Personalized cancer diagnosis

Description

Memorial Sloan Kettering Cancer Center (MSKCC) launched this competition, accepted by the NIPS 2017 Competition Track, because we need your help to take personalized medicine to its full potential.

Once sequenced, a cancer tumor can have thousands of genetic mutations. But the challenge is distinguishing the mutations that contribute to tumor growth (drivers) from the neutral mutations (passengers).

Currently this interpretation of genetic mutations is being done manually. This is a very time-consuming task where a clinical pathologist has to manually review and classify every single genetic mutation based on evidence from text-based clinical literature.

We need to develop a Machine Learning algorithm that, using this knowledge base as a baseline, automatically classifies genetic variations.

Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/
Data: Memorial Sloan Kettering Cancer Center (MSKCC)

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.

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-almostheres-what-it-teaches-us/#2a44ee2f6b25

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

Real-world/Business objectives and constraints.

1. No low-latency requirement. Since we are more focused on the accuracy of the result it can certainly take secs or even mins to make predictions as long as it makes correct predictions.

2. Interpretability is important. We want to know to that on what basis does our model makes a particular prediction so that the doctor can verify the result because wrong decision can lead to life and death of a patient.

3. Errors can be very costly.

4. Probability of a data-point belonging to each class is needed. Why probability is needed because if say we have pobabilities of any two class values very close to each other say 0.4 and 0.6 then the doctor should manually verify the prediction made by the model

Machine Learning Problem Formulation

Data Overview

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

- We have two data files: one conatins the information about the genetic mutations and the other contains the clinical evidence

(text) that human experts/pathologists use to classify the genetic mutations.
- Both these data files are have a common column called ID
- Data file's information:
  - training_variants (ID , Gene, Variations, Class)
  - training_text (ID, Text)

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). ...

Mapping the real-world problem to an ML problem

Type of Machine Learning Problem

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

Performance Metric

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

Metric(s):

- Multi class log-loss (KPI)
- Confusion matrix

What is multi-class log-loss? https://stats.stackexchange.com/questions/113301/multi-class-logarithmic-loss-function-per-class

Train, CV and Test Datasets

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

Exploratory Data Analysis

Importing necessary libraries

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import os
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD #to reduce dimension
#https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html
from sklearn.preprocessing import normalize #to normalize the data
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE #visualization of high dimesional data to 2D
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

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

What is smote? SMOTE stands for Synthetic Minority Oversampling Technique. This is a statistical technique for increasing the number of cases in your dataset in a balanced way. ... SMOTE takes the entire dataset as an input, but it increases the percentage of only the minority cases (source : Google)

Reading the data

Traning Variants

In [3]:

```
os.chdir("D:\\Projects\\Machine-Learning\\Personalized Cancer\\Data Files")
```

In [4]:

```
data1 = pd.read_csv('training_variants')
```

```
part1 = pd.read_csv('training_variants')
print('Number of data points : ',part1.shape[0])
print('Number of features : ', part1.shape[1])
print('Features : ', part1.columns.values)
part1.head()
```

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

Out[4]:

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

In [5]:

```
part1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3321 entries, 0 to 3320
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   ID         3321 non-null   int64 
 1   Gene       3321 non-null   object
 2   Variation  3321 non-null   object
 3   Class      3321 non-null   int64 
dtypes: int64(2), object(2)
memory usage: 103.9+ KB
```

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(target label)

No null values are present

Size of the data is 3321 with 4 columns

Variants Text

In [6]:

```
part2 = pd.read_csv('training_text',sep="\|\|",engine="python",names=["ID","TEXT"],skiprows=1)
print('Number of data points : ',part2.shape[0])
print('Number of features : ', part2.shape[1])
print('Features : ', part2.columns.values)
part2.head()
```

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

Out[6]:

	ID	TEXT
0	0	Cyclin-dependent kinases (CDKs) regulate a var...
1	1	Abstract Background Non-small cell lung cance...

	ID	TEXT
1	1	Abstract Background Non-small cell lung canc...
2	2	Abstract Background Non-small cell lung canc...
3	3	Recent evidence has demonstrated that acquired...
4	4	Oncogenic mutations in the monomeric Casitas B...

In [7]:

```
part2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3321 entries, 0 to 3320
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   ID      3321 non-null   int64 
 1   TEXT    3316 non-null   object
dtypes: int64(1), object(1)
memory usage: 52.0+ KB
```

We have some missing values in the text column

Preprocessing the text

In [8]:

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


def nlp_preprocessing(total_text, index, column):
    """
    This function takes text, index value and the name of the column as input and then does text c
leaning;
    first it removes special characters;
    then replaces multiple spaces into one;
    and converts all the text into lowercase;
    and then finally removes all the stopwords
    """
    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 + " "

        part2[column][index] = string
```

In [9]:

```
#text processing stage.
start_time = time.clock()
for index, row in part2.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 : 28.899775599999998 seconds
```

In [10]:

```
#merging both gene_variations and text data based on ID
final = pd.merge(part1, part2,on='ID', how='left')
final.head()
```

Out[10]:

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

In [11]:

```
final[final.isnull().any(axis=1)] ##printing the values which are null
```

Out[11]:

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

In [12]:

```
final.loc[final['TEXT'].isnull(),'TEXT'] = final['Gene'] +' '+final['Variation']
#gettingg the location where the text column is null and replacing it with the value of gene+varia
tion
```

In [13]:

```
final[final['ID']==2755]
```

Out[13]:

	ID	Gene	Variation	Class	TEXT
2755	2755	BRAF	G596C	7	BRAF G596C

We can see that the index value 2755 was has a nan value in the text column which we replaced it with the concatinating the gene and variation value

Feature Engineering

In [14]:

```
final['n_words'] = final['TEXT'].apply(lambda row: len(row.split(" ")))
```

In [15]:

```
final.head()
```

Out[15]:

	ID	Gene	Variation	Class	TEXT	n_words
0	0	FAM58A	Truncating Mutations	1	cyclin dependent kinases cdks regulate variety...	4371
1	1	CBL	W802*	2	abstract background non small cell lung cancer...	4140
2	2	CBL	Q249E	2	abstract background non small cell lung cancer...	4140
3	3	CBL	N454D	3	recent evidence demonstrated acquired uniparen...	3842
4	4	CBL	L399V	4	oncogenic mutations monomeric casitas b lineag...	4255

In [16]:

```
final.n_words.describe()
```

Out[16]:

```
count     3321.000000
mean      6857.457995
std       5660.037888
min          2.000000
25%       3370.000000
50%       4962.000000
75%       8519.000000
max      56426.000000
Name: n_words, dtype: float64
```

In [17]:

```
final.n_words.quantile(0.99)
```

Out[17]:

```
29567.20000000008
```

around 99% of the words in a text feature is 29.5K or less with minimum number of words in a sentence is 2

Splitting the data into train, cross-validate and test data

In [18]:

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

In [19]:

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

we split the data into train, cross-validate and test and preserve the distribution of the output class

In [20]:

```
print('Number of data points in train data:', train_df.shape[0])
```

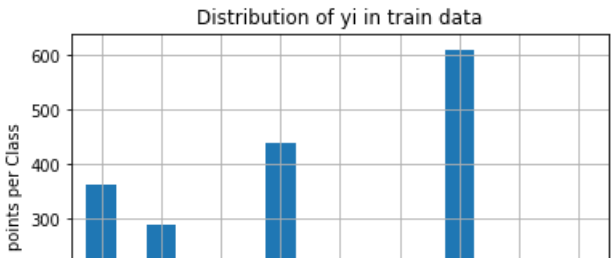
```
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

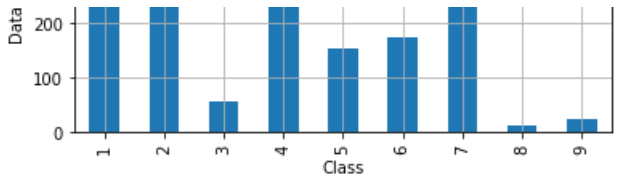
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

Checking the distribution of y's in our train, test and cross-validate data

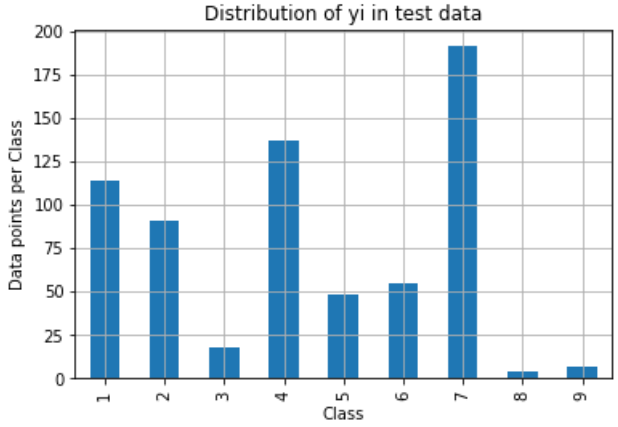
In [21]:

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

my_colors = '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.ro
und((train_class_distribution.values[i]/train_df.shape[0]*100), 3), '%)')


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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.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.rou
nd((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_distribution.values[i]/cv_df.shape[0]*100), 3), '%)')
```

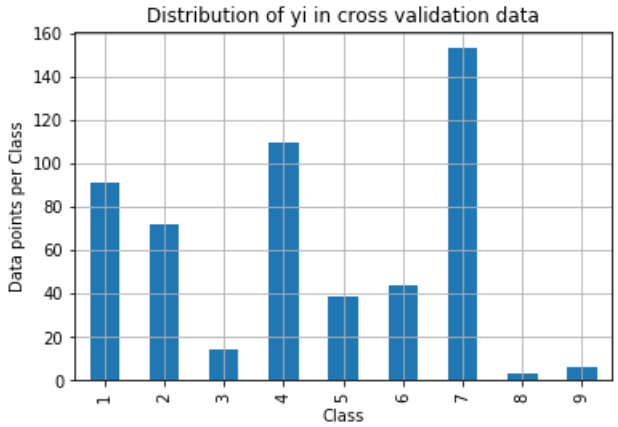





```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298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




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




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

From the above we can observe that :

1. All the train, test and cross-validate data has same distributin
2. class 7, 4 ,1 and 2 are the dominant classes with class 7 has the highest value
3. Imbalanced Data

In [22]:

```
X_train.columns
```

Out[22]:

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

Prediction using Random Model

Why we need Random Model? Random Model acts as a way to compare our model, our model should perform better than our dumb or random model.

How to predict the log-loss of our random model in multi-class problem? We will randomly generate numbers equal to our number of classes(10 in our problem) for every point in our test and CV data and then normalize them to sum it to one.

In [23]:

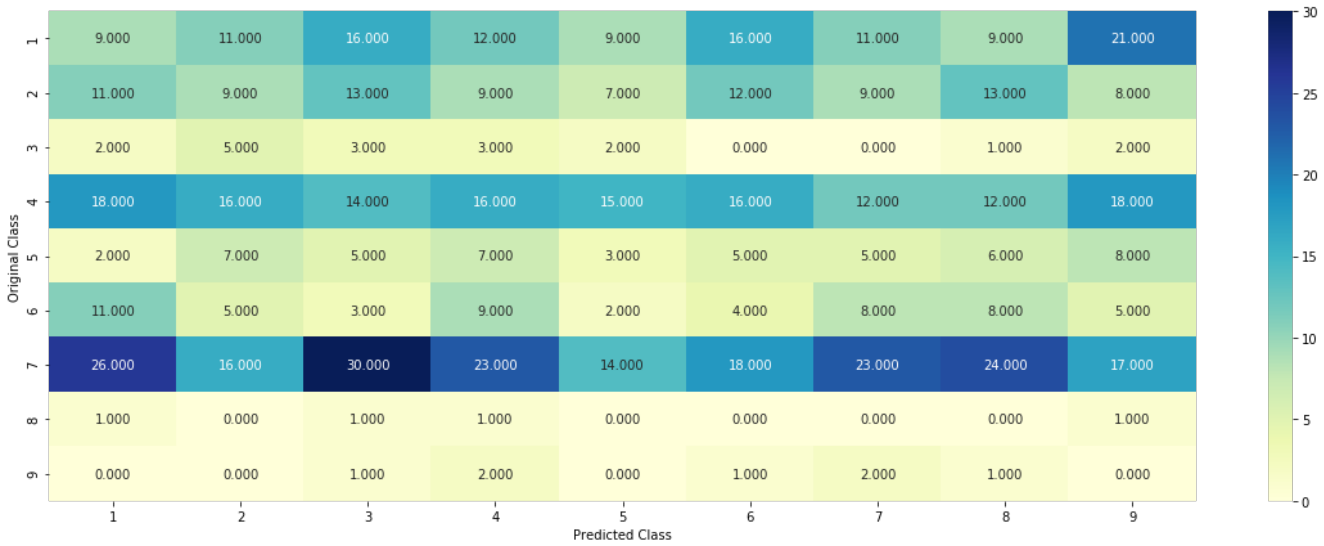
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

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

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

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

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

    labels = [1,2,3,4,5,6,7,8,9]
    # representing 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()
```

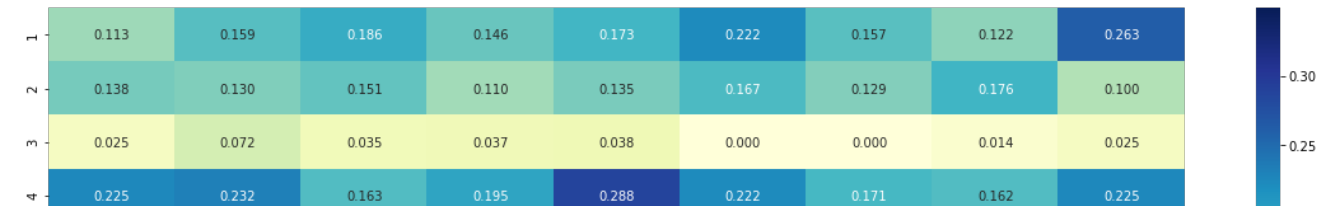
In [24]:

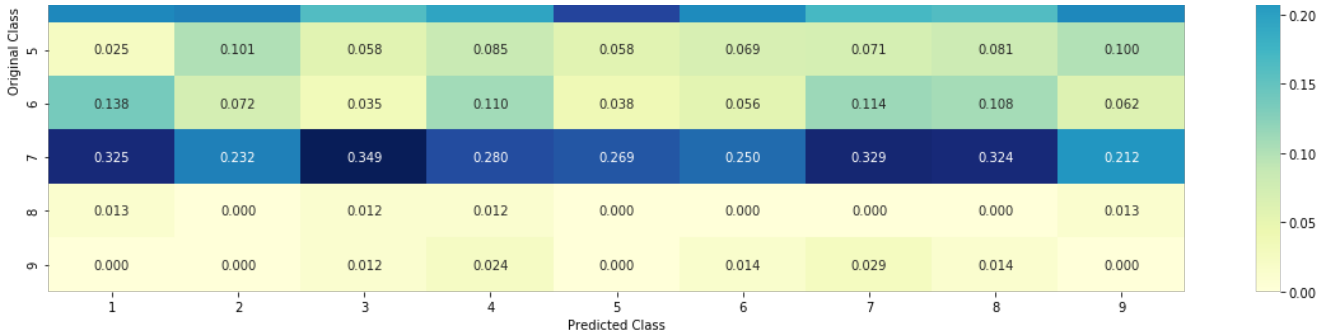
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,9))#for every value in our CV data we create a array of all
zeros with size 9
for i in range(cv_data_len):#iterating to each value in cv data(row)
    rand_probs = np.random.rand(1,9) #generating randoms form 1 to 9
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0]) #normalizing to sum to 1
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 2.4693386381513727
Log loss on Test Data using Random Model 2.468275409770682
-------------------- Confusion matrix --------------------
```

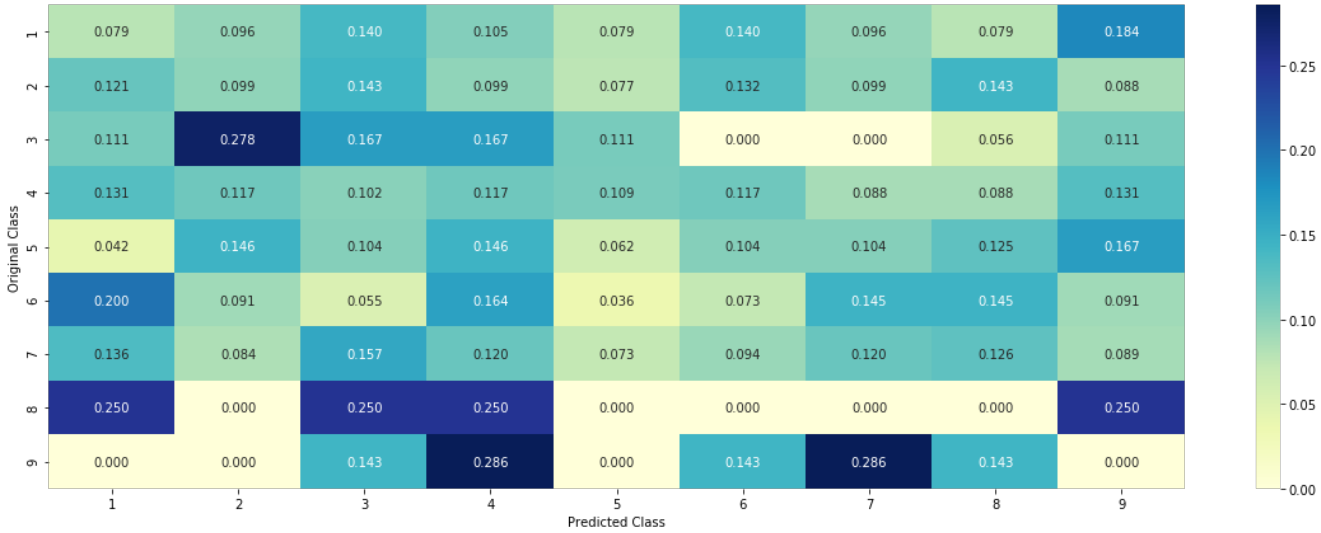



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```






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



We can see that our random model has a log-loss of 2.5 so we need a model that performs better than this model. Ideally we want our model's log-loss to be close to 0

Interpreting Precision And Recall Matrix

Precision

1. Taking an example of cell(1x1) it has value of 0.127 ; it says of all the points that are predicted to be class 1 only 12.7% values are actually class 1
2. for original class4 and predicted class 2 wer can say that of the values that aur model predicted to class 2 23.6% values actually belong to class 4

Recall

1. check cell (1X1) it has a value of 0.079 which means for all the points which actually belongs to class 1 our model predicted only 7% values to be class 1
2. for orginal class 8 and predicted class 5 values is 0.250 means of all the values which are acutally class 8 are model predicted 25% values to be class 5

Ideally we want values in our digonal elements to be high and non-diagonal values to be low.

We want our model to have high precision and high recall

Univariate Analysis

Analysis of each feature and how they contribute in predicting our class labels

In [25]:

```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
```

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

# get_gv_fea_dict: Get Gene varaition Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    # value_count: it contains a dict like
    # print(train_df['Gene'].value_counts())
    # output:
    #        {BRCA1      174
    #         TP53       106
    #         EGFR        86
    #         BRCA2       75
    #         PTEN        69
    #         KIT         61
    #         BRAF        60
    #         ERBB2       47
    #         PDGFRA      46
    #         ...}
    # print(train_df['Variation'].value_counts())
    # output:
    # {
    # Truncating_Mutations                     63
    # Deletion                                 43
    # Amplification                            43
    # Fusions                                  22
    # Overexpression                            3
    # E17K                                      3
    # Q61L                                      3
    # S222D                                     2
    # P130S                                     2
    # ...
    # }
    value_count = train_df[feature].value_counts()
    #value_count gets the features and its couabsnts
    #e.g BRCA1 155
    #    TP53  101

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

    # denominator will contain the number of time that particular feature occured in whole data
    for i, denominator in value_count.items():# i gets feature name ie. BRCA1 , TP53 etc and
denonimator gets value count for that feature

        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to perticular class
        # vec is 9 diamensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=='BRCA1')])
            #          ID   Gene             Variation  Class
            # 2470  2470  BRCA1               S1715C      1
            # 2486  2486  BRCA1               S1841R      1
            # 2614  2614  BRCA1                  M1R      1
            # 2432  2432  BRCA1               L1657P      1
            # 2567  2567  BRCA1               T1685A      1
            # 2583  2583  BRCA1               E1660G      1
            # 2634  2634  BRCA1               W1718L      1
            # cls_cnt.shape[0] will return the number of rows

            cls_cnt = train_df.loc[(train_df['Class']==k) & (train_df[feature]==i)]
            #cls_cnt gets a dataframe where this condition matches

            # cls_cnt.shape[0](numerator) will contain the number of time that particular feature o
ccured in whole data
```

```
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.0378787878787878, 0.068181818181818177,
0.13636363636363635, 0.25, 0.19318181818181818, 0.0378787878787878, 0.0378787878787878,
0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366,
0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408
163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177,
0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608,
0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608,
0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917,
0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081
761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295,
0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702,
0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334,
0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999,
0.066666666666666666, 0.066666666666666666],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

when we caculate the probability of a feature belongs to any particular class, we apply laplace smoothing

- (numerator + 10*alpha) / (denominator + 90*alpha)

Need of laplace Smoothing? While calculating probabilities it could so happen that the probabilities becomes close to zero and to avoid this we will add alpha which is to encounter this kind of problem

Gene

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?

Distribution of gene and number of gene category

In [26]:

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

```
Number of Unique Genes : 231
BRCA1      173
TP53       101
EGFR        81
PTEN        78
BRCA2       76
BRAF        66
KIT         58
ERBB2       44
ALK         42
PDGFRA      38
Name: Gene, dtype: int64
```

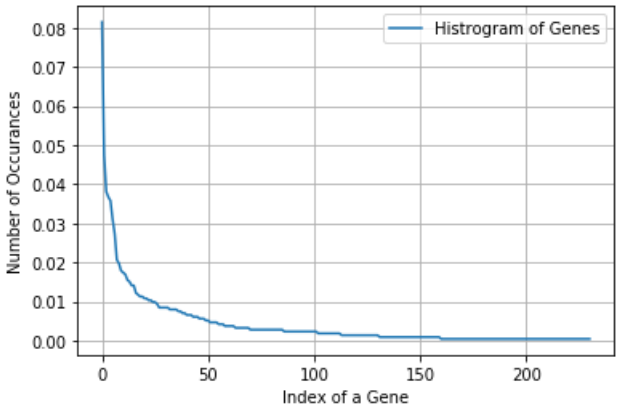
In [27]:

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

```
Ans: There are 231 different categories of genes in the train data, and they are distibuted as abbove
```

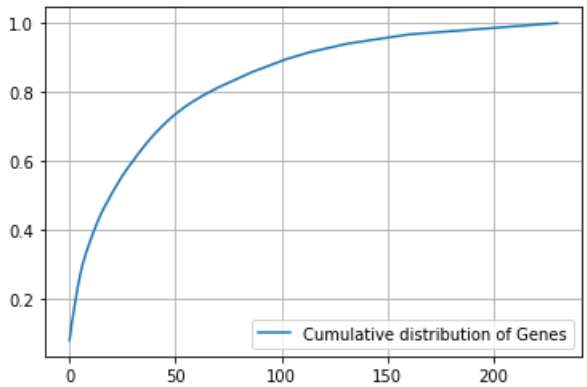
In [28]:

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



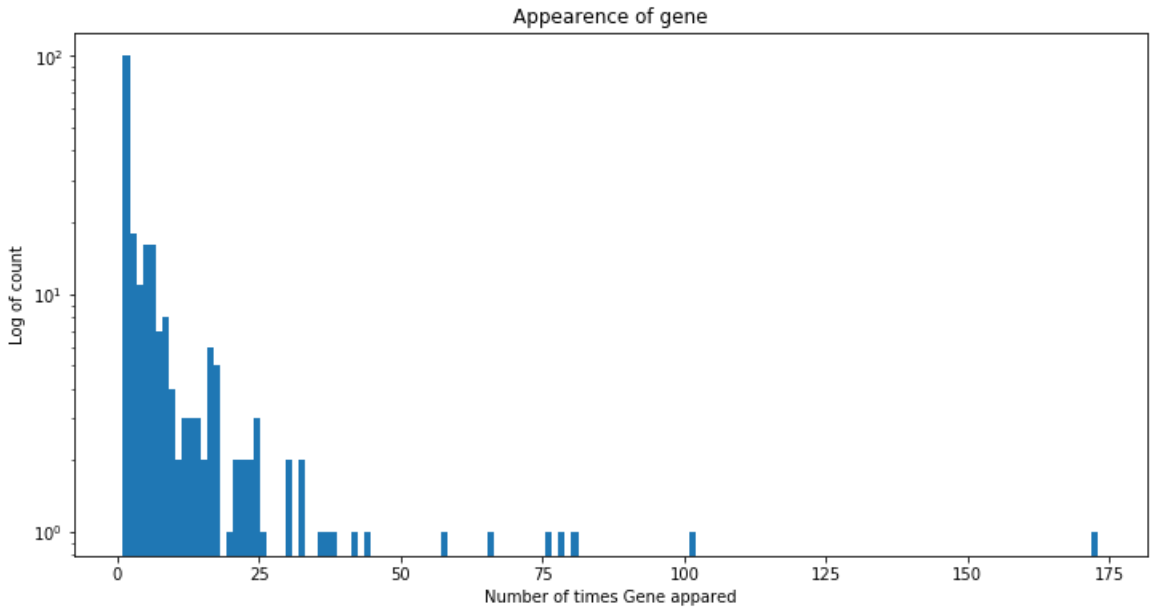
In [29]:

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



In [30]:

```
k = train_df.groupby('Gene')['Gene'].count()

plt.figure(figsize=(12,6))
plt.hist(k, bins=150,log=True)
plt.xlabel('Number of times Gene appared')
plt.ylabel('Log of count')
plt.title('Appearence of gene')
plt.show()
```



In [31]:

```
genecount = Counter(train_df['Gene'])
print('Genes and their appearence:')
print(genecount,'\n',len(genecount))
```

```
Genes and their appearence:
Counter({'BRCA1': 173, 'TP53': 101, 'EGFR': 81, 'PTEN': 78, 'BRCA2': 76, 'BRAF': 66, 'KIT': 58, 'E
RBB2': 44, 'ALK': 42, 'PDGFRA': 38, 'PIK3CA': 37, 'FGFR2': 36, 'FLT3': 33, 'TSC2': 32, 'KRAS': 30,
'CDKN2A': 30, 'MAP2K1': 26, 'MTOR': 25, 'VHL': 24, 'RET': 24, 'FGFR3': 23, 'SMAD4': 23, 'CTNNB1':
22, 'AKT1': 22, 'MLH1': 21, 'MET': 21, 'JAK2': 20, 'CBL': 18, 'NOTCH1': 18, 'MSH2': 18, 'PTPRT': 1
8, 'FBXW7': 18, 'SMAD3': 17, 'NFE2L2': 17, 'PIK3R1': 17, 'ABL1': 17, 'AR': 16, 'PTPN11': 16, 'HRAS
': 15, 'ROS1': 15, 'JAK1': 14, 'RUNX1': 14, 'SMAD2': 14, 'NF1': 13, 'SMO': 13, 'PDGFRB': 13, 'ERBB
4': 12, 'NTRK1': 12, 'TSC1': 12, 'SPOP': 11, 'RHOA': 11, 'ESR1': 10, 'ERCC2': 10, 'PMS2': 10, 'STK
11': 10, 'IDH1': 9, 'CCND1': 9, 'POLE': 9, 'EWSR1': 8, 'KEAP1': 8, 'SF3B1': 8, 'NRAS': 8, 'TET2':
8, 'ETV6': 7, 'AKT2': 7, 'FGFR1': 7, 'PPP2R1A': 7, 'NF2': 7, 'DICER1': 7, 'CARD11': 7, 'EP300': 6,
'ERBB3': 6, 'TGFBR1': 6, 'EPAS1': 6, 'MSH6': 6, 'MAP2K2': 6, 'BRIP1': 6, 'BAP1': 6, 'PIM1': 6,
'MAP2K4': 6, 'RAC1': 6, 'PIK3CB': 6, 'EZH2': 6, 'MYC': 6, 'TERT': 6, 'FANCA': 6, 'AGO2': 5, 'CIC':
5, 'RAF1': 5, 'KDR': 5, 'CDKN2B': 5, 'NKX2-1': 5, 'ELF3': 5, 'CREBBP': 5, 'CDK12': 5, 'B2M': 5, 'D
DR2': 5, 'CDH1': 5, 'ATM': 5, 'RB1': 5, 'STAT3': 5, 'CTCF': 5, 'CASP8': 4, 'PIK3R2': 4, 'SMARCA4':
4, 'SOS1': 4, 'KDM5C': 4, 'APC': 4, 'IDH2': 4, 'MED12': 4, 'CHEK2': 4, 'FOXA1': 4, 'BCOR': 4, 'ARA
F': 3, 'PPP6C': 3, 'PRDM1': 3, 'MEF2B': 3, 'FGFR4': 3, 'CCND3': 3, 'FAT1': 3, 'PTPRD': 3,
'MAP3K1': 3, 'ERG': 3, 'TGFBR2': 3, 'KNSTRN': 3, 'TET1': 3, 'PIK3CD': 3, 'CDK4': 3, 'DNMT3A': 3, '
SOX9': 3, 'ETV1': 3, 'AKT3': 2, 'AURKA': 2, 'PBRM1': 2, 'BCL10': 2, 'KMT2A': 2, 'GNAS': 2, 'MPL':
2, 'GATA3': 2, 'RAD50': 2, 'CARM1': 2, 'U2AF1': 2, 'NSD1': 2, 'HNF1A': 2, 'XPO1': 2, 'TMPRSS2': 2,
'XRCC2': 2, 'FOXL2': 2, 'ARID1B': 2, 'SRC': 2, 'NTRK3': 2, 'NFKBIA': 2, 'KMT2C': 2, 'RHEB': 2, 'ER
CC4': 2, 'CCNE1': 2, 'RAB35': 2, 'YAP1': 2, 'RBM10': 2, 'CDKN1A': 2, 'RARA': 1, 'CDKN2C': 1,
'SHOC2': 1, 'TCF3': 1, 'BTK': 1, 'KMT2B': 1, 'VEGFA': 1, 'SMARCB1': 1, 'CDKN1B': 1, 'RRAS2': 1, 'A
XL': 1, 'ACVR1': 1, 'CTLA4': 1, 'RICTOR': 1, 'NOTCH2': 1, 'KLF4': 1, 'ASXL2': 1, 'FOXP1': 1, 'MDM4
': 1, 'KDM5A': 1, 'STAG2': 1, 'RAD51C': 1, 'IGF1R': 1, 'ERCC3': 1, 'MYD88': 1, 'TCF7L2': 1, 'MGA':
1, 'FGF4': 1, 'ATRX': 1, 'RAD21': 1, 'CCND2': 1, 'PTCH1': 1, 'PIK3R3': 1, 'CDK6': 1, 'JUN': 1, 'FO
XO1': 1, 'WHSC1': 1, 'NCOR1': 1, 'SETD2': 1, 'EPCAM': 1, 'NTRK2': 1, 'DNMT3B': 1, 'IKBKE': 1, 'INP
P4B': 1, 'RAD54L': 1, 'RASA1': 1, 'SDHC': 1, 'SHQ1': 1, 'FANCC': 1, 'CEBPA': 1, 'DUSP4': 1,
'WHSC1L1': 1, 'GNA11': 1, 'HLA-B': 1, 'FLT1': 1, 'TP53BP1': 1, 'MYOD1': 1, 'RNF43': 1, 'PMS1': 1,
'HLA-A': 1, 'RAD51D': 1, 'H3F3A': 1, 'GNAQ': 1, 'LATS2': 1, 'NPM1': 1, 'ERRFI1': 1, 'MYCN': 1,
'FGF19': 1, 'RAD51B': 1, 'RIT1': 1, 'NUP93': 1})
 231
```

We can see that all the 229 genes available top 50 genes contribute to around 75% of the total values so we can conclude that are 25% genes which are rare.

