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

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.

- 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 [41]:

```
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
# Visualization Libraries
import matplotlib.pyplot as plt
from matplotlib.patches import Patch
from matplotlib.markers import MarkerStyle
import seaborn as sns

# Text analysis helper libraries
from gensim.summarization import summarize
from gensim.summarization import keywords

# Text analysis helper libraries for word frequency etc..
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from string import punctuation

# Word cloud visualization libraries
from scipy.misc import imresize
from PIL import Image
from wordcloud import WordCloud, ImageColorGenerator
from collections import Counter
# Word2Vec related libraries
from gensim.models import KeyedVectors

# Dimensionaly reduction libraries
```

```
# Dimensionality reduction libraries
from sklearn.decomposition import PCA

# Clustering library
from sklearn.cluster import KMeans

# Set figure size a bit bigger than default so everything is easily red
plt.rcParams["figure.figsize"] = (11, 7)
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [23]:

```
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()
```

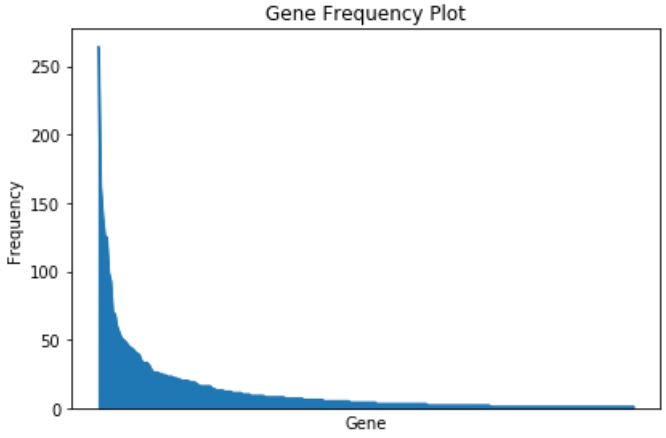
```
Number of data points :  3321
Number of features :  4
Features :  ['ID' 'Gene' 'Variation' 'Class']
```

Out[23]:

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

In [32]:

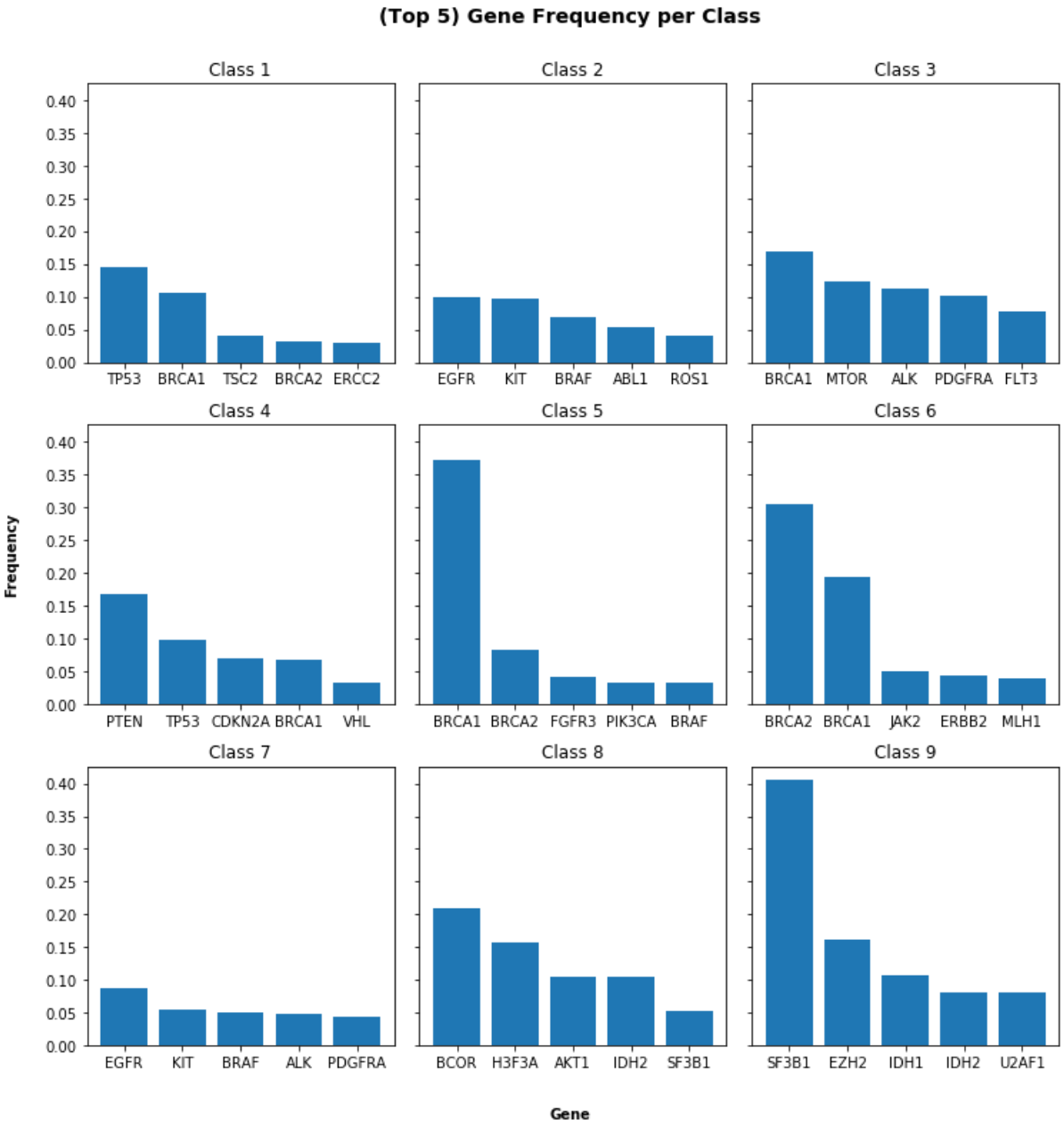
```
plt.figure()
ax = data['Gene'].value_counts().plot(kind='area')

ax.get_xaxis().set_ticks([])
ax.set_title('Gene Frequency Plot')
ax.set_xlabel('Gene')
ax.set_ylabel('Frequency')

plt.tight_layout()
plt.show()
```



Even with domination of some gene's, it still gives a nice insight from their class distributions.

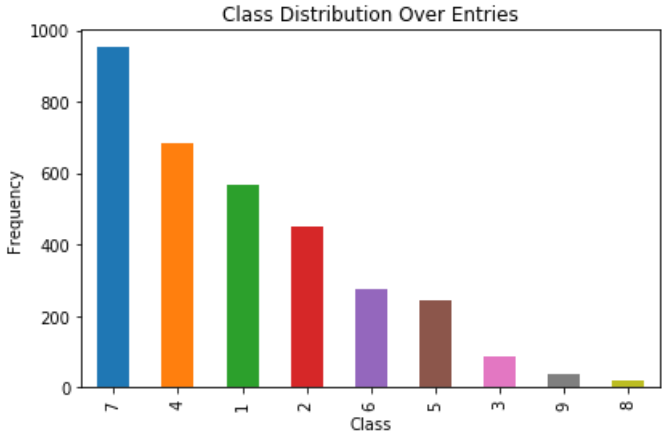
In [34]:

```
fig, axes = plt.subplots(nrows=3, ncols=3, sharey=True, figsize=(11,11))

# Normalize value counts for better comparison
def normalize_group(x):
    label, repetition = x.index, x
    t = sum(repetition)
    r = [n/t for n in repetition]
    return label, r

for idx, g in enumerate(data.groupby('Class')):
    label, val = normalize_group(g[1]["Gene"].value_counts())
    ax = axes.flat[idx]
    ax.bar(np.arange(5), val[:5],
           tick_label=label[:5])
    ax.set_title("Class {}".format(g[0]))

fig.text(0.5, 0.97, '(Top 5) Gene Frequency per Class', ha='center', fontsize=14, fontweight='bold'
)
fig.text(0.5, 0, 'Gene', ha='center', fontweight='bold')
fig.text(0, 0.5, 'Frequency', va='center', rotation='vertical', fontweight='bold')
fig.tight_layout(rect=[0.03, 0.03, 0.95, 0.95])
```



In [35]:

```
plt.figure()
ax = data['Class'].value_counts().plot(kind='bar')

ax.set_title('Class Distribution Over Entries')
ax.set_xlabel('Class')
ax.set_ylabel('Frequency')

plt.tight_layout()
plt.show()
```



Distribution looks skewed towards some classes, there are not enough examples for classes 8 and 9. During training, this can be solved using bias weights, careful sampling in batches or simply removing some of the dominant data to equalize the field.

In [24]:

```
data['Gene_And_Variation']=data['Gene']+' '+data['Variation']
data.head()
```

Out[24]:

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

In [25]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3321 entries, 0 to 3320
Data columns (total 5 columns):
ID                  3321 non-null int64
Gene                3321 non-null object
Variation           3321 non-null object
Class               3321 non-null int64
Gene_And_Variation  3321 non-null object
dtypes: int64(2), object(3)
memory usage: 129.8+ KB
```

Checking number of unique values for gene and variation

In [26]:

```
data['Gene_And_Variation'].value_counts().head()
```

Out[26]:

```
IDH1 R132C      1
SPOP P94A       1
ALK G1269S      1
CDKN2A R79P     1
BRCA2 L2721H    1
Name: Gene_And_Variation, dtype: int64
```

In [27]:

```
data['Gene_And_Variation'].value_counts().max()
```

Out[27]:

```
1
```

From above we can say that combination of Gene and variation are unique in this dataset

Introduce feature 'IsFusion' to tag all entries with variation of 'Gene1-Gene2 Fusion'

In [28]:

```
import numpy as np

def IsFusion(s):
    n = np.array(s.split()).size
    lastWord=s.split()[n-1]
    if (lastWord.lower()=='fusion'): return True
    return False

data['IsFusion']=data['Variation'].apply(lambda s: IsFusion(s))
data[data['IsFusion']==1].head()
```

Out[28]:

	ID	Gene	Variation	Class	Gene_And_Variation	IsFusion
164	164	EGFR	EGFR-RAD51 Fusion	7	EGFR EGFR-RAD51 Fusion	True
268	268	EGFR	EGFR-PURB Fusion	2	EGFR EGFR-PURB Fusion	True
279	279	NKX2-1	IGH-NKX2 Fusion	2	NKX2-1 IGH-NKX2 Fusion	True
280	280	NKX2-1	TRB-NKX2-1 Fusion	2	NKX2-1 TRB-NKX2-1 Fusion	True
283	283	NKX2-1	TRA-NKX2-1 Fusion	2	NKX2-1 TRA-NKX2-1 Fusion	True

In the variations of the pattern 'Gene1-Gene2 Fusion' one of the gene is always the value from the 'Gene' column of the same row, or almost this value (279: NKX2-1 vs NKX2)

Is the second gene with the same fusion also present in the dataset? Not necessarily. But sometimes yes. It is found that several fused 'Gene1-Gene2' pairs that appear twice in the training dataset.

Here are three examples:

In [29]:

```
data[data['Variation']=='TMPRSS2-ETV1 Fusion']
```

Out[29]:

	ID	Gene	Variation	Class	Gene_And_Variation	IsFusion
300	300	TMPRSS2	TMPRSS2-ETV1 Fusion	7	TMPRSS2 TMPRSS2-ETV1 Fusion	True
978	978	ETV1	TMPRSS2-ETV1 Fusion	7	ETV1 TMPRSS2-ETV1 Fusion	True

In [30]:

```
data[data['Variation']=='EWSR1-ETV1 Fusion']
```

Out[30]:

	ID	Gene	Variation	Class	Gene_And_Variation	IsFusion
980	980	ETV1	EWSR1-ETV1 Fusion	7	ETV1 EWSR1-ETV1 Fusion	True
1060	1060	EWSR1	EWSR1-ETV1 Fusion	7	EWSR1 EWSR1-ETV1 Fusion	True

In [31]:

```
data[data['Variation']=='ETV6-NTRK3 Fusion']
```

Out[31]:

	ID	Gene	Variation	Class	Gene_And_Variation	IsFusion
988	988	ETV6	ETV6-NTRK3 Fusion	7	ETV6 ETV6-NTRK3 Fusion	True
3234	3234	NTRK3	ETV6-NTRK3 Fusion	7	NTRK3 ETV6-NTRK3 Fusion	True

In [3]:

```
data.shape
```

Out[3]:

```
(3321, 4)
```

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

In [4]:

```
# check nan value
data.isnull().sum()
```

Out[4]:

```
ID           0
Gene         0
Variation    0
Class        0
dtype: int64
```

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...

In [76]:

```
data_text = data_text[data_text['TEXT'].notnull()]
```

In [77]:

```
data_text.loc[:, 'Text_count']  = data_text["TEXT"].apply(lambda x: len(x.split()))
data_text.head()
```

Out[77]:

	ID	TEXT	Text_count
0	0	Cyclin-dependent kinases (CDKs) regulate a var...	6089
1	1	Abstract Background Non-small cell lung canc...	5722
2	2	Abstract Background Non-small cell lung canc...	5722
3	3	Recent evidence has demonstrated that acquired...	5572
4	4	Oncogenic mutations in the monomeric Casitas B...	6202

In [78]:

```
train_full = data.merge(data_text, how="inner", left_on="ID", right_on="ID")
train_full[train_full["Class"]==1].head()
```

Out[78]:

	ID	Gene	Variation	Class	Gene_And_Variation	IsFusion	TEXT	Text_count
0	0	FAM58A	Truncating Mutations	1	FAM58A Truncating Mutations	False	Cyclin-dependent kinases (CDKs) regulate a var...	6089
7	7	CBL	Deletion	1	CBL Deletion	False	CBL is a negative regulator of activated recep...	14684
16	16	CBL	Truncating Mutations	1	CBL Truncating Mutations	False	To determine if residual cylindrical refractiv...	8118
37	37	DICER1	D1709E	1	DICER1 D1709E	False	Sex cordâ€“stromal tumors and germ-cell tumors...	2710
38	38	DICER1	D1709A	1	DICER1 D1709A	False	Sex cordâ€“stromal tumors and germ-cell tumors...	2710

In [79]:

```
count_grp = train_full.groupby('Class')["Text_count"]
count_grp.describe()
```

Out[79]:

	count	mean	std	min	25%	50%	75%	max
Class								
1	566.0	9478.075972	6500.833412	183.0	4974.25	7305.0	12930.75	52970.0
2	452.0	9306.362832	7624.322787	116.0	4184.25	6810.0	12209.50	61923.0
3	89.0	6751.157303	3724.432760	1737.0	4283.00	5572.0	7415.00	27371.0
4	686.0	8978.202624	7276.259637	53.0	4566.00	6351.0	11521.75	43893.0
5	242.0	7504.462810	3890.263510	183.0	5245.00	6451.0	9513.50	24214.0
6	273.0	7195.391941	3792.535663	230.0	4688.00	6587.0	7626.00	24597.0
7	952.0	11449.925420	10103.702739	448.0	4876.25	8254.0	14647.25	76708.0
8	19.0	10810.105263	5645.073662	2111.0	5586.00	11237.0	15535.00	20612.0
9	37.0	12798.567568	10208.668344	1146.0	4942.00	10917.0	15797.00	45126.0

In [80]:

```
train_full[train_full["Text_count"]==1.0]
```

Out[80]:

	ID	Gene	Variation	Class	Gene_And_Variation	IsFusion	TEXT	Text_count

In [81]:

```
train_full[train_full["Text_count"]<500.0]
```

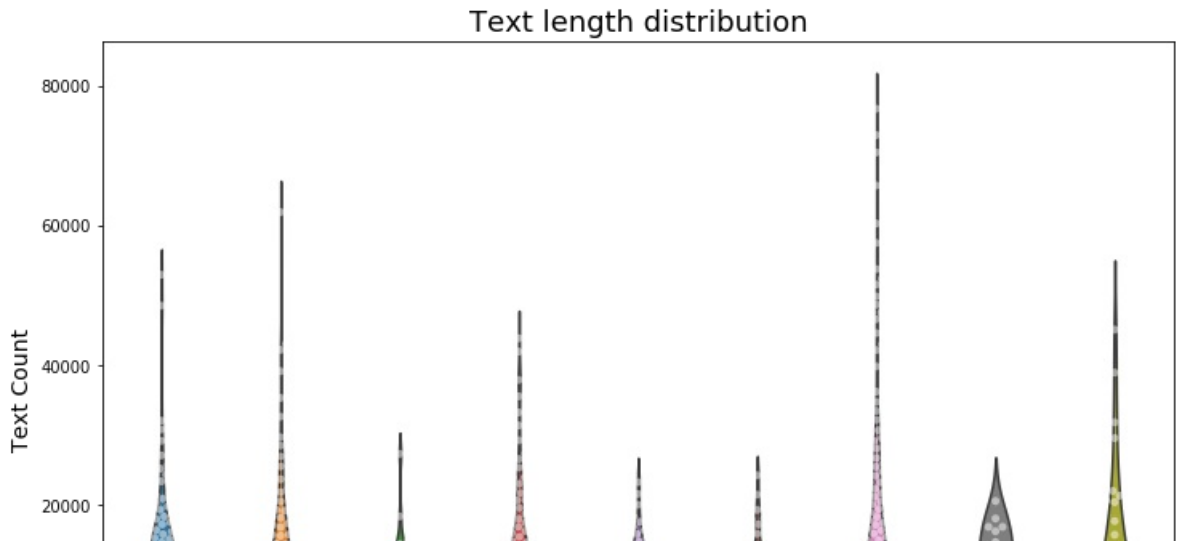
Out[81]:

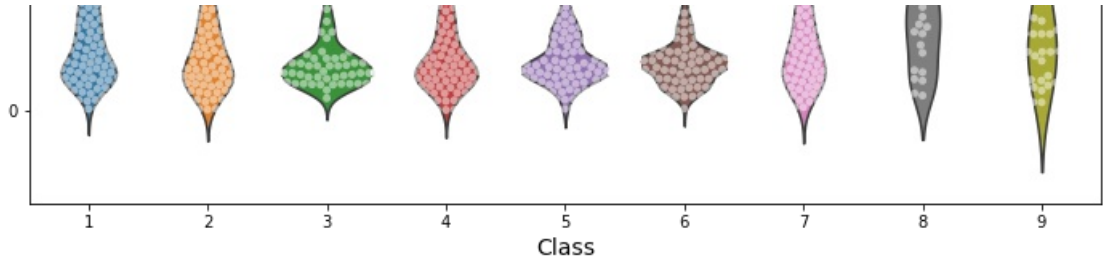
	ID	Gene	Variation	Class	Gene_And_Variation	IsFusion	TEXT	Text_count
140	140	EGFR	I491M	5	EGFR I491M	False	The accurate determination of perfluoroalkyl s...	430
145	145	EGFR	K467T	2	EGFR K467T	False	The accurate determination of perfluoroalkyl s...	430
259	259	EGFR	S464L	2	EGFR S464L	False	The accurate determination of perfluoroalkyl s...	430
344	344	CDH1	A617T	4	CDH1 A617T	False	E-cadherin is involved in the formation of cel...	187
346	346	CDH1	A634V	4	CDH1 A634V	False	E-cadherin is involved in the formation of cel...	187
348	348	CDH1	T340A	4	CDH1 T340A	False	E-cadherin is involved in the formation of cel...	187
648	648	CDKN2A	Q50*	4	CDKN2A Q50*	False	The p16 gene is located in chromosome 9p21, a ...	103
688	688	CDKN2A	R79P	4	CDKN2A R79P	False	Cell division is controlled by a series of pos...	228
692	692	CDKN2A	G93W	4	CDKN2A G93W	False	Cell division is controlled by a series of pos...	228
693	693	CDKN2A	V118D	4	CDKN2A V118D	False	Cell division is controlled by a series of pos...	228
868	868	HLA-A	596_619splice	1	HLA-A 596_619splice	False	A new variant of the HLA-A*010101 allele desig...	184
941	941	PDGFRB	ATF7IP-PDGFRB Fusion	2	PDGFRB ATF7IP-PDGFRB Fusion	True	Chronic myelomonocytic leukemia (CMML) is a my...	116

	ID	Gene	Variation	Class	Gene_And_Variation	IsFusion	TEXT	Text_count
1057	1057	EWSR1	EWSR1-FLI1 Fusion	2	EWSR1-EWSR1-FLI1 Fusion	True	As a result of chromosome translocations, the ...	207
1153	1154	KMT2C	S3660L	5	KMT2C S3660L	False	Several studies indicated that the expression ...	341
1284	1286	HRAS	A146V	2	HRAS A146V	False	Costello syndrome is a rare congenital disorde...	244
1288	1290	HRAS	T58I	2	HRAS T58I	False	We report a 10-year-old girl presenting with s...	231
1366	1368	AKT2	D32H	7	AKT2 D32H	False	The activating E17K mutations recently discove...	448
1376	1378	AKT2	D399N	7	AKT2 D399N	False	The activating E17K mutations recently discove...	448
1580	1583	PMS1	Q233*	4	PMS1 Q233*	False	HEREDITARY nonpolyposis colorectal cancer (HNP...	114
1610	1613	VHL	L158Q	4	VHL L158Q	False	The case of a 40-year-old woman with severe ed...	53
1617	1620	VHL	P25L	5	VHL P25L	False	Background: von Hippel-Lindau (VHL) disease is...	431
2142	2146	PTCH1	Truncating Mutations	1	PTCH1 Truncating Mutations	False	Basal cell carcinoma (BCC) is the most common ...	212
2500	2504	BRCA1	V11A	6	BRCA1 V11A	False	Identification of protein-protein interaction...	230
2895	2900	NF2	E106G	1	NF2 E106G	False	Neurofibromatosis 2 (NF2) is a tumor predispos...	183
2901	2906	NF2	Q538P	1	NF2 Q538P	False	Neurofibromatosis 2 (NF2) is a tumor predispos...	183
2903	2908	NF2	Q324L	5	NF2 Q324L	False	Neurofibromatosis 2 (NF2) is a tumor predispos...	183
2975	2980	KIT	N655K	2	KIT N655K	False	TO THE EDITOR: I commend Drs. Freeman and Bank...	362

In [82]:

```
plt.figure(figsize=(12,8))
gene_count_grp = train_full.groupby('Gene')["Text_count"].sum().reset_index()
sns.violinplot(x="Class", y="Text_count", data=train_full, inner=None)
sns.swarmplot(x="Class", y="Text_count", data=train_full, color="w", alpha=.5);
plt.ylabel('Text Count', fontsize=14)
plt.xlabel('Class', fontsize=14)
plt.title("Text length distribution", fontsize=18)
plt.show()
```

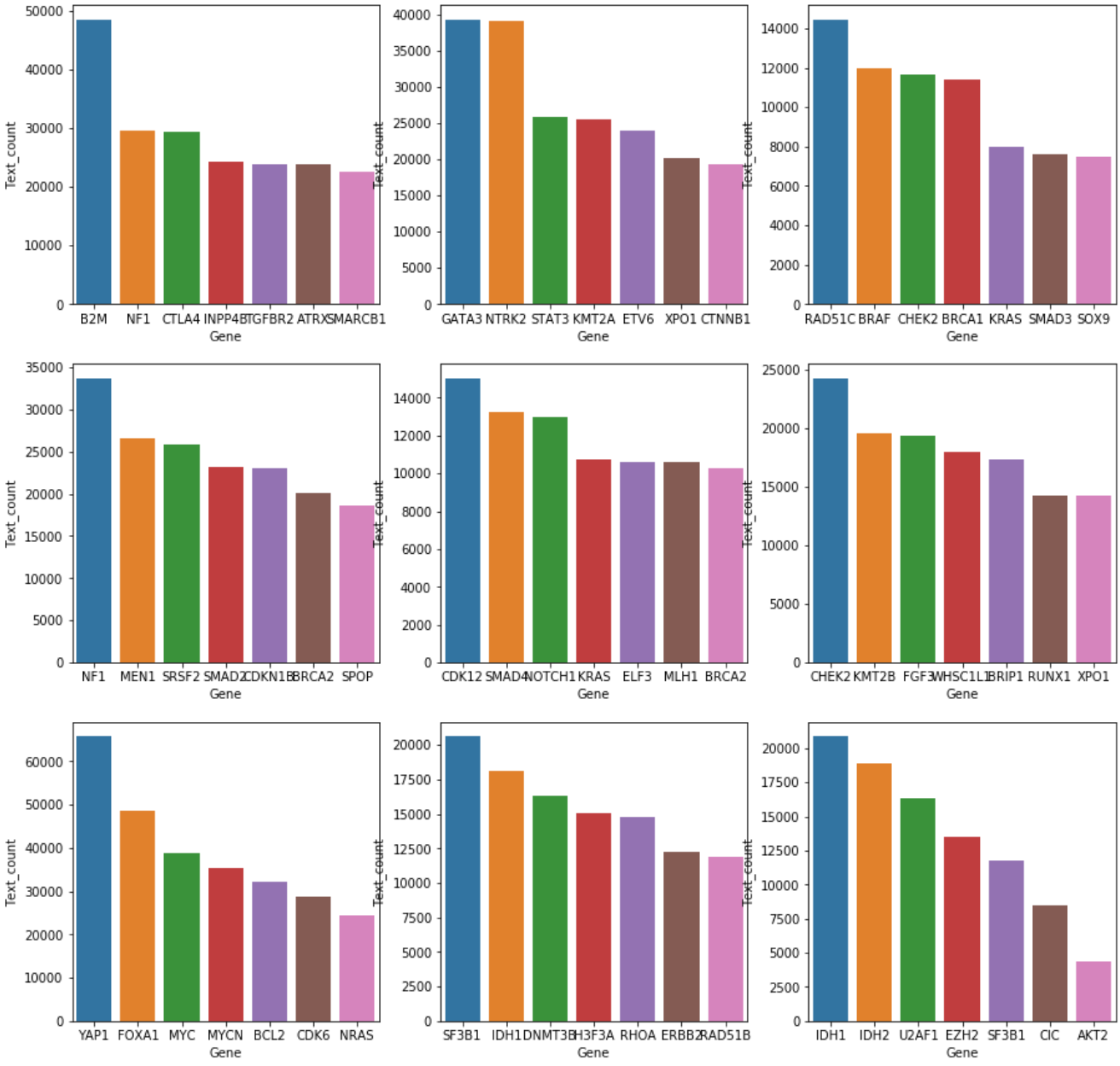




Distribution looks quite interesting and now I am in love with violin plots. All classes have most counts in between 0 to 20000. Just as expected. There should be some

In [83]:

```
ig, axs = plt.subplots(ncols=3, nrows=3, figsize=(15,15))

for i in range(3):
    for j in range(3):
        gene_count_grp = train_full[train_full["Class"]==((i*3+j)+1)].groupby('Gene')["Text_count"]
.mean().reset_index()
        sorted_gene_group = gene_count_grp.sort_values('Text_count', ascending=False)
        sorted_gene_group_top_7 = sorted_gene_group[:7]
        sns.barplot(x="Gene", y="Text_count", data=sorted_gene_group_top_7, ax=axs[i][j])
```



3.1.3. Preprocessing of text

In [7]:

```
# preprocessing for data_text

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

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

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

In [8]:

```
#text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    nlp_preprocessing(row['TEXT'], index, 'TEXT')
print('Time took for preprocessing the text :',time.clock() - start_time, "seconds")
```

```
Time took for preprocessing the text : 211.4403301736817 seconds
```

In [ ]:

```
train_text_df.loc[:, 'Text_count']  = train_text_df["Text"].apply(lambda x: len(x.split()))
train_text_df.head()
```

In [ ]:

```
dfSum['Gene_And_Variation']=dfSum['Gene']+' '+dfSum['Variation']
dfSum.head()
```

In [14]:

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

Out[14]:

	ID	Gene	Variation	Class	TEXT
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...

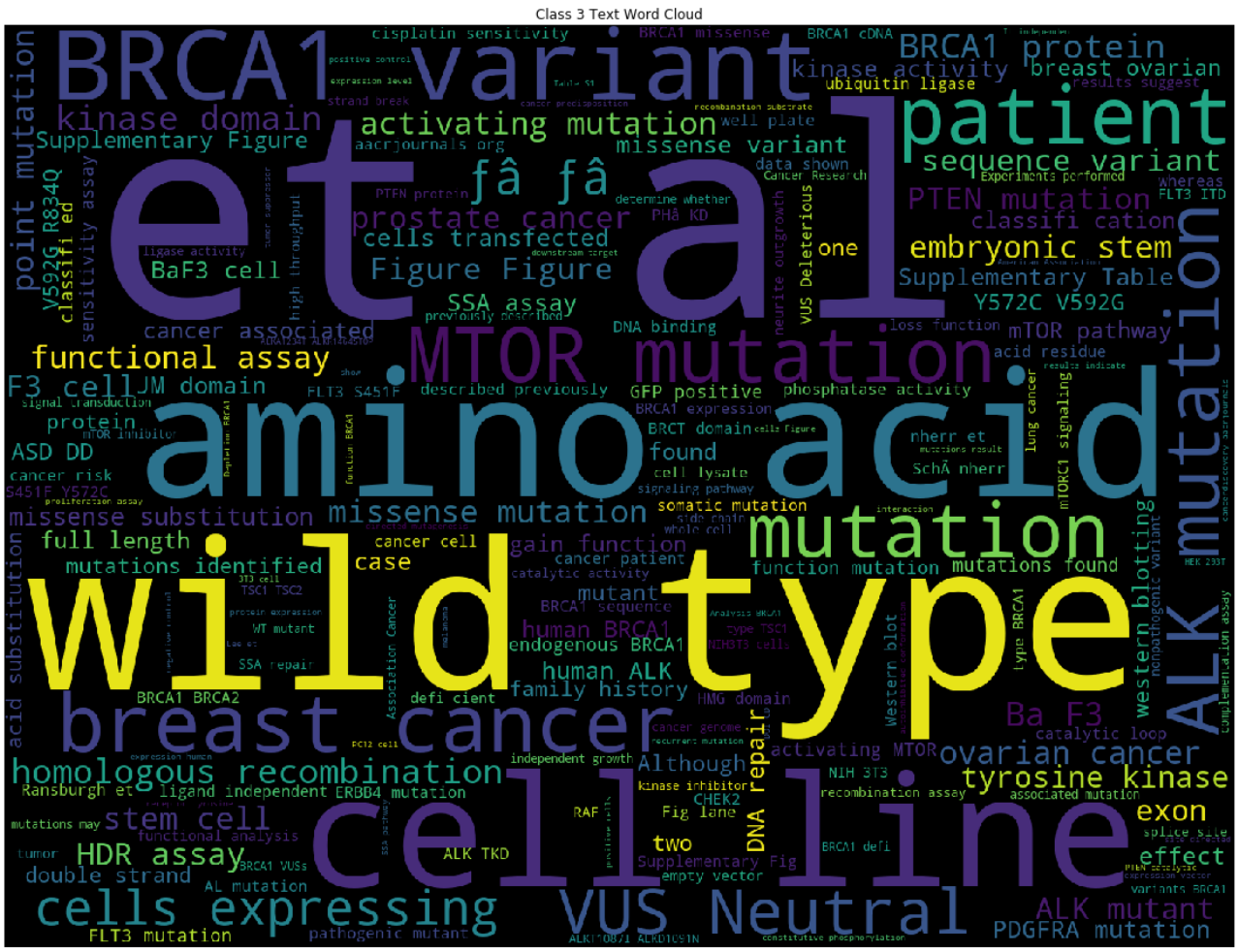
insights and Topic Modelling

In [18]:

```
from wordcloud import WordCloud

cloud = WordCloud(width=1440, height=1080).generate(" ".join(result[result.Class == 3]['TEXT']))
plt.figure(figsize=(20, 15))
plt.imshow(cloud)
plt.axis('off')
ax = plt.axes()
ax.set_title('Class 3 Text Word Cloud')
```

Out[18]:

```
Text(0.5,1,'Class 3 Text Word Cloud')
```

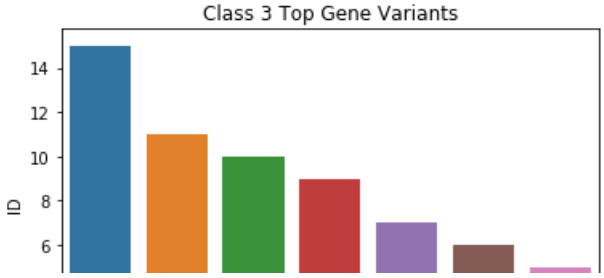


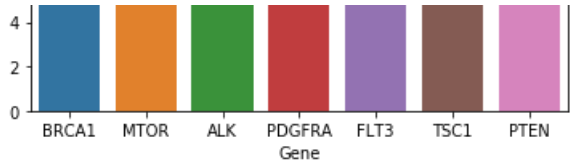
In [20]:

```
data = result[result['Class'] == 3].groupby('Gene')["ID"].count().reset_index()
sns.barplot(x="Gene", y="ID", data=data.sort_values('ID', ascending=False)[:7])
ax = plt.axes()
ax.set_title('Class 3 Top Gene Variants')
```

Out[20]:

```
Text(0.5,1,'Class 3 Top Gene Variants')
```

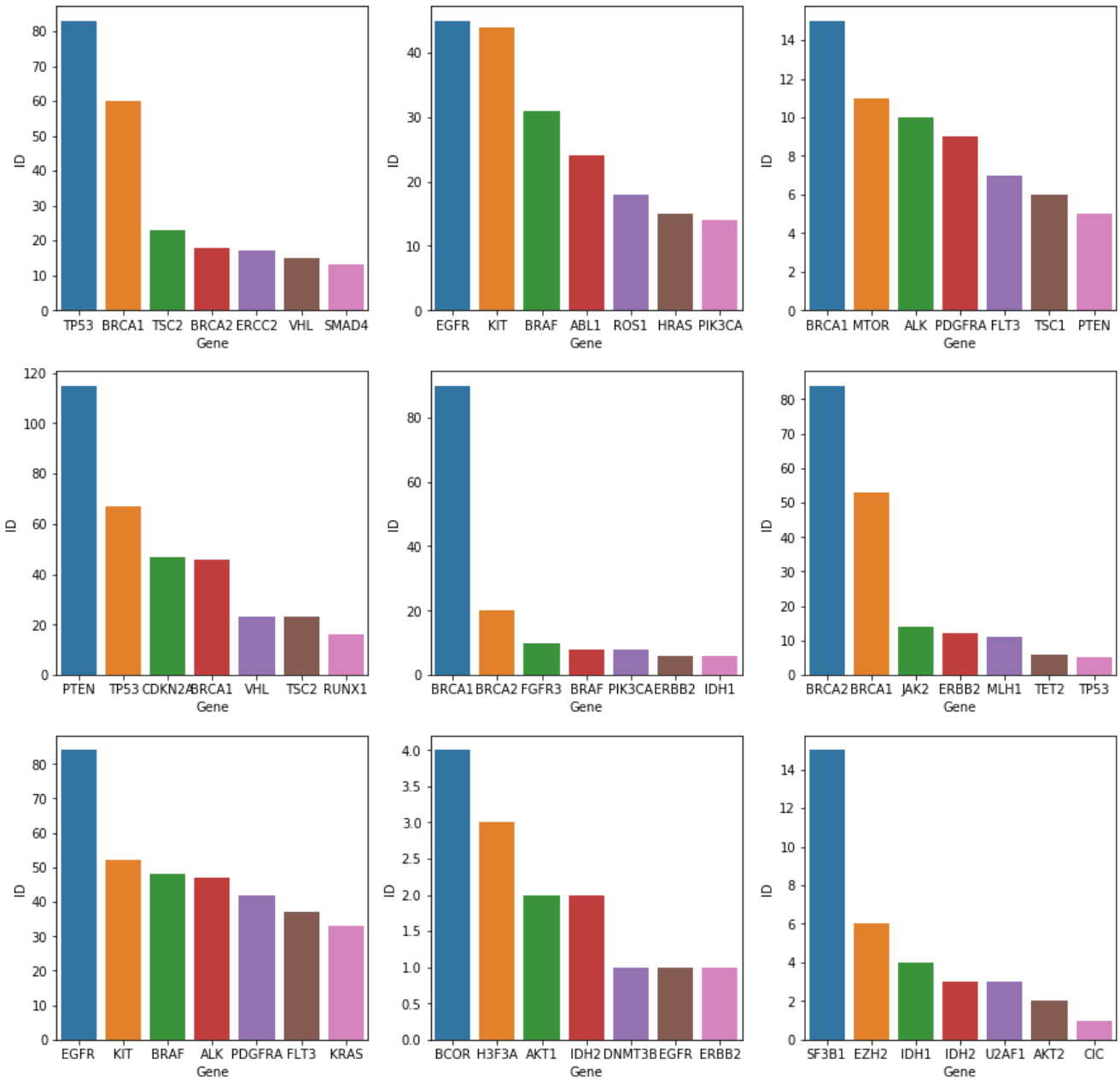




THere are so many keywords that straight up tell us what the class is about. The most important insight of the word cloud, however, is the importance of bigrams in our text. "Amino Acid", "Homologous Recombination", "Breast Cancer" are only a few examples of many. Conclusion drawn here is that is that we'll find terms like these extremely prevalent in technical literature such as our text.

In [72]:

```
fig, axs = plt.subplots(ncols=3, nrows=3, figsize=(15,15))

for i in range(3):
    for j in range(3):
        gene_count_grp = data[data["Class"]==((i*3+j)+1)].groupby('Gene')["ID"].count().reset_index
()
        sorted_gene_group = gene_count_grp.sort_values('ID', ascending=False)
        sorted_gene_group_top_7 = sorted_gene_group[:7]
        sns.barplot(x="Gene", y="ID", data=sorted_gene_group_top_7, ax=axs[i][j])
```



Conclusion

1.BRCA1 is highly dominating Class 5

2.SF3B1 is highly dominating Class 9

3.BRCA1 and BRCA2 are dominating Class 6

In [37]:

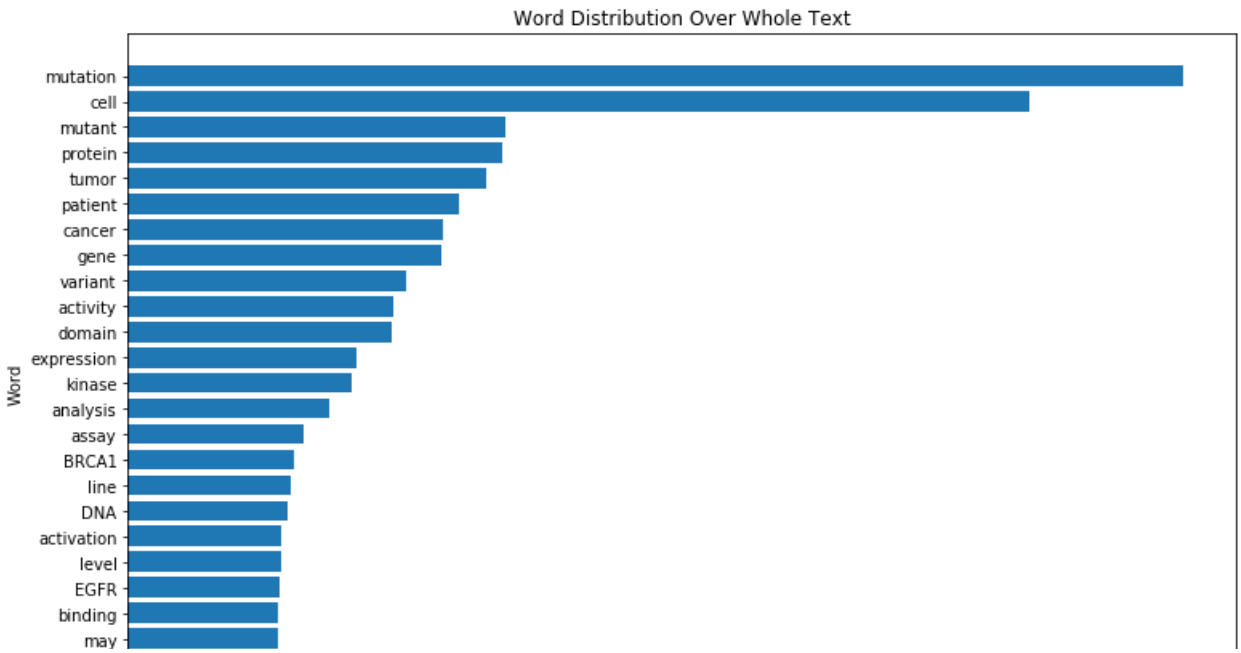
```
result.drop(['Gene', 'Variation'], axis=1, inplace=True)

# Additionaly we will drop the null labeled texts too
result = result[result['TEXT'] != 'null']
```

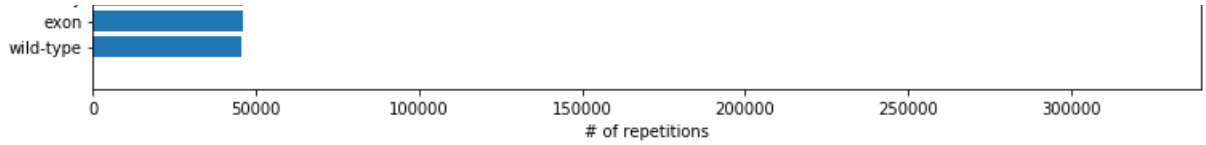
In [44]:

```
custom_words = ["fig", "figure", "et", "al", "al.", "also",
                "data", "analyze", "study", "table", "using",
                "method", "result", "conclusion", "author",
                "find", "found", "show", '"', "’", "‘", "”"]

stop_words = set(stopwords.words('english') + list(punctuation) + custom_words)
wordnet_lemmatizer = WordNetLemmatizer()

class_corpus = result.groupby('Class').apply(lambda x: x['TEXT'].str.cat())
class_corpus = class_corpus.apply(lambda x: Counter(
    [wordnet_lemmatizer.lemmatize(w)
     for w in word_tokenize(x)
     if w.lower() not in stop_words and not w.isdigit()]
))
```

In [57]:

```
whole_text_freq = class_corpus.sum()

fig, ax = plt.subplots()

label, repetition = zip(*whole_text_freq.most_common(25))

ax.barh(range(len(label)), repetition, align='center')
ax.set_yticks(np.arange(len(label)))
ax.set_yticklabels(label)
ax.invert_yaxis()

ax.set_title('Word Distribution Over Whole Text')
ax.set_xlabel('# of repetitions')
ax.set_ylabel('Word')

plt.tight_layout()
plt.show()
```





In [54]:

```
print(class_freq)
```

```
Class
1    [(mutation, 0.30734819737412), (cell, 0.277979...
2    [(mutation, 0.3347555391485223), (cell, 0.2603...
3    [(mutation, 0.34504984530766586), (cell, 0.210...
4    [(mutation, 0.30819369229120197), (cell, 0.266...
5    [(mutation, 0.2661633411228953), (variant, 0.2...
6    [(â^, 0.27073968956666983), (mutation, 0.24195...
7    [(mutation, 0.3232889925505305), (cell, 0.3050...
8    [(cell, 0.28964474678760394), (mutation, 0.270...
9    [(cell, 0.31066809736262496), (mutation, 0.255...
dtype: object
```

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 [6]:

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

# split the data into test and train by maintaining same distribution of output varaible 'y_true'
[stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true, test_size=0.2
)
# split the train data into train and cross validation by maintaining same distribution of output
varaible 'y_train' [stratify=y_train]
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test_size=0.2
)
```

We split the data into train, test and cross validation data sets, preserving the ratio of class distribution in the original data set

In [7]:

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

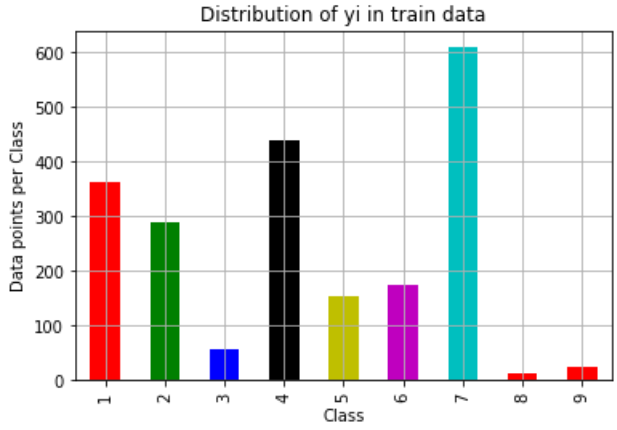
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

In [12]:

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

my_colors = ['r','g','b','k','y','m','c']
train_class_distribution.plot(kind='bar',color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
```

```
plt.grid()
plt.show()


# show the distribution of each class
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distribution.values[i],
          '(', np.round((train_class_distribution.values[i]/train_df.shape[0]*100), 3), '%)')
```

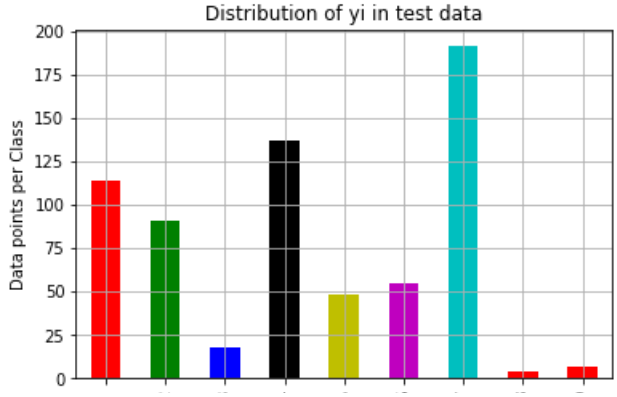


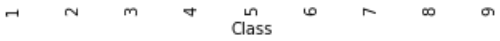

```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298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

Majority class is 7,4,2 and 1 .Dataset is imbalance

In [13]:

```
# test_class distribution
my_colors = ['r','g','b','k','y','m','c']
test_class_distribution.plot(kind='bar',color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()


# show the distribution of each class
# -(test_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-test_class_distribution)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.values[i],
          '(', np.round((test_class_distribution.values[i]/test_df.shape[0]*100), 3), '%)')
```

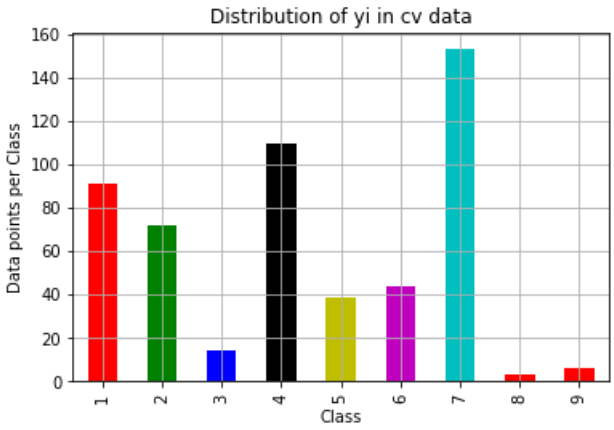





```
Number of data points in class 7 : 191 ( 28.722 %)
Number of data points in class 4 : 137 ( 20.602 %)
Number of data points in class 1 : 114 ( 17.143 %)
Number of data points in class 2 : 91 ( 13.684 %)
Number of data points in class 6 : 55 ( 8.271 %)
Number of data points in class 5 : 48 ( 7.218 %)
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 %)
```

In [14]:

```
# cv _class distribution
my_colors = ['r','g','b','k','y','m','c']
cv_class_distribution.plot(kind='bar',color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cv data')
plt.grid()
plt.show()


# show the distribution of each class
# -(cv_class_distribution): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-cv_class_distribution)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.values[i],
          '(', np.round((cv_class_distribution.values[i]/cv_df.shape[0]*100), 3), '%)')
```




```
Number of data points in class 7 : 153 ( 28.759 %)
Number of data points in class 4 : 110 ( 20.677 %)
Number of data points in class 1 : 91 ( 17.105 %)
Number of data points in class 2 : 72 ( 13.534 %)
Number of data points in class 6 : 44 ( 8.271 %)
Number of data points in class 5 : 39 ( 7.331 %)
Number of data points in class 3 : 14 ( 2.632 %)
Number of data points in class 9 : 6 ( 1.128 %)
Number of data points in class 8 : 3 ( 0.564 %)
```

3.2 Prediction using a 'Random' Model

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

In [15]:

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

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

```
    #divid each element of the confusion matrix with the sum of elements in that column

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in two
diamensional array
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                            [2/3, 4/7]]

    # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
    #                            [3/7, 4/7]]
    # sum of row elements = 1

    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in two
diamensional array
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]

    labels = [1,2,3,4,5,6,7,8,9]
    # representing C in heatmap format
    print("-"*20, "Confusion matrix", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

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

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

In [16]:

```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039

test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

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))
```

```
Log loss on Cross Validation Data using Random Model 2.5197100998140907
```

In [17]:

```
# Test-Set error.
#we create a output array that has exactly same as the test data
```

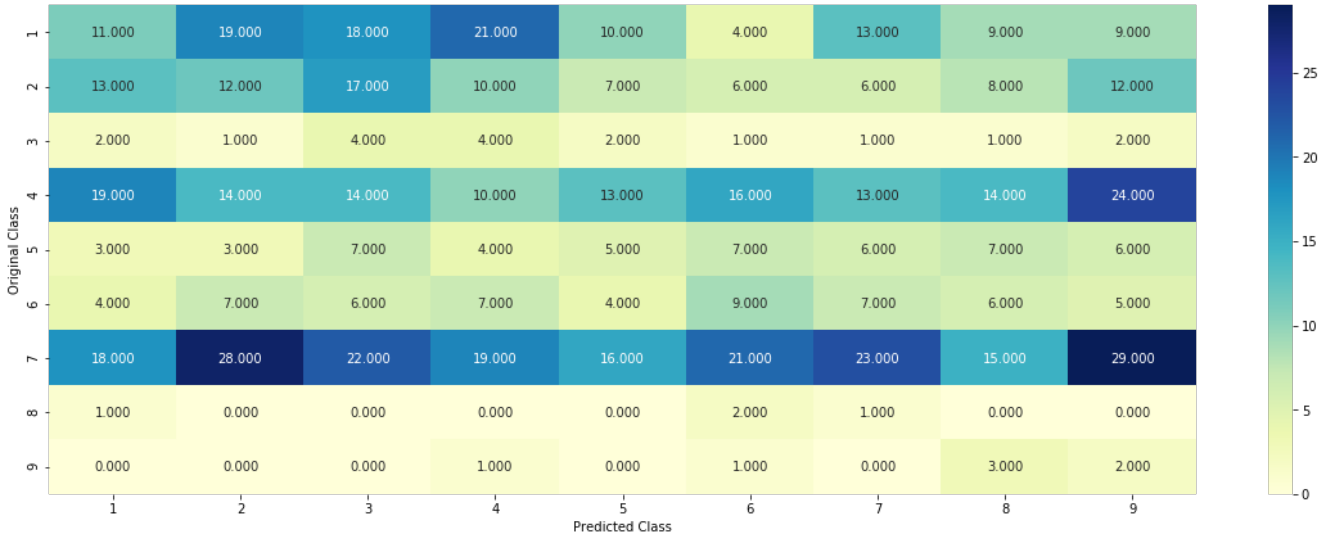
```
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y, eps=1e-15))
```

```
Log loss on Test Data using Random Model 2.4683943293864123
```

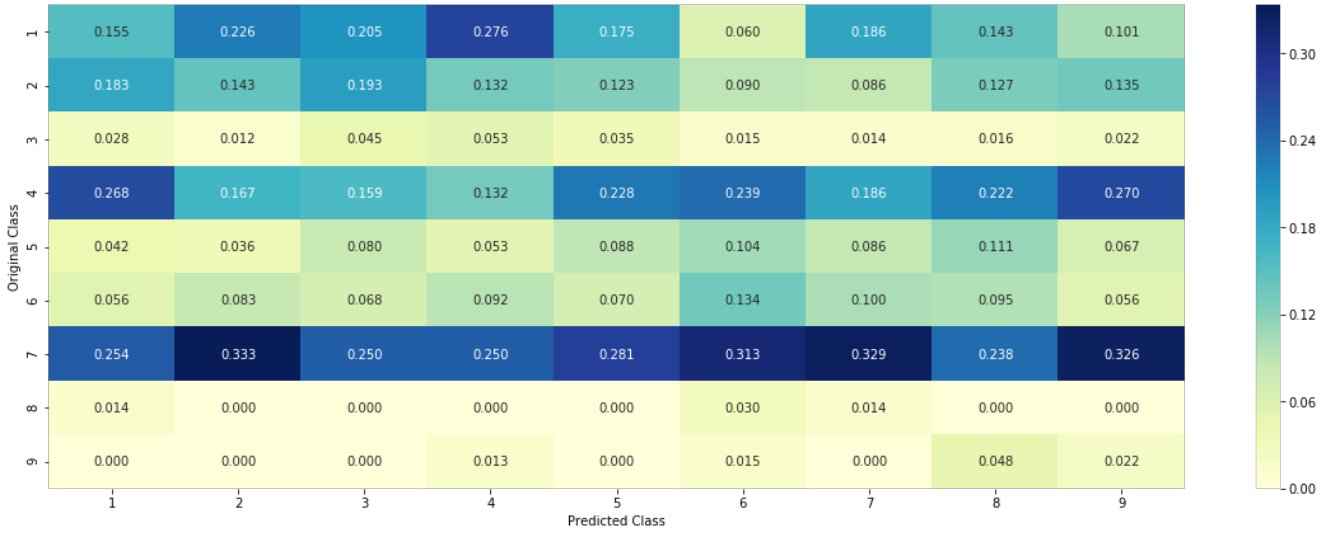
In [18]:

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

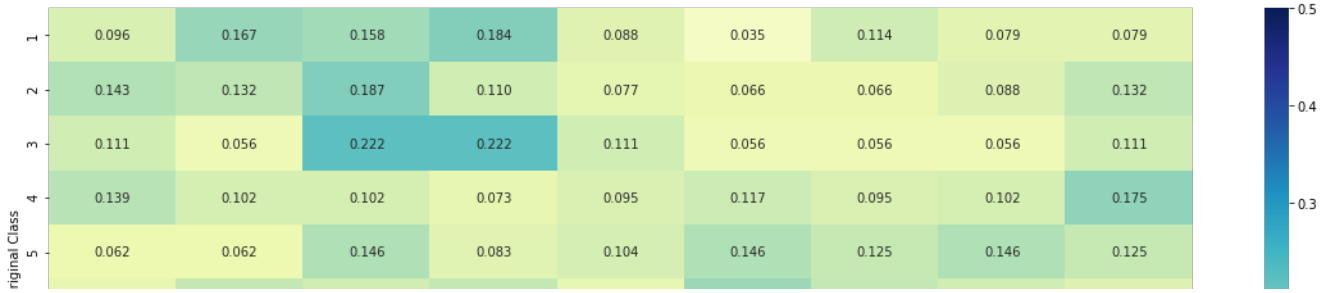
```
-------------------- Confusion matrix --------------------
```

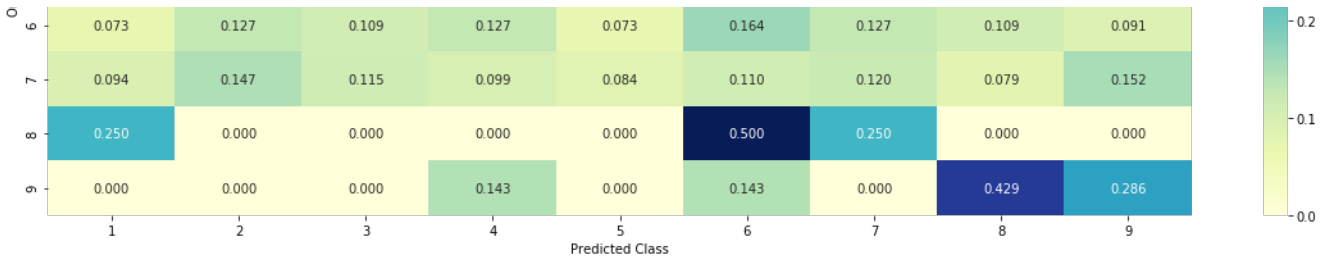



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





3.3 Univariate Analysis

In [19]:

```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feature in train data dataframe
# build a vector (1*9) , the first element = (number of times it occured in class1 + 10*alpha / nu
mber of time it occurred in total data+90*alpha)
# gv_dict is like a look up table, for every gene it store a (1*9) representation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

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

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

    # denominator will contain the number of time that particular feature occured in whole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to perticular class
        # vec is 9 diamensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=='BRCA1')])
            #         ID   Gene             Variation  Class
```

```
            # 2470  2470  BRCA1                S1715C      1
            # 2486  2486  BRCA1                S1841R      1
            # 2614  2614  BRCA1                   M1R      1
            # 2432  2432  BRCA1                L1657P      1
            # 2567  2567  BRCA1                T1685A      1
            # 2583  2583  BRCA1                E1660G      1
            # 2634  2634  BRCA1                W1718L      1
            # cls_cnt.shape[0] will return the number of rows

            cls_cnt = train_df.loc[(train_df['Class']==k) & (train_df[feature]==i)]

            # cls_cnt.shape[0](numerator) will contain the number of time that particular feature o
ccured in whole data
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))

        # we are adding the gene/variation to the dict as key and vec as value
        gv_dict[i]=vec
    return gv_dict

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177,
0.13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788,
0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366,
0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408
163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177,
0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608,
0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608,
0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917,
0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081
761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295,
0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702,
0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334,
0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999,
0.066666666666666666, 0.066666666666666666],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

    # gv_fea: Gene_variation feature, it will contain the feature for each feature value in the da
ta
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there in the train
data then we will add the feature to gv_fea
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_fea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#             gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea
```

when we caculate the probability of a feature belongs to any particular class, we apply laplace smoothing (numerator + 10 *alpha) / (denominator* + *90*alpha)

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

Q2. How many categories are there and How they are distributed?

In [20]:

```
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 : 237
BRCA1      152
TP53       111
EGFR        88
BRCA2       76
PTEN        75
KIT         61
BRAF        61
ALK         45
PDGFRA      44
ERBB2       42
Name: Gene, dtype: int64
```

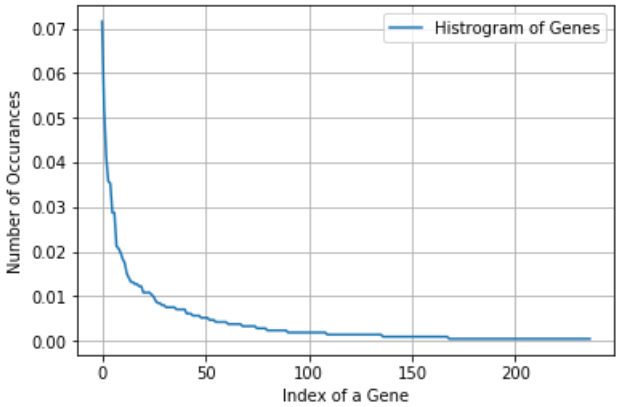
In [21]:

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

```
Ans: There are 237 different categories of genes in the train data, and they are distibuted as fol
lows
```

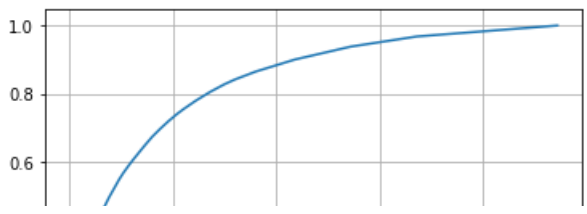
In [22]:

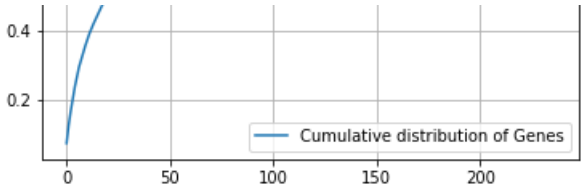
```
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histrogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [23]:

```
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 [24]:

```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
```

In [29]:

```
print("train_gene_feature_responseCoding is converted feature using respone coding method. The sha
pe of gene feature:", train_gene_feature_responseCoding.shape)
```

```
train_gene_feature_responseCoding is converted feature using respone coding method. The shape of g
ene feature: (2124, 9)
```

Assignment Section

1. Task 1, Use TfidfVectorizer for all models

2. Task 2, Use top 1000 words for tfidf

3. Task 3, Apply Logistic regression with CountVectorizer Features, including both unigrams and bigrams

4. Task 4, Apply feature engineering that gives log loss less than 1

1. Task 1, Use TfidfVectorizer for all models

In [25]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer(ngram_range=(1,1))
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]:

```
train_df['Gene'].head()
```

Out[26]:

```
540      SMAD2
913     PDGFRA
3276       RET
274       EGFR
1658      FLT3
Name: Gene, dtype: object
```

In [27]:

```
gene_vectorizer.get_feature_names()
```

Out[27]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid2',
 'asxl2',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'aurkb',
 'axin1',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcl2',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk6',
 'cdk8',
 'cdkn1a',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cdkn2c',
 'cebpa',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctla4',
 'ctnnb1',
 'ddr2',
 'dicer1',
 'dnmt3a',
 'dnmt3b',
 'dusp4',
 'egfr',
```

```
'eif1ax',
'elf3',
'ep300',
'epas1',
'erbb2',
'erbb3',
'erbb4',
'ercc2',
'ercc4',
'erg',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fam58a',
'fanca',
'fat1',
'fbxw7',
'fgf3',
'fgf4',
'fgfr1',
'fgfr2',
'fgfr3',
'fgfr4',
'flt1',
'flt3',
'foxa1',
'foxl2',
'foxo1',
'foxp1',
'fubp1',
'gata3',
'gli1',
'gna11',
'gnaq',
'gnas',
'h3f3a',
'hist1h1c',
'hla',
'hnf1a',
'hras',
'idh1',
'idh2',
'igf1r',
'ikzf1',
'il7r',
'inpp4b',
'jak1',
'jak2',
'jun',
'kdm5c',
'kdm6a',
'kdr',
'keap1',
'kit',
'klf4',
'kmt2a',
'kmt2b',
'kmt2c',
'kmt2d',
'knstrn',
'kras',
'lats1',
'lats2',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
'mapk1',
'mdm2',
'mdm4',
'med12',
'mef2b',
'men1',
'met',
'mga',
```

```
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'mycn',
'myd88',
'myod1',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',
'nkx2',
'notch1',
'notch2',
'npm1',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'nup93',
'pak1',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pim1',
'pms1',
'pms2',
'pole',
'ppm1d',
'ppp2r1a',
'ppp6c',
'prdm1',
'ptch1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rac1',
'rad21',
'rad50',
'rad51b',
'rad51c',
'rad51d',
'raf1',
'rara',
'rasa1',
'rb1',
'rbm10',
'ret',
'rheb',
'rhoa',
'rictor',
'rit1',
'ros1',
'rras2',
'runx1',
'rxra',
'rybp',
'sdhc',
'setd2',
'sf3b1',
'shoc2',
'smad2',
'smad3',
'smad4',
'smarca4',
'smarcb1',
'smo',
'sos1',
```

```
 'sox9',
 'spop',
 'stat3',
 'stk11',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'xpo1',
 'yap1']
```

In [28]:

```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The sha
pe of gene feature:", train_gene_feature_onehotCoding.shape)
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of g
ene feature: (2124, 236)
```

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.

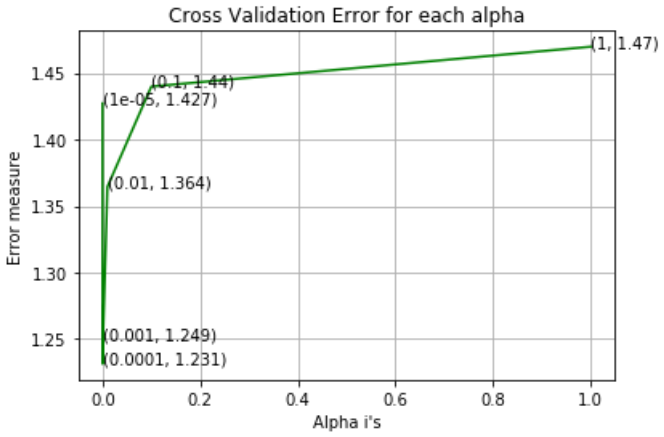
In [29]:

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

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

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


cv_log_error_array = []

for i in alpha:
    clf = SGDClassifier(loss='log',penalty='l2',alpha = i,random_state=42)
    clf.fit(train_gene_feature_onehotCoding,y_train)
    sig_clf = CalibratedClassifierCV(clf,method='sigmoid')
    sig_clf.fit(train_gene_feature_onehotCoding,y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.clas
ses_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.4272373172054902
For values of alpha =  0.0001 The log loss is: 1.23135132074729
For values of alpha =  0.001 The log loss is: 1.2493753174785496
For values of alpha =  0.01 The log loss is: 1.3644453655107456
For values of alpha =  0.1 The log loss is: 1.4398366621199954
For values of alpha =  1 The log loss is: 1.4697713241556722
```

In [30]:

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



In [31]:

```
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)
```

Out[31]:

```
CalibratedClassifierCV(base_estimator=SGDClassifier(alpha=0.0001, average=False,
class_weight=None, epsilon=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='log', max_iter=None, n_iter=None,
       n_jobs=1, penalty='l2', power_t=0.5, random_state=42, shuffle=True,
       tol=None, verbose=0, warm_start=False),
            cv=3, method='sigmoid')
```

In [32]:

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

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

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

```
For values of best alpha =  0.0001 The train log loss is: 1.0351808300975969
For values of best alpha =  0.0001 The cross validation log loss is: 1.23135132074729
For values of best alpha =  0.0001 The test log loss is: 1.187524626965312
```

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 [35]:

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

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  237  genes in train dataset?
Ans
1. In test data 650 out of 665 : 97.74436090225564
2. In cross validation data 515 out of  532 : 96.80451127819549
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

In [36]:

```
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 : 1921
Truncating_Mutations    56
Deletion                49
Amplification           45
Fusions                 26
Overexpression           6
G12V                     3
Q61R                     2
EWSR1-ETV1_Fusion        2
C618R                    2
E17K                     2
Name: Variation, dtype: int64
```

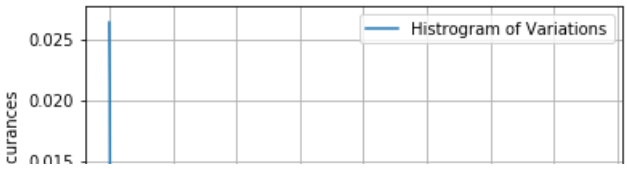
In [37]:

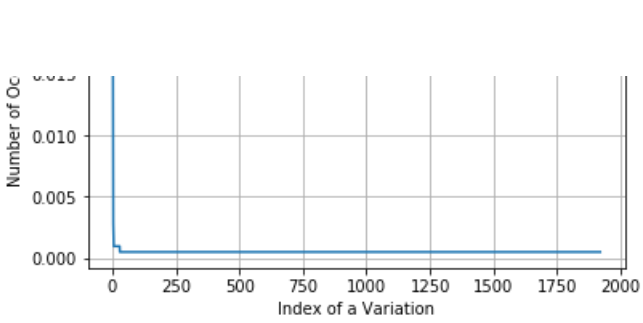
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of variations in the
train data, and they are distibuted as follows",)
```

```
Ans: There are 1921 different categories of variations in the train data, and they are distibuted
as follows
```

In [39]:

```
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()
```

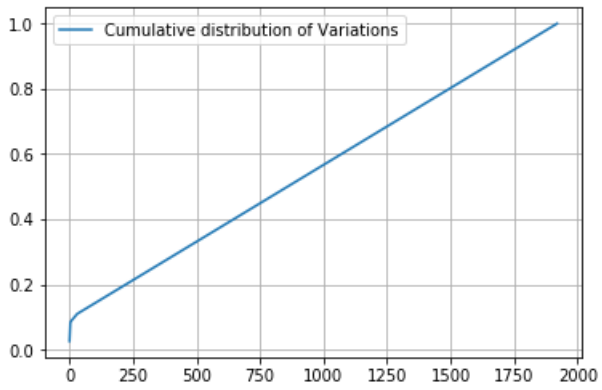




In [40]:

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

```
[0.02636535 0.04943503 0.07062147 ... 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 [41]:

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

In [42]:

```
print("train_variation_feature_responseCoding is a converted feature using the response coding met
hod. The shape of Variation feature:", train_variation_feature_responseCoding.shape)
```

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

In [43]:

```
# one-hot encoding of variation feature.
```

```
variation_vectorizer = TfidfVectorizer(ngram_range=(1,1))
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [44]:

```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding meth
od. The shape of Variation feature:", train_variation_feature_onehotCoding.shape)
```

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The
shape of Variation feature: (2124, 1954)
```

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

Let's build a model just like the earlier!

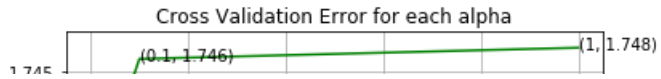
In [45]:

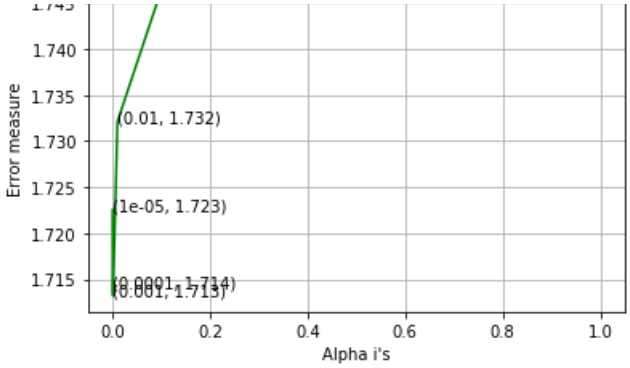
```
alpha = [10 ** x for x in range(-5, 1)]

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

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


cv_log_error_array=[]

for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_variation_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_variation_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)

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

```
For values of alpha =  1e-05 The log loss is: 1.722537529215768
For values of alpha =  0.0001 The log loss is: 1.7140402967823012
For values of alpha =  0.001 The log loss is: 1.7131978778543966
For values of alpha =  0.01 The log loss is: 1.7321143610184577
For values of alpha =  0.1 The log loss is: 1.7464034773948471
For values of alpha =  1 The log loss is: 1.7476172929214595
```

In [46]:

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





In [47]:

```
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)
```

Out[47]:

```
CalibratedClassifierCV(base_estimator=SGDClassifier(alpha=0.001, average=False, class_weight=None,
epsilon=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='log', max_iter=None, n_iter=None,
       n_jobs=1, penalty='l2', power_t=0.5, random_state=42, shuffle=True,
       tol=None, verbose=0, warm_start=False),
            cv=3, method='sigmoid')
```

In [48]:

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

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

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

```
For values of best alpha =  0.001 The train log loss is: 1.0711606295584264
For values of best alpha =  0.001 The cross validation log loss is: 1.7131978778543966
For values of best alpha =  0.001 The test log loss is: 1.699500188536603
```

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 [49]:

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

```
Q12. How many data points are covered by total  1921  genes in test and cross validation data
sets?
Ans
1. In test data 66 out of 665 : 9.924812030075188
2. In cross validation data 48 out of  532 : 9.022556390977442
```

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 [50]:

```
# 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 [51]:

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

In [91]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = TfidfVectorizer(min_df=3,ngram_range=(1,1),max_features=2000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).
# A1 will sum every row and returns (1*number of features) vector
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

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


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

```
Total number of unique words in train data : 2000
```

In [54]:

```
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 [92]:

```
#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 [93]:

```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding =
(train_text_feature_responseCoding.T/train_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding =
(test_text_feature_responseCoding.T/test_text_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_responseCoding.
sum(axis=1)).T
```

In [94]:

```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

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

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

In [95]:

```
#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 [96]:

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

```
Counter({213.05404937216616: 1, 145.48335108930874: 1, 116.28064548803435: 1, 111.97538533559793:
1, 103.34302333129982: 1, 96.84782379806364: 1, 95.15715427160316: 1, 95.00316513399032: 1,
93.3583823019333: 1, 89.04672079106403: 1, 88.07813775922871: 1, 78.82537741236077: 1,
75.8333415355309: 1, 73.65473793460421: 1, 72.06517348599124: 1, 68.9150967677613: 1,
68.1709652487405: 1, 67.37651397938315: 1, 66.59277347881994: 1, 65.55676972458537: 1,
64.00275769180712: 1, 61.51293884120097: 1, 59.17743495005033: 1, 57.345514653017794: 1,
56.84619385726747: 1, 56.213363432092066: 1, 55.71172174095374: 1, 54.979254609044816: 1,
54.69920233631086: 1, 54.19202259440998: 1, 53.15114923114311: 1, 52.985029715602714: 1,
52.9629016606074: 1, 51.746435688751035: 1, 51.375008403292554: 1, 50.47672619652131: 1,
46.88471600207915: 1, 46.71166956018978: 1, 45.82454436147726: 1, 45.37382811631242: 1,
45.31440058409513: 1, 44.58029232921621: 1, 43.03822278448253: 1, 41.940306338725186: 1,
41.433471147997615: 1, 41.327915897978016: 1, 40.88662988294909: 1, 40.852675631532755: 1,
40.65881996792257: 1, 38.9433332838782: 1, 37.65362546185081: 1, 37.05297834282399: 1,
36.81253189829053: 1, 36.68548962291095: 1, 36.53126765718264: 1, 36.364302089093734: 1
```

36.61253189829053: 1, 36.68546962291095: 1, 36.55126765718264: 1, 36.50450268095754: 1,
35.969996828421976: 1, 35.96006409877189: 1, 35.8798062307661: 1, 35.83748412525601: 1,
35.82214005466069: 1, 34.88161506373885: 1, 34.788677396067726: 1, 34.72833331270111: 1,
34.29547016839316: 1, 34.240182466116224: 1, 33.82361628546223: 1, 33.730799064547824: 1,
33.68077777174709: 1, 33.4191363688041: 1, 33.338513796089536: 1, 31.88276592596464: 1,
31.707560239600355: 1, 31.17389895889569: 1, 31.127357331405783: 1, 30.891362612271138: 1,
30.728571321315584: 1, 30.620264195462813: 1, 30.493363008160433: 1, 30.44758684195868: 1,
30.262761956253996: 1, 30.087286665983317: 1, 29.825878710617168: 1, 29.77042505985218: 1,
29.512977645170217: 1, 29.394902773166088: 1, 29.20726928241941: 1, 29.171033633869808: 1,
29.104082092295602: 1, 28.70848935088853: 1, 28.35815911630255: 1, 28.265709733642773: 1,
27.615452463319187: 1, 27.51158383689686: 1, 27.160375943485292: 1, 27.032712637541717: 1,
27.001998523633397: 1, 26.928765547569757: 1, 26.751780535593976: 1, 26.731149293278044: 1,
26.71504716644489: 1, 26.52801786941335: 1, 26.455434874633497: 1, 26.40215013396594: 1,
26.34129829511627: 1, 26.281876580120564: 1, 26.241235361896326: 1, 26.145776864939293: 1,
26.139776405210004: 1, 25.8567012636532: 1, 25.821847847354206: 1, 25.76260953900228: 1,
25.745146630310877: 1, 25.698242111314343: 1, 25.634903600061644: 1, 25.611405736324l4: 1,
25.397892478859113: 1, 25.260740327106657: 1, 25.212180286380153: 1, 25.205534286341916: 1,
25.017451704198546: 1, 24.837027508289516: 1, 24.796499020188953: 1, 24.682291887496 95: 1,
24.622103586563554: 1, 24.418180161465795: 1, 24.38148553819295: 1, 24.366593996541752: 1,
24.319376226121257: 1, 24.14223541506807: 1, 24.13534294604787: 1, 24.093632523195353: 1,
24.04175644906779: 1, 23.90552271526701: 1, 23.888310343977242: 1, 23.638263268909423: 1,
23.246863884640472: 1, 23.20383103336642: 1, 22.92274973679799: 1, 22.710461179178 67: 1, 22.6980253
33488744: 1, 22.682247224724332: 1, 22.674228596558475: 1, 22.597841428557768: 1,
22.529824912180885: 1, 22.522264520085113: 1, 22.484254712827305: 1, 22.21870926489597: 1,
22.13531134500571: 1, 21.87559036844754: 1, 21.8181956502857: 1, 21.72495639824731: 1,
21.6490644177146 94: 1, 21.578786136811715: 1, 21.5714585726457 03: 1, 21.45890469149049: 1,
21.314146136859584: 1, 21.277643863986324: 1, 21.2199909265221 83: 1, 21.1069399601429 38: 1,
20.77918723172775: 1, 20.742207164451354: 1, 20.624551125872134: 1, 20.591211559835514: 1,
20.45791217541458: 1, 20.43984427734933: 1, 20.40453433376474: 1, 20.36059779039014: 1,
20.311367762783483: 1, 20.27365870250435: 1, 20.270942504535853: 1, 20.253897871563233: 1,
20.247858943262127: 1, 20.22671531423491: 1, 20.220708147484 13: 1, 20.2090526334030 83: 1,
20.171481221114487: 1, 20.116046971699568: 1, 20.107759968003386: 1, 19.813414718762107: 1,
19.622803520824462: 1, 19.604213095480137: 1, 19.571381961861853: 1, 19.510644178370 55: 1,
19.51004273262726: 1, 19.498852434564395: 1, 19.46602738025691: 1, 19.43228401086949: 1,
19.429104124627404: 1, 19.413205476059037: 1, 19.370120346339743: 1, 19.354491724656 04: 1,
19.31049220911171: 1, 19.157496434506992: 1, 19.099188750314173: 1, 19.094484955002887: 1,
19.0483856340463: 1, 19.011065016637033: 1, 18.916354190594838: 1, 18.906777868388097: 1,
18.90665824606034: 1, 18.884915817541227: 1, 18.74464076960766: 1, 18.671362932160392: 1,
18.644205117410056: 1, 18.637202093696473: 1, 18.63537594853157: 1, 18.588026359963 98: 1,
18.574113035267118: 1, 18.551901873495506: 1, 18.544921981319995: 1, 18.407907679829176: 1,
18.386955839112897: 1, 18.364669941228478: 1, 18.362686909342457: 1, 18.308194105474 64: 1,
18.294368532548233: 1, 18.284047675929404: 1, 18.265247683120958: 1, 18.165272421614898: 1,
18.10196619422281: 1, 18.078071715218567: 1, 18.043935457872823: 1, 18.008266018114107: 1,
17.992508151660555: 1, 17.96384457611813: 1, 17.902406818026012: 1, 17.888463214336454: 1,
17.868925705196922: 1, 17.834498695670042: 1, 17.779348832015057: 1, 17.755868787414016: 1,
17.671895373795177: 1, 17.627182048208105: 1, 17.619453068153373: 1, 17.612013811803774: 1,
17.57349913578914: 1, 17.569871212133922: 1, 17.566861106453498: 1, 17.520001440551 06: 1,
17.412856328853234: 1, 17.3664625415196: 1, 17.332700384940868: 1, 17.327868421557 23: 1,
17.28047242376902: 1, 17.13260892399929: 1, 17.120629987588345: 1, 17.089049284724 38: 1,
16.93677118672786: 1, 16.899413437465 09: 1, 16.859971570257017: 1, 16.793035162208 73: 1,
16.783161136043798: 1, 16.780569798055 73: 1, 16.740037396369218: 1, 16.646579839973 23: 1,
16.643658759265772: 1, 16.537220496672084: 1, 16.526671805268798: 1, 16.525587638454066: 1,
16.519762496961903: 1, 16.518057775769 08: 1, 16.511076624422355: 1, 16.496645491858718: 1,
16.35474765274164: 1, 16.340526273741073: 1, 16.32424517515763: 1, 16.297598204139387: 1,
16.29376223200574: 1, 16.235639425381564: 1, 16.163651102885396: 1, 16.119249921698728: 1,
16.110168618283122: 1, 16.096163313793262: 1, 16.056444779918063: 1, 16.045933017277278: 1,
16.02973499895425: 1, 16.01568152209416: 1, 16.005961101532336: 1, 15.989158056258443: 1,
15.956541690197913: 1, 15.934570779768507: 1, 15.928117290395239: 1, 15.921699799317786: 1,
15.888115472368447: 1, 15.845944158301 22: 1, 15.8200745247717 74: 1, 15.8139650988449 03: 1,
15.807187058416941: 1, 15.780148204501883: 1, 15.741115148421041: 1, 15.633776322603104: 1,
15.624066082172753: 1, 15.566432298358597: 1, 15.439098243374618: 1, 15.427992211955848: 1,
15.414037145430969: 1, 15.407723677706128: 1, 15.32406845675637: 1, 15.256691916746057: 1,
15.243602787481839: 1, 15.213161022022279: 1, 15.199870545666839: 1, 15.179326139519164: 1,
15.131665038071537: 1, 15.013926014599585: 1, 14.973595528200297: 1, 14.970770605833 35: 1,
14.936882878363164: 1, 14.933561634333891: 1, 14.927667099194116: 1, 14.913773761252386: 1,
14.899545105962408: 1, 14.842146999512178: 1, 14.838663085898084: 1, 14.836861714909 42: 1,
14.797153541658979: 1, 14.777873311862884: 1, 14.74544053323569: 1, 14.704890775258143: 1,
14.697879125643183: 1, 14.655534256184502: 1, 14.648115215587413: 1, 14.594420810789778: 1,
14.582925086255678: 1, 14.579529044770645: 1, 14.535271705926027: 1, 14.533666161023623: 1,
14.528668283367967: 1, 14.516960199925967: 1, 14.515789037663 78: 1, 14.4154472622779 97: 1,
14.34781172932768: 1, 14.342664424008 58: 1, 14.266490991952283: 1, 14.261043449459438: 1,
14.250754035272577: 1, 14.242175949705464: 1, 14.186943929740695: 1, 14.169983246915441: 1,
14.166207761338486: 1, 14.151901515328868: 1, 14.136218514301762: 1, 14.081027359461302: 1,
14.076962627035 19: 1, 14.0544828217471 63: 1, 14.0524136850989 78: 1, 14.0476401247561 2: 1,
14.021315108447203: 1, 13.959298853245533: 1, 13.925578688608503: 1, 13.917703133757243: 1,
13.916894215273517: 1, 13.9149257751387: 1, 13.898510651167266: 1, 13.883276507138051: 1,
13.856312212431396: 1, 13.852628280782351: 1, 13.732346972357563: 1, 13.689225108480896: 1,
13.625725127419871: 1, 13.618903848430932: 1, 13.551902388143946: 1, 13.547139245274906: 1,

13.625725127419871: 1, 13.618903848430932: 1, 13.551902368143946: 1, 13.547139245274906: 1,
13.490043256618952: 1, 13.483524082111447: 1, 13.472239760698018: 1, 13.450296156101246: 1,
13.440469808605874: 1, 13.412820259939016: 1, 13.411217062417085: 1, 13.391326713493331: 1,
13.366260308337681: 1, 13.36555617452063: 1, 13.345212983352628: 1, 13.336857463173555: 1,
13.311633893457786: 1, 13.26650865268587: 1, 13.263484744671699: 1, 13.229591318884446: 1,
13.205807731699233: 1, 13.188085224298248: 1, 13.152975000199659: 1, 13.13644410026511: 1,
13.131567311893791: 1, 13.127749051059288: 1, 13.127047299997328: 1, 13.119707609197125: 1,
13.117840622667075: 1, 13.110494960809557: 1, 13.107991905397558: 1, 13.095149211959713: 1,
13.074532898902616: 1, 13.004410217648111: 1, 13.003572021417842: 1, 12.997866443070123: 1,
12.997694861518005: 1, 12.972286004093048: 1, 12.971604926915363: 1, 12.957741985159746: 1,
12.946681284939434: 1, 12.881761032278709: 1, 12.862280517547987: 1, 12.84455785356101: 1,
12.835231387457098: 1, 12.797922071638446: 1, 12.787138255924633: 1, 12.781914967647069: 1,
12.751559007528371: 1, 12.731702224696514: 1, 12.720536082485598: 1, 12.71983080173298: 1,
12.714029104834058: 1, 12.708213440204933: 1, 12.67331682417353: 1, 12.670633516804267: 1,
12.66580816360253: 1, 12.632877047408252: 1, 12.63041125102164: 1, 12.613469436405193: 1,
12.594611733001635: 1, 12.583329221920836: 1, 12.577083082870168: 1, 12.56438927330687: 1,
12.559206567429767: 1, 12.531528379266454: 1, 12.519863841257441: 1, 12.514820940927162: 1,
12.406872882105601: 1, 12.403953639502411: 1, 12.402984184670117: 1, 12.381768729593428: 1,
12.377617340967861: 1, 12.374725464341902: 1, 12.352487593476613: 1, 12.346098351995778: 1,
12.255221329460914: 1, 12.246722156268623: 1, 12.246602235295128: 1, 12.201546938313834: 1,
12.18533602175951: 1, 12.177709060525192: 1, 12.146397160469238: 1, 12.142175880754797: 1,
12.123433971638809: 1, 12.12173743513785: 1, 12.083573131297836: 1, 12.08108457324975: 1,
12.061226305374884: 1, 12.060892582031407: 1, 12.053471762311867: 1, 12.033179081964208: 1,
12.004704504923454: 1, 12.003834516187432: 1, 12.001385009986686: 1, 11.983547472475893: 1,
11.960872222936363: 1, 11.951930520079982: 1, 11.931012144223432: 1, 11.927650253121376: 1,
11.915488140958995: 1, 11.914129227261949: 1, 11.902310164779925: 1, 11.896750410887554: 1,
11.894390395253174: 1, 11.847489577040708: 1, 11.833000365199805: 1, 11.800359813064347: 1,
11.795419518905929: 1, 11.756938716740438: 1, 11.755444782611939: 1, 11.73546837083661: 1,
11.72222929207402: 1, 11.705223239891152: 1, 11.703674334644122: 1, 11.660897509594818: 1,
11.65830320456096: 1, 11.627993022291312: 1, 11.579853897114742: 1, 11.556171293318748: 1,
11.533857646136171: 1, 11.531844317404794: 1, 11.52303615212762: 1, 11.51249115053464: 1,
11.505031298594046: 1, 11.504014602944943: 1, 11.444675438065318: 1, 11.426315102271495: 1,
11.422749125952395: 1, 11.419584354724892: 1, 11.405616761794837: 1, 11.378803242775007: 1,
11.364810111112826: 1, 11.323435506051725: 1, 11.305197871877187: 1, 11.29604780222423: 1,
11.284255386997522: 1, 11.277313379150653: 1, 11.272503662245569: 1, 11.25623205061998: 1,
11.246874846797008: 1, 11.226352235305635: 1, 11.201850667103239: 1, 11.20072345519994: 1,
11.184727523520989: 1, 11.184142168943156: 1, 11.172022006754682: 1, 11.138318886960016: 1,
11.103188967309261: 1, 11.098980306640117: 1, 11.095754385011796: 1, 10.986295539568939: 1,
10.980804015986514: 1, 10.945325565234874: 1, 10.910982325501793: 1, 10.909357436503266: 1,
10.895347119820265: 1, 10.894359224867666: 1, 10.889979917161808: 1, 10.880766745120791: 1,
10.869455382864892: 1, 10.847711537389511: 1, 10.831402699624437: 1, 10.81458154059533: 1,
10.809537717621051: 1, 10.763863366481793: 1, 10.737562455133332: 1, 10.724046193679964: 1,
10.713241351065262: 1, 10.709862093926162: 1, 10.693564223608172: 1, 10.687802102521181: 1,
10.68665493675233: 1, 10.671952860412561: 1, 10.670239657868372: 1, 10.646172664082046: 1,
10.63357159079092: 1, 10.622664629454443: 1, 10.613251050808701: 1, 10.590178398477226: 1,
10.56726321162545: 1, 10.553748316847196: 1, 10.551287413559125: 1, 10.545165853086663: 1,
10.537717046088936: 1, 10.517448252366295: 1, 10.48549376357761: 1, 10.481692823110912: 1,
10.477282889224854: 1, 10.462427723928034: 1, 10.45671031535591: 1, 10.432516477342311: 1,
10.420811484466629: 1, 10.419614414743677: 1, 10.415617761292003: 1, 10.413940484714304: 1,
10.408039655042455: 1, 10.405514501953917: 1, 10.397234893180407: 1, 10.393323433171771: 1,
10.38937957535876: 1, 10.368662452441123: 1, 10.349268078181886: 1, 10.320165107100761: 1,
10.317452685377367: 1, 10.298471825022679: 1, 10.294387103107203: 1, 10.293334160591982: 1,
10.286355861638759: 1, 10.284065271184941: 1, 10.276815304576331: 1, 10.27617305867102: 1,
10.270063777883422: 1, 10.264204484347996: 1, 10.243629779133332: 1, 10.229203973650426: 1,
10.200810412444188: 1, 10.192412929731594: 1, 10.156885234105001: 1, 10.138702519513625: 1,
10.131420542727831: 1, 10.124695830859249: 1, 10.123156217222476: 1, 10.037517449336557: 1,
9.976246227318473: 1, 9.945004507693518: 1, 9.931237723028348: 1, 9.91514894206831: 1,
9.889843643611025: 1, 9.886184441757766: 1, 9.87437240655647: 1, 9.8702447322212772: 1,
9.866662128386256: 1, 9.865767969073287: 1, 9.861950596133465: 1, 9.852244214019112: 1,
9.84478060324909: 1, 9.843030003361777: 1, 9.840847607984903: 1, 9.839612989144772: 1,
9.83111833787216: 1, 9.809714355253202: 1, 9.803829472181796: 1, 9.8015228597596741: 1,
9.800205559770891: 1, 9.790339790477781: 1, 9.786259582695834: 1, 9.783037895390288: 1,
9.751487809265118: 1, 9.743073485285862: 1, 9.730016921326657: 1, 9.722710395944155: 1,
9.714150133971831: 1, 9.703390841789684: 1, 9.6781064575214431: 1, 9.676956034941991: 1,
9.659363617727385: 1, 9.6358368590666: 1, 9.612235425674982: 1, 9.583908484192364: 1,
9.564169548739605: 1, 9.560061772542435: 1, 9.557006407214221: 1, 9.555444988151597: 1,
9.553844900325526: 1, 9.547952797030806: 1, 9.542556976737293: 1, 9.530050960520851: 1,
9.513346199350043: 1, 9.507927999380192: 1, 9.496522846953438: 1, 9.493351210380741: 1,
9.489250518197796: 1, 9.484721088388303: 1, 9.448456175970172: 1, 9.436559003061168: 1,
9.435369193490853: 1, 9.420548798247607: 1, 9.41654887752827: 1, 9.4158299280716471: 1,
9.415142518625921: 1, 9.4031583135071021: 1, 9.398878406892488: 1, 9.3708880078504321: 1,
9.359947681240353: 1, 9.359360893421574: 1, 9.3470833293411441: 1, 9.337651703485722: 1,
9.33277107970235: 1, 9.326641381384: 1, 9.303762116350455: 1, 9.28908748239576: 1,
9.283345895938814: 1, 9.270243862871126: 1, 9.267320290310293: 1, 9.2608091550690141: 1,
9.243980657627747: 1, 9.235593111369997: 1, 9.232383165034262: 1, 9.2181753101196671: 1,
9.201124966159862: 1, 9.199958045278706: 1, 9.199532758034541: 1, 9.1652525886573951: 1,
9.164694915497613: 1, 9.159962615854159: 1, 9.137727110477801: 1, 9.137387208480341: 1,

9.137229507373 5: 1, 9.128850205524758: 1, 9.106904512365759: 1, 9.095892758423672: 1, 9.07223290526
701: 1, 9.0636393294903: 1, 9.062499688948249: 1, 9.061783341293811: 1, 9.057960939381495: 1,
9.053172175680526: 1, 9.043040410198815: 1, 9.03559275293654: 1, 9.018850256982796: 1,
9.01591386951948: 1, 9.001444018682992: 1, 8.991022145330888: 1, 8.987817775195865: 1,
8.983769390611117: 1, 8.971160928574168: 1, 8.969220323478908: 1, 8.966693439295192: 1,
8.960505968702641: 1, 8.94932367452037: 1, 8.948783344979352: 1, 8.944500164792082: 1,
8.937436001295284: 1, 8.935752665269453: 1, 8.930362580742417: 1, 8.924302600938185: 1,
8.899140202141682: 1, 8.897898059520186: 1, 8.897184661727186: 1, 8.896000708021592: 1,
8.892978178774438: 1, 8.859251408485344: 1, 8.857720365936139: 1, 8.857557130110727: 1,
8.850114379739413: 1, 8.847742123705547: 1, 8.84275994893946: 1, 8.836855504055107: 1,
8.833527111148216: 1, 8.823330681370573: 1, 8.812588580879247: 1, 8.807455087683834: 1,
8.804468162652404: 1, 8.797144012699478: 1, 8.792998825073202: 1, 8.789515720895912: 1,
8.788436658316243: 1, 8.778093658759888: 1, 8.771627403586379: 1, 8.762513590663653: 1,
8.762383756223901: 1, 8.759014880584008: 1, 8.73495694765284: 1, 8.731487588351325: 1,
8.726806989645565: 1, 8.72572033190282: 1, 8.724384556748662: 1, 8.71975147869472: 1,
8.714874937935637: 1, 8.69945814465267: 1, 8.69314976858306: 1, 8.689631173873659: 1,
8.677300174784666: 1, 8.670106589824096: 1, 8.666174830134294: 1, 8.664784323706904: 1,
8.66285747350142: 1, 8.6502787044576: 1, 8.647337308219482: 1, 8.64157356115185: 1,
8.63923013767591: 1, 8.631612234162002: 1, 8.630928665337668: 1, 8.629871674696192: 1,
8.625697016845937: 1, 8.611902829974424: 1, 8.59392985050303: 1, 8.588182718735577: 1,
8.582489162273603: 1, 8.569501858290156: 1, 8.565876836639223: 1, 8.564623025371251: 1,
8.554361797009069: 1, 8.553503925156926: 1, 8.547286717135021: 1, 8.536530456449762: 1,
8.535138713061714: 1, 8.527988583550549: 1, 8.526611900616269: 1, 8.524777687042217: 1,
8.518227318164106: 1, 8.48322833561121: 1, 8.454173683758547: 1, 8.443847968328557: 1,
8.440346168988079: 1, 8.437483370539285: 1, 8.43708074395167: 1, 8.429306557473828: 1,
8.405844156997487: 1, 8.384754782703633: 1, 8.3747247128716 48: 1, 8.369414343898532: 1,
8.349868784202167: 1, 8.343149615434415: 1, 8.340061975385309: 1, 8.335275380270746: 1,
8.32614854935355: 1, 8.325041061532778: 1, 8.316737761016787: 1, 8.31572557067677: 1,
8.31473463653294: 1, 8.311332144846018: 1, 8.294025306944945: 1, 8.280344580462943: 1,
8.268303852298935: 1, 8.263424931381785: 1, 8.24335690541458: 1, 8.231121776307479: 1,
8.226263992751072: 1, 8.219777242321799: 1, 8.21114375974809: 1, 8.206838460861707: 1,
8.203780082289093: 1, 8.194274338358527: 1, 8.189906548704165: 1, 8.17893363629347: 1,
8.17056051501659: 1, 8.167846324587236: 1, 8.158299036256363: 1, 8.155512501116277: 1,
8.153899926242412: 1, 8.153397606633678: 1, 8.149554153192895: 1, 8.140075386951393: 1,
8.121202848909775: 1, 8.117492500233059: 1, 8.115757856506507: 1, 8.109255314777753: 1,
8.106789556620578: 1, 8.09735714497157: 1, 8.071244199954174: 1, 8.06908290995648: 1,
8.062739067136864: 1, 8.058895702394839: 1, 8.052023563010247: 1, 8.049906800987495: 1,
8.045595524037521: 1, 8.035723528163116: 1, 8.0097420167315: 1, 8.00338778113941: 1,
8.0012280421828: 1, 7.997887319996847: 1, 7.99710145873116: 1, 7.996521900727092: 1,
7.993066552990249: 1, 7.965043289838017: 1, 7.963190052933032: 1, 7.954768387426912: 1,
7.95187426638673: 1, 7.947967208665908: 1, 7.947770005657376: 1, 7.947370802184349: 1,
7.946224068089099: 1, 7.934991900633666: 1, 7.929321090663234: 1, 7.92803261896445: 1,
7.928002009755031: 1, 7.922103294130345: 1, 7.910233404127978: 1, 7.902454294697278: 1,
7.893263510313896: 1, 7.890594652401362: 1, 7.884708166429794: 1, 7.8820560709463665: 1,
7.872271342496071: 1, 7.861747214126006: 1, 7.858298017277356: 1, 7.848152324884496: 1,
7.84640374785164: 1, 7.844350927987393: 1, 7.842357407006808: 1, 7.83303076554747: 1,
7.820839210423709: 1, 7.813283659211128: 1, 7.802407662167254: 1, 7.8020294756269015: 1,
7.798689843126692: 1, 7.79861864939115: 1, 7.798541475256206: 1, 7.788038097425428: 1,
7.784875649406426: 1, 7.784640971032555: 1, 7.780798188995789: 1, 7.776231906955396: 1,
7.764779680221616: 1, 7.762502936152912: 1, 7.752620259947757: 1, 7.750594383893756: 1,
7.744278162323053: 1, 7.7422461421058735: 1, 7.740392252066785: 1, 7.737657671694809: 1,
7.73759817556095: 1, 7.693090054899273: 1, 7.691119819648637: 1, 7.6878798627217115: 1,
7.683687795949142: 1, 7.676150127545532: 1, 7.6625055267049005: 1, 7.64657401017438: 1,
7.637441024584559: 1, 7.628134528393446: 1, 7.627709531654527: 1, 7.62475717440521: 1,
7.622783957546339: 1, 7.620769028549234: 1, 7.619958803487029: 1, 7.6175003776732 09: 1,
7.61088716578109: 1, 7.6073016450227975: 1, 7.606949811884066: 1, 7.595249993430317: 1,
7.587107359135822: 1, 7.578663579256662: 1, 7.562221977865209: 1, 7.561138969661614: 1,
7.556751563022012: 1, 7.553501638948945: 1, 7.54638577846793: 1, 7.54413159661806: 1,
7.543956171868232: 1, 7.533977270317201: 1, 7.531184380414962: 1, 7.5302954571178 64: 1,
7.5190548919040525: 1, 7.515741012313866: 1, 7.5132553475791 53: 1, 7.5090546551614 95: 1, 7.50856013
368673: 1, 7.501156813277625: 1, 7.500718732248874: 1, 7.499108445010487: 1, 7.488285769641318: 1,
7.473919283606508: 1, 7.4726220384933315: 1, 7.472300699898394: 1, 7.457504822402534: 1,
7.455482183392896: 1, 7.454016678285991: 1, 7.453633219044137: 1, 7.441401980845073: 1,
7.428353879669855: 1, 7.426832286928211: 1, 7.420193551243696: 1, 7.41717088823616: 1,
7.404877871649536: 1, 7.394942348433241: 1, 7.392042572203696: 1, 7.384801696892634: 1,
7.377155267884638: 1, 7.360196869325808: 1, 7.3593443672993155: 1, 7.3544071942346685: 1,
7.351896959756381: 1, 7.345790876271203: 1, 7.34381204049475: 1, 7.335264784375461: 1,
7.333850395910722: 1, 7.319734996004856: 1, 7.315667460866951: 1, 7.306591423777689: 1,
7.300881933119008: 1, 7.298121838068105: 1, 7.279989345026307: 1, 7.255703208716958: 1,
7.253590183361759: 1, 7.251192482820909: 1, 7.247125628760591: 1, 7.245165530199609: 1,
7.226809093418702: 1, 7.2148762713927805: 1, 7.213995900546406: 1, 7.213718581350782: 1,
7.211509378337818: 1, 7.20958565661147: 1, 7.1891740084308715: 1, 7.186495303279108: 1,
7.1848550127815285: 1, 7.18463665210885: 1, 7.183225081181433: 1, 7.182250510772264: 1,
7.173350334932524: 1, 7.172311062905572: 1, 7.166339486730218: 1, 7.161463813037114: 1,
7.157374800047909: 1, 7.157089477817664: 1, 7.153294178568079: 1, 7.144075801006581: 1,
7.143976814243779: 1, 7.133948653704651: 1, 7.130495618969044: 1, 7.1276079339494505: 1,
7.126672337034641: 1, 7.122274656030502: 1, 7.121080279858884: 1, 7.119759483967616: 1,

7.1164029036267O5: 1, 7.110113724892S8: 1, 7.100720313872892: 1, 7.100034997587578: 1,
7.097974212655977: 1, 7.09655195700439: 1, 7.09619996554801: 1, 7.095324805446567: 1,
7.0916061071746475: 1, 7.090749637589886: 1, 7.080081122778086: 1, 7.074721905630448: 1, 7.07053046
7786233: 1, 7.0599130119853: 1, 7.045117283220475: 1, 7.034751863154738: 1, 7.025698371055274: 1,
7.020514398350534: 1, 7.009800720510048: 1, 7.007965439227272: 1, 7.006225136028085: 1,
7.003536787961298: 1, 6.998099640481116: 1, 6.986657506085239: 1, 6.981827336484156: 1,
6.965177890432452: 1, 6.963580394874161: 1, 6.961993144581865: 1, 6.958666720322199: 1,
6.957222003443041: 1, 6.946428149458542: 1, 6.934140031135903: 1, 6.932110739375174: 1,
6.927937942429983: 1, 6.918890553342481: 1, 6.917601236151336: 1, 6.912370443447097: 1,
6.909373774983258: 1, 6.902755153509138: 1, 6.897188346562157: 1, 6.896938469493263: 1,
6.889705965521001: 1, 6.884379643214S2: 1, 6.880894268090917: 1, 6.866209346577112: 1,
6.85233186037904: 1, 6.851488124369203: 1, 6.849578320031533: 1, 6.848250869712546: 1,
6.847083050268967: 1, 6.847054437783318: 1, 6.846284717861327: 1, 6.845473733940983: 1,
6.843499429984099: 1, 6.842826163199879: 1, 6.840781827499486: 1, 6.840071578464845: 1,
6.83882002848546: 1, 6.8368795581377055: 1, 6.83564789898534: 1, 6.827017710224312: 1,
6.826878521319558: 1, 6.820324644252899: 1, 6.819295534311352: 1, 6.8134052425926: 1,
6.7976814344698635: 1, 6.793407935319368: 1, 6.79233136706863: 1, 6.7875900281711425: 1,
6.781142061803491: 1, 6.764248941464921: 1, 6.756940243793238: 1, 6.7546083334420794: 1,
6.752337707698757: 1, 6.749535981249132: 1, 6.7478300462083425: 1, 6.745960188808402: 1,
6.740716056620714: 1, 6.735019365699793: 1, 6.726547470186282: 1, 6.723758830546393: 1,
6.714050255581617: 1, 6.711379000328713: 1, 6.6995963066679085: 1, 6.695159021439881: 1,
6.67873919109043: 1, 6.672877715456055: 1, 6.670030638628791: 1, 6.66155160308035: 1,
6.660683216989587: 1, 6.656557774401357: 1, 6.653694259002858: 1, 6.6532268859224315: 1,
6.65277118161067: 1, 6.6479228375053S8: 1, 6.6352714330482545: 1, 6.634765810998404: 1,
6.6346130216879535: 1, 6.632199600600926: 1, 6.63163765157962: 1, 6.628813112368G9: 1,
6.62168685363957: 1, 6.6187024530630I3: 1, 6.6141735587450I8: 1, 6.599273142139S9: 1,
6.596551463924504: 1, 6.595149807158831: 1, 6.594631795015989: 1, 6.594021894996043: 1,
6.583416053213752: 1, 6.581540259574203: 1, 6.580145865446065: 1, 6.5772742842290235: 1,
6.5604411755816034: 1, 6.557117334927986: 1, 6.5561167454099785: 1, 6.555864759463766: 1,
6.54940308822373: 1, 6.53645065904917: 1, 6.531513909071078: 1, 6.528483398932387: 1,
6.527796349140902: 1, 6.527643958365563: 1, 6.525889219194991: 1, 6.522511564022999: 1,
6.518777620200754: 1, 6.518202893573136: 1, 6.517819642588107: 1, 6.516293355867587: 1,
6.513446705831252: 1, 6.513186267572259: 1, 6.500112240744504: 1, 6.499863785043934: 1,
6.498073930910348: 1, 6.492396724707025: 1, 6.474635886716406: 1, 6.458533178116513: 1,
6.457946830107174: 1, 6.450882354735795: 1, 6.448437604279898: 1, 6.444089340179999: 1,
6.443854045417191: 1, 6.442344671852118: 1, 6.426170445897186: 1, 6.422773526024996: 1,
6.4226150374178665: 1, 6.417018761486014: 1, 6.416813421942487: 1, 6.416056693480281: 1, 6.41259868
7052221: 1, 6.410341653915514: 1, 6.408694614824735: 1, 6.408344353601194: 1, 6.40178778642947: 1,
6.38530479902824: 1, 6.374085279972631: 1, 6.364846449212691: 1, 6.362386692803633: 1,
6.3607668258927585: 1, 6.359705428222474: 1, 6.356169485749092: 1, 6.350585412099423: 1, 6.34311097
2419299: 1, 6.336242554340125: 1, 6.336052333587181: 1, 6.335705596010504: 1, 6.334599493532367: 1,
6.3306292405083635: 1, 6.32947677172858: 1, 6.322820378908155: 1, 6.318188689267435: 1,
6.312971847576126: 1, 6.3097773811072315: 1, 6.301911500530034: 1, 6.296425274765857: 1,
6.289692441795137: 1, 6.278337694421849: 1, 6.266956667049364: 1, 6.263361476047121: 1,
6.260823860409945: 1, 6.260120231858197: 1, 6.2528875308848395: 1, 6.248814461065131: 1,
6.236263213332828: 1, 6.223180547440056: 1, 6.220793188590704: 1, 6.218048404985525: 1,
6.212603405063168: 1, 6.212356346394935: 1, 6.210208976707498: 1, 6.204962049674353: 1,
6.199293943989431: 1, 6.196775659767447: 1, 6.196128664048798: 1, 6.180079204266441: 1,
6.178909614556113: 1, 6.175234736934869: 1, 6.1710687785295795: 1, 6.168377512876S8: 1,
6.168348133298902: 1, 6.166897535938664: 1, 6.157858534291891: 1, 6.1518795202833765: 1,
6.148339760436735: 1, 6.144555422144078: 1, 6.139596084172676: 1, 6.13768961339933: 1,
6.128710879644378: 1, 6.128176495013632: 1, 6.114700913261248: 1, 6.113316623672547: 1,
6.110949381432534: 1, 6.10840751379785: 1, 6.099770924376529: 1, 6.089359791178831: 1,
6.087092027934297: 1, 6.085792963615403: 1, 6.085511813823195: 1, 6.0836357895913595: 1,
6.0805861552978255: 1, 6.0693708505905395: 1, 6.066246547053723: 1, 6.06411294582436: 1,
6.062058394969885: 1, 6.060690786159046: 1, 6.050105935710976: 1, 6.049368229410515: 1,
6.044103150456811: 1, 6.041465645105894: 1, 6.038841789773344: 1, 6.03523569102789: 1,
6.025788709010425: 1, 6.0233339519951725: 1, 6.022783501253936: 1, 6.019483448511803: 1,
6.008969816461896: 1, 6.008019784467605: 1, 6.000821113477028: 1, 5.987316655573658: 1,
5.986332451253398: 1, 5.983726377355223: 1, 5.980128063744171: 1, 5.979888632223399: 1,
5.973387807610976: 1, 5.970098899666936: 1, 5.9669654464263875: 1, 5.958187440629817: 1,
5.955757656428958: 1, 5.9550668768575274: 1, 5.948304837854494: 1, 5.945226253332939: 1,
5.938513200055589: 1, 5.935518291605116: 1, 5.930707629449545: 1, 5.923419549993868: 1,
5.919856093966111: 1, 5.916948848429369: 1, 5.913068888054981: 1, 5.909976160039123: 1,
5.907702563231978: 1, 5.904078556657456: 1, 5.896420408935932: 1, 5.895178780616053: 1,
5.892257787762262: 1, 5.884944989670723: 1, 5.882043468055985: 1, 5.879991530120707: 1,
5.879761970949384: 1, 5.878681043868348: 1, 5.876817993718315: 1, 5.871280445902944: 1,
5.870498458598765: 1, 5.849346403872222: 1, 5.841788635300087: 1, 5.841004535816932: 1,
5.839451121618091: 1, 5.8369408523190005: 1, 5.834476885310511: 1, 5.825611314815662: 1,
5.813554326119632: 1, 5.803946207861432: 1, 5.798008780455287: 1, 5.791318597380873: 1,
5.7779499548071S8: 1, 5.77645389269861: 1, 5.771569589889507: 1, 5.7699369810650625: 1,
5.765118530212562: 1, 5.764279789416571: 1, 5.764276886145091: 1, 5.756618766628407: 1,
5.755527570761063: 1, 5.735076854280258: 1, 5.7339541642709735: 1, 5.73142354623241: 1,
5.729542924876505: 1, 5.726245915707945: 1, 5.72544725248582: 1, 5.716009125707371: 1,
5.715777642601618: 1, 5.706194659615758: 1, 5.705266519054878: 1, 5.703375443364065: 1,
5.703132850186903: 1, 5.697267613342563: 1, 5.693047257509241: 1, 5.688873479797I8: 1,
5.6859890472593415: 1, 5.683885362700S035: 1, 5.682431392730688: 1, 5.680368778095852: 1,

5.6783382870364O1: 1, 5.668404347034375: 1, 5.6670727673070B4: 1, 5.6655376938041B7: 1,
5.662960036730З4: 1, 5.65686032988019: 1, 5.6511201042227635: 1, 5.648234125243009: 1,
5.644586336322174: 1, 5.642094962143141: 1, 5.641071536310223: 1, 5.640283314360884: 1,
5.637364162139B3: 1, 5.630528889636S4: 1, 5.629048031312951: 1, 5.6269369779361895: 1,
5.615208816814434: 1, 5.614762003792651: 1, 5.6115346813658435: 1, 5.603294776957047: 1,
5.603143219313755: 1, 5.602471583373899: 1, 5.602288107611447: 1, 5.6020599403688545: 1,
5.599907144758189: 1, 5.599167478090768: 1, 5.594992684952171: 1, 5.588014431333455: 1,
5.587818884415956: 1, 5.579681320225463: 1, 5.575709883214061: 1, 5.566395515717655: 1,
5.565506389276766: 1, 5.562190127714855: 1, 5.549276546238932: 1, 5.545145664647202: 1,
5.543575244327559: 1, 5.540742653532958: 1, 5.535502083791321: 1, 5.533653112970924: 1,
5.531176918524097: 1, 5.526793794249293: 1, 5.512148323031815: 1, 5.509411385786965: 1,
5.508614911921077: 1, 5.508562994626044: 1, 5.502273917924131: 1, 5.496262765818317: 1,
5.496195545789199: 1, 5.489678493732357: 1, 5.483986379434311: 1, 5.483871701405674: 1,
5.4766980555591935: 1, 5.475441734404668: 1, 5.464124286172236: 1, 5.460129847854868: 1, 5.45412920
5684028: 1, 5.453773234877123: 1, 5.448804685933597: 1, 5.444355647229234: 1, 5.440931053070483: 1,
5.435703470704568: 1, 5.43423740074222: 1, 5.433583055612992: 1, 5.43177298200418: 1,
5.431281686196108: 1, 5.430965166814735: 1, 5.4253828619522055: 1, 5.423416601581893: 1,
5.42276092221522: 1, 5.421248299527289: 1, 5.420033492336989: 1, 5.410636830180657: 1,
5.409368901951283: 1, 5.404235570361404: 1, 5.402793708644665: 1, 5.402583688650929: 1,
5.400527479558191: 1, 5.396338684546884: 1, 5.387132176399775: 1, 5.38642124592444: 1,
5.381864538972697: 1, 5.376499634371092: 1, 5.375788021768519: 1, 5.367172807496589: 1,
5.366819016505077: 1, 5.361150394192562: 1, 5.360404177420664: 1, 5.353461038389598: 1,
5.353454605798932: 1, 5.351367565902264: 1, 5.349543936883206: 1, 5.34839515777052: 1,
5.348167091943649: 1, 5.336553172790405: 1, 5.319489051087664: 1, 5.317021631283151: 1,
5.314631607775901: 1, 5.312640577837038: 1, 5.311109333828458: 1, 5.310830431672981: 1,
5.301480348640596: 1, 5.29977552116331: 1, 5.29685448286828: 1, 5.296663635212143: 1,
5.295199797120771: 1, 5.293731624498991: 1, 5.28330123099061: 1, 5.2716867181473I8: 1,
5.270592591281242: 1, 5.269762637600788: 1, 5.268568310933913: 1, 5.2657611764798045: 1,
5.256033875902162: 1, 5.2545326418558425: 1, 5.251859377185303: 1, 5.249296055960755: 1,
5.246758091418884: 1, 5.244780807150424: 1, 5.242754073687961: 1, 5.242723998050775: 1,
5.238532588267058: 1, 5.226148941014462: 1, 5.219559477542952: 1, 5.21686995333661B5: 1,
5.21181773172802: 1, 5.211283188562362: 1, 5.210248856767714: 1, 5.198892975466593: 1,
5.196324731150585: 1, 5.194895144282I6: 1, 5.191312532283574: 1, 5.190405433090889: 1,
5.189899873147247: 1, 5.186395905240633: 1, 5.1863617876659545: 1, 5.183168971359888: 1,
5.181231316591093: 1, 5.172844812993093: 1, 5.170718905642906: 1, 5.161431909051438: 1,
5.160777384604555: 1, 5.154834048598I8: 1, 5.154315603496348: 1, 5.153703401057188: 1,
5.150925249719406: 1, 5.148839441995435: 1, 5.145969448193I5: 1, 5.145032171455171: 1,
5.144075507587105: 1, 5.139419903539078: 1, 5.1386177340137715: 1, 5.136371350198115: 1,
5.134009184065539: 1, 5.132271746234319: 1, 5.130003377901269: 1, 5.127218793473597: 1,
5.126853866519967: 1, 5.119180969489101: 1, 5.11874671067986: 1, 5.117311885270465: 1,
5.116007007794829: 1, 5.115053613478111: 1, 5.107117810445458: 1, 5.104759985782161: 1,
5.102466669047116: 1, 5.102000040597626: 1, 5.100710581876617: 1, 5.099027511287633: 1,
5.092505745476237: 1, 5.090756682371245: 1, 5.086907396427491: 1, 5.081467851309668: 1,
5.080148832625692: 1, 5.077687445112377: 1, 5.075009629868765: 1, 5.074997062771201: 1,
5.069508234009916: 1, 5.067142317147617: 1, 5.062184634146901: 1, 5.055648589979273: 1,
5.052645217381951: 1, 5.052477857016623: 1, 5.049875639178115: 1, 5.042180921024982: 1,
5.041493074437811: 1, 5.039992432231709: 1, 5.031021441838694: 1, 5.029295186154419: 1,
5.0291336255543095: 1, 5.029118182173584: 1, 5.026252429023694: 1, 5.024660740731447: 1, 5.02167433
4626385: 1, 5.017932503422412: 1, 5.017902364521263: 1, 5.016735614186167: 1, 5.016583645230935: 1,
5.014095500797037: 1, 5.013731644930394: 1, 5.0101245926728515: 1, 5.0081031267172165: 1,
5.007606001869011: 1, 5.00262625393234: 1, 4.999589047757388: 1, 4.995895365360378: 1,
4.994524198583678: 1, 4.989620321253432: 1, 4.983232394898927: 1, 4.980035823702426: 1,
4.978627479027298: 1, 4.977934914819871: 1, 4.974406175950743: 1, 4.969493941831343: 1,
4.96417664994055: 1, 4.9628847629244: 1, 4.962769084170481: 1, 4.953952697239474: 1,
4.951281878814001: 1, 4.949493230606202: 1, 4.948570207499323: 1, 4.945840999734784: 1,
4.943277597711329: 1, 4.93753508518163: 1, 4.931066071776522: 1, 4.930804657021749: 1,
4.930075796751298: 1, 4.9266612296801835: 1, 4.923818051281539: 1, 4.918560820998372: 1,
4.9180375778496: 1, 4.908137697451077: 1, 4.90607335521744: 1, 4.905635619383458: 1,
4.904957183363852: 1, 4.9039089362889206: 1, 4.903122037715364: 1, 4.897898781768788: 1,
4.897400435209831: 1, 4.891512780790314: 1, 4.889933799820027: 1, 4.887647566960464: 1,
4.887539766906434: 1, 4.884448894442626: 1, 4.8793019893557545: 1, 4.867782286217855: 1,
4.866929760979443: 1, 4.8656800895746155: 1, 4.863076639575179: 1, 4.860586074838851: 1,
4.859349490793872: 1, 4.8583692103202605: 1, 4.855976414983221: 1, 4.8418267778054B4: 1,
4.8375662726054065: 1, 4.836839817623921: 1, 4.836684594203501: 1, 4.8354222146793O5: 1,
4.832697296290885: 1, 4.827938332648816: 1, 4.827872680487593: 1, 4.8242157741670I9: 1,
4.819599977285252: 1, 4.8195726370478305: 1, 4.818424381160033: 1, 4.814721789090117: 1,
4.810341179364851: 1, 4.809779918918377: 1, 4.807805685035716: 1, 4.8044348714233I9: 1,
4.800940783055644: 1, 4.796894433464302: 1, 4.7954763577364075: 1, 4.794446829850678: 1,
4.788891758040848: 1, 4.786816875876396: 1, 4.783035716895871: 1, 4.780624641603I4: 1,
4.778581833654088: 1, 4.777326989267754: 1, 4.777319650573472: 1, 4.777214396005987: 1,
4.768384624707461: 1, 4.7632226375474715: 1, 4.762579107521667: 1, 4.761073401253865: 1,
4.757541247339039: 1, 4.752999163100775: 1, 4.738169989356533: 1, 4.7378694672204S5: 1,
4.735937527520966: 1, 4.732406322824909: 1, 4.716860712966727: 1, 4.71425694994010I5: 1,
4.712096877159351: 1, 4.710469964154914: 1, 4.709707184103994: 1, 4.709269290207246: 1,
4.707662249754717: 1, 4.7034445141809975: 1, 4.7015698897435: 1, 4.701028281422984: 1,
4.700525607909655: 1, 4.69702462319084: 1, 4.695207767114224: 1, 4.68686716221422: 1,
4.685266489009479: 1, 4.678946760848311: 1, 4.6784382928421016: 1, 4.6769348696538895: 1,

4.6763561568171 67: 1, 4.67356769710201: 1, 4.665717896772664: 1, 4.6644967564370114: 1,
4.6623969990475445: 1, 4.660189835785779: 1, 4.658186519648474: 1, 4.654488948306224: 1, 4.64987644
6074931: 1, 4.64743761933614: 1, 4.6466383406406235: 1, 4.641740971788985: 1, 4.640067209453662:
1, 4.63475272197047: 1, 4.634642339018799: 1, 4.6318218856214814: 1, 4.631393157276 22: 1,
4.626045507148748: 1, 4.6222204172519135: 1, 4.62074458710015: 1, 4.620701460283205: 1,
4.615517167738533: 1, 4.615112726558051: 1, 4.612069827554093: 1, 4.611409056897478: 1,
4.610234707844601: 1, 4.608859752504524: 1, 4.60725988357679: 1, 4.600392749493639: 1,
4.5994839187292005: 1, 4.594076011782028: 1, 4.592696983744408: 1, 4.589882223712678: 1, 4.58856138
4776716: 1, 4.5871649483162305: 1, 4.586708951252754: 1, 4.583393251104992: 1, 4.582044133839339:
1, 4.577728045005919: 1, 4.576085979033095: 1, 4.56611423408101: 1, 4.563663988471053: 1,
4.562363043420425: 1, 4.554042655378764: 1, 4.549878914389996: 1, 4.549328334269894: 1,
4.542283609469717: 1, 4.537906051449174: 1, 4.530511586211784: 1, 4.5248440591348675: 1,
4.520362461478 45: 1, 4.519409844910675: 1, 4.514395683576526: 1, 4.500093914851594: 1,
4.495897861543564: 1, 4.492951387190735: 1, 4.4915638942722405: 1, 4.491439198848068: 1,
4.485828423433195: 1, 4.4837449921193215: 1, 4.47361890335818: 1, 4.4726704745571 54: 1,
4.472227269854398: 1, 4.470677445201177: 1, 4.450018458991706: 1, 4.4473336313416 69: 1,
4.443836353793821: 1, 4.442006603050659: 1, 4.441142745984019: 1, 4.4381960989223 42: 1,
4.436980549001522: 1, 4.4320686989517375: 1, 4.4306047036047405: 1, 4.4249049965546785: 1,
4.4248092544244715: 1, 4.423212914765763: 1, 4.420796788166639: 1, 4.4173328181692 99: 1, 4.41550142
2450678: 1, 4.411061879460371: 1, 4.41015243064947: 1, 4.403156997431729: 1, 4.399383200200714: 1,
4.398644629252382: 1, 4.397269170313374: 1, 4.395183506522522: 1, 4.389637043704488: 1,
4.388850632344407: 1, 4.388408511869737: 1, 4.388342680259485: 1, 4.383371406454342: 1,
4.3820437015701845: 1, 4.3817044097050495: 1, 4.381090569184448: 1, 4.3799590542186 75: 1,
4.3796382231689615: 1, 4.372103753273835: 1, 4.3719394960458 18: 1, 4.3695874084727535: 1,
4.364022834180881: 1, 4.363349287748394: 1, 4.362797892230584: 1, 4.3598598995700 98: 1,
4.355401020079885: 1, 4.354178130218184: 1, 4.3533767197905915: 1, 4.3472605875171 39: 1,
4.347150278385738: 1, 4.345492388951265: 1, 4.343586014236683: 1, 4.3432554347380 04: 1,
4.340457292093092: 1, 4.338535130513305: 1, 4.336466613278833: 1, 4.3360880265733 68: 1,
4.330941325001489: 1, 4.319513198301525: 1, 4.3167846172176585: 1, 4.313298257001 25: 1,
4.311796578901513: 1, 4.311170256535495: 1, 4.3092664128788 84: 1, 4.3081181821298 85: 1,
4.306288640131159: 1, 4.305987439721851: 1, 4.305327745390036: 1, 4.3031022535278085: 1,
4.302466255533361: 1, 4.2961593820927: 1, 4.2898088617155095: 1, 4.283152986854322: 1,
4.282530462987633: 1, 4.281316579388311: 1, 4.278411110467004: 1, 4.2752205962637095: 1,
4.272260157856016: 1, 4.270579989886548: 1, 4.268131812408914: 1, 4.2652971912899 13: 1,
4.262955105362999: 1, 4.262780247970115: 1, 4.260481746559271: 1, 4.2578968485972 66: 1,
4.256391942574655: 1, 4.256169087522439: 1, 4.253405602417983: 1, 4.2509721571335 42: 1,
4.249341795841634: 1, 4.246491714660945: 1, 4.244505152245974: 1, 4.2431132165880 16: 1,
4.2426059385535355: 1, 4.23973706807475: 1, 4.238760799234933: 1, 4.2366959236808 62: 1,
4.229640835366545: 1, 4.229521889223298: 1, 4.228450100366967: 1, 4.2268982942822 24: 1,
4.223381866723567: 1, 4.2230307809878145: 1, 4.2212177357572696: 1, 4.221145460898 02: 1,
4.220252222166631: 1, 4.214183784367761: 1, 4.203385270542271: 1, 4.1989648960032 52: 1,
4.193090030211632: 1, 4.19116320919066: 1, 4.1866651052288795: 1, 4.1852321411469 97: 1,
4.17765096835484: 1, 4.172813428207236: 1, 4.167637983642272: 1, 4.1669361369093 93: 1,
4.164802515662211: 1, 4.162553757475 56: 1, 4.1624580783238905: 1, 4.1580418551090 45: 1,
4.156476016626708: 1, 4.155312878690391: 1, 4.152928183840095: 1, 4.1522904775032 56: 1,
4.1507160444118325: 1, 4.1452319752243385: 1, 4.123052048836899: 1, 4.1221648434520 98: 1,
4.117865327504909: 1, 4.116883522461716: 1, 4.113879577335912: 1, 4.1122235061280925: 1,
4.1058988541751384: 1, 4.085906228134855: 1, 4.0844795822074005: 1, 4.083639277825794: 1,
4.081096877244098: 1, 4.080624883085019: 1, 4.078964934562453: 1, 4.0745665277840 36: 1,
4.073629353191114: 1, 4.070032669665863: 1, 4.069048766679572: 1, 4.0677156274292 04: 1,
4.064379435870314: 1, 4.059659444495902: 1, 4.054858084666943: 1, 4.051917244919 85: 1,
4.048911368615294: 1, 4.0456137116695094: 1, 4.045613641104494: 1, 4.0453397608291 41: 1,
4.044665042295297: 1, 4.042552078857285: 1, 4.038520786379703: 1, 4.0301419917169 45: 1,
4.022864955588053: 1, 4.022569481008556: 1, 4.021925496373085: 1, 4.0217627683586 76: 1,
4.021060938541769: 1, 4.018844016861123: 1, 4.009416588499615: 1, 4.0085394986262 54: 1,
4.006498063557107: 1, 4.004226031722624: 1, 3.9988787494180733: 1, 3.9983676188294117: 1,
3.9974804355411884: 1, 3.9968057256040974: 1, 3.994436988619486: 1, 3.9917791337759754: 1,
3.9729963298569264: 1, 3.965754018856519: 1, 3.9624671762941626: 1, 3.960386950706 55: 1,
3.958016272234097: 1, 3.9560022754630255: 1, 3.9543147457210006: 1, 3.9475217342126303: 1,
3.945263337235 92: 1, 3.9400487742175936: 1, 3.939010671580001: 1, 3.9380812823828 28: 1,
3.9379841018161805: 1, 3.937398737700175: 1, 3.9322222552781922: 1, 3.9303489157859 21: 1,
3.926890492766075: 1, 3.92671746624559: 1, 3.9183185554552113: 1, 3.9151225769284 56: 1,
3.9074290636067914: 1, 3.904539532809201: 1, 3.9031960715127525: 1, 3.9022519586708 3: 1,
3.8936071159230723: 1, 3.8874594228978543: 1, 3.8846430577683693: 1, 3.8842139395645 78: 1,
3.882236024708117: 1, 3.8788882565525045: 1, 3.874164579146062: 1, 3.8667446004626993: 1,
3.859308646773938: 1, 3.853094735496538: 1, 3.8446731805669736: 1, 3.8404142690267222: 1,
3.8387245798217258: 1, 3.830179786187268: 1, 3.82561770868013: 1, 3.8239087217431 07: 1,
3.822583093646429: 1, 3.8203504873501624: 1, 3.8127089160049334: 1, 3.8101827692467 38: 1,
3.8012881742869498: 1, 3.798094531277684: 1, 3.7979309118059543: 1, 3.796481497903 78: 1,
3.7914742607897916: 1, 3.7842462724625716: 1, 3.7829698802370837: 1, 3.7810253951181068: 1,
3.779336110428287: 1, 3.7752804156925706: 1, 3.7718081462272357: 1, 3.7713385677058544: 1,
3.7679623268623943: 1, 3.7669429286348888: 1, 3.764384772021873: 1, 3.7640957129837145: 1,
3.763767884571313: 1, 3.757826143784568: 1, 3.755425946381367: 1, 3.7497548829851604: 1,
3.7413612325472414: 1, 3.7395892660350216: 1, 3.7394936419944664: 1, 3.7352635314949 02: 1,
3.7228548483379638: 1, 3.7217222428607486: 1, 3.7188956080551603: 1, 3.7183359200927 16: 1,
3.713221744526184: 1, 3.702932470152858: 1, 3.7018063253420626: 1, 3.7011631832058685: 1,
3.69477427073535: 1, 3.693814345085112: 1, 3.687309751532212: 1, 3.6726681917794455: 1,

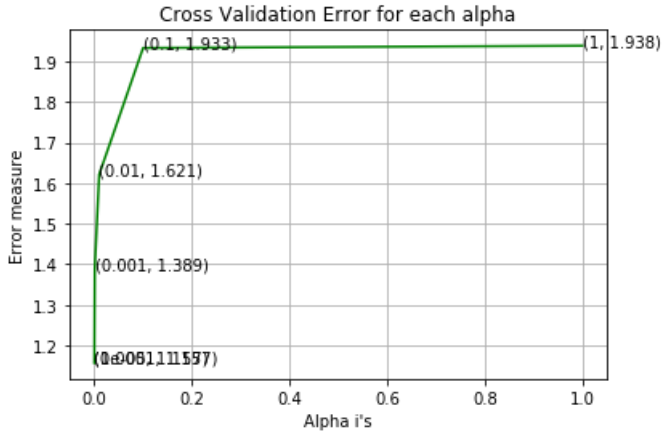
```
3.6712129514079193: 1, 3.6684419727034974: 1, 3.666022380181523: 1, 3.644569830083389: 1,
3.641969723167137: 1, 3.641615144420881: 1, 3.6286368674695297: 1, 3.62341659910993: 1,
3.6206077220540167: 1, 3.6118228639343286: 1, 3.605991180325432: 1, 3.602835512192168: 1,
3.6014383390163798: 1, 3.5890808528368194: 1, 3.5863319333581924: 1, 3.5852613626922754: 1,
3.5797747506672235: 1, 3.5784115389958333: 1, 3.571054784649513: 1, 3.565170412613919: 1,
3.5542272769361953: 1, 3.55021284020216: 1, 3.5404270677734133: 1, 3.5365619532570776: 1,
3.5337196721183033: 1, 3.5318550243622395: 1, 3.5252304551219638: 1, 3.5242870768348245: 1,
3.5191111066654996: 1, 3.5178163204647195: 1, 3.5148032643324294: 1, 3.504509913504955: 1,
3.4987525726777697: 1, 3.490502918810465: 1, 3.4753373256050906: 1, 3.472172006989067: 1,
3.4692064159035723: 1, 3.4674449156140197: 1, 3.465965976462892: 1, 3.4498390828226513: 1,
3.4479060280209493: 1, 3.4455654696628866: 1, 3.4454141590716776: 1, 3.4425188077034687: 1,
3.438389312590644: 1, 3.4381554593194688: 1, 3.4380411447079267: 1, 3.431817412593107: 1,
3.427310348567626: 1, 3.4257572700266143: 1, 3.414505188130077: 1, 3.409973510004126: 1,
3.409768744492449: 1, 3.407469561477794: 1, 3.403079318129928: 1, 3.3910346685850006: 1,
3.3858834432756675: 1, 3.3822225620970547: 1, 3.379801398440211: 1, 3.378839802615931: 1,
3.3575737249665702: 1, 3.354575389557075: 1, 3.349776150102972: 1, 3.3453535417655127: 1,
3.3347763652227624: 1, 3.3234951317721615: 1, 3.3122990758531197: 1, 3.308263454865639: 1,
3.3070467487868185: 1, 3.304539012142637: 1, 3.2994528872858275: 1, 3.292376392242798: 1,
3.2903258890727356: 1, 3.2808455916659254: 1, 3.275681295167738: 1, 3.2754547135440126: 1,
3.267782055829977: 1, 3.2636454875130103: 1, 3.2551985531980248: 1, 3.2527868118837953: 1,
3.20327472692304: 1, 3.1436119189860676: 1, 3.130156193684464: 1, 3.120825199665926: 1,
2.9362500530875026: 1, 2.918470248780679: 1, 2.90397863667126: 1, 2.87200487337829: 1,
2.832519459879881: 1, 2.757233222299743: 1})
```

In [97]:

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

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

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

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


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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.1566428130797404
For values of alpha =  0.0001 The log loss is: 1.156869942130933
For values of alpha =  0.001 The log loss is: 1.388916077145637
For values of alpha =  0.01 The log loss is: 1.6209630483478343
For values of alpha =  0.1 The log loss is: 1.9328103552385318
For values of alpha =  1 The log loss is: 1.938361542533854
```




```
For values of best alpha =  1e-05 The train log loss is: 0.801874584220678
For values of best alpha =  1e-05 The cross validation log loss is: 1.1566428130797404
For values of best alpha =  1e-05 The test log loss is: 0.999782757456051
```

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

Ans. Yes, it seems like!

In [98]:

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

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

In [99]:

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

```
7.059 % of word of test data appeared in train data
7.865 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [100]:

```
#Data preparation for ML models.

#Misc. functionns for ML models


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

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

In [101]:

```
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 [102]:

```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3,ngram_range=(1,2))

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

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

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

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

In [103]:

```
# merging gene, variance and text features

# building train, test and cross validation data sets
# a = [[1, 2],
```

```
#       [3, 4]]
# b = [[4, 5],
#      [6, 7]]
# hstack(a, b) = [[1, 2, 4, 5],
#                 [ 3, 4, 6, 7]]

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,
                                      train_variation_feature_onehotCoding))

test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,
                                     test_variation_feature_onehotCoding))

cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,
                                   cv_variation_feature_onehotCoding))


train_x_onehotCoding = hstack((train_gene_var_onehotCoding,
                               train_text_feature_onehotCoding)).tocsr()

train_y = np.array(list(train_df['Class']))

test_x_onehotCoding = hstack((test_gene_var_onehotCoding,
                              test_text_feature_onehotCoding)).tocsr()

test_y = np.array(list(test_df['Class']))

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding,
                            cv_text_feature_onehotCoding)).tocsr()

cv_y = np.array(list(cv_df['Class']))


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,
                                           train_variation_feature_responseCoding))

test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,
                                          test_variation_feature_responseCoding))

cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,
                                        cv_variation_feature_responseCoding))

train_x_responseCoding = np.hstack((train_gene_var_responseCoding,
                                    train_text_feature_responseCoding))

test_x_responseCoding = np.hstack((test_gene_var_responseCoding,
                                   test_text_feature_responseCoding))

cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding,
                                 cv_text_feature_responseCoding))
```

In [104]:

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

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 4190)
(number of data points * number of features) in test data =  (665, 4190)
(number of data points * number of features) in cross validation data = (532, 4190)
```

In [105]:

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

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 27)
```

```
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data = (532, 27)
```

Feature engineering on one hot encoded features

In [106]:

```
train_x_onehotCodingFE=np.sqrt(train_x_onehotCoding)
test_x_onehotCodingFE=np.sqrt(test_x_onehotCoding)
cv_x_onehotCodingFE=np.sqrt(cv_x_onehotCoding)
```

In [107]:

```
print("One hot encoding of features engineered features:")
print("(number of data points * number of features) in train data = ", train_x_onehotCodingFE.shap
e)
print("(number of data points * number of features) in test data = ", test_x_onehotCodingFE.shape)
print("(number of data points * number of features) in cross validation data =",
cv_x_onehotCodingFE.shape)
```

```
One hot encoding of features engineered features:
(number of data points * number of features) in train data =  (2124, 4190)
(number of data points * number of features) in test data =  (665, 4190)
(number of data points * number of features) in cross validation data = (532, 4190)
```

In [108]:

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

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

4.1. Base Line Model

4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [109]:

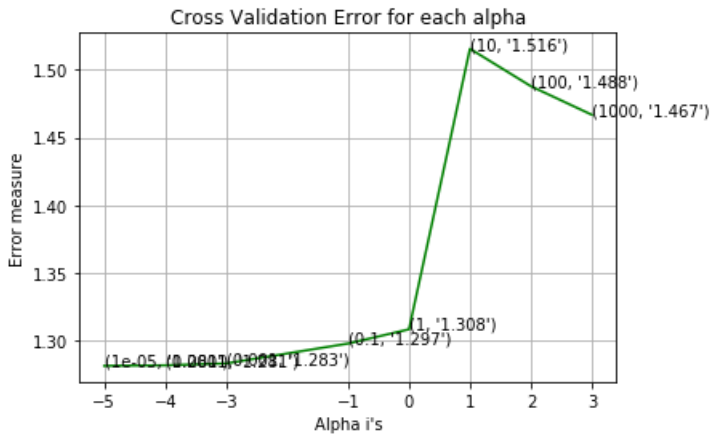
```
# find more about Multinomial Naive base function here http://scikit-
learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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

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


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

```
for alpha = 1e-05
Log Loss : 1.280948886174295
for alpha = 0.0001
Log Loss : 1.281344931734386
for alpha = 0.001
Log Loss : 1.2827335161885036
for alpha = 0.1
Log Loss : 1.2974585169960682
for alpha = 1
Log Loss : 1.308013379291052
for alpha = 10
Log Loss : 1.5157263060764643
for alpha = 100
Log Loss : 1.4879657908937827
for alpha = 1000
Log Loss : 1.4668097923142367
```

In [110]:

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



In [111]:

```
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)
```

Out[111]:

```
CalibratedClassifierCV(base_estimator=MultinomialNB(alpha=1e-05, class_prior=None,
fit_prior=True),
            cv=3, method='sigmoid')
```

In [112]:

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

```
For values of best alpha =  1e-05 The train log loss is: 0.5461223539894796
For values of best alpha =  1e-05 The cross validation log loss is: 1.280948886174295
For values of best alpha =  1e-05 The test log loss is: 1.1774021649764976
```

4.1.1.2. Testing the model with best hyper paramters

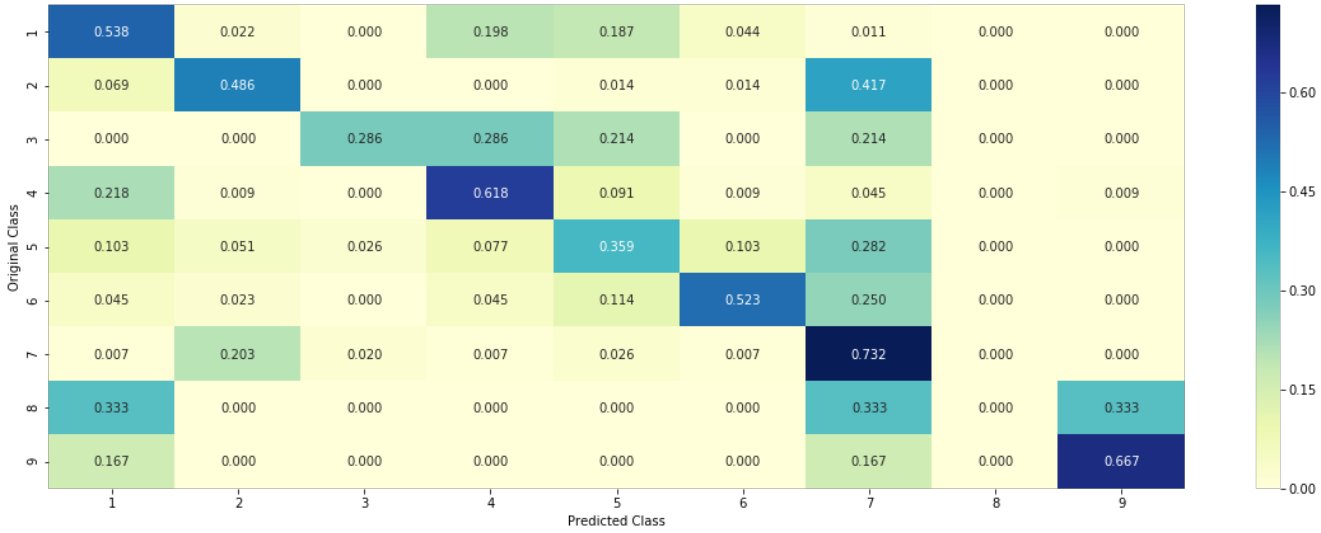
In [113]:

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

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


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

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

```
Log Loss : 1.280948886174295
Number of missclassified point : 0.4191729323308271
-------------------- Confusion matrix --------------------
```


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


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



4.1.1.3. Feature Importance, Correctly classified point

In [116]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0599 0.0482 0.0118 0.0729 0.0335 0.0341 0.7323 0.0039 0.0034]]
Actual Class : 7
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [117]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0598 0.0466 0.0117 0.0743 0.0334 0.034  0.733  0.0038 0.0034]]
Actual Class : 7
--------------------------------------------------
32 Text feature [009775] present in test data point [True]
78 Text feature [163950] present in test data point [True]
Out of the top  100  features  2 are present in query point
```

4.2. K Nearest Neighbour Classification

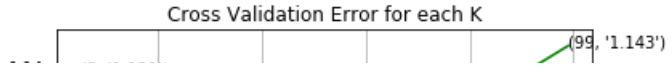
4.2.1. Hyper parameter tuning

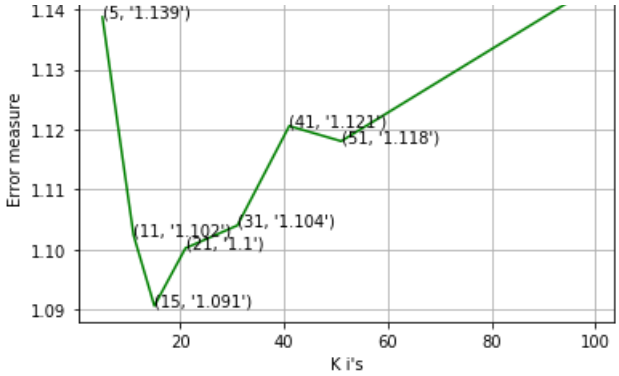
In [118]:

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

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

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


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

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


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

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best K = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, pre
dict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best K = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y
_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best K = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predi
ct_y, labels=clf.classes_, eps=1e-15))
```

```
for k = 5
Log Loss : 1.1386929517031628
for k = 11
Log Loss : 1.1024106710315047
for k = 15
Log Loss : 1.0906601236649491
for k = 21
Log Loss : 1.100256981420462
for k = 31
Log Loss : 1.1040404147241507
for k = 41
Log Loss : 1.1205903615043382
for k = 51
Log Loss : 1.118056778486448
for k = 99
Log Loss : 1.1434187667719888
```





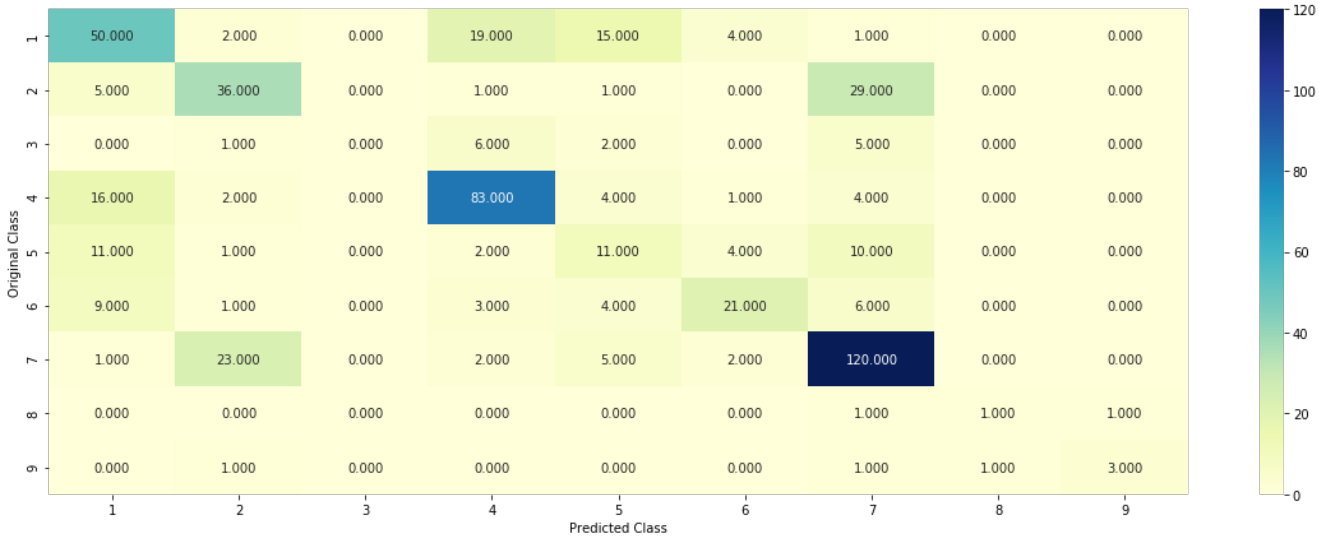

```
For values of best K =  15 The train log loss is: 0.7061777168724075
For values of best K =  15 The cross validation log loss is: 1.0906601236649491
For values of best K =  15 The test log loss is: 0.9902402237603753
```

4.2.2. Testing the model with best hyper paramters

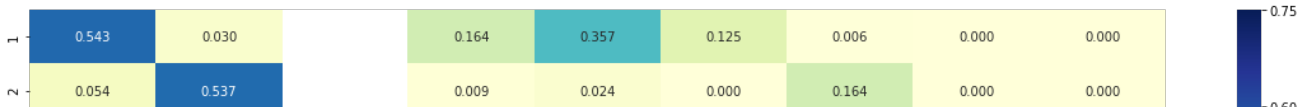
In [119]:

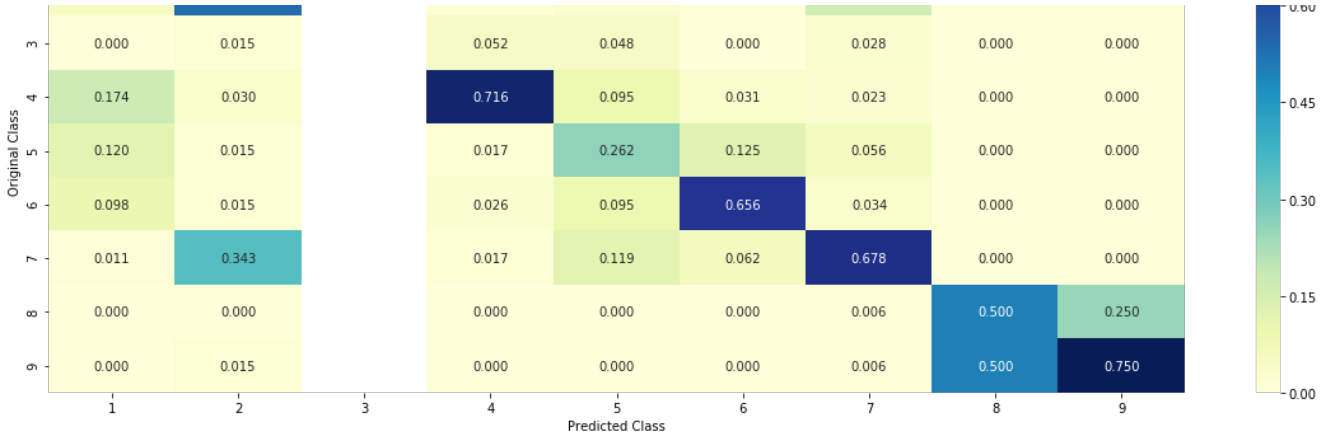
```
# find more about KNeighborsClassifier() here http://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski',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

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

```
Log loss : 1.0906601236649491
Number of mis-classified points : 0.3890977443609023
-------------------- Confusion matrix --------------------
```

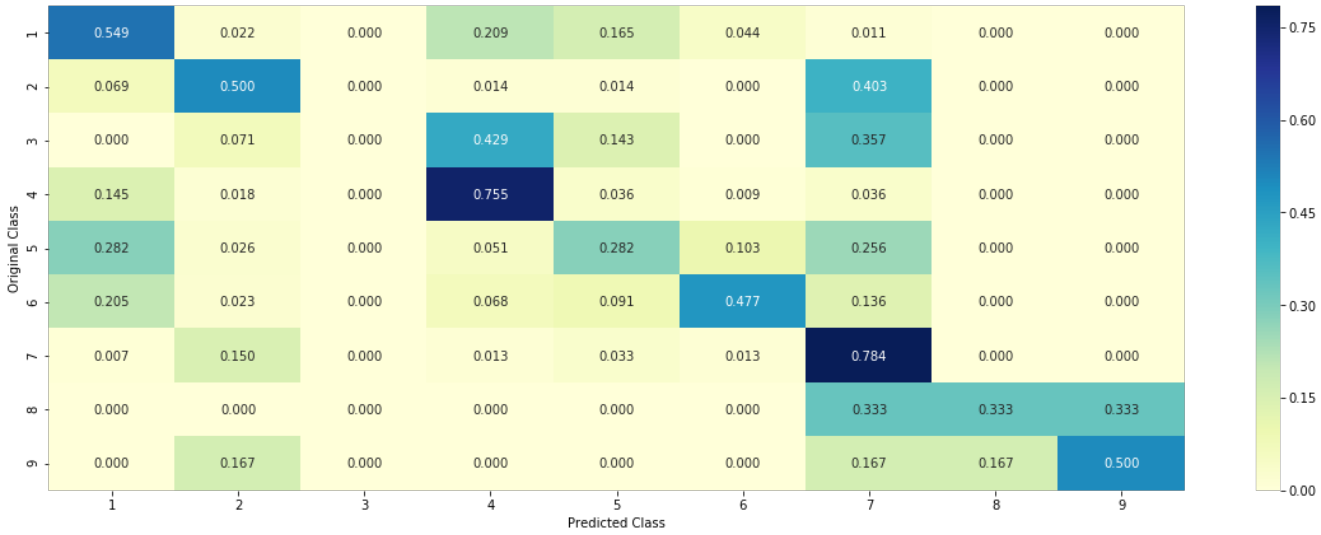



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



4.2.3.Sample Query point -1 for KNN

In [120]:

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

test_point_index = 1
predicted_cls = sig_clf.predict(test_x_responseCoding[0].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), alpha[best_alpha
])
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to classes",train_y
[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 7
The  15  nearest neighbours of the test points belongs to classes [7 7 7 7 7 7 7 7 7 7 7 7 7 2 7]
Fequency of nearest points : Counter({7: 14, 2: 1})
```

4.2.4. Sample Query Point-2 for KNN

In [121]:

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

```
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 100

predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), alpha[best_alpha
])
print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of the test points be
longs to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 7
the k value for knn is 15 and the nearest neighbours of the test points belongs to classes [2 7 7
7 7 7 7 2 7 7 7 7 7 4 7]
Fequency of nearest points : Counter({7: 12, 2: 2, 4: 1})
```

Task 3: Apply Logistic Regression with count vectorizer with unigram and bigram

4.3. Logistic Regression

In [133]:

```
#Making one_hot encoding features for logistic regression model by count vectorizer using unigram
and bigram

# one-hot encoding of Gene feature
gene_vectorizer_LR = CountVectorizer()
train_gene_feature_onehotCoding_LR = gene_vectorizer_LR.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding_LR = gene_vectorizer_LR.transform(test_df['Gene'])
cv_gene_feature_onehotCoding_LR = gene_vectorizer_LR.transform(cv_df['Gene'])

# one-hot encoding of variation feature.
variation_vectorizer_LR = CountVectorizer()
train_variation_feature_onehotCoding_LR =
variation_vectorizer_LR.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding_LR = variation_vectorizer_LR.transform(test_df['Variation'])
cv_variation_feature_onehotCoding_LR = variation_vectorizer_LR.transform(cv_df['Variation'])

#one-hot encoding for Text feature
text_vectorizer_LR = CountVectorizer(min_df=3,ngram_range=(1, 2))
train_text_feature_onehotCoding_LR = text_vectorizer_LR.fit_transform(train_df['TEXT'])
train_text_feature_onehotCoding_LR = normalize(train_text_feature_onehotCoding_LR ,axis=0)

test_text_feature_onehotCoding_LR = text_vectorizer_LR.transform(test_df['TEXT'])
test_text_feature_onehotCoding_LR = normalize(test_text_feature_onehotCoding_LR ,axis=0)

cv_text_feature_onehotCoding_LR = text_vectorizer_LR.transform(cv_df['TEXT'])
cv_text_feature_onehotCoding_LR = normalize(cv_text_feature_onehotCoding_LR ,axis=0)

#stacking all the features(gene,vartions,text of one-hot encoded)
train_gene_var_onehotCoding_LR = hstack((train_gene_feature_onehotCoding_LR
,train_variation_feature_onehotCoding_LR))
test_gene_var_onehotCoding_LR =
hstack((test_gene_feature_onehotCoding_LR,test_variation_feature_onehotCoding_LR))
cv_gene_var_onehotCoding_LR =
hstack((cv_gene_feature_onehotCoding_LR,cv_variation_feature_onehotCoding_LR))

train_x_onehotCoding_LR = hstack((train_gene_var_onehotCoding_LR,
train_text_feature_onehotCoding_LR)).tocsr()
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding_LR = hstack((test_gene_var_onehotCoding_LR, test_text_feature_onehotCoding_LR)
).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_onehotCoding_LR = hstack((cv_gene_var_onehotCoding_LR, cv_text_feature_onehotCoding_LR)).tocs
r()
```

```
cv_y = np.array(list(cv_df['Class']))

print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding_LR.sha
pe)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding_LR.shape
)
print("(number of data points * number of features) in cross validation data =",
cv_x_onehotCoding_LR.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 770060)
(number of data points * number of features) in test data =  (665, 770060)
(number of data points * number of features) in cross validation data = (532, 770060)
```

4.3.1. With Class balancing

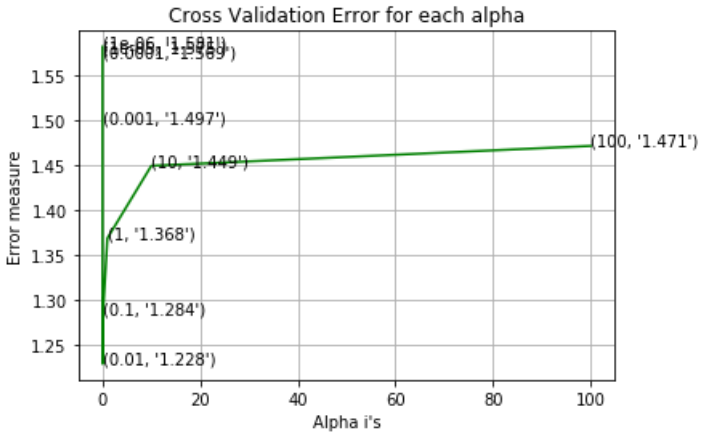
4.3.1.1. Hyper paramter tuning

In [134]:

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

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

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


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

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

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

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


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

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

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

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

```
for alpha = 1e-06
Log Loss : 1.5812971303552297
for alpha = 1e-05
Log Loss : 1.5754541997216394
for alpha = 0.0001
Log Loss : 1.5691393961180111
for alpha = 0.001
Log Loss : 1.4967345883579186
for alpha = 0.01
Log Loss : 1.2284416953850126
for alpha = 0.1
Log Loss : 1.2837734023476142
for alpha = 1
Log Loss : 1.3681827923621568
for alpha = 10
Log Loss : 1.4488148287559937
for alpha = 100
Log Loss : 1.470908374410702
```




```
For values of best alpha =  0.01 The train log loss is: 0.8489764307464034
For values of best alpha =  0.01 The cross validation log loss is: 1.2284416953850126
For values of best alpha =  0.01 The test log loss is: 1.1317105932669755
```

4.3.1.2. Testing the model with best hyper paramters

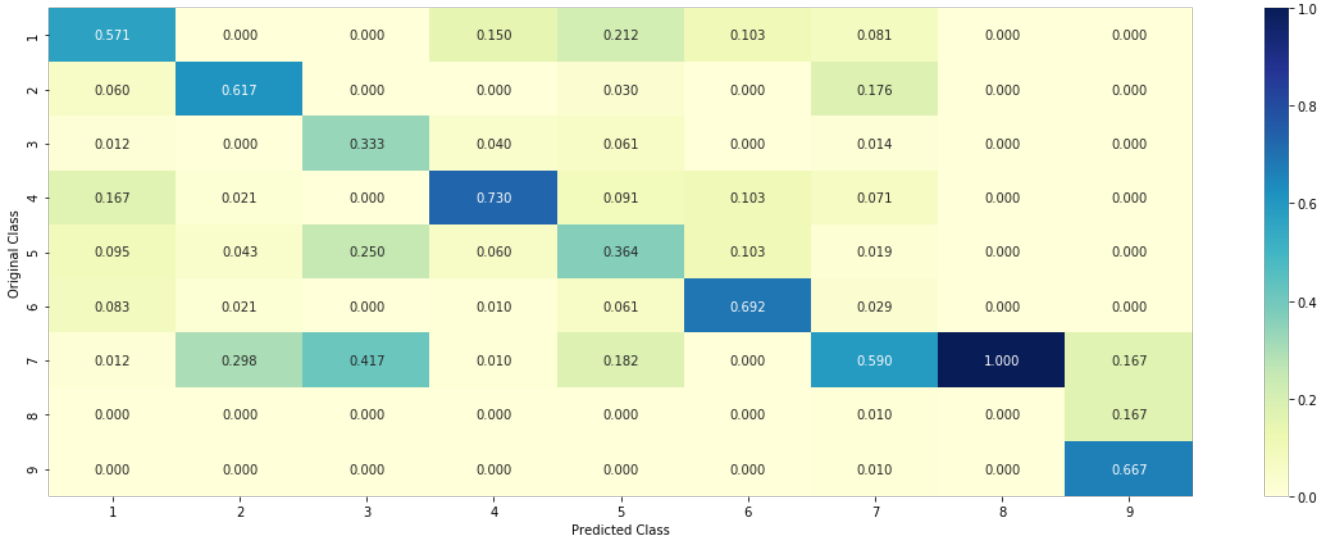
In [135]:

```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
```

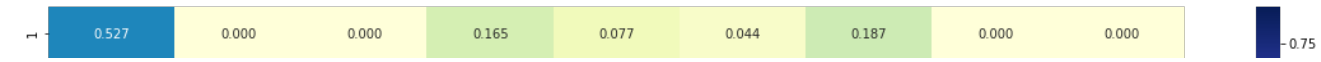
```
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

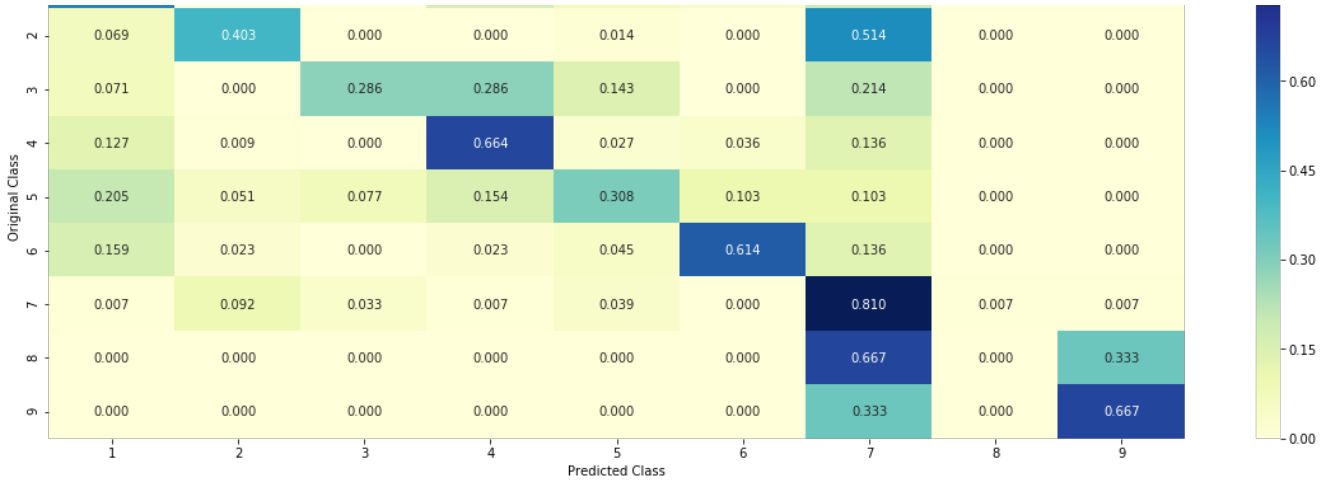
```
Log loss : 1.2284416953850126
Number of mis-classified points : 0.3966165413533835
-------------------- Confusion matrix --------------------
```


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




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





4.3.1.3. Feature Importance

In [136]:

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

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

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

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

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

4.3.1.3.1. Correctly Classified point

In [129]:

```
train_x_onehotCoding.shape
train_y.shape
```

Out[129]:

```
(2124,)
```

In [138]:

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
clf.fit(train_x_onehotCoding_LR,train_y)
test_point_index =1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_LR[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding_LR[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names_LR(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0148 0.0435 0.0025 0.0144 0.0138 0.003  0.8998 0.0052 0.0031]]
Actual Class : 7
--------------------------------------------------
369 Text feature [deregulated] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [140]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0406 0.1277 0.0127 0.0417 0.0347 0.0145 0.7136 0.0068 0.0077]]
Actual Class : 7
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.3.2. Without Class balancing

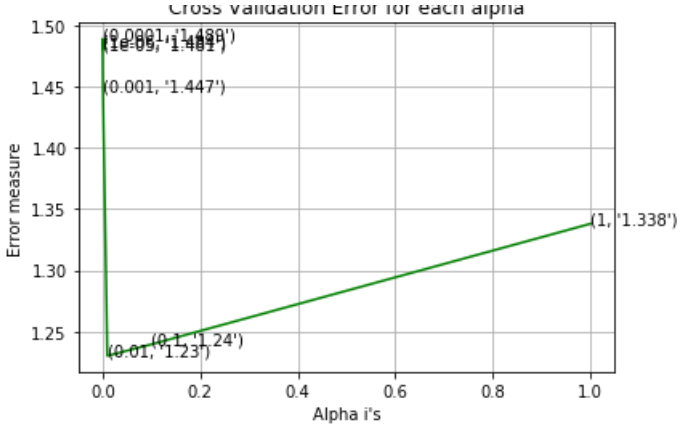
4.3.2.1. Hyper paramter tuning

In [141]:

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

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

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

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



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

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.4835450271542474
for alpha = 1e-05
Log Loss : 1.48108498881563
for alpha = 0.0001
Log Loss : 1.4892861030232736
for alpha = 0.001
Log Loss : 1.4467428208530542
for alpha = 0.01
Log Loss : 1.2301290832816334
for alpha = 0.1
Log Loss : 1.2395246521335315
for alpha = 1
Log Loss : 1.3380641122948627
```



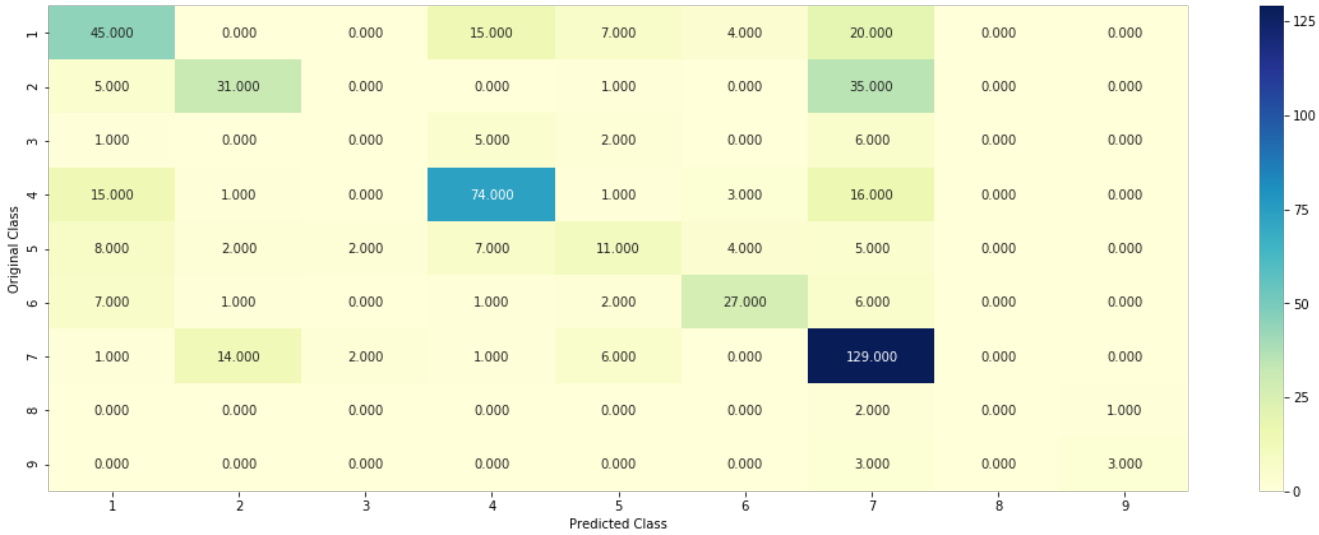

```
For values of best alpha =  0.01 The train log loss is: 0.8533386495248257
For values of best alpha =  0.01 The cross validation log loss is: 1.2301290832816334
For values of best alpha =  0.01 The test log loss is: 1.1485705963326276
```

4.3.2.2. Testing model with best hyper parameters

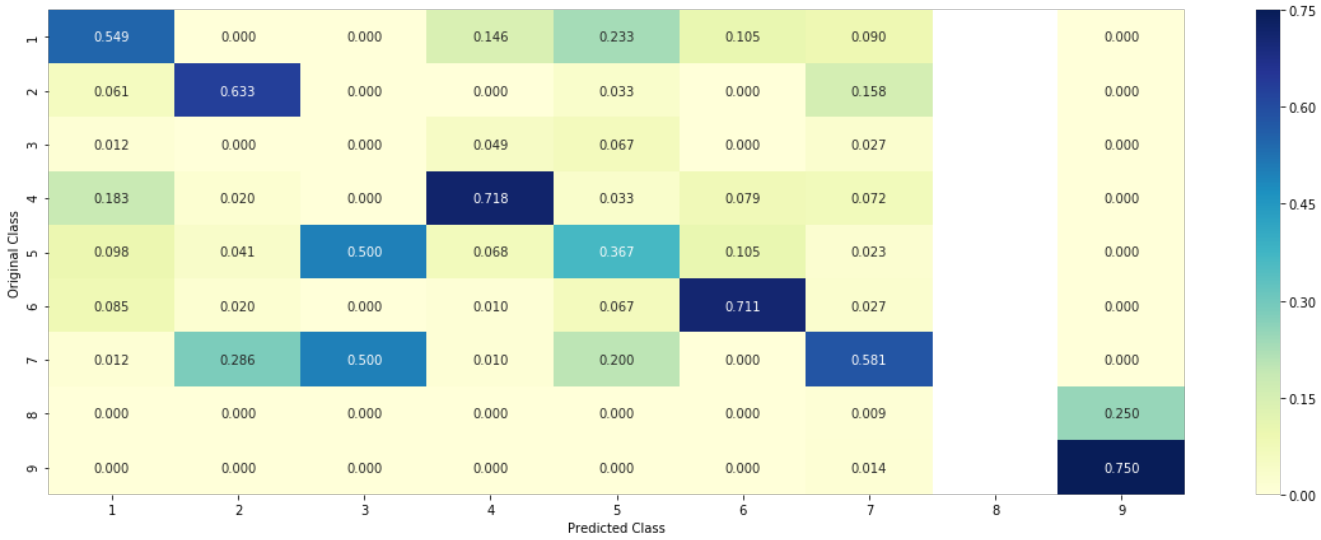
In [142]:

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

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

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

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding_LR, train_y, cv_x_onehotCoding_LR, cv_y, cl
f)
```

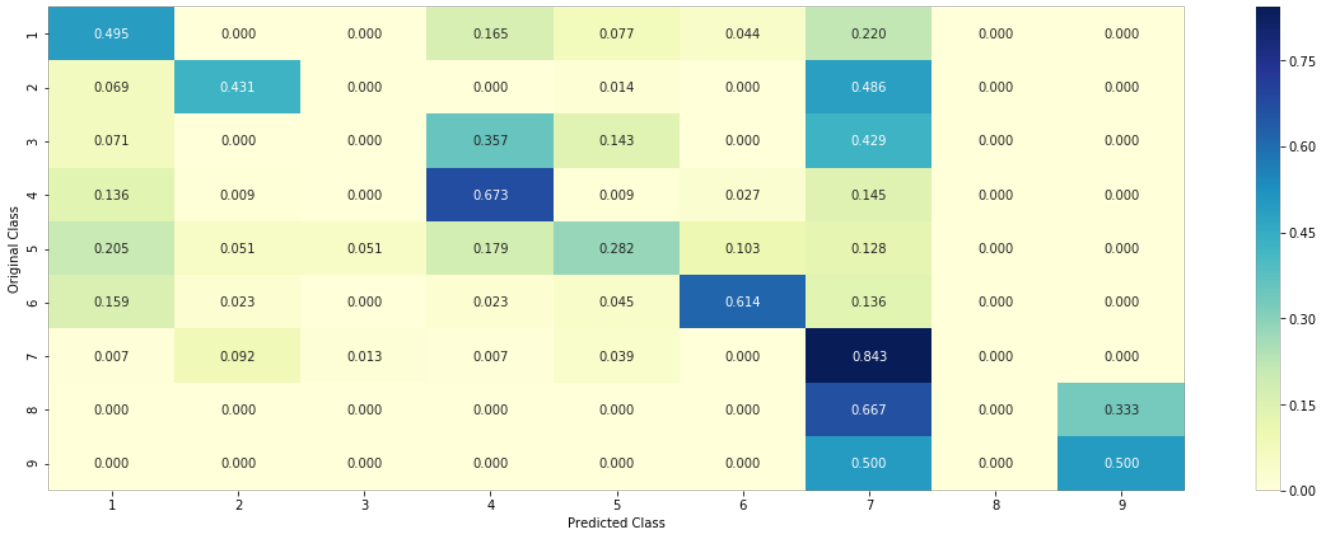
```
Log loss : 1.2301290832816334
Number of mis-classified points : 0.39849624060150374
-------------------- Confusion matrix --------------------
```




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




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



4.3.2.3. Feature Importance, Correctly Classified point

In [143]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0221 0.0476 0.0018 0.0214 0.01   0.0023 0.892  0.0016 0.0011]]
Actual Class : 7
--------------------------------------------------
412 Text feature [deregulated] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [95]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_LR[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding_LR[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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: [[4.600e-03 1.110e-02 6.000e-03 1.634e-01 2.500e-03 8.030e-01 6.100
e-03
  2.800e-03 6.000e-04]]
Actual Class : 6
--------------------------------------------------
69 Text feature [simplex] present in test data point [True]
214 Text feature [weakened] present in test data point [True]
215 Text feature [spermatogenesis] present in test data point [True]
243 Text feature [hospitals] present in test data point [True]
254 Text feature [s80] present in test data point [True]
392 Text feature [leaves] present in test data point [True]
496 Text feature [ccdc98] present in test data point [True]
Out of the top  500  features  7 are present in query point
```

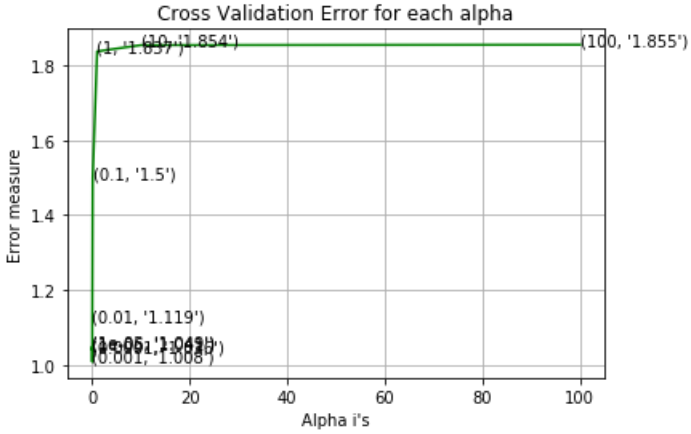
Logistic Regression using TFIDF vectorizer with feature engineering features

In [144]:

```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------

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

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCodingFE)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
```

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

```
for alpha = 1e-06
Log Loss : 1.0414656210661817
for alpha = 1e-05
Log Loss : 1.048969571261706
for alpha = 0.0001
Log Loss : 1.035949074641973
for alpha = 0.001
Log Loss : 1.0082205642755715
for alpha = 0.01
Log Loss : 1.1185278487364054
for alpha = 0.1
Log Loss : 1.499775089335978
for alpha = 1
Log Loss : 1.8374417191476866
for alpha = 10
Log Loss : 1.8541594247428932
for alpha = 100
Log Loss : 1.855203858668674
```

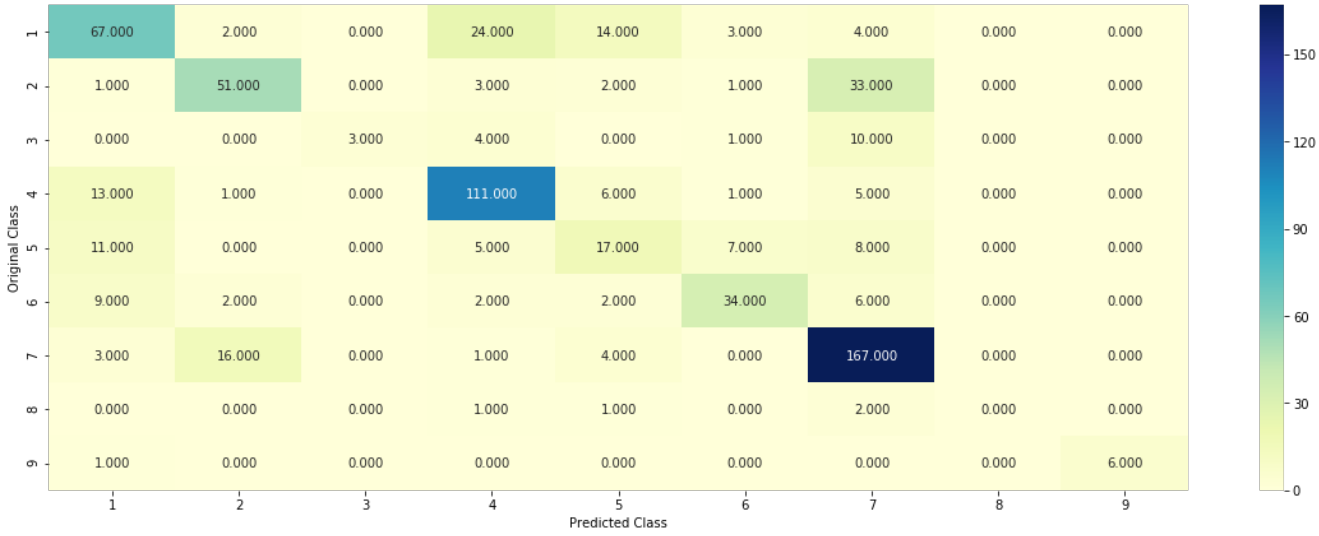



```
For values of best alpha =  0.001 The train log loss is: 0.6053482400111575
For values of best alpha =  0.001 The cross validation log loss is: 1.0082205642755715
For values of best alpha =  0.001 The test log loss is: 0.8791831559958935
```

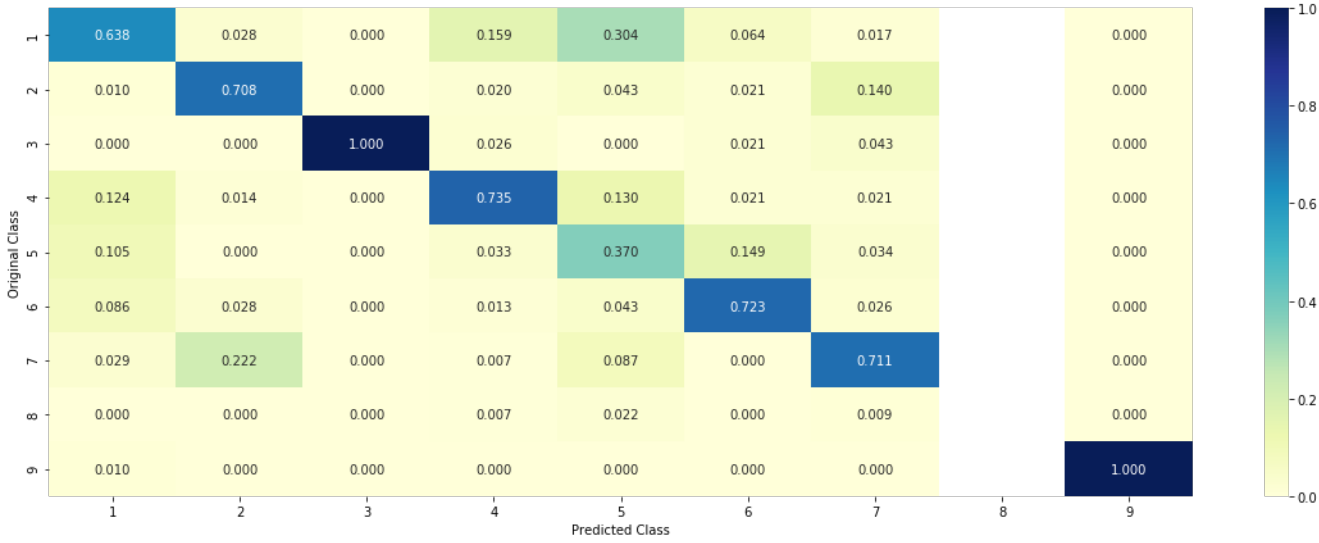
In [145]:

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

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

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

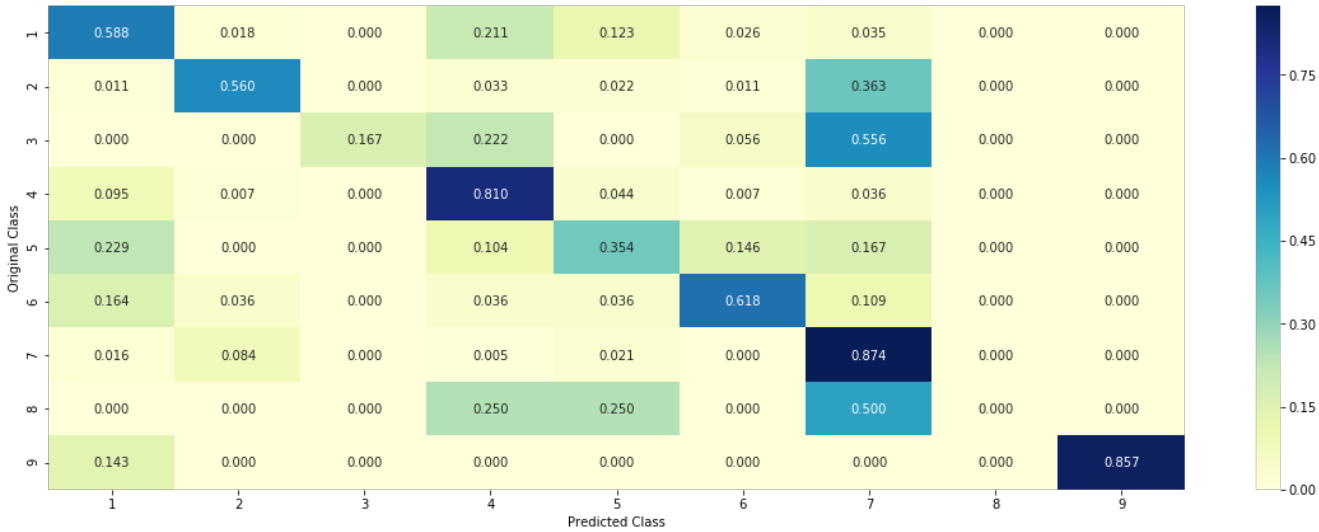
```
Log loss : 0.8791831559958935
Number of mis-classified points : 0.3142857142857143
-------------------- Confusion matrix --------------------
```




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




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



Obervation

Observation:

We use logistic regression using TFIDF vectorizer for all three feature such as Gene,Variation,Text and implemented feature engineering as square root of vectorizer and found log-loss is 0.879 and misclassified points are 0.314

Correctly Classified poin

In [147]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1392 0.0958 0.0726 0.162  0.0548 0.0735 0.3614 0.009  0.0318]]
Actual Class : 7
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

Incorrectly Classified point

In [148]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0834 0.1385 0.0385 0.1705 0.0484 0.0701 0.4169 0.0092 0.0246]]
Actual Class : 7
--------------------------------------------------
69 Text feature [009775] present in test data point [True]
81 Text feature [12ca5] present in test data point [True]
89 Text feature [163950] present in test data point [True]
116 Text feature [150] present in test data point [True]
153 Text feature [1504t] present in test data point [True]
158 Text feature [194] present in test data point [True]
270 Text feature [0239] present in test data point [True]
435 Text feature [14] present in test data point [True]
Out of the top  500  features  8 are present in query point
```

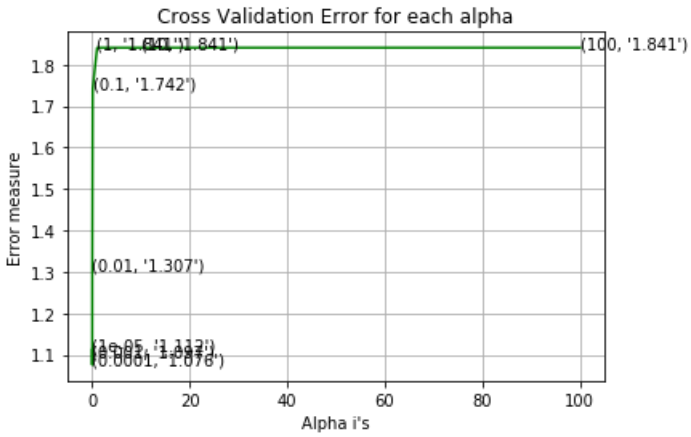
4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [149]:

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

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

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



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

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

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


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

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

```
for C = 1e-05
Log Loss : 1.112197645746317
for C = 0.0001
Log Loss : 1.0757728410707021
for C = 0.001
Log Loss : 1.0967580435072164
for C = 0.01
Log Loss : 1.3066265081474093
for C = 0.1
Log Loss : 1.7421417026393622
for C = 1
Log Loss : 1.8409124227562488
for C = 10
Log Loss : 1.840912433617069
for C = 100
Log Loss : 1.8409125245439981
```



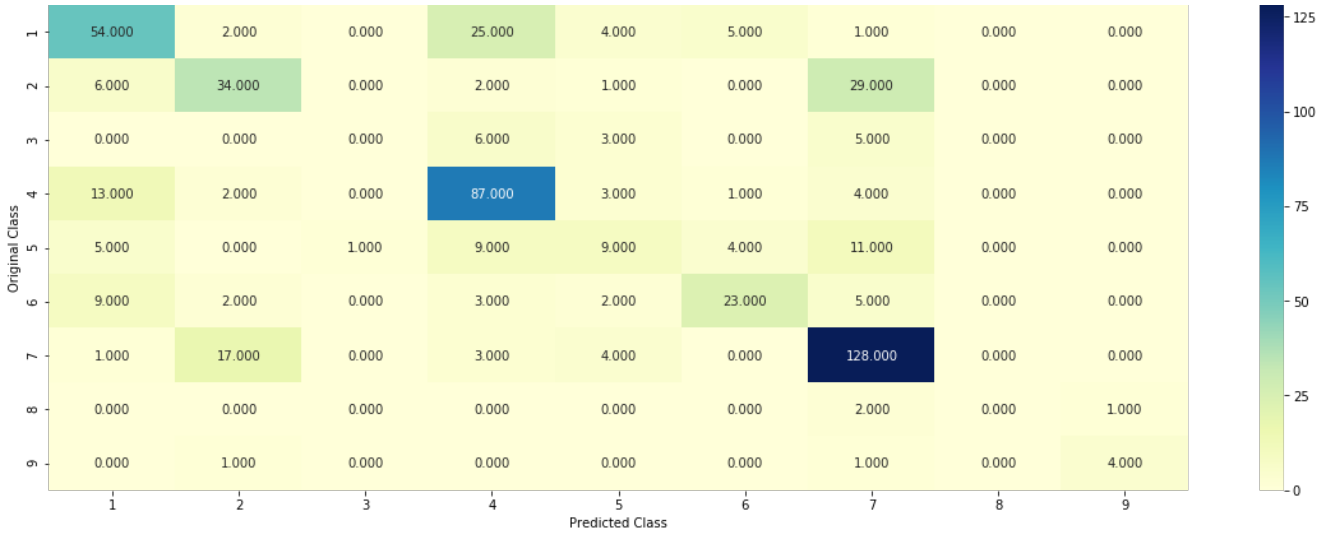

```
For values of best alpha =  0.0001 The train log loss is: 0.49047828894697604
For values of best alpha =  0.0001 The cross validation log loss is: 1.0757728410707021
For values of best alpha =  0.0001 The test log loss is: 0.9601915118587216
```

4.4.2. Testing model with best hyper parameters

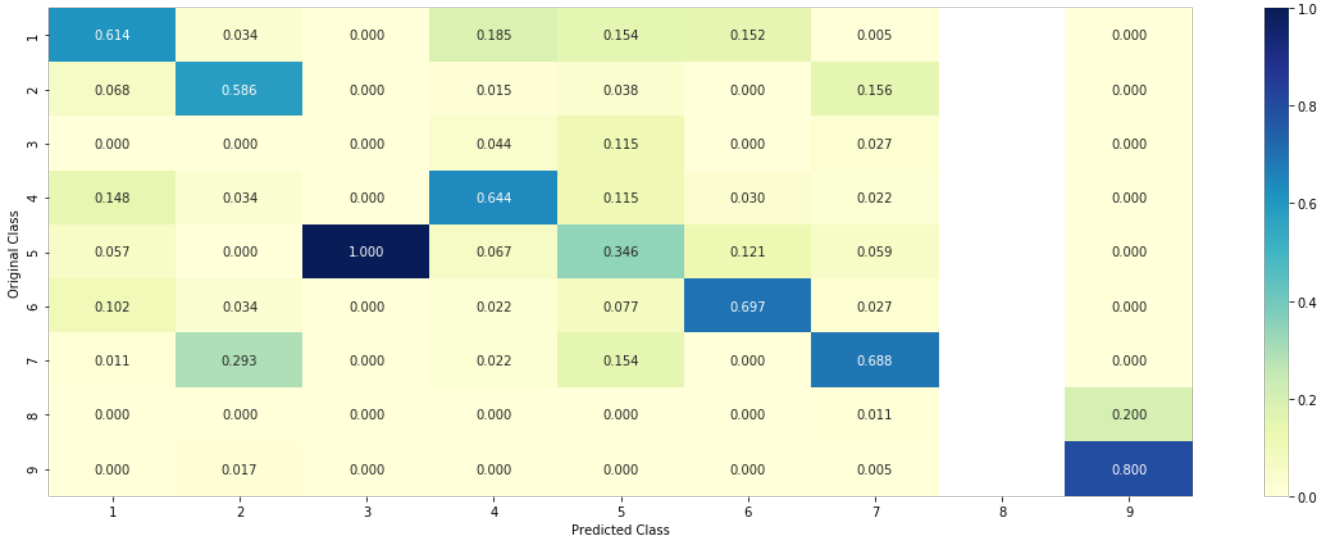
In [150]:

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

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

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


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

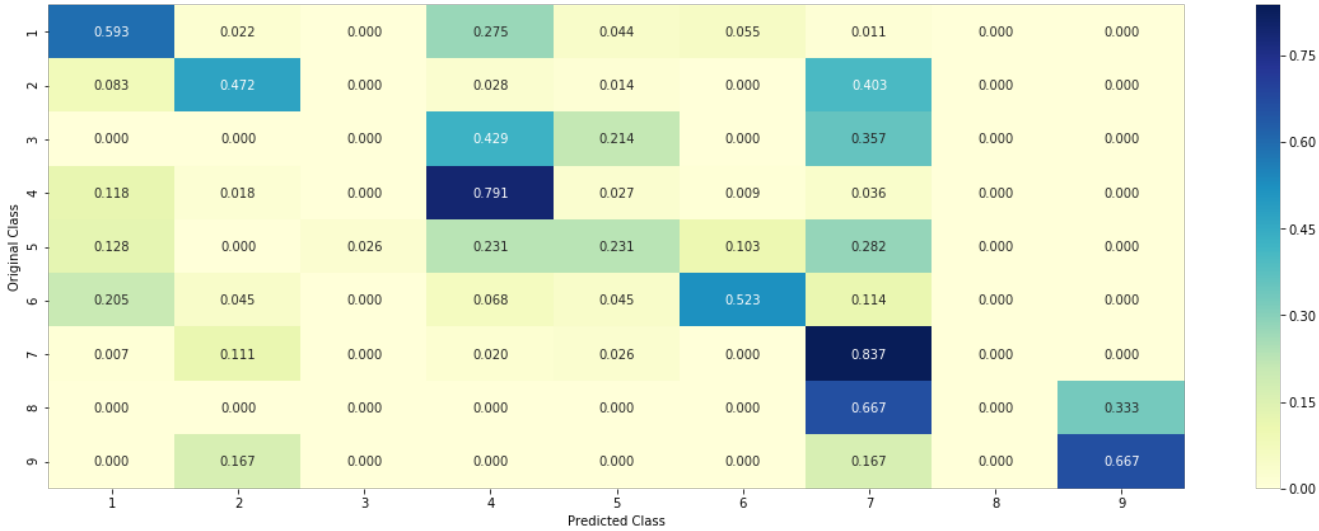
```
Log loss : 1.0757728410707021
Number of mis-classified points : 0.36278195488721804
-------------------- Confusion matrix --------------------
```



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



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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [151]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0523 0.0563 0.0077 0.0537 0.01   0.0119 0.8047 0.0018 0.0016]]
Actual Class : 7
--------------------------------------------------
289 Text feature [1c] present in test data point [True]
329 Text feature [14] present in test data point [True]
336 Text feature [1640] present in test data point [True]
341 Text feature [000] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [152]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[8.40e-03 1.79e-02 6.20e-03 9.80e-03 1.09e-02 1.18e-02 9.33e-01 1.
20e-03
  9.00e-04]]
Actual Class : 7
--------------------------------------------------
204 Text feature [194] present in test data point [True]
261 Text feature [1504t] present in test data point [True]
279 Text feature [195] present in test data point [True]
323 Text feature [163950] present in test data point [True]
329 Text feature [14] present in test data point [True]
341 Text feature [000] present in test data point [True]
362 Text feature [05] present in test data point [True]
Out of the top  500  features  7 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [153]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
```

```
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
verbose=0, warm_start=False,
# class_weight=None)

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

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

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


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

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

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

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

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

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

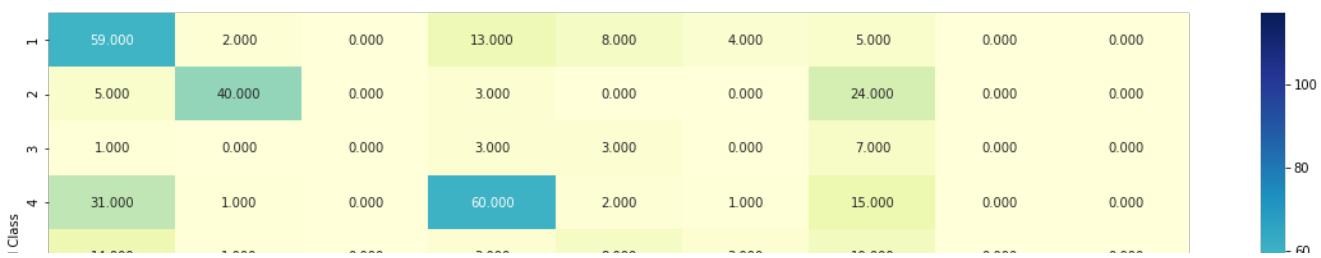
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2268712128459236
for n_estimators = 100 and max depth =  10
Log Loss : 1.2309770275444263
for n_estimators = 200 and max depth =  5
Log Loss : 1.2109460689643996
for n_estimators = 200 and max depth =  10
Log Loss : 1.222339690918756
for n_estimators = 500 and max depth =  5
Log Loss : 1.2059694808144303
for n_estimators = 500 and max depth =  10
Log Loss : 1.2202818226804968
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2041731737427057
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2196191054226597
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2008907940240243
for n_estimators = 2000 and max depth =  10
Log Loss : 1.219579346889432
For values of best estimator =  2000 The train log loss is: 0.8566282536067621
For values of best estimator =  2000 The cross validation log loss is: 1.2008907940240243
For values of best estimator =  2000 The test log loss is: 1.1409377121222668
```

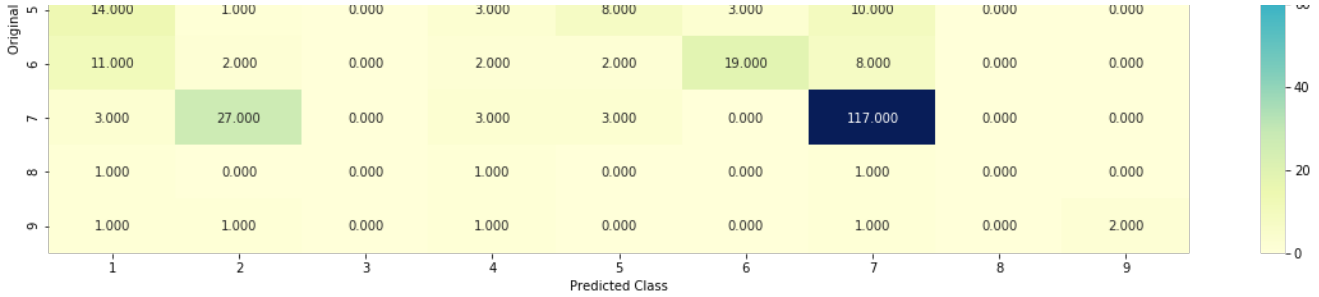
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [154]:

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

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

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

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

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

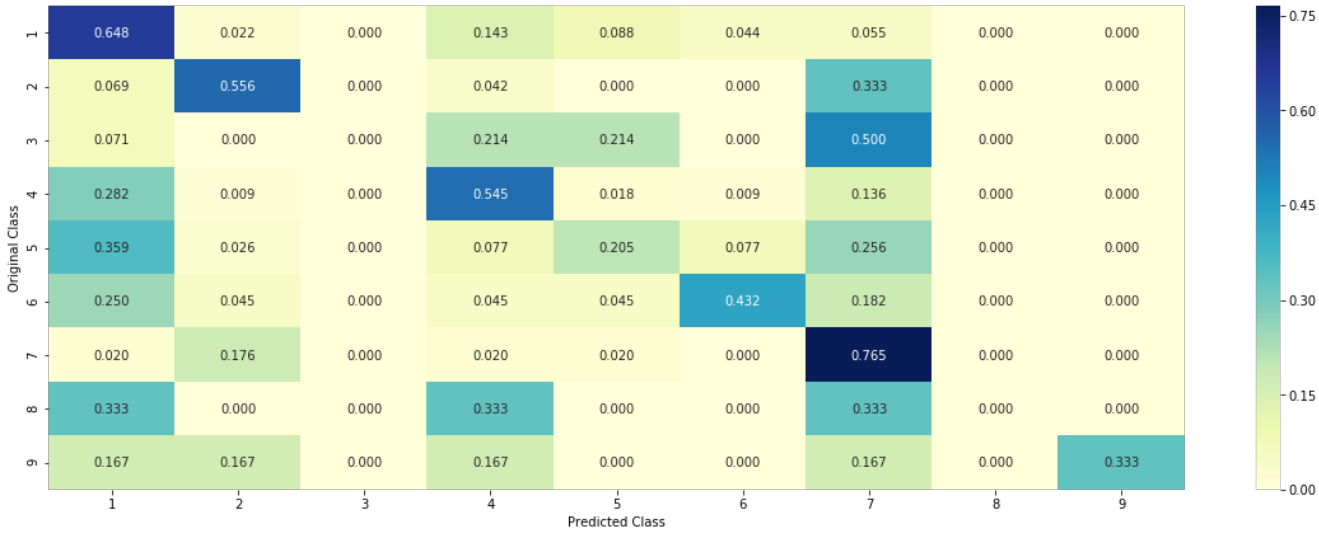
```
Log loss : 1.2008907940240243
Number of mis-classified points : 0.4266917293233083
-------------------- Confusion matrix --------------------
```





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

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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [155]:

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

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

```
Predicted Class : 7
Predicted Class Probabilities: [[6.300e-03 1.464e-01 1.410e-02 1.140e-02 2.930e-02 2.670e-02 7.627
e-01
  2.700e-03 4.000e-04]]
Actual Class : 7
--------------------------------------------------
58 Text feature [107] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [156]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0394 0.1445 0.0213 0.0688 0.0508 0.0426 0.6265 0.0044 0.0017]]
Actuall Class : 7
--------------------------------------------------
1 Text feature [009775] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [157]:

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

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

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

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

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

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

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

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

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.3595332869429897
for n_estimators = 10 and max depth =  3
Log Loss : 1.7840207373131647
for n_estimators = 10 and max depth =  5
Log Loss : 1.552965751525729
for n_estimators = 10 and max depth =  10
Log Loss : 1.7988891944968532
for n_estimators = 50 and max depth =  2
Log Loss : 1.7796594846441527
for n_estimators = 50 and max depth =  3
Log Loss : 1.4844984411786692
for n_estimators = 50 and max depth =  5
Log Loss : 1.4236064963616453
for n_estimators = 50 and max depth =  10
```

```
Log Loss : 1.7573932607327836
for n_estimators = 100 and max depth =  2
Log Loss : 1.6312706192706732
for n_estimators = 100 and max depth =  3
Log Loss : 1.506039413939742
for n_estimators = 100 and max depth =  5
Log Loss : 1.387696051663093
for n_estimators = 100 and max depth =  10
Log Loss : 1.743778982567626
for n_estimators = 200 and max depth =  2
Log Loss : 1.7177208328312574
for n_estimators = 200 and max depth =  3
Log Loss : 1.547167856462774
for n_estimators = 200 and max depth =  5
Log Loss : 1.4745291225115147
for n_estimators = 200 and max depth =  10
Log Loss : 1.7551183429633224
for n_estimators = 500 and max depth =  2
Log Loss : 1.7724788542874965
for n_estimators = 500 and max depth =  3
Log Loss : 1.625903383414368
for n_estimators = 500 and max depth =  5
Log Loss : 1.5169257714788038
for n_estimators = 500 and max depth =  10
Log Loss : 1.8238243367145062
for n_estimators = 1000 and max depth =  2
Log Loss : 1.7414212136974099
for n_estimators = 1000 and max depth =  3
Log Loss : 1.647123734866771
for n_estimators = 1000 and max depth =  5
Log Loss : 1.525021151666043
for n_estimators = 1000 and max depth =  10
Log Loss : 1.819265954257124
For values of best alpha =  100 The train log loss is: 0.05560946847868068
For values of best alpha =  100 The cross validation log loss is: 1.387696051663093
For values of best alpha =  100 The test log loss is: 1.2320061173618524
```

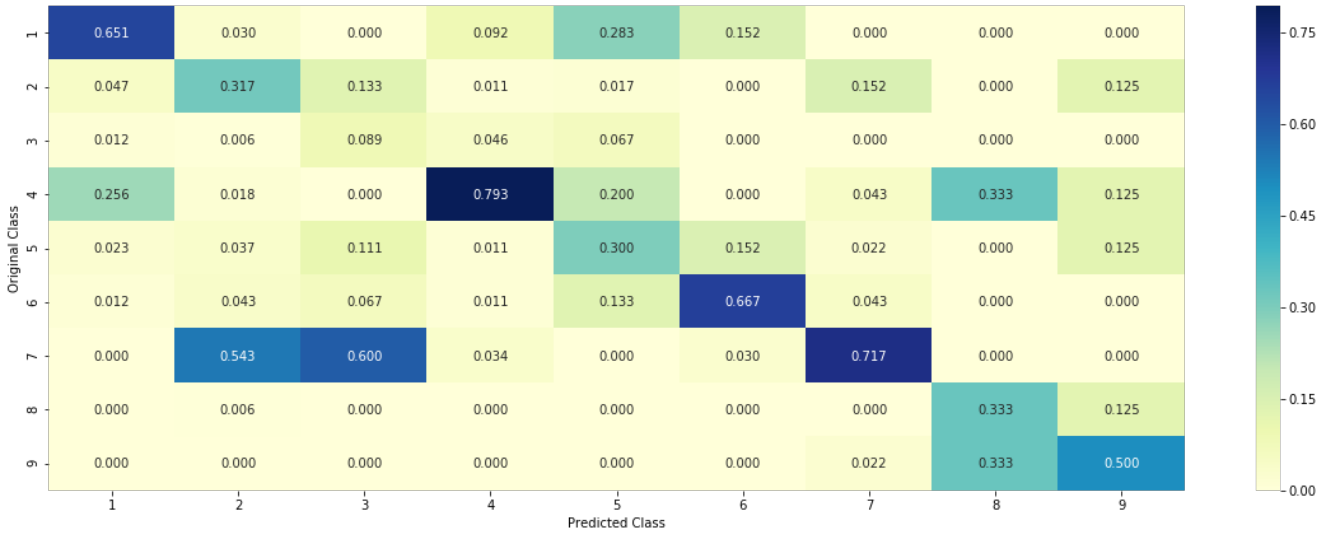
4.5.4. Testing model with best hyper parameters (Response Coding)

In [158]:

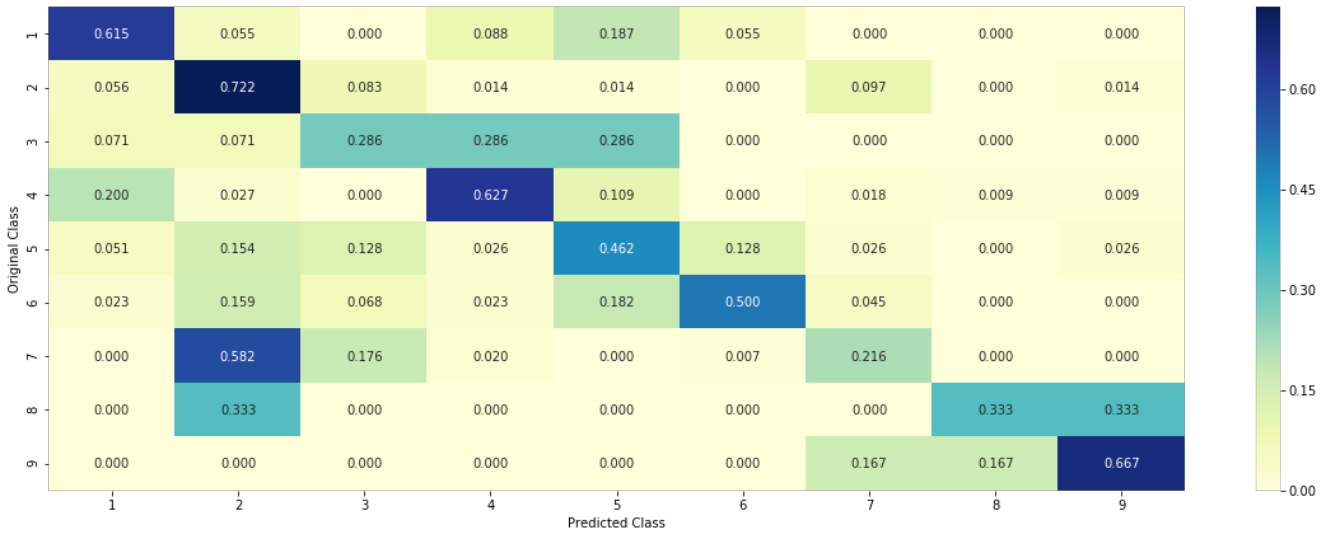
```
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None,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

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

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

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

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

```
Log loss : 1.387696051663093
Number of mis-classified points : 0.5131578947368421
-------------------- 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 [159]:

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


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

```
Predicted Class : 3
Predicted Class Probabilities: [[0.0117 0.0889 0.4995 0.0093 0.0146 0.0257 0.309  0.0218 0.0195]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [160]:

```
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 : 7
Predicted Class Probabilities: [[0.0146 0.2466 0.2491 0.0181 0.0223 0.0432 0.3461 0.0351 0.0248]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [161]:

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

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

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


# read more about support vector machines with linear kernals here http://scikit-
learn.org/stable/modules/generated/sklearn.svm.SVC.html
# ---------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, t
ol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', ra
ndom_state=None)

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

```
# predict(X) Perform classification on samples in X.
# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/mathematical-derivation-copy-8/
# --------------------------------


# read more about support vector machines with linear kernals here http://scikit-
learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
verbose=0, warm_start=False,
# class_weight=None)

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

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

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


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

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


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

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehot
Coding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y,
sig_clf2.predict_proba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding)))
)
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_p
robas=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, sc
lf.predict_proba(cv_x_onehotCoding))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

```
Logistic Regression :  Log Loss: 1.09
Support vector machines : Log Loss: 1.84
Naive Bayes : Log Loss: 1.28
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.033
```

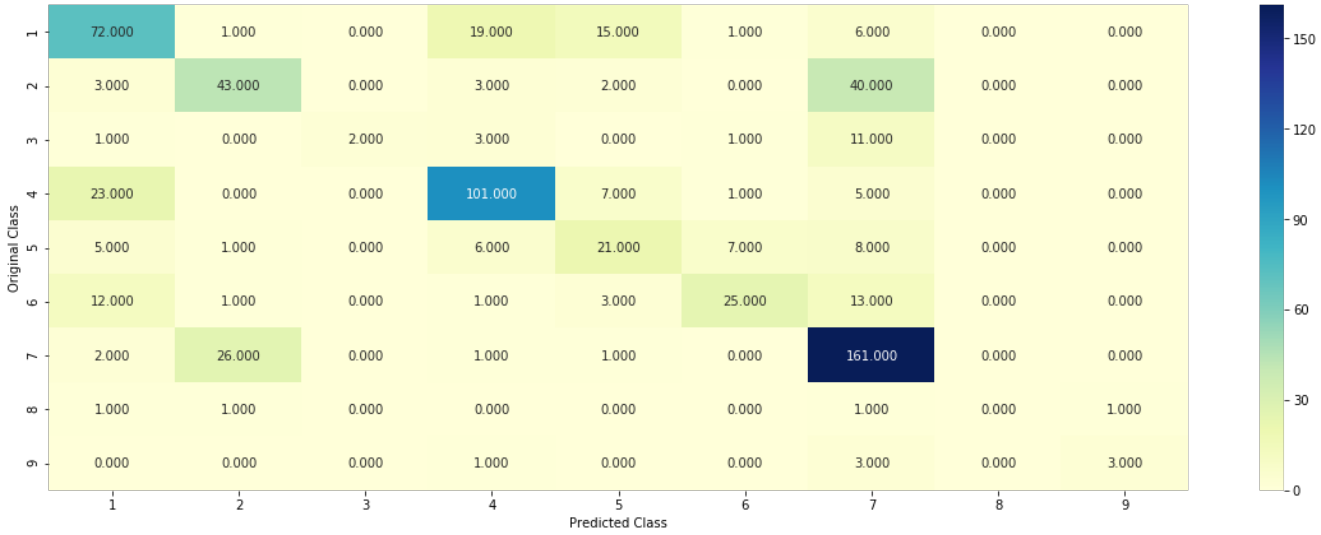
```
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.504
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.209
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.471
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.976
```

4.7.2 testing the model with the best hyper parameters

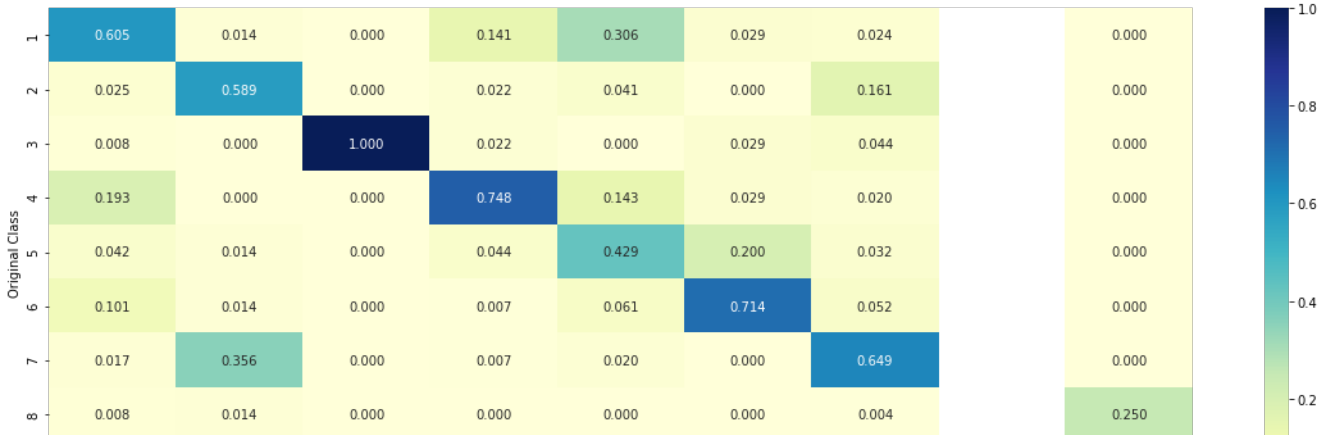
In [162]:

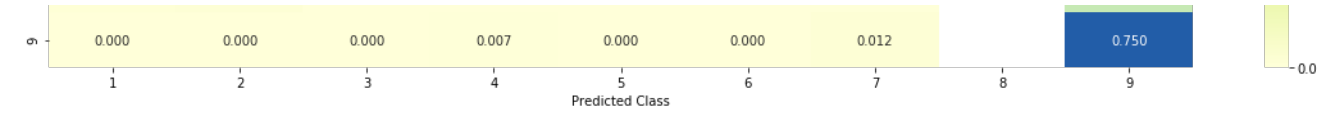
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba
s=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

```
Log loss (train) on the stacking classifier : 0.5806226615149678
Log loss (CV) on the stacking classifier : 1.2092076721338127
Log loss (test) on the stacking classifier : 1.1105841259866862
Number of missclassified point : 0.35639097744360904
-------------------- Confusion matrix --------------------
```

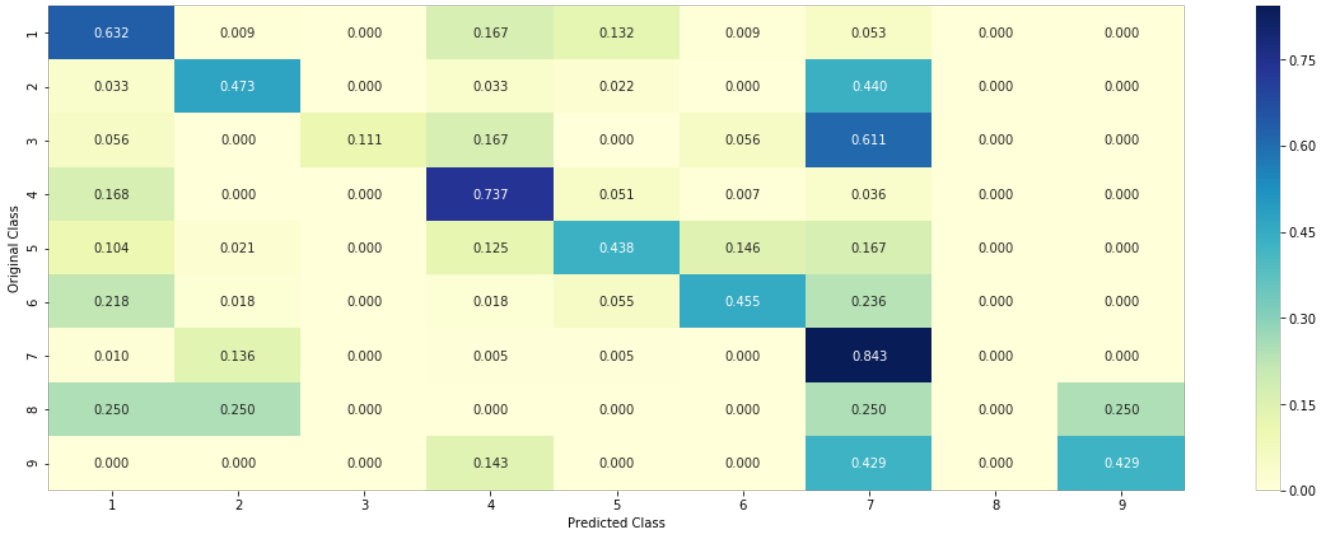



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```






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

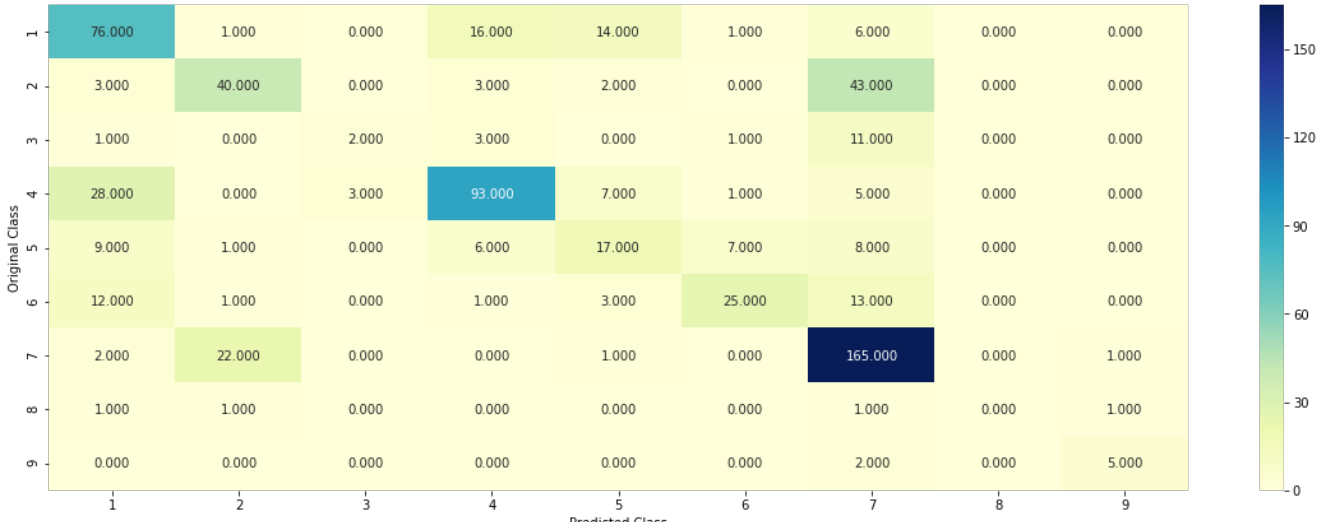


4.7.3 Maximum Voting classifier

In [163]:

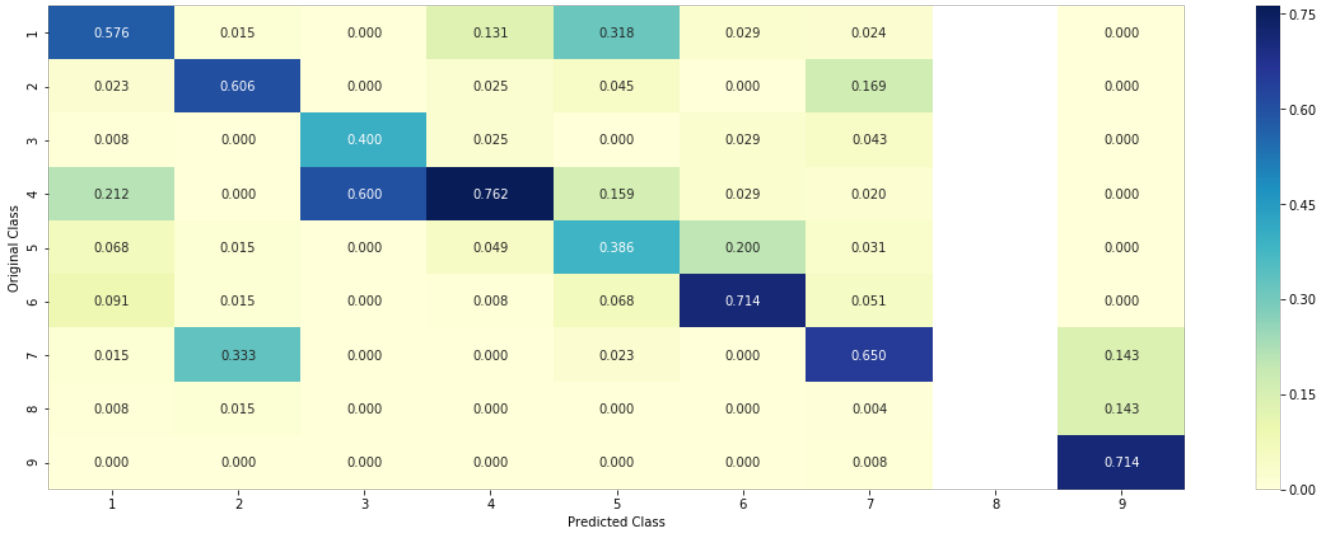
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting=
'soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log 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 : 0.8446533654712157
Log loss (CV) on the VotingClassifier : 1.2182985911105255
Log loss (test) on the VotingClassifier : 1.1378483821927472
Number of missclassified point : 0.36390977443609024
-------------------- Confusion matrix --------------------
```

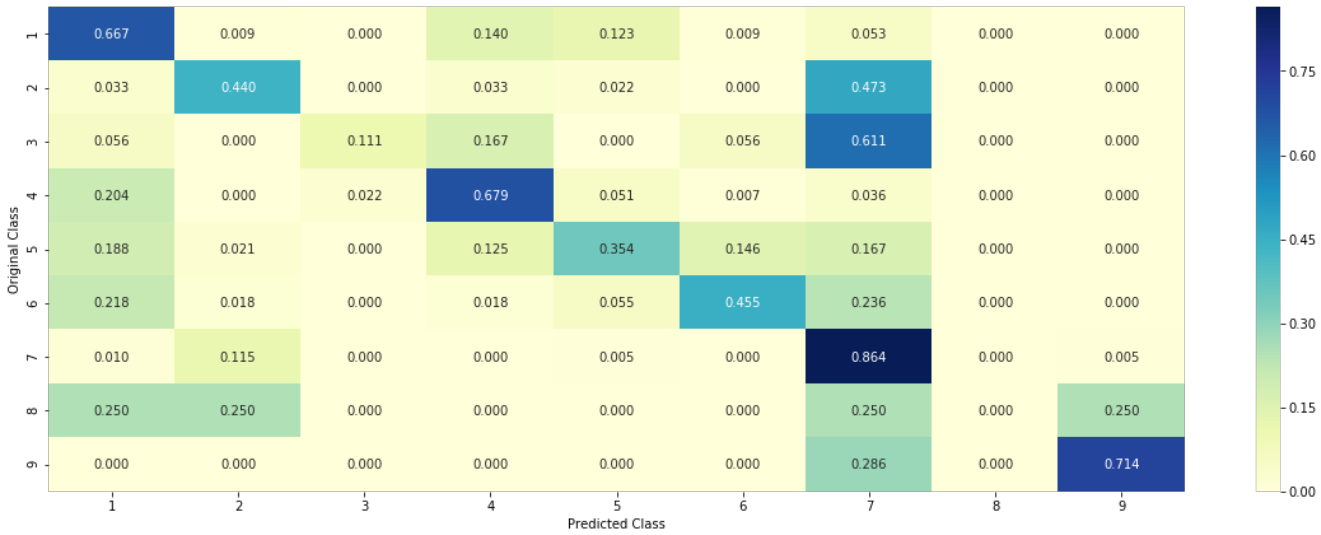


Predicted Class

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



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



Analysis of Personalized Cancer Diagnosis Case Study

Model	Best Hyperparameters	Train error	CV error	Test error	Misclassified Points	Log - Loss
Naive Bayes + one-hot Encoding	alpha=1e-05	0.54	1.28	1.17	41.91%	1.28
KNN + Response coding	K=15	0.70	1.09	0.99	41.91%	1.28	38.89%	1.09
Logistic Regression unigram+bigram + class balance	alpha= 0.01	0.84	1.22	1.13	39.66%	1.22
Logistic Regression unigram+bigram + without class balance	alpha= 0.01	0.85	1.23	1.14	39.84%	1.23
Logistic Regression tfidf vectorizer(with 2000 max words)+FE(square root)	alpha= 0.001	0.60	1.00	0.87	31.42%	0.87
Linear SVM + one-hot encoding	alpha=0.0001	0.49	1.07	0.96	36.27%	1.07
Random Forest + one-hot encoding	best-estimators=2000	0.85	1.20	1.14	42.66%	1.20
Random Forest + one-hot encoding	alpha=100	0.05	1.38	1.23	51.33%	1.38
Stack Classifier(LR+LrSVM+NB)	alpha=0.10	0.58	1.20	1.11	31.56%	1.19

Model	Best Hyperparameters	Train error	CV error	Test error	Misclassified Points	Log - Loss
Maximum voting Classifier	alpha=0.10	0.84	1.21	1.13	31.39%	1.19

Observation

- We did onehot encoding featurization for TfidfVectorizer() with 2000 features
- We accomplished onehot encoding for Logistic Regression models with un-igram and bi-gram
- On Logistic Regression model we did tfidf vectorizer with 2000 features and did Feature Engineering such as square root which have given log-loss 0.87. Using square root feature engineering we find the best model among all models.
- Second good model is Liner Support Vector Machine