Personalized Cancer diagnosis using Genetic Data (Feature Engineering)

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

ID,Text
0||Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity revealed. Previous work has shown that CDK10 silencing increases ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The precise mechanisms by which CDK10 modulates ETS2 activity, and more generally the functions of CDK10, remain elusive. Here we demonstrate that CDK10 is a cyclin-dependent kinase by identifying cyclin M as an activating cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR syndrome, a human developmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malformations. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and in conferring tamoxifen resistance to breast cancer cells. CDK10/cyclin M phosphorylates ETS2 in vitro, and in cells it positively controls ETS2 degradation by the proteasome. ETS2 protein levels are increased in cells derived from a STAR patient, and this increase is attributable to decreased cyclin M levels. Altogether, our results reveal an additional regulatory mechanism for ETS2, which plays key roles in cancer and development. They also shed light on the molecular mechanisms underlying STAR syndrome.Cyclin-dependent kinases (CDKs) play a pivotal role in the control of a number of fundamental cellular processes (1). The human genome contains 21 genes encoding proteins that can be considered as members of the CDK family owing to their sequence similarity with bona fide CDKs, those known to be activated by cyclins (2). Although discovered almost 20 y ago (3, 4), CDK10 remains one of the two CDKs without an identified cyclin partner. This knowledge gap has largely impeded the exploration of its biological functions. CDK10 can act as a positive cell cycle regulator in some cells (5, 6) or as a tumor suppressor in others (7, 8). CDK10 interacts with the ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity through an unknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK pathway, and induces resistance of MCF7 cells to tamoxifen (6). ...

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [81]:

```
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))

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 collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")
from mlxtend.classifier import StackingClassifier
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:

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

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

Out[2]:

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

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

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

3.1.2. Reading Text Data

In [3]:

```
# note '||' the seprator in this file
data_text =pd.read_csv("training/training_text",sep="\|\|",engine="python",names=["ID","TEXT"],skiprows=1)
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

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

Out[3]:

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

3.1.3. Getting number of sentences in each Text

In [4]:

```
#We will add a colum containing the approximate number of sentences in a literature
data_text['NumOfSent'] = data_text['TEXT'].apply(lambda sent : len(str(sent).split(".")))
data_text.head()
```

Out[4]:

	ID	TEXT	NumOfSent
0	0	Cyclin-dependent kinases (CDKs) regulate a var...	398
1	1	Abstract Background Non-small cell lung canc...	317
2	2	Abstract Background Non-small cell lung canc...	317
3	3	Recent evidence has demonstrated that acquired...	225
4	4	Oncogenic mutations in the monomeric Casitas B...	323

In [5]:

```
data_text.NumOfSent.describe()
```

Out[5]:

```
count    3321.000000
mean      572.594098
std       486.904970
min         1.000000
25%       260.000000
50%       427.000000
75%       744.000000
max      4850.000000
Name: NumOfSent, dtype: float64
```

3.1.4. Preprocessing of text

In [6]:

```
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))


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

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

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

In [7]:

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

```
There is no text description for id: 1109
There is no text description for id: 1277
There is no text description for id: 1407
There is no text description for id: 1639
There is no text description for id: 2755
Time took for preprocessing the text : 220.29534 seconds
```

In [8]:

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

Out[8]:

	ID	Gene	Variation	Class	TEXT	NumOfSent
0	0	FAM58A	Truncating Mutations	1	cyclin dependent kinases cdks regulate variety...	398
1	1	CBL	W802*	2	abstract background non small cell lung cancer...	317
2	2	CBL	Q249E	2	abstract background non small cell lung cancer...	317
3	3	CBL	N454D	3	recent evidence demonstrated acquired uniparen...	225
4	4	CBL	L399V	4	oncogenic mutations monomeric casitas b lineag...	323

In [9]:

```
#Check for null entries
result[result.isnull().any(axis=1)]
```

Out[9]:

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

In [10]:

```
#Replace the null entries in Text by Gene + Variation Data
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

In [11]:

```
#Display row after null replacement
result[result['ID']==1109]
```

Out[11]:

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

In [12]:

```
#Display column names
result.columns
```

Out[12]:

```
Index(['ID', 'Gene', 'Variation', 'Class', 'TEXT', 'NumOfSent'], dtype='object')
```

In [13]:

```
# Python program to find the k most frequent words in a given text
def freq_text(text,k):
    data_set = text
    from collections import Counter
    split_it = data_set.split()
    Counter = Counter(split_it)
    most_occur = list(dict(Counter.most_common(k)).keys())
    most_occur_str = ' '.join(word for word in most_occur)
    return(most_occur_str)

#Merge the variation + Text features into one single text
#Remove unwanted words like "cell","line","et","all" etc
#I have plotted word cloud and saw that the words "cell","line","et","all","fig" occur in almost all the classes
of cancers. So we will drop them. Because, they add no purpose to the model that we are designing.
#Remove * sign from Variants
#Convert everything in lower case
result["Gene"] = result["Gene"].apply(lambda x: x.lower())
result["Variation"] = result["Variation"].apply(lambda x: x.lower())
result["MergedText"] = result["Variation"].apply(lambda x: x.replace("*",'')) + " " + result["TEXT"].apply(lambda
x : x.replace('cell','').replace('line','').replace('et','').replace('al','').replace('fig',''))
result["TopWordsInText"] = result["MergedText"].apply(lambda x: freq_text(str(x),75))
```

In [14]:

```
#Save the final dataframe to "result.csv"
result.to_csv("result.csv",columns=result.columns,index=None)
```

In [82]:

```
#Load the pre-processed data frame
result=pd.read_csv('result.csv')
result.head(5)
```

Out[82]:

	ID	Gene	Variation	Class	TEXT	NumOfSent	MergedText	TopWordsInText
0	0	fam58a	truncating mutations	1	cyclin dependent kinases cdks regulate variety...	398	truncating mutations cyclin dependent kinases ...	cdk10 cyclin s2 1 s expression protein c level...
1	1	cbl	w802*	2	abstract background non small cell lung cancer...	317	w802 abstract background non sml lung cancer ...	c cbl mutations lung egfr mutation s 1 m sampl...
2	2	cbl	q249e	2	abstract background non small cell lung cancer...	317	q249e abstract background non sml lung cancer...	c cbl mutations lung egfr mutation s 1 m sampl...
3	3	cbl	n454d	3	recent evidence demonstrated acquired uniparen...	225	n454d recent evidence demonstrated acquired un...	cbl mutations cases aupd mutation 3 upn 2 ure ...
4	4	cbl	l399v	4	oncogenic mutations monomeric casitas b lineag...	323	l399v oncogenic mutations monomeric casitas b ...	cbl mutations cancer binding e2 stability acti...

3.1.5. Test, Train and Cross Validation Split

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

In [83]:

```
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_tru
e]
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_tra
in' [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)

train_df.head()
```

Out[83]:

	ID	Gene	Variation	Class	TEXT	NumOfSent	MergedText	TopWordsInText
2055	2055	myc	fusions	2	burkitt lymphoma highly aggressive b cell non ...	591	fusions burkitt lymphoma highly aggressive b ...	myc lymphoma burkitt c s treatment cancer pati...
2372	2372	ptpn11	e69k	2	protein tyrosine phosphatases ptps key positiv...	212	e69k protein tyrosine phosphatases ptps key po...	mutations ptpn11 2 jmml n 1 sh2 ras mutation s...
2924	2924	nfe2l2	d29h	7	nrf2 nuclear factor erythroid 2 nf e2 related ...	665	d29h nrf2 nuclear factor erythroid 2 nf e2 rel...	nrf2 keap1 cancer s mutations protein so expre...
484	484	tp53	r158l	4	transcription factor tumor suppressor protein ...	715	r158l transcription factor tumor suppressor pr...	p53 mutants wt cancer mutations yeast tp53 fun...
2402	2402	nf1	s1463f	4	ras p21 interacts directly catalytic subunit p...	2671	s1463f ras p21 interacts directly catytic subu...	ras 1 s nf1 raf 0 3 p mutations activation c g...

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

In [17]:

```
#Save the 3 splitted datasets
train_df.to_csv("datasets2/train_df.csv",columns=train_df.columns)
cv_df.to_csv("datasets2/cv_df.csv",columns=cv_df.columns,index=None)
test_df.to_csv("datasets2/test_df.csv",columns=test_df.columns,index=None)
```

In [18]:

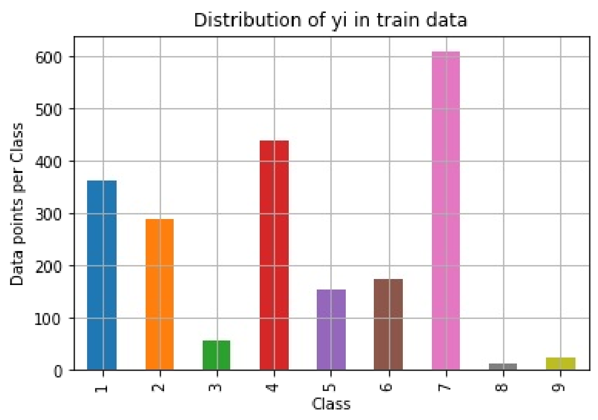
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
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
```

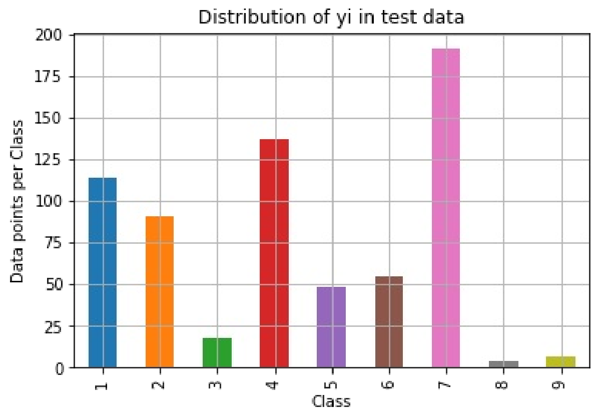
3.1.5.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [19]:

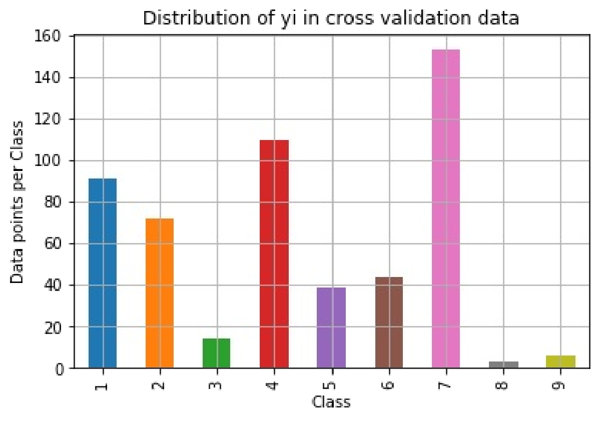
```
# it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distribution.values[i], '(', np.round((train_cla
ss_distribution.values[i]/train_df.shape[0]*100), 3), '%)')


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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.values[i], '(', np.round((test_class
_distribution.values[i]/test_df.shape[0]*100), 3), '%)')

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.values[i], '(', np.round((cv_class_dis
tribution.values[i]/cv_df.shape[0]*100), 3), '%)')
```



```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298 %)
Number of data points in class 3 : 57 ( 2.684 %)
Number of data points in class 9 : 24 ( 1.13 %)
Number of data points in class 8 : 12 ( 0.565 %)
--------------------------------------------------------------------------------
```



```
Number of data points in class 7 : 191 ( 28.722 %)
Number of data points in class 4 : 137 ( 20.602 %)
Number of data points in class 1 : 114 ( 17.143 %)
Number of data points in class 2 : 91 ( 13.684 %)
Number of data points in class 6 : 55 ( 8.271 %)
Number of data points in class 5 : 48 ( 7.218 %)
Number of data points in class 3 : 18 ( 2.707 %)
Number of data points in class 9 : 7 ( 1.053 %)
Number of data points in class 8 : 4 ( 0.602 %)
--------------------------------------------------------------------------------
```



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

Key Take-away:

We can see the class labels are distributed fairly equally in all the three dataset - train, cross validation and test data. If it hadn't been the case our models won't perform good. The distributions has to be similar in all the three datasets for all the 9 classes.

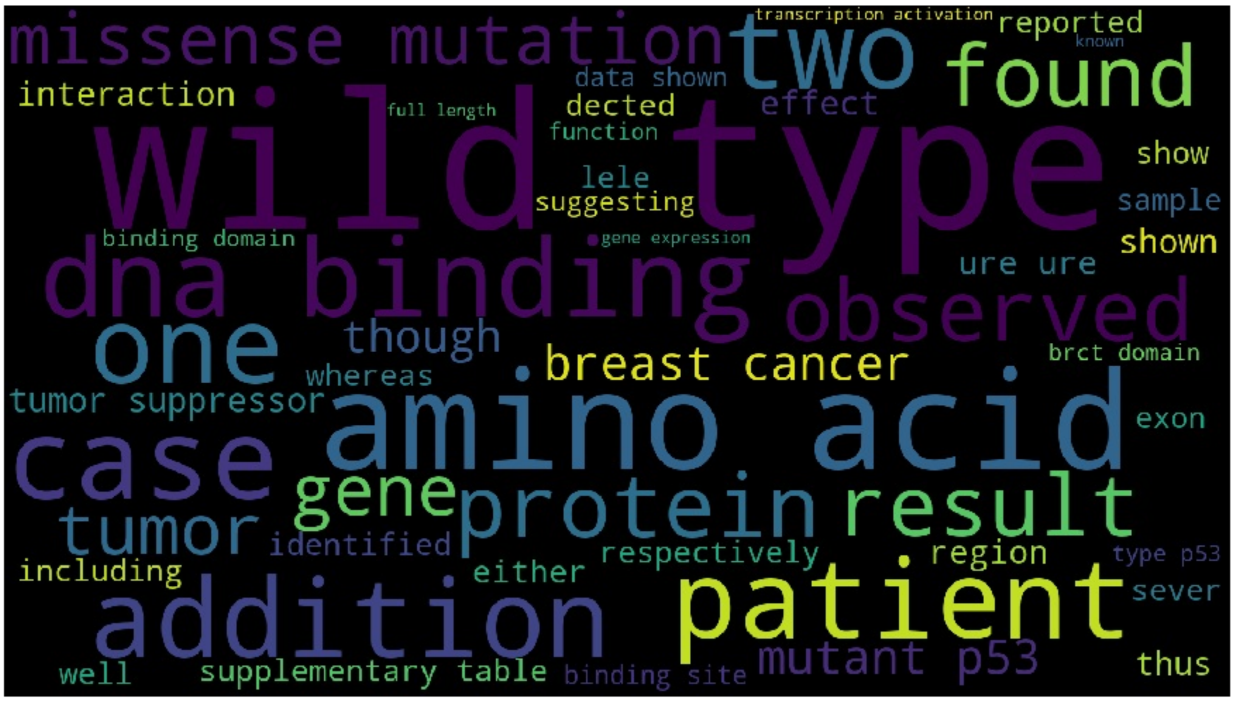
3.1.6. Draw word clouds to check for the most frequently occuring words in each classes in train data

In [84]:

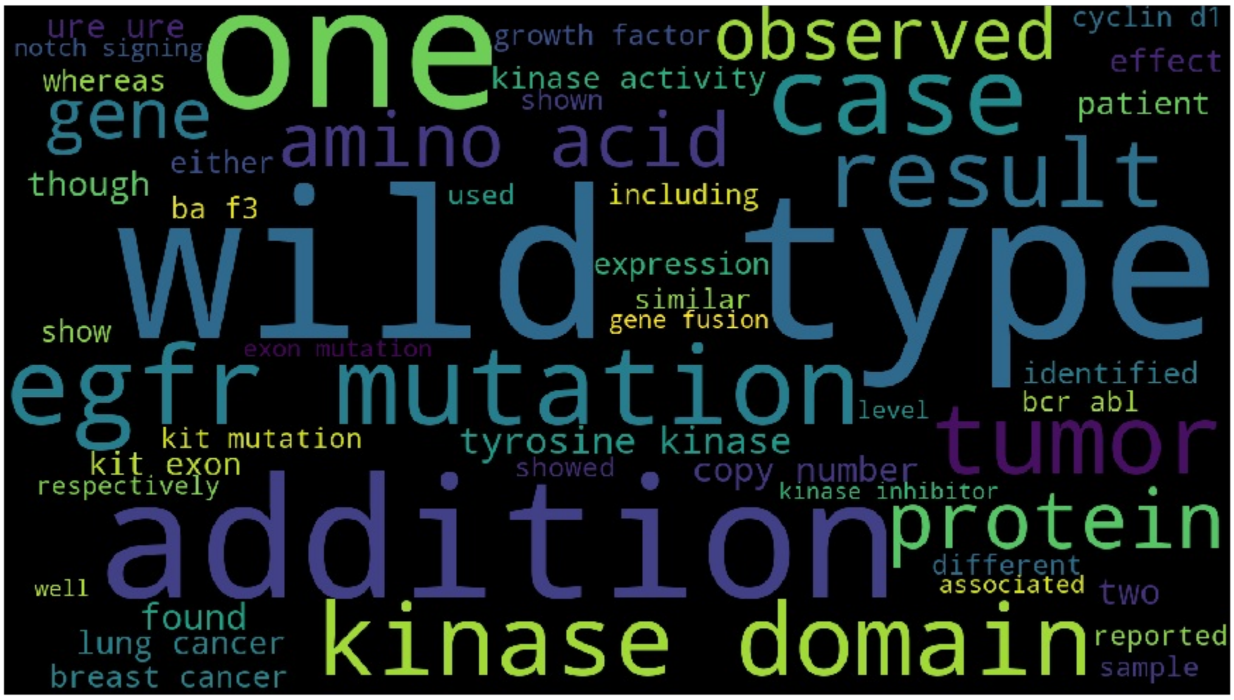
```
from tqdm import tqdm
from wordcloud import WordCloud
for i in tqdm(range(1,len(train_df.Class.unique())+1)):
    cloud = WordCloud(prefer_horizontal=1,width=1600, height=900,max_words=50).generate(" ".join(train_df[train_d
f.Class == i]['MergedText']))
    cloud.to_file("wordclouds/Class{}.jpg".format(str(i)))
    plt.figure(figsize=(16,9))
    plt.imshow(cloud)
    plt.axis('off')
    ax = plt.axes()
    ax.set_title('Word Cloud for Class = {}'.format(i))
```