BRCA1 gene is the most common appearing 159 times

Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

Response Encoding

In [32]:

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

In [33]:

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

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

One-Hot Encoding

In [34]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])

test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])

cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [35]:

```
train_df["Gene"].head(10)
```

Out[35]:

```
2650    BRCA1
258      EGFR
442      TP53
3146     KRAS
2307     JAK1
2230     PTEN
2979      KIT
1318     MLH1
1130      MET
1784       AR
```

```
Name: Gene, dtype: object
```

In [36]:

```
gene_vectorizer.get_feature_names()[0:10]
```

Out[36]:

```
['abl1', 'acvr1', 'ago2', 'akt1', 'akt2', 'akt3', 'alk', 'apc', 'ar', 'araf']
```

In [37]:

```
print("train_gene_feature_onehotCoding is converted feature using tfidf encoding method. The shape
of gene feature:", train_gene_feature_onehotCoding.shape)
```

```
train_gene_feature_onehotCoding is converted feature using tfidf encoding method. The shape of
gene feature: (2124, 230)
```

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

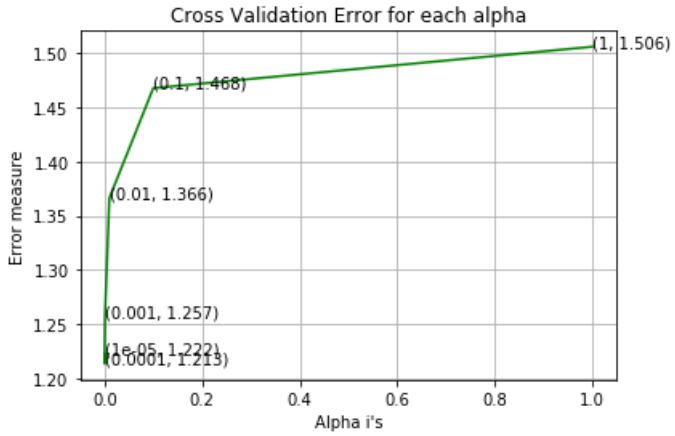
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [38]:

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

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

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

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


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

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


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

```
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_gene_feature_onehotCoding, y_train)

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

```
For values of alpha =  1e-05 The log loss is: 1.2224468428932354
For values of alpha =  0.0001 The log loss is: 1.2133832745665445
For values of alpha =  0.001 The log loss is: 1.256974963563508
For values of alpha =  0.01 The log loss is: 1.3661417709528163
For values of alpha =  0.1 The log loss is: 1.467529677362026
For values of alpha =  1 The log loss is: 1.5056996586267142
```




```
For values of best alpha =  0.0001 The train log loss is: 0.9934141010437093
For values of best alpha =  0.0001 The cross validation log loss is: 1.2133832745665445
For values of best alpha =  0.0001 The test log loss is: 1.1459988560050576
```

We can see that the log-loss our model build only on the feature gives a log loss of around 1.15 across the whole data(train,test and cv) and which is far more better as compared to our random model and hence can conclude that **Gene** features is important in predicting our yi's , and since the train , test and cv loss are pretty much similar we can conclude that our model is not underfitting or overfitting

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.

Stabilitiy means that the distribution of our input variable is pretty much similar between the train,cv and test data.

why stability is important? Stability is important because imagine a condition where the cv and test data is not present in our training data then our model performance will be lowest becuase during training time our model hasn't seen the points and hence nothing to learn about it.

In [39]:

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

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  231  genes in train dataset?
Ans
```

```
1. In test data 641 out of 665 : 96.39097744360903
2. In cross validation data 516 out of  532 : 96.99248120300751
```

Variation

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

In [40]:

```
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(7))
```

```
Number of Unique Variations : 1929
Truncating_Mutations    56
Amplification           50
Deletion                42
Fusions                 22
Overexpression           4
Q61H                     3
G12V                     3
Name: Variation, dtype: int64
```

We can see that the occurence of most of the variations is very low around 1-3 and only 4 variations occured in large numbers like **Truncating_Mutations occur 60 times**

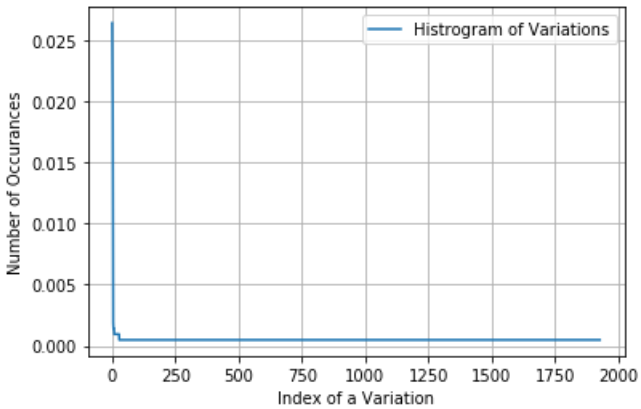
In [41]:

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

```
Ans: There are 1929 different categories of variations in the train data, and they are distibuted
as follows
```

In [42]:

```
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()
```

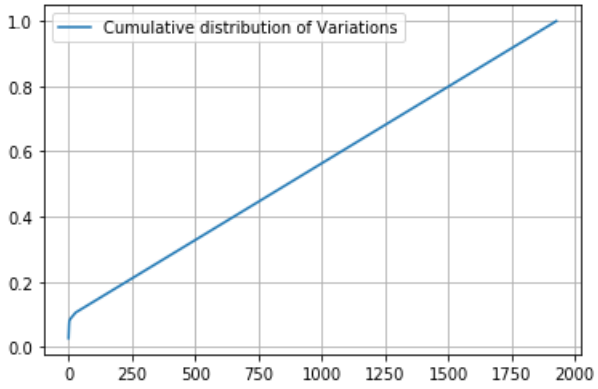


We can see that the graph between number of occurences falls very sharply

In [43]:

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

```
[0.02636535 0.04990584 0.06967985 ... 0.99905838 0.99952919 1.        ]
```



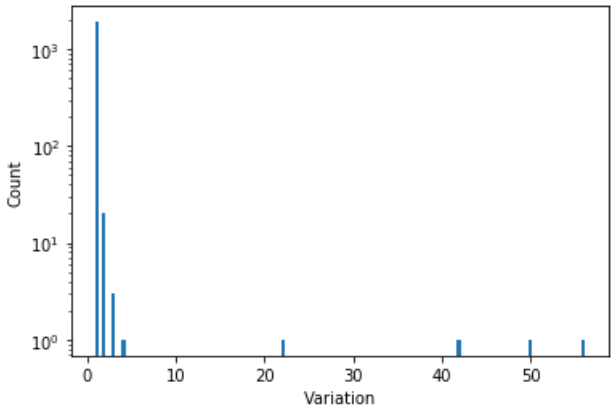
Around 80% of 1927 values lies under 1500 which concludes that most of the points repeats only once or twice 1500/1924 = 0.77

In [44]:

```
k = train_df.groupby('Variation')['Variation'].count()
plt.hist(k, bins=150,log=True)
plt.xlabel('Variation')
plt.ylabel("Count")
plt.figure(figsize=(12,6))
```

Out[44]:

```
<Figure size 864x432 with 0 Axes>
```




```
<Figure size 864x432 with 0 Axes>
```

In [45]:

```
varcount = Counter(train_df['Variation'])
print('variations and their appearence:')
print(varcount,'\n',len(varcount))
```

```
variations and their appearence:
Counter({'Truncating_Mutations': 56, 'Amplification': 50, 'Deletion': 42, 'Fusions': 22, 'Overexpr
ession': 4, 'G12V': 3, 'Q61H': 3, 'Q61R': 3, 'R170W': 2, 'E17K': 2, 'S308A': 2, 'T73I': 2, 'Y42C':
2, 'K117N': 2, 'F384L': 2, 'G13V': 2, 'A146T': 2, 'G12D': 2, 'F28L': 2, 'T167A': 2, 'R173C': 2, 'T
58I': 2, 'M1R': 2, 'P130S': 2, 'Y64A': 2, 'C618R': 2, 'G35R': 2, 'O209L': 2, 'O1756C': 1, 'E330K':
```

1, 'H179L': 1, 'D153V': 1, 'V658F': 1, 'R130L': 1, 'H697Y': 1, 'R217C': 1, 'D1010H': 1, 'P380R': 1
, 'R139G': 1, 'D835N': 1, 'R1627': 1, 'S1025C': 1, 'E1356G': 1, 'V155A': 1, 'G1079D': 1, 'V597A':
1, 'E518A': 1, 'Y537C': 1, 'G776S': 1, 'L838P': 1, 'K382E': 1, 'E501G': 1, 'R487Q': 1, 'V765A': 1,
'R276W': 1, 'Truncating_Mutations_Upstream_of_Transactivation_Domain': 1, 'K147E': 1, 'D761Y': 1,
'C61G': 1, 'S170R': 1, 'R418G': 1, 'N549T': 1, 'L1433S': 1, 'R2450*': 1, 'R158C': 1, 'T529N': 1, '
P463L': 1, 'D119N': 1, 'Q545A': 1, 'T790M': 1, 'MKRN1-BRAF_Fusion': 1, 'P34R': 1, 'S214T': 1, 'A72
7V': 1, 'S860L': 1, 'N841I': 1, 'G161V': 1, 'EWSR1-ETV4_Fusion': 1, 'Q22E': 1, 'G13D': 1, 'R132C':
1, 'L1584R': 1, 'D289_D292del': 1, 'LIMA1-ROS1_Fusion': 1, 'R264C': 1, 'S297F': 1,
'W290_I291delinsC': 1, '550_592del': 1, 'N1878K': 1, 'C275S': 1, '533_534del': 1, 'V560D': 1,
'M1783I': 1, 'Y149D': 1, 'R69C': 1, 'L790F': 1, 'I843del': 1, 'I1170S': 1, 'R415G': 1, 'FGFR1-TACC
1_Fusion': 1, 'A1669S': 1, 'N387P': 1, 'Y234H': 1, 'P33S': 1, 'K935I': 1, 'D60N': 1, 'R201W': 1, '
E1682K': 1, 'K517R': 1, 'E554_K558del': 1, 'E545G': 1, 'R978*': 1, 'L348F': 1, 'R836C': 1,
'G1232D': 1, 'R315*': 1, 'R268A': 1, 'E731K': 1, 'H123D': 1, 'L30F': 1, 'V343L': 1, 'C44Y': 1, 'A2
717S': 1, 'P278A': 1, 'S227F': 1, 'D808N': 1, 'I1616N': 1, 'E77K': 1, 'T37R': 1, 'C124R': 1,
'S33F': 1, 'R71G': 1, 'V851A': 1, 'A391E': 1, 'E1735K': 1, 'K666M': 1, 'R108H': 1, 'D67N': 1, 'F31
I': 1, 'P654L': 1, 'K57E': 1, 'TPM3-NTRK1_Fusion': 1, 'D816N': 1, 'S1297del': 1, 'E579K': 1,
'F71I': 1, 'K428A': 1, 'V648G': 1, 'R121Q': 1, 'E580*': 1, 'L19F': 1, 'P124L': 1, 'G602R': 1, 'E54
2K': 1, 'M136R': 1, 'L118P': 1, 'N1333Gfs*': 1, 'Q546E': 1, 'G356A': 1, 'D835del': 1, 'ETV6-NTRK3_
Fusion': 1, 'R182W': 1, 'V1833M': 1, 'Y599_D600insGLYVDFREYEY': 1, 'PDE4DIP-PDGFRB_Fusion': 1, 'K1
436Q': 1, 'E267G': 1, 'N71I': 1, 'DNA_binding_domain_missense_mutations': 1, 'R776C': 1, 'Q1826H':
1, 'W802*': 1, 'F1088Sfs*2': 1, 'S222D': 1, 'E746_A750del': 1, 'K38N': 1, 'N564D': 1, 'M2676T': 1,
'E40N': 1, 'R167Q': 1, 'V659E': 1, 'K52R': 1, 'S37A': 1, '981_1028splice': 1, 'D594V': 1, 'G334R':
1, 'S2G': 1, 'P1812S': 1, 'L370fs': 1, 'K656E': 1, 'Q809R': 1, 'P1139S': 1, 'D537Y': 1, 'F79S': 1,
'D520N': 1, 'S252W': 1, 'S243C': 1, 'V777M': 1, 'D171N': 1, 'Y591D': 1, 'G106D': 1, 'S33C': 1, 'R2
0Q': 1, 'F81V': 1, 'A148T': 1, 'G127E': 1, 'I2627F': 1, 'Y53H': 1, 'S1498N': 1, 'H355M': 1,
'T468M': 1, 'G665A': 1, 'A349P': 1, 'A126G': 1, 'K650Q': 1, 'L57del': 1, 'Q331R': 1, 'SSBP2-JAK2_F
usion': 1, 'RET-CCDC6_Fusion': 1, 'A23E': 1, 'P655R': 1, 'F212Y': 1, 'Exon_11_mutations': 1,
'T131I': 1, 'Y555C': 1, 'P1776S': 1, 'I1183T': 1, 'P531A': 1, 'L1195V': 1, 'P417A': 1, 'P539R': 1,
'K2411T': 1, 'E746_T751delinsA': 1, 'C105F': 1, 'F958S': 1, 'R130A': 1, 'D65N': 1, 'G13R': 1, 'R30
4*': 1, 'Y253H': 1, 'Q227L': 1, 'E207K': 1, 'P29S': 1, 'G660D': 1, 'M117I': 1, 'K413E': 1, 'E1682V
': 1, 'BCAN-NTRK1_Fusion': 1, 'I836del': 1, 'V197L': 1, 'L348S': 1, 'M18T': 1, 'D835H': 1,
'G423V': 1, 'Q1396R': 1, 'R181C': 1, 'S432L': 1, 'A636P': 1, 'E545Q': 1, 'S501_A502dup': 1,
'E545K': 1, 'G123R': 1, 'R669G': 1, 'V369G': 1, 'ZNF198-FGFR1_Fusion': 1, 'I559_D560insDKRMNS': 1,
'P95L': 1, 'D1778G': 1, 'L1904V': 1, 'N551K': 1, 'A1830T': 1, 'K603Q': 1, 'D108N': 1, 'A648T': 1,
'L424I': 1, 'G914R': 1, 'R373H': 1, 'S215C': 1, 'N56T': 1, 'W515L': 1, 'I290R': 1, 'T1343I': 1, 'E
355A': 1, 'R132H': 1, 'D84V': 1, 'S1463F': 1, 'G1035S': 1, 'N276S': 1, 'KANK1-PDGFRB_Fusion': 1, '
TFG-ROS1_Fusion': 1, 'V118D': 1, 'T485K': 1, 'L861Q': 1, 'V157F': 1, 'N553S': 1, 'T875N': 1, 'A339
V': 1, 'L234fs': 1, 'L1705P': 1, 'Q1756fs': 1, 'A634D': 1, 'P179L': 1, 'V600D': 1, 'G434R': 1, 'P1
637L': 1, 'D1067Y': 1, 'T80A': 1, 'TP53BP1-PDGFRB_Fusion': 1, 'R292A': 1, 'K189N': 1, 'G216R': 1,
'T80K': 1, 'V2006L': 1, 'L1854P': 1, 'R1594Q': 1, 'I834V': 1, 'D1029Y': 1, 'K753M': 1, 'D162H': 1,
'Y646C': 1, 'E69G': 1, 'Y1003*': 1, 'G309A': 1, 'F1592S': 1, 'SND1-BRAF_Fusion': 1, 'X475_splice':
1, 'E812K': 1, 'L726F': 1, 'H597Y': 1, 'C71Y': 1, 'R321Q': 1, 'R544S': 1, 'K179M': 1, 'V2006I': 1,
'P278S': 1, 'K1299E': 1, 'EWSR1-CREB1_Fusion': 1, 'Y599_D600insSTDNEYFYVDFREYEY': 1, 'R342W': 1, '
C242S': 1, 'R38H': 1, 'D842I': 1, 'G248V': 1, 'Exon_19_deletion/insertion': 1, 'R310H': 1,
'D845A': 1, 'E40L': 1, 'N1044K': 1, 'K45T': 1, 'E362H': 1, 'L838V': 1, 'G724S': 1, 'E285K': 1, 'V2
42F': 1, 'K642E': 1, 'L158P': 1, 'M535I': 1, 'T1481fs': 1, 'P42T': 1, 'Q58L': 1, 'P142H': 1,
'D770_N771insNPG': 1, 'D544H': 1, 'V559_V560del': 1, 'R479Q': 1, 'T599_V600insEAT': 1, 'R47Q': 1,
'Y113*': 1, 'S653C': 1, 'C157Y': 1, 'D1352Y': 1, 'A829P': 1, 'D402Y': 1, 'D473G': 1, 'S70fsX93': 1
, 'I251S': 1, 'N45S': 1, 'T582P': 1, 'A750P': 1, 'T195I': 1, 'I111P': 1,
'Promoter_Hypermethylation': 1, 'TRIM24-BRAF_Fusion': 1, 'T37A': 1, 'Y236S': 1, 'Y280H': 1,
'E239A': 1, 'K467T': 1, 'A1020V': 1, 'BCR-PDGFRA_Fusion': 1, 'R625D': 1, 'D2665G': 1,
'L485_Q494del': 1, 'K1026E': 1, 'H1918Y': 1, 'I2500M': 1, 'D473H': 1, 'T1852S': 1, 'R156C': 1, 'V1
55F': 1, 'T77P': 1, 'G382D': 1, 'K590R': 1, 'L82P': 1, 'R479H': 1, 'F1174I': 1, 'C1483Y': 1,
'K376N': 1, 'W557_K558del': 1, 'P96Q': 1, 'P123M': 1, 'D887N': 1, 'D324N': 1, 'L2865V': 1, 'FGFR2-
FAM76A_Fusion': 1, 'D351H': 1, 'K558delinsNP': 1, 'X963_splice': 1, 'R1515H': 1, 'N219D': 1,
'L1198P': 1, 'P179R': 1, 'K129E': 1, 'C1156Y': 1, 'E746_S752delinsI': 1, 'Q60K': 1, 'I408V': 1, 'V
777A': 1, 'A389T': 1, 'R611W': 1, 'T992I': 1, 'G23D': 1, 'N319T': 1, 'R110L': 1, 'R249S': 1, 'CASP
8L': 1, 'M1T': 1, 'V109G': 1, 'W24R': 1, 'C77F': 1, 'D194Y': 1, 'Q72L': 1, 'S217A': 1, 'P44A': 1,
'D1709A': 1, 'L78T': 1, 'A717G': 1, 'Y98H': 1, 'R2520Q': 1, 'R130*': 1, 'R2842H': 1, 'Q96P': 1, 'R
177*': 1, 'E836K': 1, 'N659S': 1, 'G796S': 1, 'R421*': 1, 'Y599_D600insPAPQIMSTSTLISENMNIA': 1, 'S
34F': 1, 'R183P': 1, 'R246K': 1, 'R2784W': 1, 'D572A': 1, 'R265C': 1, 'T1151dup': 1, 'H650Q': 1, '
L67P': 1, 'S1002R': 1, 'I1250T': 1, 'A459V': 1, 'D84G': 1, 'C18Y': 1, 'P336S': 1, 'L23F': 1,
'K78A': 1, 'G505S': 1, 'G751R': 1, 'S1206Y': 1, 'G774V': 1, 'R203C': 1, 'A4419S': 1, 'S425C': 1, '
T401I': 1, 'C482R': 1, 'H61D': 1, 'I111R': 1, 'W1502A': 1, 'F241S': 1, 'EWSR1-ERG_Fusion': 1, 'R11
0P': 1, 'S1841R': 1, 'R172S': 1, 'A1519T': 1, 'L221R': 1, 'Y647C': 1, 'L191H': 1, 'V600M': 1, 'N10
68fs*4': 1, 'D2723H': 1, 'N1647K': 1, 'F156L': 1, 'D2512G': 1, 'D737N': 1, 'V1688del': 1, 'D835A':
1, 'T1977I': 1, 'TRKAIII_Splice_Variant': 1, 'S206C': 1, 'E3002K': 1, 'R11K': 1, 'S1172L': 1,
'C277R': 1, 'Y297A': 1, 'V1714G': 1, 'I195T': 1, 'N71K': 1, 'G1286R': 1, 'C39Y': 1, 'P291Qfs*51':
1, 'L1224F': 1, 'Q201H': 1, 'V804M': 1, 'I347M': 1, 'L485F': 1, 'G1128A': 1, 'L1152P': 1, 'R159G':
1, 'G17E': 1, 'P798L': 1, 'V277D': 1, 'Q58_E62del': 1, 'V842I': 1, 'S1206R': 1, 'E717K': 1,
'P551_E554del': 1, 'H701P': 1, 'A41P': 1, 'S723F': 1, 'Y846C': 1, 'R258H': 1, 'W349C': 1, 'E571K':
1, 'I219V': 1, 'F1761S': 1, 'I26N': 1, 'IGH-MYC_Fusion': 1, 'R206H': 1, 'R1589P': 1, 'S646F': 1, '
AGK-BRAF_Fusion': 1, 'Y1294A': 1, 'R200W': 1, 'G1763V': 1, 'E525K': 1, 'C39R': 1, 'Q395*': 1, 'H21
4N': 1, 'Y472C': 1, 'I157T': 1, 'R1343L': 1, 'D1384V': 1, 'Q324L': 1, 'H61R': 1, 'I42V': 1,
'P838L': 1, 'A763_Y764insFQEA': 1, 'E40Q': 1, 'S729C': 1, 'STRN-ALK_Fusion': 1, 'Y842C': 1,
'D609E': 1, 'T17A': 1, 'I463S': 1, 'T878A': 1, 'S37C': 1, 'G1788D': 1, 'V271L': 1, 'M237K': 1, 'L3
99V': 1, 'P152L': 1, 'S584L': 1, 'G250E': 1, 'L117P': 1, 'S1841N': 1, 'D3170G': 1, 'K335I': 1, 'L5

