cbl negative regulation rtks well substantiated
thereby suggesting natural tumor suppressor studies cancer cells reveal
ed tumor suppressor tumor promoting activities depending type c cbl mut
ation number alleles c cbl locus 24 agreement three c cbl mutants descr
ibed appear tumor growth metastasis promoting properties although mutan
ts outside ring finger linker region c cbl downstream effects significa
nt cause increased proliferation migration substrate affected mutations
known yet raises possibility cellular functions c cbl independent ubiqu
itin ligase activity area currently investigating oncogenic nature rtks
addiction cancers growth signals given clustering c cbl egfr met mutati
ons possible transforming effect c cbl mutations likely combinatorial e
ffect three also show loh c cbl found significant number samples harbor
ed met egfr mutations fact 7 lung tumor samples likely c cbl mutations
additional 22 likely harbor c cbl related loh makes c cbl highly mutate
d molecule lung cancer since loh alone enough cause transforming event
35 36 37 associated mutation met egfr locus yet another rtk discussed m
ay play role carcinogenesis predict loh c cbl results haploinsufficienc
y downplays rtk ubiquitination leading hyperactivity rtks however wheth
er sufficient cause tumorigenesis remains determined consistent hypothe
sis fact c cbl mice increased kinase activity lymphocytes sufficient tu
mor formation 35 36 37 c cbl loh could also lead increased expression c
cbl allele compensate loss allele alternately could form synergy workin
g reduced c cbl levels mutated receptors exacerbate phenotype alone pre
vious studies lab others shown east asians lung cancers relatively high
frequencies gain function mutations rtks egfr met 31 cohort japanese pa
tients activating met mutation identified splice region deletes juxtame
mbrane domain involved e3 activity c cbl 38 study also found activation
met mutually exclusive egfr kras her2 gene mutations 38 failed detect m
utations significant numbers lung tumor samples obtained african americ
ans n 29 caucasian n 50 patients one met mutation identified groups whe

```
reas 1 3 egfr mutations identified african american caucasian cohorts r
espectively egfr mutations earlier identified one key mutations affecti
ng lung adenocarcinoma patients comprehensive study 188 patients 39 stu
dy encompasses different histologies nsclc however published series fin
d mutations c cbl met unlike study encompassed different subtypes nsclc
important note recently shown met mutations lung cancer majority germli
ne 31 reported earlier c cbl mutations small cohort taiwanese lung canc
er samples 40 efforts understand ethnic differences lung oncogenome als
o looked pax transcription factors pax5 pax8 highly expressed lung canc
ers however preferential expression mutations genes lung tumor samples
african americans study show relatively high frequency c cbl mutations
lung cancers especially large cell type among caucasians particularly a
mong african americans therefore propose c cbl efficacious target lung
cancers african americans needs substantiated important prognosis afric
an americans lung cancer especially men much poorer compared caucasian
counterparts 41 conclusion results presented study demonstrate c cbl fr
equently mutated even lost lung cancers results support role c cbl muta
nts independent ubiquitination activity given relatively high mutation
rates c cbl well rtks met egfr likely combined effect could synergistic
promoting tumorigenesis '
```

Feature engineering

In [10]:
```
#http://www.datasciencemadesimple.com/get-string-length-column-datafram
e-python-pandas/
result['TEXT_length'] = result['TEXT'].map(str).apply(len)
result.head(5)

#https://stackoverflow.com/questions/49984905/count-number-of-words-per
-row/49984998
result['Total_Words_in_TEXT'] = result['TEXT'].str.split().str.len()
result.head(5)

result['Gene_length'] = result['Gene'].map(str).apply(len)
result.head(5)

result['Variation_length'] = result['Variation'].map(str).apply(len)
result.head(5)
```

Out[10]:

	ID	Gene	Variation	Class	TEXT	TEXT_length	Total_Words_in_TEXT	Gene
0	0	FAM58A	Truncating Mutations	1	cyclin dependent kinases cdks regulate variety...	30836	4370	6
1	1	CBL	W802*	2	abstract background non small cell lung cancer...	27844	4139	3
2	2	CBL	Q249E	2	abstract background non small cell lung cancer...	27844	4139	3
3	3	CBL	N454D	3	recent evidence demonstrated acquired uniparen...	28093	3841	3
4	4	CBL	L399V	4	oncogenic mutations monomeric casitas b lineag...	31649	4254	3

In [11]:
```
#here combining both cleaned summary text and cleaned review text and c
leaned text length for getting better results

result['len_&_words'] = result['TEXT_length'] + result['Total_Words_in_
```

```
TEXT'] + result['Gene_length'] + result['Variation_length']
result['TEXT']=result['TEXT'] + result['len_&_words'].map(str)
result=result.drop(['TEXT_length','Total_Words_in_TEXT','len_&_words',
'Gene_length','Variation_length'],axis=1)
result.head(3)
```

Out[11]:

	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...

In [12]:
```
result.TEXT[1]
```

Out[12]:
```
'abstract background non small cell lung cancer nsclc heterogeneous gro
up disorders number genetic proteomic alterations c cbl e3 ubiquitin li
gase adaptor molecule important normal homeostasis cancer determined ge
netic variations c cbl relationship receptor tyrosine kinases egfr met
functionality nsclc methods findings using archival formalin fixed para
ffin embedded ffpe extracted genomic dna show c cbl mutations occur som
atic fashion lung cancers c cbl mutations mutually exclusive met egfr m
utations however independent p53 kras mutations normal tumor pairwise a
nalysis significant loss heterozygosity loh c cbl locus 22 n 8 37 none
samples revealed mutation remaining copy c cbl c cbl loh also positivel
y correlated egfr met mutations observed samples using select c cbl som
atic mutations s80n h94y q249e w802 obtained caucasian taiwanese africa
n american samples respectively transfected nsclc cell lines increased
cell viability cell motility conclusions taking overall mutation rate c
cbl combination somatic missense mutation loh clear c cbl highly mutate
d lung cancers may play essential role lung tumorigenesis metastasis go
introduction us alone year approximately 219 400 people diagnosed lung
cancers 145 000 succumb disease 1 number roughly equivalent combined mo
rtality rates cancers breast prostate colon liver kidney melanoma 1 add
ition prognosis usually poor five year survival rate less 15 also signi
ficant ethnic differences lung cancer outcome worse blacks compared whi
tes gender differences also striking women significantly better prognos
```

is compared men number genetic alterations occur lung cancer example ns clc mutations kras p53 egfr met identified many pathways especially rec eptor tyrosine kinases rtks controlled c cbl cbl casitas b lineage lymp homa mammalian gene located human chromosome 11q23 3 2 involved cell si gnaling protein ubiquitination 3 cbl proteins belong ring finger class ubiquitin ligases e3 three homologues c cbl cbl b cbl 3 4 c cbl cbl b g enes ubiquitously expressed highest levels hematopoietic tissues 5 c cb l consists four regions encoding functionally distinct protein domains n terminal tyrosine kinase binding tkb domain linker region catalytic r ing finger domain proline rich region c terminal ubiquitin associated u ba domain also overlaps leucine zipper lz domain 3 tkb ring finger doma ins essential ligand induced ubiquitination rtks 6 7 8 9 ring finger do main required recruitment e2 ubiquitin conjugating enzymes tkb domain i ncludes four helix bundle 4h calcium biding ef hand modified sh2 domain binds phosphotyrosine residues 3 10 11 12 addition proline rich region c cbl associate sh3 domain grb2 indirectly recruit c cbl rtks via grb2 adaptor protein 7 13 14 c cbl also binds egfr acts e3 targets egfr ubiq uitination degradation furthermore cbl desensitizes egf signaling oppos es cellular proliferation induced egf 15 egf activation also appears ac tivate tyrosine kinase src phosphorylates c cbl turn activates ubiquiti nation degradation egfr 16 17 18 recent study shows defective endocytos is egfr characterized deletion mutant point mutation l858r whereby asso ciation c cbl subsequent ubiquitination impaired 19 recently first huma n c cbl mutations reported acute myeloid leukemia aml patients 20 mutat ion r420q inhibits fms like tyrosine kinase 3 flt3 internalization ubiq uitination 20 e3 activity important oncogenesis c cbl dual separate fun ction signal transduction molecule previously shown c cbl important bin ding crkl bcr abl hematopoietic cells also bind modulate functions cyto skeleton binding proteins like talin paxillin tkb domain important bind ing number molecules function signal transduction given critical role c bl normal homeostasis cancer hypothesized might mutated lung cancers st udy report novel c cbl somatic mutations s80n h94y q249e w802 caucasian taiwanese african american lung cancer patients respectively expressing mutations nsclc cell lines lead increased proliferation cell motility s how c cbl mutations occur without met egfr mutations mutually exclusive loh c cbl locus additionally c cbl loh associated either met egfr mutat ions thus hypothesize c cbl mutations might contribute oncogenic potent ial met egfr lung cancer go methods ethics statement written consent re search human subjects obtained institutional review board university ch

icago covers research performed laboratory following contact informatio
n institutional review board university chicago mcgiffert hall 5751 woo
dlawn ave 2nd floor chicago il 60637 written informed consents received
patients whose tissue samples used study tissue samples lung cancer tis
sue paired adjacent normal lung tissues obtained 50 caucasian 29 africa
n americans 40 taiwanese nsclc patients recruited university chicago ho
spital chicago usa caucasian african american patients taipei veterans
general hospital taiwan taiwanese patients obtaining appropriate instit
utional review board permission informed consent patients 119 samples 7
7 men 38 women 4 unknown age diagnosis ranging 47 90 years terms tumor
types 53 adenocarcinoma 32 squamous cell carcinoma 34 large cell carcin
oma 49 stage 14 stage ii 34 stage iii 13 stage iv table s1 cell culture
human non small cell lung carcinoma cells a549 h358 maintained dmem rpm
i 1640 respectively human embryonic kidney 293t cells cultured dmem med
ia supplemented 10 fetal bovine serum 100 units ml penicillin 100 g ml
streptomycin invitrogen carlsbad ca cells cultured 37 c humidified incu
bator containing 5 co2 c cbl gene mutational analysis exons 2 16 c cbl
gene individually amplified polymerase chain reaction pcr primers liste
d table s2 pcr conditions 1 cycle 95 c 5 minutes 35 cycles 94 c 30 seco
nds 58 c 30 seconds 72 c 2 minutes one cycle 72 c 10 minutes pcr produc
ts treated exosap usb corporation cleveland oh sequenced big dye termin
ator chemistry applied biosystems foster city ca sequencing performed f
orward coding strand confirmation c cbl alterations performed sequencin
g reverse strand well chromatograms analyzed mutations using mutation s
urveyor v2 61 softgenetics state college pa plasmid constructs site dir
ected mutagenesis wild type c cbl cdna insert subcloned paltermax expre
ssion vector using xhoi sali restriction enzyme sites promega madison w
i using parental plasmid paltermax c cbl tkb domain double mutation s80
n h94y point mutation q249e c terminal point mutation w802 c cbl create
d using following primers 5 gctggcgctaaagaataacccaccttatatcttagac 3 5 c
taccagatacctaccagtatctccgtactatcttgtc 3 double mutation s80n h94y 5 ctt
tacccgactctttgagccctggtcctctttgc 3 q249e 5 cagctcctcctttggctgattgtctctg
gatggtgatc 3 w802 along complementary primers using quickchange site di
rected mutagenesis xl kit stratagene la jolla ca according manufacturer
instructions constructs confirmed point mutations standard dna sequenci
ng strands loss heterozygosity loh analysis five microsatellites chromo
some 11 3 11q within 200 kb downstream c cbl gene 2 control markers 11p
selected analysis table s3 established microsatellite markers respectiv
e primer sequences selected geneloc database http genecards weizmann ac

il geneloc index shtml weizmann institute science rehovot israel primer s custom designed forward primer fluorescently labeled 5 end fam pet ne d vic applied biosystems primer annealing temperatures duplex scores ev aluated nist primer tools http yellow nist gov 8444 dnaanalysis primert oolspage national institute standards technology gaithersburg md primer s verified performing pcr control dna isolated tk6 cells resolving prod ucts agarose gels bands visualized uv transilluminator genomic dna extr acted tumor samples paired normal lung tissue primers grouped multiplex combinations shown table s4 marker d11s929 served internal control chec k consistency pcrs peaks capillary electrophoresis multiplex pcrs carri ed volume 10 l contained 1 l genomic dna 20 50 ng 0 5 primer 1 0 total primer pair 400 dntps 1x pcr buffer containing mgcl2 0 2 u taq dna poly merase pcr performed abi geneamp 9700 pcr system following conditions 5 min 94 c 30 cycles 30 sec 94 c 1 min 60 c 1 min 72 c 5 min 72 c pcr pro ducts separated capillary electrophoresis abi 3130xl dna analyzer chrom atograms analyzed peak scanner 1 0 genemapper 3 7 software applied bios ystems allelic alterations area peaks produced dna pcr products quantif ied allele ratio allelic areas calculated tumor paired normal dna sampl e qloh allelic ratio tumor peaks divided allelic ratio paired normal sa mple 0 5 2 0 c cbl least one 11q marker least two separate experiments sample considered allelic imbalance interpreted loh samples evaluated l east two separate experiments samples showing prospective loh c cbl rep eated third time included new control marker bax locus data shown chrom osome 19 verify integrity sample dna transfection c cbl constructs a549 cell line transfected using fugene hd roche nutley nj reagent according manufacturer instructions eight g plasmid dna containing either insert empty vector wild type c cbl s80n h94y c cbl q249e c cbl w802 cbl used transfection 6 well culture plate cells harvested 48 h transfection ana lyzed expression c cbl knockdown c cbl knockdown performed using lentiv iral transduction using mission lentiviral transduction particles sigma aldrich st louis mo per manufacturer instructions briefly 1 105 h358 ce lls well seeded 6 well plates infected following day c cbl lentiviral s hrna constructs generate stable c cbl knockdown cell lines cells select ed 2 days 1 g ml puromycin c cbl levels determined using whole cell lys ates immunoblotting anti cbl antibody santa cruz biotechnologies santa cruz ca cell viability assay cells transfected described transfection a ssay forty eight hours transfection viability cells assessed using tryp an blue exclusion wound healing assay a549 cells seeded 6 well plates c ultured 48 h 100 confluent medium changed cells transfected described t

ransfection assay twelve hours transfection straight scratch made acros
s cell layer using 1 ml pipette tip cells gently washed 1 pbs remove ce
llular debris media replaced photographs taken wound region every 12 h
48 h western blot analysis forty eight hours transfection cells collect
ed washed twice 1x pbs lysed ice cold lysis buffer 0 5m tris hcl ph 7 4
1 5 nacl 2 5 deoxycholic acid 10 mm edta 10 np 40 0 5 mm dtt 1 mm pheny
lmethylsulfonyl fluoride 5 g ml leupeptin 10 g ml aprotinin 5 minutes l
ysate centrifuged 13 000 rpm 20 minutes 4 c protein content supernatant
measured total cell lysates 50 g well separated sds page electrophoresi
s gels transferred onto nitrocellulose membranes whatman piscataway nj
membranes blocked 5 non fat dry milk phosphate buffered saline containi
ng tween 20 pbst 1x pbs 0 1 tween 20 1 h room temperature incubated app
ropriate primary antibody 4 c overnight membranes washed three times pb
st probed appropriate horseradish peroxidase hrp conjugated secondary a
ntibody 1 h room temperature membranes washed three times pbst bands vi
sualized using western blot chemiluminescence reagent biorad valencia c
a chemidoc gel documentation system biorad valencia ca antibodies obtai
ned santa cruz biotechnologies used following dilutions c cbl 1 5000 c
met 1 5000 egfr 1 5000 ubiquitin 1 1000 ha 1 5000 actin 1 10 000 flow c
ytometry cell cycle analysis carried flow cytometry approximately 2 106
cells grown media containing 10 fbs cells harvested trypsin edta treatm
ent washed 1x pbs three times fixed ice cold 70 ethanol 2 h cells washe
d cold pbs stained solution containing 25 g ml propidium iodide 200 g m
l rnase 0 1 triton x 100 30 minutes dark cell cycle analysis performed
using guava pca 96 flow cytometer guava technologies millipore billeric
a ubiquitin ligase activity 293t cells maintained culture dmem suppleme
nted 10 fbs 1 penicillin 100 units ml streptomycin 100 g ml transfected
0 2 g egfr pcdna3 2 g ha tagged c cbl constructs indicated using calciu
m phosphate according manufacturer protocol profection promega madison
wi twenty four hours post transfection cells starved overnight dmem sup
plemented 0 5 fbs treated without egf 100 ng ml 15 min cells collected
washed two times ice cold pbs containing 0 2 mm sodium orthovanadate ly
sed ice cold lysis buffer 10 mm tris hcl ph 7 5 150 mm nacl 5 mm edta 1
triton x100 10 glycerol 2 mm sodium orthovanadate protease inhibitors l
ysates cleared debris centrifugation 16 000 g 10 min 4 c egfr immunopre
cipitations performed 200 g cleared lysate using 250 ng rabbit anti egf
r protein g plus sepharose overnight 4 c precipitations washed 5 times
lysis buffer boiling laemmli buffer elutions immunoblotted anti ubiquit
in egfr twenty micrograms cleared lysate immunoblotted c cbl constructs

using anti ha statistical analysis mutation rates different groups compared using fisher exact test continuous variables group comparisons performed using analysis variance anova followed sidak adjustment multiple comparisons experiments involving repeated measurements time analyzed using repeated measures anova greenhouse geisser adjustment degrees freedom analyses conducted using stata v10 1 software stata corporation college station tx go results c cbl gene mutations lung cancer investigate role c cbl lung cancer analyzed genomic dna tumor paired normal samples drawn multiple ethnicities lung tumor samples represented caucasians n 50 african americans n 29 taiwanese n 40 lung cancer patients designed 12 pairs primers sequence coding region c cbl gene spans exons 2 16 table s2 identified 8 unique somatic mutations c cbl exons among 8 different patients variation l620f known snp rs2227988 exon 11 also detected importantly eight novel non synonymous mutations confirmed sequencing strands c cbl genomic dna obtained lung tumor samples table 1 moreover none 8 mutations detected corresponding normal tissue indicating somatic mutations four synonymous single nucleotide variations snvs also identified used study table 1 table 1 c cbl mutation analysis 119 lung cancer patient tumor tissues three 8 novel non synonymous mutations located tkb tyrosine kinase binding domain s80n h94y q249e one ring finger domain v391i one proline rich region 72515 72517 del atg three c terminal region w802 r830k a848t c cbl protein figure 1a figure s1 figure 1b show model chromatograms representative samples figure 1 figure 1 c cbl mutations loh non small cell lung cancer 11q loh c cbl gene paired lung tumor normal lung tissue samples taiwanese patients n 37 investigated loh eight 21 6 showed loh c cbl locus chromosome 11 29 samples 78 4 revealed normal allelic contribution microsatellite markers figures 1c c cbl mutations different ethnic groups c cbl double mutant s80n h94y found patient overall mutation rate c cbl lung tumors 6 7 8 119 frequency c cbl mutation highest large cell carcinoma 14 7 5 34 patients followed squamous carcinoma 6 3 2 32 patients least observed adenocarcinoma ad 1 8 1 53 patients although rates statistically significant p 0 292 mutation rates 6 0 among caucasians 0 20 ad 0 10 sq 3 20 lc 13 8 african americans 1 10 ad 1 10 sq 2 9 lc 2 5 0 23 ad 1 12 sq 0 5 lc taiwanese population additionally two taiwanese patients lung cancer one squamous one adenocarcinoma known snp l620f ethnic differences statistically significant however power detect differences low mutations met egfr co associated c cbl alterations since east asians lung cancer higher frequency egfr met mutations lung tumors 21 22 also determined mutations egfr met taiwanese

cohort samples compared results observed c cbl alterations loh mutation
s 37 samples tested find overlap c cbl mutations c cbl loh figure 2 thr
ee c cbl mutants including known l620f snp rs2227988 one samples met mu
tation n375s egfr mutation l858r among 8 samples loh c cbl locus 5 addi
tional mutation met n375s 2 egfr exon 19 deletion twenty six samples ne
ither c cbl mutation c cbl loh 3 patients c cbl mutation c cbl loh amon
g 26 samples 9 met mutation 8 n375s 1 l211w 13 egfr mutation 7 exon 9 d
eletion 6 l858r 4 met egfr mutation thus rate met egfr mutations among
patients loh c cbl locus 7 8 similar seen patients without c cbl mutati
on loh 22 26 patients p 0 99 4 patients identifiable mutation c cbl met
egfr represented 10 8 37 patients analyzed taiwanese patient cohort con
versely 89 2 taiwanese lung cancer patients identifiable mutation eithe
r c cbl met egfr combination three genes figure 2 additionally determin
ed p53 kras mutations taiwanese cohorts two p53 1 kras mutation detecte
d single kras mutation overlapped one p53 mutation patient also egfr ex
on 19 deletion c cbl mutation p53 mutation sample c cbl loh concurrent
met n375s mutation thus taiwanese samples analyzed p53 kras mutations c
cbl mutations mutually exclusive data shown figure 2 figure 2 c cbl mut
ations relationship met egfr mutations lung cancer cellular functions c
cbl alterations context lung tumorigenesis e3 activity intact mutant c
cbl proteins investigate whether different c cbl mutations affect e3 ac
tivity egfr chosen model substrate c cbl e3 function c cbl mutants test
ed enhanced ubiquitination activated egfr similar wild type c cbl prote
in result demonstrates catalytic activity c cbl mutants impaired egfr s
ubstrate figure 3a figure 3 figure 3 ubiquitination viability expressio
n cell cycle analysis various c cbl mutants b effect lung cancer cell v
iability effect representative c cbl mutant three ethnic backgrounds lu
ng cancer cell viability cell lines determined s80n h94y double mutatio
n q249e w802 identified lung tumor samples obtained caucasian taiwanese
african american respectively described methods c cbl wild type wt thre
e mutants expressed cloning paltermax vector a549 cells cells express r
elatively low basal levels endogenous c cbl data shown transfection eff
iciency comparable different groups number cells transfected c cbl wild
type construct 70 compared control cells transfected empty vector cells
transfected s80n h94y q249e w802 c cbl mutant constructs resulted incre
ased number viable cells 132 3 120 8 147 9 higher respectively relative
empty vector control transfected cells significantly different wild typ
e construct p 0 022 p 0 049 p 0 008 respectively figure 3b relative lev
els c cbl protein whole cell lysates prepared samples obtained parallel

experiment determined c cbl protein levels samples representing untrans
fected empty vector transfected cells comparable representing c cbl wt
three c cbl mutants comparable figure 3c c effect cell cycle investigat
e increases cell viability different c cbl mutants due increased cellul
ar proliferation cell cycle analysis performed a549 cells transfected c
cbl wt three different mutants s80n h94y q249e w802 empty vector transf
ectant used control forty eight hours transfection cell cycle analysis
performed described materials methods significant change subg1 g1 phase
cell cycle among different mutants compared wt construct p 0 64 p 0 40
p 0 28 respectively g2 phase cell cycle showed increase cell numbers th
ree mutants s80n h94y q249e w802 compared wt difference statistically s
ignificant p 0 25 figure 3d effect cell motility investigate effect exp
ression three c cbl mutants cell migration carried wound healing assay
described materials methods closing scratch wound monitored 0 12 24 36
48 h figure 4a samples represented cells transfected mutants wound gap
much smaller seen sample represented cells transfected c cbl wt p 0 001
also determined rate wound closure five groups 48 h wild type c cbl tra
nsfectants showed 61 1 open wound s80n h94y q249e w802 mutants showed 1
8 7 23 9 34 3 open wound respectively p 0 001 figure 4b figure 4 figure
4 c cbl mutations affect wound healing a549 cells e c cbl knockdown inc
reases cell viability hypothesized loh seen samples could lead decrease
d expression c cbl thus tested effect c cbl knockdown lung cancer cells
compared a549 h358 lung cancer cells express relatively high levels end
ogenous c cbl data shown c cbl expression knocked using lentiviral cons
truct expressed c cbl specific shrna compared results transduced scramb
led shrna results shown figure 5 identified several clones revealed var
ying degrees c cbl knockdown showing different sets c cbl lentiviral sh
rna knockdown efficiency figure 5a clones tested clone 27 chosen experi
ments equal amount cells seeded 6 well plate cell proliferation measure
d various times results depicted figure 5b expected number cells increa
sed time dependent fashion 100 190 relative scrambled shrna control spa
n 48 h p 0 0002 figure 5b cell cycle phases h358 cells knocked c cbl sh
rna looked compared scrambled shrna discernable differences two constru
cts different phases cell cycle data shown figure 5 figure 5 knockdown
c cbl using shrna increases cell proliferation go discussion results de
monstrate c cbl somatically mutated loh lung cancers significantly cont
ribute enhanced cell viability motility also high prevalence loh respec
t c cbl lung tumors harbored met egfr mutation present study demonstrat
ed occurrence c cbl mutations lung cancer patients especially different

ancestral variations mutations c cbl recently reported juvenile myelomo
nocytic leukemia myeloid malignancies aml study mutation r420q located
junction ring finger linker region inhibited fms like tyrosine kinase 3
flt3 internalization ubiquitination 20 thus contributing gain function
rtk addition mutations h398y c384r l380p mapped ring finger domain link
er region c cbl required e3 activity 23 24 25 26 27 additionally homozy
gous mutations ring finger domain c cbl gene described result acquired
uniparental disomy upd 26 important note results indicate loh 11q23 loc
us mutually exclusive missense mutations c cbl somatic mutations hetero
zygous mutations aml led abrogation e3 activity leading prolonged rtk a
ctivation addition mutants located linker region surrounding ring finge
r domain exhibited enhanced akt signaling response cytokine stimulation
26 addition shown nh3t3 cells neither mutations ring finger linker regi
on causes transformation however certain mutations perturbs ubiquitinat
ion others affect receptor recycling prolong kinase activity 28 report
c cbl mutations mapped ring finger domain also tkb domain proline rich
domain c terminal region none mapped linker region reported aml studies
described 23 24 25 26 29 addition 8 mutants detected found different et
hnic backgrounds example s80n h94y q249e w802 detected caucasians taiwa
nese african americans respectively results point difference lung cance
r cancers also genetic polymorphism among different races cancer intere
stingly large disparity african american ethnic populations lung cancer
30 previously shown low frequency egfr met mutation african americans c
ompared taiwanese caucasians 31 study number african american samples a
nalyzed relatively fewer found 3 mutations unique ethnicity would behoo
ve us study genetic alterations occur determine targeted therapeutics a
frican americans results provide evidence importance c cbl tumorigenesi
s potential signaling prediction based aml data would v391i ring finger
domain mutation would affect e3 activity also important determine bindi
ng partners c cbl tkb domain proline rich domain mutations previously s
hown tkb domain bind growth factor receptors important determine cross
binding mutants met egfr would also important future look fluorescence
situ hybridization copy number changes c cbl lung cancer c cbl plays im
portant role regulating rtk mediated signaling k63 poly ubiquitination
subsequent downregulation rtks followed lysosomal degredation 3 mono ub
iquitination ubiquitinated k63 linked chains substrates c cbl may lead
enhancement biological biochemical functions reviewed hermann et al 200
7 32 mutations analyzed studies point fact e3 activity c cbl egfr intac
t egfr levels various mutants remain figure s2 multiple kinases rtks no

n rtks could acted upon c cbl including erbs pdgfr fms met c kit vegfr flt 1 ron fgfr ir well syk fyn lck fgr lyn c abl 3 lung cancers relevant substrates c cbl terms degradation signal transduction yet identified observation c cbl somatic mutations especially s80n h94y q249e w802 showed increased cell viability cell motility agreement physiological role cbl regulation apoptosis differentiation identified drosophila significant 33 previously shown activating c cbl mutation downregulates egfr signaling decreases cellular proliferation migration breast cancer cell lines 34 although role c cbl negative regulation rtks well substantiated thereby suggesting natural tumor suppressor studies cancer cells revealed tumor suppressor tumor promoting activities depending type c cbl mutation number alleles c cbl locus 24 agreement three c cbl mutants described appear tumor growth metastasis promoting properties although mutants outside ring finger linker region c cbl downstream effects significant cause increased proliferation migration substrate affected mutations known yet raises possibility cellular functions c cbl independent ubiquitin ligase activity area currently investigating oncogenic nature rtks addiction cancers growth signals given clustering c cbl egfr met mutations possible transforming effect c cbl mutations likely combinatorial effect three also show loh c cbl found significant number samples harbored met egfr mutations fact 7 lung tumor samples likely c cbl mutations additional 22 likely harbor c cbl related loh makes c cbl highly mutated molecule lung cancer since loh alone enough cause transforming event 35 36 37 associated mutation met egfr locus yet another rtk discussed may play role carcinogenesis predict loh c cbl results haploinsufficiency downplays rtk ubiquitination leading hyperactivity rtks however whether sufficient cause tumorigenesis remains determined consistent hypothesis fact c cbl mice increased kinase activity lymphocytes sufficient tumor formation 35 36 37 c cbl loh could also lead increased expression c cbl allele compensate loss allele alternately could form synergy working reduced c cbl levels mutated receptors exacerbate phenotype alone previous studies lab others shown east asians lung cancers relatively high frequencies gain function mutations rtks egfr met 31 cohort japanese patients activating met mutation identified splice region deletes juxtamembrane domain involved e3 activity c cbl 38 study also found activation met mutually exclusive egfr kras her2 gene mutations 38 failed detect mutations significant numbers lung tumor samples obtained african americans n 29 caucasian n 50 patients one met mutation identified groups whereas 1 3 egfr mutations identified african american caucasian cohorts r

```
espectively egfr mutations earlier identified one key mutations affecti
ng lung adenocarcinoma patients comprehensive study 188 patients 39 stu
dy encompasses different histologies nsclc however published series fin
d mutations c cbl met unlike study encompassed different subtypes nsclc
important note recently shown met mutations lung cancer majority germli
ne 31 reported earlier c cbl mutations small cohort taiwanese lung canc
er samples 40 efforts understand ethnic differences lung oncogenome als
o looked pax transcription factors pax5 pax8 highly expressed lung canc
ers however preferential expression mutations genes lung tumor samples
african americans study show relatively high frequency c cbl mutations
lung cancers especially large cell type among caucasians particularly a
mong african americans therefore propose c cbl efficacious target lung
cancers african americans needs substantiated important prognosis afric
an americans lung cancer especially men much poorer compared caucasian
counterparts 41 conclusion results presented study demonstrate c cbl fr
equently mutated even lost lung cancers results support role c cbl muta
nts independent ubiquitination activity given relatively high mutation
rates c cbl well rtks met egfr likely combined effect could synergistic
promoting tumorigenesis 31991'
```

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 [13]:

```
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 o
f output varaible 'y_true' [stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(result, y_true, st
ratify=y_true, test_size=0.2)
# split the train data into train and cross validation by maintaining s
ame distribution of output varaible 'y_train' [stratify=y_train]
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, str
atify=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

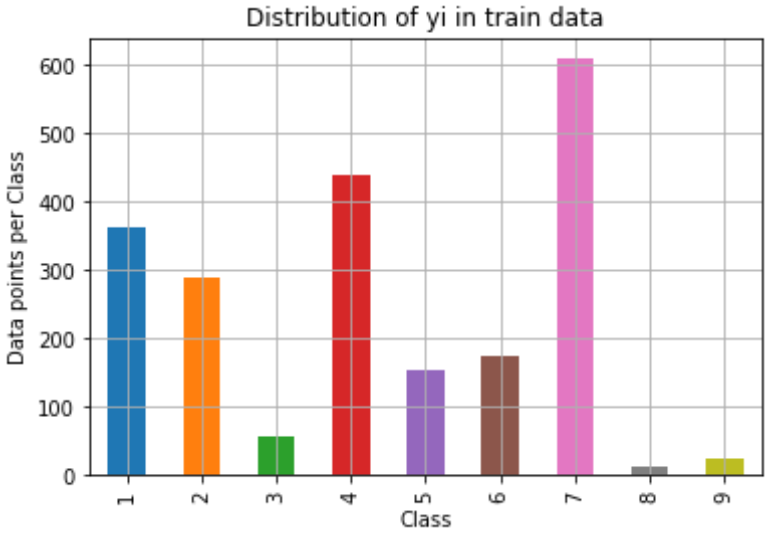
In [14]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0
])
```

```
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
```

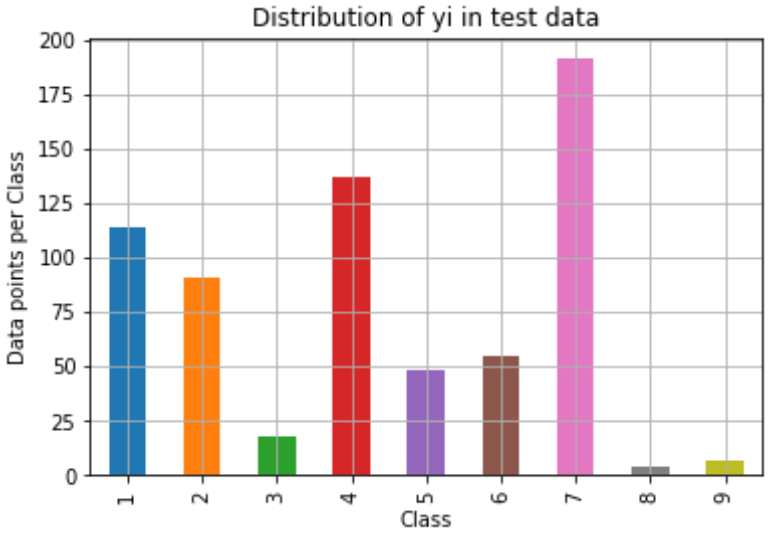
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [15]:
```
# it returns a dict, keys as class labels and values as the number of d
ata points in that class
train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/num
py.argsort.html
# -(train_class_distribution.values): the minus sign will give us in de
creasing 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_distri
bution.values[i], '(', np.round((train_class_distribution.values[i]/tra
in_df.shape[0]*100), 3), '%)')
```