```
100%|██████████| 9/9 [01:31<00:00,  8.65s/it]
```

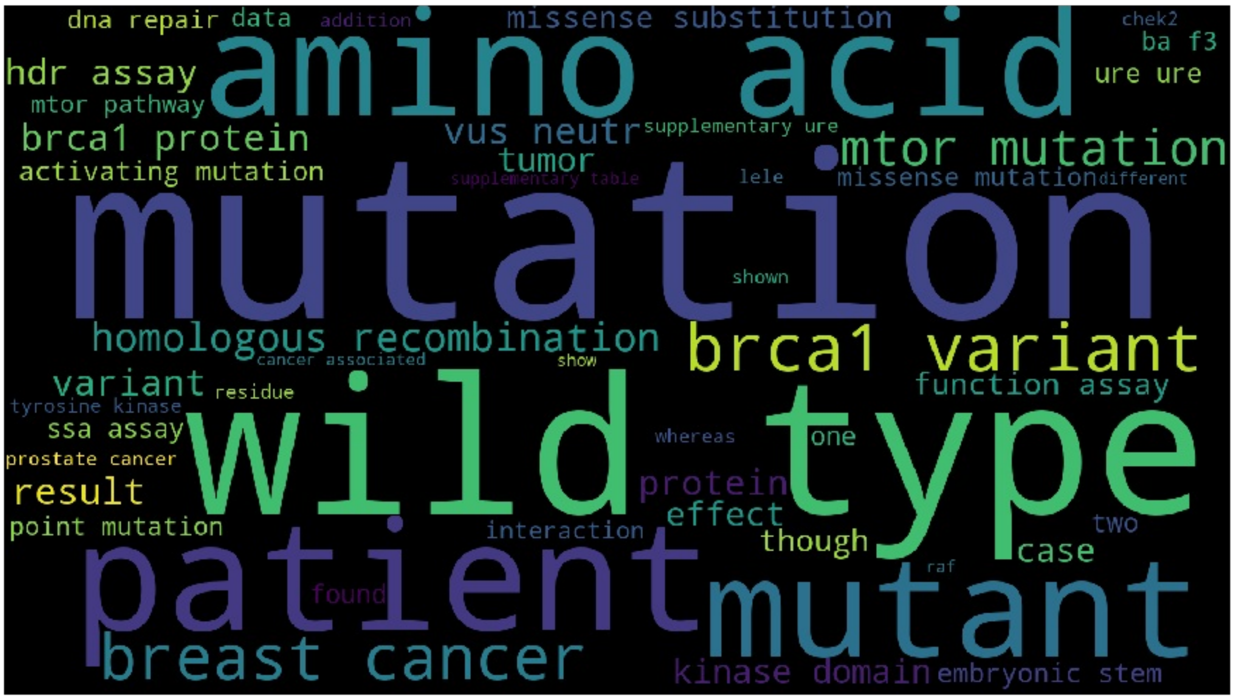
Word Cloud for Class = 1



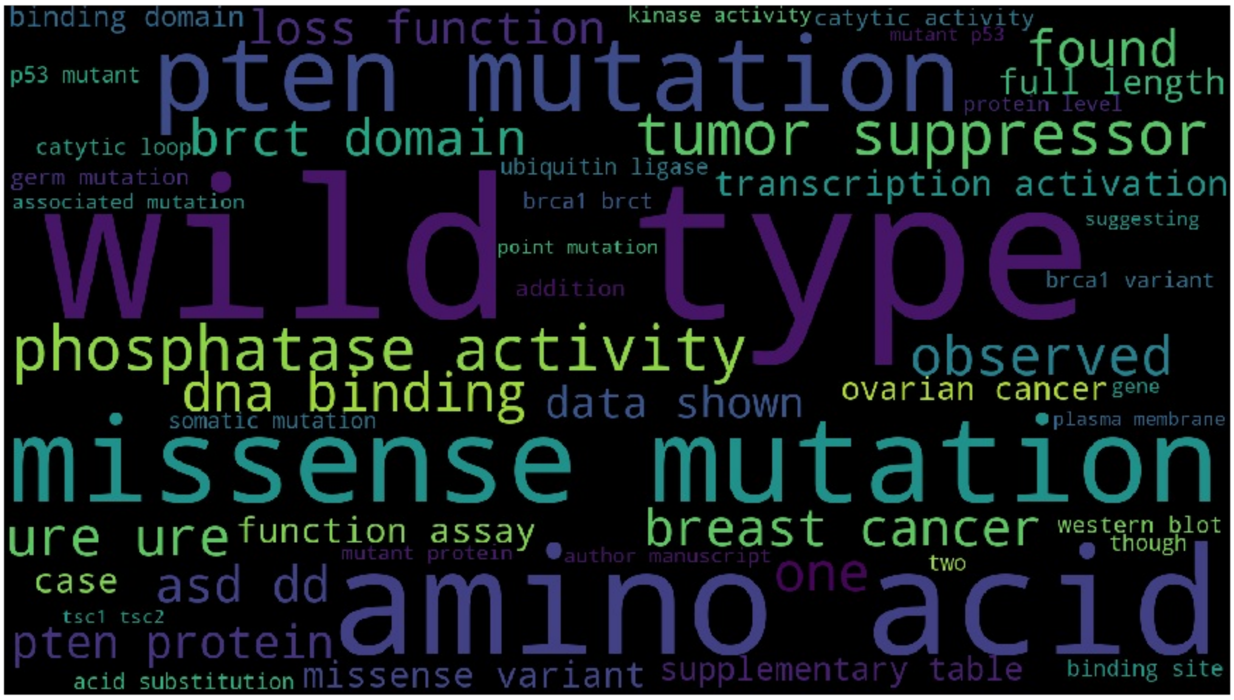
Word Cloud for Class = 2



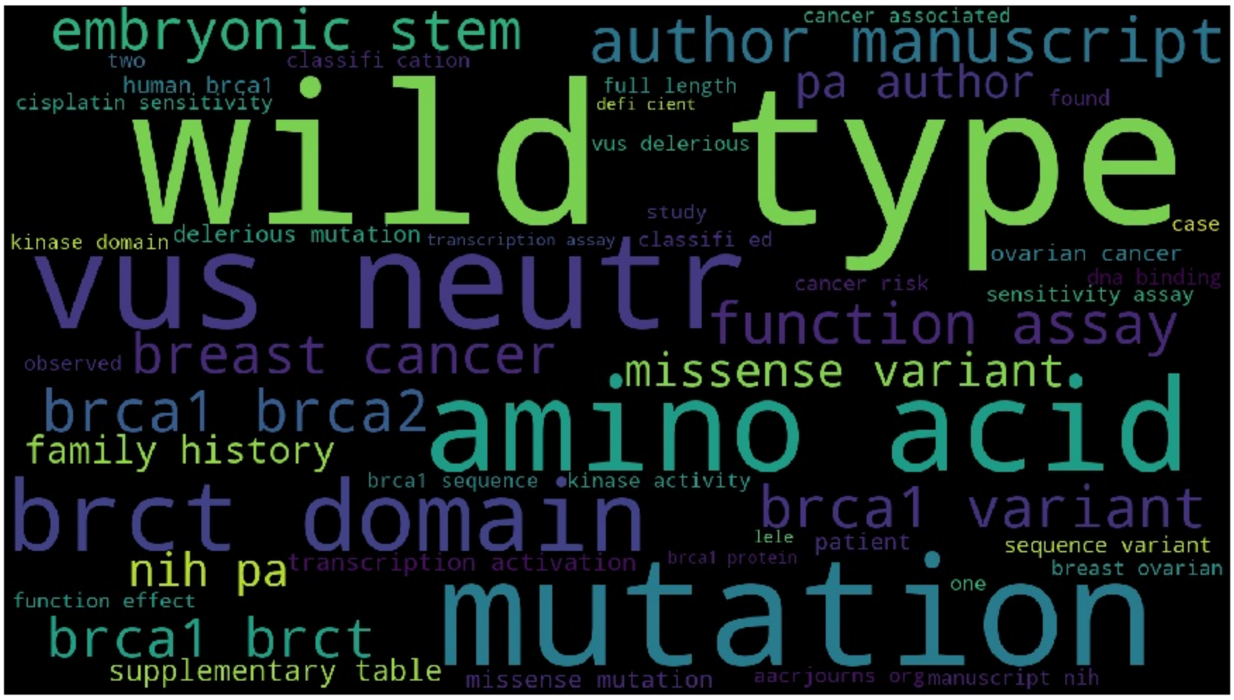
Word Cloud for Class = 3



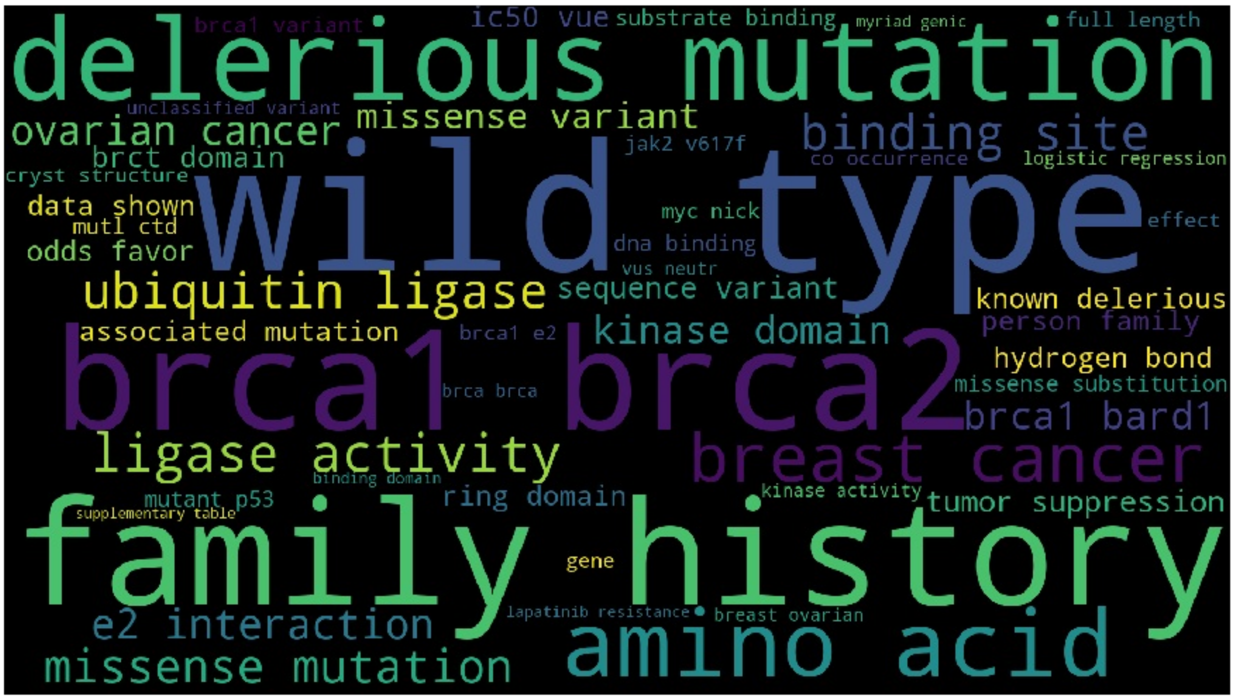
Word Cloud for Class = 4



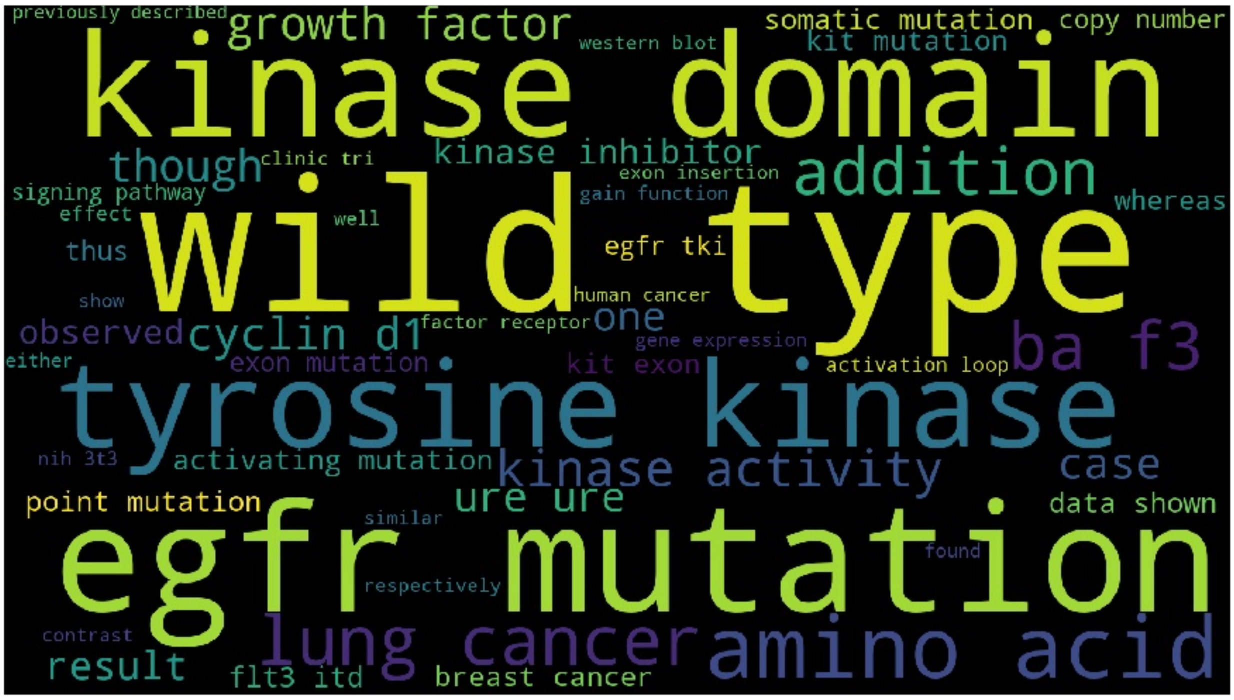
Word Cloud for Class = 5



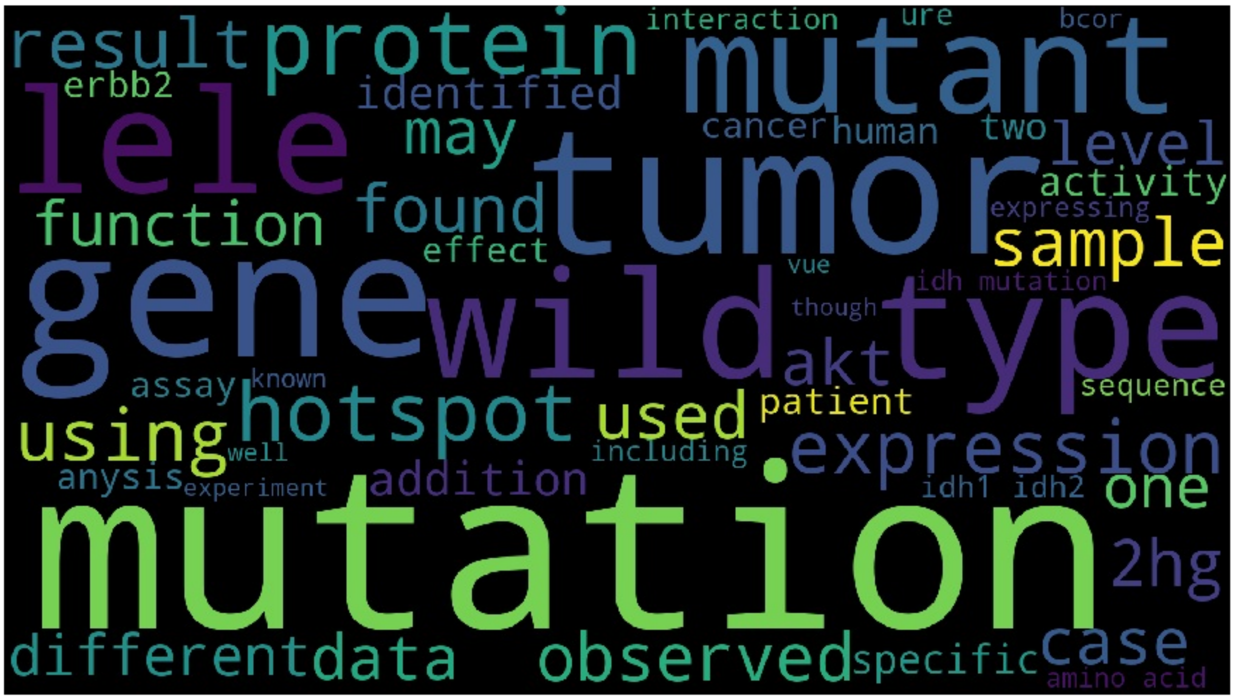
Word Cloud for Class = 6



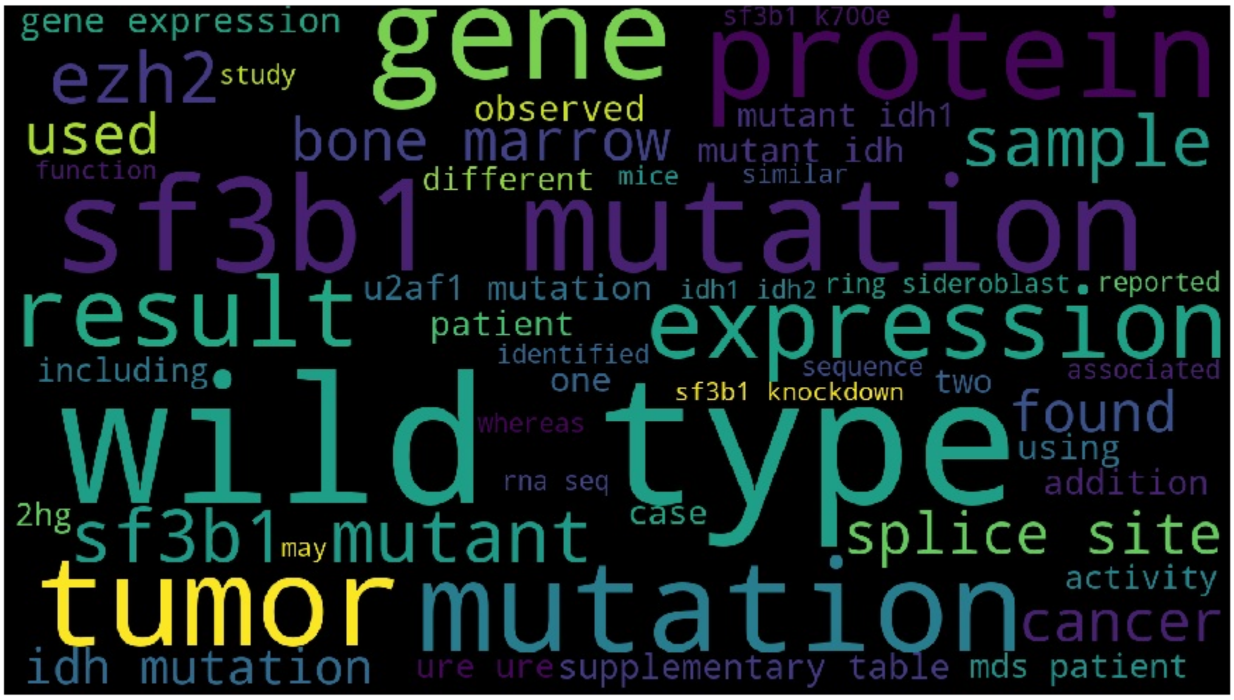
Word Cloud for Class = 7



Word Cloud for Class = 8



Word Cloud for Class = 9



If we look at the above word cloud, we can straight away see some keywords which tells us what a particular class is all about. For example we see keywords like "breast cancer", "lung cancer", "ovarian cancer", "gene mutation", "tyrosie kinase", "egfr mutation","amino acids" and so on, belonging to each of the classes. Here I have given just a few examples. We also see that there are some classes which talks about some particular variation/mutant type more frequently than others. One important conclusion that we can draw from the word cloud is that bigrams are very important for this problem, since words which occurs in pairs adds great insight in our diagnosis. As we have seen the occurrences of bigrams words like "breast cancer","lung cancer" etc.

3.2 Prediction using a 'Random' Model

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

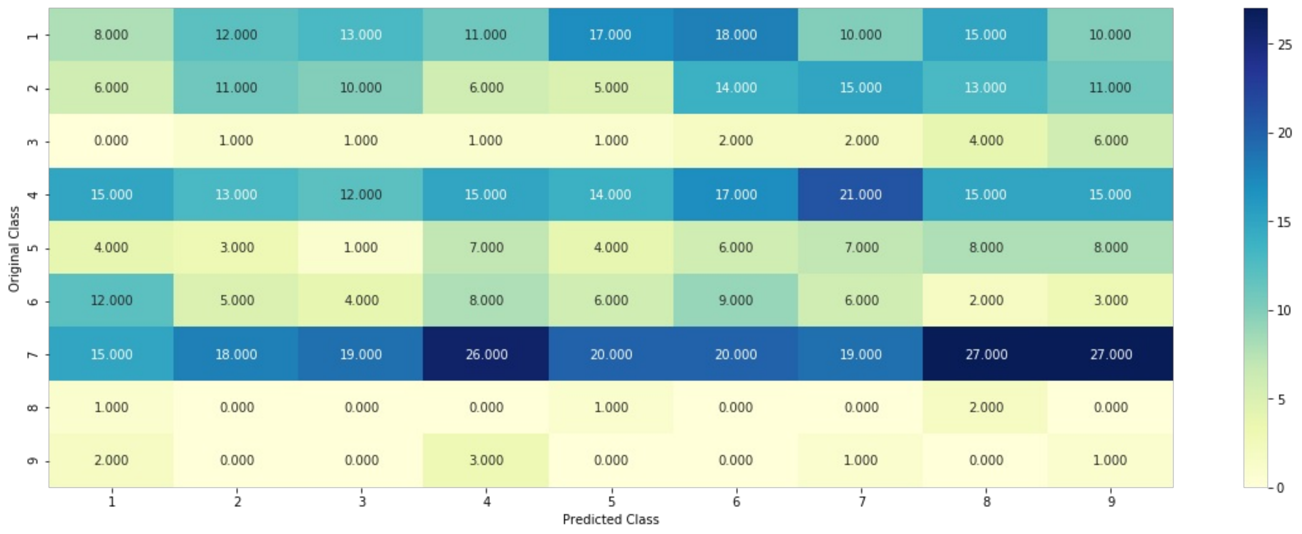
In [21]:

```python
# 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)