97Q': 1, 'V769E': 1, 'F1888V': 1, 'S227A': 1, 'T417_D419delinsRG': 1, 'D289del': 1, 'L617F': 1, 'S121A': 1, 'G1194D': 1, 'W1837C': 1, 'Y298A': 1, 'K57T': 1, 'E719G': 1, 'R479L': 1, 'L755S': 1, 'W509R': 1, 'A633V': 1, 'S376F': 1, 'K483M': 1, 'Epigenetic_Silencing': 1, 'H133Q': 1, 'T1720A': 1, 'R1204G': 1, 'G116S': 1, 'E868G': 1, 'N48K': 1, 'R181L': 1, 'D32Y': 1, 'P278L': 1, 'K50L': 1, 'Y532C': 1, 'I326V': 1, 'L461V': 1, 'I122S': 1, 'T244R': 1, 'S384F': 1, 'V1398D': 1, 'A36P': 1, 'E627D': 1, 'R678Q': 1, 'R140Q': 1, 'G469V': 1, '2010_2471trunc': 1, 'D287H': 1, 'F328V': 1, 'V197E': 1, 'L97R': 1, 'L536Q': 1, 'C630R': 1, 'N659K': 1, 'S768I': 1, 'TFG-NTRK1_Fusion': 1, 'Y62C': 1, 'T798M': 1, 'L485_P490delinsY': 1, 'K409Q': 1, 'F367S': 1, 'C49Y': 1, 'A859_L883delinsV': 1, 'L37P': 1, 'L1951R': 1, 'F129L': 1, 'D2870A': 1, 'BCR-ABL1_Fusion': 1, 'P253R': 1, 'S703I': 1, 'K291Q': 1, 'S2483N': 1, 'C134W': 1, 'SEC16A1-NOTCH1_Fusion': 1, 'L611V': 1, 'V471F': 1, 'D595V': 1, 'S566_E571delinsR': 1, 'P81L': 1, 'S35Q': 1, 'T413N': 1, 'G375C': 1, 'K700E': 1, 'D92E': 1, 'D1546N': 1, 'A750_E758delinsP': 1, 'I290A': 1, 'E106G': 1, 'S567L': 1, 'Y553_Q556del': 1, 'V1808A': 1, 'EP300-MOZ_Fusion': 1, 'D842V': 1, 'K745_A750del': 1, 'E82V': 1, 'R882L': 1, 'E127G': 1, 'Y98N': 1, 'R1189*': 1, 'KDELR2-ROS1_Fusion': 1, 'G881D': 1, 'E746V': 1, 'Q579R': 1, 'Y32S': 1, 'H105R': 1, 'I1018W': 1, 'V422del': 1, 'S464L': 1, 'E1799K': 1, 'FGFR2-BICC1_Fusion': 1, 'R838Q': 1, 'H396R': 1, 'R154W': 1, 'S387Y': 1, 'R361C': 1, 'R882C': 1, 'S37F': 1, 'D579del': 1, 'R905G': 1, 'A864T': 1, 'H179N': 1, 'H1620R': 1, 'E142D': 1, 'A530V': 1, 'R331P': 1, 'V1676D': 1, 'V430M': 1, 'F12L': 1, 'R177Pfs*126': 1, 'R465H': 1, 'K830R': 1, 'G508S': 1, 'ATG7-RAF1_Fusion': 1, 'R80C': 1, 'F57V': 1, 'G309E': 1, 'D769Y': 1, 'S45A': 1, 'T844M': 1, 'D1739V': 1, 'R905Q': 1, 'Q984K': 1, 'F522C': 1, 'I35S': 1, 'C554W': 1, 'Q59E': 1, 'K641R': 1, 'N2875K': 1, 'D816A': 1, 'H114Y': 1, 'G67S': 1, 'P1771L': 1, 'R280T': 1, 'Q1503P': 1, 'Y174N': 1, 'Y901C': 1, 'S427G': 1, 'W2626C': 1, 'L861F': 1, 'R170Q': 1, 'G17A': 1, 'E1286V': 1, 'R841Q': 1, 'K291E': 1, 'D390Y': 1, 'TPR-NTRK1_Fusion': 1, 'Y646F': 1, 'H773Y': 1, 'PAPSS1-BRAF_Fusion': 1, 'S362L': 1, 'S56I': 1, 'Q367P': 1, 'D839G': 1, 'C1265S': 1, 'V557I': 1, 'R625C': 1, 'W1038C': 1, 'E946*': 1, 'S46I': 1, 'R1040L': 1, 'D1739E': 1, 'G165V': 1, 'R337L': 1, 'Y87N': 1, 'NSD1-NUP98_Fusion': 1, 'G333S': 1, 'X1008_splice': 1, 'D1692N': 1, 'S241Y': 1, 'S241L': 1, 'E1705A': 1, 'C91A': 1, 'L1019V': 1, 'L597R': 1, 'N549K': 1, 'P83L': 1, 'R342P': 1, 'S502T': 1, 'L535P': 1, 'S1715N': 1, 'D770_N771insD': 1, 'L115R': 1, 'E1794D': 1, 'P480L': 1, 'K78I': 1, 'G106_R108del': 1, 'L210R': 1, 'A72V': 1, 'M199del': 1, 'L283_D294del': 1, 'P531S': 1, 'D171G': 1, 'G81S': 1, 'H41R': 1, 'V561A': 1, 'V794M': 1, 'P214L': 1, 'G1125A': 1, 'R115L': 1, 'Y236C': 1, 'Y537S': 1, 'T1219I': 1, 'M269R': 1, 'V839G': 1, 'Q61K': 1, 'V379I': 1, 'R109*': 1, 'L387M': 1, 'L770V': 1, 'E1552del': 1, 'F691L': 1, 'V344A': 1, 'R552S': 1, 'W1718L': 1, 'R833C': 1, 'S243N': 1, 'K128T': 1, 'L1301R': 1, 'G469del': 1, 'K292I': 1, 'N217I': 1, 'N581D': 1, 'S1733F': 1, 'N676D': 1, 'C176F': 1, 'L158Q': 1, 'H1862L': 1, 'A59G': 1, 'L455M': 1, 'D29H': 1, 'D594Y': 1, 'R282Q': 1, 'E1021K': 1, 'V773A': 1, 'T730S': 1, 'BCOR-CCNB3_Fusion': 1, 'G31R': 1, 'L145R': 1, 'Y849S': 1, 'V559C': 1, 'T599_V600insV': 1, 'F1200I': 1, 'Y371H': 1, 'CD74-ROS1_Fusion': 1, 'E518K': 1, 'A159T': 1, 'P1614S': 1, 'F341V': 1, 'A1131T': 1, 'G85R': 1, 'E255K': 1, 'V559G': 1, 'G67W': 1, 'K11R': 1, 'P428L': 1, 'H214R': 1, 'D603G': 1, 'L272F': 1, 'M774_A775insAYVM': 1, 'H94Y': 1, 'D661Y': 1, 'R164Q': 1, 'P2476L': 1, 'V217D': 1, 'A205T': 1, 'L747S': 1, 'A1789S': 1, 'K342N': 1, 'MYC-nick': 1, 'P316L': 1, 'TRA-NKX2-1_Fusion': 1, 'L1240V': 1, 'S65N': 1, 'C91S': 1, 'T34_A289del': 1, 'V1075F': 1, 'R724H': 1, 'R1204W': 1, 'T417_D419delinsI': 1, 'K62R': 1, 'F1245C': 1, 'L469V': 1, 'F346V': 1, 'G1656D': 1, 'E275K': 1, 'T599R': 1, 'P1856T': 1, 'L345Q': 1, 'F384V': 1, 'S858R': 1, 'R1276Q': 1, 'I111N': 1, 'R1835P': 1, 'S1473P': 1, 'R1896M': 1, 'H68Y': 1, 'A767_V769del': 1, 'I1766S': 1, 'Y285C': 1, 'N546K': 1, 'Y652H': 1, 'E326L': 1, 'Q79R': 1, 'I162M': 1, 'A723D': 1, 'L582F': 1, 'P596L': 1, 'C443Y': 1, 'V60E': 1, 'R481G': 1, 'I1171T': 1, 'A530T': 1, 'MPRIP-NTRK1_Fusion': 1, 'D101Y': 1, 'V191I': 1, 'F2108L': 1, 'E311_K312del': 1, 'R683K': 1, 'M351T': 1, 'M1775E': 1, 'H115N': 1, 'W515K': 1, 'ETV6-PDGFRB_Fusion': 1, 'C1483F': 1, 'P490_Q494del': 1, 'S65W': 1, 'N296I': 1, 'Promoter_Mutations': 1, 'E203K': 1, 'N1730S': 1, 'P648L': 1, 'R383*': 1, 'G271E': 1, 'N653H': 1, 'S1986F': 1, 'T286A': 1, 'Hypermethylation': 1, 'T599I': 1, 'Q50P': 1, 'D603N': 1, 'N334K': 1, 'R2318Q': 1, 'F123I': 1, 'I1018F': 1, 'I15T': 1, 'Y179C': 1, 'E586K': 1, 'T529M': 1, 'Y823D': 1, 'R158L': 1, 'BCR-JAK2_Fusion': 1, 'H284P': 1, 'S170N': 1, 'P261L': 1, 'G623R': 1, 'F1734S': 1, 'R796G': 1, 'K65M': 1, 'L43V': 1, 'T123A': 1, 'M552_K558del': 1, 'D816Y': 1, 'P29L': 1, 'S46N': 1, 'L755P': 1, 'I255F': 1, 'K507A': 1, 'A627T': 1, 'L1198F': 1, 'FGFR2-CCDC6_Fusion': 1, 'Q56P': 1, 'D1818G': 1, 'R82P': 1, 'F133L': 1, 'R100A': 1, 'D326N': 1, 'Y570H': 1, 'A1200V': 1, 'I122V': 1, 'C1697R': 1, 'P250L': 1, 'Q546P': 1, 'V1643A': 1, 'IGH-FGFR3_Fusion': 1, 'V592A': 1, 'M1775K': 1, 'R87L': 1, 'T488_P492del': 1, 'R290H': 1, 'E79Q': 1, 'M117V': 1, 'L52R': 1, 'C135S': 1, 'R897Q': 1, 'T160I': 1, 'K483E': 1, 'D600_L601insFREYEYD': 1, 'M1400V': 1, 'N372H': 1, 'C456_R481del': 1, 'S760A': 1, 'A126V': 1, 'S453fs*': 1, 'T50I': 1, 'D846Y': 1, 'K117R': 1, 'G596V': 1, 'V750E': 1, 'V32G': 1, 'S36Y': 1, 'L57V': 1, 'R420H': 1, 'V1092I': 1, 'V705E': 1, 'C569Y': 1, 'EZR-ROS1_Fusion': 1, 'D325A': 1, 'V710A': 1, 'Q816*': 1, 'A1022E': 1, 'H570R': 1, 'H168R': 1, 'K288Q': 1, 'V384D': 1, 'H538Q': 1, 'N510K': 1, 'P691S': 1, 'Y105C': 1, 'S279Y': 1, 'C528S': 1, 'Q79K': 1, 'Q429*': 1, 'Y1414C': 1, 'T150I': 1, 'V769_D770insGVV': 1, 'L1780P': 1, 'R1758G': 1, 'F351L': 1, 'T1691K': 1, 'L1152R': 1, 'L826P': 1, 'Q50*': 1, 'S768_D770dup': 1, 'P648S': 1, 'N82K': 1, 'L188V': 1, 'S459F': 1, 'D646Y': 1, 'D408H': 1, 'IGH-NKX2_Fusion': 1, 'D717V': 1, 'F1888I': 1, 'F133V': 1, 'G13C': 1, 'L146R': 1, 'S1497A': 1, 'R496H': 1, 'R162*': 1, 'R1231Q': 1, 'A1234T': 1, 'K218T': 1, 'A121P': 1, 'R505C': 1, 'F354L': 1, 'S217C': 1, 'C238F': 1, 'A546D': 1, 'K22A': 1, 'S1715C': 1, 'W603_E604insDREYEYDLKW': 1, 'D92N': 1, 'C456_N468del': 1, 'N655K': 1, 'E60L': 1, 'S1651F': 1, 'T1977K': 1, 'M134L': 1, 'K659N': 1, 'K558_E562del': 1, 'A197T': 1, 'R841K': 1, 'H1047R': 1, 'P47A': 1, 'V1741G': 1, 'T576del': 1, 'H179Y': 1, 'N659R': 1, 'E483*': 1, 'Y1295A': 1, 'L112P': 1, 'F808L': 1, 'S240R': 1, 'L225LI': 1, 'G719C': 1, 'S10N': 1, 'R287A': 1, 'L747_T751delinsP': 1, 'R2505Q': 1, 'L118R': 1, 'E2856A': 1, 'C712R': 1, 'S45F': 1, 'T47D': 1, 'L536R': 1, 'T24A': 1, 'D32A': 1, 'FLT3_internal_tandem_duplications': 1, 'EGFR-KDD': 1, 'D408Y': 1, 'S1486C': 1, 'ATF7IP-JAK2_Fusion': 1, 'L747V': 1, 'C334S': 1, 'S428F': 1, 'G469A': 1, 'D661V': 1, 'R812A': 1, 'CEP110-FGFR1_Fusion': 1, 'R3052W': 1, 'N116H': 1, 'D1270G': 1, 'E709K': 1, 'V774M': 1, 'ETV6-FLT3_Fusion': 1, 'G1128S': 1, 'G81R': 1, 'R80L': 1, 'H36P': 1, 'KIAA1509-PDGFRB_Fusion': 1, 'H191D': 1, 'T507K': 1, 'D1091N': 1, 'V2908G': 1, 'G34V': 1, 'K650E': 1, 'L1122V': 1, 'H1805P': 1, 'K97M': 1, 'E40W': 1, 'V299L': 1, 'M1663K': 1, 'R172K': 1, 'E460L': 1, 'P551_V555del': 1, 'G124S': 1, 'Y35N': 1,

'E1214K': 1, 'P287T': 1, 'E286K': 1, 'W742L': 1, 'S1039F': 1, 'T1365M': 1, 'E40K': 1, 'R117G': 1,
'Y1703S': 1, 'R248L': 1, 'V600K': 1, 'C378R': 1, 'C136Y': 1, 'E1099K': 1, 'G266R': 1, 'L64P': 1, '
L52F': 1, 'T3211K': 1, 'R265S': 1, 'T1977R': 1, 'F119S': 1, 'W257C': 1, 'EWSR1-FLI1_Fusion': 1, 'T
654M': 1, 'G776delinsLC': 1, 'P577S': 1, 'V274F': 1, 'EWSR1-DDIT3_Fusion': 1, 'E258V': 1, 'N564K':
1, 'E218*': 1, 'D1709E': 1, 'E31K': 1, 'R177Q': 1, 'I122L': 1, 'G12A': 1, 'K753A': 1, 'N454D': 1,
'A2351G': 1, 'T599dup': 1, 'K499E': 1, 'S186Y': 1, 'T352M': 1, 'T733I': 1, 'L1273F': 1, 'Y68H': 1,
'D560Y': 1, 'W731L': 1, 'K120M': 1, 'R505L': 1, 'R64P': 1, 'R1209W': 1, 'C381A': 1, 'C1156F': 1, '
F1524V': 1, 'W742C': 1, 'R201H': 1, 'N2113S': 1, 'L833V': 1, 'PTPRZ1-MET_Fusion': 1, 'L232LI': 1,
'H193N': 1, 'D450H': 1, 'S451E': 1, 'C609Y': 1, 'V299G': 1, 'V60M': 1, 'G1803A': 1, 'P114S': 1, 'R
574fs': 1, 'R272C': 1, 'L1600P': 1, 'Q58_Q59insL': 1, 'Q157P': 1, 'S45Y': 1, 'R669C': 1, 'D140G':
1, 'E632_L633del': 1, 'E734Q': 1, 'I538V': 1, 'KIF5B-RET_Fusion': 1, 'D1733G': 1, 'P1806A': 1, 'R2
73L': 1, 'P153H': 1, 'M18K': 1, 'E82G': 1, 'I279P': 1, 'H168N': 1, 'K375A': 1, 'H193P': 1,
'S1140G': 1, 'H123Q': 1, 'V220F': 1, 'R1446H': 1, 'G106V': 1, 'L325F': 1, 'P114L': 1,
"3'_Deletion": 1, 'Exon_19_insertion': 1, 'I744_K745delinsKIPVAI': 1, 'S2215Y': 1, 'V344G': 1,
'BIN2-PDGFRB_Fusion': 1, 'R482Q': 1, 'R368C': 1, 'S1841A': 1, 'R88Q': 1, 'D84Y': 1,
'G778_P780dup': 1, 'E709_T710delinsD': 1, 'V1673D': 1, 'R175L': 1, 'S1651P': 1, 'V411L': 1,
'EWSR1-FEV_Fusion': 1, 'C634R': 1, 'LMNA-NTRK1_Fusion': 1, 'R93Q': 1, 'N1026S': 1, 'D769A': 1, 'N7
1S': 1, 'D2312V': 1, 'V726M': 1, 'V561D': 1, 'T878S': 1, 'Y646S': 1, 'E746_T751insIP': 1, 'Y371S':
1, 'G1123D': 1, 'E466K': 1, 'D1420Y': 1, 'R1076C': 1, 'A1065T': 1, 'H870R': 1, 'L861P': 1,
'W257G': 1, 'L747_A750delinsP': 1, 'W308C': 1, 'G305W': 1, 'V157D': 1, 'L747_A750del': 1,
'H1686R': 1, 'H93D': 1, 'C1767S': 1, 'G31A': 1, 'R698W': 1, 'G2748D': 1, 'A532H': 1, 'E746G': 1, '
S562L': 1, 'S45P': 1, 'L747_P753delinsS': 1, 'W557R': 1, 'K398A': 1, 'V1188L': 1, 'R324L': 1, 'L15
74P': 1, 'S151A': 1, 'IGL-MYC_Fusion': 1, 'G311D': 1, 'S752_I759del': 1, 'I638F': 1, 'FGFR2-
TACC3_Fusion': 1, 'G1770V': 1, 'L785F': 1, 'V658A': 1, 'R1751Q': 1, 'V1605del': 1, 'V248D': 1, 'M1
328I': 1, 'D32H': 1, 'L704N': 1, 'E746_A750delinsQ': 1, 'K1062M': 1, 'F161L': 1, 'R24C': 1,
'I99M': 1, 'D245V': 1, 'R262T': 1, 'P449T': 1, 'D816E': 1, 'R174*': 1, 'G325R': 1, 'V1653M': 1, 'N
581Y': 1, 'Y1045W': 1, 'E78K': 1, 'Q56_V60del': 1, 'D1071N': 1, 'P124S': 1, 'R905W': 1, 'V550E': 1
, 'S59R': 1, 'Y412F': 1, 'Y472H': 1, 'G35A': 1, 'D84N': 1, 'A120S': 1, 'T131S': 1, 'A2425T': 1, 'D
NA_binding_domain_insertions': 1, 'E475K': 1, 'G245A': 1, 'Q1554H': 1, 'N132K': 1, 'G596R': 1, 'R4
41P': 1, 'S1101N': 1, 'Y406H': 1, 'F311L': 1, 'R23A': 1, 'P287A': 1, 'V35M': 1, 'S68W': 1,
'V487_P492delinsA': 1, 'E124Q': 1, 'R1088C': 1, 'T605M': 1, 'E1935G': 1, 'I130M': 1, 'I90T': 1, 'M
980T': 1, 'R133*': 1, 'E5K': 1, 'S904F': 1, 'S462Y': 1, 'W557_V559delinsC': 1, 'F1088Lfs*5': 1, 'R
2973C': 1, 'L861R': 1, 'V271A': 1, 'E81K': 1, 'I18V': 1, 'D92A': 1, 'G466A': 1, 'STRN-
PDGFRA_Fusion': 1, 'A707T': 1, 'W1837R': 1, 'T241P': 1, 'K2472T': 1, 'M1775V': 1, 'E35*': 1,
'I68K': 1, 'N1100Y': 1, 'C277W': 1, 'A614D': 1, 'V705M': 1, 'P262H': 1, 'P753S': 1, 'E157G': 1, 'C
CDC6-ROS1_Fusion': 1, 'F594L': 1, 'R174C': 1, 'H1094L': 1, 'G785S': 1, 'W24C': 1, 'E459K': 1, 'D32
3H': 1, 'N463S': 1, 'C634S': 1, 'I2675V': 1, 'C1483R': 1, 'A246P': 1, 'L1204F': 1, 'L858R': 1, 'V1
73L': 1, 'D631A': 1, 'X434_splice': 1, 'TMPRSS2-ETV1_Fusion': 1, 'Wildtype': 1,
'Y599_D600insEYEYEYEY': 1, 'TEL-JAK2_Fusion': 1, 'A290T': 1, 'K420A': 1, 'N234I': 1, 'H643D': 1, '
E545A': 1, 'L584F': 1, 'N239S': 1, 'L1407P': 1, 'A209T': 1, 'L390F': 1, 'V84L': 1, 'H1047L': 1, 'K
45N': 1, 'Y68D': 1, 'E76A': 1, 'T196A': 1, 'V1180L': 1, 'V600D_K601insFGLAT': 1, 'V319D': 1,
'G44S': 1, 'R1726G': 1, 'R2842C': 1, 'K659E': 1, 'D493A': 1, 'K310R': 1, 'C809G': 1, 'P186S': 1, '
G1706A': 1, 'E321G': 1, 'R1391G': 1, 'G128V': 1, 'Y353L': 1, 'R273H': 1, 'C1483W': 1, 'E255V': 1,
'V294M': 1, 'I168F': 1, 'F568fs': 1, 'C1365Y': 1, 'R282W': 1, 'T3349A': 1, 'ATF7IP-PDGFRB_Fusion':
1, 'L747P': 1, 'L108P': 1, 'A75P': 1, 'L1657P': 1, 'V555_L576del': 1, 'L597V': 1, 'M1652K': 1, 'F5
3C': 1, 'E598_Y599insDVDFREYE': 1, 'I204T': 1, 'R3052Q': 1, 'R172M': 1, 'A298T': 1, 'R1753T': 1, '
G42R': 1, 'C324Y': 1, 'L1593P': 1, 'D32N': 1, 'M552_W557del': 1, 'L1844R': 1, 'H1094R': 1, 'D92H':
1, 'S1303N': 1, 'G207E': 1, 'V1578del': 1, 'L622H': 1, 'C1787S': 1, 'F893L': 1, 'D300N': 1, 'C228T
': 1, 'K513R': 1, 'F170I': 1, 'TMPRSS2-ERG_Fusion': 1, 'L274P': 1, 'N870S': 1, 'Y646H': 1, 'CUX1-F
GFR1_Fusion': 1, 'H1966Y': 1, 'H214Q': 1, 'K1690N': 1, 'W345*': 1, 'H68R': 1, 'Exon_2_mutations':
1, 'G857E': 1, 'M1250T': 1, 'L46F': 1, 'F876L': 1, 'V1838E': 1, 'T1691I': 1, 'R537P': 1, 'R515G':
1, 'N375S': 1, 'EBF1-PDGFRB_Fusion': 1, 'S765P': 1, 'D384N': 1, 'Y253F': 1, 'K125E': 1, 'S330A': 1
, 'D350G': 1, 'Y806C': 1, 'T205A': 1, 'S259A': 1, 'K2950N': 1, 'R1060H': 1, 'N822H': 1, 'L424V': 1
, 'E285V': 1, 'C278F': 1, 'S1424C': 1, 'Q61L': 1, 'P395A': 1, 'V665A': 1, 'Y16C': 1, 'E14*': 1, 'N
486_P490del': 1, 'R625H': 1, 'D254N': 1, 'V1804A': 1, 'H2074N': 1, 'K83E': 1, 'Y791F': 1, 'A59T':
1, 'R267Q': 1, 'V600E': 1, 'G464E': 1, 'G1738R': 1, 'PRKG2-PDGFRB_Fusion': 1, 'C248T': 1, 'L844R':
1, 'Q1811R': 1, 'G863S': 1, 'T2250A': 1, 'T74P': 1, 'R583A': 1, 'V354E': 1, 'R462C': 1, 'D408E': 1
, 'H1047Y': 1, 'EWSR1-ETV1_Fusion': 1, 'E685V': 1, 'V356R': 1, 'L384M': 1, 'A2770T': 1, 'V560E': 1
, 'G31V': 1, 'H845Y': 1, 'R2659T': 1, 'C27A': 1, 'Y35C': 1, 'D513Y': 1, 'N987I': 1, 'R1446C': 1, '
V774A': 1, 'W237_Y242del': 1, 'H412Y': 1, 'C44F': 1, 'A1789T': 1, 'F1761I': 1, 'R957Q': 1,
'385_418del': 1, 'G2274V': 1, 'KIF5B-PDGFRA_Fusion': 1, 'N564_Y578del': 1, 'L536P': 1, 'G1596V': 1
, 'T41A': 1, 'V1833E': 1, 'V1673F': 1, 'E41A': 1, 'G87R': 1, 'G370C': 1, 'Q538P': 1, 'P491S': 1, '
E23fs': 1, 'RANBP1-ALK_Fusion': 1, 'K525E': 1, 'E598_Y599insGLVQVTGSSDNEYFYVDFREYE': 1, 'N676S': 1
, 'Exon_1_mutations': 1, 'G34E': 1, 'H845_N848delinsP': 1, 'E1282V': 1, 'T1720I': 1, 'G423R': 1, '
E49K': 1, 'W131G': 1, 'E1644G': 1, 'ACPP-PIK3CB_Fusion': 1, 'G373R': 1, 'N1102Y': 1, 'R470C': 1, '
V559del': 1, 'H93R': 1, 'T131A': 1, 'D357Y': 1, 'FGFR2-AHCYL1_Fusion': 1, 'MLL-TET1_Fusion': 1, 'M
1K': 1, 'E2014K': 1, 'PCM1-JAK2_Fusion': 1, 'D1399Y': 1, 'Y155C': 1, 'R369W': 1, 'R487*': 1, 'A883
T': 1, 'W398V': 1, 'R337H': 1, 'E563K': 1, 'L2427R': 1, 'N1819Y': 1, 'N535K': 1, 'G1529R': 1, 'RUN
X1-RUNX1T1_Fusion': 1, 'A1066V': 1, 'R248W': 1, 'A1843T': 1, 'YAP1-FAM118B_Fusion': 1, 'M737I': 1,
'N387K': 1, 'G464V': 1, 'G909R': 1, 'H876Q': 1, 'S226D': 1, 'N319D': 1, 'S241T': 1, 'C47S': 1, 'Y4
0A': 1, 'S3660L': 1, 'K1434I': 1, 'S33A': 1, 'T28I': 1, 'T75M': 1, 'H284N': 1, 'K39N': 1,
'E554_V559del': 1, 'A500T': 1, 'F158C': 1, 'W24S': 1, 'N822I': 1, 'S1512I': 1, 'N1236K': 1,
'S215G': 1, 'G469E': 1, 'E172K': 1, 'S1613C': 1, 'R987W': 1, 'G129A': 1, 'D83V': 1, 'S1613G': 1, '
K558N': 1, 'V1070E': 1, 'K59del': 1, 'EP300-MLL_Fusion': 1, 'E279K': 1, 'L28P': 1, 'K111E': 1, 'CI
C-DUX4_Fusion': 1, 'Y375_K455del': 1, 'Y489C': 1, 'L747_P753del': 1, 'P1859R': 1, 'Y3092C': 1,
'V370D': 1, 'D1010Y': 1, 'M1689T': 1, 'H123Y': 1, 'R671Q': 1, 'S1036P': 1, 'V561_I562insER': 1, 'E
xon_20_insertions/deletions': 1, 'R2659K': 1, 'G13E': 1, 'D769H': 1, 'Y27S': 1, 'G831E': 1,

```
'I28T': 1, 'R373Q': 1, 'R251Q': 1, 'E1322*': 1, 'E719K': 1, 'EWSR1-ATF1_Fusion': 1, 'Q1500P': 1, '
D162G': 1, 'R658Q': 1, 'Q1785H': 1, 'S310Y': 1, 'A19V': 1, 'A1701P': 1, 'C384R': 1, 'K82T': 1, 'H2
31R': 1, 'E746_S752delinsA': 1, 'G101W': 1, 'S784F': 1, 'S23R': 1, 'G12C': 1, 'K2729N': 1,
'W1782C': 1, 'E1705K': 1, 'Y1463S': 1, 'K700R': 1, 'Q145H': 1, 'EGFRvII': 1, 'R389*': 1, 'PVT1-MYC
_Fusion': 1, 'T283A': 1, 'K341A': 1, 'L485_P490del': 1, 'R647A': 1, 'P2417A': 1, 'R552G': 1, 'E633
K': 1, 'S1722F': 1, 'Y835F': 1, 'E281K': 1, 'MIR143-NOTCH1_Fusion': 1, 'S196N': 1, 'A18D': 1, 'Y10
03C': 1, 'TGFBR1*6A': 1, 'V272L': 1, 'GIT2-PDGFRB_Fusion': 1, 'A72S': 1, 'R342Q': 1,
'L747_T751del': 1, 'T574insTQLPYD': 1, 'R1391S': 1, 'D2033N': 1, 'NPM-ALK_Fusion': 1, 'L122R': 1,
'R1699W': 1, 'R497H': 1, 'D399N': 1, 'G1269A': 1, 'Y163C': 1, 'Y598C': 1, 'C238S': 1, 'R1751P': 1,
'R1608S': 1, 'M1783L': 1, 'P326L': 1, 'N826Y': 1, 'D1344H': 1, 'Q233*': 1, 'E490K': 1, 'R732Q': 1,
'R1276P': 1, 'R453C': 1, 'D258N': 1, 'K292T': 1, '596_619splice': 1, 'P219S': 1, 'N676K': 1, 'Q531
*': 1, 'V560G': 1, 'L63F': 1, 'C582F': 1, 'W1718C': 1, 'PAX5-JAK2_Fusion': 1, 'ZC3H7B-
BCOR_Fusion': 1, 'R873Q': 1, 'L576P': 1, 'ROS1-CD74_Fusion': 1, 'Y426A': 1, 'R348*': 1, 'R175C': 1
, 'A2643G': 1, 'F248S': 1, 'G101S': 1, 'F958V': 1, 'Q689R': 1, 'L188Q': 1, 'L668F': 1, 'E542V': 1,
'W714*': 1, 'R133H': 1, 'L1764P': 1, 'S1655F': 1, 'G2430A': 1, 'AR-V7': 1, 'S72R': 1, 'CUL1-
BRAF_Fusion': 1, 'T798I': 1, 'E921K': 1, 'L128F': 1, 'P577_D579del': 1, 'T117M': 1, 'K975E': 1, 'L
362R': 1, 'FAM131B-BRAF_Fusion': 1, 'K1452N': 1, 'BCOR-RARA_Fusion': 1, 'P286H': 1, 'A151T': 1, 'E
768D': 1, 'S840_N841insGS': 1, 'H1904R': 1, 'T654I': 1, 'V45L': 1, 'E161del': 1, 'C620Y': 1,
'D3095E': 1, 'L1678P': 1, 'R93W': 1, 'R100*': 1, 'Q337*': 1, 'S267_D273dup': 1, 'V348L': 1,
'G2032R': 1, 'F154L': 1, 'P81T': 1, 'G165R': 1, 'S153R': 1, 'R462E': 1, 'K50E': 1, 'K509I': 1, 'D8
16G': 1, 'T771R': 1, 'K181M': 1, 'S33Y': 1, 'T263P': 1, 'EWSR1-NR4A3_Fusion': 1, 'F468C': 1,
'P44L': 1, 'V104M': 1, 'S1088F': 1, 'S1986Y': 1, 'R15S': 1, 'D1810A': 1, 'H93Q': 1, 'D67Y': 1,
'G127N': 1, 'E40T': 1, 'P1709L': 1, 'M224R': 1, 'Y640F': 1, 'S371C': 1, 'E552K': 1, 'G701S': 1, 'P
1311T': 1, 'W1610G': 1, 'V1534M': 1, 'Q110R': 1, 'T725M': 1, 'Q144R': 1, 'A728V': 1, 'R428A': 1, '
L493V': 1, 'AKAP9-BRAF_Fusion': 1, 'R158H': 1, 'S250P': 1, 'A40E': 1, 'P573_D579del': 1, 'K650M':
1, 'G245D': 1, 'W535L': 1, 'V1804D': 1, 'P1675L': 1, 'K526E': 1, 'I1307K': 1, 'G503V': 1, 'R306S':
1, 'P26S': 1, 'Q79E': 1, 'R48W': 1, 'HMGA2-RAD51B_Fusion': 1, 'D816H': 1, 'G465E': 1, 'V536M': 1,
'V716M': 1, 'V1809F': 1, 'T341P': 1, 'L1196Q': 1, 'R498L': 1, 'A77P': 1, 'D537E': 1, 'M1V': 1, 'N5
40S': 1, 'D1853N': 1, 'M1293A': 1, 'D404G': 1, 'R5Q': 1, 'R201Q': 1, 'A750_E758del': 1, 'L2396F':
1, 'L412F': 1, 'R462I': 1, 'H78Q': 1, 'G244S': 1, 'H875Y': 1,
'L611_E612insCSSDNEYFYVDFREYEYDLKWEFPRENL': 1, 'M504V': 1, 'Exon_13_deletion': 1, 'G1567D': 1, 'C3
9S': 1, 'K666N': 1, 'R689Q': 1, 'N549S': 1, 'P48L': 1, 'R339W': 1, 'P286R': 1, 'M1_E165DEL': 1, 'Q
579_L581del': 1, 'D86N': 1, 'L783F': 1, 'Q249E': 1})
 1929
```

Q9. How to featurize this Variation feature ?