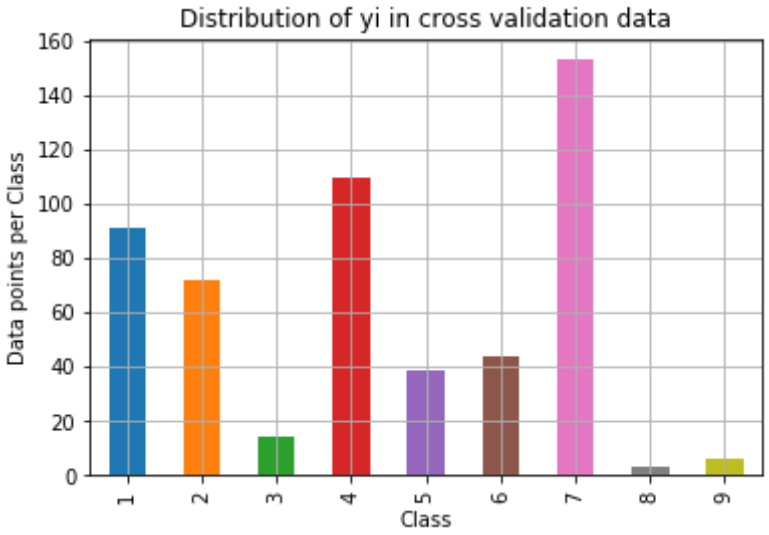
```
print('-'*80)
my_colors = 'rgbkymc'
test_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/num
py.argsort.html
# -(train_class_distribution.values): the minus sign will give us in de
creasing 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_distrib
ution.values[i], '(', np.round((test_class_distribution.values[i]/test_
df.shape[0]*100), 3), '%)')

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/num
py.argsort.html
# -(train_class_distribution.values): the minus sign will give us in de
creasing 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_distribut
ion.values[i], '(', np.round((cv_class_distribution.values[i]/cv_df.sha
pe[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 [16]:

```
# 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 cl
ass i are predicted class j

    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of element
s in that column

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 correspo
nds 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 element
s in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 correspo
nds 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=la
bels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    print("-"*20, "Precision matrix (Columm Sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=la
bels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    # representing B in heatmap format
    print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(A, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=la
bels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

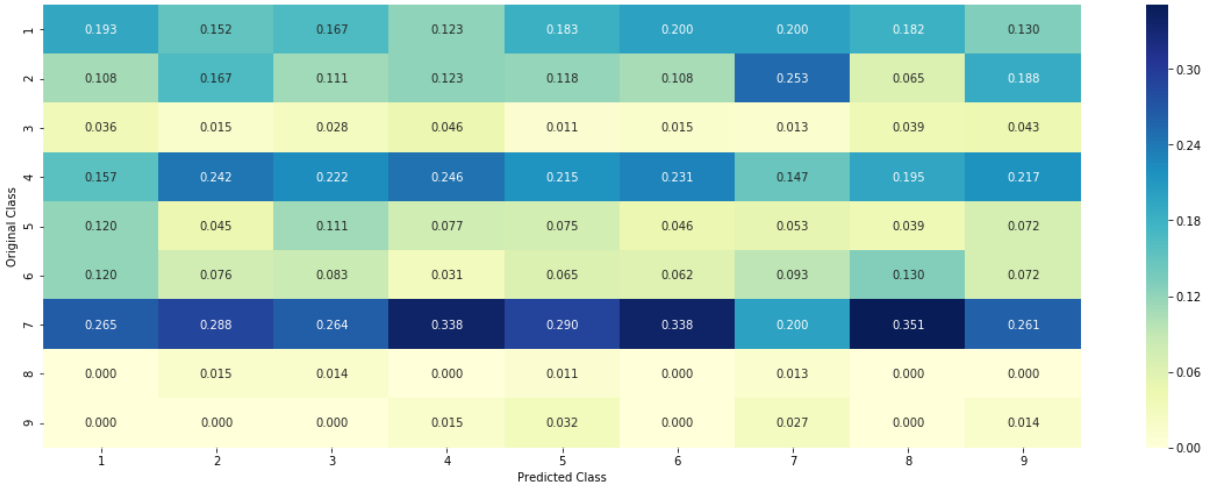
In [17]:

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

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,9))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,9)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Cross Validation Data using Random Model",log_loss(y
_cv,cv_predicted_y, eps=1e-15))
```

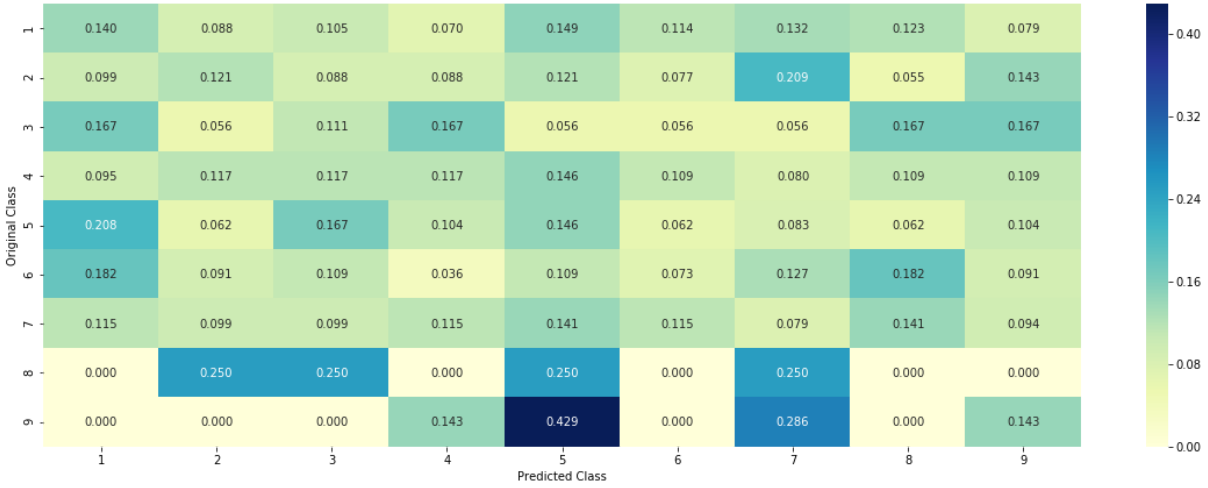
```
# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_p
redicted_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 2.5794810179762995
Log loss on Test Data using Random Model 2.5199756761514513
-------------------- Confusion matrix --------------------
```


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



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



3.3 Univariate Analysis

In [18]:
```
# 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 feat
ure in train data dataframe
# build a vector (1*9) , the first element = (number of times it occure
d in class1 + 10*alpha / number of time it occurred in total data+90*al
pha)
# gv_dict is like a look up table, for every gene it store a (1*9) repr
esentation 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_f
ea'
# 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 arr
ay for each gene/variation
    gv_dict = dict()

    # denominator will contain the number of time that particular featu
re occured in whole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation b
elongs 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['Ge
ne']=='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[f
eature]==i)]

            # cls_cnt.shape[0](numerator) will contain the number of ti
```

```
me 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 a
s 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, 0.068181
818181818177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.0378787
8787878788, 0.03787878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224
489795918366, 0.27040816326530615, 0.061224489795918366, 0.066326530612
244902, 0.051020408163265307, 0.051020408163265307, 0.05612244897959183
7],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625,
 0.06818181818181818177, 0.06818181818181818177, 0.0625, 0.3465909090909091
2, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.06060
6060606060608, 0.078787878787878782, 0.1393939393939394, 0.345454545454
54546, 0.060606060606060608, 0.060606060606060608, 0.0606060606060606060
8],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.06918
2389937106917, 0.46540880503144655, 0.075471698113207544, 0.06289308176
1006289, 0.069182389937106917, 0.062893081761006289, 0.0628930817610062
89],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.0728476
82119205295, 0.072847682119205295, 0.066225165562913912, 0.066225165562
913912, 0.27152317880794702, 0.066225165562913912, 0.06622516556291391
2],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333
333333333334, 0.073333333333333334, 0.093333333333333338, 0.080000000000
0000002, 0.29999999999999999, 0.066666666666666666, 0.06666666666666666
6],
    #      ...
```

```
    #     }
    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 e
ach feature 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_fe
a
    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 [19]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
```

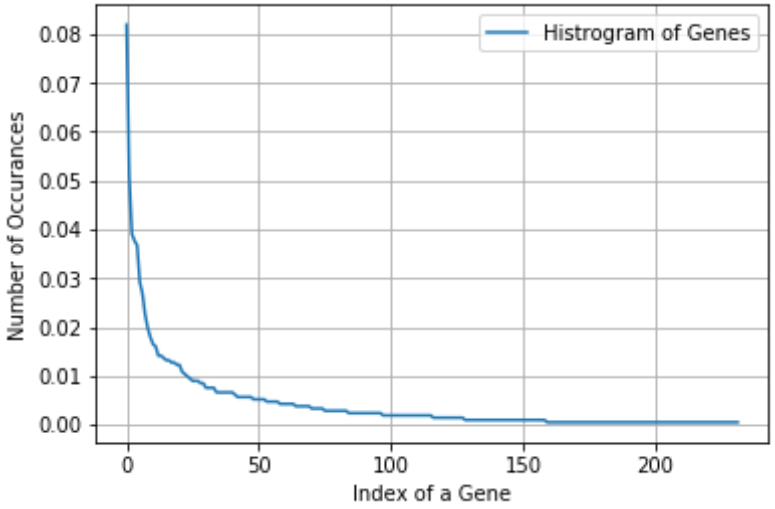
```
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 232
BRCA1     174
TP53      108
EGFR       83
BRCA2      80
PTEN       78
BRAF       62
KIT        57
ALK        48
PDGFRA     42
ERBB2      38
Name: Gene, dtype: int64
```

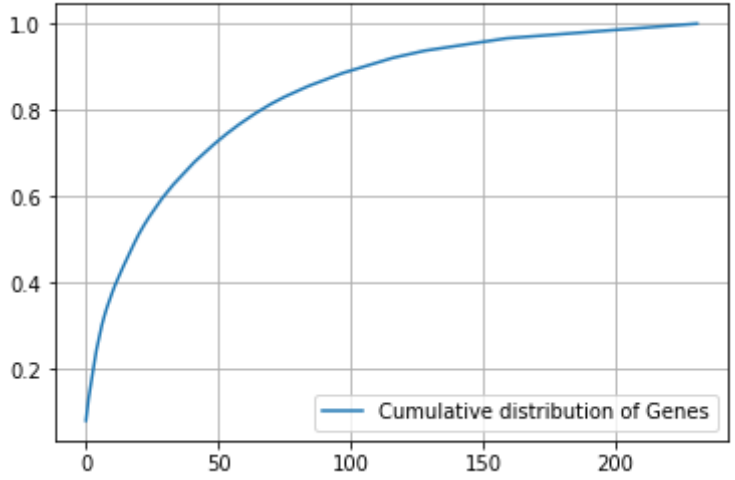
In [20]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of
 genes in the train data, and they are distibuted as follows",)
```

```
Ans: There are 232 different categories of genes in the train data, and
they are distibuted as follows
```

In [21]:
```
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 [22]:
```
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 [23]:
```
#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, "Gen
e", train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gen
e", test_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene",
 cv_df))
```

In [24]:
```
print("train_gene_feature_responseCoding is converted feature using res
pone coding method. The shape of gene feature:", train_gene_feature_res
ponseCoding.shape)
```

```
train_gene_feature_responseCoding is converted feature using respone co
ding method. The shape of gene feature: (2124, 9)
```

GENE TFIDF VECTORIZER </h3>

In [25]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer(max_features=4000)
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature: (2124, 231)
```

GENE BOW VECTORIZER

In [113]:
```
# one-hot encoding BOW of Gene feature.
bow_gene_vectorizer = CountVectorizer()
bow_train_gene_feature_onehotCoding = bow_gene_vectorizer.fit_transform(train_df['Gene'])
bow_test_gene_feature_onehotCoding = bow_gene_vectorizer.transform(test_df['Gene'])
bow_cv_gene_feature_onehotCoding = bow_gene_vectorizer.transform(cv_df['Gene'])
```

In [114]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature:", bow_train_gene_feature_onehotCoding.shape)
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature: (2124, 231)
```

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

There are many ways to estimate how good a feature is, in predicting y_i. One of the good

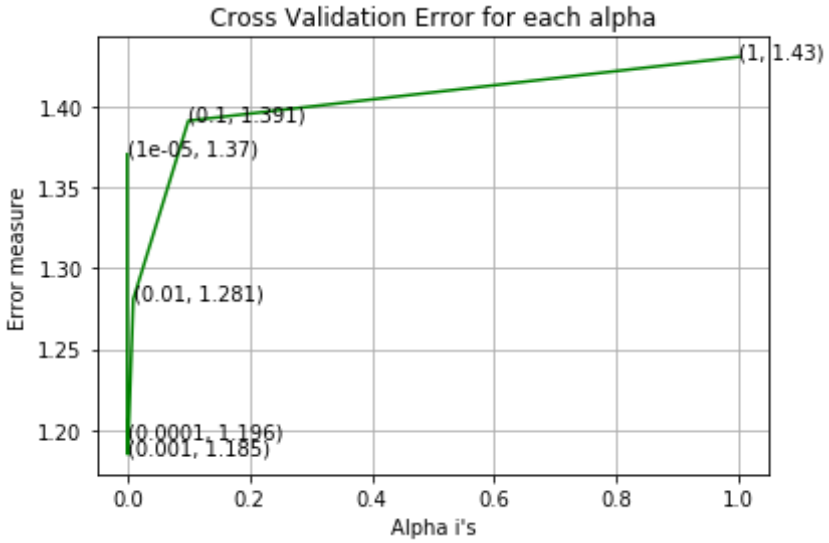
methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [29]:

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic 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.clas
ses_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv
, predict_y, labels=clf.classes_, eps=1e-15))
```

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.3702939516978163
For values of alpha =  0.0001 The log loss is: 1.1956573579303584
For values of alpha =  0.001 The log loss is: 1.1850169183885546
For values of alpha =  0.01 The log loss is: 1.2808985840389566
For values of alpha =  0.1 The log loss is: 1.3910067795683112
For values of alpha =  1 The log loss is: 1.4304837097247387
```




```
For values of best alpha =  0.001 The train log loss is: 1.109868759688
2466
For values of best alpha =  0.001 The cross validation log loss is: 1.1
850169183885546
For values of best alpha =  0.001 The test log loss is: 1.2086235774486
003
```

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

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

In [30]:
```
print("Q6. How many data points in Test and CV datasets are covered by
 the ", unique_genes.shape[0], " genes in train dataset?")

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

```
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0],
":",(test_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[
0],":" ,(cv_coverage/cv_df.shape[0])*100)
```

```
Q6. How many data points in Test and CV datasets are covered by the  23
2  genes in train dataset?
Ans
1. In test data 645 out of 665 : 96.99248120300751
2. In cross validation data 512 out of  532 : 96.2406015037594
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

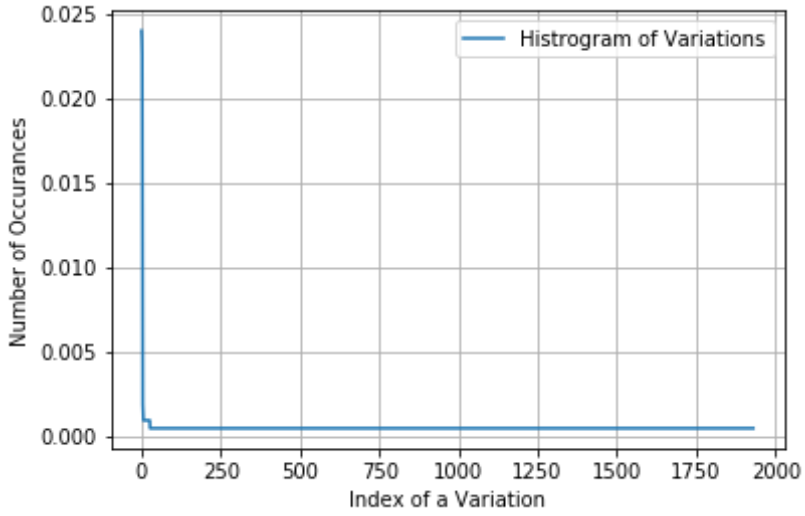
In [31]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1931
Truncating_Mutations    51
Deletion                50
Amplification           48
Fusions                 22
Overexpression           4
G12V                     3
R841K                    2
A146T                    2
S222D                    2
M1R                      2
Name: Variation, dtype: int64
```

In [32]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categori
es of variations in the train data, and they are distibuted as follows"
,)
```

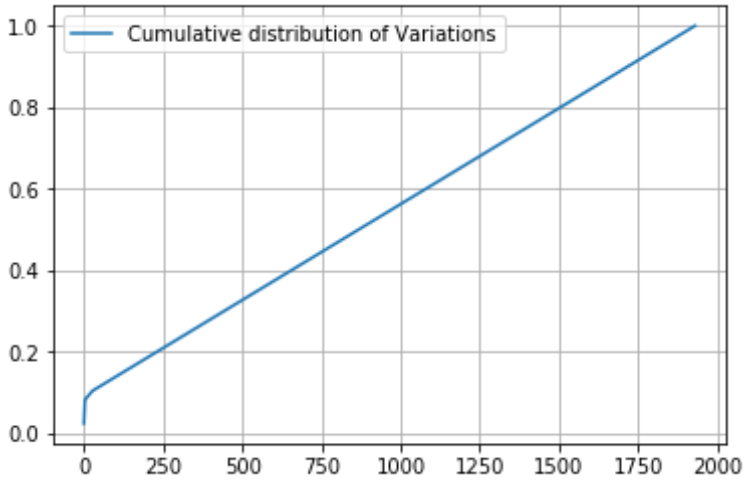
```
Ans: There are 1931 different categories of variations in the train dat
a, and they are distibuted as follows
```

In [33]:
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [34]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.0240113  0.04755179 0.07015066 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [35]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
 "Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
"Variation", test_df))
# cross validation gene feature
```

```
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "V
ariation", cv_df))
```

In [36]:
```
print("train_variation_feature_responseCoding is a converted feature us
ing 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)
```

VARIATION TFIDF VECTORIZER

In [37]:
```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer(max_features=4000)
train_variation_feature_onehotCoding = variation_vectorizer.fit_transfo
rm(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(te
st_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_d
f['Variation'])
```

In [38]:
```
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 on
ne-hot encoding method. The shape of Variation feature: (2124, 1962)
```

VARIATION BOW VECTORIZER

In [115]:
```
# one-hot encoding of variation feature.
bow_variation_vectorizer = CountVectorizer()
bow_train_variation_feature_onehotCoding = bow_variation_vectorizer.fit
_transform(train_df['Variation'])
bow_test_variation_feature_onehotCoding = bow_variation_vectorizer.tran
```

```
sform(test_df['Variation'])
bow_cv_variation_feature_onehotCoding = bow_variation_vectorizer.transf
orm(cv_df['Variation'])
```

In [116]:
```
print("train_variation_feature_onehotEncoded is converted feature using
 the onne-hot encoding method. The shape of Variation feature:", bow_tr
ain_variation_feature_onehotCoding.shape)
```

```
train_variation_feature_onehotEncoded is converted feature using the on
ne-hot encoding method. The shape of Variation feature: (2124, 1962)
```

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

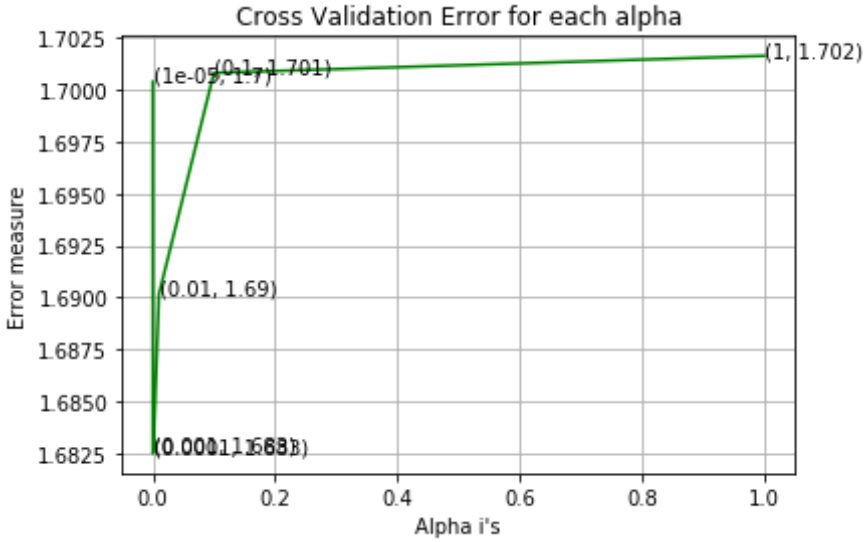
Let's build a model just like the earlier!

In [41]:
```
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

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

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

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

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

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


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

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

```
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation 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 l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.7003701421230708
For values of alpha =  0.0001 The log loss is: 1.6825063672025062
For values of alpha =  0.001 The log loss is: 1.6826515693609063
For values of alpha =  0.01 The log loss is: 1.6901419914156803
For values of alpha =  0.1 The log loss is: 1.700797179791544
For values of alpha =  1 The log loss is: 1.7016150167109874
```




```
For values of best alpha =  0.0001 The train log loss is: 0.76644850641
97324
For values of best alpha =  0.0001 The cross validation log loss is: 1.
6825063672025062
For values of best alpha =  0.0001 The test log loss is: 1.702371093723
9243
```

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 [42]:
```
print("Q12. How many data points are covered by total ", unique_variati
ons.shape[0], " genes in test and cross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Vari
ation'])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'
])))].shape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0],
":",(test_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[
0],":" ,(cv_coverage/cv_df.shape[0])*100)
```

```
Q12. How many data points are covered by total  1931  genes in test and
cross validation data sets?
Ans
1. In test data 67 out of 665 : 10.075187969924812
2. In cross validation data 58 out of  532 : 10.902255639097744
```

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 [43]:
```
# 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 [44]:
```
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 )/(t
otal_dict.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_pr
ob/len(row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

TEXT TFIDF VECTORIZER

In [45]:
```
# one-hot encoding of TEXT feature.
text_vectorizer = TfidfVectorizer(max_features=4000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_d
f['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

```
ber of times it occured
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_count
s))


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

```
Total number of unique words in train data : 4000
```

In [46]:
```
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = train_df[train_df['Class']==i]
    # build a word dict based on the words in that class
    dict_list.append(extract_dictionary_paddle(cls_text))
    # append it to dict_list

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


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

In [47]:
```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
test_text_feature_responseCoding  = get_text_responsecoding(test_df)
cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)
```

In [48]:
```
# https://stackoverflow.com/a/16202486
```

```
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.
T/train_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/
test_text_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_t
ext_feature_responseCoding.sum(axis=1)).T
```

In [49]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCo
ding, axis=0)

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

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

In [50]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x:
 x[1] , reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

In [51]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({5.480200774942549: 2, 2.964123624559329: 2, 178.6816586361004
2: 1, 122.3133766902471: 1, 120.12979637642165: 1, 90.98887820333383:
1, 85.7514272065363: 1, 83.95911975731728: 1, 80.6405456863985: 1, 79.0
5200211229308: 1, 78.84325119688197: 1, 75.44516296768867: 1, 72.095150
60111607: 1, 68.67407534756198: 1, 67.58529536529814: 1, 66.79718286301
```

947: 1, 64.35982342822643: 1, 60.036214662599875: 1, 56.83729241092101:
1, 56.731449010746935: 1, 56.231571248086176: 1, 53.306864502464016: 1,
52.7927115491636: 1, 51.24480757716615: 1, 49.92720784786565: 1, 49.779
11461603238: 1, 48.74784478581815: 1, 46.335778504076224: 1, 46.2141530
055469: 1, 45.53963463502576: 1, 45.27478745552529: 1, 44.6474354146400
86: 1, 44.47394929567082: 1, 44.300148394027225: 1, 44.19561865237816:
1, 43.348483452353165: 1, 43.29970133305245: 1, 43.07378926215833: 1, 4
1.74889949003584: 1, 41.63166898469786: 1, 40.55376621128512: 1, 40.489
26473616273: 1, 38.94343533247123: 1, 38.31771627764596: 1, 38.19941225
164234: 1, 38.077659439701534: 1, 36.17456829843447: 1, 35.502111938996
35: 1, 34.51384143758203: 1, 34.20192674906085: 1, 34.18399274042492:
1, 33.38506526588402: 1, 32.924206869701436: 1, 32.455214298666114: 1,
32.16153833195462: 1, 31.42931522375013: 1, 31.40592021262197: 1, 31.36
3959746380957: 1, 31.12208816003353: 1, 31.02017266260104: 1, 30.540814
736987077: 1, 30.156308018063804: 1, 30.109093664248558: 1, 29.94770394
072446: 1, 29.923400550337092: 1, 29.776700762043603: 1, 29.72189469387
333: 1, 29.68285518894492: 1, 29.578773666865313: 1, 29.37618530628348:
1, 29.1570399083217: 1, 28.997689178590377: 1, 28.175794131077776: 1, 2
7.410745053931045: 1, 27.028591856640507: 1, 26.7281548347843: 1, 26.60
9124022941632: 1, 26.366938386167263: 1, 26.110934108151675: 1, 26.0894
09744716818: 1, 26.030006089687348: 1, 25.915627777935896: 1, 25.427804
42565446: 1, 25.39081919851997: 1, 25.23024240452651: 1, 25.14295458548
3896: 1, 24.889413162333614: 1, 24.818591975121407: 1, 24.7968906318729
05: 1, 24.77382161227746: 1, 24.54880444478745: 1, 24.518511896105707:
1, 24.154280140484477: 1, 24.017372107737973: 1, 23.863522708361828: 1,
23.545855683638063: 1, 23.4373065681154: 1, 23.277819090885167: 1, 23.2
69878487755804: 1, 23.10903286717636: 1, 23.020810660277107: 1, 22.9954
1903745438: 1, 22.939516248434742: 1, 22.89643281280314: 1, 22.85039401
5512016: 1, 22.776893284552994: 1, 22.713384686312672: 1, 22.5719533975
3151: 1, 22.452480278749636: 1, 22.197713896561428: 1, 22.1164950005527
9: 1, 22.08239773383864: 1, 22.06567773287577: 1, 21.902902245304027:
1, 21.762948044240673: 1, 21.536260518642756: 1, 21.450431206120047: 1,
21.447619729073185: 1, 21.36922942527752: 1, 21.345991684919525: 1, 21.
314507916602295: 1, 21.263741946544883: 1, 21.144276151976694: 1, 21.12
505093600503: 1, 21.00388115873428: 1, 20.994944658881554: 1, 20.840982
689678523: 1, 20.80624780778173: 1, 20.642661611201778: 1, 20.471543670
563605: 1, 20.423129419206184: 1, 20.33216099187232: 1, 20.229991973541
864: 1, 20.198909360981855: 1, 20.177362127936554: 1, 19.9731015733791
8: 1, 19.778260902286714: 1, 19.724186635373417: 1, 19.70261055966526:

1, 19.299395401270232: 1, 19.259790241078445: 1, 19.12982842428519: 1,
19.06339311504805: 1, 19.062888104154364: 1, 19.018457486561772: 1, 19.
004099728251955: 1, 18.885796021936923: 1, 18.774021561967448: 1, 18.67
503576196992: 1, 18.652658155838903: 1, 18.566464253092178: 1, 18.43597
9708251217: 1, 18.38709521510393: 1, 18.370339137467006: 1, 18.36619028
527008: 1, 18.28787817186431: 1, 18.189578549619448: 1, 18.133118707567
615: 1, 18.131844807810385: 1, 18.117687135377807: 1, 18.11048897949569
5: 1, 18.01142457702502: 1, 17.82472360632187: 1, 17.813439600742345:
1, 17.80260473878819: 1, 17.765696137586648: 1, 17.56285711582158: 1, 1
7.551671115034544: 1, 17.507274385358198: 1, 17.503629763158088: 1, 17.
491176160474154: 1, 17.42941171601798: 1, 17.354335835541878: 1, 17.347
556284825504: 1, 17.321120005506295: 1, 17.10292222694711: 1, 17.043032
51473375: 1, 17.036606219187107: 1, 17.015628557204266: 1, 17.015012825
0056: 1, 16.944691778964867: 1, 16.79640506910169: 1, 16.79448312529353
5: 1, 16.757392279114587: 1, 16.753769234881894: 1, 16.668012430814812:
1, 16.64639670939256: 1, 16.604876445031124: 1, 16.591508156832504: 1,
16.562883602225728: 1, 16.535511516083012: 1, 16.425846272269464: 1, 1
6.399489300150186: 1, 16.37228996097536: 1, 16.370233882194412: 1, 16.2
95792782789366: 1, 16.28707332902742: 1, 16.28512507125693: 1, 16.24205
067990699: 1, 16.216830155885383: 1, 16.124391486693654: 1, 16.01057902
8198947: 1, 16.004007891240924: 1, 15.989540498854177: 1, 15.9835428147
81268: 1, 15.98134605345005: 1, 15.87419960635194: 1, 15.8461016636997
5: 1, 15.829806355769891: 1, 15.775063707950062: 1, 15.721200798008326:
1, 15.581306781812607: 1, 15.50063215128039: 1, 15.485674281102328: 1,
15.464544708825052: 1, 15.457725550189428: 1, 15.445854684835126: 1, 1
5.44408409759376: 1, 15.418250398222062: 1, 15.41109838990785: 1, 15.40
1244285451535: 1, 15.343386378721698: 1, 15.342960890081498: 1, 15.3387
03005310181: 1, 15.32331065415123: 1, 15.314034116515462: 1, 15.3101027
418192: 1, 15.288208868737412: 1, 15.247606609085928: 1, 15.21589474952
1096: 1, 15.102403384861391: 1, 15.088814623878841: 1, 15.0526475472823
35: 1, 15.026190447769425: 1, 14.944085234541015: 1, 14.90653110040680
2: 1, 14.90000166523978: 1, 14.888833254274656: 1, 14.87989797555961:
1, 14.8416052398232: 1, 14.839840401378686: 1, 14.828600364142831: 1, 1
4.812960660655776: 1, 14.805972830647562: 1, 14.722481436015721: 1, 14.
695653018168349: 1, 14.681758380751521: 1, 14.661386028700788: 1, 14.65
6499747790852: 1, 14.652036587738078: 1, 14.643628376515341: 1, 14.6333
30984125124: 1, 14.599575254701762: 1, 14.588128349591251: 1, 14.556632
901378157: 1, 14.416396278839784: 1, 14.363800375214156: 1, 14.31111183
8377645: 1, 14.24528762203241: 1, 14.22277310515722: 1, 14.086239293992

```
86: 1, 14.07126055783893: 1, 14.064908458324858: 1, 14.05675310812227:
1, 14.047607273337524: 1, 14.034985709799697: 1, 14.003156521158154: 1,
13.948136692990687: 1, 13.925246843872939: 1, 13.91987721825382: 1, 13.
879834880723994: 1, 13.870987986624208: 1, 13.809357579484445: 1, 13.80
4382756233906: 1, 13.77662287723564: 1, 13.744794925646092: 1, 13.73473
8914433201: 1, 13.731774054328511: 1, 13.712944118579673: 1, 13.6974501
730238: 1, 13.662672614659789: 1, 13.632116892332672: 1, 13.57384153228
5494: 1, 13.572738566334813: 1, 13.55687926188643: 1, 13.55190553859385
5: 1, 13.530414900916423: 1, 13.493116519704799: 1, 13.47312877405245:
1, 13.46510179191451: 1, 13.4063544008491: 1, 13.372861496865344: 1, 1
3.369093734830603: 1, 13.32440550601232: 1, 13.28848498110656: 1, 13.28
3532242053992: 1, 13.178994974581588: 1, 13.166635652625555: 1, 13.1631
21883583944: 1, 13.125359392374502: 1, 13.05992579281849: 1, 13.0149889
3527547: 1, 12.994349297996218: 1, 12.975195304430722: 1, 12.9006493575
551: 1, 12.877030239723682: 1, 12.873032114122053: 1, 12.7650024801536
6: 1, 12.7291702698511: 1, 12.727809409598965: 1, 12.676298109484078:
1, 12.66900878700117: 1, 12.642999298540834: 1, 12.62873352612986: 1, 1
2.59517405009057: 1, 12.589110166161607: 1, 12.56700174363219: 1, 12.56
6442147463595: 1, 12.535560909324886: 1, 12.459390892290882: 1, 12.4315
05774954543: 1, 12.419641085334977: 1, 12.405756005237471: 1, 12.403455
264542245: 1, 12.371897188756938: 1, 12.364632089484003: 1, 12.32688701
5308323: 1, 12.308140459084829: 1, 12.272123677262693: 1, 12.1963801217
72455: 1, 12.180076089945262: 1, 12.121248372885992: 1, 12.078277342641
451: 1, 12.072876802843675: 1, 12.065106287582664: 1, 12.00944234519602
3: 1, 11.994266271987529: 1, 11.981411801445502: 1, 11.979872878499075:
1, 11.952688801337684: 1, 11.949351404925888: 1, 11.913245307658963: 1,
11.91196272760507: 1, 11.905821651417183: 1, 11.891775086124138: 1, 11.
843810563628923: 1, 11.833765621510029: 1, 11.821363680698799: 1, 11.81
8911592715853: 1, 11.79643411644177: 1, 11.783055605330901: 1, 11.77071
5948724153: 1, 11.74215058405025: 1, 11.725508109373205: 1, 11.71296392
9378484: 1, 11.682066460815292: 1, 11.61792889284322: 1, 11.59155199744
2295: 1, 11.585168153411: 1, 11.584460092117979: 1, 11.576361592128098:
1, 11.565967031547233: 1, 11.52064091467485: 1, 11.517629789957883: 1,
11.50746226157981: 1, 11.498187338817042: 1, 11.489108601802936: 1, 11.
484882614916113: 1, 11.458610439074032: 1, 11.457360954323152: 1, 11.44
3796567477811: 1, 11.442143393905146: 1, 11.43881839940187: 1, 11.37811
2727792708: 1, 11.360329244679216: 1, 11.350478522720513: 1, 11.3353803
81274804: 1, 11.300172599885: 1, 11.265953040348151: 1, 11.252277441818
89: 1, 11.234994489193593: 1, 11.19488736576258: 1, 11.189514991171105:
```