    B =(C/C.sum(axis=0))
    #C = Precision Matrix
    labels = [1,2,3,4,5,6,7,8,9]
    # representing A in heatmap format
    print("-"*20, "Confusion matrix", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

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

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

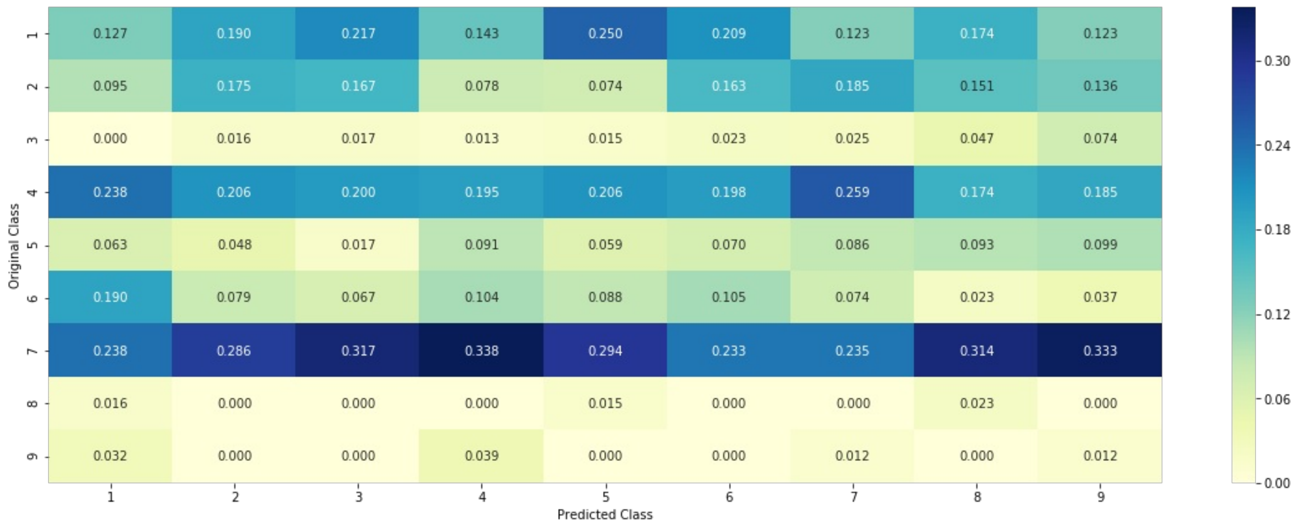
In [22]:

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

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

# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y, eps=1e-15))

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

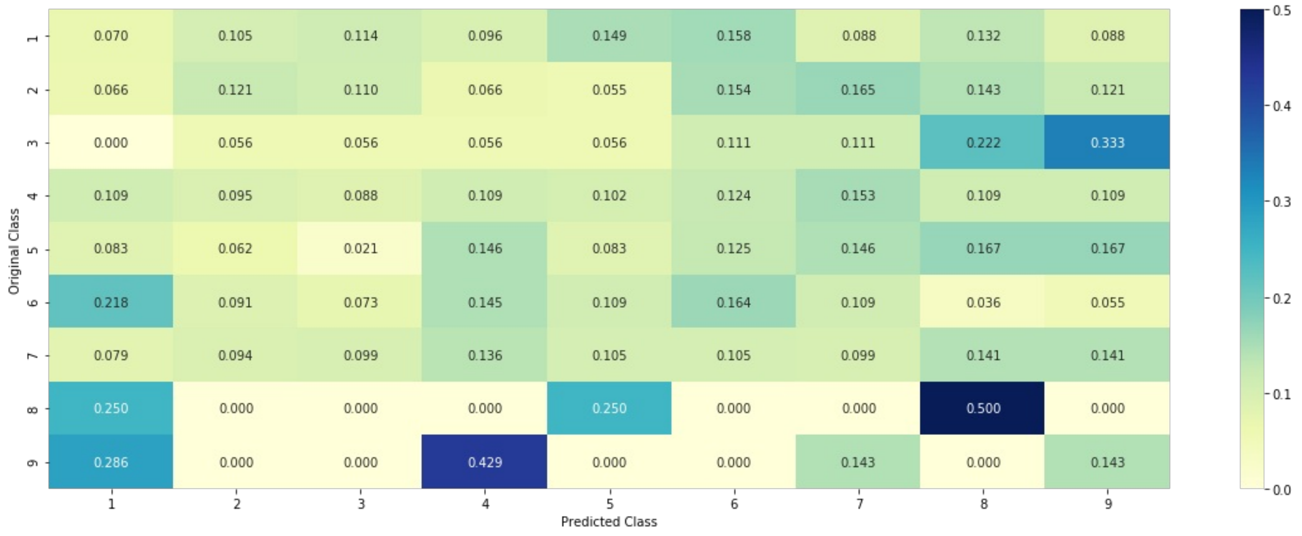
```
Log loss on Cross Validation Data using Random Model 2.478951780062388
Log loss on Test Data using Random Model 2.511980242361641
-------------------- Confusion matrix --------------------
```



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



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



3.3 Univariate Analysis

In [23]:

```
# 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 a given feature in train data dataframe
# build a vector (1*9) , the first element = (number of times it occured in class1 + 10*alpha / number of time it
occurred in total data+90*alpha)
# gv_dict is like a look up table, for every gene it store a (1*9) 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
    # i = feature name, denominator = feature count
    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 dimensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=='BRCA1')])
            #         ID   Gene             Variation  Class
            # 2470  2470  BRCA1                S1715C      1
            # 2486  2486  BRCA1                S1841R      1
            # 2614  2614  BRCA1                   M1R      1
            # 2432  2432  BRCA1                L1657P      1
            # 2567  2567  BRCA1                T1685A      1
            # 2583  2583  BRCA1                E1660G      1
            # 2634  2634  BRCA1                W1718L      1
            # cls_cnt.shape[0] will return the number of rows

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

            # cls_cnt.shape[0](numerator) will contain the number of time that particular feature occured in the
dataset of class 1(Let)
            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, 0.06122
4489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408163265307, 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, 0.139
3939393939394, 0.34545454545454546, 0.060606060606060608, 0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.46540880503144655, 0.0754
71698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.072847682119205295, 0.06622
5165562913912, 0.066225165562913912, 0.27152317880794702, 0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.06666666666666666, 0.17999999999999999, 0.073333333333333334, 0.073333333333333334, 0.0933
33333333333338, 0.080000000000000002, 0.29999999999999999, 0.06666666666666666, 0.06666666666666666],
    #      ...
    #     }
    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 data
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there in the train data then we wil
l 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 [24]:

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

```
Number of Unique Genes in train data: 232
brca1     164
tp53      112
egfr       98
pten       78
brca2      78
braf       67
kit        58
erbb2      52
alk        43
pdgfra     38
Name: Gene, dtype: int64
```

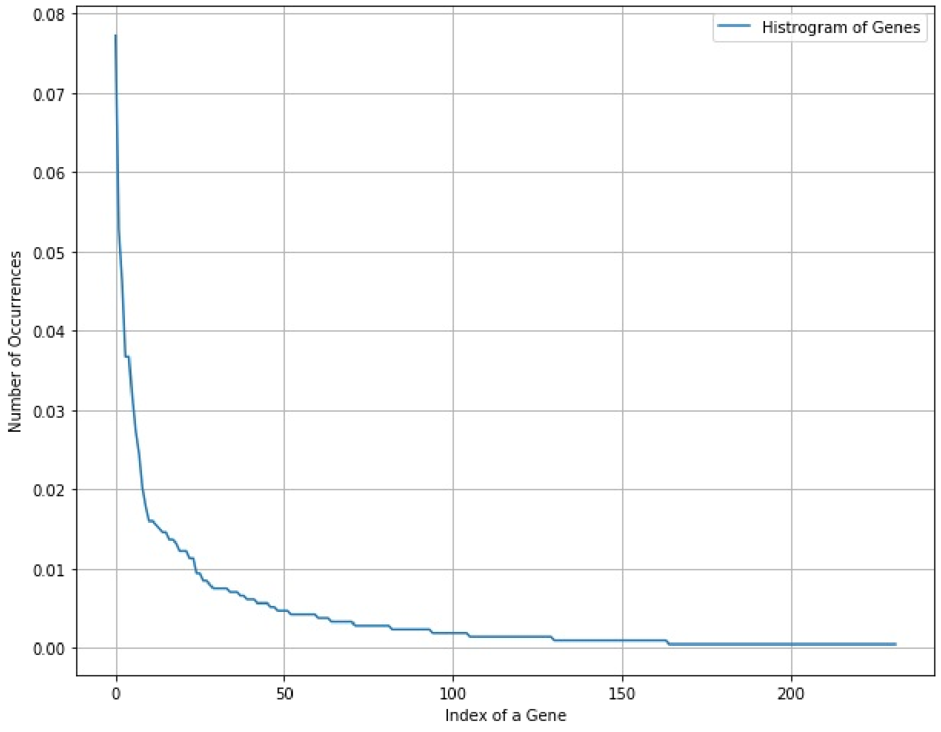
In [25]:

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

```
Ans: There are 232 different categories of genes in the training data, and they are distibuted as fo
llows
```

In [26]:

```
#Histogram of distribution of genes
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.figure(figsize=(10,8))
plt.plot(h, label="Histrogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurrences')
plt.legend()
plt.grid()
plt.show()
```

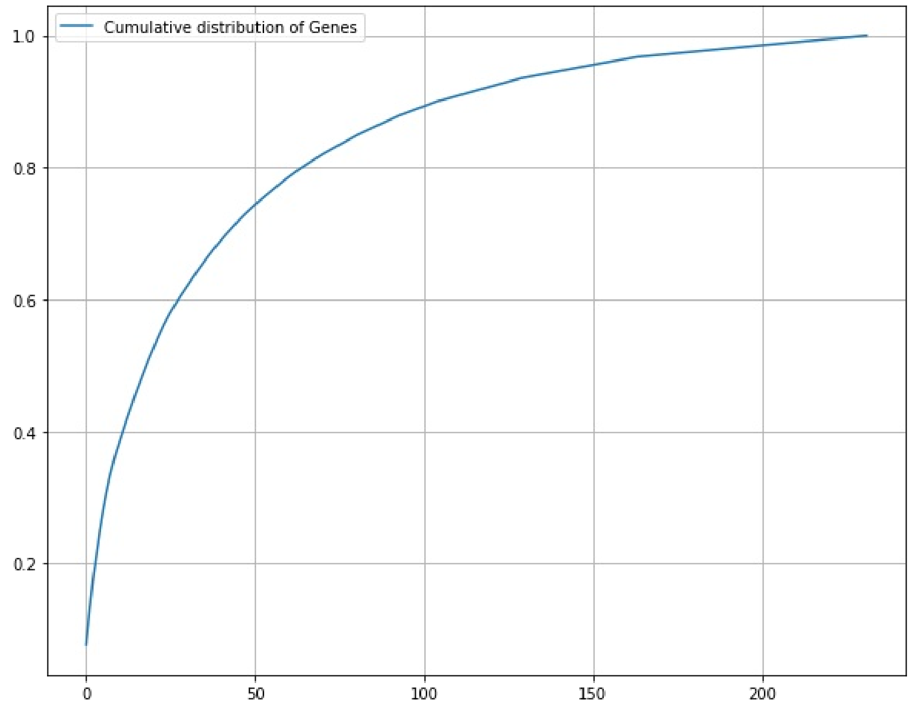


Key Take-away from the above Histogram:

The dsitributions of the gene is very skewed towards the left. There are almost 50 genes which occurs most frequently, and there are almost 200 odd genes which occurs less frequently than the first 50.

In [27]:

```
#Get the CDF of gene
c = np.cumsum(h)
plt.figure(figsize=(10,8))
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Key Take-away from the above CDF Plot:

The top 50 genes contribute to almost 75% of our data. The rest of the 185 genes contribute to only 25% of the data. This says that the top 50% gene mutation type occurs most frquently. There are also many gene/mutation pair types which occurs very less frequently.

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 wont use response coding for this problem, since we will apply Logistic Regression only)

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 [28]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer()
train_gene_feature_ngrams = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_ngrams = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_ngrams = gene_vectorizer.transform(cv_df['Gene'])

print("train_gene_feature_ngrams is converted to a feature vector using one-hot coding method. The shape of gene
feature:", train_gene_feature_ngrams.shape)
print("test_gene_feature_ngrams is converted to a feature vector using one-hot coding method. The shape of gene f
eature:", test_gene_feature_ngrams.shape)
print("cv_gene_feature_ngrams is converted to a feature vector using one-hot coding method. The shape of gene fea
ture:", cv_gene_feature_ngrams.shape)
```

```
train_gene_feature_ngrams is converted to a feature vector using one-hot coding method. The shape of
gene feature: (2124, 231)
test_gene_feature_ngrams is converted to a feature vector using one-hot coding method. The shape of
gene feature: (665, 231)
cv_gene_feature_ngrams is converted to a feature vector using one-hot coding method. The shape of ge
ne feature: (532, 231)
```

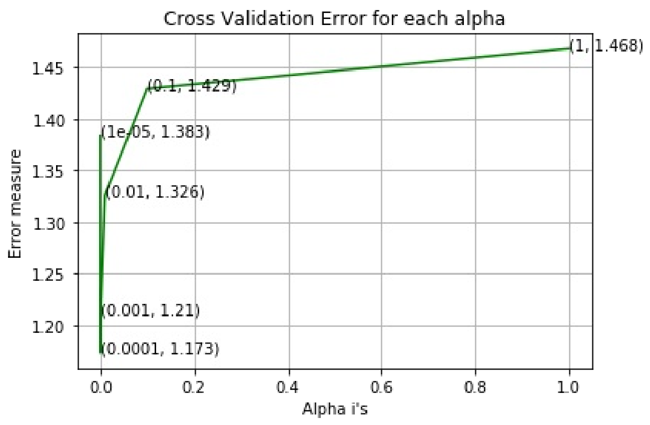
Q4. How good is this gene feature in predicting y_i?

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

In [29]:

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

cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42, n_jobs=-1)
    clf.fit(train_gene_feature_ngrams, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_ngrams, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_ngrams)
    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-1
5))

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


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

predict_y = sig_clf.predict_proba(train_gene_feature_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, lab
els=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_gene_feature_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predic
t_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_gene_feature_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, label
s=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.3834638133376587
For values of alpha =  0.0001 The log loss is: 1.1728398961194682
For values of alpha =  0.001 The log loss is: 1.210049153634091
For values of alpha =  0.01 The log loss is: 1.3259866999653074
For values of alpha =  0.1 The log loss is: 1.4288925889081667
For values of alpha =  1 The log loss is: 1.4677930068349208
```



```
For values of best alpha =  0.0001 The train log loss is: 1.0521022774745334
For values of best alpha =  0.0001 The cross validation log loss is: 1.1728398961194682
For values of best alpha =  0.0001 The test log loss is: 1.181597942368326
```

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

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

In [30]:

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

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.shape[0])*100)
```

```
Q6. How many data points in Test and CV datasets are covered by the  232  genes in train dataset?

Ans.
1. In test data 648 out of 665 : 97.44360902255639
2. In cross validation data 513 out of  532 : 96.42857142857143
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

In [31]:

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

```
Number of Unique Variations : 1941
truncating_mutations    60
deletion                48
amplification           37
fusions                 16
overexpression           6
e17k                     3
g12v                     3
q61h                     3
g13c                     2
s222d                    2
Name: Variation, dtype: int64
```

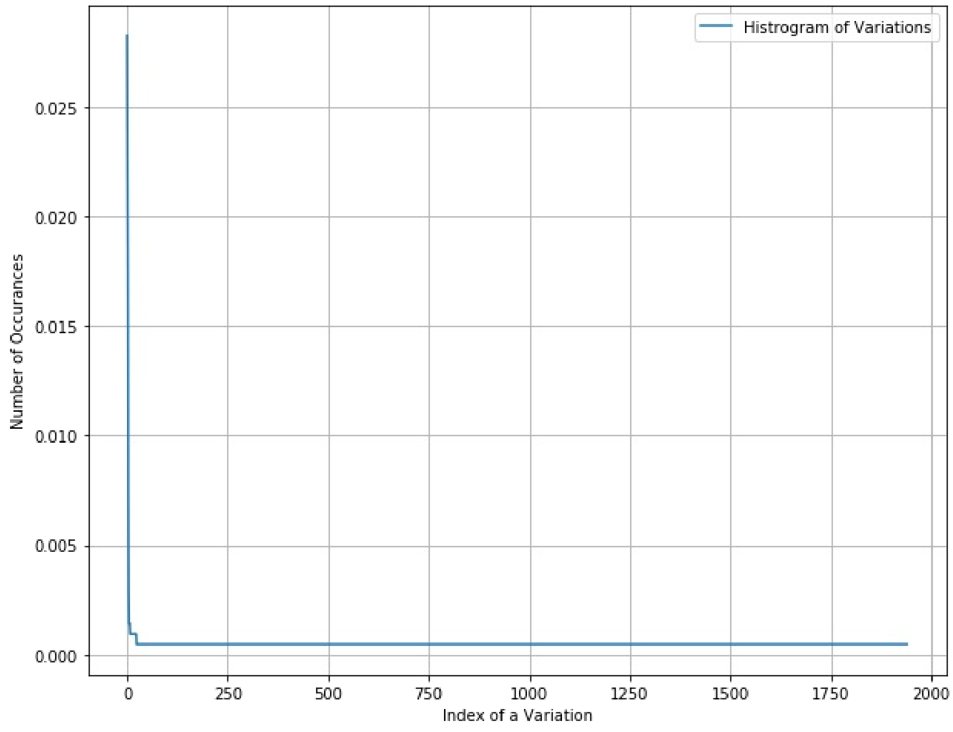
In [32]:

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

```
Ans: There are 1941 different categories of variations in the train data, and they are distibuted as
follows
```

In [33]:

```python
#Histogram for the distribution of the fetaure Variation
plt.figure(figsize=(10,8))
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()
```



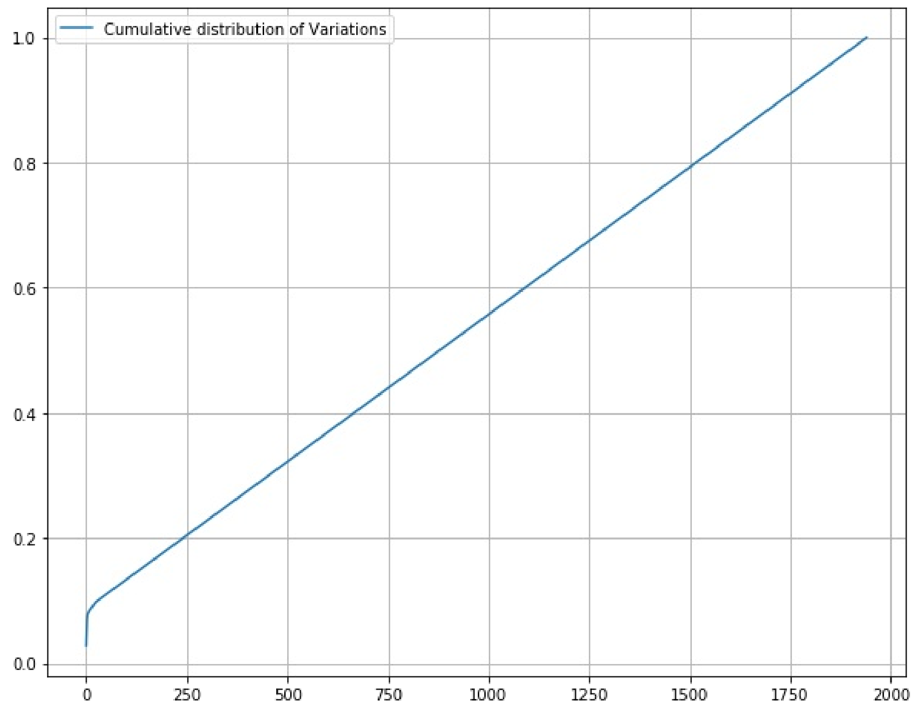
Key Take-aways:

The distribtion is highly skewed. This means there are extremely small number of variations which have a significant higher value. Also, most variations doesn't occur more than once. Most of the variations occurs 1 or less number of times.

In [34]:

```
#CDF Of variation feature
plt.figure(figsize=(10,8))
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.02824859 0.05084746 0.06826742 ... 0.99905838 0.99952919 1.        ]
```



Key Take-aways:

This CDF suggest that almost 80% of the dataset is explained by 1500 variations (out of a total number of 1927). If you see a CDF which is almost a straight line, you can quickly tell that most variations occur once or maybe twice in the entire training data.