Ans. There are two ways we can featurize this variable

1. One hot Encoding
2. Response coding

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

Response Encoding

In [46]:

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

In [47]:

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

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

One-Hot Encoding

In [48]:

```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])

test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])

cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [49]:

```
print("train_variation_feature_onehotEncoded is converted feature using the one-hot encoding metho
d. The shape of Variation feature:", train_variation_feature_onehotCoding.shape)
```

```
train_variation_feature_onehotEncoded is converted feature using the one-hot encoding method. The
shape of Variation feature: (2124, 1960)
```

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

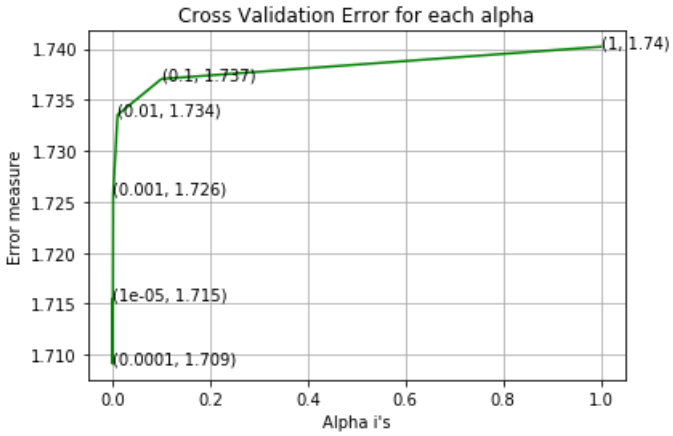
Let's Build a model to know if this feature is useful in predicting yi's or not

In [50]:

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

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

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

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


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

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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.7154726472935515
For values of alpha =  0.0001 The log loss is: 1.7091110696062324
For values of alpha =  0.001 The log loss is: 1.7257752403874869
For values of alpha =  0.01 The log loss is: 1.7335376395157922
For values of alpha =  0.1 The log loss is: 1.7370610953730878
For values of alpha =  1 The log loss is: 1.7402181802338526
```




```
For values of best alpha =  0.0001 The train log loss is: 0.6833367719641565
For values of best alpha =  0.0001 The cross validation log loss is: 1.7091110696062324
For values of best alpha =  0.0001 The test log loss is: 1.6971607589078401
```

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

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

In [51]:

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

```
Q12. How many data points are covered by total  1929  genes in test and cross validation data
sets?
Ans
1. In test data 64 out of 665 : 9.624060150375941
2. In cross validation data 60 out of  532 : 11.278195488721805
```

Reasons which leads to conclusion that this is less stable than gene feature

1. Difference between the train and the cross-validation loss is large
2. Difference between the cross-validation loss of gene and variation
3. we say the frequency of most of the points is 3 or less than three so the probability of points to be present in all the three datapoints is low

Text

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 [52]:

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

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

Response Encoding

In [53]:

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

One-Hot Encoding

In [54]:

```
# building a TfidfVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = TfidfVectorizer(min_df=3,max_features=1000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

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


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

```
Total number of unique words in train data : 1000
```

In [55]:

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

```
    # append it to dict_list

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


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

In [56]:

```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
test_text_feature_responseCoding  = get_text_responsecoding(test_df)
cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)
```

In [57]:

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

In [58]:

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

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

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

In [59]:

```
#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 [60]:

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

```
Counter({250.87610984103: 1, 176.79916020092318: 1, 138.23121675104167: 1, 128.44716794258406: 1,
127.57600675285438: 1, 119.07038664912598: 1, 118.68389597203644: 1, 117.27259370485463: 1,
111.71121031693042: 1, 108.6886670980195: 1, 106.00972935892453: 1, 89.18878148758789: 1,
88.5054703682457: 1, 83.41742782983223: 1, 81.95919937581621: 1, 80.08019013509966: 1,
79.6715600507228: 1, 78.84888888406081: 1, 78.06804936473391: 1, 76.20729477068754: 1,
74.62563127770305: 1, 74.18151235615429: 1, 71.05237836924928: 1, 71.02016259832831: 1,
70.98476137557876: 1, 68.59394331230058: 1, 67.24300356546203: 1, 65.74615454296132: 1,
64.19342263701873: 1, 63.38406723483904: 1, 63.35432208582237: 1, 62.844137974617354: 1,
62.53732526938505: 1, 62.21608803581767: 1, 58.924420304517135: 1, 58.64146278054272: 1,
56.68843980898594: 1, 56.547937465825484: 1, 55.55907680585716: 1, 52.143931412777675: 1,
50.75388700845129: 1, 49.25361637700274: 1, 47.27836897924851: 1, 47.08567222962372: 1,
```

46.95456827351657: 1, 46.1435911539424: 1, 45.97816765461034: 1, 45.74203149615347: 1,
43.9969967808404: 1, 43.95696311499752: 1, 43.75541018787669: 1, 43.7130790433276: 1,
43.305556067545965: 1, 43.184878662749455: 1, 42.932875511835778: 1, 42.698737335817405: 1,
42.02747120777853: 1, 41.62497206962553: 1, 41.468886755526974: 1, 41.30844698522794: 1,
41.18622097418815: 1, 40.25383822744927: 1, 40.24980329088161: 1, 39.93385871810244: 1,
39.85105114461757: 1, 39.232946189739735: 1, 39.18351057894729: 1, 38.79425708255932: 1,
38.77854097203554: 1, 38.75120906697678: 1, 38.28345801638602: 1, 38.19952119054532: 1,
37.829602002592885: 1, 37.79471382144138: 1, 37.749528155512791: 1, 37.7335260821351: 1,
37.69854187886125: 1, 36.46137896676566: 1, 36.08698008638998: 1, 36.04188505392952: 1,
35.9814927432487: 1, 35.29813700323032: 1, 35.19196819626601: 1, 35.11146768392654: 1,
34.76279083613373: 1, 33.27223476581588: 1, 33.266366890545164: 1, 33.03113402102028: 1,
33.00701850704758: 1, 32.91365055106429: 1, 32.7100721587186461: 1, 32.62300048481577: 1,
32.599940182752945: 1, 32.543741168873: 1, 32.5044943020176: 1, 32.22789520452151: 1,
32.21655885507618: 1, 32.20650599906811: 1, 32.03914704157562: 1, 31.988612400912825: 1,
31.987042059775792: 1, 31.88421745792765: 1, 31.591968545398096: 1, 31.474761728359752: 1,
31.45706322136305: 1, 31.37448265784642: 1, 31.337333414343444: 1, 31.001422766111154: 1,
30.91810203016701: 1, 30.87950107836657: 1, 30.77706484622753: 1, 30.39667031041286: 1,
30.379949406162755: 1, 30.356513286931712: 1, 30.167766944668603: 1, 30.106167357198757: 1,
29.95747077768526: 1, 29.68145569574235: 1, 29.622791158898107: 1, 29.606094581503275: 1,
29.492539161477037: 1, 29.48764513153212: 1, 29.40964606018586: 1, 29.35945866166614: 1, 29.3449929
3321899: 1, 29.342032922030057: 1, 29.20409555698134: 1, 29.20252164472429: 1, 28.787407003233913:
1, 28.723842373908447: 1, 28.525583277717526: 1, 28.456525128068783: 1, 28.22687698050581: 1,
28.192600339378906: 1, 28.092999675556317: 1, 27.579824953143028: 1, 27.55988297866666: 1,
27.487671679440812: 1, 27.352662986206383: 1, 27.299690170985645: 1, 27.278708651732681: 1,
26.826278437224637: 1, 26.72713509021585: 1, 26.585949237649622: 1, 26.539106450370195: 1,
26.38100011072623: 1, 26.16676379949066: 1, 26.0263663113758131: 1, 26.023558636086523: 1,
25.746596394883987: 1, 25.71974894747741: 1, 25.486159283390403: 1, 25.441903583723471: 1,
25.388332190534193: 1, 25.365513707101854: 1, 25.33754048578696: 1, 25.124157161467142: 1,
25.046238502324034: 1, 24.98990048854396: 1, 24.948036617791328: 1, 24.682533996652797: 1,
24.662985225569077: 1, 24.65146489775564: 1, 24.50820660614593: 1, 24.488198853794575: 1,
24.323882921778345: 1, 24.31071712923076: 1, 24.305265338522936: 1, 24.270080761178191: 1,
24.156530575055367: 1, 24.139526517317616: 1, 24.05242925080048: 1, 23.997320952361431: 1,
23.7705490181312: 1, 23.69557006118024: 1, 23.682693462996216: 1, 23.591082076652803: 1,
23.455404728161255: 1, 23.422835790493075: 1, 23.413164024903928: 1, 23.407800636566911: 1,
23.369421998309946: 1, 23.338533335700486: 1, 23.261023889711247: 1, 23.189613903216532: 1,
23.141155096836723: 1, 23.118681508217417: 1, 22.991613991049203: 1, 22.910576443278728: 1,
22.674751273782196: 1, 22.622100945391892: 1, 22.498763867613782: 1, 22.461945563538627: 1,
22.445258700350717: 1, 22.35715925230384: 1, 22.323544108429235: 1, 22.271560840579046: 1,
22.269606286540846: 1, 22.235777552115195: 1, 22.23323707993404: 1, 22.191466150470061: 1,
22.183387918595404: 1, 22.15283680193238: 1, 22.137840028990073: 1, 22.081589442557213: 1,
22.07471083891926: 1, 22.040922527878386: 1, 21.871655072356102: 1, 21.845540550248405: 1,
21.81369876996887: 1, 21.808206813409353: 1, 21.765905885046685: 1, 21.754171463661393: 1,
21.749317164172144: 1, 21.617671892080796: 1, 21.58133542953314: 1, 21.57450297864646: 1,
21.56221654257337: 1, 21.445813605241373: 1, 21.423980865082918: 1, 21.37115087844689: 1,
21.269946044978028: 1, 21.165670955273956: 1, 21.155190325536232: 1, 21.148694668013846: 1,
21.113224681772984: 1, 21.076117175182798: 1, 21.021540582347072: 1, 21.018606564864683: 1,
20.99069930043286: 1, 20.90080129227887: 1, 20.834521800029915: 1, 20.775577601514623: 1,
20.619525857828936: 1, 20.466520201818412: 1, 20.459889422655603: 1, 20.411050146915528: 1,
20.257655472075687: 1, 20.251987331550236: 1, 20.210365608252207: 1, 20.202189017411932: 1,
20.162602013331437: 1, 20.118458819850805: 1, 20.10039335391167: 1, 20.03727693373766: 1,
19.996687000653743: 1, 19.952486036962554: 1, 19.936942268531105: 1, 19.881545859368682: 1,
19.84426797604806: 1, 19.756682583806093: 1, 19.74749417589969: 1, 19.63523261658379: 1,
19.62104556427333: 1, 19.608380798191061: 1, 19.604678957689661: 1, 19.59866449472121: 1,
19.560326619023044: 1, 19.527361956191413: 1, 19.43492828703609: 1, 19.41637000420824: 1,
19.341724341862957: 1, 19.25878858280346: 1, 19.18636257689196: 1, 19.173974760199425: 1,
19.16797877081794: 1, 19.166555690994812: 1, 19.153979912511698: 1, 19.15153901637692: 1,
19.145398854362526: 1, 19.139005965039555: 1, 19.138872021023015: 1, 19.133214974349563: 1,
19.05944949776401: 1, 19.023118381963073: 1, 18.95224569814608: 1, 18.896308619136331: 1,
18.8882758595999: 1, 18.844418356326074: 1, 18.839563575336: 1, 18.835085492291384: 1,
18.72763917241042: 1, 18.697773140686508: 1, 18.66818766081207: 1, 18.601965367983073: 1,
18.601322068638133: 1, 18.59613387404979: 1, 18.57959143890509: 1, 18.5748153766061: 1,
18.556611400450283: 1, 18.54917036046181: 1, 18.481004924520274: 1, 18.45347086999955: 1,
18.429089715732683: 1, 18.322319551906126: 1, 18.31516822162384: 1, 18.241537015538736: 1,
18.239273663829916: 1, 18.232089912294118: 1, 18.197774281953055: 1, 18.183685866155091: 1,
18.15345558856611: 1, 18.060429663399756: 1, 18.05578740138072: 1, 17.970983369997505: 1,
17.875837823663456: 1, 17.835516764896: 1, 17.80207086904126: 1, 17.744106765362613: 1,
17.71918567052438: 1, 17.710146086104164: 1, 17.654718771088316: 1, 17.615377728609175: 1,
17.553706866579464: 1, 17.55069953834071: 1, 17.546782469125507: 1, 17.542454965007703: 1,
17.53680226415054: 1, 17.52808453411736: 1, 17.505525244247035: 1, 17.460222897070963: 1,
17.444128139675524: 1, 17.38977196315548: 1, 17.381091311045481: 1, 17.357538490474916: 1,
17.33205599848136: 1, 17.324769319626135: 1, 17.311864009054222: 1, 17.267642787741078: 1,
17.266505808715202: 1, 17.23413501114851: 1, 17.223383250339204: 1, 17.215842388160791: 1,
17.185447367776515: 1, 17.179205620220873: 1, 17.089229691952912: 1, 17.071463222625294: 1,
17.051043421809027: 1, 17.016951364276761: 1, 16.983794941228506: 1, 16.97886228963191: 1,
16.942941300595255: 1, 16.935976832963533: 1, 16.931900740012324: 1, 16.929384088586612: 1,
16.922922207963992: 1, 16.906014312839538: 1, 16.902185752121866: 1, 16.852438074248451: 1,
16.850952432964938: 1, 16.803977245759842: 1, 16.802498114163438: 1, 16.741876448591473: 1,

16.723015212104414: 1, 16.6685970378307: 1, 16.668392177753756: 1, 16.660407084336775: 1,
16.644571388026165: 1, 16.608502256420596: 1, 16.540815739109945: 1, 16.529514033658394: 1,
16.520020946591696: 1, 16.513608405792778: 1, 16.5037113528122: 1, 16.475531198529517: 1,
16.39087034549788: 1, 16.366429026859347: 1, 16.353746539334416: 1, 16.35330886796995: 1,
16.338730619792994: 1, 16.270004161607567: 1, 16.239877083863412: 1, 16.130575031840767: 1,
16.122510230454843: 1, 16.10360794856755: 1, 16.03621865745505: 1, 16.028224544614l8: 1, 15.9597429
63443777: 1, 15.916827846041508: 1, 15.837750138276558: 1, 15.784040345150913: 1,
15.763086440617553: 1, 15.748880959810103: 1, 15.746075188627165: 1, 15.741045461194526: 1,
15.7092678167091: 1, 15.6671515987642: 1, 15.645051909741293: 1, 15.582488003382075: 1,
15.552902696132817: 1, 15.483792909267185: 1, 15.474689665333786: 1, 15.432667510756705: 1,
15.428306127288787: 1, 15.420382563294497: 1, 15.409407583801691: 1, 15.374919274595001: 1,
15.269619526077935: 1, 15.261819726971213: 1, 15.25477883627724: 1, 15.232783189648673: 1,
15.232530985649277: 1, 15.185490567752538: 1, 15.181337731652837: 1, 15.141536684132454: 1,
15.114529352971688: 1, 15.104891277070056: 1, 15.099070920501054: 1, 15.0931814276466: 1,
15.077733668556668: 1, 15.074656238215896: 1, 15.065268749945615: 1, 15.028812639708573: 1,
15.023542742118195: 1, 15.00506699955596: 1, 14.950793625062783: 1, 14.9294684923356: 1,
14.898732491820347: 1, 14.85964502429919: 1, 14.836066318965177: 1, 14.835085849332014: 1,
14.81279234901688: 1, 14.779650423513681: 1, 14.771683236590603: 1, 14.760006901754615: 1,
14.718777148029076: 1, 14.716730131613978: 1, 14.699156998796395: 1, 14.680905571951815: 1,
14.672132134197033: 1, 14.662080951390l8: 1, 14.646250080518865: 1, 14.609971255646546: 1,
14.591592633331068: 1, 14.588000557691l2: 1, 14.585981884129l7: 1, 14.571414948423724: 1,
14.548644472107112: 1, 14.463172939116545: 1, 14.443813151541345: 1, 14.421491862964935: 1,
14.355412143637823: 1, 14.340712238778675: 1, 14.336816367766767: 1, 14.325277408240927: 1,
14.301407216854244: 1, 14.276832139739518: 1, 14.272055948070777: 1, 14.269554098973247: 1,
14.249906762176378: 1, 14.23698136235033: 1, 14.22873033076676: 1, 14.210412304848091: 1,
14.169155474927706: 1, 14.167695663677103: 1, 14.166215162514l4: 1, 14.109079631148l8: 1,
14.108891620962519: 1, 14.045553541345273: 1, 14.015916493869707: 1, 14.004289450634957: 1,
13.939206018623874: 1, 13.893552415332817: 1, 13.883820900908681: 1, 13.875116739406351: 1,
13.866636191051445: 1, 13.840552228882913: 1, 13.829035380897508: 1, 13.815461831325281: 1,
13.807293586495517: 1, 13.785879990814015: 1, 13.687386027798253: 1, 13.68508028137324: 1,
13.674163598276438: 1, 13.622082484252708: 1, 13.617262597170544: 1, 13.60078071384332: 1,
13.580988823083379: 1, 13.55964172985385: 1, 13.549178053050007: 1, 13.540934545001l6: 1,
13.504767983116638: 1, 13.49806525842861: 1, 13.443939331119452: 1, 13.403262152562808: 1,
13.38346266151596: 1, 13.382389458140239: 1, 13.349047733422786: 1, 13.348745434022767: 1,
13.317485013761035: 1, 13.309676395282292: 1, 13.305338455600316: 1, 13.303311521153004: 1,
13.289021238717888: 1, 13.275796569307767: 1, 13.271878114758223: 1, 13.260216968656984: 1,
13.241841196369647: 1, 13.224889382137322: 1, 13.187262197347268: 1, 13.181399819009l2: 1,
13.178402486068416: 1, 13.177395152997052: 1, 13.139033346362908: 1, 13.116395645943257: 1,
12.98826935452568: 1, 12.982816606912724: 1, 12.946095695729033: 1, 12.930373534376686: 1,
12.924134517042388: 1, 12.909749674343542: 1, 12.874325868039296: 1, 12.872015228834l4: 1,
12.857823963020097: 1, 12.852918141146944: 1, 12.82995565046224: 1, 12.784008394700185: 1,
12.761479247942615: 1, 12.737626587198493: 1, 12.721017145395344: 1, 12.683119017932247: 1,
12.669449261863068: 1, 12.66777598306787: 1, 12.66643100247661: 1, 12.661341634334937: 1,
12.659303604910l8: 1, 12.636488492945l: 1, 12.613177498557l: 1, 12.606880867068l8: 1,
12.603225634131196: 1, 12.57874739681599: 1, 12.56005815381367: 1, 12.556599060253584: 1,
12.549439820587555: 1, 12.530493027091977: 1, 12.509142614356456: 1, 12.504411382373782: 1,
12.494607304476077: 1, 12.4922676166974: 1, 12.472050726017175: 1, 12.4174270755324l6: 1,
12.405585755233l4: 1, 12.399839773849095: 1, 12.387713740339045: 1, 12.370356938154666: 1,
12.363105320736514: 1, 12.329738136885362: 1, 12.32664513193521: 1, 12.310710905617533: 1,
12.298415748472813: 1, 12.296889237588266: 1, 12.291879068168726: 1, 12.257553899991622: 1,
12.25316550943308: 1, 12.23240138310893: 1, 12.216005785175023: 1, 12.213161045528526: 1,
12.191565189310035: 1, 12.140254675186528: 1, 12.117688151395866: 1, 12.11226504722686: 1,
12.111928057810767: 1, 12.111924937164437: 1, 12.0938165507076: 1, 12.068935990200817: 1,
12.05178682129301: 1, 12.030753830520045: 1, 12.025071964791932: 1, 11.970564733813783: 1,
11.962241908114228: 1, 11.960195132065923: 1, 11.954819974108895: 1, 11.930443281473698: 1,
11.925534466048038: 1, 11.909778473145188: 1, 11.891645476055624: 1, 11.87168365714133: 1,
11.85751189157313: 1, 11.807851308762l9: 1, 11.794141228065l7: 1, 11.777874893983338: 1,
11.769433349235l5: 1, 11.751120211545528: 1, 11.727839986373311: 1, 11.719704841190953: 1,
11.701344720811486: 1, 11.701041115715663: 1, 11.673386591859314: 1, 11.648675562871453: 1,
11.590388452602l7: 1, 11.589248762437l8: 1, 11.568519550811146: 1, 11.557138569360722: 1,
11.537329182255693: 1, 11.529137546465l3: 1, 11.528702378861619: 1, 11.524354287351882: 1,
11.518884877727l5: 1, 11.510701329926674: 1, 11.505885616668138: 1, 11.503634376741415: 1,
11.487339552960l8: 1, 11.448346732802024: 1, 11.442890066747l5: 1, 11.442461765801l8: 1,
11.428360015689362: 1, 11.411582314282924: 1, 11.397255331168989: 1, 11.377535107723l8: 1,
11.374029131638943: 1, 11.365700157810565: 1, 11.351989788766941: 1, 11.344340655821362: 1,
11.338093348458592: 1, 11.333967873456851: 1, 11.307676693694837: 1, 11.289545279744992: 1,
11.232957183766224: 1, 11.216298722088927: 1, 11.210095885852022: 1, 11.209225978243975: 1,
11.173678180673392: 1, 11.161815342360716: 1, 11.136805419770138: 1, 11.132889976074582: 1,
11.126649714919639: 1, 11.11575430747396: 1, 11.110926750265468: 1, 11.089288902333092: 1,
11.068482721697093: 1, 11.052303539208912: 1, 11.026336230227573: 1, 11.021404488286151: 1,
11.013987564360155: 1, 11.003005333429263: 1, 10.972332780821732: 1, 10.9663109038461: 1,
10.962303299571966: 1, 10.953982800412149: 1, 10.95392299432133: 1, 10.937730237589035: 1,
10.931042109939629: 1, 10.923785552068683: 1, 10.911352057615295: 1, 10.895795814701113: 1,
10.886754339886206: 1, 10.876825976269204: 1, 10.871228268949535: 1, 10.855574087520898: 1,
10.853453573315658: 1, 10.836982976775127: 1, 10.826049913017107: 1, 10.822454057737223: 1,
10.800022589926035: 1, 10.783900106552606: 1, 10.767619676814943: 1, 10.766421411437443: 1,
10.747269031002599: 1, 10.734881725051537: 1, 10.732820434198265: 1, 10.708793421238664: 1,

10.706586388695962: 1, 10.705065643886122: 1, 10.703099283203652: 1, 10.686881827780036: 1,
10.68636571437633: 1, 10.653210753811486: 1, 10.65064666047839: 1, 10.636637634977461: 1,
10.62906460238498: 1, 10.61122249863733: 1, 10.587453962703187: 1, 10.584604673346602: 1,
10.547498653275891: 1, 10.547238544518661: 1, 10.546273071316344: 1, 10.536029571972126: 1,
10.526368914179349: 1, 10.524611801186417: 1, 10.515667157460497: 1, 10.508305527639363: 1,
10.501823178252891: 1, 10.4767442537589: 1, 10.44598995580925: 1, 10.44355052210047: 1,
10.443531907617265: 1, 10.435547168779058: 1, 10.428787044745022: 1, 10.407929540285359: 1,
10.4022741684026: 1, 10.38726933840075: 1, 10.372965532268058: 1, 10.344877042558084: 1,
10.344705491295988: 1, 10.335168746274796: 1, 10.322447476339322: 1, 10.322145327018248: 1,
10.288796301442087: 1, 10.280801288873398: 1, 10.273904349794691: 1, 10.25186212575542: 1,
10.24690972022559: 1, 10.232405303139581: 1, 10.232302509675778: 1, 10.201826048541584: 1,
10.172853324242181: 1, 10.171512282640911: 1, 10.15269472931592: 1, 10.14528018892036: 1,
10.143041495849284: 1, 10.141930128595094: 1, 10.09568237946255: 1, 10.075994394738327: 1,
10.067931961537244: 1, 10.066229025830184: 1, 10.057578044815266: 1, 10.051492970092514: 1,
10.050497320216474: 1, 10.045665802441839: 1, 10.045483348868471: 1, 10.026591569010536: 1,
9.999678742870781: 1, 9.990441469722011: 1, 9.988974681261585: 1, 9.98348026220731: 1,
9.980693037266926: 1, 9.971773514568817: 1, 9.966830528294103: 1, 9.964572693774603: 1,
9.948896792086506: 1, 9.947442848685844: 1, 9.922139101473448: 1, 9.89963914138756: 1,
9.892157012924061: 1, 9.861482390054483: 1, 9.855105546153052: 1, 9.821921484895968: 1,
9.815527294896311: 1, 9.813106756236705: 1, 9.808504654029923: 1, 9.805003231653659: 1,
9.789464078691793: 1, 9.755649704327853: 1, 9.750707085170914: 1, 9.732311253424427: 1,
9.721625637742429: 1, 9.710959219662696: 1, 9.71062699212417: 1, 9.705412698804878: 1,
9.693428707365094: 1, 9.690124865258467: 1, 9.689608128670391: 1, 9.689129172468183: 1,
9.688759796950642: 1, 9.687832206009276: 1, 9.652001033551059: 1, 9.630818238393392: 1,
9.628966935663389: 1, 9.62559118424406: 1, 9.62090680893844: 1, 9.590137761058637: 1,
9.587570705710935: 1, 9.58061554374981: 1, 9.543073572319283: 1, 9.527727023245651: 1,
9.525759002712869: 1, 9.524707093245448: 1, 9.50640633965954: 1, 9.504351098442546: 1,
9.493000006567433: 1, 9.492703299781125: 1, 9.492607723705243: 1, 9.491617242833007: 1,
9.491150988409116: 1, 9.485689408758052: 1, 9.452677914801354: 1, 9.450406975078378: 1,
9.447013440198011: 1, 9.441302659716074: 1, 9.441080725715027: 1, 9.396653634003952: 1,
9.392003260576942: 1, 9.39014471359231: 1, 9.378626670713722: 1, 9.344981568219676: 1,
9.326369850085582: 1, 9.31005546186667: 1, 9.297391741745637: 1, 9.281728281331072: 1,
9.280832057876367: 1, 9.26259009335267: 1, 9.255302207988747: 1, 9.25146689064411: 1,
9.250924856655386: 1, 9.242125332989643: 1, 9.237159934405229: 1, 9.237100963887366: 1,
9.236238897697211: 1, 9.224452187116718: 1, 9.220644243747744: 1, 9.21260226816517: 1,
9.211142171891826: 1, 9.204708253259959: 1, 9.20235431499531: 1, 9.198478473941195: 1,
9.167495651491114: 1, 9.160739805117082: 1, 9.157090865488282: 1, 9.154725212792217: 1,
9.141660501828477: 1, 9.119209817524336: 1, 9.105333154151845: 1, 9.1012682182119: 1,
9.085689823417843: 1, 9.083738137294173: 1, 9.071536072907506: 1, 9.067331990136642: 1,
9.062693818559591: 1, 9.053802977054724: 1, 9.043791429511673: 1, 9.036763466477838: 1,
9.008910302322562: 1, 8.993895569186328: 1, 8.99124837130458: 1, 8.969754103252615: 1,
8.962225899420346: 1, 8.951981670570927: 1, 8.948628967408254: 1, 8.936184681923278: 1,
8.921801218294354: 1, 8.903892470244132: 1, 8.903111053646448: 1, 8.892087361088875: 1,
8.890861359746296: 1, 8.877734856365965: 1, 8.870381364488782: 1, 8.857397186568852: 1,
8.848906744992176: 1, 8.837241629653757: 1, 8.806736050474214: 1, 8.785555138785576: 1,
8.779865362641477: 1, 8.744351736518444: 1, 8.743485577402561: 1, 8.738414542992082: 1,
8.72646719400302: 1, 8.720648119703656: 1, 8.713951863549145: 1, 8.70192010377777: 1,
8.683002334531135: 1, 8.678039678888666: 1, 8.673991678595188: 1, 8.660185300900048: 1,
8.657225138773448: 1, 8.653610336744013: 1, 8.652734089998019: 1, 8.645315644743347: 1,
8.633702238187393: 1, 8.628243274669508: 1, 8.622966119846785: 1, 8.620147224973653: 1,
8.61933579650594: 1, 8.617104683120287: 1, 8.612296763743426: 1, 8.592783442055609: 1,
8.587733225745303: 1, 8.585058107211557: 1, 8.575847716897108: 1, 8.569289674970232: 1,
8.553180463716503: 1, 8.547525036198552: 1, 8.517043230909522: 1, 8.491673264928727: 1,
8.46239644198235: 1, 8.448534536397297: 1, 8.442194642549444: 1, 8.428782186323925: 1,
8.418633990816032: 1, 8.412937049898703: 1, 8.398490475364493: 1, 8.39436361002384: 1,
8.385873105511118: 1, 8.377032014123518: 1, 8.372078236888163: 1, 8.343989008860113: 1,
8.337483695838815: 1, 8.335110328030206: 1, 8.284748798919608: 1, 8.279516063531938: 1,
8.27417313745142: 1, 8.267443912182639: 1, 8.238517008419947: 1, 8.233770870851464: 1,
8.21901859312391: 1, 8.165825112824908: 1, 8.161978408749077: 1, 8.13735663462633: 1,
8.135353613265227: 1, 8.1095750182538: 1, 8.103667917812071: 1, 8.097795259716381: 1,
8.095342601698388: 1, 8.081554905641257: 1, 8.060426203623564: 1, 8.051464418614543: 1,
8.046836495953219: 1, 8.032142556865674: 1, 7.994926201129688: 1, 7.983223527408432: 1,
7.976409194711301: 1, 7.968244064732743: 1, 7.9673064162236695: 1, 7.947249889948149: 1,
7.946968686737317: 1, 7.936093195407568: 1, 7.932593297656972: 1, 7.925713053044954: 1,
7.903914633353068: 1, 7.8941223007839: 1, 7.885201119479666: 1, 7.875725451065824: 1,
7.871254737272953: 1, 7.86953717073505: 1, 7.859531304790925: 1, 7.856505595801341: 1,
7.831714623074352: 1, 7.82193332787439: 1, 7.816141305199108: 1, 7.814450762468663: 1,
7.814351071128192: 1, 7.791248453859259: 1, 7.785912496080803: 1, 7.781197862672143: 1,
7.757159740534274: 1, 7.7559472023892875: 1, 7.744933622098121: 1, 7.744589281757106: 1,
7.7393723977802855: 1, 7.710832112752336: 1, 7.6982170963912475: 1, 7.692711051190231: 1,
7.665209960130688: 1, 7.665166462058131: 1, 7.661999460964327: 1, 7.642713497948688: 1,
7.615487873518079: 1, 7.610928698401357: 1, 7.60152301068574: 1, 7.5975846446429145: 1,
7.596550215931377: 1, 7.590409822541649: 1, 7.582186035669615: 1, 7.574653527050101: 1,
7.557605061446475: 1, 7.547980273906755: 1, 7.496661743890522: 1, 7.48060237262502: 1,
7.457874354475245: 1, 7.453758186018209: 1, 7.430172501905035: 1, 7.429242097172217: 1,
7.4133519575657525: 1, 7.391636911625199: 1, 7.387059649948466: 1, 7.352019194277529: 1, 7.34279327
6402963: 1, 7.3357952134243805: 1, 7.289270322314771: 1, 7.273299011293414: 1, 7.269301849194416:

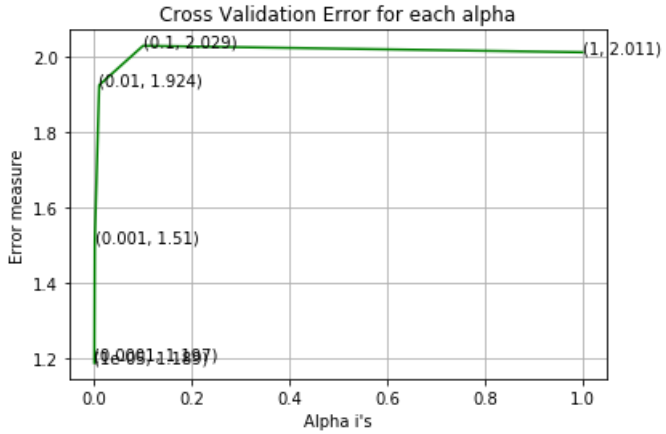
```
1, 7.265049308403936: 1, 7.238356320232765: 1, 7.210523778392332: 1, 7.18300237673326: 1,
7.175289905427371: 1, 7.165153905856623: 1, 7.158527204672461: 1, 7.152677955306988: 1,
7.147460843513956: 1, 7.141703759589404: 1, 7.133453646628293: 1, 7.1133741655888665: 1,
7.044222561970371: 1, 6.992492217817071: 1, 6.972812170460913: 1, 6.957259537193941: 1,
6.868035386409895: 1, 6.839627526248526: 1, 6.8324732525216145: 1, 6.815163920512879: 1,
6.809889078496352: 1, 6.806059368274459: 1, 6.790815469910419: 1, 6.781776595772913: 1,
6.772615784453927: 1, 6.695631067050219: 1, 6.468287217032312: 1, 6.465086316273243: 1,
6.419929143321666: 1, 6.395522151820028: 1})
```

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

In [61]:

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

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

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

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


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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.1886812680107 99
For values of alpha =  0.0001 The log loss is: 1.1971212167862755
For values of alpha =  0.001 The log loss is: 1.509590152505134
For values of alpha =  0.01 The log loss is: 1.9242428463357346
For values of alpha =  0.1 The log loss is: 2.0286544019668864
For values of alpha =  1 The log loss is: 2.011290163534806
```




```
For values of best alpha =  1e-05 The train log loss is: 0.6787278239005274
For values of best alpha =  1e-05 The cross validation log loss is: 1.188681268010799
For values of best alpha =  1e-05 The test log loss is: 1.1083275192189663
```