1, 11.17942325929999: 1, 11.167259662044543: 1, 11.140164114436564: 1,
11.131580904631107: 1, 11.124781960345281: 1, 11.072662794793278: 1, 1
1.047031206635252: 1, 11.039744407879335: 1, 11.037763752077105: 1, 11.
030882418302493: 1, 11.016170552245462: 1, 10.959037202244941: 1, 10.95
1831017280464: 1, 10.947792493101463: 1, 10.94184577393278: 1, 10.91289
065736721: 1, 10.886164648489105: 1, 10.871013611129127: 1, 10.84413305
4213279: 1, 10.828541996561269: 1, 10.825570877456297: 1, 10.8111354940
4783: 1, 10.801534358132342: 1, 10.788165683064753: 1, 10.7631764174892
07: 1, 10.753568023870656: 1, 10.750291970136423: 1, 10.74285181540676
2: 1, 10.739414477289555: 1, 10.728138480998496: 1, 10.693120146953937:
1, 10.663610835622004: 1, 10.654454484226342: 1, 10.65018732932306: 1,
10.612409119389248: 1, 10.603743212105925: 1, 10.593168436628257: 1, 1
0.592236967361726: 1, 10.590120124191479: 1, 10.580727530699878: 1, 10.
571377641183012: 1, 10.569398223775746: 1, 10.55437381832713: 1, 10.531
856339216741: 1, 10.525221059766391: 1, 10.5141683610211: 1, 10.5020928
30489586: 1, 10.499673873564364: 1, 10.497947386950838: 1, 10.495738534
29706: 1, 10.465601246565287: 1, 10.457369219457957: 1, 10.442738722482
75: 1, 10.44262594602449: 1, 10.408726646089747: 1, 10.40682763969863:
1, 10.391907906827058: 1, 10.368953430728675: 1, 10.363713329149434: 1,
10.359446398032453: 1, 10.345464904213658: 1, 10.309293453479876: 1, 1
0.304491958348164: 1, 10.30247814549808: 1, 10.298345463816927: 1, 10.2
80279384071719: 1, 10.279407221654045: 1, 10.26133553801204: 1, 10.2015
78451980108: 1, 10.177925648955178: 1, 10.162540142192936: 1, 10.152806
953888815: 1, 10.147801259148832: 1, 10.135504689417719: 1, 10.13399524
9684897: 1, 10.117131313621295: 1, 10.114300293136813: 1, 10.0918470894
03654: 1, 10.072938694002618: 1, 10.053909728006152: 1, 10.041586780306
089: 1, 10.029283697645173: 1, 10.025678290976384: 1, 10.0138100549516
2: 1, 10.0101313860082: 1, 9.997006542835084: 1, 9.988101843645993: 1,
9.987567344155197: 1, 9.982440913263584: 1, 9.955647331979046: 1, 9.949
664569928855: 1, 9.948493670784574: 1, 9.92565761095384: 1, 9.916871282
553299: 1, 9.906647019178498: 1, 9.906339905887243: 1, 9.89905844664536
3: 1, 9.89603010576033: 1, 9.891228716826049: 1, 9.833528420293845: 1,
9.828893510953622: 1, 9.822176730563305: 1, 9.820606269743996: 1, 9.817
141688906787: 1, 9.809204839597799: 1, 9.797288180987236: 1, 9.79395094
0188328: 1, 9.732296713651605: 1, 9.71457001407349: 1, 9.7059766292747
3: 1, 9.682603796610602: 1, 9.66104602219281: 1, 9.655974937837607: 1,
9.636649660385988: 1, 9.603629419224484: 1, 9.587841769204173: 1, 9.581
864408351999: 1, 9.580725945986009: 1, 9.578368925519703: 1, 9.56330465
1540097: 1, 9.557649073119384: 1, 9.526660006143494: 1, 9.5162994299420

62: 1, 9.499152682730237: 1, 9.474382590377708: 1, 9.472862145665113:
1, 9.46208595677693: 1, 9.459206124479373: 1, 9.449458165832919: 1, 9.4
4913390502253: 1, 9.446328937762745: 1, 9.425143433180775: 1, 9.4247833
193142: 1, 9.411045497297145: 1, 9.399571133163796: 1, 9.34930627882757
3: 1, 9.326227558695091: 1, 9.319243221301425: 1, 9.29483463941915: 1,
9.290538096760509: 1, 9.277531542618728: 1, 9.258184758512026: 1, 9.256
467252560673: 1, 9.245593115269784: 1, 9.239270709802001: 1, 9.23658502
2440856: 1, 9.20824916212796: 1, 9.205400454578712: 1, 9.19108297715237
1: 1, 9.188838088817835: 1, 9.18121726198734: 1, 9.174616147804572: 1,
9.138748462648321: 1, 9.126231318412696: 1, 9.121729175480262: 1, 9.100
201884888369: 1, 9.096557717458618: 1, 9.086623076044484: 1, 9.04117115
5086035: 1, 8.985622639028035: 1, 8.969289581684945: 1, 8.9674855883509
9: 1, 8.958082929650113: 1, 8.946874248661855: 1, 8.9443668012858: 1,
8.864473173816137: 1, 8.813740942847572: 1, 8.806848652411167: 1, 8.798
059300789856: 1, 8.782609861641085: 1, 8.767767732979157: 1, 8.76274630
491917: 1, 8.755450290985644: 1, 8.748209232554412: 1, 8.7428099072917
6: 1, 8.737560815677229: 1, 8.736162348060077: 1, 8.72568963023154: 1,
8.720598559066737: 1, 8.694378541966666: 1, 8.688711567308411: 1, 8.682
176993865571: 1, 8.675282525505047: 1, 8.666630003535557: 1, 8.66282145
7026304: 1, 8.659375662647822: 1, 8.652845464300738: 1, 8.6341910456361
22: 1, 8.62498947401116: 1, 8.61647733154019: 1, 8.60549352597857: 1,
8.591394514122655: 1, 8.57645490657924: 1, 8.57169850969493: 1, 8.56444
5112826927: 1, 8.557684742031242: 1, 8.555113846180003: 1, 8.5155620400
5407: 1, 8.497749915673781: 1, 8.495214634998758: 1, 8.493439620136485:
1, 8.490105978192979: 1, 8.486985138629862: 1, 8.477661462187559: 1, 8.
455150130773673: 1, 8.451645932361481: 1, 8.43427148213964: 1, 8.432945
208501257: 1, 8.42320766865967: 1, 8.422329723580127: 1, 8.401312316321
498: 1, 8.396823394022055: 1, 8.358614565707038: 1, 8.351047495648311:
1, 8.349854090651103: 1, 8.33470480494858: 1, 8.331513122693995: 1, 8.3
15639839207323: 1, 8.310436013500082: 1, 8.303406184360627: 1, 8.301433
735832914: 1, 8.269545086398502: 1, 8.263049329638982: 1, 8.24591854827
4629: 1, 8.243268525361403: 1, 8.226141231113873: 1, 8.215663179403233:
1, 8.212722431951677: 1, 8.190477212812533: 1, 8.16494531415558: 1, 8.1
64771607147582: 1, 8.151709077405007: 1, 8.146614363016079: 1, 8.134569
703403326: 1, 8.1242998254023: 1, 8.096116947053861: 1, 8.0782366013779
08: 1, 8.068453946394394: 1, 8.0608822379481: 1, 8.049304930062583: 1,
8.048575792171727: 1, 8.042417659208088: 1, 8.031100843922165: 1, 8.015
107908959: 1, 7.996721471544086: 1, 7.981938949296295: 1, 7.98045920485
3747: 1, 7.980197924682142: 1, 7.9673664291481305: 1, 7.96601446549036

```
3: 1, 7.965557281582312: 1, 7.946993654655489: 1, 7.946152984383134: 1,
7.9318587415419: 1, 7.920828645838794: 1, 7.918285349496359: 1, 7.91648
7756084829: 1, 7.897987345264938: 1, 7.891888067805767: 1, 7.8775414453
67323: 1, 7.865237875429203: 1, 7.852416032484026: 1, 7.84589623569798
1: 1, 7.841621110338973: 1, 7.835919625169017: 1, 7.809288026393324: 1,
7.804300934405831: 1, 7.801722761574322: 1, 7.7923761759695145: 1, 7.77
3736324443038: 1, 7.765106723738948: 1, 7.7598151858975015: 1, 7.743716
025715095: 1, 7.7389688471572375: 1, 7.72664638064484: 1, 7.72417541220
67405: 1, 7.719610591411134: 1, 7.717976627188496: 1, 7.69735144634070
2: 1, 7.695489990966515: 1, 7.681184065491638: 1, 7.678650737991416: 1,
7.670503057150035: 1, 7.664399803666975: 1, 7.655650263751058: 1, 7.650
008502014257: 1, 7.637320697757972: 1, 7.633020736607399: 1, 7.62720085
5046613: 1, 7.6199411974765185: 1, 7.619620208449653: 1, 7.616310936726
95: 1, 7.602987652642577: 1, 7.600585098643631: 1, 7.592016769292826:
1, 7.587153750823654: 1, 7.586820380278969: 1, 7.5760080444861435: 1,
7.563226333165034: 1, 7.5584381082278345: 1, 7.557058695445271: 1, 7.55
6488942525074: 1, 7.55409276962923: 1, 7.552281682575681: 1, 7.55152752
3094947: 1, 7.540291393858504: 1, 7.526556026873334: 1, 7.5151044185828
71: 1, 7.5053832380779975: 1, 7.503161217931331: 1, 7.497987087826184:
1, 7.495279261553943: 1, 7.491161648444154: 1, 7.490883179953963: 1, 7.
489348802598097: 1, 7.483332792534539: 1, 7.483053230125304: 1, 7.47846
9611926781: 1, 7.470816974247713: 1, 7.467125083679907: 1, 7.4578893839
60194: 1, 7.450182280904755: 1, 7.431456026370562: 1, 7.41053907306974
5: 1, 7.410274369721174: 1, 7.406067135311 59: 1, 7.405656110003132: 1,
7.405462655052556: 1, 7.399663720325714: 1, 7.393994896849989: 1, 7.392
279527460169: 1, 7.388156345042179: 1, 7.3878645136839864: 1, 7.3842487
68794241: 1, 7.38375433561463: 1, 7.38031884963471: 1, 7.37745320603960
4: 1, 7.373952699803753: 1, 7.369735404511624: 1, 7.360854346791741: 1,
7.357698407362132: 1, 7.338535748856218: 1, 7.322571730700982: 1, 7.311
515921615299: 1, 7.309977647376214: 1, 7.291336687787669: 1, 7.28181608
6095766: 1, 7.281262886146553: 1, 7.279351258545435: 1, 7.2629526275808
125: 1, 7.256700857580349: 1, 7.245907249003431: 1, 7.239594781559556:
1, 7.228705867972533: 1, 7.22827409101796: 1, 7.223102772441144: 1, 7.2
18525421385012: 1, 7.216795237466048: 1, 7.21415357575656: 1, 7.2053012
49830462: 1, 7.202938228568511: 1, 7.196945732771252: 1, 7.193692110739
745: 1, 7.186836182028571: 1, 7.163534431536543: 1, 7.15824676333188:
1, 7.154187328174005: 1, 7.146316886300269: 1, 7.141187524132024: 1, 7.
124302860590513: 1, 7.120893498701799: 1, 7.119947444301326: 1, 7.11864
5371741589: 1, 7.118034785436082: 1, 7.115636526729924: 1, 7.1129268050
```

```
02209: 1, 7.109233531044128: 1, 7.102953271702486: 1, 7.10087699143721
6: 1, 7.098231179567717: 1, 7.096956335057927: 1, 7.096827256660423: 1,
7.085627666211384: 1, 7.083859001738705: 1, 7.083796982239664: 1, 7.071
366161533314: 1, 7.067874709561439: 1, 7.058382972002825: 1, 7.05517438
6300458: 1, 7.054254405582341: 1, 7.053528157881228: 1, 7.0495417862230
19: 1, 7.031573374490948: 1, 7.030303119496931: 1, 7.0198948129420975:
1, 7.017116496167492: 1, 7.016785145609126: 1, 7.015163561967328: 1, 7.
009402918133299: 1, 7.00652965438075: 1, 6.981660127299045: 1, 6.981576
2115390125: 1, 6.970740613808184: 1, 6.970466481482896: 1, 6.9697305492
531685: 1, 6.962660691025974: 1, 6.956676869173492: 1, 6.94130083999351
1: 1, 6.939196487219956: 1, 6.925185340563617: 1, 6.9011732348065475:
1, 6.896799104755959: 1, 6.893962319618326: 1, 6.89018120959777: 1, 6.8
84443034834783: 1, 6.882352314907151: 1, 6.868237239791497: 1, 6.854359
762779802: 1, 6.8492184412508585: 1, 6.848521959747222: 1, 6.8480918252
568115: 1, 6.846681904631802: 1, 6.846306142223529: 1, 6.84245809775231
8: 1, 6.813590590480605: 1, 6.810679897335827: 1, 6.809420952552229: 1,
6.807858486923639: 1, 6.792453702386779: 1, 6.786440822582535: 1, 6.782
398886466613: 1, 6.781813951441992: 1, 6.781270712237 3075: 1, 6.7811678
47613974: 1, 6.775114455883732: 1, 6.763311806958888: 1, 6.760916050984
408: 1, 6.756277179838968: 1, 6.75389061003962: 1, 6.749535571463773:
1, 6.744218704283839: 1, 6.743444746192891: 1, 6.7321042708512016: 1,
6.730199202435491: 1, 6.729766987085073: 1, 6.724260176174159: 1, 6.707
477010404431: 1, 6.699579937897173: 1, 6.683273835440315: 1, 6.67843753
8789375: 1, 6.672462305060319: 1, 6.66947911471677: 1, 6.6685746781843
6: 1, 6.663629541565799: 1, 6.6583237528595145: 1, 6.6511386039550455:
1, 6.650320144254095: 1, 6.641970618513702: 1, 6.637485422242792: 1, 6.
636685756226263: 1, 6.621399261654599: 1, 6.616954963577581: 1, 6.61682
5411572579: 1, 6.6166395119125605: 1, 6.615782473939032: 1, 6.605126916
441264: 1, 6.602074103689072: 1, 6.599576975174947: 1, 6.59680358748998
15: 1, 6.590944249782354: 1, 6.589936026996772: 1, 6.589730853282553:
1, 6.589565185057015: 1, 6.577433518450707: 1, 6.565508821002771: 1, 6.
547182327013253: 1, 6.5468635999650635: 1, 6.546136678388504: 1, 6.5404
32999047936: 1, 6.534214412903726: 1, 6.527556054104996: 1, 6.520097458
530752: 1, 6.518182767214457: 1, 6.511926545063422: 1, 6.50567676011983
6: 1, 6.4994810551489826: 1, 6.494481416448734: 1, 6.493279278474716:
1, 6.492257498007613: 1, 6.4807815578037244: 1, 6.477309188612408: 1,
6.475162218582432: 1, 6.473294286569705: 1, 6.47054550707886: 1, 6.4683
00671625446: 1, 6.462834841652907: 1, 6.4502372997231605: 1, 6.44444747
9096997: 1, 6.4441533271091105: 1, 6.441835886576394: 1, 6.439838485995
```

48: 1, 6.438534562020784: 1, 6.433827713362168: 1, 6.431357940732879:
1, 6.429488362542904: 1, 6.4284542711892305: 1, 6.424436441625344: 1,
6.417939692453682: 1, 6.411650067928337: 1, 6.39940707950077: 1, 6.3951
08313771473: 1, 6.386292810831648: 1, 6.385406974701535: 1, 6.384492650
459792: 1, 6.38114511572725: 1, 6.3735452386461935: 1, 6.37268387018794
4: 1, 6.367709372434352: 1, 6.363719210506029: 1, 6.361144587330882: 1,
6.35614588270069: 1, 6.3479350186490615: 1, 6.347921268425239: 1, 6.326
524346165006: 1, 6.3252522604993215: 1, 6.317679555247685: 1, 6.3130203
93492188: 1, 6.312064299784094: 1, 6.30985427104172: 1, 6.3087088016362
01: 1, 6.30679156370265: 1, 6.289836442034554: 1, 6.289486978881186: 1,
6.286824809293834: 1, 6.270734937450378: 1, 6.261520116625068: 1, 6.259
6928941476415: 1, 6.259232762560848: 1, 6.259091884633059: 1, 6.2525902
07730505: 1, 6.245955931295732: 1, 6.243745640339767: 1, 6.241948946170
518: 1, 6.240195907800665: 1, 6.23391464324329: 1, 6.222715808184161:
1, 6.21980630329911: 1, 6.217919119050733: 1, 6.211868137585593: 1, 6.2
08672667123396: 1, 6.207189344978898: 1, 6.204660753086153: 1, 6.203118
808415013: 1, 6.201707352804573: 1, 6.19629708043241: 1, 6.185821942921
822: 1, 6.184764617398206: 1, 6.183620993697036: 1, 6.18360510110225:
1, 6.17439242505981: 1, 6.17408383936227: 1, 6.164504890509403: 1, 6.16
3136204644396: 1, 6.153805780979536: 1, 6.152661922492252: 1, 6.1496403
30491367: 1, 6.149057442022856: 1, 6.147777058461805: 1, 6.141443772792
333: 1, 6.132974327171281: 1, 6.128803473736716: 1, 6.122535372361422:
1, 6.1161768371898635: 1, 6.115679100392982: 1, 6.113942236417286: 1,
6.1081069874145495: 1, 6.099270015696145: 1, 6.085764204298778: 1, 6.05
7312198821961: 1, 6.048018940732028: 1, 6.038474987572424: 1, 6.0353056
293265785: 1, 6.025917940117378: 1, 6.02380203190391: 1, 6.019789972618
099: 1, 6.0131219763108135: 1, 6.005343223988212: 1, 5.995154500250239:
1, 5.9845994474714095: 1, 5.97934371126239: 1, 5.975796281487311: 1, 5.
973821986460078: 1, 5.9711796565390225: 1, 5.969691909166858: 1, 5.9666
16708490492: 1, 5.965932655568126: 1, 5.965864143526524: 1, 5.965176185
6186045: 1, 5.964011429262142: 1, 5.960807094633372: 1, 5.9593798047321
815: 1, 5.957355976531288: 1, 5.955524823997184: 1, 5.9532401072084475:
1, 5.951614008369311: 1, 5.948308453523022: 1, 5.945186163637619: 1, 5.
936572358356324: 1, 5.931208991781317: 1, 5.927924386869252: 1, 5.92566
3364304915: 1, 5.924628945781626: 1, 5.913168170650861: 1, 5.9121429916
28018: 1, 5.908190709858932: 1, 5.907918984378394: 1, 5.90457561203600
2: 1, 5.904473637041063: 1, 5.895716393348701: 1, 5.894260260733862: 1,
5.891239049869604: 1, 5.889702171823263: 1, 5.8864953652495835: 1, 5.88
26221943223445: 1, 5.870082444667013: 1, 5.86680303387267: 1, 5.8640141

86688648: 1, 5.860947087655915: 1, 5.859897119821264: 1, 5.850899977186
751: 1, 5.848070954980744: 1, 5.845263102830848: 1, 5.839761739804372:
1, 5.834784967601931: 1, 5.825964143269118: 1, 5.824525424273094: 1, 5.
821967888129228: 1, 5.821738624385373: 1, 5.819505891254302: 1, 5.81647
6669841977: 1, 5.8126536789520005: 1, 5.810944679414618: 1, 5.808803151
001638: 1, 5.806964743287918: 1, 5.798090586674363: 1, 5.79772470735590
1: 1, 5.791874376548921: 1, 5.79165055943878: 1, 5.791459298677686: 1,
5.786124585532362: 1, 5.778731950157934: 1, 5.777799042008005: 1, 5.766
162967734821: 1, 5.765273878123386: 1, 5.750011112506679: 1, 5.74883275
6022794: 1, 5.734566812265799: 1, 5.729499763844493: 1, 5.7266685476868
88: 1, 5.726528600895631: 1, 5.717939574953383: 1, 5.717248246671256:
1, 5.717227349472368: 1, 5.712962544708291: 1, 5.709192760411799: 1, 5.
69413284453524: 1, 5.686213624402814: 1, 5.684994228762041: 1, 5.684212
122269737: 1, 5.684143552768215: 1, 5.67042599379641: 1, 5.669396971669
301: 1, 5.664866705008223: 1, 5.663890207203769: 1, 5.660724224688576:
1, 5.6599231796216385: 1, 5.6586903467752245: 1, 5.654636162334193: 1,
5.648333699690266: 1, 5.646260921646818: 1, 5.64325479486872: 1, 5.6421
39759837035: 1, 5.641592471791919: 1, 5.626717390606044: 1, 5.626170661
568747: 1, 5.622190844476381: 1, 5.61889618977519: 1, 5.61613606145424
7: 1, 5.614731857141484: 1, 5.614677565203578: 1, 5.609553591260202: 1,
5.601992718938834: 1, 5.598304708273368: 1, 5.59791943486119: 1, 5.5968
404888869365: 1, 5.592618576877221: 1, 5.590027441338485: 1, 5.58152128
0732189: 1, 5.5799952611677766: 1, 5.577811314388625: 1, 5.574929658808
125: 1, 5.572328927551889: 1, 5.566327714634518: 1, 5.565268634027176:
1, 5.5642762654741444: 1, 5.555088182339873: 1, 5.55458605652167: 1, 5.
544433865028185: 1, 5.534837585838456: 1, 5.532514521606033: 1, 5.53213
0076621749: 1, 5.52743342283643: 1, 5.52739660153653: 1, 5.523263328943
367: 1, 5.520813294869487: 1, 5.519836637749955: 1, 5.502981051977295:
1, 5.498994054952263: 1, 5.493194219937163: 1, 5.493052818144975: 1, 5.
4895031525541516: 1, 5.489448004650567: 1, 5.489036749505245: 1, 5.4885
73728622276: 1, 5.484976442796837: 1, 5.480420427202987: 1, 5.465409562
948443: 1, 5.460141939908597: 1, 5.458963353961462: 1, 5.45830609316408
3: 1, 5.450750854593178: 1, 5.446836889314573: 1, 5.431785958950451: 1,
5.428496792521504: 1, 5.427199473746833: 1, 5.425784375889575: 1, 5.423
365792378932: 1, 5.420938468840825: 1, 5.419959149715072: 1, 5.41747351
2649884: 1, 5.4172884285006875: 1, 5.417051115651733: 1, 5.412731599528
242: 1, 5.409146996980102: 1, 5.397249551157043: 1, 5.389158899707679:
1, 5.387051717190137: 1, 5.372485425681041: 1, 5.371758468057256: 1, 5.
36652333540759: 1, 5.361402061076825: 1, 5.358112173583782: 1, 5.355949

277125241: 1, 5.347918469018817: 1, 5.346113869341468: 1, 5.34192524635
6794: 1, 5.340743900078506: 1, 5.339154758107973: 1, 5.338833161928865:
1, 5.331953510995293: 1, 5.33169225746385: 1, 5.33099493020471: 1, 5.32
5221125427006: 1, 5.321916595028885: 1, 5.321578345424726: 1, 5.3213905
47199938: 1, 5.319273873373175: 1, 5.319076857041264: 1, 5.317356857995
353: 1, 5.314868125238756: 1, 5.3132956533053495: 1, 5.311530798115588:
1, 5.310435534175684: 1, 5.29788462312788: 1, 5.293470610400509: 1, 5.2
8718350192844: 1, 5.286986538549733: 1, 5.285578390780486: 1, 5.2738474
88726604: 1, 5.273744153027902: 1, 5.2720458483212695: 1, 5.27074783648
3544: 1, 5.270636427019209: 1, 5.263588809142021: 1, 5.255644954717626:
1, 5.253872588397512: 1, 5.248453109387911: 1, 5.246536477145827: 1, 5.
245726536614119: 1, 5.24424231701126: 1, 5.241092932275941: 1, 5.237472
610949083: 1, 5.2333650063308275: 1, 5.23307117859082: 1, 5.22986427138
6595: 1, 5.2295850534427775: 1, 5.228651146762342: 1, 5.22631442206377
2: 1, 5.221281668262087: 1, 5.215745565836915: 1, 5.21345816608039: 1,
5.21068362584213: 1, 5.208649015280727: 1, 5.201087521364286: 1, 5.1989
44333077071: 1, 5.194838058851867: 1, 5.194426916557171: 1, 5.193453820
491169: 1, 5.1896801472529885: 1, 5.187012425050285: 1, 5.1862569357014
33: 1, 5.183467668532036: 1, 5.175690552300633: 1, 5.1717700289763: 1,
5.16783299419011: 1, 5.1674360617414: 1, 5.166569200676666: 1, 5.164617
075909374: 1, 5.1604150477063335: 1, 5.160305424557091: 1, 5.1564988078
07257: 1, 5.14590733200427: 1, 5.137756208027191: 1, 5.131963584235795:
1, 5.1317925768365384: 1, 5.130854009640061: 1, 5.126448047692431: 1,
5.119518640675543: 1, 5.119247440685043: 1, 5.118346201613353: 1, 5.116
639862166282: 1, 5.115493776570733: 1, 5.110852907585538: 1, 5.10183860
84312874: 1, 5.099097066382567: 1, 5.091230424574872: 1, 5.088599281713
685: 1, 5.0884497716901045: 1, 5.087675486400721: 1, 5.086042833735116:
1, 5.0813549419209005: 1, 5.0763715309138675: 1, 5.076338636034307: 1,
5.073098021852852: 1, 5.070982521832636: 1, 5.061127070562226: 1, 5.055
692527331138: 1, 5.055092128907348: 1, 5.053447529771989: 1, 5.05315902
4723539: 1, 5.050749316816734: 1, 5.046937734060932: 1, 5.0437360325427
685: 1, 5.029161408821221: 1, 5.028960634378261: 1, 5.028892861120889:
1, 5.028580008245556: 1, 5.026997126706848: 1, 5.0233717964648985: 1,
5.0166458632866275: 1, 5.01550198708536: 1, 5.0135962963751695: 1, 5.00
9677067939039: 1, 5.008698370587759: 1, 4.999130040656996: 1, 4.9980155
58346326: 1, 4.988842325340276: 1, 4.988380963908532: 1, 4.987293587541
601: 1, 4.987244101936843: 1, 4.98371115869545: 1, 4.981447233229629:
1, 4.9782250054220025: 1, 4.973035418310976: 1, 4.9672159946634435: 1,
4.957492820185446: 1, 4.955552089069595: 1, 4.951075459774139: 1, 4.951

```
068341542201: 1, 4.949482710189842: 1, 4.946931880432081: 1, 4.94586917
5376867: 1, 4.94478978261649: 1, 4.940250593038775: 1, 4.93739857971206
2: 1, 4.928240132008919: 1, 4.927439903814207: 1, 4.92736903411645: 1,
4.926426736007852: 1, 4.925896789955964: 1, 4.912377343449353: 1, 4.907
418046590295: 1, 4.89838832477344: 1, 4.897950711489278: 1, 4.887961726
955614: 1, 4.886509519807138: 1, 4.885319638136784: 1, 4.88404754151167
7: 1, 4.8819600353143615: 1, 4.869124777192533: 1, 4.868605363346575:
1, 4.864279070698051: 1, 4.857088868609872: 1, 4.855749994185196: 1, 4.
853156712586899: 1, 4.8523928252185735: 1, 4.851825032465167: 1, 4.8517
18419019331: 1, 4.849422580872501: 1, 4.848822853356956: 1, 4.843070699
74495: 1, 4.841223709268365: 1, 4.840978188498376: 1, 4.83941024661615
6: 1, 4.838668166462098: 1, 4.837714224295738: 1, 4.835262864132722: 1,
4.824793444141937: 1, 4.824104609098554: 1, 4.8237378605639645: 1, 4.82
0109702333421: 1, 4.819098687238912: 1, 4.81623737716948: 1, 4.80875542
1849076: 1, 4.807943158315472: 1, 4.806995549011067: 1, 4.8012684585920
03: 1, 4.798566943255976: 1, 4.797474012083779: 1, 4.7973793890105405:
1, 4.797338925576474: 1, 4.795307490587768: 1, 4.791157769395133: 1, 4.
778172814087036: 1, 4.77519663276691: 1, 4.771891337701986: 1, 4.763821
554700625: 1, 4.760648589189281: 1, 4.7581576939525565: 1, 4.7568094595
44194: 1, 4.755309700930631: 1, 4.752383737080859: 1, 4.74195312984849
8: 1, 4.73818172050169: 1, 4.73511736800262: 1, 4.731763753819416: 1,
4.730575015453263: 1, 4.724985123711208: 1, 4.723630025189446: 1, 4.722
513941270072: 1, 4.722004006282845: 1, 4.719593840178594: 1, 4.71946476
9817808: 1, 4.71340832513276: 1, 4.6993899155312215: 1, 4.6992154031660
2: 1, 4.697275078256797: 1, 4.696180980695582: 1, 4.6958624306916725:
1, 4.6951067132820645: 1, 4.686863677861576: 1, 4.6850935598829455: 1,
4.683717568065039: 1, 4.68025593073268: 1, 4.67952131502837: 1, 4.67438
08653331955: 1, 4.67289581408026: 1, 4.666884274371661: 1, 4.6615065457
51556: 1, 4.6577763248770125: 1, 4.657480980682591: 1, 4.65546932888302
1: 1, 4.653227104731119: 1, 4.650906818579557: 1, 4.648624170569727: 1,
4.647203219855032: 1, 4.641332338408146: 1, 4.635946538410789: 1, 4.633
188046493202: 1, 4.631200132148464: 1, 4.630938249176155: 1, 4.63090976
1242911: 1, 4.63027700116948: 1, 4.630107367829438: 1, 4.62817235628628
2: 1, 4.624874047800055: 1, 4.618340447331521: 1, 4.616790391524543: 1,
4.613900664495922: 1, 4.613114045951764: 1, 4.612731514120489: 1, 4.611
46572519676: 1, 4.6092984965011805: 1, 4.60011817510633: 1, 4.596064565
029976: 1, 4.592418462212748: 1, 4.5892508455311765: 1, 4.5853608574046
97: 1, 4.583701432878269: 1, 4.582637211577766: 1, 4.570867043165529:
1, 4.567821603650744: 1, 4.566844082466635: 1, 4.564559894268146: 1, 4.
```

563507241646438: 1, 4.563393044980228: 1, 4.558384185291357: 1, 4.55746
2202047547: 1, 4.556466651848763: 1, 4.5493100459161955: 1, 4.546072555
796179: 1, 4.539758900199602: 1, 4.538442244714897: 1, 4.53161713535588
4: 1, 4.531248943212778: 1, 4.529992619700786: 1, 4.525043098205821: 1,
4.52484103388587: 1, 4.522798860466349: 1, 4.513896371259264: 1, 4.5074
90520458684: 1, 4.504412656355949: 1, 4.502346882099666: 1, 4.494538144
243029: 1, 4.48956525007098: 1, 4.488926309207651: 1, 4.48541740337572
1: 1, 4.484998710842562: 1, 4.481146402027119: 1, 4.478993699483537: 1,
4.478062720265495: 1, 4.476471926461059: 1, 4.475168470684252: 1, 4.474
549859388656: 1, 4.471561234568529: 1, 4.467367481624184: 1, 4.46556899
7543202: 1, 4.465125268623829: 1, 4.46479202808122: 1, 4.46427976033898
7: 1, 4.462421793715201: 1, 4.4550442627351385: 1, 4.452933018613565:
1, 4.440818446822249: 1, 4.437587218931459: 1, 4.437368141233835: 1, 4.
426628274287808: 1, 4.424921080805625: 1, 4.4182023781809425: 1, 4.4164
8652869061: 1, 4.4131551363802615: 1, 4.407852814589012: 1, 4.407596795
704237: 1, 4.405096062157261: 1, 4.404842487228165: 1, 4.40392369125937
3: 1, 4.401095547373407: 1, 4.399511035778008: 1, 4.397330413600422: 1,
4.396882274323723: 1, 4.395784576883639: 1, 4.392243787670544: 1, 4.390
205494239869: 1, 4.3859818529226: 1, 4.385006791712694: 1, 4.3846786867
26543: 1, 4.382182112202363: 1, 4.380647685137275: 1, 4.38053353801302
1: 1, 4.374094110292396: 1, 4.372354313076517: 1, 4.368272380777929: 1,
4.3674535610694845: 1, 4.3636849974636: 1, 4.363143283280557: 1, 4.3621
57505848819: 1, 4.361413642947495: 1, 4.360804354531557: 1, 4.359357405
90606: 1, 4.358884815731534: 1, 4.357144253467998: 1, 4.35592573511275
7: 1, 4.355434367447757: 1, 4.352270946174546: 1, 4.3502654374060885:
1, 4.347921026226137: 1, 4.343863429043561: 1, 4.340948685072524: 1, 4.
340892903589979: 1, 4.3407497597524785: 1, 4.3394329294630944: 1, 4.337
599720655127: 1, 4.3344483509717655: 1, 4.333636418975679: 1, 4.3321188
31246659: 1, 4.330154984120386: 1, 4.329478286549858: 1, 4.329130816901
1735: 1, 4.328617998816496: 1, 4.327091258931885: 1, 4.326522056108503:
1, 4.326216815089973: 1, 4.3245458979955105: 1, 4.324159189898356: 1,
4.319487680416344: 1, 4.304199329121868: 1, 4.303484966280457: 1, 4.301
864280270619: 1, 4.3011836080698105: 1, 4.2958290915609165: 1, 4.294451
810776356: 1, 4.293973474830071: 1, 4.291684306869705: 1, 4.28501091903
4052: 1, 4.2849193858097765: 1, 4.282931001868085: 1, 4.28263667867245:
1, 4.280662684085613: 1, 4.2751680301277215: 1, 4.273619895710951: 1,
4.272063398243165: 1, 4.271862281881399: 1, 4.27117861861892: 1, 4.2706
17382905092: 1, 4.269752068210088: 1, 4.269736561437082: 1, 4.266443275
185339: 1, 4.259248438799355: 1, 4.257505390512467: 1, 4.25544012985699