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

Very Important note about response coding: Avoiding response leakage.

1. We have to be extremely careful not to use the test and cross validate data for response coding. This is because we don't the issue of data leakage.
2. Suppose we have a variant V2 present in Test / Cross Val dataset. But V2 is not present in Train. So in that case, while building the response bales for V2, we will just assign equal probability values to each of the 9 array values. Proba = 1/9 for each og them.
3. We will take the help of laplace smoothing in order to achieve this. Without laplace smoothing, we would get a 0/0 error.
4. We are seeing the data that is present in the cross validation and test data during the time of training. So we are literally leaking the information that is present in test/cv data at the time of training.

This should be strictly avoided, as we do not want a data leakage issue.

In [35]:

```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_ngrams = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_ngrams = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_ngrams = variation_vectorizer.transform(cv_df['Variation'])

print("train_variation_feature_ngrams is converted feature using the one-hot encoding method. The shape of Variat
ion feature:", train_variation_feature_ngrams.shape)
print("test_variation_feature_ngrams is converted feature using the one-hot encoding method. The shape of Variati
on feature:", test_variation_feature_ngrams.shape)
print("cv_variation_feature_ngrams is converted feature using the one-hot encoding method. The shape of Variation
feature:", cv_variation_feature_ngrams.shape)
```

```
train_variation_feature_ngrams is converted feature using the one-hot encoding method. The shape of
Variation feature: (2124, 1978)
test_variation_feature_ngrams is converted feature using the one-hot encoding method. The shape of V
ariation feature: (665, 1978)
cv_variation_feature_ngrams is converted feature using the one-hot encoding method. The shape of Var
iation feature: (532, 1978)
```

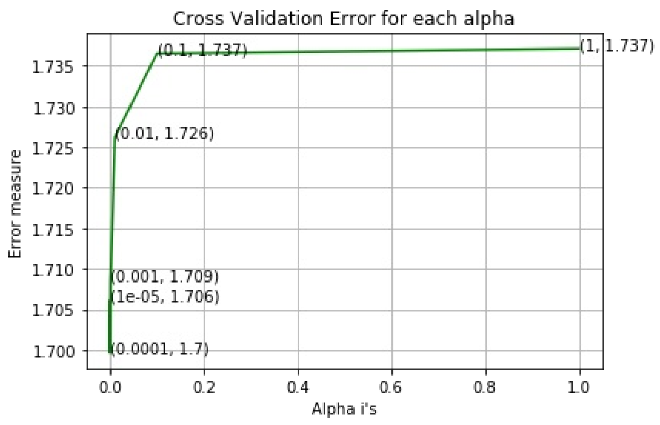
Q10. How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

In [36]:

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

cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42, n_jobs=-1)
    clf.fit(train_variation_feature_ngrams, y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_variation_feature_ngrams, y_train)
    predict_y = sig_clf.predict_proba(cv_variation_feature_ngrams)

    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-1
5))

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


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

predict_y = sig_clf.predict_proba(train_variation_feature_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, lab
els=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_variation_feature_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predic
t_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_variation_feature_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, label
s=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.7060629965879814
For values of alpha =  0.0001 The log loss is: 1.6996368037692522
For values of alpha =  0.001 The log loss is: 1.7085125166114756
For values of alpha =  0.01 The log loss is: 1.7261785496961948
For values of alpha =  0.1 The log loss is: 1.7365149999806846
For values of alpha =  1 The log loss is: 1.73710769672235
```



```
For values of best alpha =  0.0001 The train log loss is: 0.7465236501313196
For values of best alpha =  0.0001 The cross validation log loss is: 1.6996368037692522
For values of best alpha =  0.0001 The test log loss is: 1.6976923557358157
```

Conclusion:

1. The variation feature is certainly useful in determining the class labels, as it has brought down the log loss of a random model by a significant level.
2. However, if we compare the test and cross validation log loss of variation features to that of the gene features, we see that test/cross validation log loss for variations is quite high as compared to genes. This suggests that the variation feature might be unstable.
3. Also, there is a significant difference in log loss for train and test/cross-validation. This means the model is overfitting using only the variation feature.
4. Having said all these, we will still keep the variation feature as it has managed to decrease the log-loss of a random model. We will use this feature along with other features to see how it behaves when we build our final model.

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.

Q12 . How many data points are covered by total 1924 variations in test and cross validation data sets?

In [37]:

```python
print("Q12. How many data points are covered by total ", unique_variations.shape[0], " variations in test and cro
ss validation data sets?\n")
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.shape[0])*100)
```

```
Q12. How many data points are covered by total  1941  variations in test and cross validation data s
ets?

Ans.
1. In test data 77 out of 665 : 11.578947368421053
2. In cross validation data 59 out of  532 : 11.090225563909774
```

3.2.3 Univariate Analysis on Text Feature

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

In [38]:

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

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

In [39]:

```
# building a TFIDF text vectorizer with all the words
text_vectorizer = TfidfVectorizer()
train_text_feature_ngrams = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

#Copy the train_text_feature_ngrams into a seperate variable
train_text_feature_ngrams_ones = train_text_feature_ngrams.copy()
train_text_feature_ngrams_ones.data = train_text_feature_ngrams_ones.data / train_text_feature_ngrams_ones.data
train_text_fea_counts=[]
col_sum= train_text_feature_ngrams_ones.sum(axis=0).A1

#Convert all the float values to integers
train_text_fea_counts = [int(col_sum[i]) for i in range(0,len(col_sum))]

# 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 the 'Text' field :", len(train_text_features))

# we use the same vectorizer that was trained on train data
test_text_feature_ngrams = text_vectorizer.transform(test_df['TEXT'])
cv_text_feature_ngrams = text_vectorizer.transform(cv_df['TEXT'])
```

```
Total number of unique words in the 'Text' field : 124023
```

In [40]:

```
dict_list = []
# dict_list =[] contains 9 dictionaries 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)
confuse_array.shape
```

Out[40]:

```
(124023, 9)
```

In [41]:

```
#train_text_feature_ngrams = normalize(train_text_feature_ngrams, axis=0)
#test_text_feature_ngrams = normalize(test_text_feature_ngrams, axis=0)
#cv_text_feature_ngrams = normalize(cv_text_feature_ngrams, axis=0)
```

In [ ]:

In [42]:

```
#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 [43]:

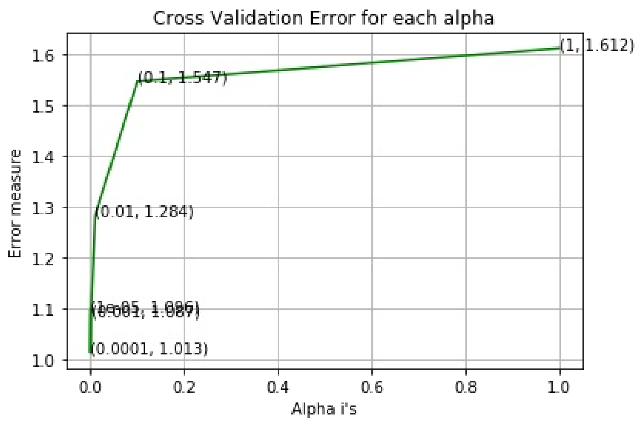
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({1: 54031, 2: 17239, 3: 8169, 4: 5464, 5: 3851, 6: 3197, 9: 2486, 7: 2302, 8: 2055, 10: 1407
, 11: 1199, 13: 1160, 14: 1058, 12: 1030, 15: 767, 16: 761, 17: 624, 19: 520, 18: 505, 23: 502, 20:
480, 22: 429, 25: 370, 21: 345, 24: 332, 40: 330, 26: 307, 31: 272, 32: 271, 28: 259, 49: 252, 27: 2
40, 39: 230, 29: 222, 30: 198, 37: 186, 35: 186, 34: 177, 33: 177, 36: 170, 41: 155, 42: 146, 43: 14
3, 54: 123, 48: 122, 50: 119, 52: 117, 38: 117, 45: 115, 51: 111, 44: 110, 46: 108, 47: 106, 57: 105
, 55: 97, 67: 94, 53: 92, 61: 91, 58: 88, 59: 87, 60: 84, 56: 84, 63: 81, 64: 78, 71: 77, 70: 77, 69
: 72, 66: 72, 65: 71, 62: 67, 78: 65, 68: 64, 76: 62, 79: 58, 81: 57, 74: 57, 73: 57, 80: 55, 72: 54
, 92: 53, 75: 51, 85: 49, 87: 47, 89: 45, 83: 45, 90: 44, 105: 43, 99: 43, 88: 43, 114: 40, 82: 40,
106: 39, 84: 39, 100: 38, 91: 38, 86: 38, 77: 38, 113: 37, 109: 37, 108: 37, 107: 37, 101: 37, 102:
35, 95: 35, 97: 34, 96: 34, 110: 33, 104: 33, 93: 33, 122: 32, 111: 31, 117: 30, 116: 30, 115: 30, 1
40: 29, 119: 29, 132: 28, 129: 28, 103: 28, 94: 28, 136: 27, 126: 27, 112: 27, 98: 27, 157: 26, 123:
26, 118: 26, 171: 25, 151: 25, 149: 25, 131: 25, 124: 25, 160: 24, 156: 24, 128: 24, 127: 24, 121: 2
4, 139: 23, 120: 23, 153: 22, 145: 22, 166: 21, 165: 21, 155: 21, 144: 21, 138: 21, 133: 21, 175: 20
, 159: 20, 147: 20, 125: 20, 218: 19, 170: 19, 167: 19, 154: 19, 150: 19, 148: 19, 142: 19, 141: 19,
130: 19, 194: 18, 190: 18, 188: 18, 177: 18, 173: 18, 169: 18, 137: 18, 249: 17, 225: 17, 204: 17, 1
85: 17, 162: 17, 161: 17, 152: 17, 135: 17, 134: 17, 246: 16, 214: 16, 203: 16, 195: 16, 181: 16, 18
0: 16, 178: 16, 168: 16, 158: 16, 251: 15, 241: 15, 196: 15, 193: 15, 187: 15, 184: 15, 182: 15, 179
: 15, 143: 15, 357: 14, 240: 14, 226: 14, 201: 14, 197: 14, 189: 14, 183: 14, 176: 14, 174: 14, 336:
13, 295: 13, 266: 13, 227: 13, 224: 13, 223: 13, 217: 13, 207: 13, 192: 13, 164: 13, 163: 13, 146: 1
3, 300: 12, 275: 12, 237: 12, 235: 12, 222: 12, 221: 12, 206: 12, 200: 12, 191: 12, 172: 12, 377: 11
, 371: 11, 320: 11, 318: 11, 311: 11, 292: 11, 291: 11, 268: 11, 264: 11, 252: 11, 243: 11, 233: 11,
228: 11, 213: 11, 212: 11, 199: 11, 198: 11, 381: 10, 338: 10, 331: 10, 305: 10, 293: 10, 280: 10, 2
63: 10, 256: 10, 255: 10, 254: 10, 253: 10, 248: 10, 247: 10, 238: 10, 220: 10, 211: 10, 209: 10, 20
5: 10, 202: 10, 682: 9, 429: 9, 392: 9, 384: 9, 380: 9, 365: 9, 362: 9, 351: 9, 349: 9, 323: 9, 316:
9, 294: 9, 288: 9, 279: 9, 272: 9, 262: 9, 245: 9, 234: 9, 232: 9, 219: 9, 208: 9, 481: 8, 454: 8, 4
41: 8, 423: 8, 406: 8, 379: 8, 373: 8, 369: 8, 347: 8, 302: 8, 298: 8, 284: 8, 282: 8, 281: 8, 276:
8, 274: 8, 269: 8, 261: 8, 244: 8, 242: 8, 215: 8, 210: 8, 742: 7, 719: 7, 691: 7, 664: 7, 594: 7, 5
38: 7, 489: 7, 458: 7, 440: 7, 437: 7, 426: 7, 422: 7, 399: 7, 398: 7, 394: 7, 393: 7, 361: 7, 352:
7, 343: 7, 341: 7, 333: 7, 325: 7, 319: 7, 306: 7, 273: 7, 271: 7, 259: 7, 250: 7, 239: 7, 236: 7, 9
57: 6, 939: 6, 908: 6, 813: 6, 789: 6, 683: 6, 665: 6, 627: 6, 593: 6, 585: 6, 581: 6, 578: 6, 562:
6, 551: 6, 543: 6, 542: 6, 527: 6, 522: 6, 516: 6, 511: 6, 504: 6, 494: 6, 491: 6, 487: 6, 468: 6, 4
63: 6, 459: 6, 457: 6, 450: 6, 449: 6, 433: 6, 428: 6, 415: 6, 413: 6, 410: 6, 408: 6, 401: 6, 389:
6, 386: 6, 375: 6, 372: 6, 368: 6, 355: 6, 354: 6, 353: 6, 342: 6, 339: 6, 335: 6, 334: 6, 330: 6, 3
26: 6, 324: 6, 322: 6, 321: 6, 317: 6, 308: 6, 303: 6, 299: 6, 289: 6, 285: 6, 283: 6, 278: 6, 277:
6, 270: 6, 257: 6, 231: 6, 230: 6, 186: 6, 1141: 5, 1083: 5, 1020: 5, 907: 5, 874: 5, 853: 5, 830: 5
, 780: 5, 748: 5, 701: 5, 686: 5, 678: 5, 676: 5, 667: 5, 654: 5, 632: 5, 610: 5, 579: 5, 558: 5, 55
4: 5, 549: 5, 539: 5, 514: 5, 510: 5, 498: 5, 490: 5, 486: 5, 485: 5, 471: 5, 461: 5, 436: 5, 420: 5
, 412: 5, 407: 5, 395: 5, 391: 5, 390: 5, 388: 5, 378: 5, 370: 5, 360: 5, 359: 5, 356: 5, 350: 5, 34
6: 5, 337: 5, 329: 5, 327: 5, 314: 5, 312: 5, 307: 5, 296: 5, 287: 5, 267: 5, 265: 5, 258: 5, 216: 5
, 1452: 4, 1377: 4, 1345: 4, 1299: 4, 1269: 4, 1027: 4, 983: 4, 934: 4, 917: 4, 910: 4, 859: 4, 838:
4, 793: 4, 769: 4, 746: 4, 720: 4, 711: 4, 692: 4, 687: 4, 685: 4, 673: 4, 669: 4, 659: 4, 650: 4, 6
44: 4, 630: 4, 628: 4, 618: 4, 600: 4, 591: 4, 586: 4, 577: 4, 573: 4, 568: 4, 566: 4, 560: 4, 544:
4, 540: 4, 526: 4, 523: 4, 520: 4, 518: 4, 513: 4, 508: 4, 507: 4, 505: 4, 501: 4, 500: 4, 488: 4, 4
80: 4, 476: 4, 465: 4, 456: 4, 453: 4, 447: 4, 446: 4, 442: 4, 431: 4, 430: 4, 427: 4, 421: 4, 414:
4, 411: 4, 409: 4, 405: 4, 403: 4, 397: 4, 396: 4, 385: 4, 382: 4, 367: 4, 366: 4, 364: 4, 363: 4, 3
32: 4, 313: 4, 310: 4, 309: 4, 304: 4, 297: 4, 290: 4, 286: 4, 1700: 3, 1637: 3, 1523: 3, 1460: 3, 1
384: 3, 1334: 3, 1309: 3, 1308: 3, 1274: 3, 1187: 3, 1175: 3, 1171: 3, 1143: 3, 1142: 3, 1087: 3, 10
75: 3, 1061: 3, 1052: 3, 1049: 3, 1039: 3, 1037: 3, 1032: 3, 1023: 3, 1018: 3, 1016: 3, 1011: 3, 100
2: 3, 990: 3, 987: 3, 979: 3, 977: 3, 952: 3, 943: 3, 940: 3, 928: 3, 924: 3, 922: 3, 902: 3, 889: 3
, 883: 3, 850: 3, 847: 3, 834: 3, 833: 3, 822: 3, 819: 3, 809: 3, 796: 3, 782: 3, 779: 3, 772: 3, 77
1: 3, 770: 3, 766: 3, 763: 3, 762: 3, 756: 3, 751: 3, 736: 3, 732: 3, 727: 3, 726: 3, 725: 3, 722: 3
, 714: 3, 707: 3, 697: 3, 693: 3, 689: 3, 679: 3, 674: 3, 668: 3, 660: 3, 652: 3, 649: 3, 647: 3, 64
5: 3, 643: 3, 642: 3, 637: 3, 625: 3, 624: 3, 617: 3, 613: 3, 611: 3, 609: 3, 603: 3, 601: 3, 597: 3
, 595: 3, 588: 3, 576: 3, 575: 3, 574: 3, 572: 3, 570: 3, 567: 3, 564: 3, 563: 3, 557: 3, 556: 3, 54
8: 3, 546: 3, 532: 3, 530: 3, 529: 3, 528: 3, 524: 3, 521: 3, 499: 3, 492: 3, 477: 3, 474: 3, 472: 3
, 469: 3, 464: 3, 460: 3, 455: 3, 451: 3, 444: 3, 443: 3, 439: 3, 438: 3, 435: 3, 434: 3, 432: 3, 42
4: 3, 419: 3, 418: 3, 417: 3, 404: 3, 402: 3, 400: 3, 383: 3, 374: 3, 348: 3, 345: 3, 344: 3, 301: 3
, 2057: 2, 1996: 2, 1934: 2, 1896: 2, 1894: 2, 1891: 2, 1889: 2, 1873: 2, 1868: 2, 1833: 2, 1827: 2,
1802: 2, 1794: 2, 1787: 2, 1781: 2, 1771: 2, 1757: 2, 1732: 2, 1718: 2, 1686: 2, 1685: 2, 1680: 2, 1
677: 2, 1652: 2, 1644: 2, 1635: 2, 1610: 2, 1592: 2, 1589: 2, 1579: 2, 1574: 2, 1570: 2, 1562: 2, 15
51: 2, 1513: 2, 1512: 2, 1504: 2, 1485: 2, 1471: 2, 1464: 2, 1459: 2, 1425: 2, 1421: 2, 1416: 2, 140
8: 2, 1395: 2, 1392: 2, 1391: 2, 1389: 2, 1379: 2, 1376: 2, 1356: 2, 1355: 2, 1349: 2, 1343: 2, 1328
: 2, 1326: 2, 1323: 2, 1320: 2, 1285: 2, 1281: 2, 1260: 2, 1255: 2, 1253: 2, 1242: 2, 1238: 2, 1228:
2, 1223: 2, 1216: 2, 1212: 2, 1210: 2, 1207: 2, 1205: 2, 1194: 2, 1192: 2, 1182: 2, 1181: 2, 1172: 2
, 1170: 2, 1160: 2, 1156: 2, 1154: 2, 1149: 2, 1138: 2, 1135: 2, 1119: 2, 1102: 2, 1101: 2, 1100: 2,
1094: 2, 1080: 2, 1078: 2, 1074: 2, 1072: 2, 1070: 2, 1067: 2, 1055: 2, 1043: 2, 1038: 2, 1034: 2, 1
033: 2, 1022: 2, 1017: 2, 1009: 2, 1008: 2, 1007: 2, 1004: 2, 996: 2, 984: 2, 980: 2, 975: 2, 973: 2
, 971: 2, 970: 2, 967: 2, 961: 2, 960: 2, 959: 2, 953: 2, 950: 2, 945: 2, 942: 2, 936: 2, 935: 2, 92
5: 2, 923: 2, 916: 2, 914: 2, 912: 2, 911: 2, 909: 2, 905: 2, 900: 2, 894: 2, 893: 2, 890: 2, 887: 2
, 885: 2, 882: 2, 877: 2, 875: 2, 867: 2, 863: 2, 848: 2, 845: 2, 844: 2, 843: 2, 840: 2, 829: 2, 82
8: 2, 826: 2, 825: 2, 818: 2, 814: 2, 811: 2, 806: 2, 805: 2, 800: 2, 797: 2, 795: 2, 785: 2, 783: 2
, 777: 2, 775: 2, 774: 2, 773: 2, 764: 2, 760: 2, 758: 2, 757: 2, 754: 2, 752: 2, 740: 2, 739: 2, 73
5: 2, 734: 2, 733: 2, 728: 2, 724: 2, 723: 2, 721: 2, 715: 2, 708: 2, 706: 2, 705: 2, 704: 2, 699: 2
, 698: 2, 696: 2, 690: 2, 688: 2, 684: 2, 681: 2, 680: 2, 677: 2, 672: 2, 671: 2, 662: 2, 661: 2, 65
8: 2, 657: 2, 656: 2, 655: 2, 646: 2, 639: 2, 636: 2, 635: 2, 633: 2, 629: 2, 621: 2, 616: 2, 615: 2
, 614: 2, 612: 2, 608: 2, 607: 2, 602: 2, 592: 2, 589: 2, 582: 2, 565: 2, 559: 2, 555: 2, 552: 2, 54
5: 2, 541: 2, 537: 2, 536: 2, 531: 2, 517: 2, 509: 2, 506: 2, 502: 2, 497: 2, 496: 2, 495: 2, 483: 2
, 479: 2, 478: 2, 473: 2, 470: 2, 467: 2, 448: 2, 445: 2, 416: 2, 387: 2, 340: 2, 328: 2, 315: 2, 26
0: 2, 229: 2, 2102: 1, 2099: 1, 2085: 1, 2083: 1, 2076: 1, 2075: 1, 2072: 1, 2069: 1, 2067: 1, 2065:
1, 2064: 1, 2059: 1, 2051: 1, 2047: 1, 2046: 1, 2031: 1, 2024: 1, 2023: 1, 2013: 1, 2010: 1, 2008: 1
, 2007: 1, 2004: 1, 2001: 1, 2000: 1, 1999: 1, 1990: 1, 1989: 1, 1987: 1, 1982: 1, 1981: 1, 1977: 1,
1973: 1, 1968: 1, 1966: 1, 1963: 1, 1956: 1, 1955: 1, 1954: 1, 1953: 1, 1952: 1, 1949: 1, 1944: 1, 1
940: 1, 1931: 1, 1921: 1, 1917: 1, 1916: 1, 1912: 1, 1911: 1, 1906: 1, 1903: 1, 1901: 1, 1899: 1, 18
80: 1, 1874: 1, 1871: 1, 1861: 1, 1858: 1, 1850: 1, 1848: 1, 1842: 1, 1841: 1, 1839: 1, 1838: 1, 183
7: 1, 1836: 1, 1830: 1, 1829: 1, 1824: 1, 1821: 1, 1818: 1, 1813: 1, 1807: 1, 1806: 1, 1805: 1, 1804
: 1, 1797: 1, 1795: 1, 1793: 1, 1790: 1, 1778: 1, 1777: 1, 1772: 1, 1770: 1, 1768: 1, 1765: 1, 1764:
1, 1754: 1, 1753: 1, 1748: 1, 1746: 1, 1743: 1, 1741: 1, 1735: 1, 1733: 1, 1730: 1, 1726: 1, 1720: 1
, 1719: 1, 1717: 1, 1715: 1, 1713: 1, 1708: 1, 1703: 1, 1688: 1, 1682: 1, 1681: 1, 1679: 1, 1671: 1,
1666: 1, 1664: 1, 1659: 1, 1658: 1, 1651: 1, 1649: 1, 1645: 1, 1642: 1, 1640: 1, 1638: 1, 1622: 1, 1
618: 1, 1615: 1, 1614: 1, 1607: 1, 1605: 1, 1603: 1, 1601: 1, 1599: 1, 1598: 1, 1594: 1, 1593: 1, 15
88: 1, 1583: 1, 1582: 1, 1580: 1, 1578: 1, 1577: 1, 1576: 1, 1568: 1, 1567: 1, 1566: 1, 1565: 1, 156
1: 1, 1559: 1, 1556: 1, 1553: 1, 1552: 1, 1549: 1, 1546: 1, 1542: 1, 1541: 1, 1540: 1, 1533: 1, 1527
: 1, 1525: 1, 1521: 1, 1518: 1, 1517: 1, 1516: 1, 1515: 1, 1510: 1, 1509: 1, 1503: 1, 1498: 1, 1497:
1, 1488: 1, 1482: 1, 1478: 1, 1475: 1, 1474: 1, 1473: 1, 1468: 1, 1467: 1, 1466: 1, 1462: 1, 1456: 1
, 1455: 1, 1451: 1, 1448: 1, 1447: 1, 1444: 1, 1441: 1, 1439: 1, 1431: 1, 1430: 1, 1428: 1, 1427: 1,
1424: 1, 1423: 1, 1420: 1, 1419: 1, 1418: 1, 1417: 1, 1414: 1, 1413: 1, 1412: 1, 1405: 1, 1404: 1, 1
403: 1, 1402: 1, 1400: 1, 1399: 1, 1398: 1, 1397: 1, 1393: 1, 1388: 1, 1386: 1, 1381: 1, 1368: 1, 13
62: 1, 1354: 1, 1348: 1, 1341: 1, 1337: 1, 1336: 1, 1329: 1, 1327: 1, 1324: 1, 1321: 1, 1315: 1, 131
4: 1, 1307: 1, 1304: 1, 1303: 1, 1302: 1, 1297: 1, 1296: 1, 1294: 1, 1291: 1, 1290: 1, 1288: 1, 1287
: 1, 1280: 1, 1279: 1, 1275: 1, 1273: 1, 1271: 1, 1270: 1, 1268: 1, 1265: 1, 1264: 1, 1263: 1, 1261:
1, 1258: 1, 1256: 1, 1251: 1, 1250: 1, 1249: 1, 1243: 1, 1239: 1, 1236: 1, 1231: 1, 1227: 1, 1221: 1
, 1218: 1, 1214: 1, 1198: 1, 1197: 1, 1196: 1, 1195: 1, 1188: 1, 1185: 1, 1184: 1, 1183: 1, 1180: 1,
1179: 1, 1178: 1, 1176: 1, 1168: 1, 1162: 1, 1161: 1, 1157: 1, 1145: 1, 1139: 1, 1136: 1, 1134: 1, 1
133: 1, 1131: 1, 1130: 1, 1129: 1, 1126: 1, 1125: 1, 1124: 1, 1121: 1, 1117: 1, 1115: 1, 1114: 1, 11
13: 1, 1112: 1, 1109: 1, 1107: 1, 1106: 1, 1105: 1, 1103: 1, 1099: 1, 1098: 1, 1097: 1, 1096: 1, 109
5: 1, 1093: 1, 1090: 1, 1089: 1, 1084: 1, 1082: 1, 1081: 1, 1073: 1, 1071: 1, 1069: 1, 1066: 1, 1063
: 1, 1062: 1, 1057: 1, 1054: 1, 1053: 1, 1051: 1, 1050: 1, 1046: 1, 1045: 1, 1036: 1, 1029: 1, 1028:
1, 1026: 1, 1025: 1, 1014: 1, 1010: 1, 1006: 1, 1001: 1, 1000: 1, 999: 1, 998: 1, 997: 1, 991: 1, 98
9: 1, 988: 1, 986: 1, 985: 1, 982: 1, 968: 1, 966: 1, 964: 1, 954: 1, 951: 1, 949: 1, 948: 1, 947: 1
, 944: 1, 931: 1, 930: 1, 921: 1, 919: 1, 918: 1, 915: 1, 906: 1, 903: 1, 901: 1, 899: 1, 897: 1, 89
2: 1, 888: 1, 886: 1, 884: 1, 881: 1, 880: 1, 878: 1, 876: 1, 873: 1, 872: 1, 869: 1, 866: 1, 865: 1
, 862: 1, 861: 1, 860: 1, 858: 1, 852: 1, 851: 1, 849: 1, 846: 1, 837: 1, 836: 1, 835: 1, 831: 1, 82
7: 1, 823: 1, 821: 1, 820: 1, 815: 1, 812: 1, 808: 1, 804: 1, 802: 1, 801: 1, 799: 1, 792: 1, 791: 1
, 790: 1, 776: 1, 767: 1, 765: 1, 761: 1, 759: 1, 755: 1, 753: 1, 750: 1, 749: 1, 745: 1, 744: 1, 72
9: 1, 717: 1, 716: 1, 713: 1, 712: 1, 710: 1, 709: 1, 703: 1, 702: 1, 695: 1, 675: 1, 670: 1, 666: 1
, 663: 1, 651: 1, 648: 1, 640: 1, 634: 1, 622: 1, 620: 1, 619: 1, 606: 1, 596: 1, 587: 1, 584: 1, 58
3: 1, 571: 1, 569: 1, 561: 1, 553: 1, 550: 1, 547: 1, 535: 1, 533: 1, 525: 1, 519: 1, 515: 1, 512: 1
, 503: 1, 493: 1, 484: 1, 482: 1, 475: 1, 466: 1, 462: 1, 452: 1, 425: 1, 376: 1, 358: 1})
```

In [44]:

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

cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42, n_jobs=-1)
    clf.fit(train_text_feature_ngrams, y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_text_feature_ngrams, y_train)
    predict_y = sig_clf.predict_proba(cv_text_feature_ngrams)
    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-1
5))

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_ngrams, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_text_feature_ngrams, y_train)

predict_y = sig_clf.predict_proba(train_text_feature_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, lab
els=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_text_feature_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predic
t_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_text_feature_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, label
s=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.0958853596608644
For values of alpha =  0.0001 The log loss is: 1.0126892719196874
For values of alpha =  0.001 The log loss is: 1.087109945747949
For values of alpha =  0.01 The log loss is: 1.2836994639471517
For values of alpha =  0.1 The log loss is: 1.5469957857409378
For values of alpha =  1 The log loss is: 1.6121995021186528
```



```
For values of best alpha =  0.0001 The train log loss is: 0.7103696214586092
For values of best alpha =  0.0001 The cross validation log loss is: 1.0126892719196874
For values of best alpha =  0.0001 The test log loss is: 1.030736006299603
```