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

Ans. Yes, it seems like!

In [64]:

```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(max_features=1000)
    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 [65]:

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

```
94.7 % of word of test data appeared in train data
94.7 % of word of Cross Validation appeared in train data
```

n_Words

Q. How many unique n_words for text column are present in train data?

In [66]:

```
print('Number of Unique n_words feature :', train_df['n_words'].value_counts().shape[0])
# the top 10 variations that occured most
print(train_df['n_words'].value_counts().head(7))
```

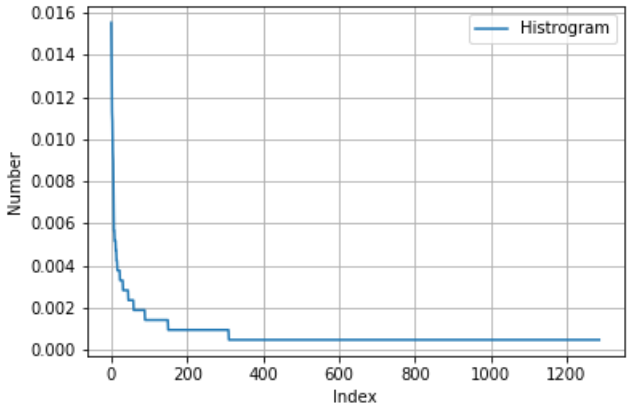
```
Number of Unique n_words feature : 1287
4486    33
```

```
6221    28
4863    24
3539    23
3850    20
4531    19
3001    16
Name: n_words, dtype: int64
```

Q. How are n_words frequencies distributed?

In [67]:

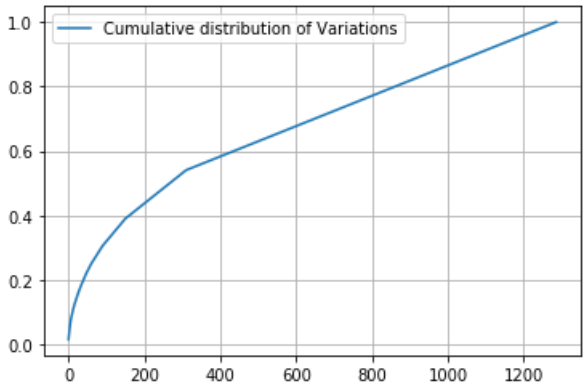
```
unique_ = train_df['n_words'].value_counts()

s = sum(unique_.values);
h = unique_.values/s;
plt.plot(h, label="Histrogram")
plt.xlabel('Index')
plt.ylabel('Number')
plt.legend()
plt.grid()
plt.show()
```



In [68]:

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

```
[0.01553672 0.0287194  0.04001883 ... 0.99905838 0.99952919 1.        ]
```



around 80% of the text contains around 800 words

In [69]:

```
ncount = Counter(train_df['n_words'])
#print('variations and their appearence:')
#print(ncount,'\n',len(ncount))
```

Q. How to featurize numerical feature?

Since it is a numerical feature we will normalize it

In [70]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
#https://stackoverflow.com/questions/30668223/how-to-change-array-shapes-in-in-numpy
from sklearn.preprocessing import MinMaxScaler
words_scalar = MinMaxScaler()
words_scalar.fit(train_df['n_words'].values.reshape(-1,1))

# Now standardize the data with above maen and variance.
word_standardized_train = words_scalar.transform(train_df['n_words'].values.reshape(-1, 1))
print('='*50)
print(word_standardized_train.shape)

word_standardized_cv = words_scalar.transform(cv_df['n_words'].values.reshape(-1, 1))
print('='*50)
print(word_standardized_cv.shape)

word_standardized_test = words_scalar.transform(test_df['n_words'].values.reshape(-1, 1))
print('='*50)
print(word_standardized_test.shape)
```

```
==================================================
(2124, 1)
==================================================
(532, 1)
==================================================
(665, 1)
```

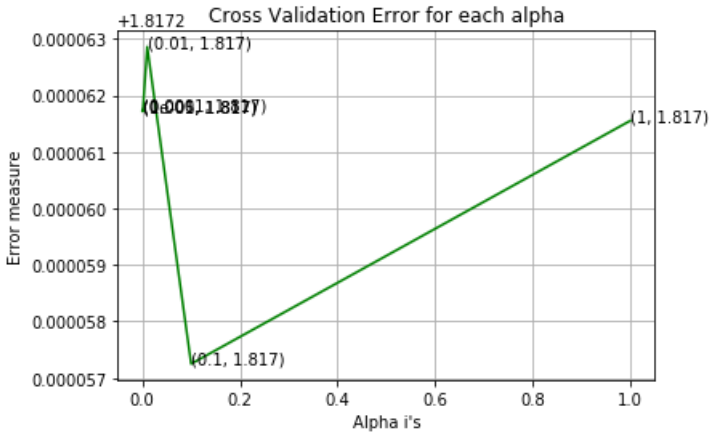
Q. Is this feature useful in predicitng y_i?

In [71]:

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

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

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

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


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

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(word_standardized_train, y_train)
    predict_y = sig_clf.predict_proba(word_standardized_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
```

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.8172617218621707
For values of alpha =  0.0001 The log loss is: 1.817261745709855
For values of alpha =  0.001 The log loss is: 1.817261720582729
For values of alpha =  0.01 The log loss is: 1.8172628605420345
For values of alpha =  0.1 The log loss is: 1.8172572487961491
For values of alpha =  1 The log loss is: 1.8172615511496548
```




```
For values of best alpha =  0.1 The train log loss is: 1.8121317248432394
For values of best alpha =  0.1 The cross validation log loss is: 1.8172572487961491
For values of best alpha =  0.1 The test log loss is: 1.8076798004849226
```

Q. Is this feature stable across train, test and CV datasets?Is the text feature stable across train, test and CV datasets?

In [72]:

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

Q12. How many data points are covered by total 2656 words in test and cross validation data sets?
Ans
1. In test data 370 out of 665 : 55.639097744360896
2. In cross validation data 277 out of 532 : 52.06766917293233

Conclusion

In [73]:

```
#Refer->http://zetcode.com/python/prettytable/
#Refer->https://het.as.utexas.edu/HET/Software/Numpy/reference/generated/numpy.percentile.html
#Refer->https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.round_.html
from prettytable import PrettyTable
x=PrettyTable()

x.field_names=["Feature","Train-Error","CV Error","Test-Error","Stability"] #column headers

x.add_row(["Gene","0.99", "1.21","1.14","Stable"])
x.add_row(["Variation","0.68", "1.70","1.69","Unstable"])
x.add_row(["Text","0.67","1.18", "1.10","Stable"])
x.add_row(["n_words","1.81","1.81", "1.80","Slighlty Stable"])

print(x)
```

```
+-----------+-------------+----------+------------+-----------------+
|  Feature  | Train-Error | CV Error | Test-Error |    Stability    |
+-----------+-------------+----------+------------+-----------------+
|    Gene   |     0.99    |   1.21   |    1.14    |      Stable     |
| Variation |     0.68    |   1.70   |    1.69    |     Unstable    |
|    Text   |     0.67    |   1.18   |    1.10    |      Stable     |
|  n_words  |     1.81    |   1.81   |    1.80    | Slighlty Stable |
+-----------+-------------+----------+------------+-----------------+
```

We can conclude that **Text Feature** has the **lowest error** and hence it is **best** in predicting yi's followed by **gene feature** and **n_words** then **variation feature** at the end.

Data Preparation

In [74]:

```
#Data preparation for ML models.

#Misc. functionns for ML models


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

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

In [75]:

```
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

In [76]:

```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3,max_features=1000)

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

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

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

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

Stacking Features into one

In [77]:

```
print(train_gene_feature_onehotCoding.shape)
print(train_variation_feature_onehotCoding.shape)
print(word_standardized_train.shape)
print(train_text_feature_onehotCoding.shape)
```

```
(2124, 230)
(2124, 1960)
(2124, 1)
(2124, 1000)
```

In [78]:

```
print(test_gene_feature_onehotCoding.shape)
print(test_variation_feature_onehotCoding.shape)
print(word_standardized_test.shape)
print(test_text_feature_onehotCoding.shape)
```

```
(665, 230)
(665, 1960)
(665, 1)
(665, 1000)
```

In [79]:

```
print(cv_gene_feature_onehotCoding.shape)
print(cv_variation_feature_onehotCoding.shape)
```

```
print(cv_variation_feature_onehotCoding.shape)
print(word_standardized_cv.shape)
print(cv_text_feature_onehotCoding.shape)
```

```
(532, 230)
(532, 1960)
(532, 1)
(532, 1000)
```

In [80]:

```
# merging gene, variance and text features

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

train_gene_var_onehotCoding =
hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding,word_standardized_trai
n))
test_gene_var_onehotCoding =
hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCoding,word_standardized_test)
)
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding,
word_standardized_cv))




train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_onehotCoding)).tocs
r()
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_onehotCoding)).tocsr()
test_y = np.array(list(test_df['Class']))

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

In [81]:

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

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 3191)
(number of data points * number of features) in test data =  (665, 3191)
(number of data points * number of features) in cross validation data = (532, 3191)
```

In [82]:

```
train_gene_var_responseCoding =
np.hstack((train_gene_feature_responseCoding,train_variation_feature_responseCoding,word_standardiz
ed_train))
test_gene_var_responseCoding =
np.hstack((test_gene_feature_responseCoding,test_variation_feature_responseCoding,word_standardizec
_test))
cv_gene_var_responseCoding =
np.hstack((cv_gene_feature_responseCoding,cv_variation_feature_responseCoding,word_standardized_cv
))


train_x_responseCoding = np.hstack((train_gene_var_responseCoding,
train_text_feature_responseCoding))
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_responseCoding)
```

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

In [83]:

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

```
Response Coded features :
(number of data points * number of features) in train data =  (2124, 28)
(number of data points * number of features) in test data =  (665, 28)
(number of data points * number of features) in cross validation data = (532, 28)
```

Since the data is in high dimension we can use Naive Bayes or Linear SVM. Naive Bayes also performs better than other when we encounter text feature

Machine Learning Models

Naive Bayes with BOW

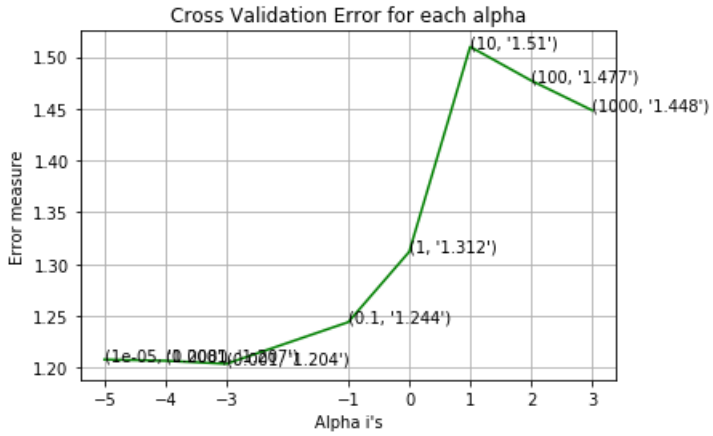
Hyperparameter Tuning

In [84]:

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

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


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


alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
```

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

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


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


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

```
for alpha = 1e-05
Log Loss : 1.2077084331868757
for alpha = 0.0001
Log Loss : 1.2065974192285567
for alpha = 0.001
Log Loss : 1.2035530074397252
for alpha = 0.1
Log Loss : 1.2438501389220906
for alpha = 1
Log Loss : 1.3117680494643689
for alpha = 10
Log Loss : 1.5097995559716062
for alpha = 100
Log Loss : 1.4771552531933678
for alpha = 1000
Log Loss : 1.4484411200852747
```




```
For values of best alpha =  0.001 The train log loss is: 0.4322091666715261
For values of best alpha =  0.001 The cross validation log loss is: 1.2035530074397252
For values of best alpha =  0.001 The test log loss is: 1.1351771834492363
```

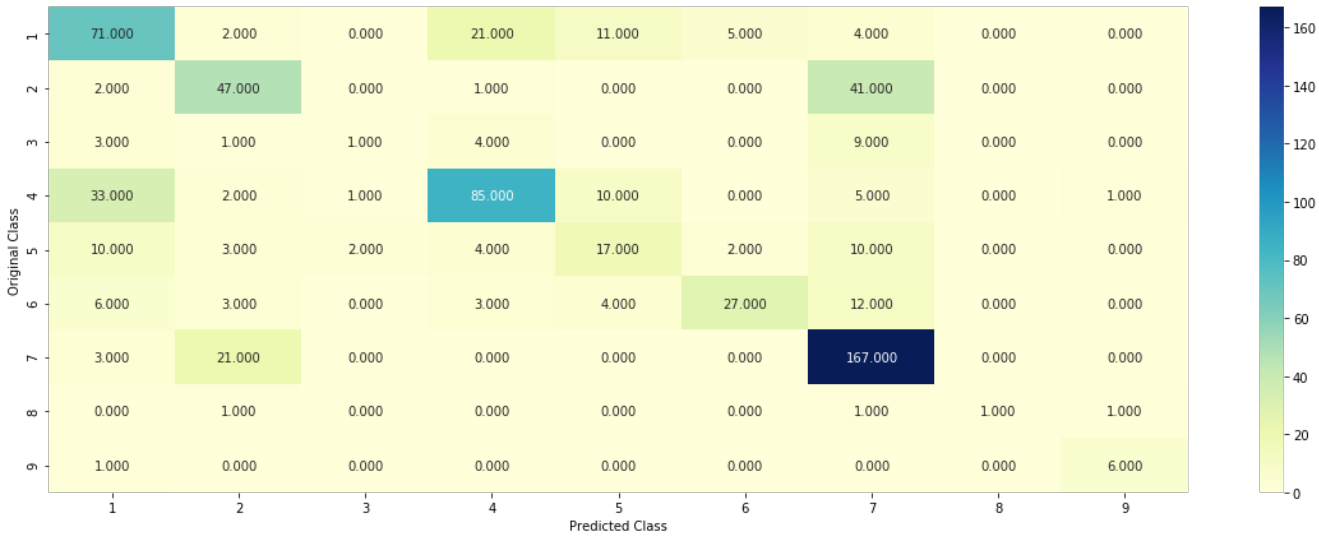
Testing the model with best hyper parameters

Testing the model with best hyper paramters

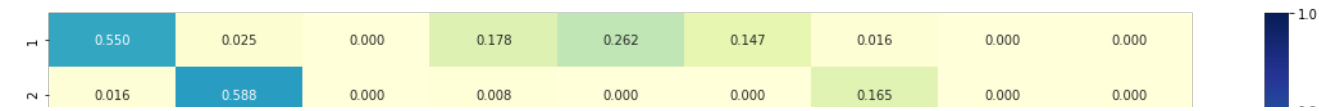
In [85]:

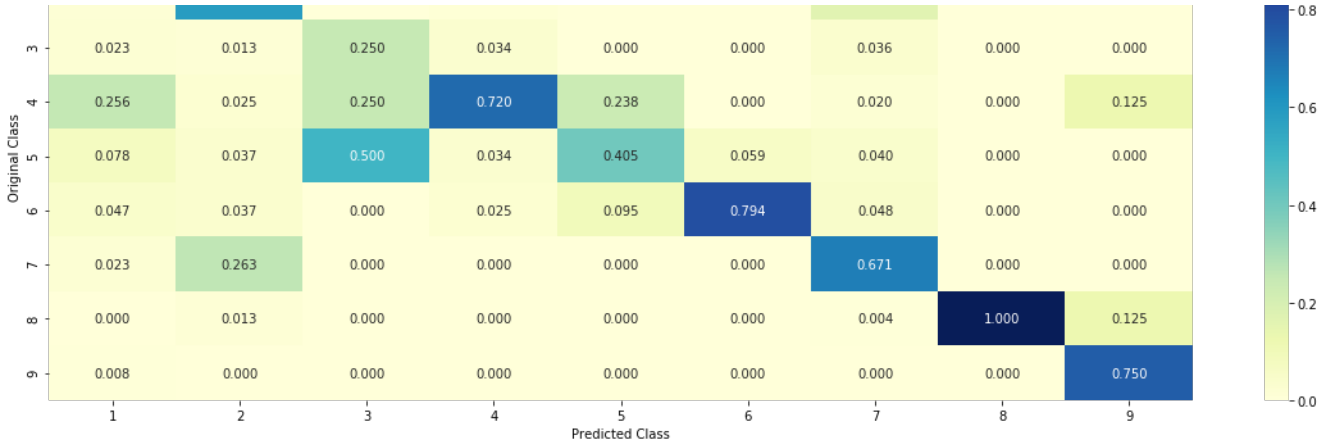
```
# find more about Multinomial Naive base function here http://scikit-
learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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

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

```
Log Loss : 1.1351771834492363
Number of missclassified point : 0.36541353383458647
-------------------- Confusion matrix --------------------
```

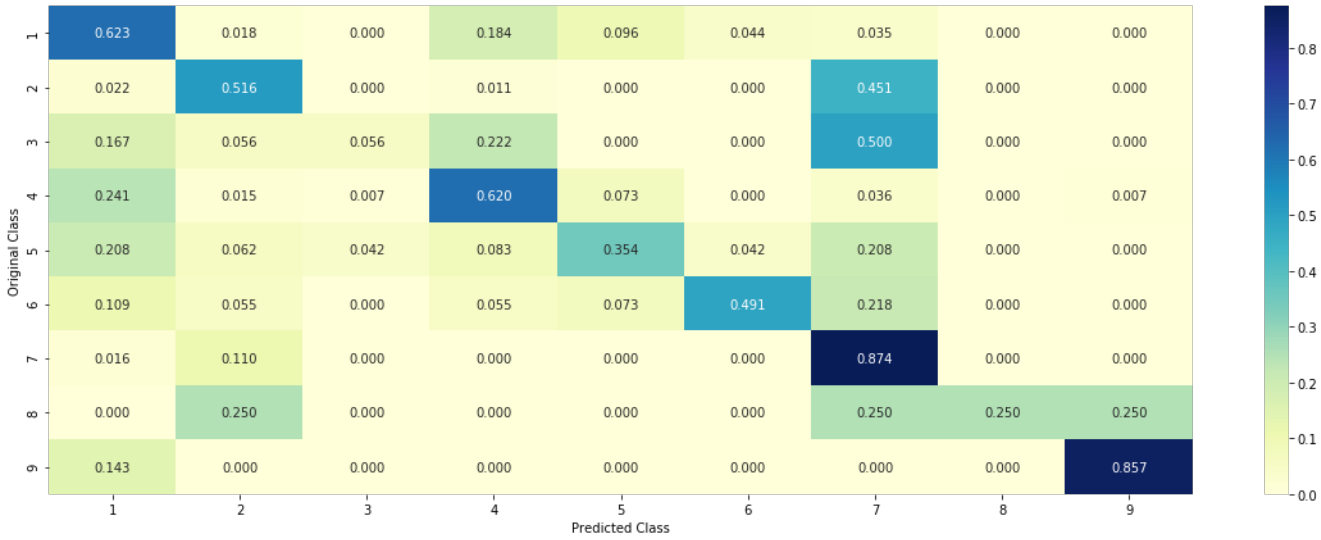



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```






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



WE will focus majority classes i.e 1 ,2,7,4

***For Precision** 55% points which are predicted to be class are actually belong class 1 and 30% of points which are predicted to class 1 are actually belong to class 4

23% points which are predicted to class 4 are actually of class 1 and 67% points are correctly classified as 4

for class 2 34% points are correctly classified and 14.7% points are predicted to be class 2 actually belong to class 7 and for class 7 83% points are correctly classified and 20% actually to class 2 predicted to be class 7

***For recall** 61%points are correclty classified whereas 23.7 of points which are actually belong to class 1 predicted to be class 4 for class 2 only 34% points are correclty classified whereas 60% of points which actually belong to class 2 are predicted to be class 7

so we can conclude that our model is confused between class 4 and class 1 and also between class 2 and class 7

Feature Importance

In [86]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0524 0.0819 0.0102 0.0574 0.0317 0.033  0.7267 0.0037 0.0029]]
Actual Class : 7
--------------------------------------------------
17 Text feature [active] present in test data point [True]
20 Text feature [activating] present in test data point [True]
21 Text feature [kinases] present in test data point [True]
23 Text feature [cellular] present in test data point [True]
25 Text feature [inhibitors] present in test data point [True]
27 Text feature [expression] present in test data point [True]
28 Text feature [indicate] present in test data point [True]
29 Text feature [gst] present in test data point [True]
33 Text feature [activation] present in test data point [True]
34 Text feature [contribute] present in test data point [True]
35 Text feature [hr] present in test data point [True]
36 Text feature [constitutively] present in test data point [True]
37 Text feature [100] present in test data point [True]
38 Text feature [additional] present in test data point [True]
41 Text feature [western] present in test data point [True]
42 Text feature [treatment] present in test data point [True]
43 Text feature [similarly] present in test data point [True]
44 Text feature [shows] present in test data point [True]
45 Text feature [primary] present in test data point [True]
47 Text feature [present] present in test data point [True]
48 Text feature [powerpoint] present in test data point [True]
49 Text feature [ubiquitin] present in test data point [True]
50 Text feature [construct] present in test data point [True]
51 Text feature [cells] present in test data point [True]
52 Text feature [inhibitory] present in test data point [True]
53 Text feature [receptor] present in test data point [True]
54 Text feature [pi3k] present in test data point [True]
55 Text feature [according] present in test data point [True]
56 Text feature [four] present in test data point [True]
57 Text feature [incubated] present in test data point [True]
58 Text feature [would] present in test data point [True]
63 Text feature [3t3] present in test data point [True]
64 Text feature [suggested] present in test data point [True]
65 Text feature [mean] present in test data point [True]
66 Text feature [mechanisms] present in test data point [True]
67 Text feature [highly] present in test data point [True]
68 Text feature [one] present in test data point [True]
70 Text feature [receptors] present in test data point [True]
71 Text feature [inhibitor] present in test data point [True]
72 Text feature [conditions] present in test data point [True]
76 Text feature [figure] present in test data point [True]
77 Text feature [activated] present in test data point [True]
79 Text feature [obtained] present in test data point [True]
81 Text feature [3b] present in test data point [True]
82 Text feature [promote] present in test data point [True]
83 Text feature [mutants] present in test data point [True]
84 Text feature [value] present in test data point [True]
86 Text feature [differentiation] present in test data point [True]
88 Text feature [despite] present in test data point [True]
89 Text feature [increased] present in test data point [True]
90 Text feature [19] present in test data point [True]
92 Text feature [pathways] present in test data point [True]
93 Text feature [type] present in test data point [True]
95 Text feature [enzyme] present in test data point [True]
96 Text feature [increase] present in test data point [True]
97 Text feature [identify] present in test data point [True]
99 Text feature [13] present in test data point [True]
Out of the top  100  features  57 are present in query point
```