8: 1, 4.2554371080398585: 1, 4.254027221018147: 1, 4.2535275369620615:
1, 4.251495719972274: 1, 4.249652456711001: 1, 4.244556380690749: 1, 4.
24183390799171: 1, 4.23682528576124: 1, 4.236272035562794: 1, 4.2307619
76303683: 1, 4.229578780424956: 1, 4.2250727408415605: 1, 4.22363099449
9108: 1, 4.221228721860707: 1, 4.218352562326398: 1, 4.217556493862228
5: 1, 4.216896802755414: 1, 4.216594569542831: 1, 4.215579836636927: 1,
4.214997529822447: 1, 4.210141888413146: 1, 4.210055151330566: 1, 4.206
735869832119: 1, 4.20357288934467: 1, 4.202727475344819: 1, 4.202474669
783845: 1, 4.202378189062457: 1, 4.202032253883166: 1, 4.20032566842503
9: 1, 4.197158918753552: 1, 4.192901988559469: 1, 4.191799187471152: 1,
4.188739335120437: 1, 4.18424755469158: 1, 4.1830075895381365: 1, 4.180
307729897433: 1, 4.178873879364551: 1, 4.176855353907757: 1, 4.17564150
8269351: 1, 4.172516921664542: 1, 4.169884838683446: 1, 4.1668559167588
6: 1, 4.165724298986648: 1, 4.16022439512939: 1, 4.159245869235046: 1,
4.157420307085709: 1, 4.145949077570805: 1, 4.139308798750269: 1, 4.134
604539084944: 1, 4.133706496025098: 1, 4.132374724372992: 1, 4.13217865
10125325: 1, 4.131964511864892: 1, 4.131521347791421: 1, 4.129365234114
558: 1, 4.12804585117706: 1, 4.122906254050055: 1, 4.121013885810388:
1, 4.119324993893078: 1, 4.116345089137342: 1, 4.116017608045279: 1, 4.
114424294405742: 1, 4.111295101615636: 1, 4.107156618704: 1, 4.10472074
9423556: 1, 4.100917833632787: 1, 4.09977973673457: 1, 4.09434702255498
9: 1, 4.093213974699187: 1, 4.092173926033595: 1, 4.091824953127192: 1,
4.0888346733721175: 1, 4.087822129416407: 1, 4.087739245267526: 1, 4.08
4010197241695: 1, 4.081766868936052: 1, 4.077098246123516: 1, 4.0767400
81344808: 1, 4.075767599071028: 1, 4.074329778131421: 1, 4.074122522977
622: 1, 4.072704177935373: 1, 4.069543260815259: 1, 4.065309895194635:
1, 4.06476224609717: 1, 4.064576598897213: 1, 4.063128654333388: 1, 4.0
62234380526457: 1, 4.060644934619858: 1, 4.059926029948916: 1, 4.058724
792665465: 1, 4.058474221475285: 1, 4.057497805724841: 1, 4.05653675467
9812: 1, 4.055535477114511: 1, 4.051574332392813: 1, 4.045288328430074
5: 1, 4.043633566828985: 1, 4.0425144782650495: 1, 4.035809380109554:
1, 4.035562532021689: 1, 4.034868103160273: 1, 4.0271749450750285: 1,
4.024642976911769: 1, 4.023565956533119: 1, 4.023500822322821: 1, 4.023
326532206653: 1, 4.023171731908567: 1, 4.022671581074831: 1, 4.01749711
0310806: 1, 4.0169136908443255: 1, 4.0157430165651355: 1, 4.01367766052
9007: 1, 4.01311931302021: 1, 4.008655435367059: 1, 4.006718754694016:
1, 4.0061911691035395: 1, 4.004841263281509: 1, 4.004682405606156: 1,
4.004665436160477: 1, 4.004586230522428: 1, 3.999090368576459: 1, 3.995
109778519113: 1, 3.9934936659690954: 1, 3.992821409741063: 1, 3.9899014

22219081: 1, 3.986565833363851: 1, 3.985828321439214: 1, 3.982531218843
646: 1, 3.9818772280651458: 1, 3.979015528274284: 1, 3.978838574844375
8: 1, 3.976554040849674: 1, 3.972391285109053: 1, 3.972343858365611: 1,
3.9701700560159985: 1, 3.9692873010512812: 1, 3.966389660728542: 1, 3.9
658693062270354: 1, 3.9644503518292744: 1, 3.9639925414057062: 1, 3.961
296691845931: 1, 3.960934082305797: 1, 3.958533701980035: 1, 3.95826835
83396263: 1, 3.958224594288631: 1, 3.9552762143849587: 1, 3.95409698225
55168: 1, 3.9523914853308533: 1, 3.950881540618032: 1, 3.9464946042169
2: 1, 3.9453747049875574: 1, 3.9434144442047447: 1, 3.940516719773287:
1, 3.938156563001733: 1, 3.9374516276623903: 1, 3.936721757308407: 1,
3.93633737005641: 1, 3.9323601963588937: 1, 3.9318903954533333: 1, 3.93
14159139668567: 1, 3.9286476501726963: 1, 3.92517179891703: 1, 3.924250
63752977: 1, 3.921595995815276: 1, 3.9211084227999335: 1, 3.91738290059
20204: 1, 3.9159789169932275: 1, 3.914087904557292: 1, 3.91362301107544
3: 1, 3.913600094122826: 1, 3.9094575537904146: 1, 3.9075449125865562:
1, 3.9057870741539813: 1, 3.9031290958117015: 1, 3.898501077167373: 1,
3.898274456320576: 1, 3.8976803743528006: 1, 3.89566447165123: 1, 3.895
0901948618033: 1, 3.8943748546804784: 1, 3.892960342925445: 1, 3.891314
952211704: 1, 3.888510114793311: 1, 3.887791339609072: 1, 3.88542386151
5784: 1, 3.8844322798751487: 1, 3.881561841129961: 1, 3.881418171549981
4: 1, 3.8797251858136272: 1, 3.879631153405228: 1, 3.877119782132243:
1, 3.876746472415224: 1, 3.876567539624933: 1, 3.870869387382846: 1, 3.
8688005902652827: 1, 3.8686331124288285: 1, 3.8668045563715143: 1, 3.86
64799274068757: 1, 3.8617705287694344: 1, 3.860519073147976: 1, 3.85877
2867248724: 1, 3.8557933716958255: 1, 3.854594529668714: 1, 3.851350296
7161664: 1, 3.8507699281101924: 1, 3.8496924334233986: 1, 3.84564609329
32686: 1, 3.8409961423080627: 1, 3.839472121623147: 1, 3.83404558174962
2: 1, 3.8332169328268306: 1, 3.833078457219505: 1, 3.832656645633888:
1, 3.829155581330812: 1, 3.825603131611268: 1, 3.8237737740471065: 1,
3.8200389099730323: 1, 3.818360752427663: 1, 3.8165667523574514: 1, 3.8
093469639387374: 1, 3.8056739215673736: 1, 3.805399904401649: 1, 3.8038
0648553129: 1, 3.8020655423961545: 1, 3.8012900227178044: 1, 3.80073327
45059896: 1, 3.80062397713302: 1, 3.8003214207864042: 1, 3.794598827311
1926: 1, 3.793705282142667: 1, 3.786135615600966: 1, 3.784141681390297
5: 1, 3.7838725145436403: 1, 3.782829585132382: 1, 3.781346689614657:
1, 3.7809754563470257: 1, 3.780135522935159: 1, 3.7790997762074063: 1,
3.776562441487809: 1, 3.7726287516941093: 1, 3.769780292805931: 1, 3.76
75853309200478: 1, 3.7619077415326223: 1, 3.7611031266573662: 1, 3.7606
244398955506: 1, 3.7576588165357157: 1, 3.7549688366168668: 1, 3.754787

```
952943259: 1, 3.7531863733896826: 1, 3.7526371998243335: 1, 3.748590722
1020773: 1, 3.747071630558613: 1, 3.7457536254761807: 1, 3.745288100935
458: 1, 3.742543504976058: 1, 3.738507876667389: 1, 3.7382667728976497:
1, 3.7381086665072054: 1, 3.737846047842764: 1, 3.735118532392605: 1,
3.735002040221243: 1, 3.734997952035223: 1, 3.7327471194986055: 1, 3.72
9209415352647: 1, 3.729136227274819: 1, 3.7282090975226745: 1, 3.726887
7076759326: 1, 3.7261538734143: 1, 3.724937884829269: 1, 3.723374346945
9286: 1, 3.7197070319326255: 1, 3.7176054764102076: 1, 3.71349921485036
8: 1, 3.712357972581992: 1, 3.7115668902066714: 1, 3.710253450233551:
1, 3.708959383717473: 1, 3.7088792089871476: 1, 3.7048422734602053: 1,
3.7022513937088193: 1, 3.7002523126335496: 1, 3.6958573421668914: 1, 3.
694412317766006: 1, 3.6935143107200457: 1, 3.692620843896151: 1, 3.6916
03677745641: 1, 3.6911561512318882: 1, 3.6900800941615377: 1, 3.6897850
473020695: 1, 3.6879239110540407: 1, 3.6878048865148094: 1, 3.687716932
227701: 1, 3.686001658768212: 1, 3.684682264441017: 1, 3.68463894006922
34: 1, 3.6844031094342906: 1, 3.6842505870169977: 1, 3.680211337896184
7: 1, 3.679368173832737: 1, 3.6749028555176144: 1, 3.6739076556271257:
1, 3.6703835342624265: 1, 3.670338497603081: 1, 3.6699558609366227: 1,
3.668307690673594: 1, 3.6677319079489905: 1, 3.661340980668705: 1, 3.66
02636732545615: 1, 3.6584607638001496: 1, 3.6579688257832847: 1, 3.6557
402388455396: 1, 3.651121301986729: 1, 3.6511114064909695: 1, 3.6508088
453042733: 1, 3.6501764483204404: 1, 3.6466975940580495: 1, 3.645953937
790478: 1, 3.6419412711240287: 1, 3.64110645776737: 1, 3.64030684426687
36: 1, 3.6356017689960063: 1, 3.63555312952088: 1, 3.6339010168259374:
1, 3.6324278762004667: 1, 3.6323671819706194: 1, 3.6312039376914442: 1,
3.6261810174796745: 1, 3.6240373905627115: 1, 3.621866011092486: 1, 3.6
212238154368883: 1, 3.620892082417895: 1, 3.620159717553621: 1, 3.61927
5929112669: 1, 3.6178563782589146: 1, 3.6171577768422023: 1, 3.61585936
3199207: 1, 3.614327594553122: 1, 3.6139058930513235: 1, 3.613743770074
424: 1, 3.6134811990908053: 1, 3.612618420917284: 1, 3.612281140724154
4: 1, 3.6104823062640614: 1, 3.607670766244066: 1, 3.6073390965801737:
1, 3.6063582322494923: 1, 3.5943037716838813: 1, 3.5870983794126143: 1,
3.583717577619502: 1, 3.582696875119826: 1, 3.5824558225056267: 1, 3.58
11732464133073: 1, 3.578820783043281: 1, 3.5785309815961908: 1, 3.57734
68543037326: 1, 3.576963887335235: 1, 3.572533480842173: 1, 3.572495908
375657: 1, 3.5709914020504794: 1, 3.5683498049629256: 1, 3.567412503814
762: 1, 3.5637576010254084: 1, 3.5599497897502257: 1, 3.55984307584109
2: 1, 3.559061944774527: 1, 3.554168104469427: 1, 3.5535815580608645:
1, 3.551766171746419: 1, 3.5506732798706055: 1, 3.5486866896663463: 1,
```

3.5486155408019706: 1, 3.547391528269778: 1, 3.5455936594816952: 1, 3.5
432854170768757: 1, 3.5419931486208798: 1, 3.5411712050677804: 1, 3.539
64063163679: 1, 3.535709588716238: 1, 3.5288362850689676: 1, 3.52474836
84852097: 1, 3.5199196908077264: 1, 3.5194738179976737: 1, 3.5168139288
71777: 1, 3.5147381683286514: 1, 3.5137182176178303: 1, 3.5135056216886
53: 1, 3.510942032314489: 1, 3.509632030662967: 1, 3.5050845081854383:
1, 3.504736181609397: 1, 3.504065189975363: 1, 3.5029662815600036: 1,
3.502824090910406: 1, 3.5022788335331705: 1, 3.501829510089549: 1, 3.50
17888262981205: 1, 3.5010465625038796: 1, 3.5001800106809617: 1, 3.4996
379380600295: 1, 3.497612948621804: 1, 3.4968704940603943: 1, 3.4926003
0322675: 1, 3.4920259057742475: 1, 3.492011427537747: 1, 3.487595640787
2257: 1, 3.4835562732935057: 1, 3.483318293657681: 1, 3.48291995349048
3: 1, 3.481605449687038: 1, 3.4779448974729212: 1, 3.4750944340513095:
1, 3.473715856580387: 1, 3.473203323825363: 1, 3.4720856355913288: 1,
3.469319415482293: 1, 3.469073584085366: 1, 3.4680986057843897: 1, 3.46
31260851724126: 1, 3.4623449958628356: 1, 3.461809830224126: 1, 3.45773
95757080596: 1, 3.4547581078837366: 1, 3.4545498984989367: 1, 3.4525441
04651199: 1, 3.448986388056403: 1, 3.4480989248706337: 1, 3.44781933363
88414: 1, 3.446861256336863: 1, 3.4455581915189324: 1, 3.44456175344333
5: 1, 3.4438585733532947: 1, 3.443739579392049: 1, 3.4435917249258505:
1, 3.4427250561260725: 1, 3.4417315600387455: 1, 3.441719007325845: 1,
3.441403731493828: 1, 3.441388026736061: 1, 3.4404264904779995: 1, 3.43
93057100385693: 1, 3.4390743169489997: 1, 3.438914873530798: 1, 3.43695
5794695934: 1, 3.435286748891081: 1, 3.4326411423108407: 1, 3.431201468
432112: 1, 3.430607666619865: 1, 3.4305969458636567: 1, 3.4300869101299
36: 1, 3.4279524740022276: 1, 3.4275287031513244: 1, 3.427078937787950
3: 1, 3.4236153881722933: 1, 3.423098983109801: 1, 3.4224803062147613:
1, 3.421105030795508: 1, 3.416578586164006: 1, 3.415917170173871: 1, 3.
414959474724485: 1, 3.413036859311482: 1, 3.4121048908272353: 1, 3.4120
28059123315: 1, 3.4093429354367237: 1, 3.4089333673694506: 1, 3.4080180
317466664: 1, 3.4062683754483647: 1, 3.4043587426794546: 1, 3.402221800
693524: 1, 3.3997101945198547: 1, 3.397198702540904: 1, 3.3944473700452
51: 1, 3.3928885949856475: 1, 3.391278233874227: 1, 3.389044357693529:
1, 3.3887802734686296: 1, 3.380454474872229: 1, 3.3798991941352785: 1,
3.37971222433889: 1, 3.3769227187307846: 1, 3.3766888871279885: 1, 3.37
6443792177426: 1, 3.3743481927727426: 1, 3.3730079788830674: 1, 3.36888
72224671744: 1, 3.3675235920813344: 1, 3.366591481991823: 1, 3.36654257
0265315: 1, 3.3664244763825284: 1, 3.3644450130895094: 1, 3.36319213493
363: 1, 3.3631362095739026: 1, 3.361036426670721: 1, 3.357636695442147

3: 1, 3.356154079268517: 1, 3.3561452202566557: 1, 3.351573124594972:
1, 3.3495413692039198: 1, 3.3484070921717377: 1, 3.3475516502200793: 1,
3.346714238563815: 1, 3.3465078905208694: 1, 3.345674537859494: 1, 3.34
4991943181955: 1, 3.3413141307473375: 1, 3.3393131405536534: 1, 3.33909
2433307886: 1, 3.3389348200223212: 1, 3.3356272224967607: 1, 3.33408173
96032265: 1, 3.333995127509181: 1, 3.3320308867326487: 1, 3.33119041539
8105: 1, 3.330062180952956: 1, 3.329860867639211: 1, 3.329724408281476:
1, 3.3284922253021607: 1, 3.3273909125902064: 1, 3.327105729100844: 1,
3.321650839421083: 1, 3.3203307655451413: 1, 3.3199585249498407: 1, 3.3
1622494839028: 1, 3.3158100687083354: 1, 3.31556100891749: 1, 3.3102168
364493627: 1, 3.3077261647126086: 1, 3.307531491290594: 1, 3.3065315549
6968: 1, 3.305932027811336: 1, 3.3014515501414876: 1, 3.301061961948134
3: 1, 3.2993339809273032: 1, 3.29854073497038: 1, 3.296838182716331: 1,
3.296059803027928: 1, 3.295197276046001: 1, 3.2908812134748344: 1, 3.28
74360553865145: 1, 3.285368808419963: 1, 3.2833115109853943: 1, 3.28234
86177212056: 1, 3.2815331922182063: 1, 3.279800122368224: 1, 3.27786168
40101438: 1, 3.2771315269927466: 1, 3.2767582181106683: 1, 3.2766443981
219044: 1, 3.2759489972846874: 1, 3.2749980531948: 1, 3.27465762881174
7: 1, 3.271839985007911: 1, 3.2706716365515898: 1, 3.269452598264712:
1, 3.265672920965075: 1, 3.265176456924782: 1, 3.263129910955063: 1, 3.
2629610723918523: 1, 3.262916440998577: 1, 3.261345602133038: 1, 3.2605
42955323619: 1, 3.259372969609293: 1, 3.256893676000616: 1, 3.255021480
5835744: 1, 3.2543697117314814: 1, 3.253651579197784: 1, 3.253527964610
6696: 1, 3.25344296583537: 1, 3.2534321461648212: 1, 3.252351669211978:
1, 3.250898660146266: 1, 3.2502167124478647: 1, 3.249203520700514: 1,
3.2482267626473256: 1, 3.2468108907213504: 1, 3.2457922892913964: 1, 3.
2452665401753724: 1, 3.242638606785873: 1, 3.2396495210165135: 1, 3.239
204622605998: 1, 3.239169968795941: 1, 3.2387005793312094: 1, 3.2379965
560234347: 1, 3.237489340671834: 1, 3.236987358153404: 1, 3.23596806827
22236: 1, 3.235347071502239: 1, 3.234640079155051: 1, 3.233871485218568
6: 1, 3.233287025925464: 1, 3.232572316444374: 1, 3.231855361920728: 1,
3.2271338063036894: 1, 3.2254425800008963: 1, 3.2201428533369176: 1, 3.
2080411048079442: 1, 3.2074242632741417: 1, 3.2050834862350186: 1, 3.20
4923578015348: 1, 3.20431581868945: 1, 3.20406231911306: 1, 3.200296643
189824: 1, 3.1984595457955143: 1, 3.190402284794014: 1, 3.1896509110245
96: 1, 3.187156254135114: 1, 3.1868578045302205: 1, 3.185708935246394:
1, 3.184708669554376: 1, 3.184141377784284: 1, 3.1839094586005174: 1,
3.182680442638883: 1, 3.181491525612076: 1, 3.1813806507582156: 1, 3.17
7922255519959: 1, 3.1771935773184046: 1, 3.1757919745186904: 1, 3.17576

77446787342: 1, 3.175127801924099: 1, 3.1746995292334357: 1, 3.17367055
70601695: 1, 3.172450826611869: 1, 3.1705441988740017: 1, 3.16914290066
726: 1, 3.165498475069849: 1, 3.165231244947113: 1, 3.1636769270367147:
1, 3.1612524339367867: 1, 3.1591150494768887: 1, 3.1584263074552617: 1,
3.1536547684132645: 1, 3.150710383064332: 1, 3.149914455860425: 1, 3.14
8114709527243: 1, 3.147095494192429: 1, 3.145909852671775: 1, 3.1452919
34090479: 1, 3.1447331882035905: 1, 3.1443843175120634: 1, 3.1428743253
70993: 1, 3.1426188880904142: 1, 3.142375762465001: 1, 3.14101585427839
15: 1, 3.1409562411748877: 1, 3.1406083376389056: 1, 3.136507918695852:
1, 3.1360964849841353: 1, 3.1352753530153343: 1, 3.1340184359408125: 1,
3.13334466736081: 1, 3.1287523925324456: 1, 3.12763869253648: 1, 3.1266
778909713193: 1, 3.1252410287712706: 1, 3.124072427809015: 1, 3.1225242
27934381: 1, 3.121389472411394: 1, 3.1203584797937345: 1, 3.11999485493
1307: 1, 3.118595302204862: 1, 3.115810484053062: 1, 3.107145647548757:
1, 3.1066508783480455: 1, 3.106112161077776: 1, 3.10351182692857: 1, 3.
103496762872079: 1, 3.1030602792065087: 1, 3.0983899372355337: 1, 3.096
783384047361: 1, 3.0959275814244007: 1, 3.0956037537027723: 1, 3.094892
350163146: 1, 3.092451724147608: 1, 3.0898714890666765: 1, 3.0864974070
385998: 1, 3.0857983257260746: 1, 3.084064849266151: 1, 3.0833001037633
334: 1, 3.082902779979273: 1, 3.082799193736091: 1, 3.080810403500259:
1, 3.080554327268835: 1, 3.079032860392924: 1, 3.078474687956599: 1, 3.
0761819788317446: 1, 3.0733700272561455: 1, 3.0732097138594785: 1, 3.07
13651148200216: 1, 3.070401768895764: 1, 3.0679499397256427: 1, 3.06721
30787262746: 1, 3.0668838751460896: 1, 3.0666738978291535: 1, 3.0654437
27517596: 1, 3.063666731600132: 1, 3.063399302432436: 1, 3.059939462303
9: 1, 3.059893022772384: 1, 3.0558401591826976: 1, 3.048922413268808:
1, 3.04723173177302: 1, 3.047033960017771: 1, 3.04167365263506: 1, 3.03
6282842506986: 1, 3.035835633363614: 1, 3.0345926654246886: 1, 3.034206
7448206946: 1, 3.0324420326781145: 1, 3.0315789709981695: 1, 3.02959877
25684883: 1, 3.0277130642012136: 1, 3.027490878990351: 1, 3.02466365039
29936: 1, 3.023362947428766: 1, 3.0231953415986057: 1, 3.02306676865371
54: 1, 3.022762413943856: 1, 3.0210115540559084: 1, 3.0208980050041245:
1, 3.0201540608481086: 1, 3.019813717138305: 1, 3.0189084194006672: 1,
3.01811503815035: 1, 3.0180160925684425: 1, 3.0171521352609: 1, 3.01463
6816693361: 1, 3.013760901791418: 1, 3.010921146217516: 1, 3.0085781584
208973: 1, 3.0082934096862597: 1, 3.008106940629522: 1, 3.0067957783204
675: 1, 3.0056591423046948: 1, 3.0055502758631514: 1, 3.000498636513031
5: 1, 3.0000202096237265: 1, 2.9969461264945223: 1, 2.996576113677568:
1, 2.994976387096519: 1, 2.9933001324240864: 1, 2.9921528640127435: 1,

2.991815001632523: 1, 2.9873933291976367: 1, 2.9873649302801812: 1, 2.9
85585256789627: 1, 2.9825449902721077: 1, 2.9798572070868223: 1, 2.9767
812760409225: 1, 2.975794961465241: 1, 2.9754180099533167: 1, 2.9747260
914201896: 1, 2.9746501233589373: 1, 2.970682222981694: 1, 2.9672307656
256747: 1, 2.9670544075002554: 1, 2.9667933045402584: 1, 2.964944141840
208: 1, 2.9642164279191086: 1, 2.962694636224093: 1, 2.960630210107094:
1, 2.959011810082371: 1, 2.9584907300574: 1, 2.957703202386564: 1, 2.95
38153733817247: 1, 2.9532187871371676: 1, 2.950992842191962: 1, 2.95040
38949463527: 1, 2.9493548365031765: 1, 2.949289689633185: 1, 2.94905727
08504335: 1, 2.946153034414958: 1, 2.944758387241913: 1, 2.941369128492
457: 1, 2.9408915996816685: 1, 2.9402262539010113: 1, 2.93988948756966
2: 1, 2.936791287030259: 1, 2.936600033301047: 1, 2.9353797694515356:
1, 2.9293154722412145: 1, 2.9292421861494593: 1, 2.9280344300156567: 1,
2.9268957441388093: 1, 2.9261608974976943: 1, 2.9260812991426968: 1, 2.
9259774752714907: 1, 2.92543265647893: 1, 2.925355442553664: 1, 2.92474
98359499478: 1, 2.9219193044501313: 1, 2.9217696288646033: 1, 2.9209912
709848225: 1, 2.9207458165463853: 1, 2.9202585573689026: 1, 2.918514605
1775064: 1, 2.918308298288411: 1, 2.9162964071380206: 1, 2.915563327611
5536: 1, 2.9148388158399285: 1, 2.913841856381901: 1, 2.913367225766920
3: 1, 2.911973851519623: 1, 2.911331722157839: 1, 2.910028002033472: 1,
2.906540698685203: 1, 2.906285901471037: 1, 2.9062615688010136: 1, 2.90
60922851205073: 1, 2.9020041060406445: 1, 2.9017189289467322: 1, 2.8967
73642396353: 1, 2.8944907840690135: 1, 2.893732577202008: 1, 2.89358721
17092267: 1, 2.893354619702355: 1, 2.892513510391566: 1, 2.891822241716
2514: 1, 2.8887540042677053: 1, 2.886645766137062: 1, 2.88620567195312
3: 1, 2.8850727261668903: 1, 2.885050334496657: 1, 2.884672005322727:
1, 2.8837902365220827: 1, 2.8818677972910667: 1, 2.881403614220625: 1,
2.879916765724155: 1, 2.8794793774704077: 1, 2.8791361804993194: 1, 2.8
738553824801993: 1, 2.873703921006765: 1, 2.8730806642048106: 1, 2.8722
807264193215: 1, 2.871989636436989: 1, 2.869606121376593: 1, 2.86953468
52003754: 1, 2.869437765399846: 1, 2.8689002408371187: 1, 2.86832342142
72573: 1, 2.868052589681617: 1, 2.8675049169362947: 1, 2.86480246076322
1: 1, 2.8628290905368146: 1, 2.8627550596382556: 1, 2.862184433643465:
1, 2.860194047771084: 1, 2.8597849449217128: 1, 2.858020775199226: 1,
2.8573902008923757: 1, 2.856796373053154: 1, 2.856584091126489: 1, 2.85
56461397485298: 1, 2.854187179451028: 1, 2.849025586295513: 1, 2.848747
9018788004: 1, 2.847915498095191: 1, 2.8473147570168784: 1, 2.844361855
370355: 1, 2.843722792240625: 1, 2.843307333660319: 1, 2.83881565457235
75: 1, 2.8383498825155242: 1, 2.8380687319629283: 1, 2.836362866485301:

```
1, 2.8357609947168267: 1, 2.8348473683337216: 1, 2.834517895042088: 1,
2.834129786839913: 1, 2.832260282602267: 1, 2.8278261135539395: 1, 2.82
51999336136824: 1, 2.823585176358341: 1, 2.8204445774379594: 1, 2.81713
92481054562: 1, 2.816822732503385: 1, 2.81557867712638: 1, 2.8153261983
729108: 1, 2.815237835025835: 1, 2.8124620583192343: 1, 2.8112669211961
93: 1, 2.8078019542278927: 1, 2.8067072262685078: 1, 2.806048557428775
2: 1, 2.805844203855156: 1, 2.804914165810109: 1, 2.801021656931687: 1,
2.8006330863617115: 1, 2.798740687301017: 1, 2.796135844158974: 1, 2.79
5966137960157: 1, 2.791735480121705: 1, 2.7902714730323708: 1, 2.789829
1993087048: 1, 2.7895430203545963: 1, 2.785478602127708: 1, 2.784987399
6351877: 1, 2.7833861489787104: 1, 2.779583366642216: 1, 2.777201555349
8997: 1, 2.776878294097641: 1, 2.7761597456341853: 1, 2.77452029460307
8: 1, 2.77413538742849: 1, 2.7716642268782032: 1, 2.7709349617935732:
1, 2.769360200564789: 1, 2.769150875928601: 1, 2.7662702171423077: 1,
2.7648105707125175: 1, 2.763898087694553: 1, 2.7614418030134567: 1, 2.7
59863743942996: 1, 2.759405639466903: 1, 2.758672505505835: 1, 2.755420
711171156: 1, 2.7551813930052127: 1, 2.755050993927327: 1, 2.7547653114
73496: 1, 2.754303361755265: 1, 2.752611834926206: 1, 2.751250712698040
3: 1, 2.751190040920561: 1, 2.749116623304598: 1, 2.7476459166296636:
1, 2.747633142830134: 1, 2.7467423807241764: 1, 2.7465699248176607: 1,
2.7462207647954884: 1, 2.7460753138235945: 1, 2.7439156926822457: 1, 2.
743800021807608: 1, 2.743648801511603: 1, 2.7432916912021197: 1, 2.7420
430079773057: 1, 2.741665654685459: 1, 2.739834705716354: 1, 2.73982080
8390675: 1, 2.7392421622045475: 1, 2.7384300926483998: 1, 2.72977844924
36197: 1, 2.728135039490941: 1, 2.7279106182819692: 1, 2.72736166973425
3: 1, 2.7269237317233: 1, 2.7264078330482184: 1, 2.7256993960861187: 1,
2.7253524907172584: 1, 2.7246882980265656: 1, 2.721143541076925: 1, 2.7
200055665411607: 1, 2.719125882202175: 1, 2.7173652066265346: 1, 2.7158
8595967954: 1, 2.7141067481649603: 1, 2.713140135623212: 1, 2.710285927
846046: 1, 2.7082218365636894: 1, 2.7071645420135035: 1, 2.706953953227
1825: 1, 2.7064189629705786: 1, 2.7043856670177675: 1, 2.70360452830982
3: 1, 2.703262098409833: 1, 2.7031017361252236: 1, 2.70089844136122: 1,
2.6983362171205867: 1, 2.6975396825762172: 1, 2.6973024593165786: 1, 2.
695949420178255: 1, 2.693361107999162: 1, 2.693158577568269: 1, 2.69299
8393987329: 1, 2.6919655876699347: 1, 2.6917342961485313: 1, 2.69167230
69497985: 1, 2.6909539409487526: 1, 2.6909132817534887: 1, 2.6896483454
390783: 1, 2.6878059960313383: 1, 2.6863669130593264: 1, 2.682580760530
422: 1, 2.6820188279222426: 1, 2.681206074515649: 1, 2.680574162336071
3: 1, 2.6794519950940394: 1, 2.6788640344796173: 1, 2.6787985755355916:
```