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

Ans. Yes, it seems like!

In [45]:

```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer()
    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 [46]:

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

```
78.897 % of words of test data appeared in train data
77.657 % of words of Cross Validation appeared in train data
```

3.2.4 Univariate Analysis on TopWordsInText Feature

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

In [47]:

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

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

In [48]:

```
# building a tfidf text vectorizer with all the words
topWords_vectorizer = TfidfVectorizer()
train_topWords_feature_ngrams = topWords_vectorizer.fit_transform(train_df['TopWordsInText'])
# getting all the feature names (words)
train_topWords_features= topWords_vectorizer.get_feature_names()

#Copy the train_topWords_feature_ngrams into a seperate variable
train_topWords_feature_ngrams_ones = train_topWords_feature_ngrams.copy()
train_topWords_feature_ngrams_ones.data = train_topWords_feature_ngrams_ones.data / train_topWords_feature_ngrams
_ones.data

# train_topWords_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of features) vector
train_topWords_fea_counts=[]
col_sum= train_topWords_feature_ngrams_ones.sum(axis=0).A1

#Convert all the float values to integers
train_topWords_fea_counts = [int(col_sum[i]) for i in range(0,len(col_sum))]

# zip(list(topWords_features),topWords_fea_counts) will zip a word with its number of times it occured
topWords_fea_dict = dict(zip(list(train_topWords_features),train_topWords_fea_counts))

print("Total number of unique words in topWords data :", len(train_topWords_features))

# we use the same vectorizer that was trained on train data
test_topWords_feature_ngrams = topWords_vectorizer.transform(test_df['TopWordsInText'])
cv_topWords_feature_ngrams = topWords_vectorizer.transform(cv_df['TopWordsInText'])
```

```
Total number of unique words in topWords data : 8416
```

In [49]:

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

print("train_topWords_feature_ngrams is converted feature using the one-hot encoding method. The shape of Variati
on feature:", train_topWords_feature_ngrams.shape)
print("test_topWords_feature_ngrams is converted feature using the one-hot encoding method. The shape of Variatio
n feature:", test_topWords_feature_ngrams.shape)
print("cv_topWords_feature_ngrams is converted feature using the one-hot encoding method. The shape of Variation
feature:", cv_topWords_feature_ngrams.shape)
```

```
train_topWords_feature_ngrams is converted feature using the one-hot encoding method. The shape of V
ariation feature: (2124, 8416)
test_topWords_feature_ngrams is converted feature using the one-hot encoding method. The shape of Va
riation feature: (665, 8416)
cv_topWords_feature_ngrams is converted feature using the one-hot encoding method. The shape of Vari
ation feature: (532, 8416)
```

In [50]:

```
dict_list = []
# dict_list =[] contains 9 dictionaries 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_topWords(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_topWords(train_df)


confuse_array = []
for i in train_topWords_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 [51]:

```
#https://stackoverflow.com/a/2258273/4084039
sorted_topWords_fea_dict = dict(sorted(topWords_fea_dict.items(), key=lambda x: x[1] , reverse=True))
sorted_topWords_occur = np.array(list(sorted_topWords_fea_dict.values()))
```

In [52]:

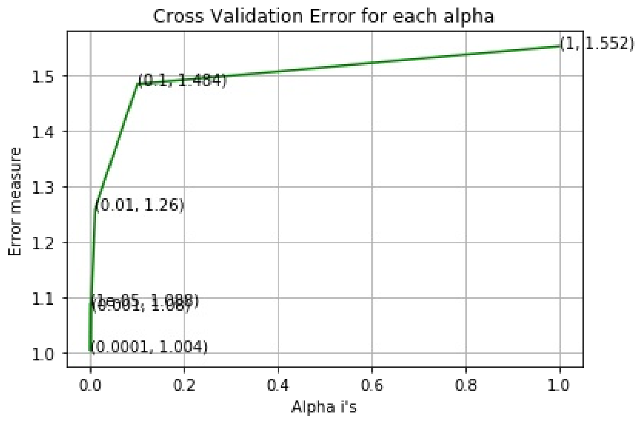
```
# Number of words for a given frequency.
print(Counter(sorted_topWords_occur))
```

```
Counter({1: 3398, 2: 1208, 3: 736, 4: 439, 5: 325, 6: 236, 7: 197, 8: 157, 9: 108, 11: 100, 10: 97,
12: 96, 13: 71, 14: 64, 15: 60, 17: 51, 18: 40, 20: 39, 22: 34, 19: 34, 26: 29, 16: 27, 21: 24, 25:
22, 30: 21, 29: 21, 31: 20, 28: 20, 24: 20, 41: 18, 38: 18, 36: 18, 35: 18, 32: 18, 23: 18, 42: 16,
33: 16, 39: 15, 45: 14, 53: 13, 37: 13, 49: 12, 40: 12, 27: 10, 69: 9, 43: 9, 77: 8, 52: 8, 48: 8, 4
4: 8, 65: 7, 63: 7, 62: 7, 50: 7, 46: 7, 91: 6, 80: 6, 66: 6, 64: 6, 60: 6, 59: 6, 56: 6, 54: 6, 114
: 5, 99: 5, 90: 5, 84: 5, 67: 5, 58: 5, 57: 5, 51: 5, 47: 5, 106: 4, 103: 4, 92: 4, 89: 4, 86: 4, 78
: 4, 74: 4, 73: 4, 72: 4, 70: 4, 68: 4, 55: 4, 34: 4, 282: 3, 245: 3, 165: 3, 159: 3, 158: 3, 146: 3
, 144: 3, 132: 3, 123: 3, 121: 3, 113: 3, 111: 3, 108: 3, 98: 3, 94: 3, 83: 3, 75: 3, 71: 3, 1039: 2
, 389: 2, 386: 2, 343: 2, 337: 2, 309: 2, 290: 2, 263: 2, 260: 2, 254: 2, 240: 2, 235: 2, 221: 2, 21
1: 2, 202: 2, 192: 2, 189: 2, 186: 2, 175: 2, 164: 2, 160: 2, 157: 2, 140: 2, 139: 2, 138: 2, 130: 2
, 129: 2, 128: 2, 126: 2, 120: 2, 117: 2, 112: 2, 110: 2, 105: 2, 96: 2, 88: 2, 81: 2, 79: 2, 76: 2,
61: 2, 1886: 1, 1587: 1, 1425: 1, 1385: 1, 1376: 1, 1324: 1, 1275: 1, 1219: 1, 1212: 1, 1192: 1, 116
0: 1, 1150: 1, 1124: 1, 1109: 1, 1094: 1, 1008: 1, 961: 1, 953: 1, 935: 1, 910: 1, 842: 1, 826: 1, 8
25: 1, 813: 1, 804: 1, 795: 1, 758: 1, 753: 1, 732: 1, 730: 1, 689: 1, 647: 1, 618: 1, 612: 1, 594:
1, 571: 1, 569: 1, 546: 1, 536: 1, 533: 1, 511: 1, 509: 1, 506: 1, 491: 1, 488: 1, 461: 1, 445: 1, 4
43: 1, 434: 1, 413: 1, 412: 1, 405: 1, 393: 1, 373: 1, 370: 1, 367: 1, 364: 1, 361: 1, 358: 1, 348:
1, 344: 1, 336: 1, 333: 1, 328: 1, 324: 1, 319: 1, 318: 1, 314: 1, 312: 1, 307: 1, 303: 1, 297: 1, 2
96: 1, 295: 1, 293: 1, 278: 1, 271: 1, 266: 1, 262: 1, 255: 1, 239: 1, 238: 1, 236: 1, 231: 1, 230:
1, 227: 1, 224: 1, 223: 1, 220: 1, 218: 1, 214: 1, 213: 1, 208: 1, 199: 1, 198: 1, 193: 1, 191: 1, 1
79: 1, 171: 1, 169: 1, 168: 1, 166: 1, 163: 1, 162: 1, 154: 1, 153: 1, 152: 1, 149: 1, 147: 1, 145:
1, 143: 1, 141: 1, 137: 1, 136: 1, 135: 1, 133: 1, 131: 1, 127: 1, 122: 1, 116: 1, 109: 1, 107: 1, 1
04: 1, 102: 1, 101: 1, 97: 1, 87: 1, 85: 1, 82: 1})
```

In [53]:

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

cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42, n_jobs=-1)
    clf.fit(train_topWords_feature_ngrams, y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_topWords_feature_ngrams, y_train)
    predict_y = sig_clf.predict_proba(cv_topWords_feature_ngrams)
    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-1
5))

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_topWords_feature_ngrams, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_topWords_feature_ngrams, y_train)

predict_y = sig_clf.predict_proba(train_topWords_feature_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, lab
els=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_topWords_feature_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predic
t_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_topWords_feature_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, label
s=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.087994837795309
For values of alpha =  0.0001 The log loss is: 1.0041434655497734
For values of alpha =  0.001 The log loss is: 1.0802389189087285
For values of alpha =  0.01 The log loss is: 1.2598898592104588
For values of alpha =  0.1 The log loss is: 1.4837069877970215
For values of alpha =  1 The log loss is: 1.5518628612987364
```



```
For values of best alpha =  0.0001 The train log loss is: 0.6157581486372681
For values of best alpha =  0.0001 The cross validation log loss is: 1.0041434655497734
For values of best alpha =  0.0001 The test log loss is: 1.0089648455179328
```

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

Ans. Yes, it seems like!