In [87]:

```
test_point_index = 5
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
```

```
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0516 0.043  0.01   0.0563 0.0311 0.0313 0.7703 0.0037 0.0028]]
Actual Class : 7
--------------------------------------------------
0 Text feature [000] present in test data point [True]
17 Text feature [active] present in test data point [True]
20 Text feature [activating] present in test data point [True]
21 Text feature [kinases] present in test data point [True]
23 Text feature [cellular] present in test data point [True]
24 Text feature [driven] present in test data point [True]
25 Text feature [inhibitors] present in test data point [True]
26 Text feature [factors] present in test data point [True]
27 Text feature [expression] present in test data point [True]
28 Text feature [indicate] present in test data point [True]
30 Text feature [signalling] present in test data point [True]
31 Text feature [sensitivity] present in test data point [True]
32 Text feature [alterations] present in test data point [True]
33 Text feature [activation] present in test data point [True]
34 Text feature [contribute] present in test data point [True]
35 Text feature [hr] present in test data point [True]
36 Text feature [constitutively] present in test data point [True]
37 Text feature [100] present in test data point [True]
38 Text feature [additional] present in test data point [True]
39 Text feature [myc] present in test data point [True]
40 Text feature [trials] present in test data point [True]
41 Text feature [western] present in test data point [True]
42 Text feature [treatment] present in test data point [True]
43 Text feature [similarly] present in test data point [True]
45 Text feature [primary] present in test data point [True]
46 Text feature [comparison] present in test data point [True]
47 Text feature [present] present in test data point [True]
50 Text feature [construct] present in test data point [True]
51 Text feature [cells] present in test data point [True]
52 Text feature [inhibitory] present in test data point [True]
53 Text feature [receptor] present in test data point [True]
54 Text feature [pi3k] present in test data point [True]
55 Text feature [according] present in test data point [True]
56 Text feature [four] present in test data point [True]
58 Text feature [would] present in test data point [True]
59 Text feature [inhibition] present in test data point [True]
63 Text feature [3t3] present in test data point [True]
64 Text feature [suggested] present in test data point [True]
65 Text feature [mean] present in test data point [True]
66 Text feature [mechanisms] present in test data point [True]
67 Text feature [highly] present in test data point [True]
68 Text feature [one] present in test data point [True]
69 Text feature [showing] present in test data point [True]
70 Text feature [receptors] present in test data point [True]
71 Text feature [inhibitor] present in test data point [True]
72 Text feature [conditions] present in test data point [True]
73 Text feature [reporter] present in test data point [True]
75 Text feature [mutational] present in test data point [True]
76 Text feature [figure] present in test data point [True]
77 Text feature [activated] present in test data point [True]
79 Text feature [obtained] present in test data point [True]
80 Text feature [example] present in test data point [True]
81 Text feature [3b] present in test data point [True]
82 Text feature [promote] present in test data point [True]
83 Text feature [mutants] present in test data point [True]
84 Text feature [value] present in test data point [True]
85 Text feature [tp53] present in test data point [True]
86 Text feature [differentiation] present in test data point [True]
87 Text feature [effects] present in test data point [True]
88 Text feature [despite] present in test data point [True]
89 Text feature [increased] present in test data point [True]
90 Text feature [19] present in test data point [True]
91 Text feature [syndrome] present in test data point [True]
92 Text feature [pathways] present in test data point [True]
93 Text feature [type] present in test data point [True]
94 Text feature [interface] present in test data point [True]
95 Text feature [enzyme] present in test data point [True]
96 Text feature [increase] present in test data point [True]
97 Text feature [identify] present in test data point [True]
99 Text feature [13] present in test data point [True]
```

99 text feature [15] present in test data point [True]
Out of the top 100 features 70 are present in query point

K Nearest Neighbour Classification

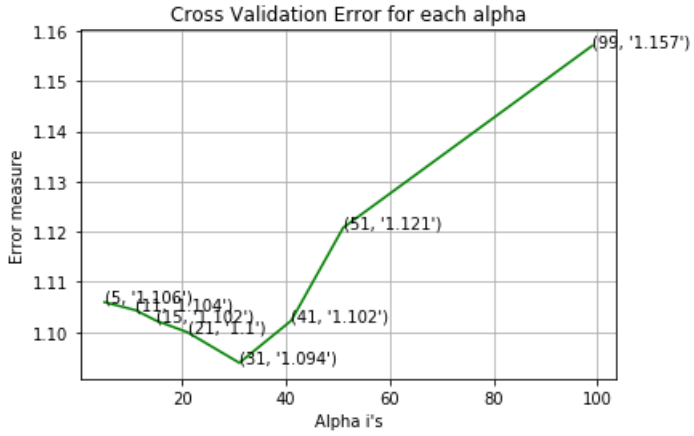
Hyper parameter tuning

In [88]:

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

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


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


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

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


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

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

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

```
for alpha = 5
Log Loss : 1.1059746354986004
for alpha = 11
Log Loss : 1.1043727519575004
for alpha = 15
Log Loss : 1.1022760209878428
for alpha = 21
Log Loss : 1.0999871487377475
for alpha = 31
Log Loss : 1.0938913101888397
for alpha = 41
Log Loss : 1.1023046021988394
for alpha = 51
Log Loss : 1.120786739869738
for alpha = 99
Log Loss : 1.1570140337220967
```




```
For values of best alpha =  31 The train log loss is: 0.8667855890364092
For values of best alpha =  31 The cross validation log loss is: 1.0938913101888397
For values of best alpha =  31 The test log loss is: 0.9936426702111708
```

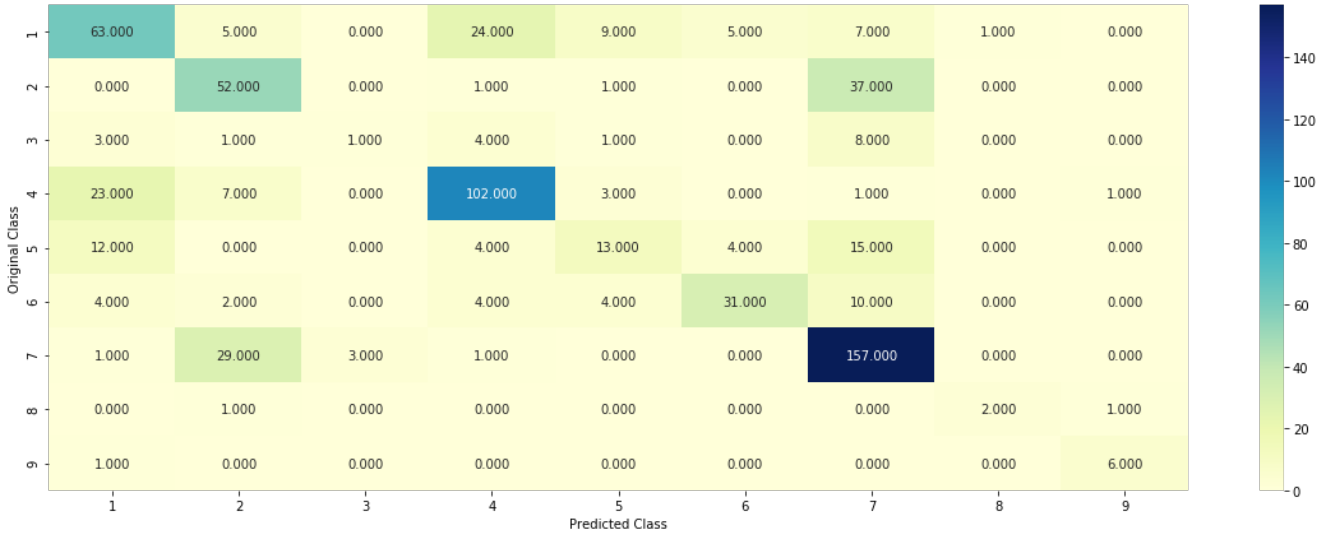
Testing the model with best hyper paramters

In [89]:

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

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

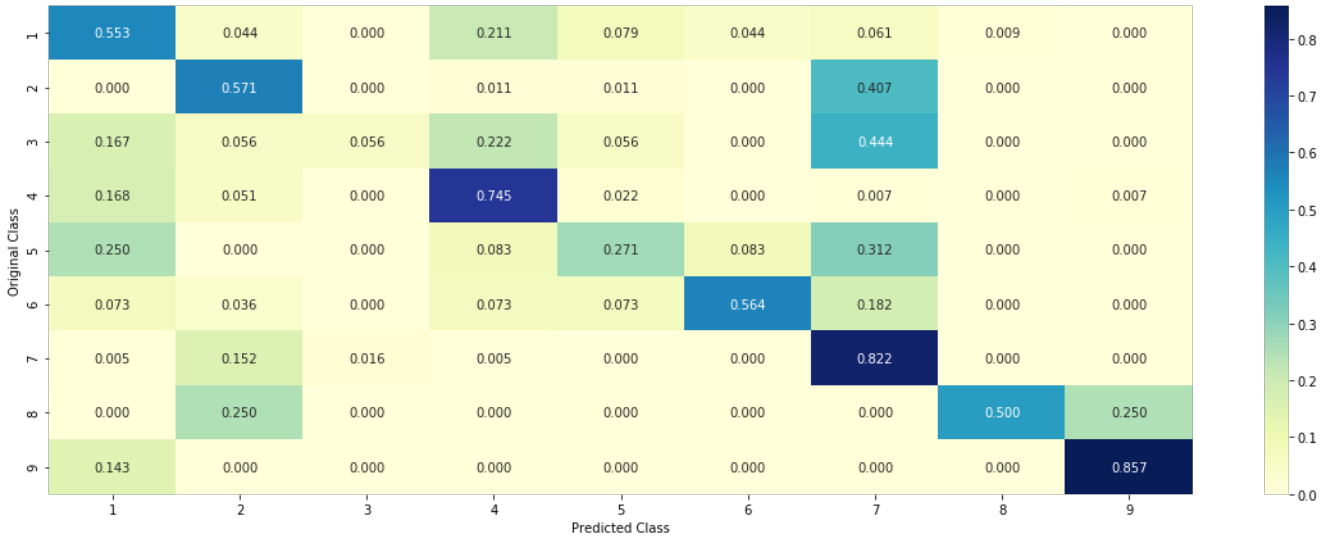
```
Log loss : 0.9936426702111708
Number of mis-classified points : 0.35789473684210527
```

-------------------- Confusion matrix --------------------



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

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



Sample Query point -1

In [90]:

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

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

```
Predicted Class : 1
Actual Class : 7
The  31  nearest neighbours of the test points belongs to classes [2 2 2 2 7 4 1 7 7 7 6 2 7 7 7 7
7 4 7 7 7 7 7 7 7 2 6 7 7 2 7]
Fequency of nearest points : Counter({7: 19, 2: 7, 4: 2, 6: 2, 1: 1})
```

Sample Query point -2

In [91]:

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

test_point_index = 100

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

```
Predicted Class : 7
Actual Class : 7
the k value for knn is 31 and the nearest neighbours of the test points belongs to classes [2 7 7
7 7 7 7 7 7 7 7 7 2 7 7 7 2 7 7 7 7 7 7 7 2 2 7 7 7 7]
Fequency of nearest points : Counter({7: 26, 2: 5})
```

Logistic Regression

With Class balancing and CountVectorizer

Hyper paramter tuning

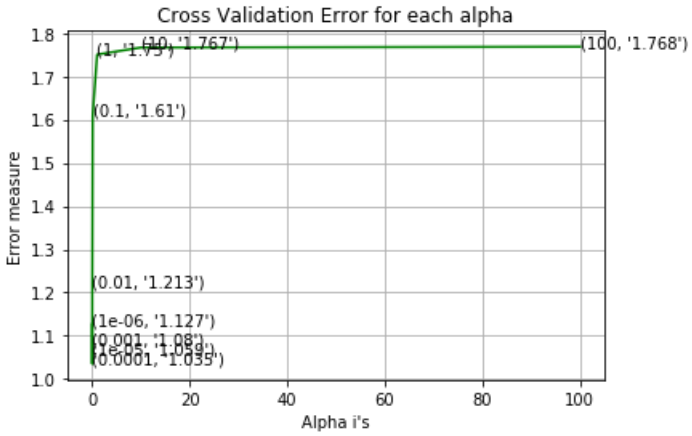
In [92]:

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

# some of methods
# fit(X, y[, coef_init, intercept_init, …]) Fit linear model with Stochastic Gradient Descent.
```

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

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


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

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.1265220706142127
for alpha = 1e-05
Log Loss : 1.0586157988834846
for alpha = 0.0001
Log Loss : 1.035192207127239
for alpha = 0.001
Log Loss : 1.0803177945645663
for alpha = 0.01
Log Loss : 1.2127936663581378
for alpha = 0.1
Log Loss : 1.6097311807925865
```

```
for alpha = 1
Log Loss : 1.7502923953586018
for alpha = 10
Log Loss : 1.7665517058519915
for alpha = 100
Log Loss : 1.768386315140542
```



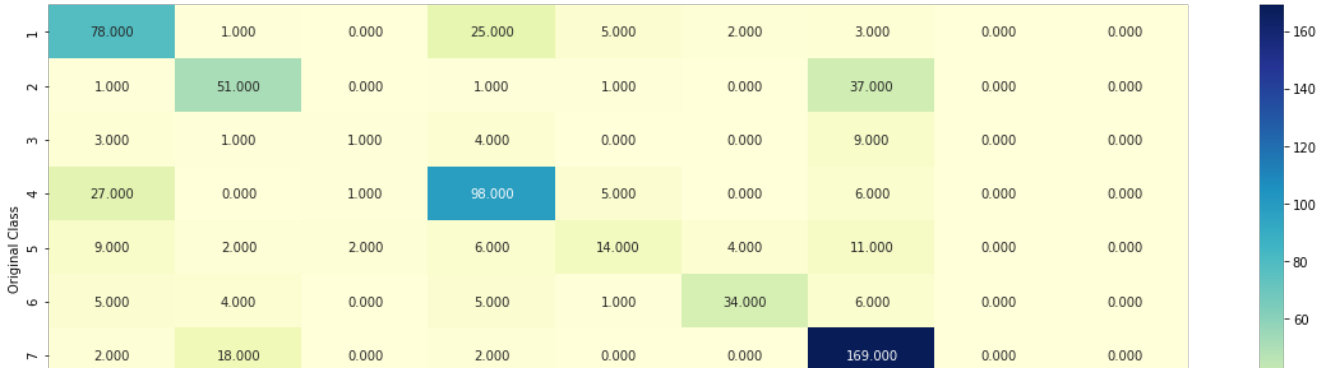

```
For values of best alpha =  0.0001 The train log loss is: 0.3951049619367007
For values of best alpha =  0.0001 The cross validation log loss is: 1.035192207127239
For values of best alpha =  0.0001 The test log loss is: 0.9113459015435094
```

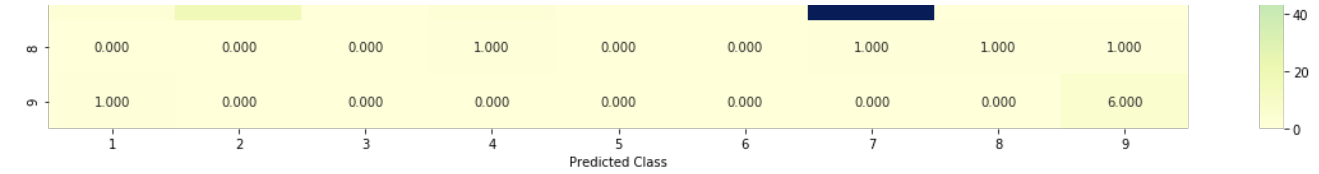
Testing the model with best hyper paramters

In [93]:

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

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

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

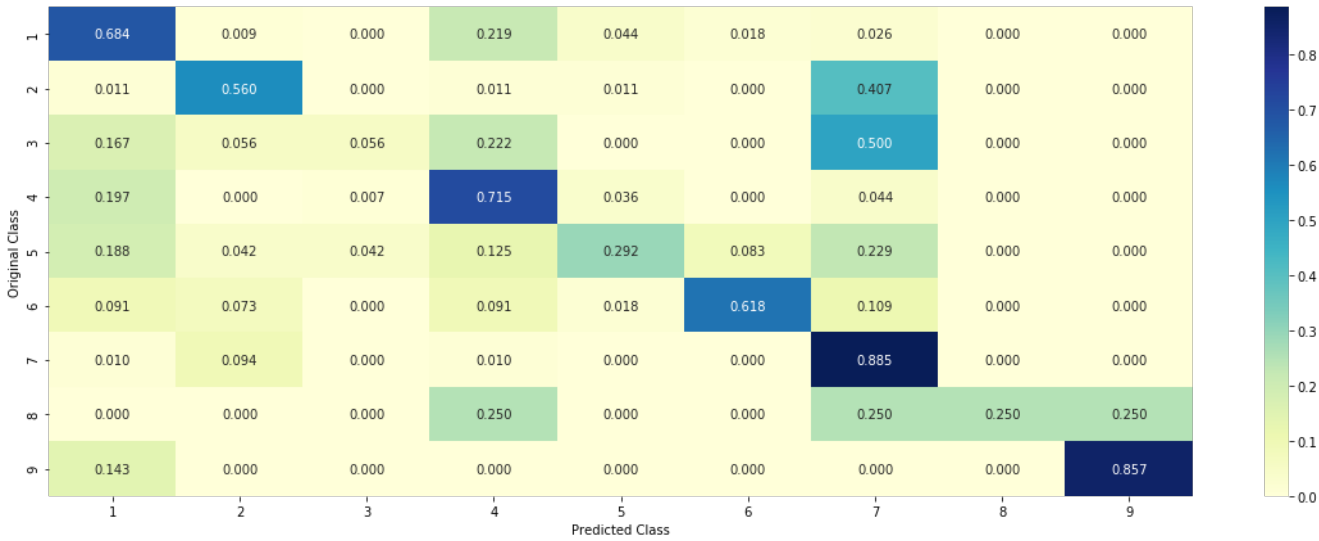
```
Log loss : 0.9113459015435094
Number of mis-classified points : 0.32030075187969925
-------------------- Confusion matrix --------------------
```





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

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



Feature Importance

In [94]:

```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i], yes_no])
```

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

In [95]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0194 0.1882 0.0068 0.0517 0.0107 0.0321 0.6822 0.0054 0.0035]]
Actual Class : 7
--------------------------------------------------
7 Text feature [activating] present in test data point [True]
20 Text feature [active] present in test data point [True]
23 Text feature [oncogenic] present in test data point [True]
27 Text feature [transforming] present in test data point [True]
34 Text feature [enzyme] present in test data point [True]
37 Text feature [constitutively] present in test data point [True]
38 Text feature [ligase] present in test data point [True]
49 Text feature [activation] present in test data point [True]
62 Text feature [treated] present in test data point [True]
69 Text feature [basal] present in test data point [True]
80 Text feature [given] present in test data point [True]
85 Text feature [expression] present in test data point [True]
Out of the top  100  features  12 are present in query point
```

In [96]:

```
test_point_index = 55
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 3
Predicted Class Probabilities: [[0.1304 0.005  0.4821 0.2543 0.0478 0.0735 0.0009 0.0047 0.0013]]
Actual Class : 3
--------------------------------------------------
69 Text feature [introduction] present in test data point [True]
74 Text feature [activation] present in test data point [True]
75 Text feature [2008] present in test data point [True]
77 Text feature [another] present in test data point [True]
78 Text feature [2011] present in test data point [True]
79 Text feature [2009] present in test data point [True]
80 Text feature [consequences] present in test data point [True]
82 Text feature [transcription] present in test data point [True]
84 Text feature [substrate] present in test data point [True]
87 Text feature [resulting] present in test data point [True]
88 Text feature [strand] present in test data point [True]
89 Text feature [class] present in test data point [True]
```

```
90 Text feature [assays] present in test data point [True]
92 Text feature [variants] present in test data point [True]
94 Text feature [2007] present in test data point [True]
98 Text feature [site] present in test data point [True]
Out of the top  100  features  16 are present in query point
```

Without Class balancing and One-Hot

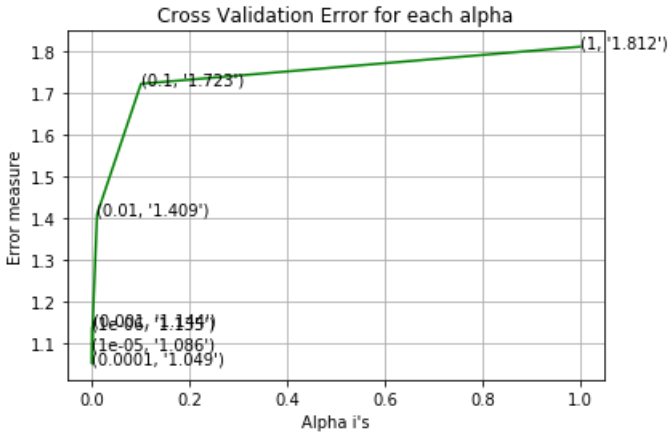
Hyper paramter tuning

In [97]:

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

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

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



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

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

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
```

```
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
for alpha = 1e-06
Log Loss : 1.1351874333121936
for alpha = 1e-05
Log Loss : 1.08573542508159
for alpha = 0.0001
Log Loss : 1.0489961362762585
for alpha = 0.001
Log Loss : 1.143828298310911
for alpha = 0.01
Log Loss : 1.40885634146432
for alpha = 0.1
Log Loss : 1.7226195098940538
for alpha = 1
Log Loss : 1.8117519182260866
```




```
For values of best alpha =  0.0001 The train log loss is: 0.3888026848558825
For values of best alpha =  0.0001 The cross validation log loss is: 1.0489961362762585
For values of best alpha =  0.0001 The test log loss is: 0.9203521569154126
```

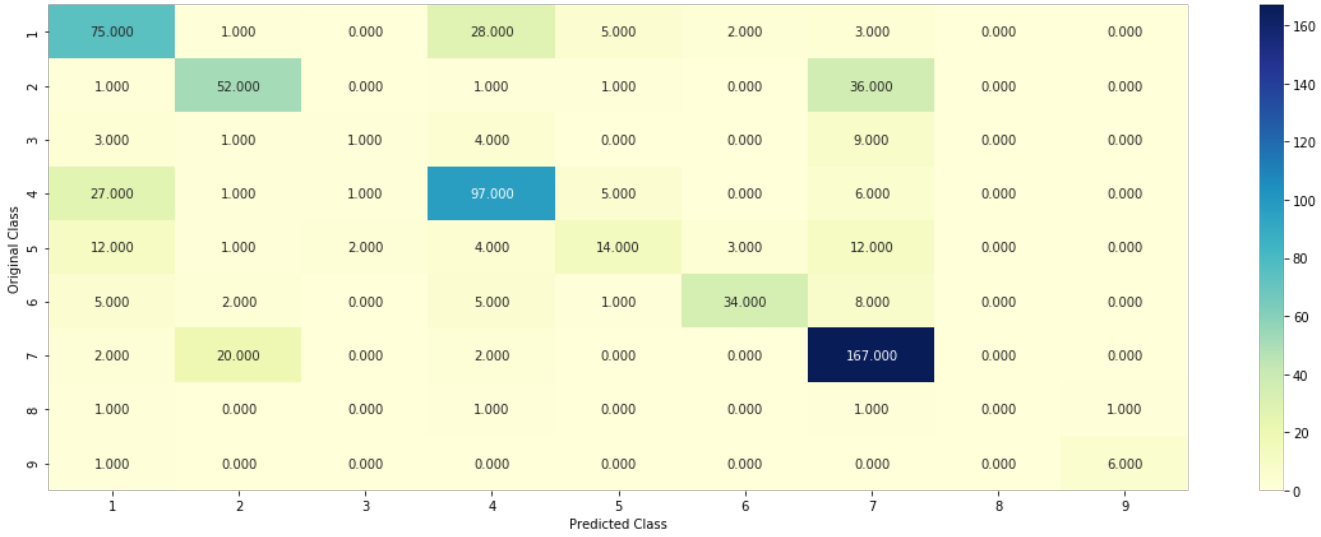
Testing model with best hyper parameters

In [98]:

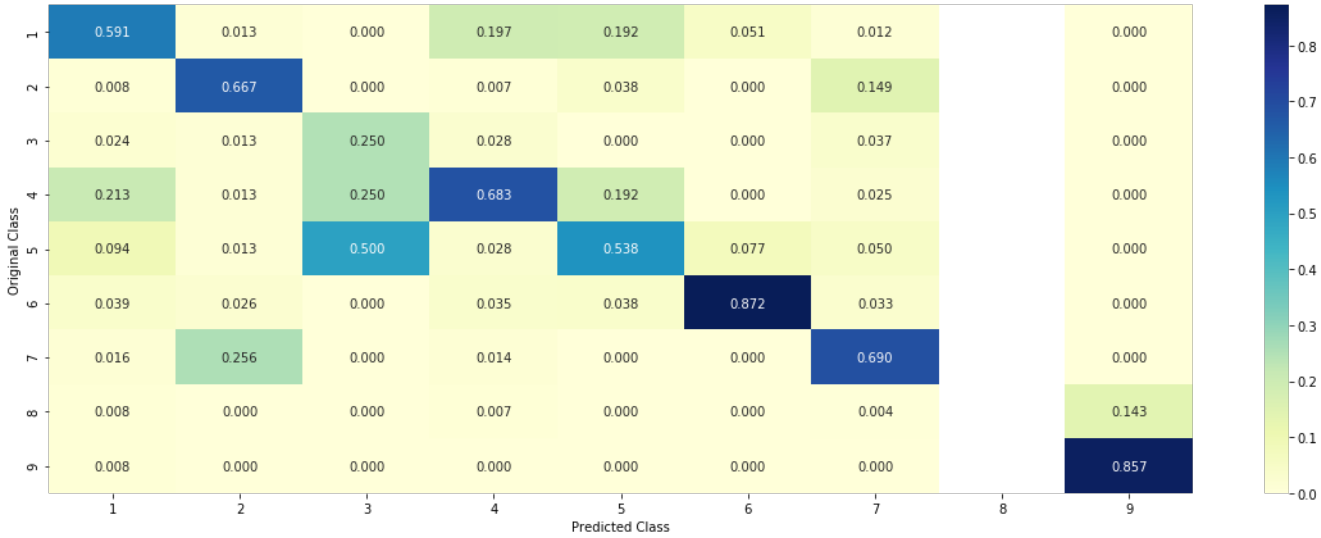
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,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