1, 2.677930181714943: 1, 2.6737239440946627: 1, 2.673612104309326: 1,
2.672926420142387: 1, 2.6728754025713917: 1, 2.67234998521703: 1, 2.671
4035968405816: 1, 2.669597405172706: 1, 2.668188950847102: 1, 2.6679298
093561483: 1, 2.667811949901935: 1, 2.6638061390979: 1, 2.6635672843450
88: 1, 2.6634723977841706: 1, 2.6630947855820395: 1, 2.662863226010599
3: 1, 2.6625415675101447: 1, 2.661409286574512: 1, 2.660912981104781:
1, 2.6595947593007083: 1, 2.6582994458742837: 1, 2.658286029632805: 1,
2.6564360076427174: 1, 2.6550996860326004: 1, 2.652998338131532: 1, 2.6
51806313070323: 1, 2.6512068369908732: 1, 2.6509052584622186: 1, 2.6500
78003914424: 1, 2.646442520059987: 1, 2.6460800319648214: 1, 2.64553098
2974859: 1, 2.6440388329457614: 1, 2.6426135092000056: 1, 2.64025101790
93053: 1, 2.6400010586826843: 1, 2.6398139037043977: 1, 2.6388230489614
064: 1, 2.6372592494942086: 1, 2.6361570672520522: 1, 2.634531829736560
7: 1, 2.6330901738284638: 1, 2.6322845632078726: 1, 2.6322625034784055:
1, 2.631876573130291: 1, 2.631294579075247: 1, 2.629863312862658: 1, 2.
6292995637960765: 1, 2.6254190212195407: 1, 2.6253002853008938: 1, 2.62
49078566564443: 1, 2.6246281478888736: 1, 2.6240711702838015: 1, 2.6240
53120090231: 1, 2.623708763413581: 1, 2.6222736461269847: 1, 2.62088585
62273862: 1, 2.620140084324818: 1, 2.6178500690332376: 1, 2.61616086453
5446: 1, 2.6141726784799966: 1, 2.613332191705747: 1, 2.61315656658652
4: 1, 2.6118377337397765: 1, 2.6115632120876726: 1, 2.6108284008561085:
1, 2.6102842903652066: 1, 2.6081481994681237: 1, 2.606155606590762: 1,
2.6047015682299826: 1, 2.6043878488905494: 1, 2.6037981172090556: 1, 2.
602485957800094: 1, 2.6024532987847286: 1, 2.6019674529491446: 1, 2.601
3033506009897: 1, 2.5991406730275375: 1, 2.595440887508961: 1, 2.595006
826210115: 1, 2.593901445357548: 1, 2.592662530211639: 1, 2.59258980041
44172: 1, 2.5867845980031388: 1, 2.586057853155498: 1, 2.58540549920943
93: 1, 2.585274701869249: 1, 2.579237692692841: 1, 2.57921319480055: 1,
2.5781100859045205: 1, 2.5780645479836455: 1, 2.5779134851542995: 1, 2.
575792024462159: 1, 2.574957783278162: 1, 2.572915262479323: 1, 2.57248
0946554951: 1, 2.571646081334006: 1, 2.5708873458908617: 1, 2.570486938
1601756: 1, 2.567090874517746: 1, 2.566324312854273: 1, 2.5651784257026
597: 1, 2.5639378511163335: 1, 2.5637514244520587: 1, 2.56367423097961
9: 1, 2.5631343838440457: 1, 2.561875404012312: 1, 2.561053659476897:
1, 2.5601562678340497: 1, 2.559928860422576: 1, 2.5552269914247194: 1,
2.5551558898444258: 1, 2.551360322372214: 1, 2.551114545879498: 1, 2.54
92648307022123: 1, 2.5487846800337217: 1, 2.547124303908891: 1, 2.54673
3317371265: 1, 2.546158512753705: 1, 2.5450774289816054: 1, 2.542539221
289794: 1, 2.5424726786758116: 1, 2.5394239505200606: 1, 2.538869284149

```
119: 1, 2.533453890149805: 1, 2.532848951096974: 1, 2.532463932405743:
1, 2.529860704206608: 1, 2.5275713244284668: 1, 2.527208933751347: 1,
2.527155951137401: 1, 2.526924662025929: 1, 2.5265061029165508: 1, 2.52
6284148671684: 1, 2.5243943286811694: 1, 2.5243781825269758: 1, 2.52397
99109442216: 1, 2.5224227409964057: 1, 2.522345272186479: 1, 2.52141922
59664405: 1, 2.5194237929584897: 1, 2.518995071282385: 1, 2.51833485060
16405: 1, 2.5172519860003333: 1, 2.5171099542791677: 1, 2.5168789102007
367: 1, 2.5167334953632583: 1, 2.5150301594704136: 1, 2.51384739114931
8: 1, 2.513535085713991: 1, 2.5086474607382567: 1, 2.5075870369004196:
1, 2.507097455184179: 1, 2.5062467300128906: 1, 2.505946272561975: 1,
2.5051518597312246: 1, 2.504121146060899: 1, 2.5032657661210003: 1, 2.5
02882103107256: 1, 2.502747750274832: 1, 2.5016203243681248: 1, 2.50120
35320881587: 1, 2.4990210479024237: 1, 2.4987567638425507: 1, 2.4983242
641932724: 1, 2.4972054315858383: 1, 2.4959875843942774: 1, 2.495924941
2463924: 1, 2.4958356585815373: 1, 2.4953245894617235: 1, 2.49260569829
6614: 1, 2.492280805601741: 1, 2.490177599064179: 1, 2.489607061021226:
1, 2.4895183931498766: 1, 2.4887756423382768: 1, 2.4882915845849523: 1,
2.4859181445827376: 1, 2.4848665636971603: 1, 2.4830680433067522: 1, 2.
4825856181668713: 1, 2.48206330333226: 1, 2.4815955404912677: 1, 2.4814
85562774422: 1, 2.4812287573962495: 1, 2.478185703927439: 1, 2.47762149
83865312: 1, 2.477213890356314: 1, 2.4751552311413967: 1, 2.47431133668
35665: 1, 2.473262180505626: 1, 2.473245535936961: 1, 2.47039167182322
5: 1, 2.4668419976798273: 1, 2.465699985534845: 1, 2.465686117060719:
1, 2.464438299059194: 1, 2.4626218091540233: 1, 2.4623782752236845: 1,
2.4617700767658417: 1, 2.4613187453723735: 1, 2.4610623745631295: 1, 2.
460157272263911: 1, 2.4590390224838026: 1, 2.4586138574380816: 1, 2.457
4677818976967: 1, 2.456975957946147: 1, 2.456875423921732: 1, 2.4568558
170722237: 1, 2.455638919171389: 1, 2.453663945914494: 1, 2.45365747522
7146: 1, 2.453498330287703: 1, 2.451639741148952: 1, 2.449373381033205:
1, 2.4482384738372502: 1, 2.446644888579028: 1, 2.4449992395140425: 1,
2.44459939627383: 1, 2.444456012174329: 1, 2.4436607757805766: 1, 2.442
7815311188463: 1, 2.442251824318821: 1, 2.44135181143451: 1, 2.44040612
0467267: 1, 2.4400107628272703: 1, 2.439823707693871: 1, 2.439204157619
5247: 1, 2.434404277308943: 1, 2.433065066104704: 1, 2.433049543165731
5: 1, 2.4324790616880865: 1, 2.431833866323328: 1, 2.42736212290935: 1,
2.427310312511469: 1, 2.423724402222701: 1, 2.423713239558748: 1, 2.422
7026441690467: 1, 2.4208288505942286: 1, 2.4186962458456183: 1, 2.41776
27926732828: 1, 2.413654270551622: 1, 2.413583390967242: 1, 2.412559022
392894: 1, 2.4124989572239266: 1, 2.4124866805961402: 1, 2.411872164791
```

045: 1, 2.41143099911127: 1, 2.410153780705032: 1, 2.4100398681026487:
1, 2.4094236293558873: 1, 2.4090078263332995: 1, 2.4082378338659804: 1,
2.40718954886383: 1, 2.406543882439921: 1, 2.4047515252145124: 1, 2.404
2231353750076: 1, 2.400511296399712: 1, 2.3960668322636587: 1, 2.395912
325353413: 1, 2.39584137001514: 1, 2.395384168306605: 1, 2.394113996595
527: 1, 2.393603049006679: 1, 2.3935882897921834: 1, 2.392895301809641
5: 1, 2.392612019944937: 1, 2.3923009676106686: 1, 2.388894575152656:
1, 2.3870761060291574: 1, 2.384269372581944: 1, 2.3840257268402207: 1,
2.383735483276329: 1, 2.383003178436823: 1, 2.382732172383735: 1, 2.382
5781625960514: 1, 2.381435301762205: 1, 2.380597876197988: 1, 2.3780586
55591129: 1, 2.374845243207766: 1, 2.374698338412248: 1, 2.372672896017
4563: 1, 2.3717144977968143: 1, 2.3711125768908055: 1, 2.36576844554051
44: 1, 2.365359264806922: 1, 2.362034292639916: 1, 2.3616576378530447:
1, 2.361574340421505: 1, 2.360864442347632: 1, 2.360574392949285: 1, 2.
359856920857273: 1, 2.3594813663545784: 1, 2.359424420779723: 1, 2.3584
02587967568: 1, 2.3577951432624: 1, 2.3564793192049236: 1, 2.3563049459
75281: 1, 2.356008762948074: 1, 2.355567151878322: 1, 2.355564793878756
3: 1, 2.355275056380893: 1, 2.3546854620319677: 1, 2.3538250418897357:
1, 2.3525738555289215: 1, 2.3519970776134533: 1, 2.349379591766424: 1,
2.348755242172397: 1, 2.348750640413795: 1, 2.3486383012237617: 1, 2.34
7964593533419: 1, 2.3471114884485718: 1, 2.3448331205006565: 1, 2.34361
8273621114: 1, 2.341387657360935: 1, 2.340496864472979: 1, 2.3402182009
581614: 1, 2.340202031620269: 1, 2.3400085007826403: 1, 2.3393218385917
42: 1, 2.339033145832382: 1, 2.336633766757872: 1, 2.3362283844216543:
1, 2.336189855684387: 1, 2.335353194789781: 1, 2.334690253364588: 1, 2.
332570744016295: 1, 2.33170129428428: 1, 2.331095717120681: 1, 2.331008
2672556836: 1, 2.3308246715952254: 1, 2.329270103980288: 1, 2.328995726
8479404: 1, 2.3285659716597857: 1, 2.327318875411839: 1, 2.326338471101
127: 1, 2.326121590644081: 1, 2.3254149000424076: 1, 2.325190835775977:
1, 2.3251880796449136: 1, 2.3233869296318272: 1, 2.323291371173549: 1,
2.322933628567926: 1, 2.322725063420619: 1, 2.3199964868666516: 1, 2.31
95084855871277: 1, 2.318991757700857: 1, 2.318946240200664: 1, 2.318643
0134259712: 1, 2.318542270857865: 1, 2.312863312629431: 1, 2.3113377200
036456: 1, 2.3088309817353143: 1, 2.3067875719415407: 1, 2.305636251989
6267: 1, 2.305537837842172: 1, 2.3039803068864506: 1, 2.303871638201109
3: 1, 2.303671774585686: 1, 2.3030331222838565: 1, 2.3022618897182374:
1, 2.299813334538036: 1, 2.2991854413205055: 1, 2.29697357891345: 1, 2.
2966106405645292: 1, 2.2964226226042337: 1, 2.2956214379666533: 1, 2.29
5057055061019: 1, 2.2950370241738507: 1, 2.2925062686860085: 1, 2.29159

```
6255597987: 1, 2.290829587933128: 1, 2.2900498861016896: 1, 2.289192442
0861796: 1, 2.289170276203122: 1, 2.28887096493767: 1, 2.28875055951902
16: 1, 2.288566701478806: 1, 2.2862322103318657: 1, 2.284013639483122:
1, 2.2837492562832837: 1, 2.2821950427286333: 1, 2.2813485308915187: 1,
2.281100127064864: 1, 2.28067275492533: 1, 2.2804895657110396: 1, 2.279
4389514689453: 1, 2.2787359085449586: 1, 2.277376814587725: 1, 2.274643
893763451: 1, 2.2732355283782217: 1, 2.272652427488035: 1, 2.2723801044
452534: 1, 2.272347864482865: 1, 2.2716738985858265: 1, 2.2704709789711
544: 1, 2.2703398200806615: 1, 2.2700054366874554: 1, 2.26998777919013
9: 1, 2.269813288303895: 1, 2.2686578177476187: 1, 2.268630494747412:
1, 2.267864141853476: 1, 2.267700993099352: 1, 2.2657568910474626: 1,
2.2653893298181553: 1, 2.2652596598618397: 1, 2.2649778956876765: 1, 2.
2629670584410215: 1, 2.261097943370265: 1, 2.259885555623495: 1, 2.2597
295912535844: 1, 2.258135663429086: 1, 2.2576317666840255: 1, 2.2572961
67873444: 1, 2.253355449921295: 1, 2.252945736380738: 1, 2.252454098037
5716: 1, 2.2523220077732233: 1, 2.252038664041895: 1, 2.251996112211073
4: 1, 2.250022638833923: 1, 2.2493786862137406: 1, 2.2458974884538345:
1, 2.24463746687666: 1, 2.244341500752588: 1, 2.243703559092783: 1, 2.2
417204811563103: 1, 2.241579285934726: 1, 2.2414118992485395: 1, 2.2413
93521447835: 1, 2.2412828731378656: 1, 2.239091196494172: 1, 2.23774293
20013915: 1, 2.237098817927227: 1, 2.2367928693619885: 1, 2.23602256899
0236: 1, 2.2350619525074005: 1, 2.2336964078232264: 1, 2.23365254922150
67: 1, 2.2329120946375904: 1, 2.231360463760693: 1, 2.230743689325617:
 1, 2.230406234026545: 1, 2.228991646963142: 1, 2.2286193366910796: 1,
 2.2256460159501295: 1, 2.224041735909715: 1, 2.2222724876719773: 1, 2.
2217494445116697: 1, 2.2209305746691568: 1, 2.219989270426512: 1, 2.219
888138463574: 1, 2.217487131003127: 1, 2.217295816410521: 1, 2.21576426
81868874: 1, 2.2149778122835886: 1, 2.2127798081211774: 1, 2.2127590707
94122: 1, 2.211695392241619: 1, 2.211529077100505: 1, 2.20978273041011
1: 1, 2.208768966767742: 1, 2.207173376957516: 1, 2.2071311457352345:
 1, 2.206609390450893: 1, 2.20497951464686: 1, 2.204323937526086: 1, 2.
2033747891603315: 1, 2.202236444149387: 1, 2.2016166102906274: 1, 2.201
3877383267357: 1, 2.2009569436526353: 1, 2.199373684453578: 1, 2.197007
542432156: 1, 2.196530739727263: 1, 2.1962845974625873: 1, 2.1938286077
84164: 1, 2.193459264560404: 1, 2.1928069633918814: 1, 2.19120117212628
83: 1, 2.1889246162308273: 1, 2.1886392899101836: 1, 2.187092494463265:
 1, 2.184064000380768: 1, 2.1838732422674894: 1, 2.183167852549451: 1,
 2.182307511930863 3: 1, 2.1806574730948847: 1, 2.1805994299936264: 1,
 2.179906249426622: 1, 2.178128187461714: 1, 2.177582729539363: 1, 2.17
```

```
71816318242925: 1, 2.1760578545777824: 1, 2.1753299657886975: 1, 2.1752
683424762584: 1, 2.1735584635384084: 1, 2.17347155477894: 1, 2.17256288
2983556: 1, 2.172302301229624: 1, 2.171754010990703: 1, 2.1702866865559
14: 1, 2.1685279631375045: 1, 2.1679880296329856: 1, 2.167271868992856
3: 1, 2.1671961004557625: 1, 2.166604689459518: 1, 2.165276166866197:
 1, 2.1640292546985305: 1, 2.163803565594426: 1, 2.1625539007890207: 1,
 2.160381468861398: 1, 2.1594880707892186: 1, 2.159179896028837: 1, 2.1
591721626912403: 1, 2.159087439586327: 1, 2.1569134938732706: 1, 2.1566
5009629616: 1, 2.156572146847752: 1, 2.15406386997702: 1, 2.15223737217
42794: 1, 2.1514489622897677: 1, 2.1512292988062742: 1, 2.1509005560986
71: 1, 2.150544485348952: 1, 2.149375075835889: 1, 2.148666557663844:
 1, 2.1485907656093235: 1, 2.1478066102741735: 1, 2.147597195475006: 1,
 2.1471710677720615: 1, 2.146429559566351: 1, 2.145272042649448: 1, 2.1
45192778329186: 1, 2.1428728175052694: 1, 2.1417312761150793: 1, 2.1415
08246300152: 1, 2.1406925328036563: 1, 2.1405189384264096: 1, 2.1399579
82944797: 1, 2.139787786754776: 1, 2.1394574557431256: 1, 2.13927592086
0592: 1, 2.138378168381775: 1, 2.1376264827076716: 1, 2.137463910683884
6: 1, 2.135141955344589: 1, 2.1339673481431207: 1, 2.133891129578619:
 1, 2.1335587939414267: 1, 2.133220907062877: 1, 2.133116568065494: 1,
 2.132310520010774: 1, 2.131017466273409: 1, 2.1301555486463064: 1, 2.1
300912025454966: 1, 2.1287136549944257: 1, 2.127213390610795: 1, 2.1266
906484250567: 1, 2.126157153742483: 1, 2.125711068505255: 1, 2.12557975
12443046: 1, 2.1250309981341364: 1, 2.1240345075908063: 1, 2.1238054900
50766: 1, 2.123708517850891: 1, 2.1228297178837017: 1, 2.12272904116199
6: 1, 2.1222033026990426: 1, 2.120951942620094: 1, 2.118322673322685:
 1, 2.1159629610285435: 1, 2.1151239676201703: 1, 2.114751261573185: 1,
 2.11428933236874: 1, 2.112974212763794: 1, 2.11245114615376: 1, 2.1118
52378233298: 1, 2.108799017233993: 1, 2.1085112603441494: 1, 2.10627989
4203235: 1, 2.106079926397881: 1, 2.104332591401225: 1, 2.1023315580423
203: 1, 2.099962473785848: 1, 2.0996216350046892: 1, 2.099143212056851:
 1, 2.099016585819516: 1, 2.0989950169003144: 1, 2.0984021239168227: 1,
 2.097538004029105: 1, 2.0970787328303135: 1, 2.0955647792421193: 1, 2.
0954666693943538: 1, 2.091122285070556: 1, 2.0902041852317943: 1, 2.088
822424743567: 1, 2.0884290857200924: 1, 2.0878779145483914: 1, 2.087063
4213209596: 1, 2.0862908223521406: 1, 2.086185736464353: 1, 2.085144158
5136667: 1, 2.08479934843634: 1, 2.0841351527674363: 1, 2.0820169863623
7: 1, 2.0815339930141263: 1, 2.0812097262481317: 1, 2.080133607008447:
 1, 2.079795360640527: 1, 2.0793670330780403: 1, 2.07877012781715: 1,
 2.0785514354377614: 1, 2.0778166708002224: 1, 2.077145193629481: 1, 2.
```

074964446126972: 1, 2.07266708113694: 1, 2.070602117356358: 1, 2.069588
494495137: 1, 2.0686357701524454: 1, 2.0679274859356522: 1, 2.067638435
439621: 1, 2.0674029591687817: 1, 2.064661462572343: 1, 2.0638266460210
466: 1, 2.0628271985750777: 1, 2.0607057560155577: 1, 2.060101931701957
8: 1, 2.059447533402248: 1, 2.057763491340234: 1, 2.0576645276472134:
1, 2.056618556445695: 1, 2.0565680621821474: 1, 2.0564600126952306: 1,
2.0555867452163565: 1, 2.054814362849546: 1, 2.0521481021791357: 1, 2.
0503465154388922: 1, 2.0500739479701053: 1, 2.048703821700816: 1, 2.048
1785635455054: 1, 2.0473307770136593: 1, 2.0446107128641176: 1, 2.04289
56261143223: 1, 2.042363302354149: 1, 2.041261828885904: 1, 2.039066965
652227: 1, 2.03853860626196: 1, 2.0378097316355537: 1, 2.03510544145021
65: 1, 2.0325495340389708: 1, 2.0317751165308393: 1, 2.030050631011932:
1, 2.0300333916563367: 1, 2.0299875550818833: 1, 2.0299075757039295:
1, 2.0296676705675347: 1, 2.028457516981576: 1, 2.0274535393068525: 1,
2.0265597325017564: 1, 2.0263994734357933: 1, 2.0262045099745514: 1,
2.02139429288625: 1, 2.020322473355336: 1, 2.0200680173342147: 1, 2.01
94700157365606: 1, 2.018973190086113: 1, 2.018667524197541: 1, 2.018135
998166181: 1, 2.0174618364881476: 1, 2.0172002452991453: 1, 2.016910857
5508847: 1, 2.0166372220473376: 1, 2.0166220443648175: 1, 2.01651804553
22944: 1, 2.014838806597713: 1, 2.0131214538892848: 1, 2.01292384725041
94: 1, 2.0128295800871334: 1, 2.0126946407236477: 1, 2.011771979667533
7: 1, 2.011228377487931: 1, 2.011088703516137: 1, 2.010873286539559: 1,
2.010244758823691: 1, 2.0095936490619972: 1, 2.007714374439802: 1, 2.0
063065636052526: 1, 2.0052543360286323: 1, 2.004813822920605: 1, 2.0026
274962840303: 1, 1.9991604588620215: 1, 1.997605757920845: 1, 1.9970361
52129877: 1, 1.9970245028575517: 1, 1.9962801187128751: 1, 1.9925646678
98471: 1, 1.990378961012157: 1, 1.9878810356928203: 1, 1.98718281559262
48: 1, 1.9865418142443603: 1, 1.985555395079903: 1, 1.9845581601554445:
1, 1.9844795582144246: 1, 1.9842665381971425: 1, 1.9816833344580744:
1, 1.9814784179142022: 1, 1.9802574781664812: 1, 1.9800349675601152:
1, 1.9776628451525446: 1, 1.9775477437926372: 1, 1.9765920350162896:
1, 1.9753201213966534: 1, 1.9750500678828837: 1, 1.9748490408674142:
1, 1.9748402417121937: 1, 1.974817340766531: 1, 1.9742133287643908: 1,
1.9740933622055308: 1, 1.972355057028727: 1, 1.9708141116903029: 1, 1.
9703529631026888: 1, 1.9689997786075246: 1, 1.9687191396644916: 1, 1.96
71878279002148: 1, 1.9662716478985547: 1, 1.9639789729463772: 1, 1.9627
18248485694: 1, 1.9612466168699836: 1, 1.9606814351840667: 1, 1.9597258
152400194: 1, 1.9583146938191178: 1, 1.95801710578467: 1, 1.95775698423
78855: 1, 1.9574789605582272: 1, 1.9564026495118796: 1, 1.9561061206432

546: 1, 1.9545112210028215: 1, 1.9539585933941532: 1, 1.953403075969234
7: 1, 1.9517421517674958: 1, 1.9509075799332314: 1, 1.9495019019447246:
 1, 1.9489024656942053: 1, 1.9487880287477797: 1, 1.947889244825322: 1,
 1.9475380540317444: 1, 1.9471661366251185: 1, 1.9465338162351078: 1,
 1.9462711378919697: 1, 1.945813725049572: 1, 1.9436811611546136: 1, 1.
9429282620760857: 1, 1.9426736447958475: 1, 1.9410996593414287: 1, 1.94
07703967217613: 1, 1.9397861724561531: 1, 1.9392183538889944: 1, 1.9392
170159264408: 1, 1.9385591831504942: 1, 1.9371732596870257: 1, 1.935072
2408507477: 1, 1.93423189540212: 1, 1.9337479555471235: 1, 1.9333231339
327734: 1, 1.9320917078773603: 1, 1.930912299029117: 1, 1.9304281679879
227: 1, 1.9303041968825994: 1, 1.9302009314756448: 1, 1.929943766898203
1: 1, 1.9292467848280597: 1, 1.9292407021634896: 1, 1.9285378177559798:
 1, 1.927621183131249: 1, 1.9273803984812004: 1, 1.9263871482024317: 1,
 1.926361453873847: 1, 1.9228510825039424: 1, 1.922805335422146: 1, 1.9
221522410457828: 1, 1.9206241825070307: 1, 1.9204772188755492: 1, 1.919
2410780610178: 1, 1.918409971631708: 1, 1.917819739910916: 1, 1.9169263
299842165: 1, 1.9168624228022897: 1, 1.9165204104820637: 1, 1.916153362
8629264: 1, 1.9154949686631426: 1, 1.9144621831065085: 1, 1.91295948084
95083: 1, 1.9123710993562213: 1, 1.9119924556413692: 1, 1.9118276547235
658: 1, 1.9113203849989342: 1, 1.9107365372940663: 1, 1.908659868323128
4: 1, 1.908390687753813: 1, 1.9082517291204357: 1, 1.9078624359772134:
 1, 1.9076011002216475: 1, 1.906270874639457: 1, 1.9058098689546084: 1,
 1.9055471280158764: 1, 1.9053554081296873: 1, 1.9041482396933849: 1,
 1.902983172985135: 1, 1.9029340520177096: 1, 1.9020791451790338: 1, 1.
9005769248017783: 1, 1.9001818194318847: 1, 1.900044011559182: 1, 1.899
1918706557898: 1, 1.8985660473801647: 1, 1.8979610109264062: 1, 1.89700
88778888337: 1, 1.8969635274352006: 1, 1.8969447371343826: 1, 1.8962395
113803572: 1, 1.896163292147984: 1, 1.8956723594457863: 1, 1.8946231610
07614: 1, 1.8934946729288795: 1, 1.8927298549764426: 1, 1.8922418507218
148: 1, 1.8903587844511116: 1, 1.890183609913184: 1, 1.88881724349179:
 1, 1.8880140560006786: 1, 1.8878178800540097: 1, 1.8878020380450535:
 1, 1.8853605724614038: 1, 1.8839729407585442: 1, 1.8837676795541776:
 1, 1.8829321735880695: 1, 1.8819892407326317: 1, 1.880683214536849: 1,
 1.8799835900537947: 1, 1.8798613852651203: 1, 1.8782223103914033: 1,
 1.8777371434400787: 1, 1.877702277863239: 1, 1.8767731433907446: 1, 1.
876765682511866: 1, 1.8764446463344862: 1, 1.8761876448670456: 1, 1.873
0537077258935: 1, 1.872685001560551: 1, 1.8724824082829599: 1, 1.871637
4125471011: 1, 1.8716196944770775: 1, 1.8710970986622006: 1, 1.87099485
97132928: 1, 1.8708188550174965: 1, 1.8688658785335872: 1, 1.8651344559

```
529868: 1, 1.8649257828963854: 1, 1.8633786233358223: 1, 1.862310463786
9654: 1, 1.86229654362527: 1, 1.8619356879027773: 1, 1.861905342775938
7: 1, 1.8611582907465989: 1, 1.860061204179251: 1, 1.8593225829819098:
 1, 1.8590649332873612: 1, 1.8588534963706143: 1, 1.8586366566438712:
 1, 1.858571051157234: 1, 1.8585133900263708: 1, 1.8566916282376544: 1,
 1.8564794933528577: 1, 1.856312184998741: 1, 1.854638157322513: 1, 1.8
537711942643402: 1, 1.8529783183247392: 1, 1.8529112208696898: 1, 1.852
3348880368298: 1, 1.8519924324709607: 1, 1.8514936395526005: 1, 1.85048
89147826633: 1, 1.8486293119230879: 1, 1.8473988338536955: 1, 1.8472172
305388084: 1, 1.8465478201585261: 1, 1.8435488437220358: 1, 1.842158007
8745246: 1, 1.8415066871988472: 1, 1.8412389869846935: 1, 1.83956718683
3827: 1, 1.839204595722639: 1, 1.8373246228174418: 1, 1.837272266758778
9: 1, 1.836986559235889: 1, 1.8367710106310045: 1, 1.835883597740064:
 1, 1.8347717924719908: 1, 1.8344855951737638: 1, 1.8343637520085285:
 1, 1.8313553273200194: 1, 1.8310884381868597: 1, 1.8293094537461954:
 1, 1.8263881744996233: 1, 1.826002316266839: 1, 1.8259938009076513: 1,
 1.8250477156503961: 1, 1.8241433448357016: 1, 1.8240170001518135: 1,
 1.823287771304906: 1, 1.8212360004292223: 1, 1.821126387031103: 1, 1.8
203134325474495: 1, 1.8189774289095173: 1, 1.8189123755032583: 1, 1.818
8128975457192: 1, 1.8179666124438036: 1, 1.8172049497910985: 1, 1.81686
17636996496: 1, 1.8165564726712373: 1, 1.816487296606605: 1, 1.81545240
16294407: 1, 1.8152437293720665: 1, 1.8140445947763122: 1, 1.8137502158
059946: 1, 1.8129779062964553: 1, 1.8113119444538643: 1, 1.809484648310
4596: 1, 1.8092945309756125: 1, 1.8090529978403447: 1, 1.80824288563760
14: 1, 1.8069106476002599: 1, 1.8060264669243729: 1, 1.805119416854583
7: 1, 1.8049757629174221: 1, 1.8049066399327707: 1, 1.8046077295088776:
 1, 1.8037603106902567: 1, 1.8029356997561388: 1, 1.8006849427915177:
 1, 1.7994276885673406: 1, 1.7985195545925365: 1, 1.7985145686435227:
 1, 1.7972083609065332: 1, 1.7953009318249242: 1, 1.7946381884208997:
 1, 1.79407617643484: 1, 1.7939071194180773: 1, 1.7935100284425238: 1,
 1.7933184741445034: 1, 1.7932110188487043: 1, 1.792228572003359: 1, 1.
7919521838941903: 1, 1.7902607064606542: 1, 1.7887200836132753: 1, 1.78
84969226025902: 1, 1.786383712113731: 1, 1.7846640200474566: 1, 1.78381
78973398193: 1, 1.7822414639630924: 1, 1.7782519108382955: 1, 1.7774207
57157794: 1, 1.777380096272138: 1, 1.776167300531264: 1, 1.774716230836
5154: 1, 1.774521925443256: 1, 1.7745054237562552: 1, 1.773816365132103
5: 1, 1.7737873030349973: 1, 1.7736683960568635: 1, 1.7735314528821253:
 1, 1.7732986155180364: 1, 1.7709757345603578: 1, 1.770196323506676: 1,
 1.7685236844920598: 1, 1.7685219527659364: 1, 1.768481838753802: 1, 1.
```

```
7683338389573036: 1, 1.7676882422929505: 1, 1.766185793369468: 1, 1.765
6197795896633: 1, 1.7652746694983739: 1, 1.7643580483527401: 1, 1.76398
37956235778: 1, 1.762465162416018: 1, 1.7611049925843705: 1, 1.76025894
65099518: 1, 1.7595395672158236: 1, 1.758783813610583: 1, 1.75619142509
7496: 1, 1.7557823513698034: 1, 1.7530575986304275: 1, 1.75188889874754
32: 1, 1.7514358851010112: 1, 1.7507425924053697: 1, 1.750710215239634
3: 1, 1.7502824235616619: 1, 1.748593917013557: 1, 1.7481968145011386:
 1, 1.7474126301178015: 1, 1.747128810871447: 1, 1.7457183556012734: 1,
 1.7453425611072: 1, 1.7446347782936542: 1, 1.7436483773898077: 1, 1.74
3624619493769: 1, 1.7423709442884643: 1, 1.742298696612885: 1, 1.741462
26445652: 1, 1.7411378130591448: 1, 1.7405957393046316: 1, 1.7401976041
531195: 1, 1.736712220847175: 1, 1.7366016218031326: 1, 1.7362108415946
964: 1, 1.7359661383009461: 1, 1.7344880695129192: 1, 1.733979759852166
2: 1, 1.7314241171883844: 1, 1.7301929480206797: 1, 1.7301330795426446:
 1, 1.7300546883843861: 1, 1.7292709288795567: 1, 1.7286684360611546:
 1, 1.7280487653340968: 1, 1.727732209585115: 1, 1.727298498476176: 1,
 1.7261983060711148: 1, 1.7258690961526333: 1, 1.7254681703196564: 1,
 1.7241405119672923: 1, 1.723310290474626: 1, 1.7216849881999077: 1, 1.
7200146332272848: 1, 1.7180868430738248: 1, 1.7179433800103325: 1, 1.71
7114167886648: 1, 1.7169182923070332: 1, 1.7166524228935858: 1, 1.71567
09279572244: 1, 1.7151082755522162: 1, 1.714012122601102: 1, 1.71383257
60430444: 1, 1.712303459892157: 1, 1.712189692202044: 1, 1.711225690458
398: 1, 1.7109549456415105: 1, 1.7107226430676614: 1, 1.710569399799871
9: 1, 1.7103447128574583: 1, 1.7093440316658304: 1, 1.7088793027370688:
 1, 1.7086604485575114: 1, 1.7074052145102632: 1, 1.706920849402048: 1,
 1.706541032302018: 1, 1.7060757191138503: 1, 1.7057953631170641: 1, 1.
7018117709792677: 1, 1.7001982657127728: 1, 1.699823821330595: 1, 1.699
7789917479542: 1, 1.6995211349470332: 1, 1.6994830306964905: 1, 1.69502
82672900934: 1, 1.6924244123339784: 1, 1.6923354586895996: 1, 1.6917068
322197009: 1, 1.690731262538594: 1, 1.6904777208220985: 1, 1.6901343037
291752: 1, 1.687993276341829: 1, 1.685827827044244: 1, 1.68553879817100
84: 1, 1.6844073997768616: 1, 1.6840058773264597: 1, 1.681226309920485
6: 1, 1.680657034498865: 1, 1.679881568032414: 1, 1.6797134886822205:
 1, 1.6782060784719797: 1, 1.677946065044504: 1, 1.6759261205292995: 1,
 1.6758850349040901: 1, 1.6756347648582417: 1, 1.675235222255191: 1, 1.
6748861016652268: 1, 1.6748642809891026: 1, 1.6739606198672163: 1, 1.67
3201740205757: 1, 1.6726655219534192: 1, 1.6718806315078718: 1, 1.67155
52821442659: 1, 1.6712368700259486: 1, 1.67085035483593: 1, 1.670759349
6377129: 1, 1.670594358538565: 1, 1.6701572531049915: 1, 1.669924515733
```

```
2407: 1, 1.6691224999104397: 1, 1.6681356970379677: 1, 1.66766687506411
44: 1, 1.6668224088834838: 1, 1.6657792753571385: 1, 1.663656278857512:
 1, 1.6631791407882441: 1, 1.663138888742503: 1, 1.6624095210336534: 1,
 1.661517693038398: 1, 1.6612210268086718: 1, 1.6602857080209241: 1, 1.
659977915719733: 1, 1.658240631170334: 1, 1.6571906166849824: 1, 1.6569
994815704425: 1, 1.6566495430810941: 1, 1.6562253996752245: 1, 1.656048
325005194: 1, 1.6559620858603075: 1, 1.6558454428162748: 1, 1.655394365
025566: 1, 1.6547437725696066: 1, 1.6543916999371924: 1, 1.654118674934
8806: 1, 1.6540051955113249: 1, 1.6535159463267755: 1, 1.65347419109657
22: 1, 1.6530941652953235: 1, 1.6526502862318324: 1, 1.652236198826854
4: 1, 1.650738894148181: 1, 1.6492836875210812: 1, 1.6468189613095388:
 1, 1.6464749621998358: 1, 1.646205271927496: 1, 1.6461731229338117: 1,
 1.6454160419259332: 1, 1.6449279235951448: 1, 1.6430168047181042: 1,
 1.6427686689607874: 1, 1.6420536946975084: 1, 1.640615577759619: 1, 1.
639042781170578: 1, 1.6384089436686355: 1, 1.638321304542243: 1, 1.6378
747262017348: 1, 1.6363350476288412: 1, 1.6361827495480965: 1, 1.635628
3531041143: 1, 1.6336275099131878: 1, 1.6325135764460967: 1, 1.63067242
83622367: 1, 1.629900903853641: 1, 1.6296083682381035: 1, 1.62941885259
5012: 1, 1.6280905643981773: 1, 1.627658906689652: 1, 1.627281667246014
6: 1, 1.6266789419745176: 1, 1.6265720485962423: 1, 1.6261419169051832:
 1, 1.622980120512152: 1, 1.6228064694281452: 1, 1.6227886899189208: 1,
 1.6202530917868778: 1, 1.6202331996198478: 1, 1.6167992378714844: 1,
 1.6158309031455822: 1, 1.6153172826650852: 1, 1.6137596359586772: 1,
 1.6132281142989637: 1, 1.612754721381369: 1, 1.6124860249957882: 1, 1.
6124593705339425: 1, 1.610648665833446: 1, 1.6071237717932303: 1, 1.604
8480271379864: 1, 1.6029221360062667: 1, 1.601819688657369: 1, 1.599787
4611121972: 1, 1.598703539365763: 1, 1.5975738833283248: 1, 1.597315213
4602715: 1, 1.5966575350752126: 1, 1.5965060866042577: 1, 1.59518063266
72917: 1, 1.595016437587314: 1, 1.594569446564823: 1, 1.594525458820770
5: 1, 1.593192663907537: 1, 1.5931271757576995: 1, 1.592954961681923:
 1, 1.5923182297851852: 1, 1.5905975722394752: 1, 1.5885813616694358:
 1, 1.5880251871324167: 1, 1.5874773210173774: 1, 1.5873807639960134:
 1, 1.5870715428635882: 1, 1.587010870964295: 1, 1.5868976422484615: 1,
 1.5860889291379767: 1, 1.585140999018618: 1, 1.5825583415596882: 1, 1.
5801453170579292: 1, 1.5799250699421572: 1, 1.579685607016564: 1, 1.578
4129505782016: 1, 1.577785693135781: 1, 1.5770922256621638: 1, 1.574884
0925032412: 1, 1.5747045067351206: 1, 1.5727946028175679: 1, 1.57071758
69732268: 1, 1.5695320733455032: 1, 1.5689964671962904: 1, 1.5665274492
54296: 1, 1.5657379511724756: 1, 1.565691110144701: 1, 1.56322950930011
```

```
36: 1, 1.562846587354459: 1, 1.5621149273049382: 1, 1.561524823926361:
 1, 1.5608155481143875: 1, 1.5589793780228853: 1, 1.5579019914290464:
 1, 1.557680794209301: 1, 1.556608670674591: 1, 1.5554901500460765: 1,
 1.555389488843537: 1, 1.5535481413674146: 1, 1.5533453373277117: 1, 1.
5520212979809958: 1, 1.55128510489544: 1, 1.550962452508286: 1, 1.55066
50333609564: 1, 1.5503398814090124: 1, 1.55016763872845: 1, 1.548773722
6823934: 1, 1.54794845734254: 1, 1.547024287937977: 1, 1.54686619698379
62: 1, 1.5453343923085074: 1, 1.5426165328692552: 1, 1.542029882224535
7: 1, 1.5418528740693591: 1, 1.5411767625875963: 1, 1.5410616278897034:
 1, 1.5402833925684813: 1, 1.539495606955441: 1, 1.5394721490484145: 1,
 1.5392565681758013: 1, 1.535773904035614: 1, 1.5356768642241119: 1, 1.
535394407213375: 1, 1.5353300678076: 1, 1.5340722675012741: 1, 1.533455
0536579516: 1, 1.532585450351761: 1, 1.53235502381567: 1, 1.53200232435
01217: 1, 1.527888554984374: 1, 1.5277535281152732: 1, 1.52734089060446
54: 1, 1.5266751475674274: 1, 1.526542925650853: 1, 1.525998017494499:
 1, 1.5259057915407261: 1, 1.5240023016769917: 1, 1.5235064820485271:
 1, 1.5233334080639898: 1, 1.5208388596400275: 1, 1.5194141940264074:
 1, 1.5190710202896376: 1, 1.517424312116574: 1, 1.5174243052102245: 1,
 1.517198100794939: 1, 1.5160908635124772: 1, 1.5159463640293194: 1, 1.
5159353140161322: 1, 1.5157624294462708: 1, 1.5151715990803398: 1, 1.51
43097403036696: 1, 1.513322903450534: 1, 1.5130044259776503: 1, 1.51245
62158994665: 1, 1.511572776704767: 1, 1.5107763716433398: 1, 1.50929544
50610927: 1, 1.5085826620159302: 1, 1.5081816724001942: 1, 1.5073716514
75925: 1, 1.5065750966136302: 1, 1.5036383641885511: 1, 1.5022936269338
23: 1, 1.501802401167753: 1, 1.5012391565554235: 1, 1.5004854790367812:
 1, 1.499465863387302: 1, 1.4988842991739744: 1, 1.4987960395874735: 1,
 1.4979016643533292: 1, 1.497848020475224: 1, 1.4967341545445578: 1, 1.
4963042453909656: 1, 1.4945529264391642: 1, 1.4929049823410931: 1, 1.49
26781761923413: 1, 1.492226940592879: 1, 1.4917973728088916: 1, 1.49120
13483618158: 1, 1.4905038334511744: 1, 1.490370174256659: 1, 1.48985798
81115745: 1, 1.4891735396398211: 1, 1.4884594193025613: 1, 1.4884477510
619156: 1, 1.4881175715238342: 1, 1.4859391388603085: 1, 1.485693286550
2313: 1, 1.4852823802162456: 1, 1.4851314245378346: 1, 1.48297657445662
74: 1, 1.4824366462626988: 1, 1.479316213957162: 1, 1.4789931685938167:
 1, 1.4789248068243892: 1, 1.4770349880039906: 1, 1.471770997027705: 1,
 1.4701968068293063: 1, 1.4690014094721462: 1, 1.4677696583685869: 1,
 1.4654866566576823: 1, 1.4631858845300385: 1, 1.4629257405608624: 1,
 1.4604833893863993: 1, 1.459834363144995: 1, 1.4594564899057862: 1, 1.
4569312997871413: 1, 1.4554048994585829: 1, 1.4552819680740334: 1, 1.45
```

```
26067646124892: 1, 1.4520142414293165: 1, 1.451581484661125: 1, 1.44979
9482618834: 1, 1.4488128634431927: 1, 1.4470805263437867: 1, 1.44682886
66519205: 1, 1.4455866936234494: 1, 1.444322732904319: 1, 1.44377841298
7983: 1, 1.4387945544780112: 1, 1.438723427254645: 1, 1.438613695290225
5: 1, 1.438121448665368: 1, 1.4370856094635625: 1, 1.4365139310543604:
 1, 1.4352847740929662: 1, 1.432445299530892: 1, 1.4305146519349348: 1,
 1.4299887542009382: 1, 1.4287838802669908: 1, 1.4269589859710818: 1,
 1.4266243923579165: 1, 1.423331110752481: 1, 1.4227415127201806: 1, 1.
4210611781588758: 1, 1.4180663345872253: 1, 1.4173989375085836: 1, 1.41
48014378845057: 1, 1.4104978172650653: 1, 1.4096165678152448: 1, 1.4090
321085279127: 1, 1.4054632290052906: 1, 1.402685610154554: 1, 1.4016613
216561613: 1, 1.4010638351231872: 1, 1.3995651627663555: 1, 1.397928007
2834046: 1, 1.3919245713875605: 1, 1.3916043649548397: 1, 1.39069801377
81342: 1, 1.3894082630158593: 1, 1.3874334211030293: 1, 1.3857080580866
625: 1, 1.380446353701326: 1, 1.3781339804836308: 1, 1.377639694245126
8: 1, 1.3761932379596837: 1, 1.3732668289522911: 1, 1.372495781428266:
 1, 1.3700336338506338: 1, 1.3695956382570733: 1, 1.3673896851570502:
 1, 1.3670818519893113: 1, 1.3658125113777342: 1, 1.3650105506798316:
 1, 1.364792322075902: 1, 1.3643369023469147: 1, 1.3625888440704343: 1,
 1.362323756972549: 1, 1.3610796097618885: 1, 1.3601945374988789: 1, 1.
3598360344382565: 1, 1.3598190639945817: 1, 1.359149493737424: 1, 1.358
9975438688033: 1, 1.3584918703547066: 1, 1.357977555225506: 1, 1.354791
0932430562: 1, 1.3542022340350397: 1, 1.3536599366873117: 1, 1.35125075
2666557: 1, 1.3475753023053691: 1, 1.3455200848237856: 1, 1.34282674779
65583: 1, 1.3398820602796437: 1, 1.3394796404933214: 1, 1.3392732643145
602: 1, 1.337022332074543: 1, 1.331950622053425: 1, 1.3310060132998136:
 1, 1.3306464008834176: 1, 1.3306181085522621: 1, 1.3296196612567757:
 1, 1.3294366910072273: 1, 1.3294284421579468: 1, 1.3274714916332244:
 1, 1.3181947242649592: 1, 1.3117177978220018: 1, 1.3105084092165942:
 1, 1.3103309007020711: 1, 1.3072782854914369: 1, 1.3057585517863428:
 1, 1.3031598702335712: 1, 1.301686443345359: 1, 1.3006300221152576: 1,
 1.2986846374201324: 1, 1.2976178207381122: 1, 1.2953093594826526: 1,
 1.293233223218545: 1, 1.290044319406891: 1, 1.2896222651204339: 1, 1.2
887707798806598: 1, 1.288576247636934: 1, 1.286694642656775: 1, 1.28427
7726771748: 1, 1.2823921023562945: 1, 1.2814337125138253: 1, 1.27972174
00576315: 1, 1.279409175684258: 1, 1.2777721174465133: 1, 1.27461212043
42447: 1, 1.2742578234181738: 1, 1.273543225247337: 1, 1.27281901345008
34: 1, 1.2598950018973658: 1, 1.2592882581784788: 1, 1.245826069731271
5: 1, 1.233135938223664: 1, 1.229724349844666: 1, 1.222989271152473: 1,
```

```
 1.2224751114375099: 1, 1.2223857863930176: 1, 1.220550290880363: 1, 1.
2201585398933807: 1, 1.2173712617997166: 1, 1.2097978415396202: 1, 1.20
34825049482116: 1, 1.2023624249154925: 1, 1.2020686906611997: 1, 1.1979
427622415284: 1, 1.1929242213740916: 1, 1.1897997652337449: 1, 1.186775
241763479: 1, 1.1826930405573737: 1, 1.1773711944951486: 1, 1.174504049
2469585: 1, 1.1718503009827892: 1, 1.1656925183637006: 1, 1.16511931389
03755: 1, 1.1460294206908124: 1, 1.1435600796709309: 1, 1.1320240704746
232: 1, 1.1309862503540078: 1, 1.1143930150579393: 1, 1.10252334743446
9: 1, 1.090905005920583: 1, 1.0902425284093626: 1, 1.0811804443438855:
 1, 1.0656339171748843: 1, 0.9487186441488827: 1, 0.9355271961246372:
 1, 0.9276679229306359: 1, 0.9199929546128722: 1, 0.8901477583777209:
 1, 0.8215534317617715: 1})
```

TEXT BOW VECTORIZER

In [117]:
```
# one-hot encoding of TEXT feature BOW.
bow_text_vectorizer = CountVectorizer(min_df=3,ngram_range=(1,2)) #Uni
 and Bi grams
bow_train_text_feature_onehotCoding = bow_text_vectorizer.fit_transform
(train_df['TEXT'])
bow_test_text_feature_onehotCoding = bow_text_vectorizer.transform(test
_df['TEXT'])
bow_cv_text_feature_onehotCoding = bow_text_vectorizer.transform(cv_df[
'TEXT'])
```

In [118]:
```
print("train_gene_feature_responseCoding is converted feature using res
pone coding method. The shape of gene feature:", bow_train_text_feature
_onehotCoding.shape)
```

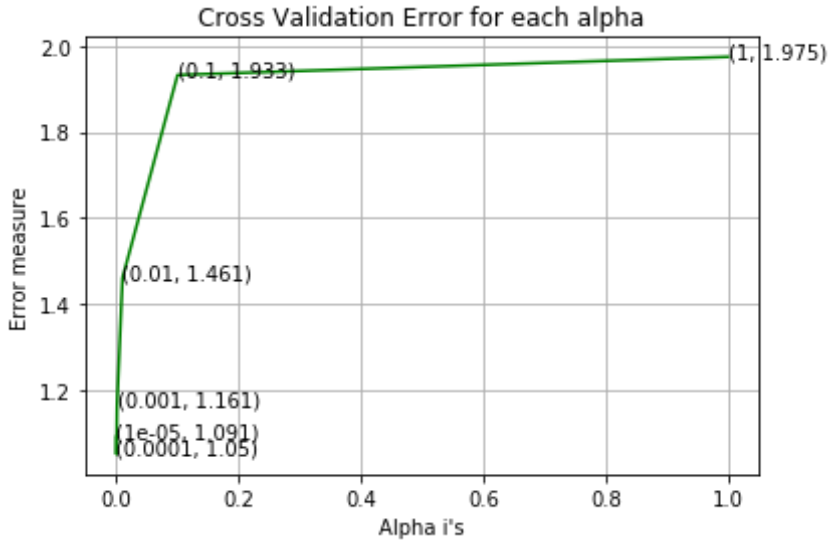
```
train_gene_feature_responseCoding is converted feature using respone co
ding method. The shape of gene feature: (2124, 765995)
```

In [54]:
```
# Train a Logistic regression+Calibration model using text features whi
cha re on-hot encoded
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
```

```
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

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


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

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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.0909167905739452
For values of alpha =  0.0001 The log loss is: 1.0500808593318378
For values of alpha =  0.001 The log loss is: 1.161497042544221
For values of alpha =  0.01 The log loss is: 1.4607221517369813
For values of alpha =  0.1 The log loss is: 1.9328487853822036
For values of alpha =  1 The log loss is: 1.975081791260054
```




```
For values of best alpha =  0.0001 The train log loss is: 0.70035436294
45774
For values of best alpha =  0.0001 The cross validation log loss is: 1.
0500808593318378
For values of best alpha =  0.0001 The test log loss is: 1.069821379364
824
```

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

Ans. Yes, it seems like!

In [55]:
```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(max_features=4000)
    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_coun
ts))
    len1 = len(set(df_text_features))
```

```
    len2 = len(set(train_text_features) & set(df_text_features))
    return len1,len2
```

In [56]:
```
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 appe
ared in train data")
```

```
92.275 % of word of test data appeared in train data
91.7 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [57]:
```
#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 probabilit
ies belongs to each class
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y
- test_y))/test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

In [58]:
```
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 [59]:

```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text
 or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(max_features=4000)

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

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

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

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

In [121]:

```
# 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_bow_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(ngram_range=(1,2),min_df=3)

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

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

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

```
                print(i, "variation feature [{}] present in test data p
oint [{}]".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
 [{}]".format(word,yes_no))

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

Stacking the three types of features

In [61]:
```
# 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,t
rain_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,tes
t_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_vari
ation_feature_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_fea
ture_onehotCoding)).tocsr()
test_y = np.array(list(test_df['Class']))

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


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseC
oding,train_variation_feature_responseCoding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCod
ing,test_variation_feature_responseCoding))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,
cv_variation_feature_responseCoding))

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, trai
n_text_feature_responseCoding))
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_t
ext_feature_responseCoding))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_fe
ature_responseCoding))
```

In [62]:

```
print("One hot TFIDF 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 = ", t
est_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation
 data =", cv_x_onehotCoding.shape)
```

```
One hot TFIDF encoding features :
(number of data points * number of features) in train data =  (2124, 61
93)
(number of data points * number of features) in test data =  (665, 619
3)
(number of data points * number of features) in cross validation data =
(532, 6193)
```

BOW VECTORIZERS STACKING

In [119]:
```
bow_train_gene_var_onehotCoding = hstack((bow_train_gene_feature_onehot
Coding,bow_train_variation_feature_onehotCoding))
bow_test_gene_var_onehotCoding = hstack((bow_test_gene_feature_onehotCo
ding,bow_test_variation_feature_onehotCoding))
bow_cv_gene_var_onehotCoding = hstack((bow_cv_gene_feature_onehotCoding
,bow_cv_variation_feature_onehotCoding))

bow_train_x_onehotCoding = hstack((bow_train_gene_var_onehotCoding, bow
_train_text_feature_onehotCoding)).tocsr()
train_y = np.array(list(train_df['Class']))

bow_test_x_onehotCoding = hstack((bow_test_gene_var_onehotCoding, bow_t
est_text_feature_onehotCoding)).tocsr()
test_y = np.array(list(test_df['Class']))

bow_cv_x_onehotCoding = hstack((bow_cv_gene_var_onehotCoding, bow_cv_te
xt_feature_onehotCoding)).tocsr()
cv_y = np.array(list(cv_df['Class']))
```

In [120]:
```
print("One hot encoding BOW features :")
print("(number of data points * number of features) in train data = ",
bow_train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", b
ow_test_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation
 data =", bow_cv_x_onehotCoding.shape)
```

```
One hot encoding BOW features :
(number of data points * number of features) in train data =  (2124, 76
8188)
(number of data points * number of features) in test data =  (665, 7681
88)
(number of data points * number of features) in cross validation data =
(532, 768188)
```

In [65]:
```
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 = ", t
est_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, 2
7)
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data =
(532, 27)
```

4.1. Base Line Model

4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

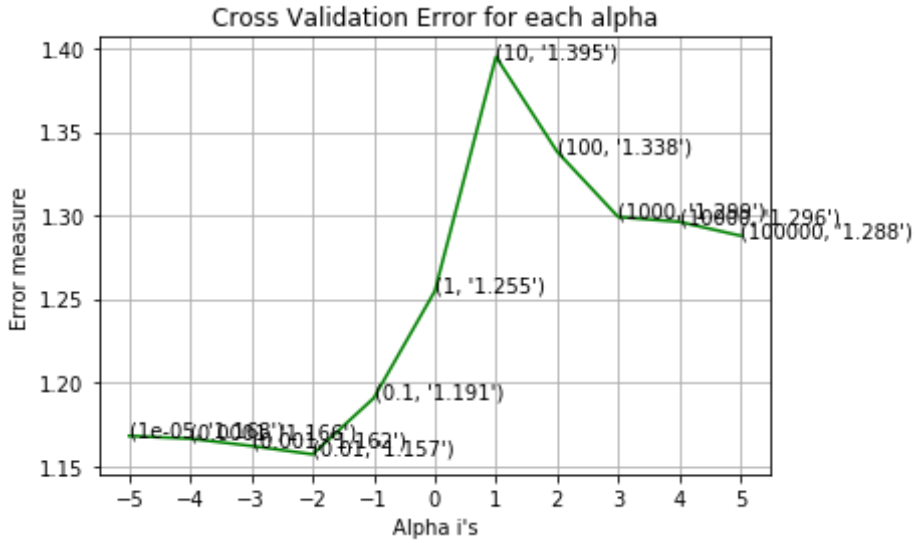
In [66]:
```
# find more about Multinomial Naive base function here http://scikit-le
arn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_pr
ior=None)

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

```
online/lessons/naive-bayes-algorithm-1/
# ----------------------


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


alpha = [0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000,100000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log
-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
```

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


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


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

```
for alpha = 1e-05
Log Loss : 1.1680386411357593
for alpha = 0.0001
Log Loss : 1.1664159197143553
for alpha = 0.001
Log Loss : 1.1620400548896137
for alpha = 0.01
Log Loss : 1.1570162533031074
for alpha = 0.1
```

```
Log Loss : 1.1906451520768753
for alpha = 1
Log Loss : 1.2546897257505363
for alpha = 10
Log Loss : 1.394902111126468
for alpha = 100
Log Loss : 1.338128544776991
for alpha = 1000
Log Loss : 1.2990542410858927
for alpha = 10000
Log Loss : 1.2961529726203291
for alpha = 100000
Log Loss : 1.2878323295482825
```




```
For values of best alpha =  0.01 The train log loss is: 0.7334120289246
701
For values of best alpha =  0.01 The cross validation log loss is: 1.15
70162533031074
For values of best alpha =  0.01 The test log loss is: 1.23490361961070
9
```

4.1.1.2. Testing the model with best hyper paramters

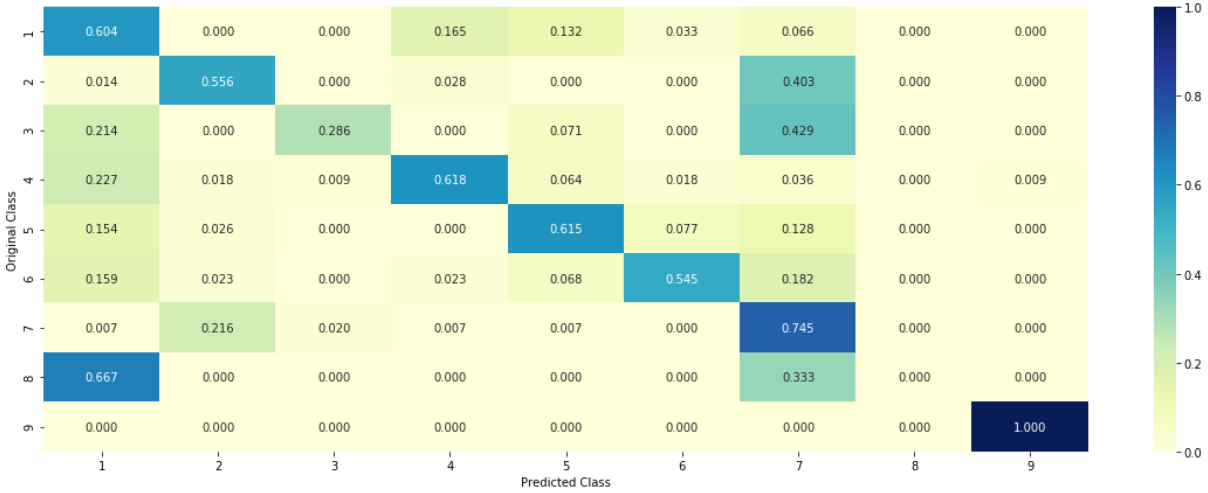
In [67]:
```
# find more about Multinomial Naive base function here http://scikit-le
arn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_pr
ior=None)

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


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

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

```
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-pro
bability estimates
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.pre
dict(cv_x_onehotCoding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray
()))
```

```
Log Loss : 1.1570162533031074
Number of missclassified point : 0.37030075187969924
-------------------- Confusion matrix --------------------
```


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


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



4.1.1.3. Feature Importance, Correctly classified point

In [68]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
```

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0642 0.0491 0.0129 0.0731 0.037  0.6
716 0.0835 0.0041 0.0045]]
Actual Class : 6
--------------------------------------------------
5 Text feature [brca] present in test data point [True]
6 Text feature [odds] present in test data point [True]
8 Text feature [57] present in test data point [True]
9 Text feature [deleterious] present in test data point [True]
10 Text feature [classified] present in test data point [True]
11 Text feature [predicted] present in test data point [True]
12 Text feature [basis] present in test data point [True]
13 Text feature [personal] present in test data point [True]
14 Text feature [carry] present in test data point [True]
15 Text feature [classify] present in test data point [True]
16 Text feature [favor] present in test data point [True]
17 Text feature [expected] present in test data point [True]
18 Text feature [combined] present in test data point [True]
19 Text feature [likelihood] present in test data point [True]
20 Text feature [history] present in test data point [True]
21 Text feature [model] present in test data point [True]
22 Text feature [family] present in test data point [True]
23 Text feature [threshold] present in test data point [True]
24 Text feature [models] present in test data point [True]
25 Text feature [alignments] present in test data point [True]
26 Text feature [433] present in test data point [True]
27 Text feature [laboratories] present in test data point [True]
28 Text feature [ethnic] present in test data point [True]
29 Text feature [43] present in test data point [True]
30 Text feature [sequence] present in test data point [True]
```

```
31 Text feature [cosegregation] present in test data point [True]
32 Text feature [56] present in test data point [True]
33 Text feature [sources] present in test data point [True]
34 Text feature [predictive] present in test data point [True]
35 Text feature [26] present in test data point [True]
36 Text feature [myriad] present in test data point [True]
37 Text feature [evidence] present in test data point [True]
38 Text feature [conservation] present in test data point [True]
39 Text feature [trans] present in test data point [True]
40 Text feature [substitutions] present in test data point [True]
41 Text feature [brca1] present in test data point [True]
42 Text feature [logistic] present in test data point [True]
43 Text feature [09] present in test data point [True]
44 Text feature [would] present in test data point [True]
45 Text feature [estimated] present in test data point [True]
46 Text feature [useful] present in test data point [True]
47 Text feature [testing] present in test data point [True]
48 Text feature [76] present in test data point [True]
49 Text feature [000] present in test data point [True]
50 Text feature [02] present in test data point [True]
52 Text feature [vuss] present in test data point [True]
53 Text feature [06] present in test data point [True]
54 Text feature [probabilities] present in test data point [True]
55 Text feature [outside] present in test data point [True]
56 Text feature [79] present in test data point [True]
57 Text feature [ovarian] present in test data point [True]
58 Text feature [powerful] present in test data point [True]
59 Text feature [occurrence] present in test data point [True]
60 Text feature [probability] present in test data point [True]
61 Text feature [use] present in test data point [True]
62 Text feature [used] present in test data point [True]
63 Text feature [known] present in test data point [True]
64 Text feature [tests] present in test data point [True]
65 Text feature [individuals] present in test data point [True]
66 Text feature [lr] present in test data point [True]
67 Text feature [04] present in test data point [True]
68 Text feature [substitution] present in test data point [True]
69 Text feature [81] present in test data point [True]
70 Text feature [49] present in test data point [True]
```

```
71 Text feature [causality] present in test data point [True]
72 Text feature [00] present in test data point [True]
73 Text feature [68] present in test data point [True]
74 Text feature [133] present in test data point [True]
75 Text feature [histopathology] present in test data point [True]
76 Text feature [likely] present in test data point [True]
77 Text feature [disrupt] present in test data point [True]
78 Text feature [variant] present in test data point [True]
79 Text feature [74] present in test data point [True]
81 Text feature [examples] present in test data point [True]
82 Text feature [bic] present in test data point [True]
83 Text feature [gvgd] present in test data point [True]
85 Text feature [ring] present in test data point [True]
86 Text feature [conserved] present in test data point [True]
87 Text feature [08] present in test data point [True]
88 Text feature [ligase] present in test data point [True]
89 Text feature [given] present in test data point [True]
90 Text feature [70] present in test data point [True]
95 Text feature [probands] present in test data point [True]
97 Text feature [07] present in test data point [True]
98 Text feature [brca2] present in test data point [True]
99 Text feature [significant] present in test data point [True]
Out of the top  100  features  86 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0686 0.05   0.0129 0.6973 0.0372 0.0
384 0.087  0.0041 0.0045]]
Actual Class : 4
--------------------------------------------------
11 Text feature [activity] present in test data point [True]
12 Text feature [proteins] present in test data point [True]
13 Text feature [protein] present in test data point [True]
17 Text feature [function] present in test data point [True]
18 Text feature [experiments] present in test data point [True]
19 Text feature [whether] present in test data point [True]
22 Text feature [shown] present in test data point [True]
24 Text feature [determined] present in test data point [True]
28 Text feature [described] present in test data point [True]
30 Text feature [two] present in test data point [True]
32 Text feature [results] present in test data point [True]
33 Text feature [acid] present in test data point [True]
35 Text feature [ability] present in test data point [True]
37 Text feature [bind] present in test data point [True]
38 Text feature [also] present in test data point [True]
40 Text feature [levels] present in test data point [True]
41 Text feature [amino] present in test data point [True]
43 Text feature [reduced] present in test data point [True]
44 Text feature [either] present in test data point [True]
45 Text feature [containing] present in test data point [True]
46 Text feature [mutations] present in test data point [True]
47 Text feature [retained] present in test data point [True]
48 Text feature [may] present in test data point [True]
50 Text feature [transfected] present in test data point [True]
51 Text feature [expressed] present in test data point [True]
52 Text feature [vitro] present in test data point [True]
55 Text feature [expression] present in test data point [True]
57 Text feature [thus] present in test data point [True]
58 Text feature [although] present in test data point [True]
60 Text feature [suggest] present in test data point [True]
62 Text feature [tagged] present in test data point [True]
63 Text feature [analyzed] present in test data point [True]
67 Text feature [lower] present in test data point [True]

68 Text feature [result] present in test data point [True]
```

```
70 Text feature [amount] present in test data point [True]
71 Text feature [using] present in test data point [True]
72 Text feature [cells] present in test data point [True]
74 Text feature [however] present in test data point [True]
75 Text feature [analysis] present in test data point [True]
76 Text feature [effects] present in test data point [True]
77 Text feature [three] present in test data point [True]
78 Text feature [suggesting] present in test data point [True]
79 Text feature [incubated] present in test data point [True]
80 Text feature [buffer] present in test data point [True]
82 Text feature [transfection] present in test data point [True]
83 Text feature [previously] present in test data point [True]
84 Text feature [fact] present in test data point [True]
85 Text feature [binding] present in test data point [True]
86 Text feature [similar] present in test data point [True]
87 Text feature [mutant] present in test data point [True]
88 Text feature [catalytic] present in test data point [True]
90 Text feature [standard] present in test data point [True]
91 Text feature [vivo] present in test data point [True]
92 Text feature [role] present in test data point [True]
93 Text feature [addition] present in test data point [True]
96 Text feature [could] present in test data point [True]
98 Text feature [possible] present in test data point [True]
99 Text feature [vector] present in test data point [True]
Out of the top  100  features  58 are present in query point
```

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

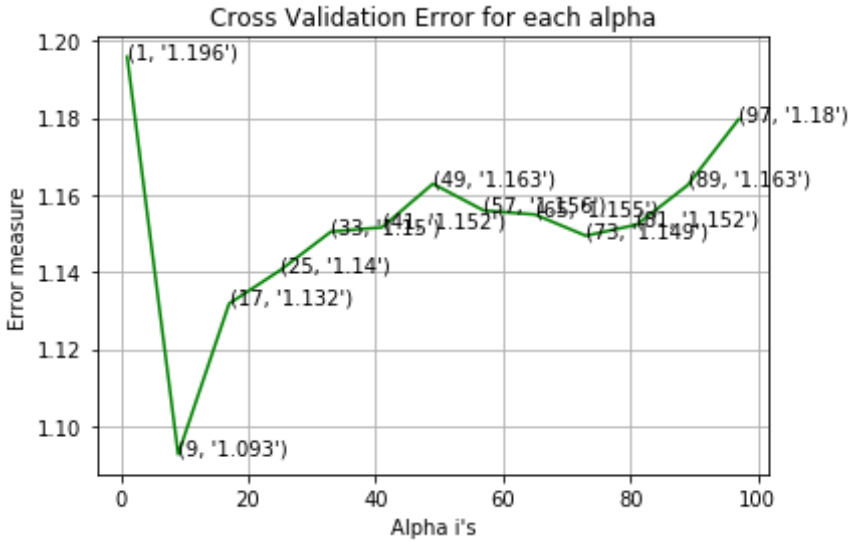
In [70]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
```

```
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target va
lues
# 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-examp
le-1/
#-------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='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 = list(range(1,100,8))
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log
-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(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_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
for alpha = 1
Log Loss : 1.1958248710093093

for alpha = 9
```

```
for alpha = 9
Log Loss : 1.092642107684917
for alpha = 17
Log Loss : 1.131754775076849
for alpha = 25
Log Loss : 1.1403972052430176
for alpha = 33
Log Loss : 1.150450206405515
for alpha = 41
Log Loss : 1.151560430674955
for alpha = 49
Log Loss : 1.1628711876143254
for alpha = 57
Log Loss : 1.1559087363908411
for alpha = 65
Log Loss : 1.1549000764442225
for alpha = 73
Log Loss : 1.1493849112818233
for alpha = 81
Log Loss : 1.1522670701047213
for alpha = 89
Log Loss : 1.1625314988660518
for alpha = 97
Log Loss : 1.1796203058051067
```



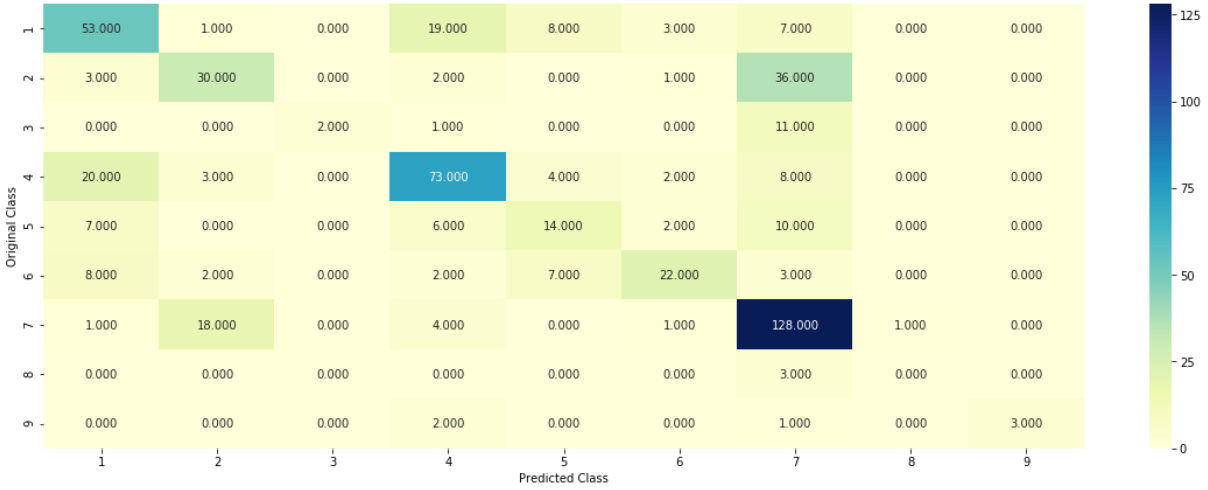

```
For values of best alpha =  9 The train log loss is: 1.013257458139026
For values of best alpha =  9 The cross validation log loss is: 1.09264
2107684917
For values of best alpha =  9 The test log loss is: 1.144284862005468
```

4.2.2. Testing the model with best hyper paramters

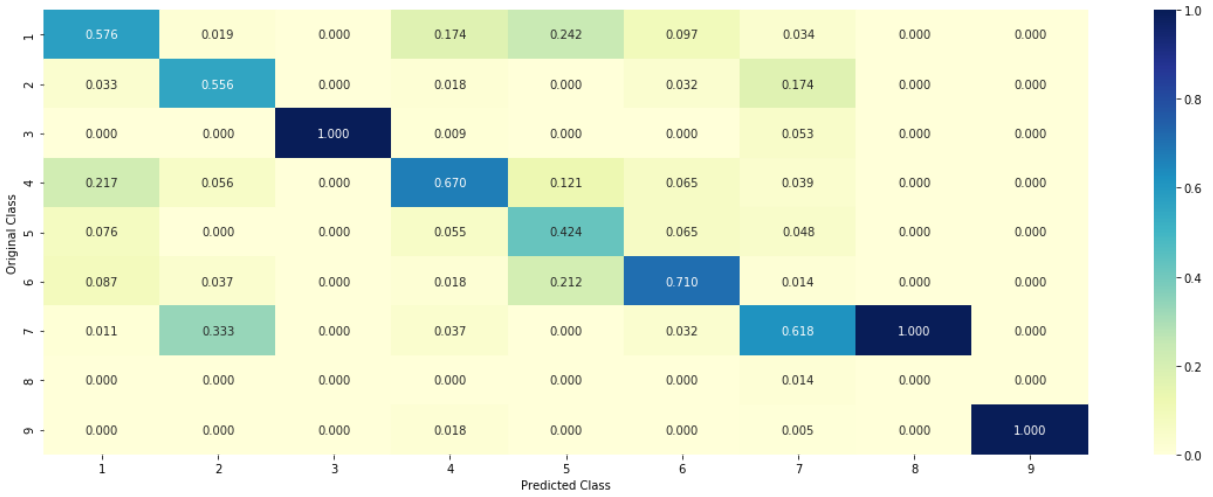
In [71]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/s
table/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='aut
o', leaf_size=30, p=2,
# metric='minkowski',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

# methods of
# fit(X, y) : Fit the model using X as training data and y as target va
lues
# 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_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

```
Log loss : 1.092642107684917
Number of mis-classified points : 0.3890977443609023
-------------------- Confusion matrix --------------------
```




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



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

4.2.3.Sample Query point -1

In [72]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```
sig_clf.fit(train_x_onehotCoding, train_y)

test_point_index = 1
predicted_cls = sig_clf.predict(test_x_onehotCoding[0].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_onehotCoding[test_point_index].reshap
e(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 : 6
The  9  nearest neighbours of the test points belongs to classes [6 6 6
6 6 6 6 6 6]
Fequency of nearest points : Counter({6: 9})
```

4.2.4. Sample Query Point-2

In [73]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

test_point_index = 100

predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index].r
eshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_onehotCoding[test_point_index].reshap
e(1, -1), alpha[best_alpha])
print("the k value for knn is",alpha[best_alpha],"and the nearest neigh
bours of the test points belongs to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 4
```

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

4.3. Logistic Regression

4.3.1. With Class balancing

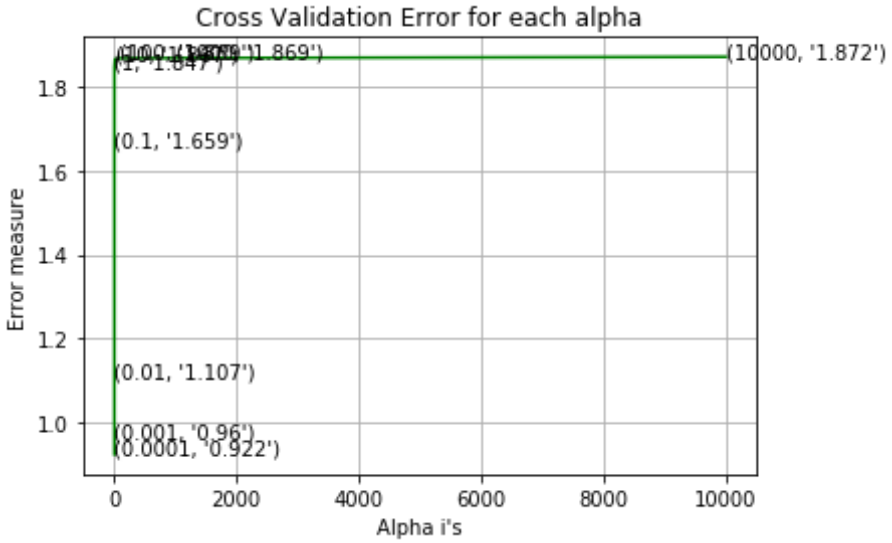
4.3.1.1. Hyper paramter tuning

In [74]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic 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.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
```

```
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='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 = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
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], p
enalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
for alpha = 0.0001
Log Loss : 0.9223695685112387
for alpha = 0.001
Log Loss : 0.9598099804765026
for alpha = 0.01
Log Loss : 1.1072331777906177
for alpha = 0.1
Log Loss : 1.6593404251433737
for alpha = 1
Log Loss : 1.8473003895697764
for alpha = 10
Log Loss : 1.8666863007367607
for alpha = 100
Log Loss : 1.8688120030952273
for alpha = 1000
Log Loss : 1.8691659063742792
for alpha = 10000
Log Loss : 1.8721283343193307
```




```
For values of best alpha =  0.0001 The train log loss is: 0.44423015484
486705
For values of best alpha =  0.0001 The cross validation log loss is: 0.
9223695685112387
For values of best alpha =  0.0001 The test log loss is: 0.965906464545
8479
```

4.3.1.2. Testing the model with best hyper paramters

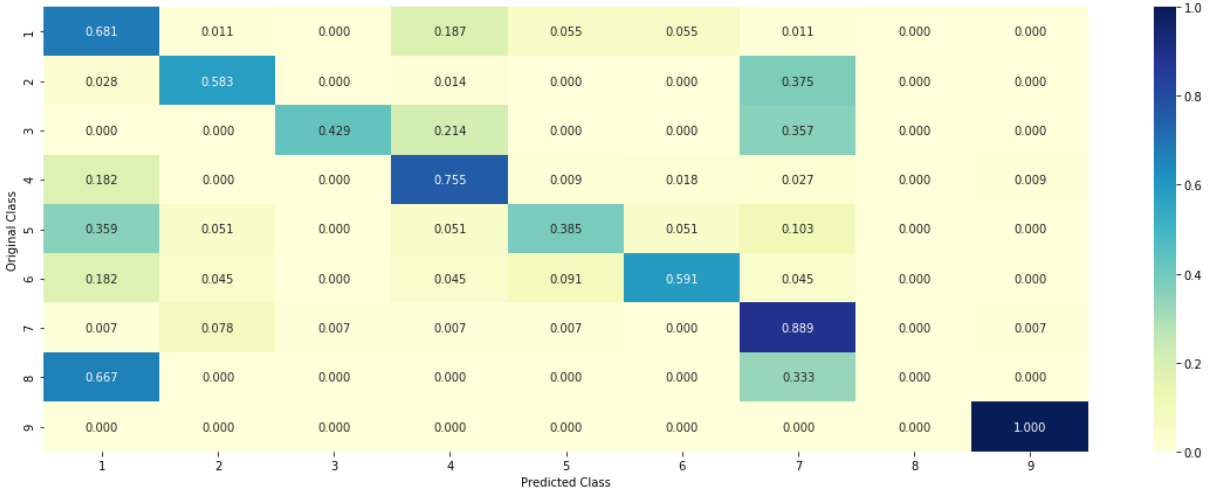
In [75]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
```

```
tochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

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

```
Log loss : 0.9223695685112387
Number of mis-classified points : 0.2932330827067669
-------------------- Confusion matrix --------------------
```


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

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



4.3.1.3. Feature Importance

In [76]:
```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
```

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

4.3.1.3.1. Correctly Classified point

In [77]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[2.040e-02 8.800e-03 2.100e-03 4.900e-0
3 1.240e-02 9.445e-01 3.000e-04
  6.500e-03 1.000e-04]]
Actual Class : 6
--------------------------------------------------
70 Text feature [women] present in test data point [True]
93 Text feature [trans] present in test data point [True]
100 Text feature [highlights] present in test data point [True]
102 Text feature [ovarian] present in test data point [True]
123 Text feature [species] present in test data point [True]
131 Text feature [73] present in test data point [True]
146 Text feature [threshold] present in test data point [True]
152 Text feature [statistically] present in test data point [True]
158 Text feature [values] present in test data point [True]
175 Text feature [area] present in test data point [True]
186 Text feature [brca] present in test data point [True]
187 Text feature [considered] present in test data point [True]
192 Text feature [regarding] present in test data point [True]
214 Text feature [significant] present in test data point [True]
216 Text feature [occurs] present in test data point [True]
217 Text feature [conferred] present in test data point [True]
219 Text feature [77] present in test data point [True]
226 Text feature [problem] present in test data point [True]
236 Text feature [heterozygosity] present in test data point [True]
248 Text feature [scores] present in test data point [True]
252 Text feature [72] present in test data point [True]
260 Text feature [lethal] present in test data point [True]
272 Text feature [ethnic] present in test data point [True]
288 Text feature [preliminary] present in test data point [True]
294 Text feature [000] present in test data point [True]
295 Text feature [typically] present in test data point [True]
302 Text feature [studied] present in test data point [True]
305 Text feature [tests] present in test data point [True]
308 Text feature [defining] present in test data point [True]
310 Text feature [133] present in test data point [True]
```

```
311 Text feature [predictive] present in test data point [True]
314 Text feature [terms] present in test data point [True]
322 Text feature [alter] present in test data point [True]
330 Text feature [areas] present in test data point [True]
331 Text feature [clinically] present in test data point [True]
333 Text feature [42] present in test data point [True]
334 Text feature [basis] present in test data point [True]
343 Text feature [110] present in test data point [True]
345 Text feature [susceptible] present in test data point [True]
355 Text feature [list] present in test data point [True]
356 Text feature [practice] present in test data point [True]
358 Text feature [models] present in test data point [True]
361 Text feature [orthologs] present in test data point [True]
368 Text feature [68] present in test data point [True]
384 Text feature [51] present in test data point [True]
385 Text feature [elsewhere] present in test data point [True]
397 Text feature [57] present in test data point [True]
401 Text feature [deleterious] present in test data point [True]
402 Text feature [uncertain] present in test data point [True]
404 Text feature [important] present in test data point [True]
405 Text feature [free] present in test data point [True]
406 Text feature [showing] present in test data point [True]
414 Text feature [proband] present in test data point [True]
421 Text feature [separately] present in test data point [True]
424 Text feature [49] present in test data point [True]
427 Text feature [determination] present in test data point [True]
429 Text feature [46] present in test data point [True]
432 Text feature [age] present in test data point [True]
433 Text feature [e3] present in test data point [True]
438 Text feature [logistic] present in test data point [True]
448 Text feature [classified] present in test data point [True]
453 Text feature [coding] present in test data point [True]
456 Text feature [family] present in test data point [True]
460 Text feature [analogous] present in test data point [True]
464 Text feature [history] present in test data point [True]
465 Text feature [reflect] present in test data point [True]
466 Text feature [population] present in test data point [True]
469 Text feature [final] present in test data point [True]
471 Text feature [histopathology] present in test data point [True]
```

```
474 Text feature [certain] present in test data point [True]
479 Text feature [lr] present in test data point [True]
480 Text feature [largely] present in test data point [True]
484 Text feature [433] present in test data point [True]
486 Text feature [occurrence] present in test data point [True]
498 Text feature [106] present in test data point [True]
Out of the top  500  features  75 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [78]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0626 0.0323 0.0143 0.7647 0.0191 0.0
156 0.0794 0.0047 0.0073]]
Actual Class : 4
--------------------------------------------------
183 Text feature [beads] present in test data point [True]
199 Text feature [bsa] present in test data point [True]
275 Text feature [transfections] present in test data point [True]
297 Text feature [plated] present in test data point [True]
299 Text feature [contributes] present in test data point [True]
327 Text feature [cloned] present in test data point [True]
331 Text feature [fibroblasts] present in test data point [True]
345 Text feature [condition] present in test data point [True]
347 Text feature [exact] present in test data point [True]
349 Text feature [plates] present in test data point [True]
```

```
350 Text feature [assessing] present in test data point [True]
353 Text feature [predominant] present in test data point [True]
373 Text feature [bind] present in test data point [True]
382 Text feature [substrate] present in test data point [True]
388 Text feature [evident] present in test data point [True]
393 Text feature [represent] present in test data point [True]
411 Text feature [plasmid] present in test data point [True]
425 Text feature [consequences] present in test data point [True]
435 Text feature [rho] present in test data point [True]
458 Text feature [comprehensive] present in test data point [True]
463 Text feature [caused] present in test data point [True]
487 Text feature [release] present in test data point [True]
489 Text feature [washed] present in test data point [True]
Out of the top  500  features  23 are present in query point
```

SGD Logistic Regression With CountVectorizer

In [79]:

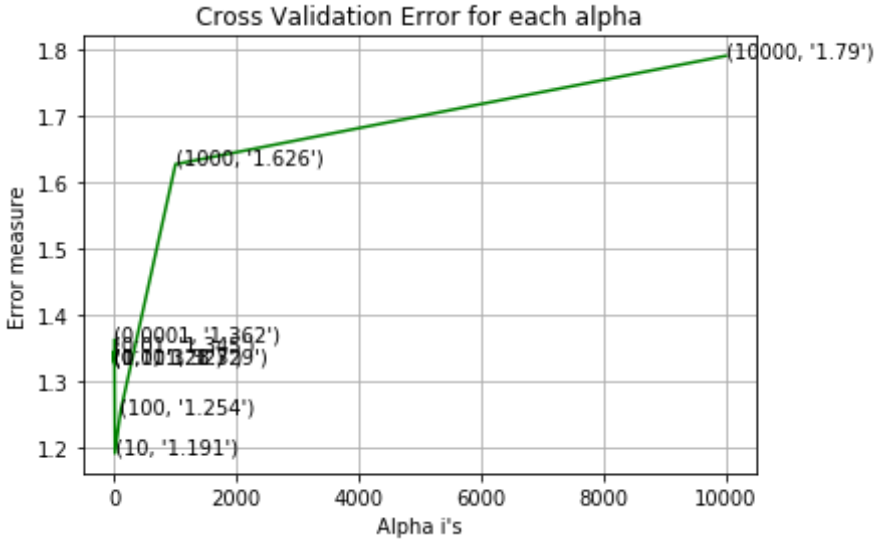
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic 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 = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
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(bow_train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(bow_train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(bow_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], p
enalty='l2', loss='log', random_state=42)
clf.fit(bow_train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(bow_train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(bow_train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log
 loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(bow_cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "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(bow_test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 0.0001
Log Loss : 1.3615072617406299
for alpha = 0.001
Log Loss : 1.3292465724017999
for alpha = 0.01
Log Loss : 1.3453792537769307
for alpha = 0.1
Log Loss : 1.3266564165333072
for alpha = 1
Log Loss : 1.3279151572884742
for alpha = 10
Log Loss : 1.1911428356945006
for alpha = 100
Log Loss : 1.253868491054133
for alpha = 1000
Log Loss : 1.6263235414251338
```

```
for alpha = 10000
Log Loss : 1.7895303731116528
```

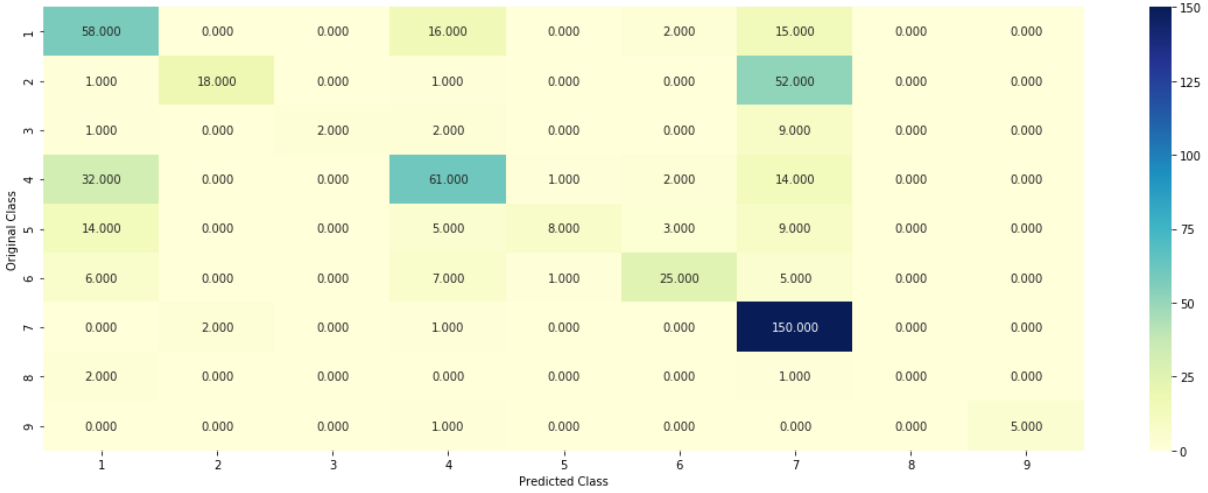



```
For values of best alpha =  10 The train log loss is: 1.001564427994144
2
For values of best alpha =  10 The cross validation log loss is: 1.1911
428356945006
For values of best alpha =  10 The test log loss is: 1.242500404711015
```

In [80]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic Gradient Descent.
```

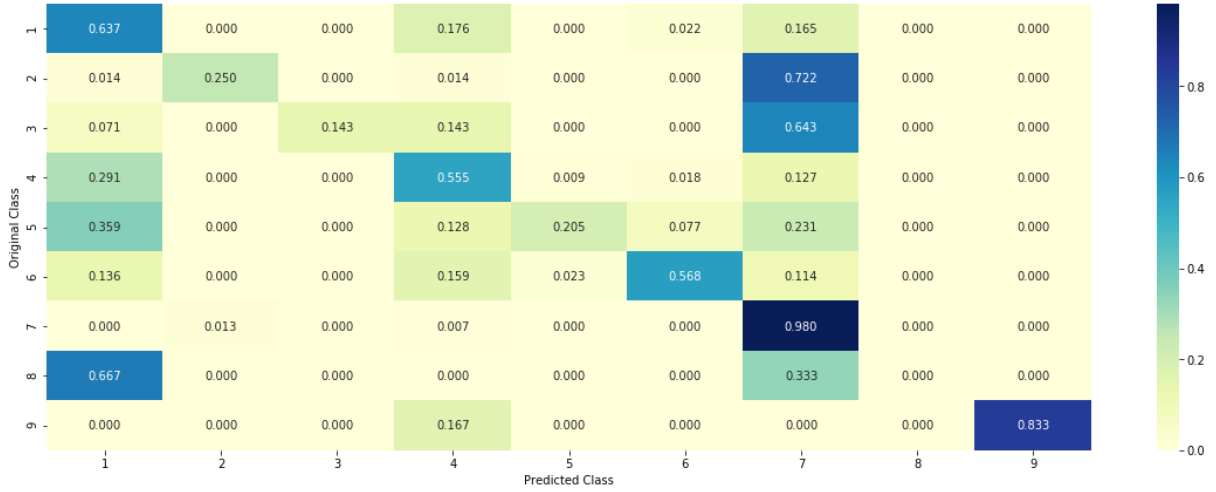
```
# predict(X)    Predict class labels for samples in X.

#--------------------------------
# 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(bow_train_x_onehotCoding, train_y, bow_cv_x_onehotCoding, cv_y, clf)
```

```
Log loss : 1.1911428356945006
Number of mis-classified points : 0.38533834586466165
-------------------- Confusion matrix --------------------
```




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

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



SGD With out Class Balancing

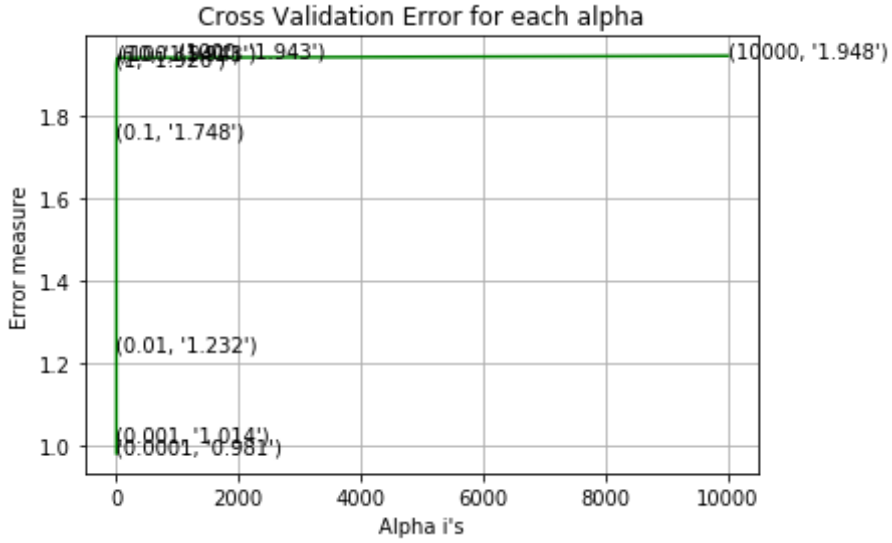
4.3.2.1. Hyper paramter tuning

In [81]:
```
# 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 = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state
=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

```
for alpha = 0.0001
Log Loss : 0.9807918653365527
for alpha = 0.001
Log Loss : 1.013512973341512
for alpha = 0.01
Log Loss : 1.232033574018136
for alpha = 0.1
Log Loss : 1.7478620416998383
for alpha = 1
Log Loss : 1.9257423369166742
for alpha = 10
Log Loss : 1.9414232134477407
for alpha = 100
Log Loss : 1.943066264889244
for alpha = 1000
Log Loss : 1.9432870178021506
for alpha = 10000
Log Loss : 1.947648483408229
```

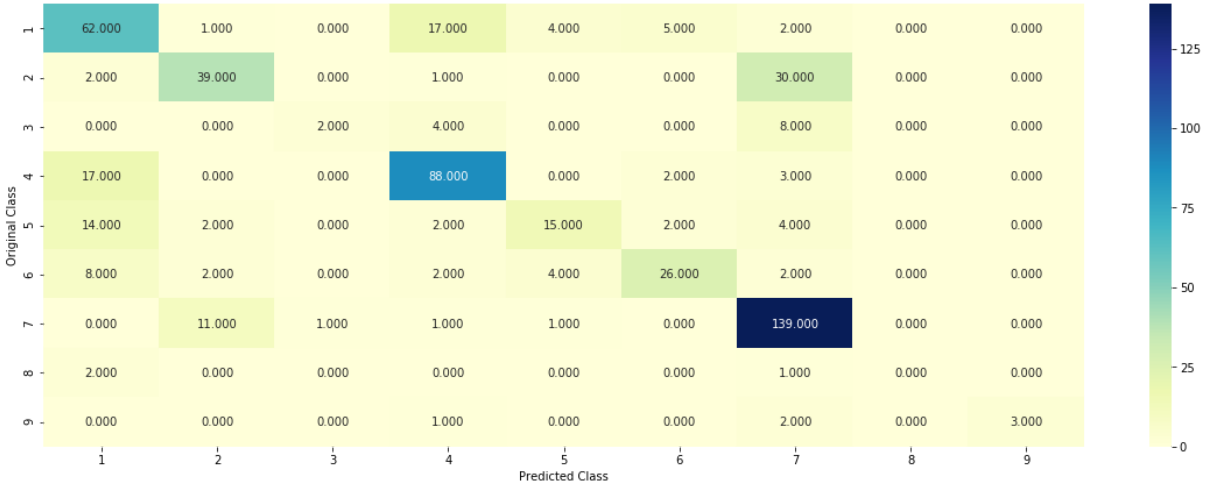


```
For values of best alpha =  0.0001 The train log loss is: 0.43799522993
79181
For values of best alpha =  0.0001 The cross validation log loss is: 0.
9807918653365527
For values of best alpha =  0.0001 The test log loss is: 0.998264180521
8792
```

4.3.2.2. Testing model with best hyper parameters

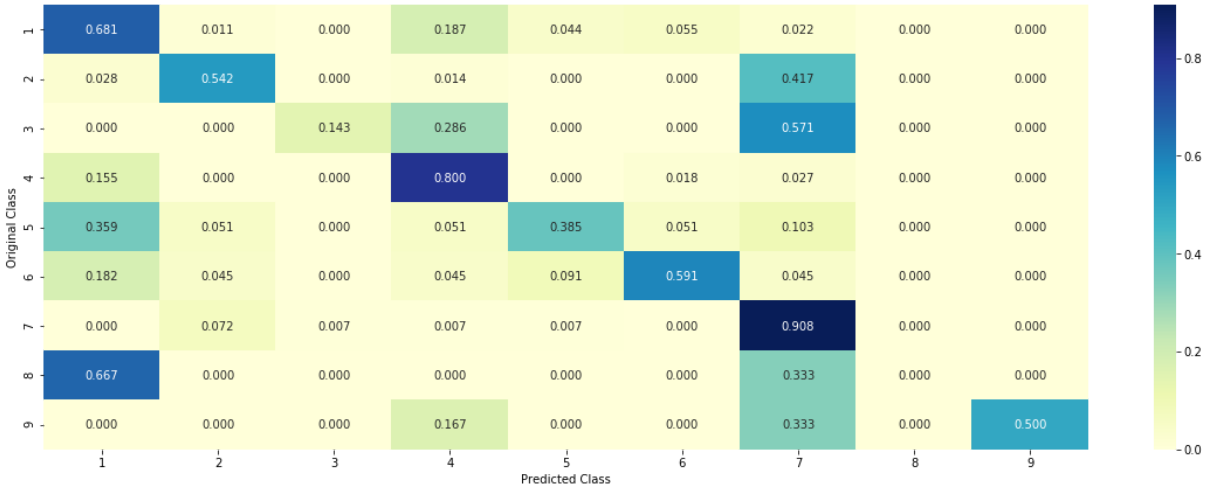
In [82]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

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

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

```
Log loss : 0.9807918653365527
Number of mis-classified points : 0.29699248120300753
-------------------- Confusion matrix --------------------
```



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

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



4.3.2.3. Feature Importance, Correctly Classified point

In [83]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[2.300e-02 9.000e-03 7.000e-04 7.400e-0
3 1.120e-02 9.301e-01 4.000e-04
  1.810e-02 1.000e-04]]
```

```
Actual Class : 6
--------------------------------------------------
70 Text feature [women] present in test data point [True]
91 Text feature [trans] present in test data point [True]
94 Text feature [highlights] present in test data point [True]
100 Text feature [ovarian] present in test data point [True]
114 Text feature [73] present in test data point [True]
117 Text feature [species] present in test data point [True]
132 Text feature [statistically] present in test data point [True]
134 Text feature [threshold] present in test data point [True]
156 Text feature [values] present in test data point [True]
174 Text feature [brca] present in test data point [True]
181 Text feature [area] present in test data point [True]
193 Text feature [considered] present in test data point [True]
199 Text feature [problem] present in test data point [True]
204 Text feature [regarding] present in test data point [True]
209 Text feature [conferred] present in test data point [True]
220 Text feature [occurs] present in test data point [True]
223 Text feature [significant] present in test data point [True]
246 Text feature [scores] present in test data point [True]
253 Text feature [77] present in test data point [True]
256 Text feature [heterozygosity] present in test data point [True]
261 Text feature [ethnic] present in test data point [True]
262 Text feature [72] present in test data point [True]
277 Text feature [preliminary] present in test data point [True]
282 Text feature [110] present in test data point [True]
284 Text feature [lethal] present in test data point [True]
286 Text feature [typically] present in test data point [True]
289 Text feature [tests] present in test data point [True]
305 Text feature [clinically] present in test data point [True]
309 Text feature [predictive] present in test data point [True]
310 Text feature [000] present in test data point [True]
315 Text feature [defining] present in test data point [True]
323 Text feature [models] present in test data point [True]
324 Text feature [terms] present in test data point [True]
333 Text feature [133] present in test data point [True]
334 Text feature [list] present in test data point [True]
338 Text feature [studied] present in test data point [True]
342 Text feature [57] present in test data point [True]
```

```
345 Text feature [68] present in test data point [True]
347 Text feature [42] present in test data point [True]
349 Text feature [elsewhere] present in test data point [True]
350 Text feature [alter] present in test data point [True]
366 Text feature [susceptible] present in test data point [True]
371 Text feature [areas] present in test data point [True]
373 Text feature [basis] present in test data point [True]
376 Text feature [orthologs] present in test data point [True]
386 Text feature [deleterious] present in test data point [True]
387 Text feature [uncertain] present in test data point [True]
390 Text feature [separately] present in test data point [True]
391 Text feature [practice] present in test data point [True]
412 Text feature [showing] present in test data point [True]
415 Text feature [433] present in test data point [True]
423 Text feature [49] present in test data point [True]
425 Text feature [51] present in test data point [True]
427 Text feature [proband] present in test data point [True]
428 Text feature [46] present in test data point [True]
430 Text feature [final] present in test data point [True]
431 Text feature [determination] present in test data point [True]
433 Text feature [e3] present in test data point [True]
434 Text feature [important] present in test data point [True]
443 Text feature [male] present in test data point [True]
445 Text feature [classified] present in test data point [True]
447 Text feature [logistic] present in test data point [True]
455 Text feature [significance] present in test data point [True]
458 Text feature [age] present in test data point [True]
460 Text feature [population] present in test data point [True]
464 Text feature [history] present in test data point [True]
466 Text feature [analogous] present in test data point [True]
467 Text feature [expected] present in test data point [True]
473 Text feature [predicted] present in test data point [True]
475 Text feature [family] present in test data point [True]
477 Text feature [histopathology] present in test data point [True]
481 Text feature [coding] present in test data point [True]
483 Text feature [dose] present in test data point [True]
494 Text feature [02] present in test data point [True]
497 Text feature [free] present in test data point [True]
```

```
498 Text feature [ligase] present in test data point [True]
Out of the top  500  features  76 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [84]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0593 0.033  0.0094 0.7746 0.018  0.0
139 0.0851 0.0049 0.0017]]
Actual Class : 4
--------------------------------------------------
182 Text feature [beads] present in test data point [True]
214 Text feature [plated] present in test data point [True]
233 Text feature [bsa] present in test data point [True]
279 Text feature [plates] present in test data point [True]
283 Text feature [transfections] present in test data point [True]
318 Text feature [exact] present in test data point [True]
339 Text feature [fibroblasts] present in test data point [True]
341 Text feature [plasmid] present in test data point [True]
348 Text feature [cloned] present in test data point [True]
354 Text feature [predominant] present in test data point [True]
360 Text feature [assessing] present in test data point [True]
366 Text feature [contributes] present in test data point [True]
370 Text feature [represent] present in test data point [True]
373 Text feature [substrate] present in test data point [True]
374 Text feature [condition] present in test data point [True]
385 Text feature [bind] present in test data point [True]
```

```
385 Text feature [bind] present in test data point [True]
398 Text feature [consequences] present in test data point [True]
424 Text feature [release] present in test data point [True]
443 Text feature [evident] present in test data point [True]
445 Text feature [comprehensive] present in test data point [True]
470 Text feature [rho] present in test data point [True]
490 Text feature [caused] present in test data point [True]
Out of the top  500  features  22 are present in query point
```

4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

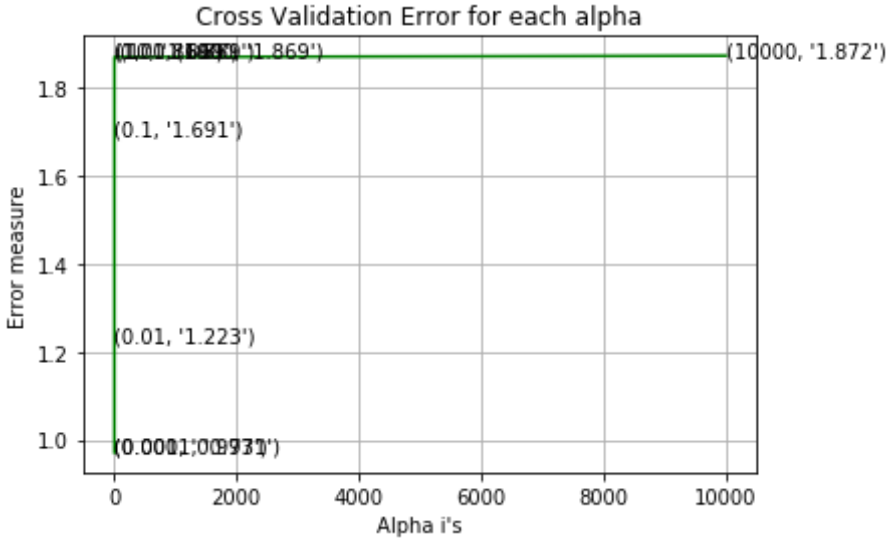
In [85]:
```
# read more about support vector machines with linear kernals here htt
p://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

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

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



# find more about CalibratedClassifierCV here at http://scikit-learn.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
```

```
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='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 = [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
#     clf = SVC(C=i,kernel='linear',probability=True, class_weight='bal
anced')
    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='balance
d')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='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 is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross 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_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
for C = 0.0001
Log Loss : 0.971207927108453
for C = 0.001
Log Loss : 0.9728865858664354
for C = 0.01
Log Loss : 1.2225974068045398
for C = 0.1
Log Loss : 1.6907660791038077
for C = 1
Log Loss : 1.869276085258842
for C = 10
Log Loss : 1.8692754421176596
for C = 100
Log Loss : 1.869275697109565
for C = 1000
Log Loss : 1.8693003601327705
```

```
for C = 10000
Log Loss : 1.8721872446286771
```




```
For values of best alpha =  0.0001 The train log loss is: 0.54982528293
25296
For values of best alpha =  0.0001 The cross validation log loss is: 0.
971207927108453
For values of best alpha =  0.0001 The test log loss is: 1.041964166678
8424
```

4.4.2. Testing model with best hyper parameters

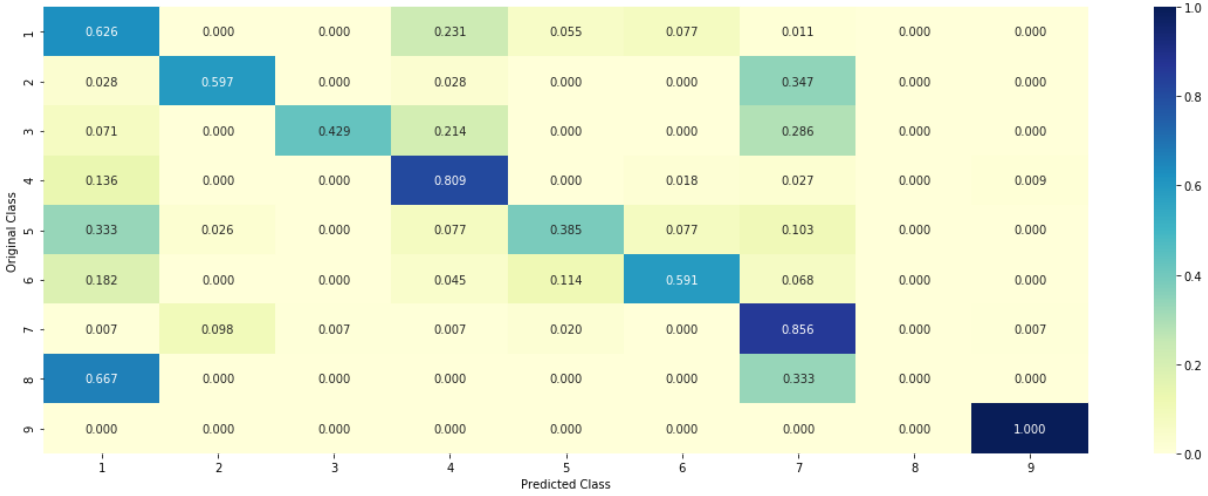
In [86]:
```
# read more about support vector machines with linear kernals here htt
p://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

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

```
on_function_shape='ovr', random_state=None)

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


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

```
Log loss : 0.971207927108453
Number of mis-classified points : 0.29887218045112784
-------------------- 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 [87]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge'
, random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[3.110e-02 5.220e-02 3.100e-03 4.660e-0
2 1.590e-02 8.371e-01 6.900e-03
  7.000e-03 1.000e-04]]
Actual Class : 6
--------------------------------------------------
86 Text feature [trans] present in test data point [True]
92 Text feature [highlights] present in test data point [True]
94 Text feature [women] present in test data point [True]
96 Text feature [species] present in test data point [True]
101 Text feature [threshold] present in test data point [True]
109 Text feature [values] present in test data point [True]
110 Text feature [regarding] present in test data point [True]
114 Text feature [scores] present in test data point [True]
121 Text feature [ovarian] present in test data point [True]
193 Text feature [statistically] present in test data point [True]
195 Text feature [brca] present in test data point [True]
212 Text feature [considered] present in test data point [True]
215 Text feature [practice] present in test data point [True]
223 Text feature [occurs] present in test data point [True]
230 Text feature [terms] present in test data point [True]
231 Text feature [significant] present in test data point [True]

238 Text feature [area] present in test data point [True]
```

```
238 Text feature [area] present in test data point [True]
243 Text feature [problem] present in test data point [True]
249 Text feature [list] present in test data point [True]
252 Text feature [ethnic] present in test data point [True]
258 Text feature [purpose] present in test data point [True]
262 Text feature [evaluate] present in test data point [True]
273 Text feature [77] present in test data point [True]
277 Text feature [lr] present in test data point [True]
279 Text feature [clinically] present in test data point [True]
284 Text feature [coding] present in test data point [True]
297 Text feature [preliminary] present in test data point [True]
305 Text feature [algorithm] present in test data point [True]
306 Text feature [orthologs] present in test data point [True]
311 Text feature [analogous] present in test data point [True]
313 Text feature [largely] present in test data point [True]
315 Text feature [classified] present in test data point [True]
320 Text feature [models] present in test data point [True]
326 Text feature [studied] present in test data point [True]
332 Text feature [occurrence] present in test data point [True]
333 Text feature [altered] present in test data point [True]
334 Text feature [deleterious] present in test data point [True]
342 Text feature [substitutions] present in test data point [True]
345 Text feature [57] present in test data point [True]
357 Text feature [susceptible] present in test data point [True]
360 Text feature [predictive] present in test data point [True]
361 Text feature [73] present in test data point [True]
362 Text feature [000] present in test data point [True]
384 Text feature [polymorphisms] present in test data point [True]
386 Text feature [433] present in test data point [True]
389 Text feature [predicted] present in test data point [True]
390 Text feature [associated] present in test data point [True]
405 Text feature [tests] present in test data point [True]
411 Text feature [overall] present in test data point [True]
416 Text feature [106] present in test data point [True]
417 Text feature [numbers] present in test data point [True]
418 Text feature [expected] present in test data point [True]
422 Text feature [basis] present in test data point [True]
424 Text feature [free] present in test data point [True]
433 Text feature [observation] present in test data point [True]

438 Text feature [observe] present in test data point [True]
```

```
438 Text feature [observe] present in test data point [True]
441 Text feature [heterozygosity] present in test data point [True]
449 Text feature [logistic] present in test data point [True]
452 Text feature [133] present in test data point [True]
456 Text feature [invariant] present in test data point [True]
464 Text feature [72] present in test data point [True]
468 Text feature [showing] present in test data point [True]
473 Text feature [significance] present in test data point [True]
478 Text feature [personal] present in test data point [True]
481 Text feature [contained] present in test data point [True]
483 Text feature [alignments] present in test data point [True]
486 Text feature [conservation] present in test data point [True]
487 Text feature [history] present in test data point [True]
489 Text feature [verify] present in test data point [True]
495 Text feature [02] present in test data point [True]
Out of the top  500  features  70 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [88]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.076  0.0868 0.0166 0.6362 0.0322 0.0
207 0.1208 0.0039 0.0067]]
Actual Class : 4
--------------------------------------------------
174 Text feature [plated] present in test data point [True]
```

```
176 Text feature [beads] present in test data point [True]
188 Text feature [bsa] present in test data point [True]
211 Text feature [represent] present in test data point [True]
212 Text feature [consequences] present in test data point [True]
223 Text feature [cloned] present in test data point [True]
231 Text feature [predominant] present in test data point [True]
232 Text feature [caused] present in test data point [True]
245 Text feature [contributes] present in test data point [True]
246 Text feature [150] present in test data point [True]
263 Text feature [release] present in test data point [True]
268 Text feature [plates] present in test data point [True]
273 Text feature [separated] present in test data point [True]
295 Text feature [suggesting] present in test data point [True]
299 Text feature [condition] present in test data point [True]
300 Text feature [mutants] present in test data point [True]
Out of the top  500  features  16 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [94]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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


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

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

```
        sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=
clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

'''fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ra
vel()
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)), (featur
es[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cri
terion='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.cl
asses_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The
 test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, ep
s=1e-15))
```

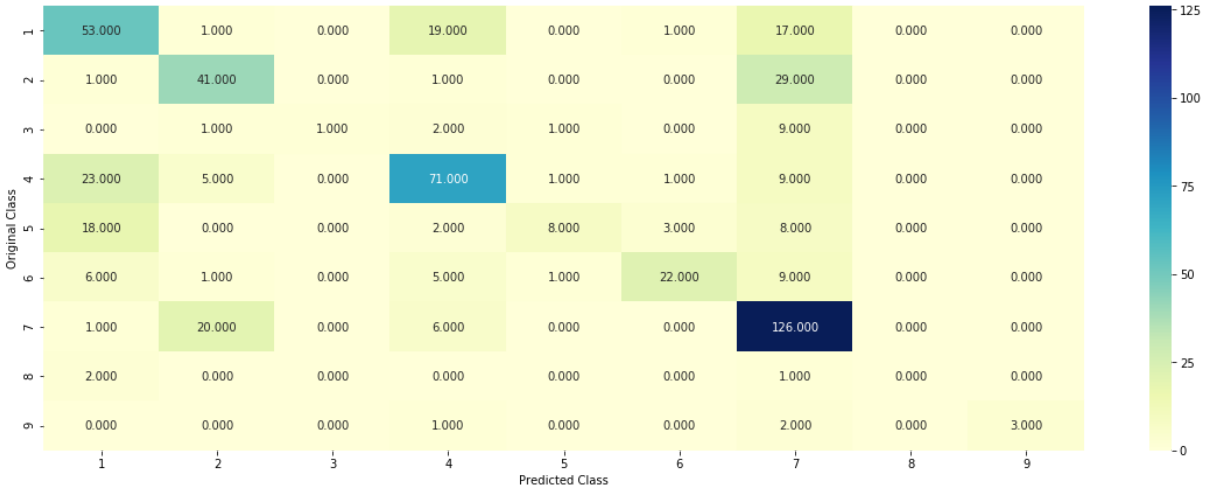
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.1638136550317513
for n_estimators = 100 and max depth =  10
Log Loss : 1.1588982780405586
for n_estimators = 200 and max depth =  5
Log Loss : 1.1350360563865103
for n_estimators = 200 and max depth =  10
Log Loss : 1.1462566371986884
for n_estimators = 500 and max depth =  5
Log Loss : 1.1350620477019249
for n_estimators = 500 and max depth =  10
Log Loss : 1.136569998697557
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1330286201665203
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1361320261456445
for n_estimators = 2000 and max depth =  5
Log Loss : 1.136093734751251
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1371271615661167
For values of best estimator =  1000 The train log loss is: 0.888260569
2004013
For values of best estimator =  1000 The cross validation log loss is:
1.1330286201665203
For values of best estimator =  1000 The test log loss is: 1.2194991792
774517
```

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

In [95]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)
```

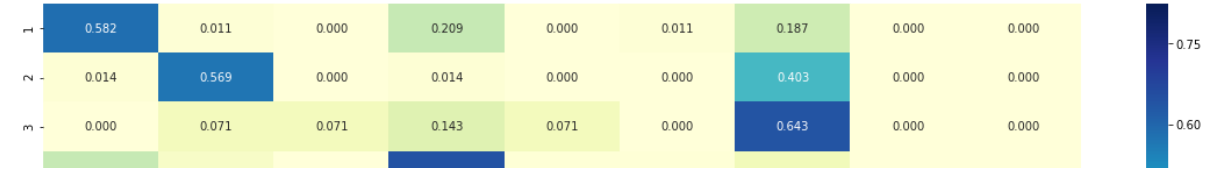
```
# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the give
n training data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X)     Perform classification on samples in X.

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

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

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

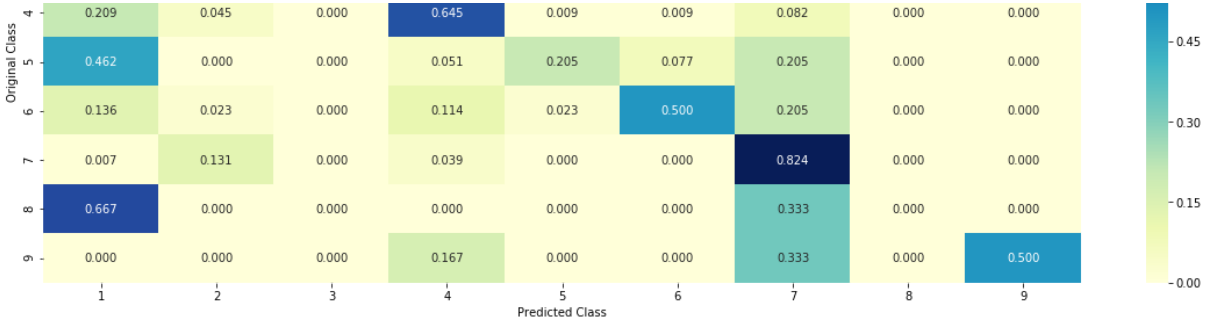
```
Log loss : 1.1330286201665203
Number of mis-classified points : 0.3890977443609023
-------------------- Confusion matrix --------------------
```



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

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





4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [96]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cri
terion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,
 n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
Predicted Class : 6
```

```
Predicted Class Probabilities: [[1.480e-02 3.500e-03 5.500e-03 7.500e-0
3 1.044e-01 8.549e-01 6.800e-03
  1.800e-03 7.000e-04]]
Actual Class : 6
--------------------------------------------------
4 Text feature [activation] present in test data point [True]
14 Text feature [function] present in test data point [True]
17 Text feature [missense] present in test data point [True]
20 Text feature [loss] present in test data point [True]
23 Text feature [brca1] present in test data point [True]
31 Text feature [functional] present in test data point [True]
35 Text feature [neutral] present in test data point [True]
36 Text feature [variants] present in test data point [True]
37 Text feature [therapeutic] present in test data point [True]
38 Text feature [cells] present in test data point [True]
39 Text feature [yeast] present in test data point [True]
49 Text feature [brca] present in test data point [True]
50 Text feature [potential] present in test data point [True]
52 Text feature [deleterious] present in test data point [True]
53 Text feature [predicted] present in test data point [True]
56 Text feature [pathogenic] present in test data point [True]
57 Text feature [classified] present in test data point [True]
58 Text feature [repair] present in test data point [True]
60 Text feature [expected] present in test data point [True]
62 Text feature [expression] present in test data point [True]
70 Text feature [likelihood] present in test data point [True]
71 Text feature [unclassified] present in test data point [True]
73 Text feature [ovarian] present in test data point [True]
74 Text feature [clinical] present in test data point [True]
75 Text feature [57] present in test data point [True]
77 Text feature [response] present in test data point [True]
79 Text feature [trans] present in test data point [True]
82 Text feature [variant] present in test data point [True]
83 Text feature [sensitivity] present in test data point [True]
85 Text feature [patients] present in test data point [True]
86 Text feature [ring] present in test data point [True]
88 Text feature [protein] present in test data point [True]
89 Text feature [predictive] present in test data point [True]
91 Text feature [brca2] present in test data point [True]
```

```
93 Text feature [dose] present in test data point [True]
97 Text feature [classify] present in test data point [True]
Out of the top  100  features  36 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [97]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_po
int_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].
iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0775 0.1198 0.0305 0.2895 0.0613 0.0
55  0.3512 0.0078 0.0074]]
Actuall Class : 4
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [phosphorylation] present in test data point [True]
3 Text feature [activating] present in test data point [True]
4 Text feature [activation] present in test data point [True]
6 Text feature [inhibitors] present in test data point [True]
7 Text feature [inhibitor] present in test data point [True]
8 Text feature [oncogenic] present in test data point [True]
10 Text feature [growth] present in test data point [True]
13 Text feature [constitutive] present in test data point [True]
14 Text feature [function] present in test data point [True]
19 Text feature [signaling] present in test data point [True]
21 Text feature [transforming] present in test data point [True]
24 Text feature [inhibition] present in test data point [True]
26 Text feature [downstream] present in test data point [True]
```

```
28 Text feature [akt] present in test data point [True]
29 Text feature [proliferation] present in test data point [True]
30 Text feature [treated] present in test data point [True]
33 Text feature [activate] present in test data point [True]
38 Text feature [cells] present in test data point [True]
44 Text feature [inhibited] present in test data point [True]
48 Text feature [cell] present in test data point [True]
50 Text feature [potential] present in test data point [True]
54 Text feature [phospho] present in test data point [True]
60 Text feature [expected] present in test data point [True]
62 Text feature [expression] present in test data point [True]
64 Text feature [expressing] present in test data point [True]
81 Text feature [oncogene] present in test data point [True]
83 Text feature [sensitivity] present in test data point [True]
88 Text feature [protein] present in test data point [True]
96 Text feature [membranes] present in test data point [True]
98 Text feature [serum] present in test data point [True]
Out of the top  100  features  31 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [98]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

# some of attributes of  RandomForestClassifier()
```

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

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


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

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

```
clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ra
vel()
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)), (featur
es[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], cri
terion='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 tra
in 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 cro
ss validation log loss is:",log_loss(y_cv, predict_y, labels=clf.classe
s_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The tes
t log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e
-15))
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.0647551502528096
```

```
for n_estimators = 10 and max depth =  3
Log Loss : 1.6042971067429963
for n_estimators = 10 and max depth =  5
Log Loss : 1.3697585221083577
for n_estimators = 10 and max depth =  10
Log Loss : 1.7094561296408668
for n_estimators = 50 and max depth =  2
Log Loss : 1.5512542732109436
for n_estimators = 50 and max depth =  3
Log Loss : 1.3210606025357186
for n_estimators = 50 and max depth =  5
Log Loss : 1.2183727254746386
for n_estimators = 50 and max depth =  10
Log Loss : 1.817525791069682
for n_estimators = 100 and max depth =  2
Log Loss : 1.439060893505722
for n_estimators = 100 and max depth =  3
Log Loss : 1.3411214175464816
for n_estimators = 100 and max depth =  5
Log Loss : 1.2033987890247009
for n_estimators = 100 and max depth =  10
Log Loss : 1.7389315533064276
for n_estimators = 200 and max depth =  2
Log Loss : 1.4815925589760153
for n_estimators = 200 and max depth =  3
Log Loss : 1.3415931840220625
for n_estimators = 200 and max depth =  5
Log Loss : 1.2637451002109208
for n_estimators = 200 and max depth =  10
Log Loss : 1.6802875430867756
for n_estimators = 500 and max depth =  2
Log Loss : 1.535831271662667
for n_estimators = 500 and max depth =  3
Log Loss : 1.3937118794600531
for n_estimators = 500 and max depth =  5
Log Loss : 1.248120550345509
for n_estimators = 500 and max depth =  10
Log Loss : 1.6887665893247819
for n_estimators = 1000 and max depth =  2
```

```
Log Loss : 1.5134001124720797
for n_estimators = 1000 and max depth =  3
Log Loss : 1.404192121695203
for n_estimators = 1000 and max depth =  5
Log Loss : 1.267291384484738
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6633493141904778
For values of best alpha =  100 The train log loss is: 0.05975624873871
1826
For values of best alpha =  100 The cross validation log loss is: 1.203
3987890247009
For values of best alpha =  100 The test log loss is: 1.262111718608465
```

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

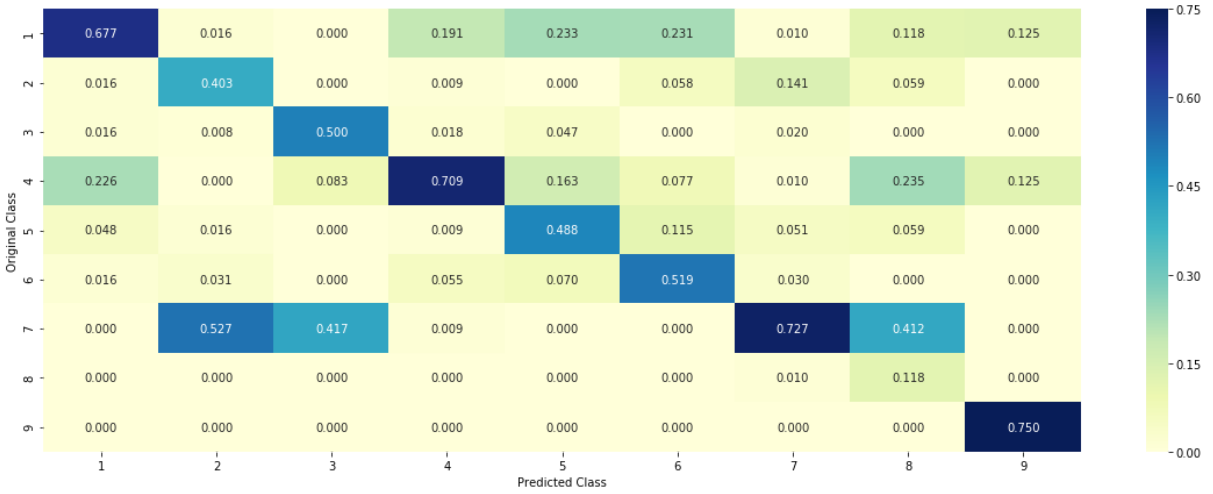
In [99]:

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

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

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

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

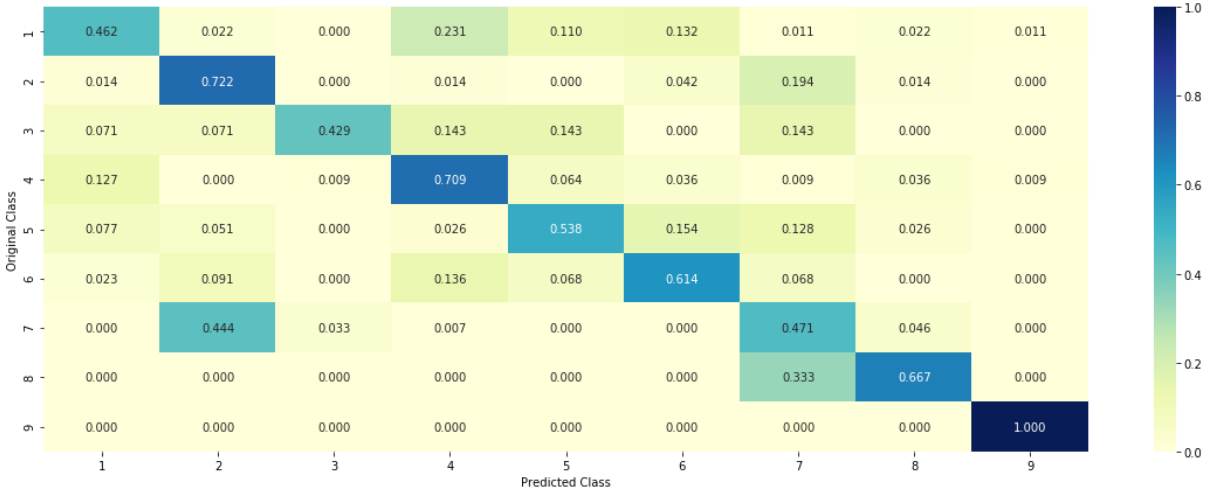
```
# --------------------------------

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_
estimators=alpha[int(best_alpha/4)], criterion='gini', max_features='au
to',random_state=42)
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_
responseCoding,cv_y, clf)
```

```
Log loss : 1.2033987890247009
Number of mis-classified points : 0.424812030075188
-------------------- 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 [100]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], cri
terion='gini', max_depth=max_depth[int(best_alpha%4)], random_state=42,
 n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)


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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0141 0.0026 0.0152 0.0085 0.1547 0.7
922 0.0025 0.0045 0.0055]]
Actual Class : 6
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
```

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

4.5.5.2. Incorrectly Classified point

In [101]:
```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index]
.reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2008 0.048  0.0884 0.4757 0.0333 0.0
466 0.0254 0.0379 0.0439]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene 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
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [102]:
```
# 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/
#-------------------------------


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

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

```
# read more about support vector machines with linear kernals here htt
p://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomFo
restClassifier.html
# ---------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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


clf1 = SGDClassifier(alpha=0.0001, penalty='l2', loss='log', class_weig
ht='balanced', random_state=0)
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

clf2 = SGDClassifier(alpha=0.0001, penalty='l2', loss='hinge', class_we
ight='balanced', random_state=0)
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")
```

```
clf3 = MultinomialNB(alpha=0.01)
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_cl
f1.predict_proba(cv_x_onehotCoding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig
_clf2.predict_proba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predic
t_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: 0.95
Support vector machines : Log Loss: 1.01
Naive Bayes : Log Loss: 1.16
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.172
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.979
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.340
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.001


Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.186
```

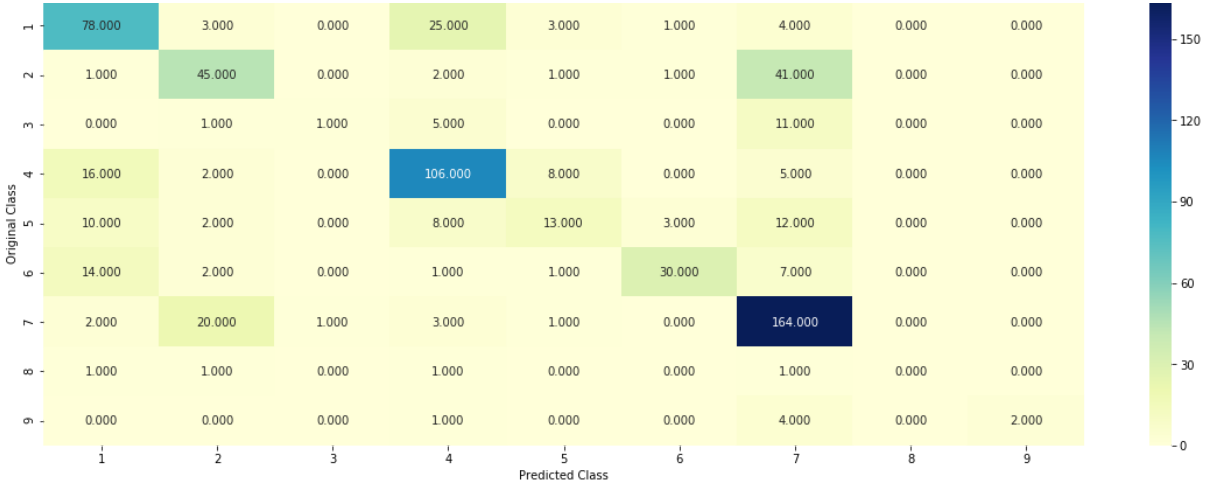
```
Stacking Classifier : for the value of alpha: 10.000000 Log Loss: 1.700
```

4.7.2 testing the model with the best hyper parameters

In [103]:

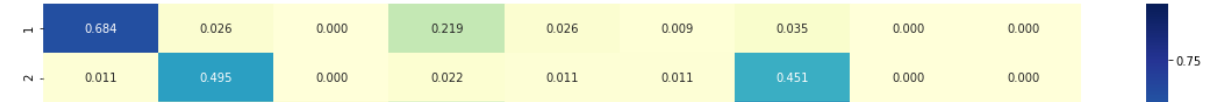
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], m
eta_classifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

```
Log loss (train) on the stacking classifier : 0.39954483365368004
Log loss (CV) on the stacking classifier : 1.0014102361019621
Log loss (test) on the stacking classifier : 1.0943442258741323
Number of missclassified point : 0.3398496240601504
-------------------- Confusion matrix --------------------
```



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

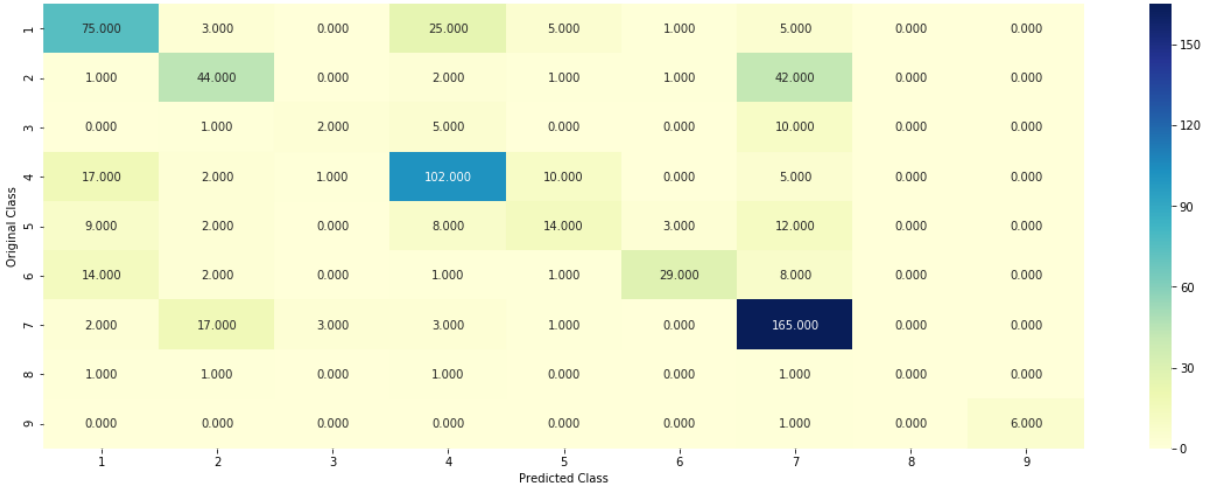
---------- Recall matrix (Row sum=1) ----------



4.7.3 Maximum Voting classifier

In [104]:
```
#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 : 0.537336950947309
Log loss (CV) on the VotingClassifier : 0.9433191862755268
Log loss (test) on the VotingClassifier : 1.0041632256227655
Number of missclassified point : 0.34285714285714286
-------------------- Confusion matrix --------------------
```



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

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

Compare all the models using preety table

In [107]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model", "Best_HyperParameter", "Vectorizer", "Train_log_loss","Cv_log_loss","Test_log_loss","% of Miss clf"]
x.add_row(["NaiveBayes","0.01","TFIDF","0.73","1.15","1.23","0.37"])
x.add_row(["KNN","9","TFIDF","1.01","1.09","1.14","0.38"])
x.add_row(["Logistic SGD with C.W","0.0001","TFIDF","0.44","0.92","0.96","0.29"])
x.add_row(["Logistic SGD BOW","10","BOW","1.00","1.19","1.24","0.38"])
x.add_row(["Logistic SGD with-out C.W","0.0001","TFIDF","0.43","0.98","0.99","0.29"])
x.add_row(["SVM SGD","0.0001","TFIDF","0.54","0.97","1.04","0.29"])
x.add_row(["RandomForest","1000","TFIDF","0.88","1.13","1.21","0.38"])
x.add_row(["RandomForest","100","RC","0.05","1.20","1.26","0.42"])
x.add_row(["Stacking Clf","0.1","TFIDF","0.39","1.00","1.09","0.33"])
x.add_row(["Voting Clf","0.1","TFIDF","0.53","0.94","1.00","0.34"])
print(x)
```

```
+--------------------------+--------------------+------------+-------
---------+-------------+---------------+---------------+
```

```
|          Model           | Best_HyperParameter | Vectorizer | Train_
log_loss | Cv_log_loss | Test_log_loss | % of Miss clf |
+--------------------------+---------------------+------------+-------
---------+-------------+---------------+---------------+
|         NaiveBayes       |         0.01        |   TFIDF    |
0.73      |     1.15    |      1.23     |      0.37     |
|            KNN           |          9          |   TFIDF    |
1.01      |     1.09    |      1.14     |      0.38     |
|   Logistic SGD with C.W  |        0.0001       |   TFIDF    |
0.44      |     0.92    |      0.96     |      0.29     |
|      Logistic SGD BOW    |          10         |    BOW     |
1.00      |     1.19    |      1.24     |      0.38     |
| Logistic SGD with-out C.W |        0.0001       |   TFIDF    |
0.43      |     0.98    |      0.99     |      0.29     |
|          SVM SGD         |        0.0001       |   TFIDF    |
0.54      |     0.97    |      1.04     |      0.29     |
|        RandomForest      |         1000        |   TFIDF    |
0.88      |     1.13    |      1.21     |      0.38     |
|        RandomForest      |         100         |     RC     |
0.05      |     1.20    |      1.26     |      0.42     |
|        Stacking Clf      |         0.1         |   TFIDF    |
0.39      |     1.00    |      1.09     |      0.33     |
|         Voting Clf       |         0.1         |   TFIDF    |
0.53      |     0.94    |      1.00     |      0.34     |
+--------------------------+---------------------+------------+-------
---------+-------------+---------------+---------------+
```

OBSERVATIONS

1. In this case study we deal with Pernolized cancer diagnosis Data ,We took only Train Data from Original Dataset and we applied some Machine Learning models
2. We got Two Files from Train data and we merged the Data and make it as Whole data
3. We Done Text pre Processing on TEXT feature and we visualize the Y Class labels 3,9,8 are gives may lesser results in models becasue those are very few points in dataset
4. We done Feature Extraction lenght and No of words in Gene and variation

5. We splitted the data into train cv and test and applied vectorizers Bow and TFIDF
6. We apply Machine Learning Models Naive Bayes,Knn,SGD with log_loss and Hinge_loss ,SGD without Class weights,Random Forest,stacking Classifier with TFIDF vectorizer
7. We got a Good results in SGD with log_loss with and with out class weights
8. All of the Y class labels class label 8 will has lesser time occur in data ,So models may or may not get the information about class 8 in confusion matrix results and precision and recall
9. Feature importance is also done which will gives which are important features by correctly classified and in correctly classified points
10. Finally Compare to all the models SGD with logg_loss with TFIDF vectoirzer gives best results compare to rest models