In [54]:

```
def get_intersec_topWords(df):
    df_topWords_vec = TfidfVectorizer()
    df_topWords_fea = df_topWords_vec.fit_transform(df['TopWordsInText'])
    df_topWords_features = df_topWords_vec.get_feature_names()

    df_topWords_fea_counts = df_topWords_fea.sum(axis=0).A1
    df_topWords_fea_dict = dict(zip(list(df_topWords_features),df_topWords_fea_counts))
    len1 = len(set(df_topWords_features))
    len2 = len(set(train_topWords_features) & set(df_topWords_features))
    return len1,len2
```

In [55]:

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

```
80.094 % of word of Test data appeared in train data
82.981 % of word of Cross Validation appeared in train data
```

4.1 Feature Engineering Part 1

In [56]:

```
#Here we will build a list which would contain all the gene and variation data into one single list.
#We will use this list to featurize our main text data. In another section, we will use the text in TopWordsInTex
t feature to
#featurize the text data. For now let's stick to gene and variation data and buil ML models on top of it.
gene_var = []

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

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

In [57]:

```
tfidfVectorizer = TfidfVectorizer(max_features=1200).fit(gene_var)
gene_var_features = tfidfVectorizer.get_feature_names()

train_text_gene_var_encoding = tfidfVectorizer.transform(train_df['TEXT'])
test_text_gene_var_encoding = tfidfVectorizer.transform(test_df['TEXT'])
cv_text_gene_var_encoding = tfidfVectorizer.transform(cv_df['TEXT'])

train_text_gene_var_encoding.shape
```

Out[57]:

```
(2124, 1200)
```

4.2 Stacking all the features

In [58]:

```
train_gene_var_ngrams = hstack((train_gene_feature_ngrams,train_variation_feature_ngrams)) #Stack gene tfidf vect
or with variation vector
test_gene_var_ngrams = hstack((test_gene_feature_ngrams,test_variation_feature_ngrams))
cv_gene_var_ngrams = hstack((cv_gene_feature_ngrams,cv_variation_feature_ngrams))

train_x_ngrams = hstack((train_gene_var_ngrams, train_text_feature_ngrams)) #Stack gene-variane vector with text
vector encoded with gene-variants tfidf
train_x_ngrams = hstack((train_x_ngrams, train_text_gene_var_encoding)).tocsr() #Stack train_x_grams with the tfi
df of text data only
train_y = np.array(list(train_df['Class']))

test_x_ngrams = hstack((test_gene_var_ngrams,test_text_feature_ngrams))
test_x_ngrams = hstack((test_x_ngrams, test_text_gene_var_encoding)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_ngrams = hstack((cv_gene_var_ngrams, cv_text_feature_ngrams))
cv_x_ngrams = hstack((cv_x_ngrams, cv_text_gene_var_encoding)).tocsr()
cv_y = np.array(list(cv_df['Class']))
```

In [59]:

```
print("TFIDF Encoding features:\n")
print("(Number of data points * number of features) in train data = ", train_x_ngrams.shape)
print("(Number of data points * number of features) in test data = ", test_x_ngrams.shape)
print("(Number of data points * number of features) in cross validation data =", cv_x_ngrams.shape)
```

```
TFIDF Encoding features:

(Number of data points * number of features) in train data =  (2124, 127432)
(Number of data points * number of features) in test data =  (665, 127432)
(Number of data points * number of features) in cross validation data = (532, 127432)
```

5. Machine Learning Models

In [60]:

```
#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("Test Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Percentage of mis-classified points :", 100*np.count_nonzero((pred_y- test_y))/test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

In [61]:

```
# 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(ngram_range=(1,2))
    var_count_vec = CountVectorizer(ngram_range=(1,2))
    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_vec.get_feature_names()) #233
    fea2_len = len(var_count_vec.get_feature_names()) #2056

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

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

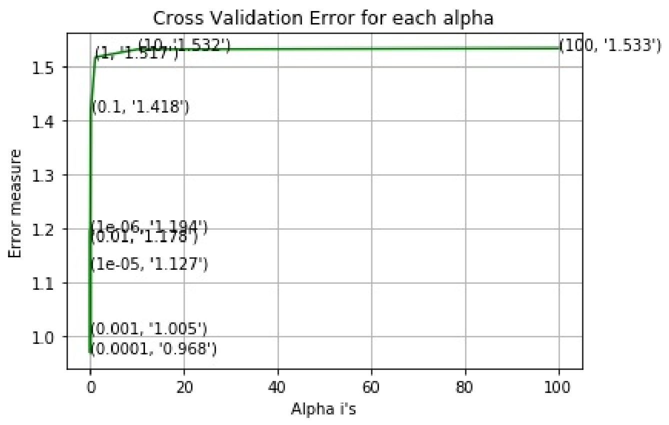
5.1 Logistic Regression With Class balancing

5.1.1 Hyperparameter tuning

In [62]:

```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42, n_jobs=-1)
    clf.fit(train_x_ngrams, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_ngrams, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_ngrams)
    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("For alpha =", i, ",Cross Validation Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

predict_y = sig_clf.predict_proba(train_x_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, lab
els=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predic
t_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, label
s=clf.classes_, eps=1e-15))
```

```
For alpha = 1e-06 ,Cross Validation Log Loss : 1.1938082219349238
For alpha = 1e-05 ,Cross Validation Log Loss : 1.1267125431135918
For alpha = 0.0001 ,Cross Validation Log Loss : 0.9680996491529648
For alpha = 0.001 ,Cross Validation Log Loss : 1.0048362899276124
For alpha = 0.01 ,Cross Validation Log Loss : 1.178157185671344
For alpha = 0.1 ,Cross Validation Log Loss : 1.4180231806524575
For alpha = 1 ,Cross Validation Log Loss : 1.5166107206619341
For alpha = 10 ,Cross Validation Log Loss : 1.531584549778998
For alpha = 100 ,Cross Validation Log Loss : 1.5333641321170333
```



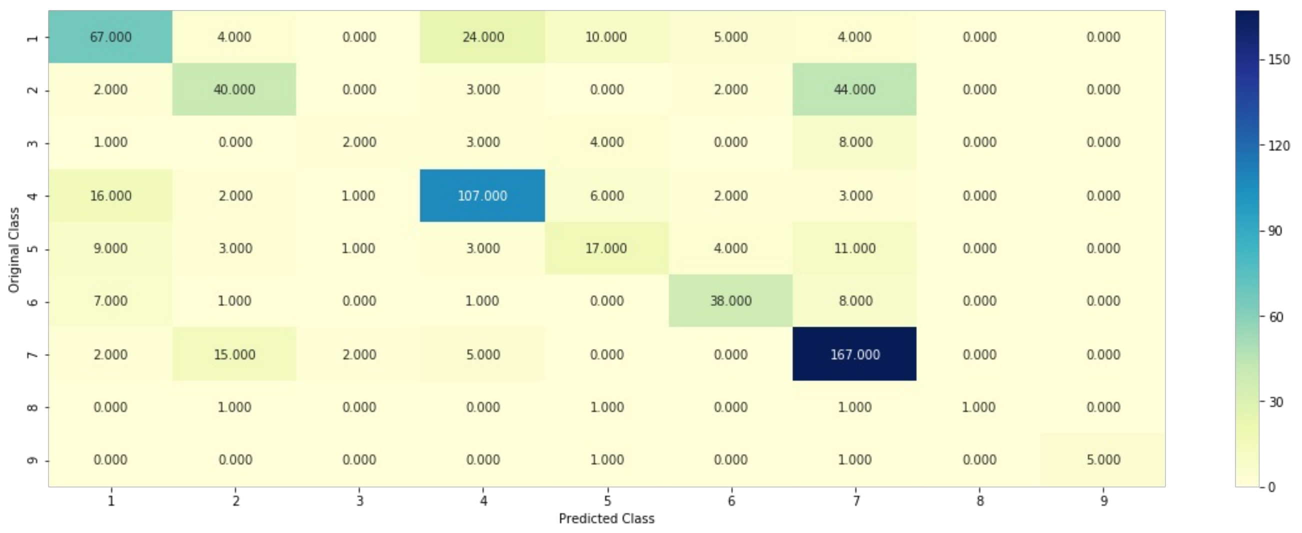
```
For values of best alpha =  0.0001 The train log loss is: 0.45617588208349236
For values of best alpha =  0.0001 The cross validation log loss is: 0.9680996491529648
For values of best alpha =  0.0001 The test log loss is: 0.9702887615398305
```

5.1.2. Testing the model with best hyper paramters

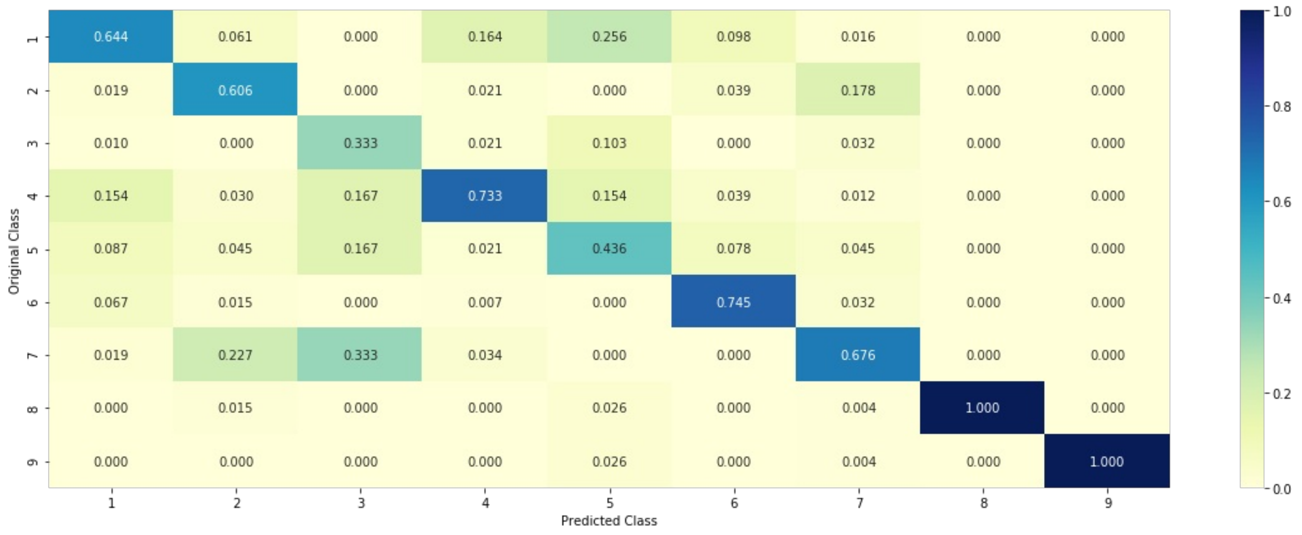
```
In [63]:
```

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_ngrams, train_y, test_x_ngrams, test_y, clf)
```

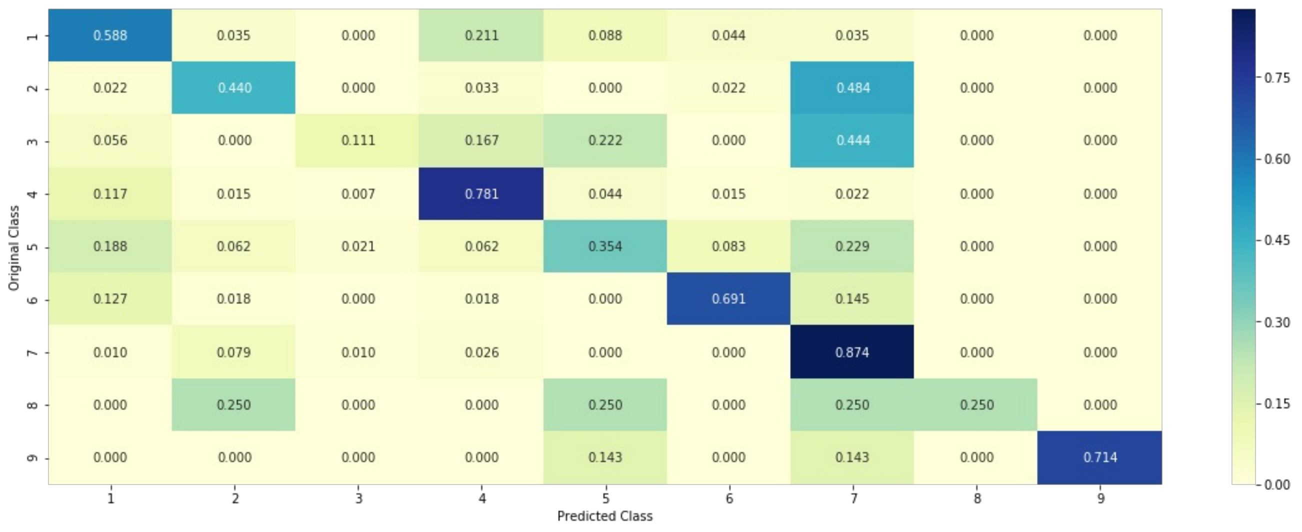
```
Test Log loss : 0.9702887615398305
Percentage of mis-classified points : 33.233082706766915
-------------------- Confusion matrix --------------------
```



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



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



5.1.3. Feature Importance

5.1.3.1 Correctly Classified point

In [64]:

```
test_point_index = 55
no_feature = 500
predicted_cls = sig_clf.predict(test_x_ngrams[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_ngrams[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices11 = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1024 0.0283 0.0149 0.7626 0.0254 0.0228 0.0331 0.0037 0.0069]]
Actual Class : 4
--------------------------------------------------
```

5.3.1.2 Incorrectly Classified point

In [65]:

```
test_point_index = 1
no_feature = 5000
predicted_cls = sig_clf.predict(test_x_ngrams[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_ngrams[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices12 = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[8.765e-01 3.300e-03 2.100e-03 5.540e-02 5.010e-02 1.000e-02 6.000e-
04
  1.100e-03 9.000e-04]]
Actual Class : 1
--------------------------------------------------
```

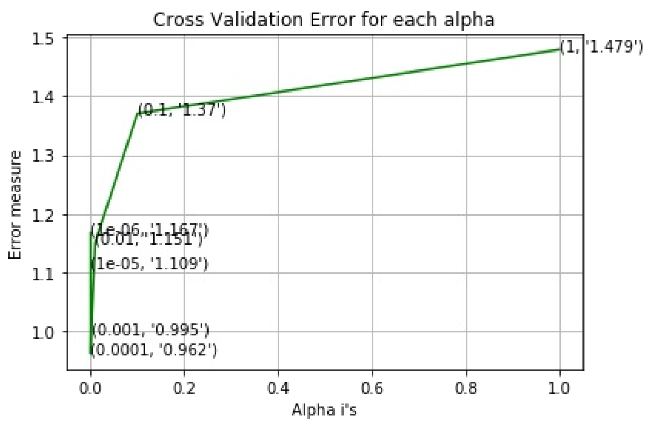
5.2 Logistic Regression Without Class balancing

5.2.1. Hyper paramter tuning

In [66]:

```
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, n_jobs=-1)
    clf.fit(train_x_ngrams, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_ngrams, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_ngrams)
    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_ngrams, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_ngrams, train_y)

predict_y = sig_clf.predict_proba(train_x_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, lab
els=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predic
t_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, label
s=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.1669110172567128
for alpha = 1e-05
Log Loss : 1.1094220807282107
for alpha = 0.0001
Log Loss : 0.9615973114690869
for alpha = 0.001
Log Loss : 0.9950731354532265
for alpha = 0.01
Log Loss : 1.1509385229596398
for alpha = 0.1
Log Loss : 1.3704186973866095
for alpha = 1
Log Loss : 1.478861107368434
```



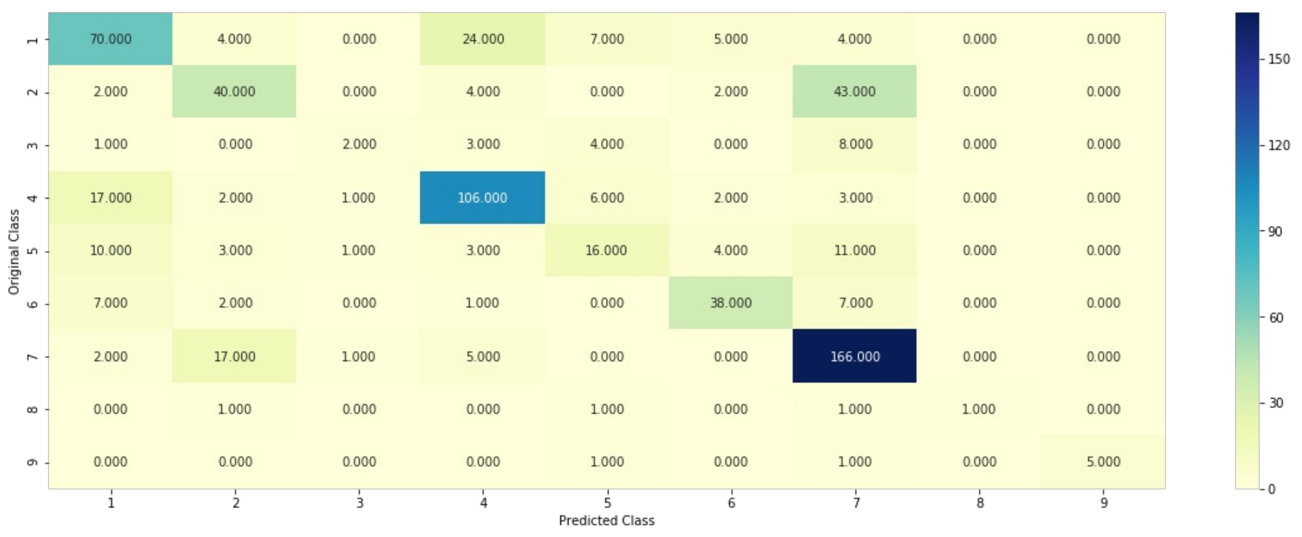
```
For values of best alpha =  0.0001 The train log loss is: 0.4451806180518291
For values of best alpha =  0.0001 The cross validation log loss is: 0.9615973114690869
For values of best alpha =  0.0001 The test log loss is: 0.9630606051594122
```

5.2.2 Testing model with best hyper parameters

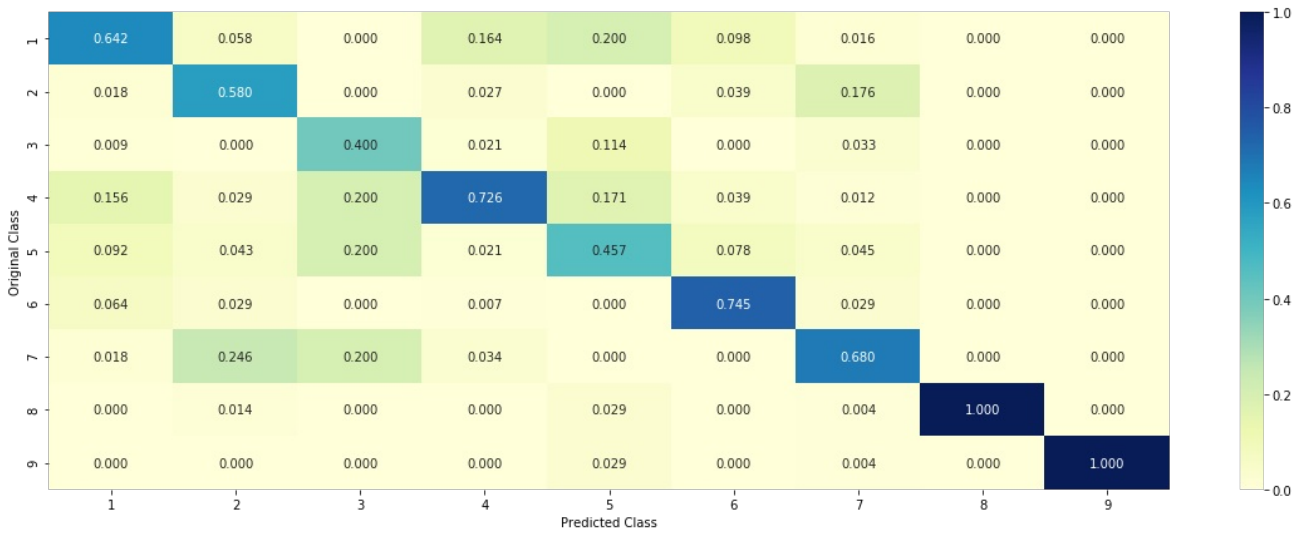
In [67]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_ngrams, train_y, test_x_ngrams, test_y, clf)
```

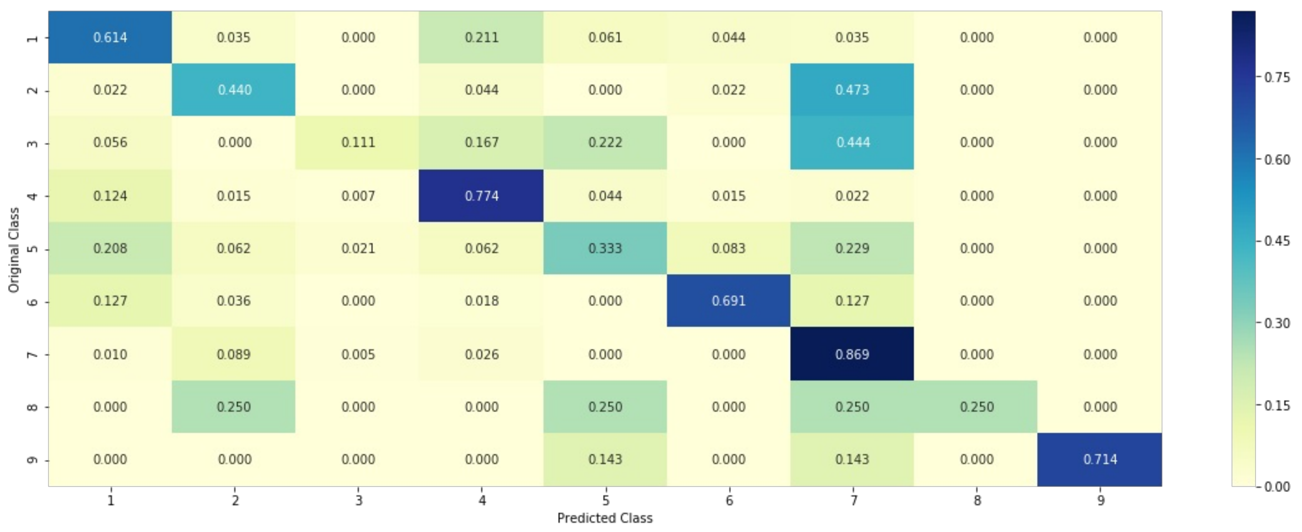
```
Test Log loss : 0.9630606051594122
Percentage of mis-classified points : 33.233082706766915
-------------------- Confusion matrix --------------------
```



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



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



5.2.3 Feature Importance, incorrectly Classified point

In [68]:

```
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_ngrams[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_ngrams[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices21 = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[8.696e-01 3.800e-03 1.600e-03 6.720e-02 4.700e-02 9.600e-03 1.100e-
03
  1.000e-04 1.000e-04]]
Actual Class : 1
--------------------------------------------------
```

5.2.4 Feature Importance, correctly Classified point

In [69]:

```
test_point_index = 4
no_feature = 500
predicted_cls = sig_clf.predict(test_x_ngrams[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_ngrams[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices22 = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0823 0.0635 0.0283 0.6464 0.0491 0.0409 0.066  0.0101 0.0134]]
Actual Class : 4
--------------------------------------------------
```

6. Feature Engineering Part 2

Stacking all the features

We will use these three featured engineered dataset that we have prepared in section: 3.2.4 =>
train_topWords_feature_ngrams,test_topWords_feature_ngrams,cv_topWords_feature_ngrams

In [70]:

```
train_x_ngrams = hstack((train_x_ngrams, train_topWords_feature_ngrams)).tocsr() #Stack train_x_ngrams with the t
fidf of train_topWords_feature_ngrams  data only
train_y = np.array(list(train_df['Class']))

test_x_ngrams = hstack((test_x_ngrams, test_topWords_feature_ngrams)).tocsr() #Stack train_x_ngrams with the tfid
f of test_topWords_feature_ngrams  data only
test_y = np.array(list(test_df['Class']))

cv_x_ngrams = hstack((cv_x_ngrams, cv_topWords_feature_ngrams)).tocsr() #Stack train_x_ngrams with the tfidf of c
v_topWords_feature_ngrams  data only
cv_y = np.array(list(cv_df['Class']))
```

In [71]:

```
print("TFIDF Encoding features:\n")
print("(Number of data points * number of features) in train data = ", train_x_ngrams.shape)
print("(Number of data points * number of features) in test data = ", test_x_ngrams.shape)
print("(Number of data points * number of features) in cross validation data =", cv_x_ngrams.shape)
```

```
TFIDF Encoding features:

(Number of data points * number of features) in train data =  (2124, 135848)
(Number of data points * number of features) in test data =  (665, 135848)
(Number of data points * number of features) in cross validation data = (532, 135848)
```

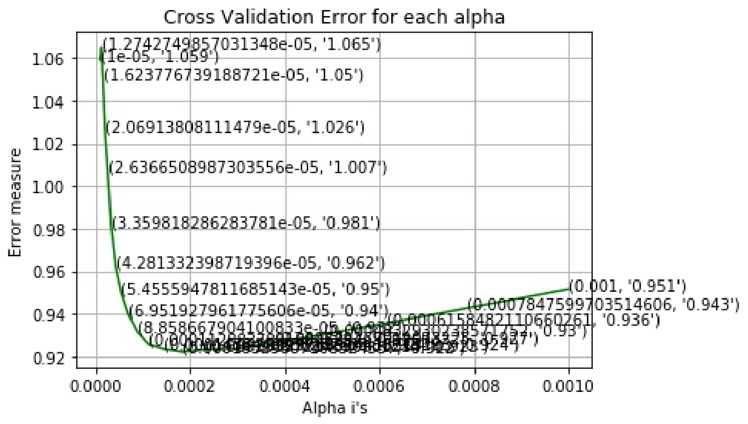
6.1 Logistic Regression with class balancing

6.1.2 Hyperparameter tuning

In [72]:

```python
alpha = np.logspace(-5,-3,20)
cv_log_error_array = []
for i in alpha:
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42, n_jobs=-1)
    clf.fit(train_x_ngrams, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_ngrams, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_ngrams)
    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("For alpha =", i, ",Cross Validation Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

predict_y = sig_clf.predict_proba(train_x_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, lab
els=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predic
t_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, label
s=clf.classes_, eps=1e-15))
```

```
For alpha = 1e-05 ,Cross Validation Log Loss : 1.0591105330844934
For alpha = 1.2742749857031348e-05 ,Cross Validation Log Loss : 1.0651157368727666
For alpha = 1.623776739188721e-05 ,Cross Validation Log Loss : 1.0501920603906307
For alpha = 2.06913808111479e-05 ,Cross Validation Log Loss : 1.025918744785124
For alpha = 2.6366508987303556e-05 ,Cross Validation Log Loss : 1.006744928970805
For alpha = 3.359818286283781e-05 ,Cross Validation Log Loss : 0.9810217275974863
For alpha = 4.281332398719396e-05 ,Cross Validation Log Loss : 0.9620411874270652
For alpha = 5.4555947811685143e-05 ,Cross Validation Log Loss : 0.9500249336094975
For alpha = 6.951927961775606e-05 ,Cross Validation Log Loss : 0.9398661694503202
For alpha = 8.858667904100833e-05 ,Cross Validation Log Loss : 0.9315514251280135
For alpha = 0.00011288378916846884 ,Cross Validation Log Loss : 0.925690053510427
For alpha = 0.0001438449888287663 ,Cross Validation Log Loss : 0.9235128784648815
For alpha = 0.00018329807108324357 ,Cross Validation Log Loss : 0.922075824133484
For alpha = 0.00023357214690901214 ,Cross Validation Log Loss : 0.92316270341751
For alpha = 0.00029763514416313193 ,Cross Validation Log Loss : 0.9236738417593916
For alpha = 0.000379269019073225 ,Cross Validation Log Loss : 0.9265607795013194
For alpha = 0.0004832930238571752 ,Cross Validation Log Loss : 0.9303136561520648
For alpha = 0.0006158482110660261 ,Cross Validation Log Loss : 0.9355547168379864
For alpha = 0.0007847599703514606 ,Cross Validation Log Loss : 0.9425928712689438
For alpha = 0.001 ,Cross Validation Log Loss : 0.9514842063189628
```



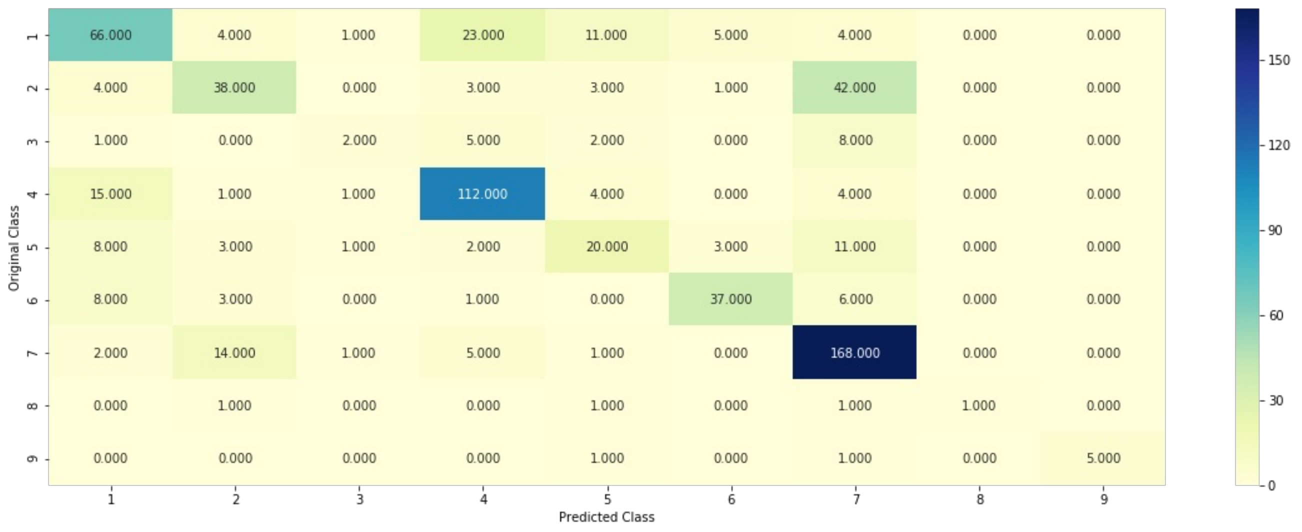
```
For values of best alpha =  0.00018329807108324357 The train log loss is: 0.4481001239769213
For values of best alpha =  0.00018329807108324357 The cross validation log loss is: 0.9220758241334
84
For values of best alpha =  0.00018329807108324357 The test log loss is: 0.9256930042114349
```

6.1.3 Testing the model with best hyper parameters

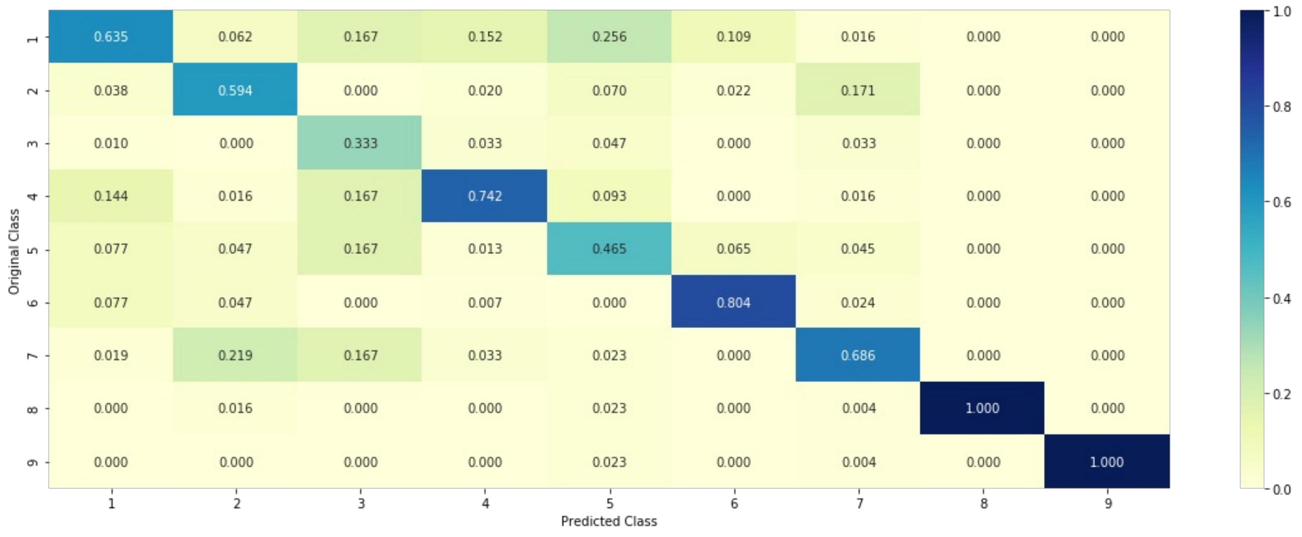
In [73]:

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_ngrams, train_y, test_x_ngrams, test_y, clf)
```

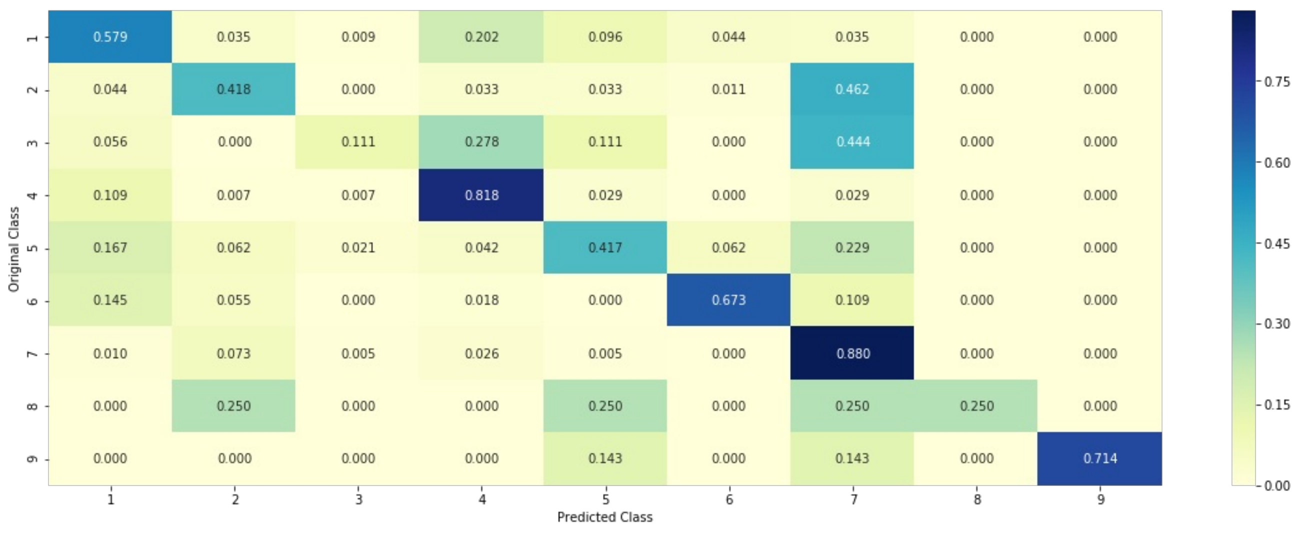
```
Test Log loss : 0.9256930042114349
Percentage of mis-classified points : 32.4812030075188
-------------------- Confusion matrix --------------------
```



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



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



6.1.4 Feature Importance

6.1.4.1 Correctly Classified point

In [74]:

```
test_point_index = 55
no_feature = 500
predicted_cls = sig_clf.predict(test_x_ngrams[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_ngrams[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices11 = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0847 0.0339 0.0153 0.7864 0.0226 0.0245 0.024  0.0034 0.0053]]
Actual Class : 4
--------------------------------------------------
```

6.1.4.2 Incorrectly Classified point

In [75]:

```
test_point_index = 1
no_feature = 5000
predicted_cls = sig_clf.predict(test_x_ngrams[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_ngrams[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices12 = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[9.132e-01 2.100e-03 2.300e-03 2.530e-02 4.000e-02 1.420e-02 1.400e-
03
  9.000e-04 6.000e-04]]
Actual Class : 1
--------------------------------------------------
```

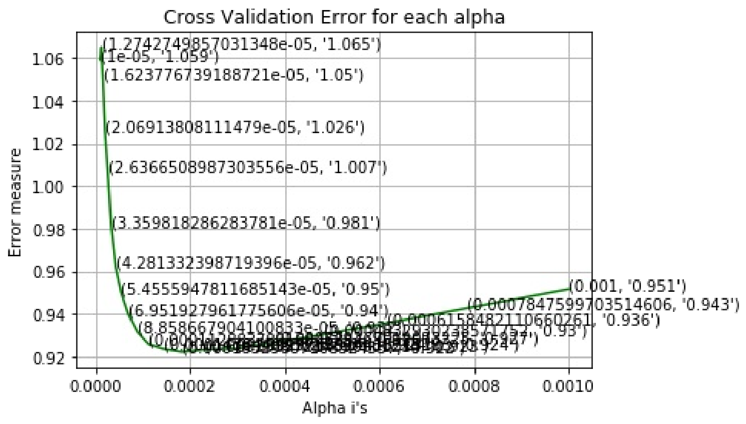
6.2 Logistic Regression without class balancing

6.2.2 Hyperparameter tuning

In [76]:

```
alpha = np.logspace(-5,-3,20)
cv_log_error_array = []
for i in alpha:
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42, n_jobs=-1)
    clf.fit(train_x_ngrams, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_ngrams, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_ngrams)
    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("For alpha =", i, ",Cross Validation 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_ngrams, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_ngrams, train_y)

predict_y = sig_clf.predict_proba(train_x_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, lab
els=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predic
t_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_ngrams)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, label
s=clf.classes_, eps=1e-15))
```

```
For alpha = 1e-05 ,Cross Validation Log Loss : 1.0591105330844934
For alpha = 1.2742749857031348e-05 ,Cross Validation Log Loss : 1.0651157368727666
For alpha = 1.623776739188721e-05 ,Cross Validation Log Loss : 1.0501920603906307
For alpha = 2.06913808111479e-05 ,Cross Validation Log Loss : 1.025918744785124
For alpha = 2.6366508987303556e-05 ,Cross Validation Log Loss : 1.006744928970805
For alpha = 3.359818286283781e-05 ,Cross Validation Log Loss : 0.9810217275974863
For alpha = 4.281332398719396e-05 ,Cross Validation Log Loss : 0.9620411874270652
For alpha = 5.4555947811685143e-05 ,Cross Validation Log Loss : 0.9500249336094975
For alpha = 6.951927961775606e-05 ,Cross Validation Log Loss : 0.9398661694503202
For alpha = 8.858667904100833e-05 ,Cross Validation Log Loss : 0.9315514251280135
For alpha = 0.00011288378916846884 ,Cross Validation Log Loss : 0.925690053510427
For alpha = 0.0001438449888287663 ,Cross Validation Log Loss : 0.9235128784648815
For alpha = 0.00018329807108324357 ,Cross Validation Log Loss : 0.922075824133484
For alpha = 0.00023357214690901214 ,Cross Validation Log Loss : 0.92316270341751
For alpha = 0.00029763514416313193 ,Cross Validation Log Loss : 0.9236738417593916
For alpha = 0.000379269019073225 ,Cross Validation Log Loss : 0.9265607795013194
For alpha = 0.0004832930238571752 ,Cross Validation Log Loss : 0.9303136561520648
For alpha = 0.0006158482110660261 ,Cross Validation Log Loss : 0.9355547168379864
For alpha = 0.0007847599703514606 ,Cross Validation Log Loss : 0.9425928712689438
For alpha = 0.001 ,Cross Validation Log Loss : 0.9514842063189628
```



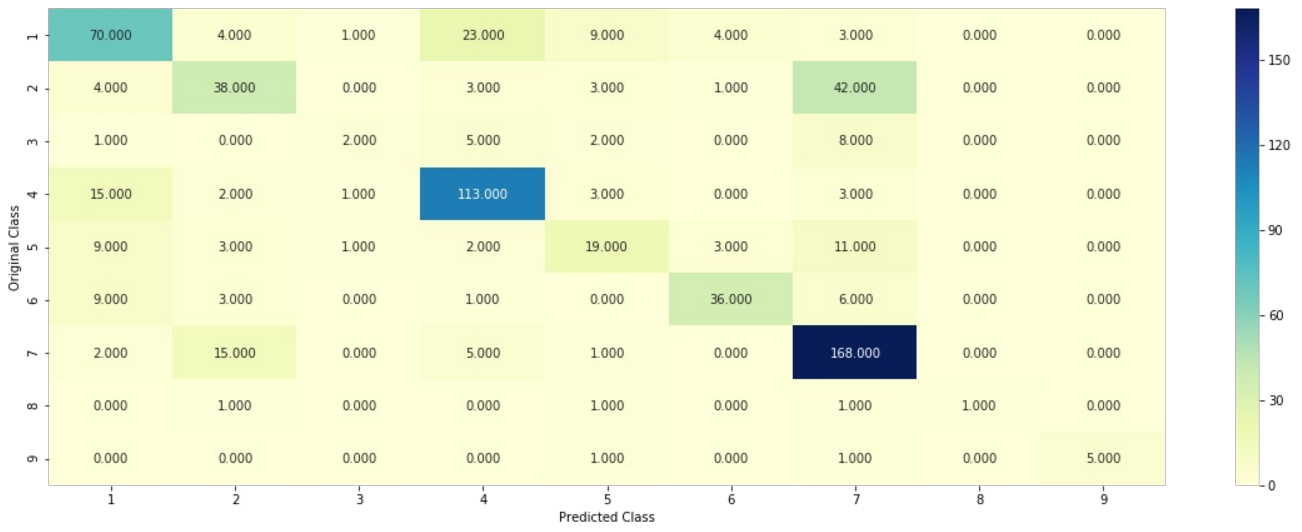
```
For values of best alpha =  0.00018329807108324357 The train log loss is: 0.4388639696718273
For values of best alpha =  0.00018329807108324357 The cross validation log loss is: 0.9189424989302
44
For values of best alpha =  0.00018329807108324357 The test log loss is: 0.9209398107353178
```

6.2.3 Testing the model with best hyper parameters

In [77]:

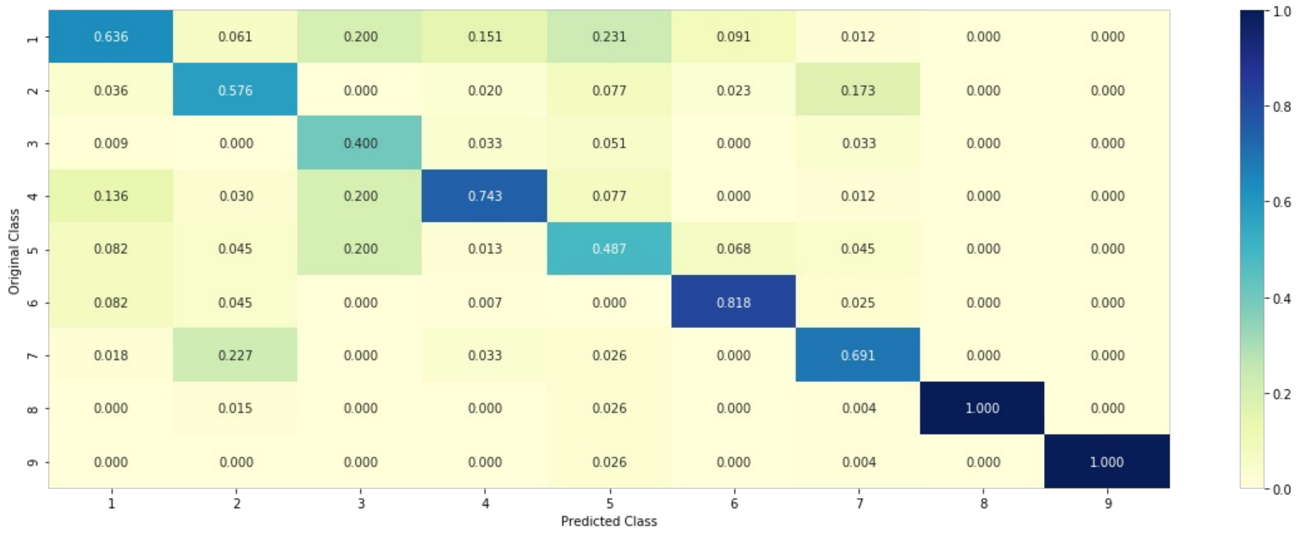
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_ngrams, train_y, test_x_ngrams, test_y, clf)
```

```
Test Log loss : 0.9209398107353178
Percentage of mis-classified points : 32.03007518796993
-------------------- Confusion matrix --------------------
```



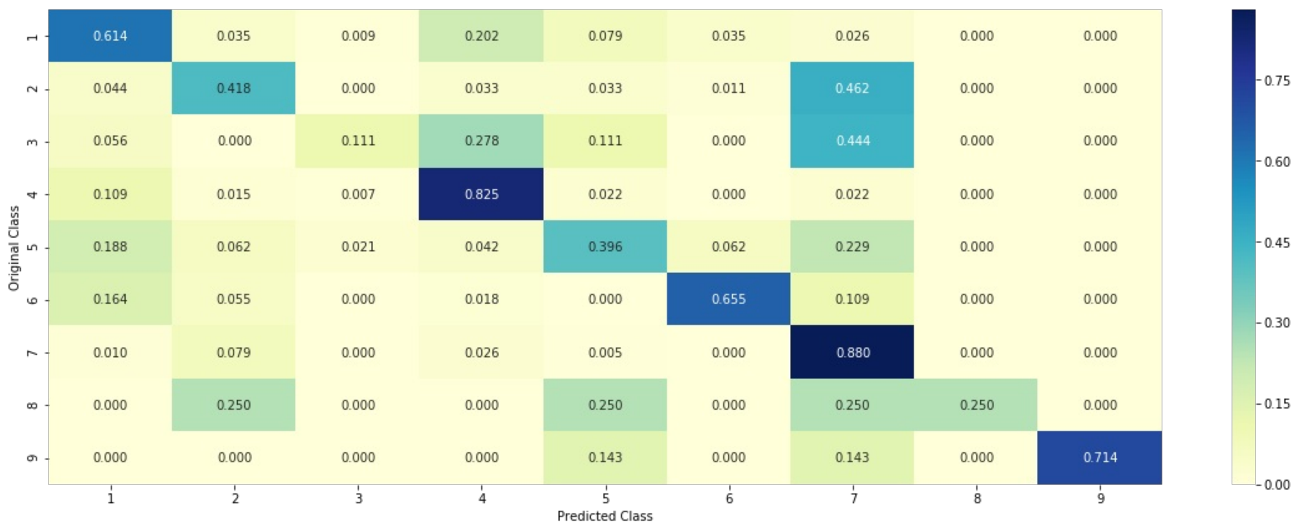
Original \ Predicted	1	2	3	4	5	6	7	8	9
1	70.000	4.000	1.000	23.000	9.000	4.000	3.000	0.000	0.000
2	4.000	38.000	0.000	3.000	3.000	1.000	42.000	0.000	0.000
3	1.000	0.000	2.000	5.000	2.000	0.000	8.000	0.000	0.000
4	15.000	2.000	1.000	113.000	3.000	0.000	3.000	0.000	0.000
5	9.000	3.000	1.000	2.000	19.000	3.000	11.000	0.000	0.000
6	9.000	3.000	0.000	1.000	0.000	36.000	6.000	0.000	0.000
7	2.000	15.000	0.000	5.000	1.000	0.000	168.000	0.000	0.000
8	0.000	1.000	0.000	0.000	1.000	0.000	1.000	1.000	0.000
9	0.000	0.000	0.000	0.000	1.000	0.000	1.000	0.000	5.000

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



Original \ Predicted	1	2	3	4	5	6	7	8	9
1	0.636	0.061	0.200	0.151	0.231	0.091	0.012	0.000	0.000
2	0.036	0.576	0.000	0.020	0.077	0.023	0.173	0.000	0.000
3	0.009	0.000	0.400	0.033	0.051	0.000	0.033	0.000	0.000
4	0.136	0.030	0.200	0.743	0.077	0.000	0.012	0.000	0.000
5	0.082	0.045	0.200	0.013	0.487	0.068	0.045	0.000	0.000
6	0.082	0.045	0.000	0.007	0.000	0.818	0.025	0.000	0.000
7	0.018	0.227	0.000	0.033	0.026	0.000	0.691	0.000	0.000
8	0.000	0.015	0.000	0.000	0.026	0.000	0.004	1.000	0.000
9	0.000	0.000	0.000	0.000	0.026	0.000	0.004	0.000	1.000

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



Original \ Predicted	1	2	3	4	5	6	7	8	9
1	0.614	0.035	0.009	0.202	0.079	0.035	0.026	0.000	0.000
2	0.044	0.418	0.000	0.033	0.033	0.011	0.462	0.000	0.000
3	0.056	0.000	0.111	0.278	0.111	0.000	0.444	0.000	0.000
4	0.109	0.015	0.007	0.825	0.022	0.000	0.022	0.000	0.000
5	0.188	0.062	0.021	0.042	0.396	0.062	0.229	0.000	0.000
6	0.164	0.055	0.000	0.018	0.000	0.655	0.109	0.000	0.000
7	0.010	0.079	0.000	0.026	0.005	0.000	0.880	0.000	0.000
8	0.000	0.250	0.000	0.000	0.250	0.000	0.250	0.250	0.000
9	0.000	0.000	0.000	0.000	0.143	0.000	0.143	0.000	0.714

6.2.4 Feature Importance

6.2.4.1 Incorrectly Classified point

In [78]:

```
test_point_index = 55
no_feature = 500
predicted_cls = sig_clf.predict(test_x_ngrams[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_ngrams[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices11 = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.084  0.0353 0.0097 0.7921 0.0221 0.0241 0.0257 0.0034 0.0036]]
Actual Class : 4
--------------------------------------------------
```

6.2.4.2 Correctly Classified point

In [80]:

```
test_point_index = 1
no_feature = 5000
predicted_cls = sig_clf.predict(test_x_ngrams[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_ngrams[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices12 = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[9.124e-01 2.300e-03 1.700e-03 2.900e-02 3.800e-02 1.350e-02 3.000e-
03
  1.000e-04 1.000e-04]]
Actual Class : 1
--------------------------------------------------
```

What we did so far?

The task for this experiment is to correctly classify a given genetic variations/mutations into one of the nine classes, based on evidences gathered from text-based clinical literature. There was a no latency requirement, which means it is okay to take some time before arriving at a conclusion. Here, we should keep in mind that making an incorrect assumption can be costly, as it will severely affect a patient's line of treatment. We will build a classifier which will give probabilistic outputs instead of just giving us the class labels.

Our dataset for this task consists of three features - Gene, Variants, Text. We will use these three features to build a cancer detection model.

In the EDA sections there are some very useful information about the structure of the given data. We have 3321 data points unequally distributed among 9 classes. The distribution of the number of sentences tells us that there are almost 50% data points which has 427 sentences at max. Almost 75% data points have 744 number of sentences. There's however a sudden increase beyond the 75th percentile mark. We will keep this in mind as we move along.

As a general rule of thumb, the pre-processing is done by removing unwanted words, characters and converting the entire text to lower case. Converting the text to lower case is important because, the model will treat the capital and small letters as different words even if they are the same!

I have some new features. These are, MergedText - Each of the variation data point merges with the the corresponding text data into one single string. TopWordsInText - This feature will give us the top 75 most frequently occurring words in a given text data. We will use these features to get tfidf representations to train our model. In feature engineering part 1, we will use the gene and variations data to build a tfidf-vectorizer. With this tfidf-vectorizer we will encode the text data, and use this as a feature. This is very relevant because, we have already seen that gene and variation itself brought down the log loss to almost half of a random model.

We need to be careful while splitting the data into train, test and cross validation datasets. This is because we want the three datasets to have almost equal distributions of classes. We can draw a simple plot to check these distributions.

Now, it's time for use to see what are the top most frequently occurring words for a given class. I have used word clouds for this purpose. The word clouds can straight away give us relevant keywords which helps us understand what type of words are mots common for every class. For example we see keywords like "breast cancer", "lung cancer", "ovarian cancer", "gene mutation", "tyrosie kinase", "egfr mutation","amino acids" and so on, belonging to each of the classes. Here I have given just a few examples. We also see that there are some classes which talks about some particular variation/mutant type more frequently than others.

Why have we chosen our metric to be 'log loss' ?

As we have discussed above, we will use our Key Performance Indicator to be 'log' loss. Minimising the Log Loss is basically equivalent to maximising the accuracy of the classifier. In order to calculate Log loss, the classifier must actually assign a probability value for each of the class labels. Ideally as the predicted class probabilities improve, the Log loss keeps on reducing. Log loss penalises the classifier very heavily if it classifies a Class 1 to be Class 4 and vice versa. For example, if for a particular observation, the classifier assigns a very small probability to the correct class then the corresponding contribution to the Log Loss will be very large. Naturally this is going to have a significant impact on the overall Log Loss for the classifier,which will then become higher. But, in other scenario if the classifier assigns a high probability value to the correct class, then the Log loss will reduce significantly. Now, imagine a scenario where the model doesn't predict probability scores. It only predicts class labels. In this case, class labels can be either 0 or 1. So we won't get a deep understanding or any interpretability about why a particular pair of question has been labeled as either 0 or 1. Chosing the KPI as Log loss gives us this freedom. We can interpret the models. We can say this two questions are 95% similar or 80% similar, instead of just bluntly classifying them as duplicates.

For deep understanding of log loss please visit: https://datawookie.netlify.com/blog/2015/12/making-sense-of-logarithmic-loss/ (https://datawookie.netlify.com/blog/2015/12/making-sense-of-logarithmic-loss/) (https://datawookie.netlify.com/blog/2015/12/making-sense-of-logarithmic-loss/ (https://datawookie.netlify.com/blog/2015/12/making-sense-of-logarithmic-loss/))

Now, the minimum value of log loss in the ideal scenario would be 0. But what about the maximum? For this we will build a random model. This random model, which is like the worst model that can be made for this problem, gave us a log-loss of almost 2.5. This gives us an upper limit. We now know what is the worst log loss that our model can give? With this in mind we will try to bring the log loss as close to zero as possible.

Typically for high dimensional data, logistic regression and linear SVMs work very well. We can use KNN, Naive Bayes for lower dimensional data representations. In a seperate experiment we can also try models like XGBoost and LightGBM to extract features. Anyway, as the first step we need to encode the text feature into numerical data. How? We have two ways - a simple one hot encoding of the text corpus using Bag of Words approach and a response coding approach. 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.

Very Important note about response coding: Avoiding response leakage.

1. We have to be extremely careful not to use the test and cross validate data for response coding. This is because we don't the issue of data leakage.
2. Suppose we have a variant V2 present in Test / Cross Val dataset. But V2 is not present in Train. So in that case, while building the response bales for V2, we will just assign equal probability values to each of the 9 array values. Proba = 1/9 for each og them.
3. We will take the help of laplace smoothing in order to achieve this. Without laplace smoothing, we would get a 0/0 error.
4. We are seeing the data that is present in the cross validation and test data during the time of training. So we are literally leaking the information that is present in test/cv data at the time of training.

This should be strictly avoided, as we do not want a data leakage issue.

After all this is done, we will perform univariate analysis using all the features. As we can see, all the individual features brought down the log loss by a significant amount. Now we will combine these three features in various ways to get the best possible log loss that we could get.

The other metric we have chosen for this problem is the confusion matrix. By suing the confusion matrix, precision matrix and the recall matrix we can actually see what percentage of points are correctly classified in each of the 9 classes.

We will use various models like naivae bayes, k-nearest neighbors, logistic regression, random forest etc. Before we begin training our models we have to make sure that we build our model such that it is interpretable. Not only the model will tell us the class type, but it will also tell us the exact reason why it thinks a given query point belongs to a certain class.

First we will use TFIDF features and run all the models. Here we see that the best log loss we get is from the logistic regression model with balanced class values. Instead of using all the words in the dataset, we will use only the top 1000 words which occur in the text feature.

Now, since we know that logistic regression performs better for high dimensional data, we will repeat our experiment by encoding the text to BOW representations using bigrams and unigrams. We have seen in the word clouds before how unigrams and bigrams are so important in determining the context of any class. The BOW features with bigrams performed fairly well. But, there wasn't any drastic improvement in the log loss.

With the initial first cut solution, we will move onto the feature engineering stage, where we will perform hyperparameter tuning and try and reduce the log loss as much as possibele.

In stage 1 of feature engineering, we will combine the words present in the Gene column with those present in the Variation column. We will build a tfidf vectorizer using this corpus and use this to transform the text column. We will use this as our 4th feature and see a how a logistic regression performs on this model. Using this feature we were able to reduce the log loss to almost 0.95, which is a significant drop.

In feature engineering stage 2, we will add a 5th feature. We will take the 75 most frequently occurring words in each text and use this corpus to build a tfidf vector representation of the top words. Using this method, I was able to bring down the log loss to 0.92 with a 31% misclassification error. This is by far the best model that we have seen.

Given below are the model performances for all the models tried so far.

Comparing model performances:

TF-IDF Feature representations:

In [19]:

```
from prettytable import PrettyTable
table = PrettyTable()
table.field_names = ["Model", "Vectorizer", "CV Log Loss", "Test Log Loss", "Misclassification Error"]
table.add_row(["Random Model",'Response Coded', "~2.5", "~2.5", ">65%"])
table.add_row(["Naive Bayes",'TFIDF', 0.525, 1.146, "36.48%"])
table.add_row(["KNN",'TFIDF', 0.642, 1.032, "36.27%"])
table.add_row(["LR (Class Balancing)",'TFIDF', 0.442, 0.979, "34.27%"])
table.add_row(["LR (No Class Balancing)",'TFIDF', 0.433, 1.005, "35.15%"])
table.add_row(["Linear-SVM",'TFIDF', 0.504, 1.0420, "35.52%"])
table.add_row(["Random Forest (OneHot)",'TFIDF', 0.852, 1.175, "41.35%"])
table.add_row(["Random Forest (ResponseCode)",'TFIDF', 0.059, 1.257, "41.91%"])
table.add_row(["StackingClassfier",'TFIDF', 0.533, 1.135, "37.29%"])
table.add_row(["MaximumVotingClassfier",'TFIDF', 0.833, 1.173, "37.44%"])
print(table)
```

```
+------------------------------+----------------+-------------+---------------+---------------------
----+
|            Model             |   Vectorizer   | CV Log Loss | Test Log Loss | Misclassification Er
ror |
+------------------------------+----------------+-------------+---------------+---------------------
----+
|         Random Model         | Response Coded |     ~2.5    |      ~2.5     |           >65%
|
|         Naive Bayes          |     TFIDF      |    0.525    |     1.146     |          36.48%
|
|             KNN              |     TFIDF      |    0.642    |     1.032     |          36.27%
|
|     LR (Class Balancing)     |     TFIDF      |    0.442    |     0.979     |          34.27%
|
|   LR (No Class Balancing)    |     TFIDF      |    0.433    |     1.005     |          35.15%
|
|          Linear-SVM          |     TFIDF      |    0.504    |     1.042     |          35.52%
|
|    Random Forest (OneHot)    |     TFIDF      |    0.852    |     1.175     |          41.35%
|
| Random Forest (ResponseCode) |     TFIDF      |    0.059    |     1.257     |          41.91%
|
|      StackingClassfier       |     TFIDF      |    0.533    |     1.135     |          37.29%
|
|    MaximumVotingClassfier    |     TFIDF      |    0.833    |     1.173     |          37.44%
|
+------------------------------+----------------+-------------+---------------+---------------------
----+
```

BOW (bigrams) Feature representations:

In [18]:

```
from prettytable import PrettyTable
table = PrettyTable()
table.field_names = ["Model", "Vectorizer", "CV Log Loss", "Test Log Loss", "Misclassification Error"]
table.add_row(["Random Model",'Response Coded', "~2.5", "~2.5", ">65%"])

table.add_row(["LR (Class Balancing)",'BOW', 0.746, 1.25, "40.78%"])
table.add_row(["LR (No Class Balancing)",'BOW', 0.742, 1.277, "39.47%"])
table.add_row(["Linear-SVM",'BOW', 0.504, 1.0420, "35.52%"])

print(table)
```

```
+-------------------------+----------------+-------------+---------------+-------------------------+
|          Model          |   Vectorizer   | CV Log Loss | Test Log Loss | Misclassification Error |
+-------------------------+----------------+-------------+---------------+-------------------------+
|       Random Model      | Response Coded |     ~2.5    |      ~2.5     |           >65%          |
|   LR (Class Balancing)  |      BOW       |    0.746    |      1.25     |          40.78%         |
| LR (No Class Balancing) |      BOW       |    0.742    |     1.277     |          39.47%         |
|        Linear-SVM       |      BOW       |    0.504    |     1.042     |          35.52%         |
+-------------------------+----------------+-------------+---------------+-------------------------+
```

Feature Engineering Results:

In [17]:

```
from prettytable import PrettyTable
table = PrettyTable()
table.field_names = ["Model", "Vectorizer", "CV Log Loss", "Test Log Loss", "Misclassification Error"]
table.add_row(["Random Model",'Response Coded', "~2.5", "~2.5", ">65%"])

table.add_row(["LR (Class Balancing) (FE1)",'TFIDF', 0.456, 0.970, "33.23%"])
table.add_row(["LR (No Class Balancing) (FE1)",'TFIDF', 0.445, 0.963, "33.24%"])
table.add_row(["LR (Class Balancing) (FE2)",'TFIDF', 0.448, 0.925, "32.48%"])
table.add_row(["LR (No Class Balancing) (FE2)",'TFIDF', 0.439, 0.92, "32.03%"])

print(table)
```

```
+-------------------------------+----------------+-------------+---------------+--------------------
-----+
|             Model             |   Vectorizer   | CV Log Loss | Test Log Loss | Misclassification E
rror |
+-------------------------------+----------------+-------------+---------------+--------------------
-----+
|          Random Model         | Response Coded |     ~2.5    |      ~2.5     |           >65%
|
|   LR (Class Balancing) (FE1)  |     TFIDF      |    0.456    |      0.97     |          33.23%
|
| LR (No Class Balancing) (FE1) |     TFIDF      |    0.445    |     0.963     |          33.24%
|
|   LR (Class Balancing) (FE2)  |     TFIDF      |    0.448    |     0.925     |          32.48%
|
| LR (No Class Balancing) (FE2) |     TFIDF      |    0.439    |      0.92     |          32.03%
|
+-------------------------------+----------------+-------------+---------------+--------------------
-----+
```