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

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

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
```

```
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, test_x_onehotCoding, test_y, clf)
```

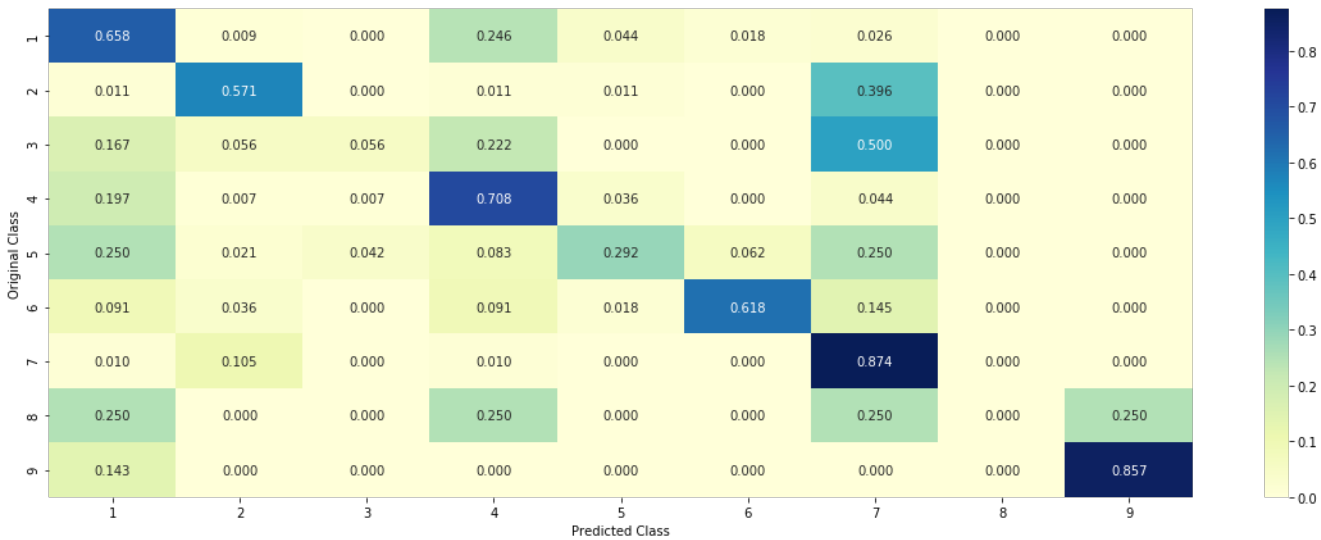
```
Log loss : 0.9203521569154126
Number of mis-classified points : 0.3293233082706767
-------------------- Confusion matrix --------------------
```




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




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



Feature Importance, Inorrectly Classified point

In [99]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0189 0.1969 0.0069 0.056  0.0113 0.0294 0.673  0.0047 0.0027]]
Actual Class : 7
--------------------------------------------------
9 Text feature [activating] present in test data point [True]
33 Text feature [active] present in test data point [True]
40 Text feature [oncogenic] present in test data point [True]
41 Text feature [enzyme] present in test data point [True]
46 Text feature [transforming] present in test data point [True]
65 Text feature [ligase] present in test data point [True]
75 Text feature [activation] present in test data point [True]
81 Text feature [constitutively] present in test data point [True]
99 Text feature [basal] present in test data point [True]
Out of the top  100  features  9 are present in query point
```

In [100]:

```
test_point_index = 2
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[1.210e-02 6.262e-01 1.500e-03 5.300e-03 1.800e-03 5.500e-03 3.424
e-01
  4.800e-03 3.000e-04]]
Actual Class : 7
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

Linear Support Vector Machines(BOW)

Hyper paramter tuning

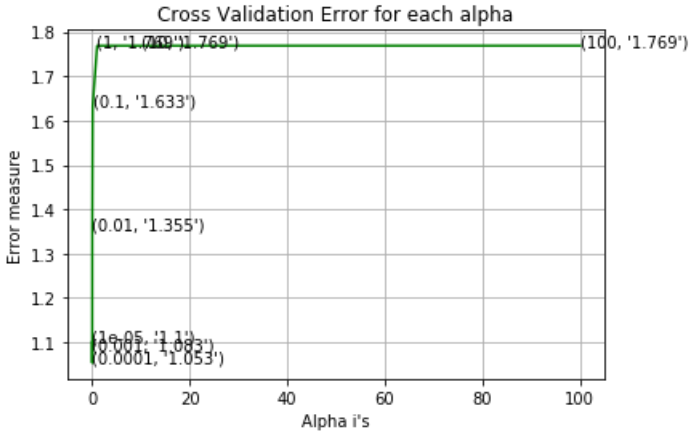
In [101]:

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

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, t
```

```
ol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', ra
ndom_state=None)

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



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

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

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


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

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

```
for C = 1e-05
Log Loss : 1.0999637159876154
for C = 0.0001
Log Loss : 1.0533711249376376
```

```
for C = 0.001
Log Loss : 1.0829814207599708
for C = 0.01
Log Loss : 1.355107693785605
for C = 0.1
Log Loss : 1.632923133115358
for C = 1
Log Loss : 1.7688225191711966
for C = 10
Log Loss : 1.7688224290314336
for C = 100
Log Loss : 1.7688224655403588
```



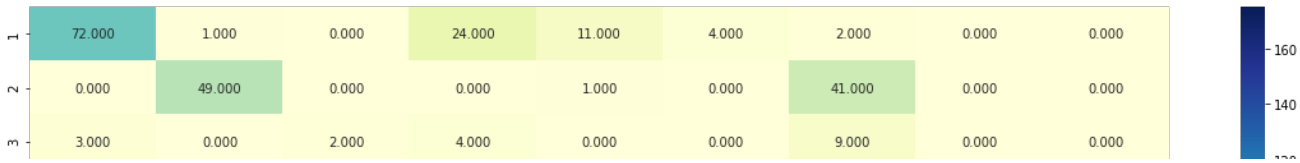

```
For values of best alpha =  0.0001 The train log loss is: 0.3162083461535605
For values of best alpha =  0.0001 The cross validation log loss is: 1.0533711249376376
For values of best alpha =  0.0001 The test log loss is: 0.9577624647134174
```

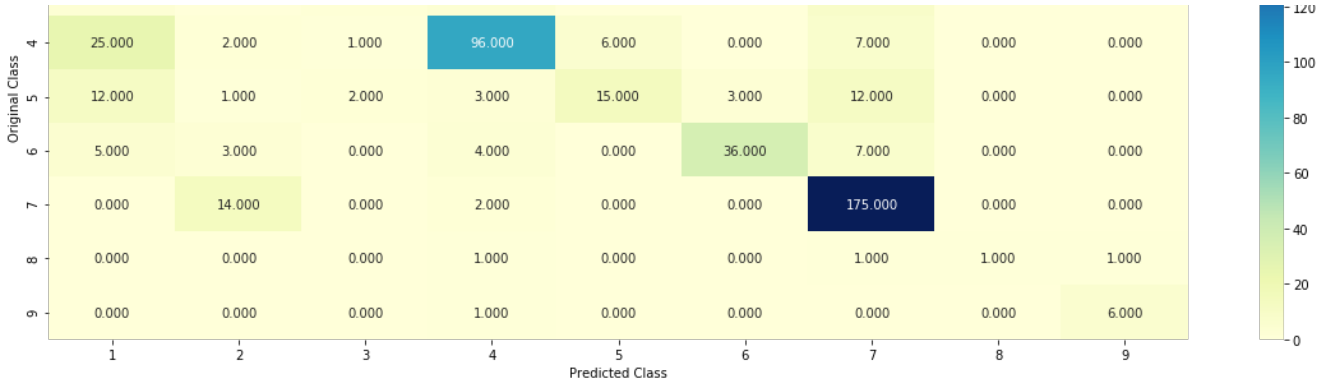
Testing model with best hyper parameters

In [102]:

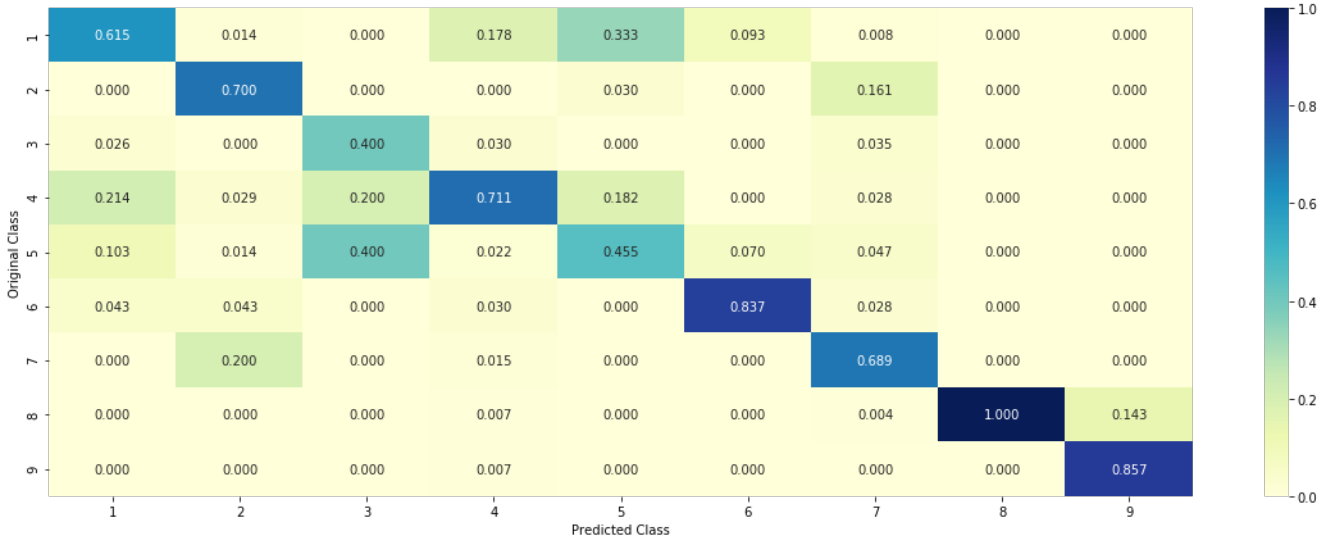
```
# read more about support vector machines with linear kernals here http://scikit-
learn.org/stable/modules/generated/sklearn.svm.SVC.html

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

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


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

```
Log loss : 0.9577624647134174
Number of mis-classified points : 0.32030075187969925
-------------------- Confusion matrix --------------------
```

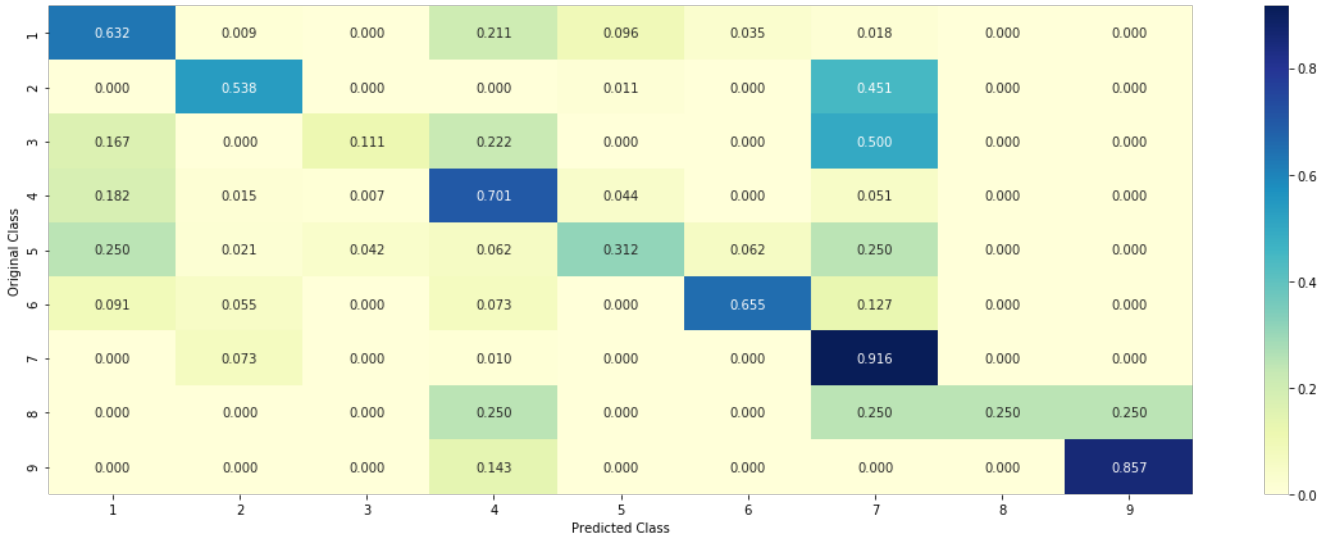





```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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



Feature Importance

In [103]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1

no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
```

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0357 0.2128 0.0082 0.075  0.0212 0.0293 0.6076 0.0054 0.0047]]
Actual Class : 7
--------------------------------------------------
35 Text feature [activating] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

In [104]:

```
test_point_index = 10
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[3.350e-02 2.980e-02 7.000e-04 2.280e-02 2.660e-02 1.062e-01 7.735
e-01
  4.000e-03 2.800e-03]]
Actual Class : 7
--------------------------------------------------
3 Text feature [000] present in test data point [True]
19 Text feature [driven] present in test data point [True]
35 Text feature [activating] present in test data point [True]
38 Text feature [cohort] present in test data point [True]
76 Text feature [sensitivity] present in test data point [True]
Out of the top  100  features  5 are present in query point
```

Random Forest Classifier

Hyper paramter tuning (With One-Hot Encoding)

In [105]:

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

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

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

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-fores
```

```
t-and-their-construction-2/
# --------------------------------


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

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

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

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

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2081004436466944
for n_estimators = 100 and max depth =  10
Log Loss : 1.2435899191551445
for n_estimators = 200 and max depth =  5
Log Loss : 1.1864528508244876
for n_estimators = 200 and max depth =  10
Log Loss : 1.2317901730288845
for n_estimators = 500 and max depth =  5
Log Loss : 1.1817635285729375
for n_estimators = 500 and max depth =  10
Log Loss : 1.2242544272283964
```

```
for n_estimators = 1000 and max depth =  5
Log Loss : 1.183601374662747
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2240463493662943
for n_estimators = 2000 and max depth =  5
Log Loss : 1.181118749555059
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2211850938437339
For values of best estimator =  2000 The train log loss is: 0.836296735160299
For values of best estimator =  2000 The cross validation log loss is: 1.181118749555059
For values of best estimator =  2000 The test log loss is: 1.1372746705790129
```

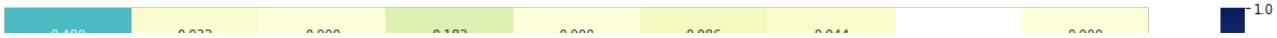
Testing model with best hyper parameters (One Hot Encoding)

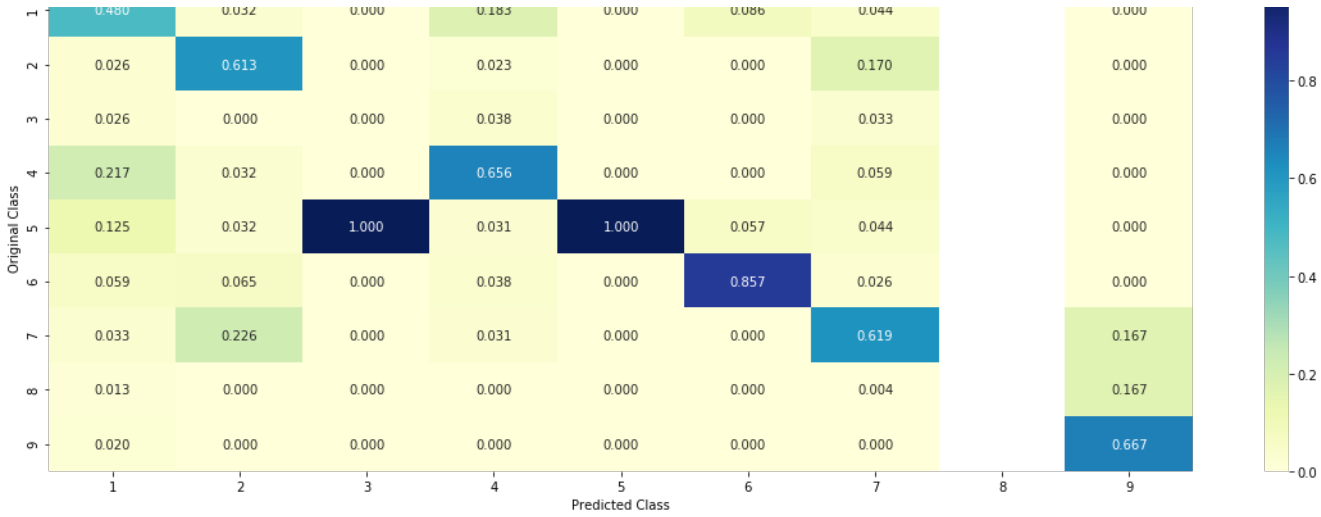
In [106]:

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

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

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

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

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

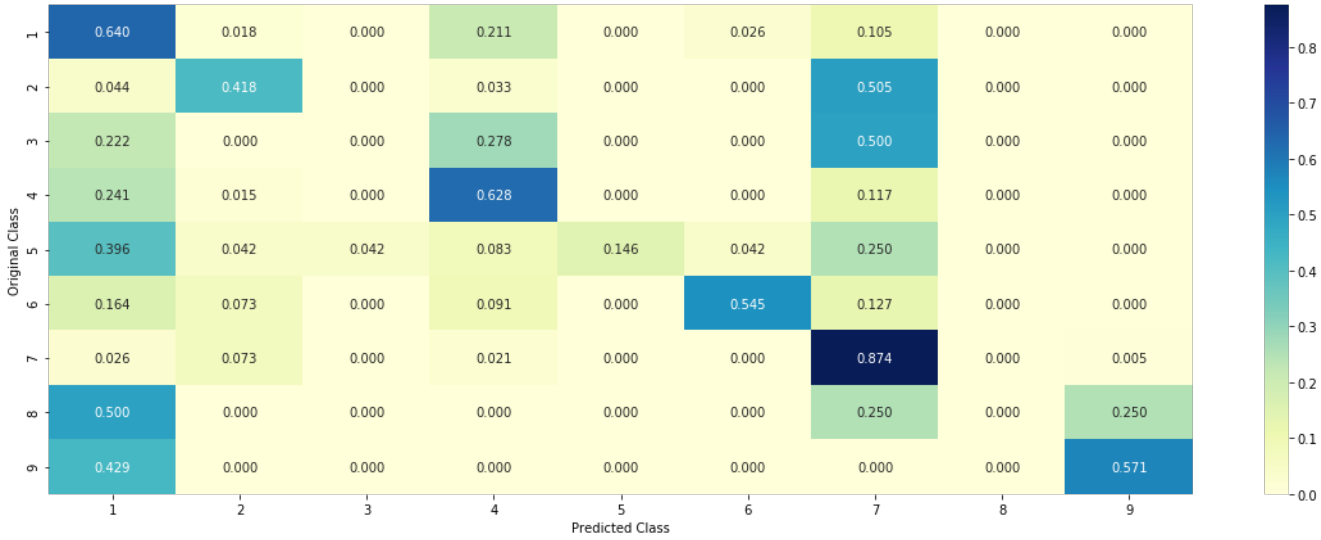
```
Log loss : 1.1372746705790129
Number of mis-classified points : 0.39097744360902253
-------------------- Confusion matrix --------------------
```


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





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



Feature Importance

In [107]:

```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=2000, criterion='gini', max_depth=5, random_state=42, n_j
obs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0611 0.1343 0.019  0.1365 0.0512 0.0473 0.5368 0.0111 0.0027]]
Actual Class : 7
--------------------------------------------------
0 Text feature [kinases] present in test data point [True]
1 Text feature [activation] present in test data point [True]
2 Text feature [active] present in test data point [True]
```

```
3 Text feature [ubiquitin] present in test data point [True]
4 Text feature [inhibitory] present in test data point [True]
5 Text feature [activating] present in test data point [True]
6 Text feature [pi3k] present in test data point [True]
7 Text feature [surface] present in test data point [True]
8 Text feature [functional] present in test data point [True]
9 Text feature [low] present in test data point [True]
10 Text feature [inhibitors] present in test data point [True]
11 Text feature [constitutively] present in test data point [True]
12 Text feature [ml] present in test data point [True]
13 Text feature [one] present in test data point [True]
16 Text feature [pathway] present in test data point [True]
17 Text feature [functions] present in test data point [True]
19 Text feature [proteins] present in test data point [True]
20 Text feature [therefore] present in test data point [True]
21 Text feature [receptors] present in test data point [True]
22 Text feature [moreover] present in test data point [True]
23 Text feature [yet] present in test data point [True]
24 Text feature [various] present in test data point [True]
25 Text feature [deletion] present in test data point [True]
26 Text feature [gst] present in test data point [True]
28 Text feature [stable] present in test data point [True]
29 Text feature [akt1] present in test data point [True]
30 Text feature [treatment] present in test data point [True]
31 Text feature [cells] present in test data point [True]
33 Text feature [construct] present in test data point [True]
34 Text feature [activated] present in test data point [True]
35 Text feature [basal] present in test data point [True]
37 Text feature [cellular] present in test data point [True]
38 Text feature [extracellular] present in test data point [True]
39 Text feature [presence] present in test data point [True]
41 Text feature [treated] present in test data point [True]
42 Text feature [kit] present in test data point [True]
45 Text feature [new] present in test data point [True]
46 Text feature [inhibitor] present in test data point [True]
47 Text feature [breast] present in test data point [True]
49 Text feature [due] present in test data point [True]
51 Text feature [identification] present in test data point [True]
53 Text feature [60] present in test data point [True]
60 Text feature [egf] present in test data point [True]
61 Text feature [provide] present in test data point [True]
62 Text feature [phosphate] present in test data point [True]
64 Text feature [phosphorylation] present in test data point [True]
65 Text feature [defects] present in test data point [True]
67 Text feature [affect] present in test data point [True]
68 Text feature [mek1] present in test data point [True]
69 Text feature [variants] present in test data point [True]
71 Text feature [sequence] present in test data point [True]
72 Text feature [furthermore] present in test data point [True]
73 Text feature [40] present in test data point [True]
75 Text feature [number] present in test data point [True]
80 Text feature [phosphorylated] present in test data point [True]
82 Text feature [taken] present in test data point [True]
83 Text feature [oncogenic] present in test data point [True]
84 Text feature [together] present in test data point [True]
85 Text feature [domain] present in test data point [True]
92 Text feature [promote] present in test data point [True]
96 Text feature [rate] present in test data point [True]
99 Text feature [substrate] present in test data point [True]
Out of the top  100  features  62 are present in query point
```

In [108]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0307 0.2216 0.0126 0.0303 0.0369 0.0338 0.6243 0.0076 0.0023]]
Actuall Class : 7
--------------------------------------------------
0 Text feature [kinases] present in test data point [True]
1 Text feature [activation] present in test data point [True]
2 Text feature [active] present in test data point [True]
3 Text feature [ubiquitin] present in test data point [True]
4 Text feature [inhibitory] present in test data point [True]
5 Text feature [activating] present in test data point [True]
6 Text feature [pi3k] present in test data point [True]
7 Text feature [surface] present in test data point [True]
8 Text feature [functional] present in test data point [True]
9 Text feature [low] present in test data point [True]
10 Text feature [inhibitors] present in test data point [True]
11 Text feature [constitutively] present in test data point [True]
12 Text feature [ml] present in test data point [True]
13 Text feature [one] present in test data point [True]
15 Text feature [trials] present in test data point [True]
16 Text feature [pathway] present in test data point [True]
17 Text feature [functions] present in test data point [True]
18 Text feature [published] present in test data point [True]
19 Text feature [proteins] present in test data point [True]
20 Text feature [therefore] present in test data point [True]
21 Text feature [receptors] present in test data point [True]
23 Text feature [yet] present in test data point [True]
24 Text feature [various] present in test data point [True]
25 Text feature [deletion] present in test data point [True]
26 Text feature [gst] present in test data point [True]
27 Text feature [erlotinib] present in test data point [True]
28 Text feature [stable] present in test data point [True]
30 Text feature [treatment] present in test data point [True]
31 Text feature [cells] present in test data point [True]
33 Text feature [construct] present in test data point [True]
34 Text feature [activated] present in test data point [True]
37 Text feature [cellular] present in test data point [True]
38 Text feature [extracellular] present in test data point [True]
39 Text feature [presence] present in test data point [True]
41 Text feature [treated] present in test data point [True]
42 Text feature [kit] present in test data point [True]
43 Text feature [clear] present in test data point [True]
45 Text feature [new] present in test data point [True]
46 Text feature [inhibitor] present in test data point [True]
48 Text feature [signalling] present in test data point [True]
49 Text feature [due] present in test data point [True]
50 Text feature [therapy] present in test data point [True]
51 Text feature [identification] present in test data point [True]
52 Text feature [resistant] present in test data point [True]
53 Text feature [60] present in test data point [True]
54 Text feature [inhibition] present in test data point [True]
55 Text feature [fact] present in test data point [True]
57 Text feature [inactive] present in test data point [True]
59 Text feature [overall] present in test data point [True]
61 Text feature [provide] present in test data point [True]
62 Text feature [phosphate] present in test data point [True]
64 Text feature [phosphorylation] present in test data point [True]
65 Text feature [defects] present in test data point [True]
66 Text feature [driven] present in test data point [True]
67 Text feature [affect] present in test data point [True]
68 Text feature [mek1] present in test data point [True]
69 Text feature [variants] present in test data point [True]
70 Text feature [often] present in test data point [True]
71 Text feature [sequence] present in test data point [True]
72 Text feature [furthermore] present in test data point [True]
73 Text feature [40] present in test data point [True]
74 Text feature [responses] present in test data point [True]
75 Text feature [number] present in test data point [True]
76 Text feature [factors] present in test data point [True]
77 Text feature [pattern] present in test data point [True]
78 Text feature [risk] present in test data point [True]
79 Text feature [syndrome] present in test data point [True]
80 Text feature [phosphorylated] present in test data point [True]
81 Text feature [added] present in test data point [True]
83 Text feature [oncogenic] present in test data point [True]
84 Text feature [together] present in test data point [True]
85 Text feature [domain] present in test data point [True]
86 Text feature [clinically] present in test data point [True]
```

```
87 Text feature [impact] present in test data point [True]
89 Text feature [nucleotide] present in test data point [True]
90 Text feature [abl] present in test data point [True]
91 Text feature [nuclear] present in test data point [True]
92 Text feature [promote] present in test data point [True]
94 Text feature [double] present in test data point [True]
95 Text feature [splicing] present in test data point [True]
96 Text feature [rate] present in test data point [True]
98 Text feature [assess] present in test data point [True]
99 Text feature [substrate] present in test data point [True]
Out of the top  100  features  83 are present in query point
```

Hyper paramter tuning (With Response Encoding)

In [109]:

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

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

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

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


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

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

```
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)),
(features[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

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

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

```
for n_estimators = 10 and max depth =  2
Log Loss : 1.6551792678996784
for n_estimators = 10 and max depth =  3
Log Loss : 1.7248292384605335
for n_estimators = 10 and max depth =  5
Log Loss : 1.7230024542790625
for n_estimators = 10 and max depth =  10
Log Loss : 1.6321242227375365
for n_estimators = 50 and max depth =  2
Log Loss : 1.9728386093537045
for n_estimators = 50 and max depth =  3
Log Loss : 1.5639016417241163
for n_estimators = 50 and max depth =  5
Log Loss : 1.4139030619596331
for n_estimators = 50 and max depth =  10
Log Loss : 1.6628785281052327
for n_estimators = 100 and max depth =  2
Log Loss : 1.9076613274354666
for n_estimators = 100 and max depth =  3
Log Loss : 1.5531191726742573
for n_estimators = 100 and max depth =  5
Log Loss : 1.4584189006046473
for n_estimators = 100 and max depth =  10
Log Loss : 1.6639734681082137
for n_estimators = 200 and max depth =  2
Log Loss : 1.6775197325084745
for n_estimators = 200 and max depth =  3
Log Loss : 1.5613194589682504
for n_estimators = 200 and max depth =  5
Log Loss : 1.4705275929360775
for n_estimators = 200 and max depth =  10
Log Loss : 1.6711599081679174
for n_estimators = 500 and max depth =  2
Log Loss : 1.6356377838244198
for n_estimators = 500 and max depth =  3
Log Loss : 1.5063881424738752
for n_estimators = 500 and max depth =  5
Log Loss : 1.4259092737166024
for n_estimators = 500 and max depth =  10
Log Loss : 1.6502190335291287
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6717140489495257
for n_estimators = 1000 and max depth =  3
Log Loss : 1.4924928793227878
for n_estimators = 1000 and max depth =  5
Log Loss : 1.407411442236435
for n_estimators = 1000 and max depth =  10
Log Loss : 1.674589120459733
For values of best alpha =  1000 The train log loss is: 0.0614228974025194
For values of best alpha =  1000 The cross validation log loss is: 1.4074114422364352
```

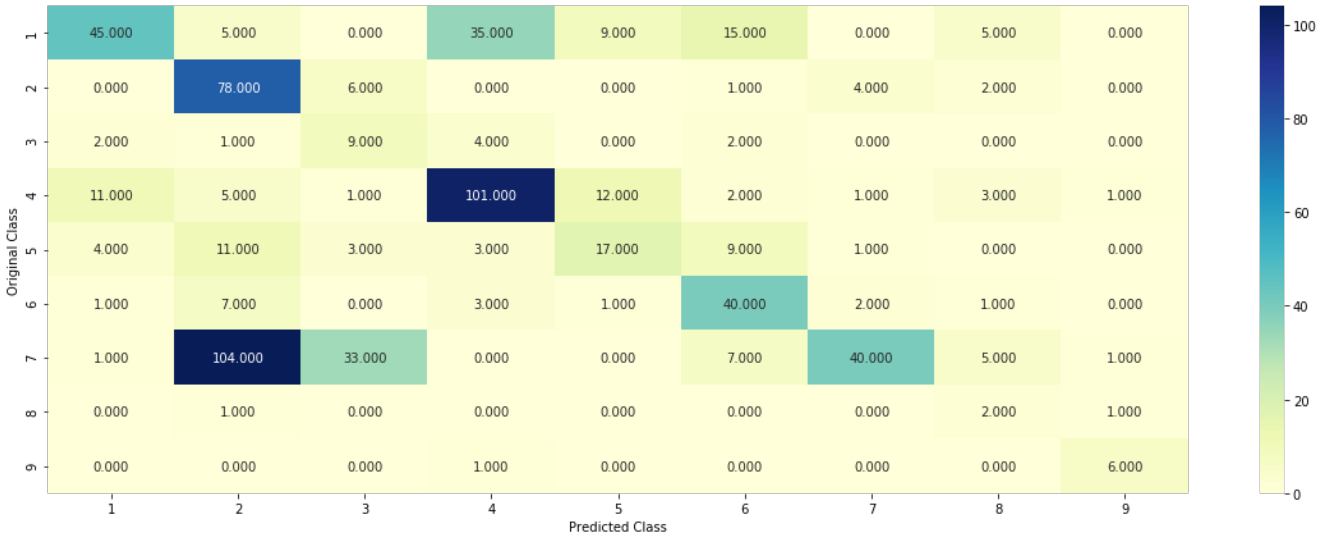
```
For values of best alpha =  1000 The test log loss is: 1.3337549208506043
```

Testing model with best hyper parameters (Response Coding)

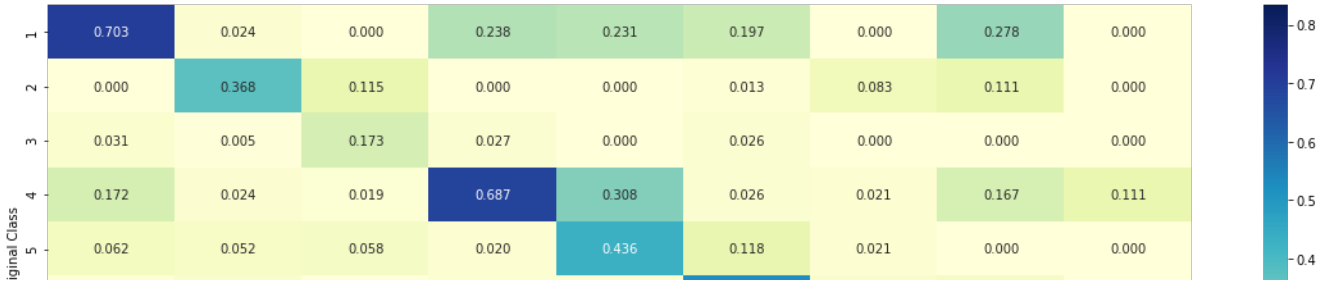
In [110]:

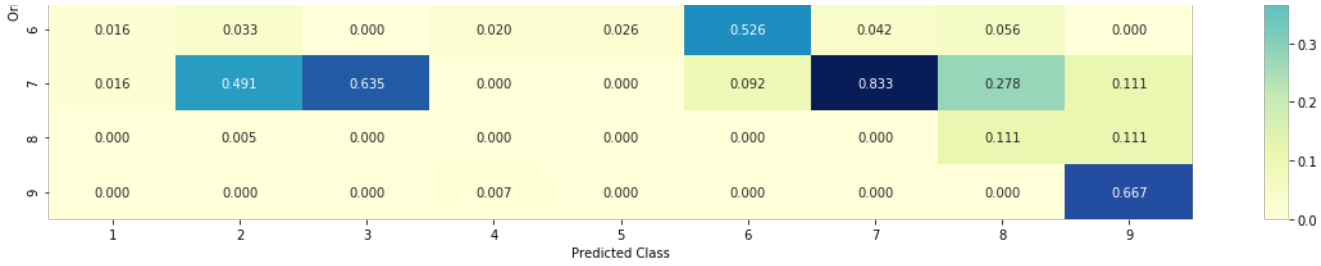
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None,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

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

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

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

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

```
Log loss : 1.3337549208506043
Number of mis-classified points : 0.4917293233082707
-------------------- Confusion matrix --------------------
```




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





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

Feature Importance

In [111]:

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

```
Predicted Class : 2
Predicted Class Probabilities: [[0.003  0.9679 0.0035 0.0044 0.0026 0.0045 0.0088 0.003  0.0024]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

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

In [112]:

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


test_point_index = 5
no_feature = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0225 0.148  0.2746 0.0361 0.0504 0.0745 0.2899 0.0605 0.0436]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

Stacking Classifier

Stacking Classifer

Hyper parameter tuning with One-Hot

In [114]:

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

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

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


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

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


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

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

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

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


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

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

#clf3 = KNeighborsClassifier(n_neighbors=31)
#clf3.fit(train_x_responseCoding, train_y)
#sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")
#sig_clf3.fit(train_x_responseCoding, train_y)



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

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

```
Logistic Regression :  Log Loss: 1.03
Support vector machines : Log Loss: 1.06
Naive Bayes : Log Loss: 1.20
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.806
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.637
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.228
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.238
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.526
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.839
```

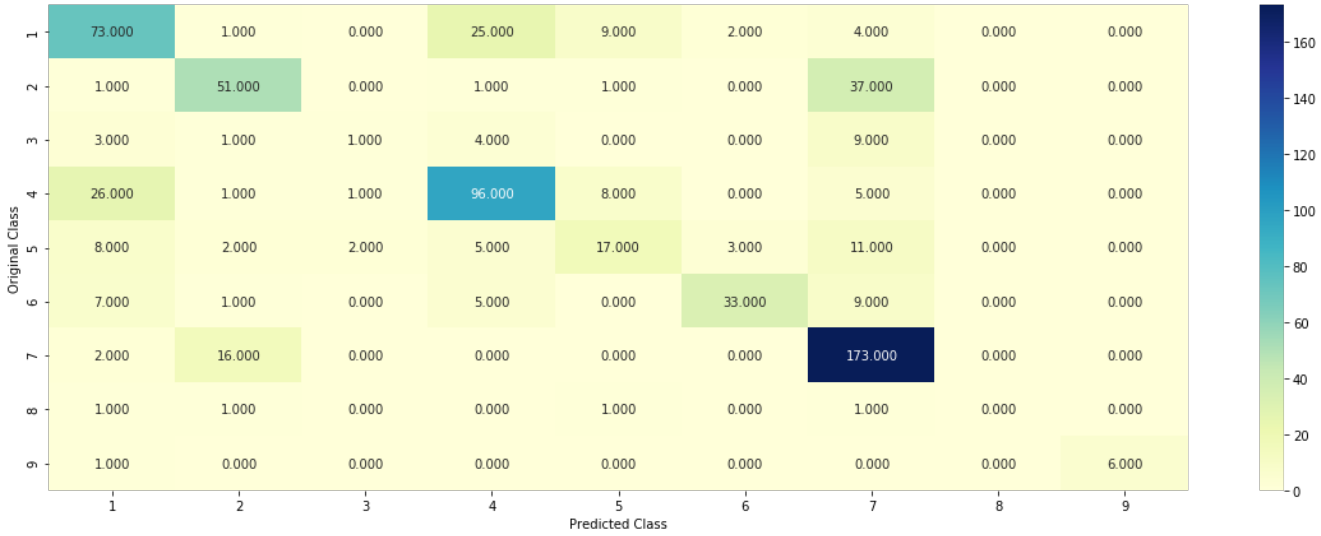
Testing the model with the best hyper parameters

In [115]:

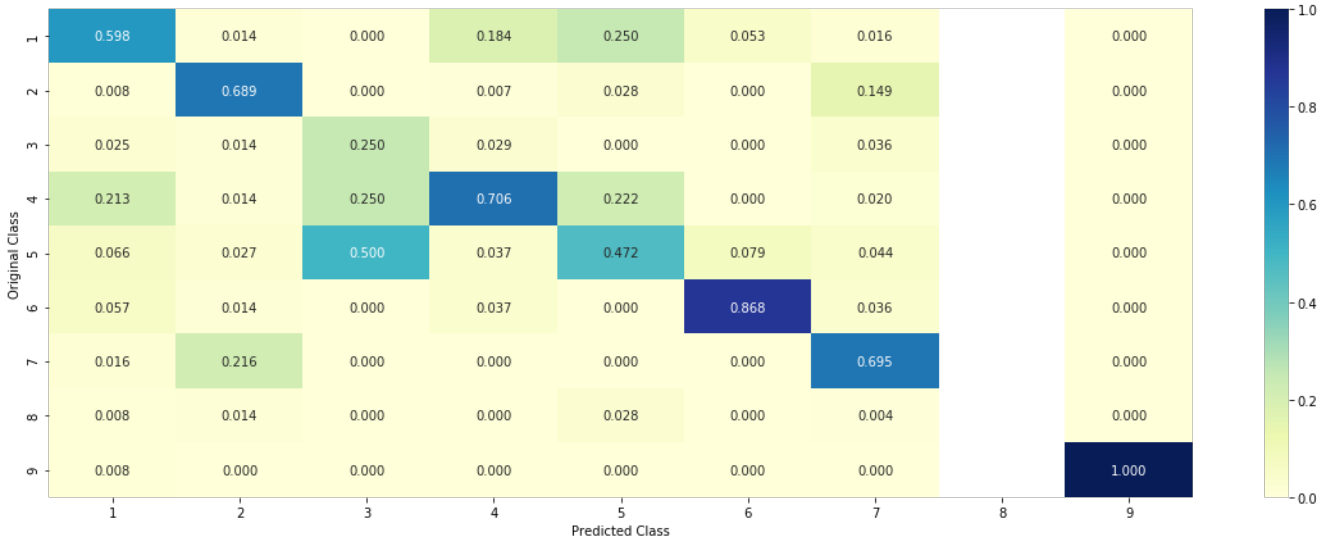
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba
s=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

```
Log loss (train) on the stacking classifier : 0.16014917220117575
```

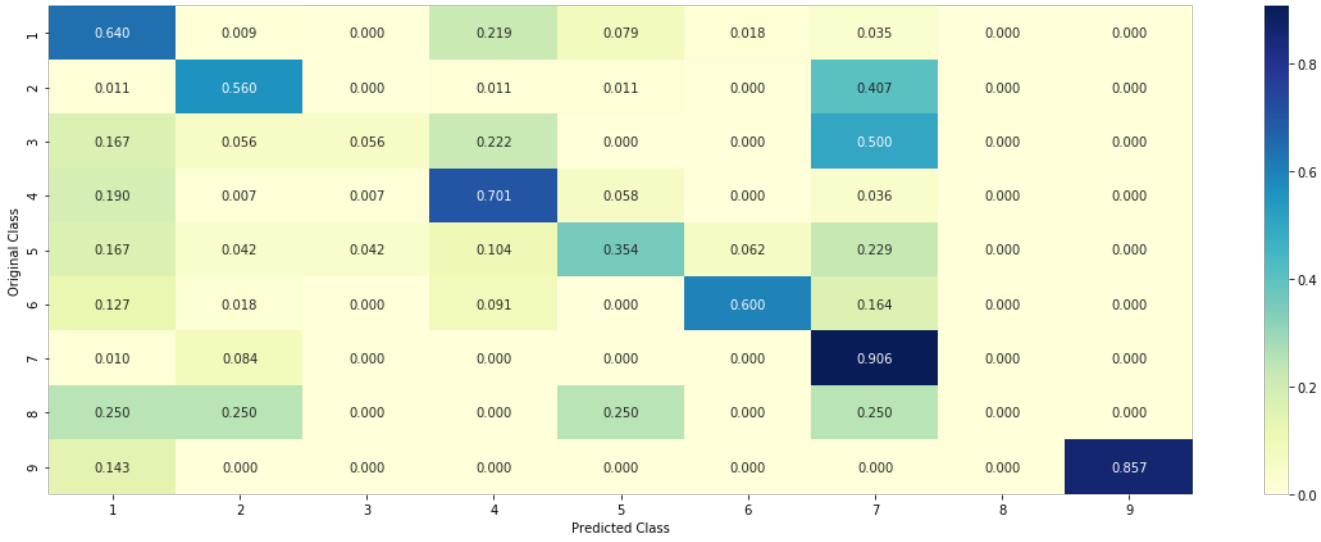
```
Log loss (CV) on the stacking classifier : 1.2383504200644957
Log loss (test) on the stacking classifier : 1.08173584980263
Number of missclassified point : 0.3233082706766917
-------------------- Confusion matrix --------------------
```




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




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



Maximum Voting classifier

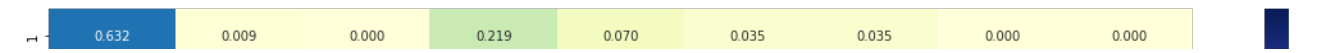
In [116]:

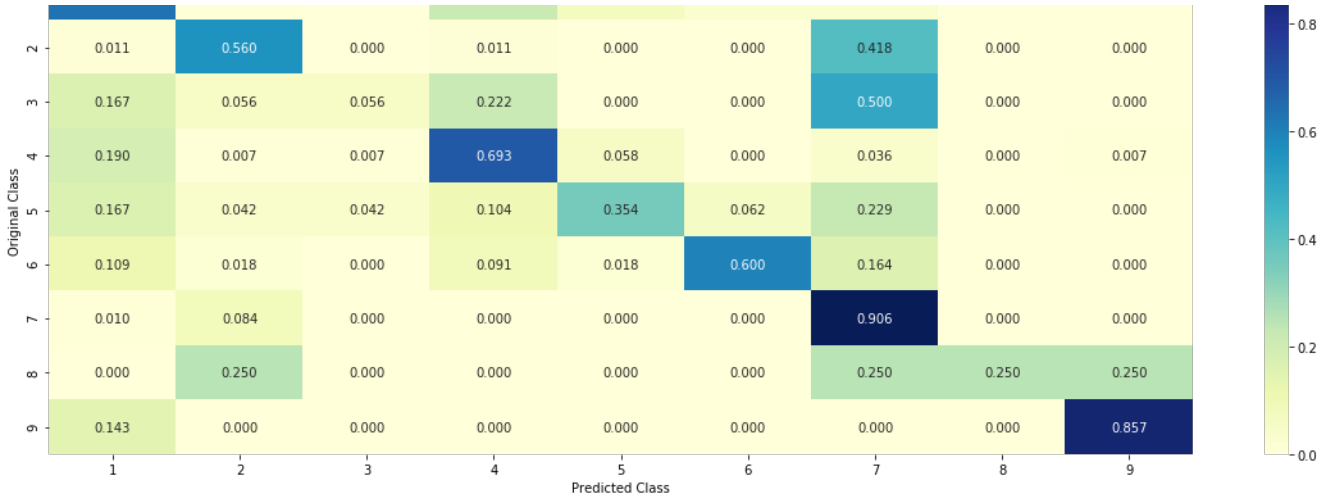
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('nb', sig_clf3)], voting=
'soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log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

```
Log loss (train) on the VotingClassifier : 0.3781529693063566
Log loss (CV) on the VotingClassifier : 1.0262488233502405
Log loss (test) on the VotingClassifier : 0.9573672115284169
Number of missclassified point : 0.324812030075188
-------------------- Confusion matrix --------------------
```


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


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





Logistic regression with CountVectorizer Features, including both unigrams and bigrams

In [117]:

```
train_df.columns
```

Out[117]:

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

In [118]:

```
#Getting the values of each features of each train/test and cv data

train_variation=train_df['Variation'].values
#print(train_variation)

test_variation=test_df['Variation'].values

cv_variation=cv_df['Variation'].values

train_gene=train_df['Gene'].values
#print(train_gene)

test_gene=test_df['Gene'].values

cv_gene=cv_df['Gene'].values

train_text=train_df['TEXT'].values
#print(train_text)


test_text=test_df['TEXT'].values

cv_text=cv_df['TEXT'].values
```

In [119]:

```
from sklearn.feature_extraction.text import CountVectorizer

cnt_vect=CountVectorizer(ngram_range=(1, 2),max_features = 2000)#Intitalization

train_variation=cnt_vect.fit_transform(train_variation) #training and transform
train_variation=normalize(train_variation,axis=0) #normalizing

test_variation=cnt_vect.transform(test_variation)
test_variation=normalize(test_variation,axis=0)

cv_variation=cnt_vect.transform(cv_variation)
cv_variation=normalize(cv_variation,axis=0)
```

In [120]:

```
train_gene=cnt_vect.fit_transform(train_gene)
train_gene=normalize(train_gene,axis=0)

test_gene=cnt_vect.transform(test_gene)
test_gene=normalize(test_gene,axis=0)

cv_gene=cnt_vect.transform(cv_gene)
cv_gene=normalize(cv_gene,axis=0)
```

In [121]:

```
cnt_vect_text=CountVectorizer(min_df=10,ngram_range=(1,2),max_features = 2000)

train_text=cnt_vect_text.fit_transform(train_text)
train_text=normalize(train_text,axis=0)

test_text=cnt_vect_text.transform(test_text)
test_text=normalize(test_text,axis=0)

cv_text=cnt_vect_text.transform(cv_text)
cv_text=normalize(cv_text,axis=0)
```

In [122]:

```
print(train_variation.shape)
print(train_gene.shape)
print(train_text.shape)
```

```
(2124, 2000)
(2124, 230)
(2124, 2000)
```

In [123]:

```
from scipy.sparse import hstack

train_data=hstack([train_variation,train_gene,train_text,word_standardized_train]).tocsr()
cv_data=hstack([cv_variation,cv_gene,cv_text,word_standardized_cv]).tocsr()
test_data=hstack([test_variation,test_gene,test_text,word_standardized_test]).tocsr()
```

In [124]:

```
print(train_data.shape)
print("*"*50)
print(cv_data.shape)
print("*"*50)
print(test_data.shape)
```

```
(2124, 4231)
**************************************************
(532, 4231)
**************************************************
(665, 4231)
```

In [125]:

```
print(train_y.shape)
print("*"*50)
print(cv_y.shape)
print("*"*50)
print(test_y.shape)
```

```
(2124,)
**************************************************
(532,)
**************************************************
(665,)
```

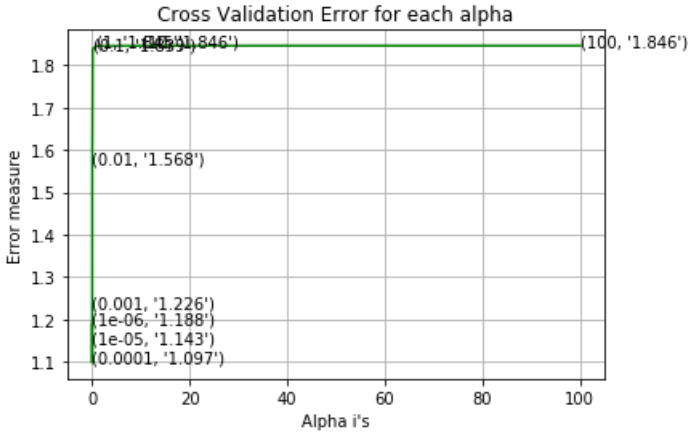
Hyperparameter(alpha) Tuning

In [126]:

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

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

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


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

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

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


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

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

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

```
for alpha = 1e-06
Log Loss : 1.1879000714151453
for alpha = 1e-05
Log Loss : 1.142753065327796
for alpha = 0.0001
Log Loss : 1.097123943358407
for alpha = 0.001
Log Loss : 1.225778409873668
for alpha = 0.01
Log Loss : 1.5677892523779993
for alpha = 0.1
Log Loss : 1.838607427632607
for alpha = 1
Log Loss : 1.845433633792432
for alpha = 10
Log Loss : 1.8458910675892837
for alpha = 100
Log Loss : 1.8459419202386218
```




```
For values of best alpha =  0.0001 The train log loss is: 0.38865522337074065
For values of best alpha =  0.0001 The cross validation log loss is: 1.097123943358407
For values of best alpha =  0.0001 The test log loss is: 1.0310054036601795
```

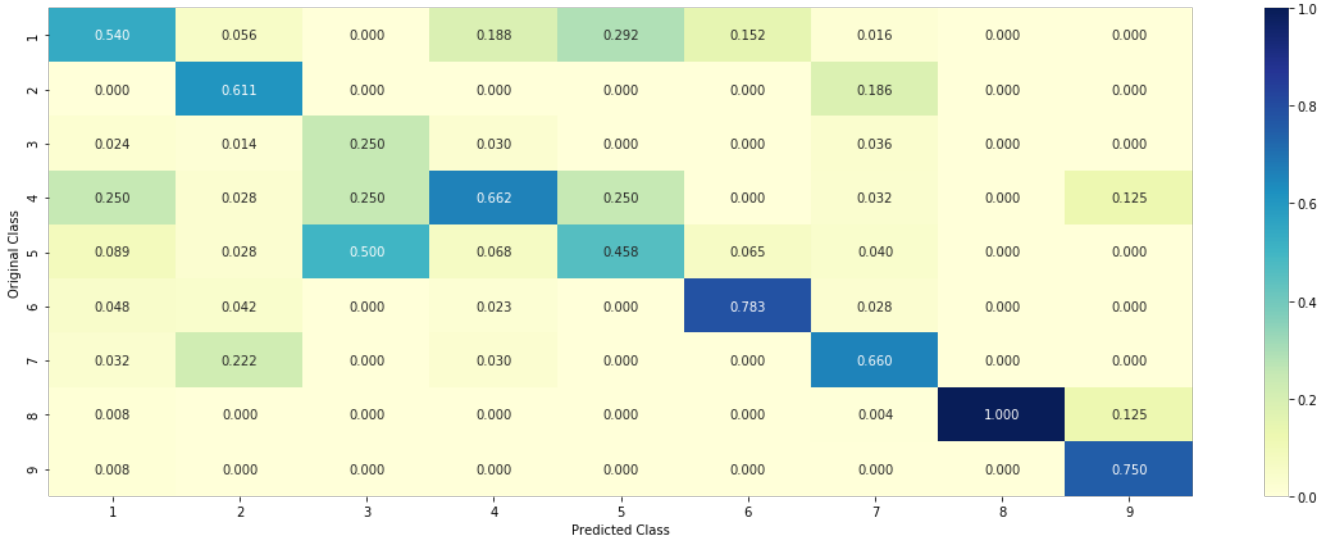
Testing on test data

In [127]:

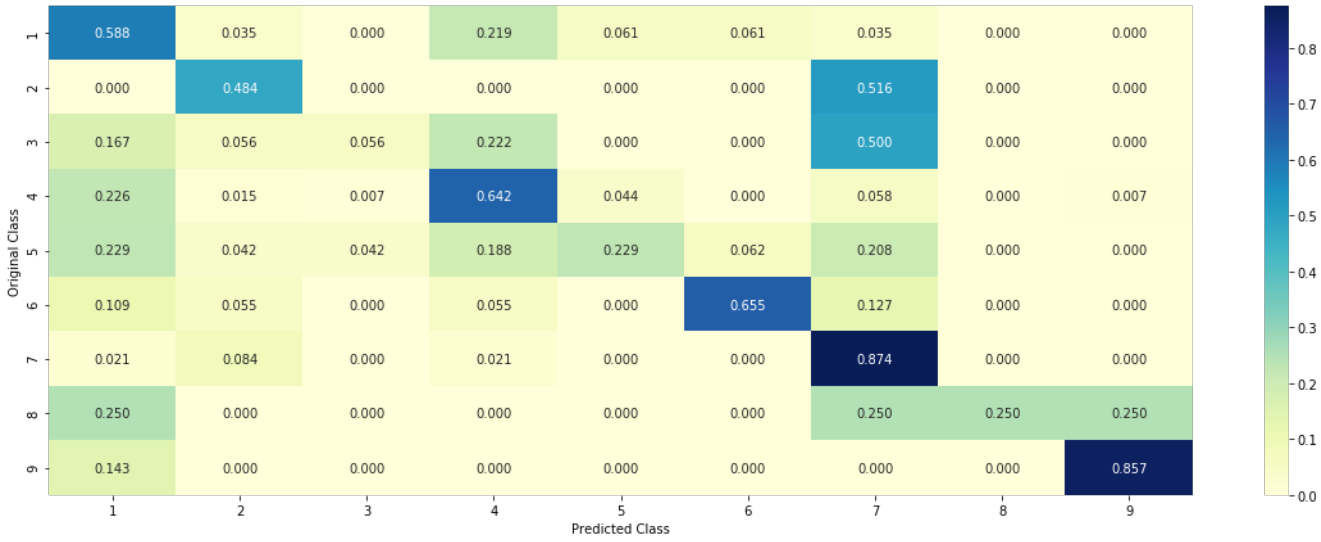
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,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

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

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

```
Log loss : 1.0310054036601795
Number of mis-classified points : 0.3669172932330827
-------------------- Confusion matrix --------------------
```


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




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



Tabulating the Result

In [129]:

```
#Refer->http://zetcode.com/python/prettytable/
#Refer->https://het.as.utexas.edu/HET/Software/Numpy/reference/generated/numpy.percentile.html
#Refer->https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.round_.html
from prettytable import PrettyTable
x=PrettyTable()

x.field_names=["Model","Best-Parameter","Log-Loss","%Misclaasified"] #column headers

x.add_row(["Naive Bayes","0.0001", "1.20","36"])
x.add_row(["K-NN","31", "0.99","35"])
x.add_row(["Logistic Regression(class_Balancing)","0.0001", "0.91","32"])
x.add_row(["Logistic Regression(without class_Balancing)","0.0001", "0.92","32"])
x.add_row(["SVM","0.0001","0.95", "32"])
x.add_row(["Random Forest(One-Hot Encoding)","2000","1.13", "39"])
x.add_row(["Random Forest(Response Encoding)","1000","1.33", "49"])
x.add_row(["Stacking Classifer(LR+SVM+NB)","0.01","1.08", "32"])
x.add_row(["Voting Classifer","soft-voting","0.95", "32"])
x.add_row(["Logistic Regression(Bag of Words)","0.0001", "1.05","36"])

print(x)
```

```
+-----------------------------------------------+----------------+----------+----------------+
|                     Model                     | Best-Parameter | Log-Loss | %Misclaasified |
+-----------------------------------------------+----------------+----------+----------------+
|                 Naive Bayes                   |     0.0001     |   1.20   |       36       |
|                     K-NN                      |       31       |   0.99   |       35       |
|     Logistic Regression(class_Balancing)      |     0.0001     |   0.91   |       32       |
| Logistic Regression(without class_Balancing)  |     0.0001     |   0.92   |       32       |
|                     SVM                       |     0.0001     |   0.95   |       32       |
|       Random Forest(One-Hot Encoding)         |      2000      |   1.13   |       39       |
|       Random Forest(Response Encoding)        |      1000      |   1.33   |       49       |
|        Stacking Classifer(LR+SVM+NB)          |      0.01      |   1.08   |       32       |
|               Voting Classifer                |  soft-voting   |   0.95   |       32       |
|      Logistic Regression(Bag of Words)        |     0.0001     |   1.05   |       36       |
+-----------------------------------------------+----------------+----------+----------------+
```

Certainly we can see that **LR ,SVM** performs better overall in terms of log-loss as well as in terms of percentage of mis-classified points

Procedure

First and foremost there is a brief description about the probem and then we defined the objective and constraints and also defined the problem and KPI for the given problem

Basic libraries are imported and reading and preprocessing of text data which involves cleaning of text.

Feature engineering where we have created another feature from the features that are already present.

Checking the distribution of the class label

Split the data into train/cv/test

Creating a random model which is used to evaluate our featuress.

Univariate analysis where we checked their dependeny on predicting y and also their stability

Data Preparaton for model building, function creation for feature importance which will help in interpretibility and staking of features into one

Applied different models , fine tuned their hyperparameters , plotted their confusion , precision and recall matrix

Tabulated their results

In [ ]: