Personalized Cancer diagnosis using Genetic Data (TFIDF Vectorization)

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [1]:

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

import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")
from mlxtend.classifier import StackingClassifier
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:

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

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

Out[2]:

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

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

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

3.1.2. Reading Text Data

In [3]:

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

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

Out[3]:

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

3.1.3. Preprocessing of text

In [4]:

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


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

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

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

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

```
There is no text description for id: 1109
There is no text description for id: 1277
There is no text description for id: 1407
There is no text description for id: 1639
There is no text description for id: 2755
Time took for preprocessing the text : 223.269307 seconds
```

```
In [6]:
#Merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[6]:

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

```
In [7]:
#Check for null entries
result[result.isnull().any(axis=1)]
```

Out[7]:

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

```
In [8]:
#Replace the null entries in Text by Gene + Variation Data
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

```
In [9]:
#Display row after null replacement
result[result['ID']==1109]
```

Out[9]:

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

```
In [10]:
result.columns
```

Out[10]:

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

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

In [4]:

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

Out[4]:

	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...

3.1.4. Test, Train and Cross Validation Split

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

In [5]:

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

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

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

In [6]:

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

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

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

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

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

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


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

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

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

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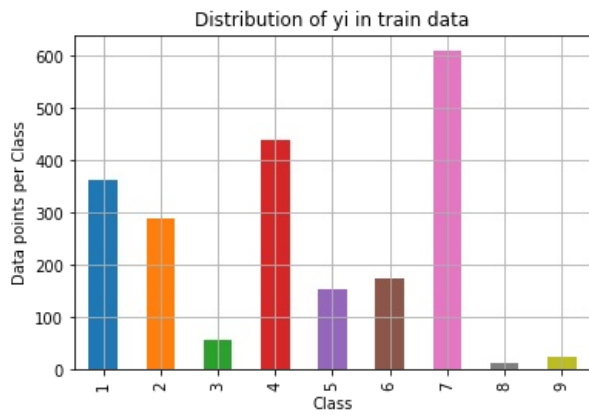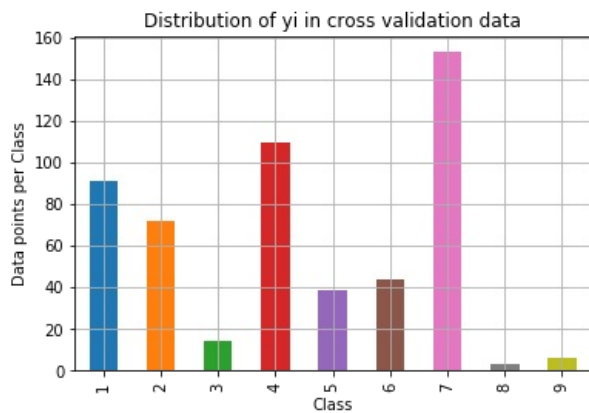
Distribution of yi in train data

```
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 %)
```

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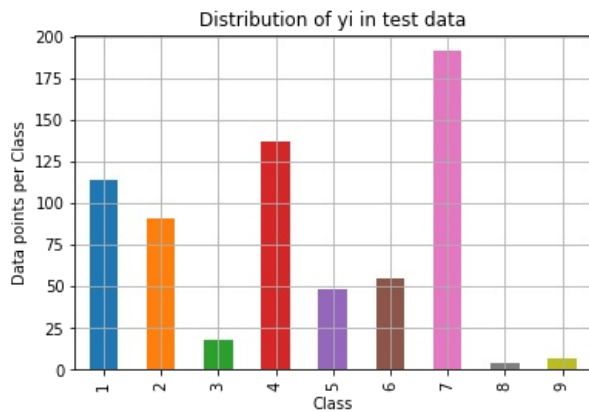
```
Number of data points in class 7 : 191 ( 28.722 %)
Number of data points in class 4 : 137 ( 20.602 %)
Number of data points in class 1 : 114 ( 17.143 %)
Number of data points in class 2 : 91 ( 13.684 %)
Number of data points in class 6 : 55 ( 8.271 %)
Number of data points in class 5 : 48 ( 7.218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

Distribution of yi in cross validation data

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

Key Take-away:

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

3.2 Prediction using a 'Random' Model

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

In [8]:

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

    A =(((C.T)/(C.sum(axis=1))).T)
    #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]]

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

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

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

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

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


# 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)
```
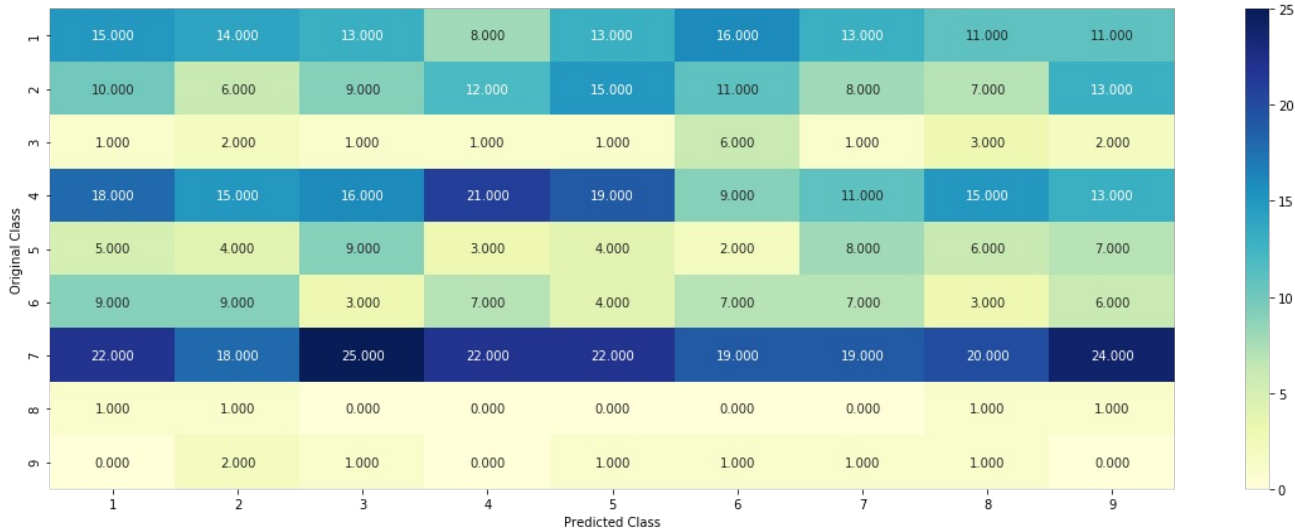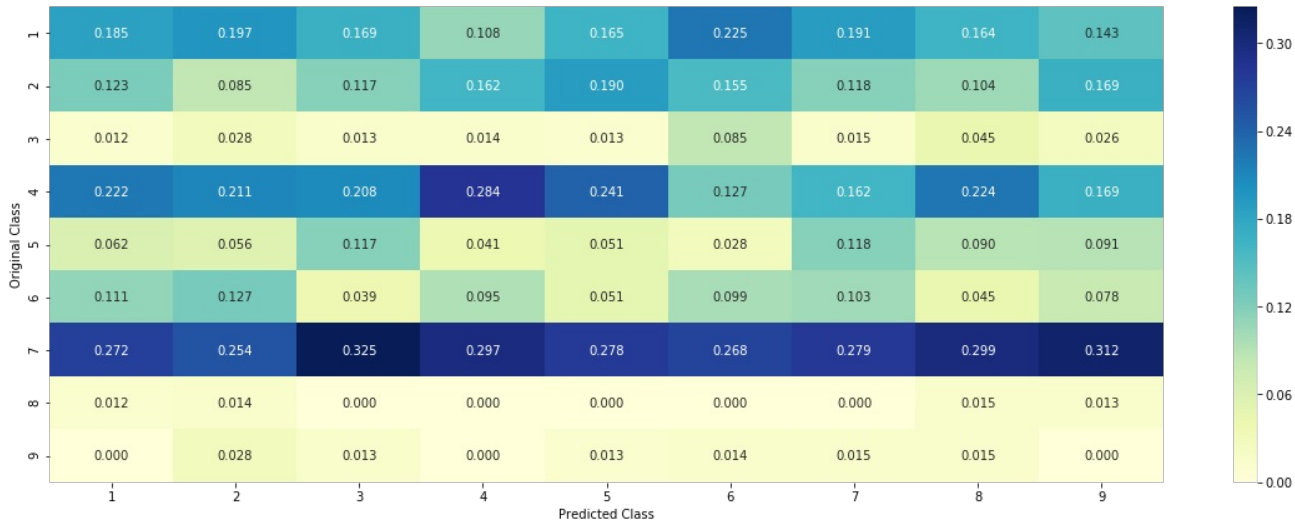
Log loss on Cross Validation Data using Random Model 2.5334072064139477

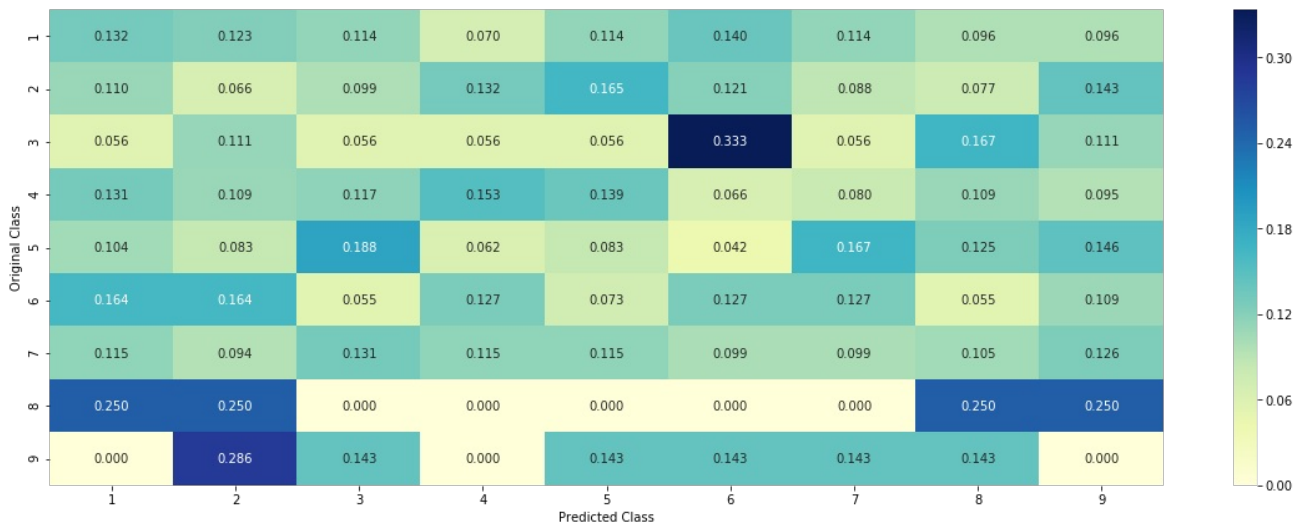Log loss on Test Data using Random Model 2.4868169912081424

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

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

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

3.3 Univariate Analysis

In [10]:

```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
```

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

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

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

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

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

            # cls_cnt.shape[0](numerator) will contain the number of time that particular feature occured in the
dataset of class 1(Let)
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177, 0.13636363636363635, 0.25, 0
.19318181818181818, 0.03787878787878788, 0.03787878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.06122448979591836, 0.061224489795918366, 0.27040816326530615, 0.06122
4489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177, 0.068181818181818177
, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
```

```python
    #         'BRCA2': [0.1333333333333333, 0.060606060606060608, 0.060606060606060608, 0.078787878787878782, 0.139
3939393939394, 0.34545454545454546, 0.060606060606060608, 0.060606060606060608, 0.060606060606060608],
    #         'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.46540880503144655, 0.0754
71698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081761006289, 0.062893081761006289],
    #         'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.072847682119205295, 0.06622
5165562913912, 0.066225165562913912, 0.27152317880794702, 0.066225165562913912, 0.066225165562913912],
    #         'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.073333333333333334, 0.0933
33333333333338, 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 data
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there in the train data then we wil
l add the feature to gv_fea
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_fea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#             gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea
```

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

In [11]:

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

```
Number of Unique Genes in train data: 232
BRCA1      165
TP53        99
EGFR        92
PTEN        79
BRCA2       75
KIT         65
BRAF        62
ERBB2       42
PDGFRA      40
ALK         38
Name: Gene, dtype: int64
```

In [20]:

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

```
Ans: There are 233 different categories of genes in the training data, and they are distibuted as fo
llows
```

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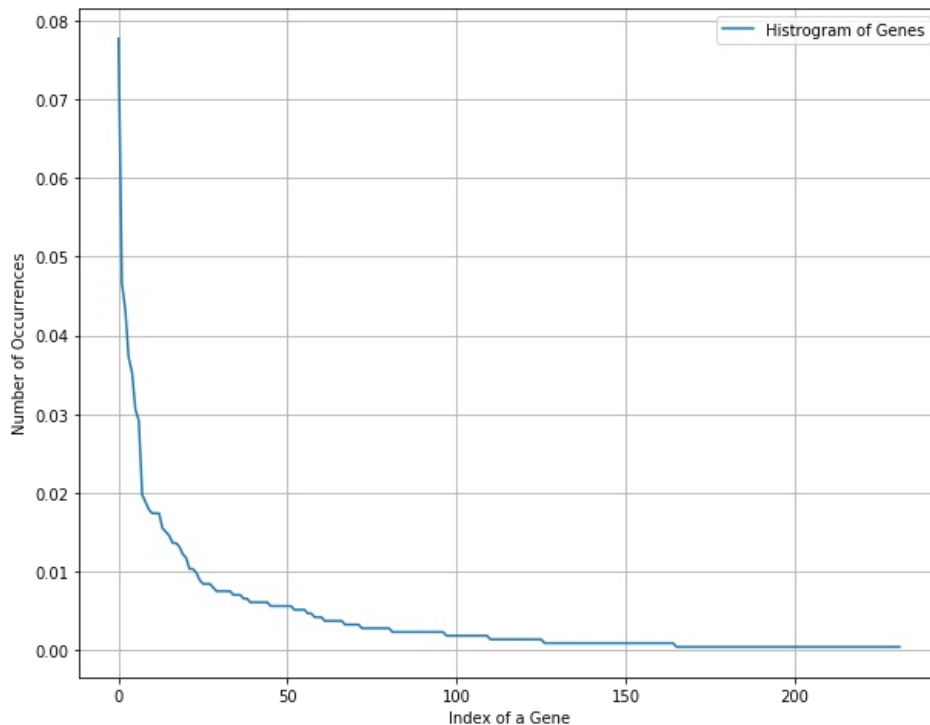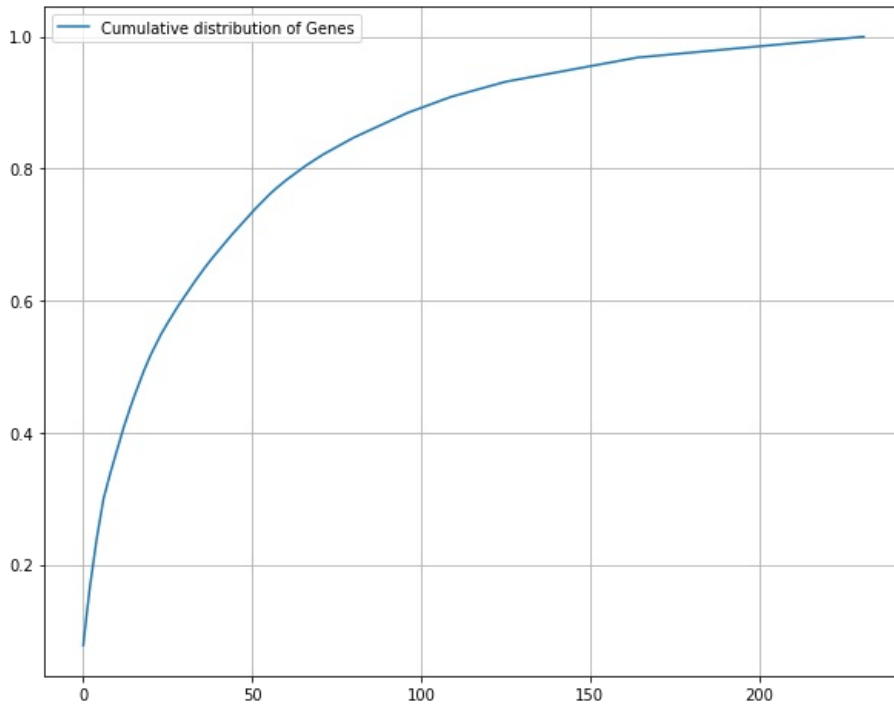

Key Take-away from the above Histogram:

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

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


Key Take-away from the above CDF Plot:

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

Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

In [15]:

```
#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))
```

```
print("train_gene_feature_responseCoding is converted to a feature vector using respone coding method. The shape
of gene feature:", train_gene_feature_responseCoding.shape)
print("test_gene_feature_responseCoding is converted to a feature vector using respone coding method. The shape o
f gene feature:", test_gene_feature_responseCoding.shape)
print("cv_gene_feature_responseCoding is converted to a feature vector using respone coding method. The shape of
gene feature:", cv_gene_feature_responseCoding.shape)
```

```
train_gene_feature_responseCoding is converted to a feature vector using respone coding method. The
shape of gene feature: (2124, 9)
test_gene_feature_responseCoding is converted to a feature vector using respone coding method. The s
hape of gene feature: (665, 9)
cv_gene_feature_responseCoding is converted to a feature vector using respone coding method. The sha
pe of gene feature: (532, 9)
```

```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer()
train_gene_feature_tfidf = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_tfidf = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_tfidf = gene_vectorizer.transform(cv_df['Gene'])

print("train_gene_feature_tfidf is converted to a feature vector using one-hot coding method. The shape of gene f
eature:", train_gene_feature_tfidf.shape)
print("test_gene_feature_tfidf is converted to a feature vector using one-hot coding method. The shape of gene fe
ature:", test_gene_feature_tfidf.shape)
print("cv_gene_feature_tfidf is converted to a feature vector using one-hot coding method. The shape of gene feat
ure:", cv_gene_feature_tfidf.shape)
```

```
train_gene_feature_tfidf is converted to a feature vector using one-hot coding method. The shape of
gene feature: (2124, 232)
test_gene_feature_tfidf is converted to a feature vector using one-hot coding method. The shape of g
ene feature: (665, 232)
cv_gene_feature_tfidf is converted to a feature vector using one-hot coding method. The shape of gen
e feature: (532, 232)
```

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.

```python
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.SGDCla
ssifier.html
# -----------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=N
one,
# 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_tfidf, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_tfidf, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_tfidf)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-1
5))

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.364359668902223
For values of alpha =  0.0001 The log loss is: 1.1785575907135069
For values of alpha =  0.001 The log loss is: 1.1951214838436675
For values of alpha =  0.01 The log loss is: 1.3200565350968942
For values of alpha =  0.1 The log loss is: 1.4398435332613628
For values of alpha =  1 The log loss is: 1.4858807734741992
```


Cross Validation Error for each alpha
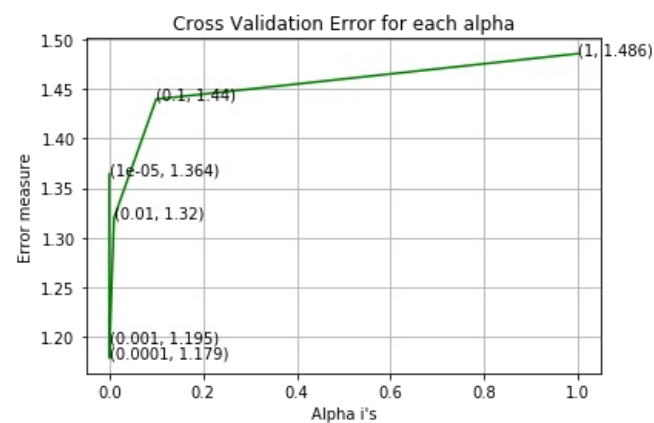
```
For values of best alpha =  0.0001 The train log loss is: 1.0494983135520082
For values of best alpha =  0.0001 The cross validation log loss is: 1.1785575907135069
For values of best alpha =  0.0001 The test log loss is: 1.1985886468068552
```

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

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

In [20]:

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

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

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

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

Ans.
1. In test data 643 out of 665 : 96.69172932330827
2. In cross validation data 513 out of  532 : 96.42857142857143
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

In [21]:

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

```
Number of Unique Variations : 1945
Truncating_Mutations    61
Deletion                44
Amplification           40
Fusions                 16
Overexpression           5
E17K                     3
P34R                     2
G13V                     2
Q22K                     2
V321M                    2
Name: Variation, dtype: int64
```

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

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

```
#Histogram for the distribution of the fetaure Variation
plt.figure(figsize=(10,8))
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```
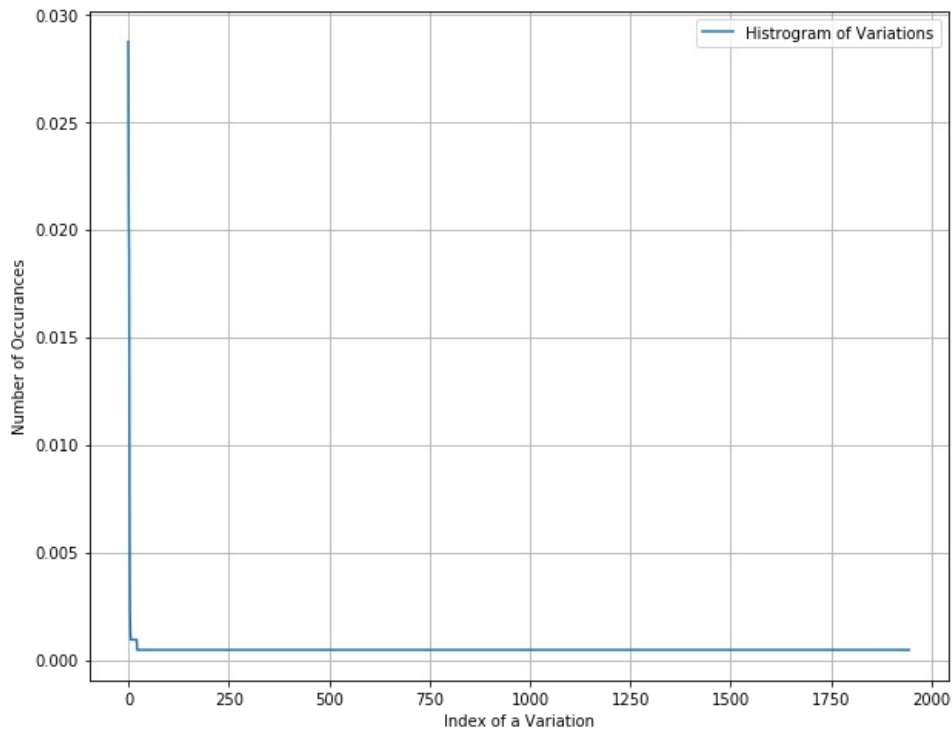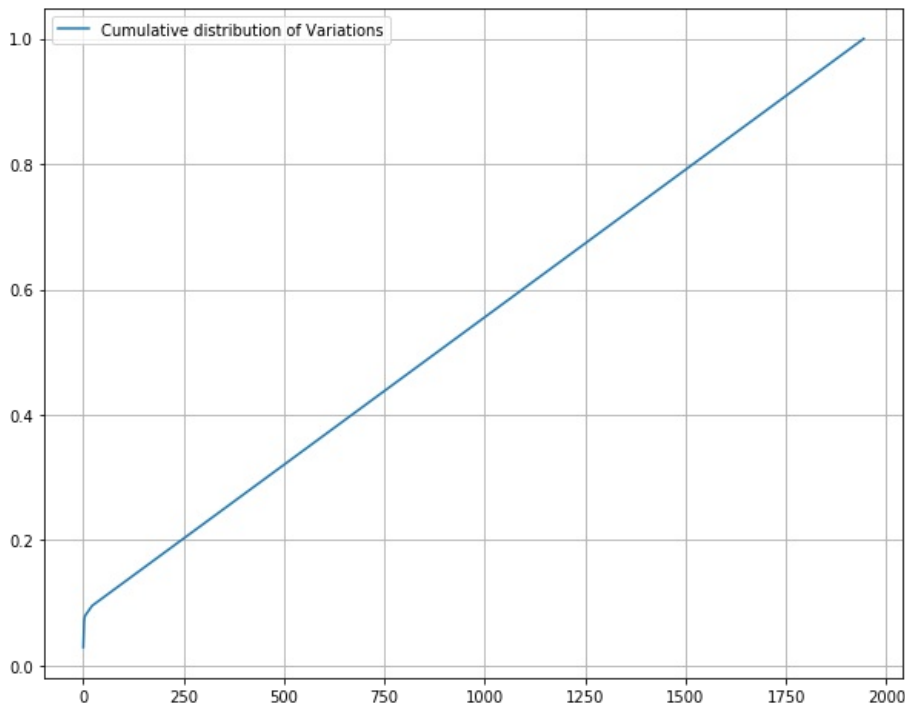

Key Take-aways:

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

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

[0.0287194 0.04943503 0.06826742 ... 0.99905838 0.99952919 1.   ]

Key Take-aways:

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

Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

Very Important note about response coding: Avoiding response leakage.

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

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

```python
# alpha is used for laplace smoothing (Naive Bayes)
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))

print("train_variation_feature_responseCoding is a converted feature using the response coding method. The shape
of Variation feature:", train_variation_feature_responseCoding.shape)
print("test_variation_feature_responseCoding is a converted feature using the response coding method. The shape o
f Variation feature:", test_variation_feature_responseCoding.shape)
print("cv_variation_feature_responseCoding is a converted feature using the response coding method. The shape of
Variation feature:", cv_variation_feature_responseCoding.shape)
```

train_variation_feature_responseCoding is a converted feature using the response coding method. The
shape of Variation feature: (2124, 9)
test_variation_feature_responseCoding is a converted feature using the response coding method. The s
hape of Variation feature: (665, 9)
cv_variation_feature_responseCoding is a converted feature using the response coding method. The sha
pe of Variation feature: (532, 9)

```python
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_tfidf = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_tfidf = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_tfidf = variation_vectorizer.transform(cv_df['Variation'])

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

train_variation_feature_tfidf is converted feature using the one-hot encoding method. The shape of V
ariation feature: (2124, 1975)
test_variation_feature_tfidf is converted feature using the one-hot encoding method. The shape of Va
riation feature: (665, 1975)
cv_variation_feature_tfidf is converted feature using the one-hot encoding method. The shape of Vari
ation feature: (532, 1975)

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

Let's build a model just like the earlier!

```python
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_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_variation_feature_tfidf, y_train)
    predict_y = sig_clf.predict_proba(cv_variation_feature_tfidf)

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

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_tfidf, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_variation_feature_tfidf, y_train)

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

```
For values of alpha =  1e-05 The log loss is: 1.7083483840630693
For values of alpha =  0.0001 The log loss is: 1.703173718148098
For values of alpha =  0.001 The log loss is: 1.705338890835227
For values of alpha =  0.01 The log loss is: 1.713543492328711
For values of alpha =  0.1 The log loss is: 1.7265172782771525
For values of alpha =  1 The log loss is: 1.7294561836480158
```


Cross Validation Error for each alpha
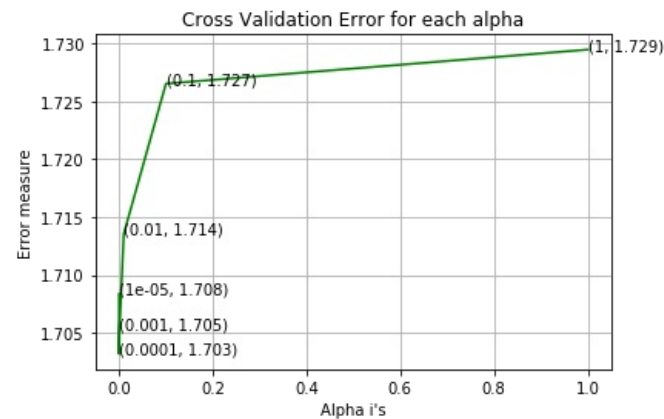
```
For values of best alpha =  0.0001 The train log loss is: 0.7493164530973311
For values of best alpha =  0.0001 The cross validation log loss is: 1.703173718148098
For values of best alpha =  0.0001 The test log loss is: 1.687683481933988
```

Conclusion:

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

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

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

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

In [30]:

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

```
Q12. How many data points are covered by total  1945  variations in test and cross validation data s
ets?

Ans.
1. In test data 78 out of 665 : 11.729323308270677
2. In cross validation data 60 out of  532 : 11.278195488721805
```

3.2.3 Univariate Analysis on Text Feature

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

In [32]:

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

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

In [33]:

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

In [34]:

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

#Copy the train_text_feature_tfidf into a seperate variable
train_text_feature_tfidf_ones = train_text_feature_tfidf.copy()
train_text_feature_tfidf_ones.data = train_text_feature_tfidf_ones.data / train_text_feature_tfidf_ones.data

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

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

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

print("Total number of unique words in train data :", len(train_text_features))
print("\nFrequency of occurence of each of the top 1000 words:")
print(text_fea_dict)
```

Total number of unique words in train data : 1000

Frequency of occurence of each of the top 1000 words:
{'000': 1178, '001': 691, '01': 820, '05': 1041, '10': 2067, '100': 1715, '11': 1961, '12': 2008, '13': 1889, '14': 1975, '15': 1982, '16': 1895, '17': 1912, '18': 1886, '19': 1856, '1997': 543, '1998': 538, '1999': 559, '1a': 1473, '1b': 1448, '20': 2009, '200': 1143, '2000': 914, '2001': 619, '2002': 642, '2003': 586, '2004': 600, '2005': 666, '2006': 634, '2007': 551, '2008': 596, '2009': 520, '2010': 569, '2011': 587, '2012': 477, '2013': 423, '21': 1825, '22': 1830, '23': 1801, '24': 1866, '25': 1865, '26': 1710, '27': 1687, '28': 1690, '29': 1699, '2a': 1498, '2b': 1382, '30': 1873, '31': 1617, '32': 1615, '33': 1575, '34': 1574, '35': 1631, '36': 1442, '37': 1555, '38': 1319, '39': 1370, '3a': 1409, '3b': 1271, '3t3': 364, '40': 1755, '41': 1344, '42': 1376, '43': 1323, '44': 1253, '45': 1383, '46': 1133, '47': 1082, '48': 1455, '49': 1037, '4a': 1219, '50': 1859, '51': 965, '52': 1013, '54': 941, '56': 854, '57': 831, '59': 775, '60': 1450, '61': 852, '70': 1178, '72': 1070, '75': 1033, '80': 1374, '90': 1179, '95': 1099, 'aberrant': 802, 'ability': 1513, 'abl': 407, 'able': 1247, 'absence': 1582, 'according': 1633, 'acid': 1814, 'acids': 1396, 'acquired': 817, 'across': 923, 'activate': 1074, 'activated': 1309, 'activating': 1305, 'activation': 1795, 'active': 1382, 'activities': 1016, 'activity': 1983, 'added': 1090, 'addition': 1957, 'additional': 1852, 'adenocarcinoma': 518, 'advanced': 702, 'affect': 1559, 'affected': 1375, 'affecting': 946, 'affinity': 967, 'age': 939, 'akt': 753, 'akt1': 235, 'al': 1520, 'alk': 253, 'allele': 1211, 'alleles': 1005, 'alone': 1185, 'also': 2102, 'alterations': 1350, 'altered': 1168, 'alternative': 1006, 'although': 1998, 'american': 741, 'amino': 1841, 'aml': 263, 'among': 1542, 'amplification': 1155, 'amplified': 1124, 'analyses': 1514, 'analysis': 2067, 'analyzed': 1778, 'another': 1541, 'anti': 1399, 'antibodies': 1340, 'antibody': 1416, 'apoptosis': 916, 'applied': 1189, 'approach': 1135, 'approximately': 1389, 'ar': 76, 'assay': 1785, 'assays': 1668, 'assess': 1233, 'assessed': 1398, 'assessment': 862, 'associated': 1937, 'association': 1329, 'atp': 732, 'author': 503, 'available': 1566, 'average': 1030, 'ba': 311, 'bard1': 178, 'basal': 787, 'based': 1857, 'basis': 1238, 'bcr': 399, 'bind': 1099, 'binding': 19

18, 'binds': 950, 'biochemical': 961, 'biological': 1072, 'blood': 939, 'blot': 1018, 'blotting': 81
0, 'blue': 1103, 'bone': 548, 'bound': 990, 'box': 440, 'bp': 715, 'braf': 511, 'brca': 182, 'brca1'
: 443, 'brca2': 355, 'brct': 218, 'breast': 1122, 'buffer': 1103, 'ca': 1068, 'calculated': 1149, 'c
ancer': 1953, 'cancers': 1547, 'cannot': 1082, 'carcinoma': 1019, 'carcinomas': 732, 'carried': 1143
, 'carrying': 974, 'case': 1527, 'cases': 1686, 'catalytic': 888, 'catenin': 181, 'cause': 1313, 'ca
used': 1200, 'cbl': 119, 'cdk12': 14, 'cdk4': 207, 'cdna': 1259, 'cell': 2044, 'cells': 2065, 'cellu
lar': 1438, 'chain': 1092, 'change': 1456, 'changes': 1574, 'characteristics': 867, 'characterized':
1456, 'chemotherapy': 574, 'chromosome': 997, 'ci': 415, 'class': 897, 'classification': 704, 'class
ified': 920, 'clear': 1224, 'clinical': 1728, 'clinically': 928, 'clones': 896, 'cluster': 747, 'co'
: 1262, 'coding': 1186, 'codon': 955, 'cohort': 771, 'colonies': 608, 'colony': 632, 'colorectal': 5
87, 'combination': 1083, 'combined': 1066, 'common': 1674, 'compared': 1954, 'comparison': 1337, 'co
mplete': 1383, 'complex': 1518, 'complexes': 776, 'concentration': 1045, 'concentrations': 1078, 'co
nditions': 1393, 'confer': 1014, 'confirmed': 1593, 'conformation': 997, 'conserved': 1383, 'conside
red': 1323, 'consistent': 1640, 'constitutive': 994, 'constitutively': 812, 'construct': 944, 'const
ructs': 1199, 'contact': 776, 'containing': 1794, 'contains': 1089, 'context': 1064, 'contrast': 176
6, 'contribute': 1350, 'control': 1838, 'controls': 1260, 'copy': 761, 'core': 946, 'correlated': 93
5, 'correlation': 1084, 'corresponding': 1478, 'could': 1890, 'criteria': 844, 'critical': 1451, 'cr
izotinib': 130, 'crystal': 967, 'culture': 1267, 'cultured': 1026, 'current': 1000, 'cycle': 1107, '
cyclin': 444, 'd1': 296, 'damage': 649, 'data': 2060, 'database': 1042, 'day': 862, 'days': 1078, 'd
e': 790, 'death': 712, 'decrease': 977, 'decreased': 1200, 'defective': 751, 'defects': 696, 'defici
ent': 718, 'defined': 1228, 'degradation': 657, 'deleterious': 579, 'deletion': 1310, 'deletions': 1
007, 'demonstrate': 1162, 'demonstrated': 1361, 'dependent': 1649, 'derived': 1575, 'described': 199
7, 'despite': 1165, 'detected': 1754, 'detection': 1155, 'determine': 1567, 'determined': 1740, 'dev
eloped': 1288, 'development': 1695, 'diagnosis': 955, 'difference': 1337, 'differences': 1437, 'diff
erent': 1942, 'differentiation': 816, 'dimerization': 521, 'direct': 1377, 'directed': 1307, 'direct
ly': 1523, 'discovery': 863, 'discussion': 1932, 'disease': 1678, 'displayed': 1057, 'distinct': 148
3, 'distribution': 976, 'dna': 1945, 'domain': 1973, 'domains': 1574, 'dominant': 921, 'dose': 754,
'double': 938, 'download': 691, 'downstream': 1410, 'driven': 800, 'driver': 584, 'drug': 966, 'due'
: 1680, 'e2': 146, 'early': 1190, 'effect': 1851, 'effective': 950, 'effects': 1748, 'efficacy': 625
, 'egf': 297, 'egfr': 592, 'eight': 1136, 'either': 1906, 'elevated': 851, 'embryonic': 702, 'encodi
ng': 1295, 'end': 1160, 'endogenous': 1053, 'enhanced': 1122, 'enzyme': 796, 'epithelial': 630, 'er'
: 337, 'erbb2': 276, 'erk': 539, 'erlotinib': 216, 'essential': 1163, 'established': 1253, 'et': 152
8, 'evaluated': 1184, 'even': 1452, 'events': 1036, 'evidence': 1606, 'examined': 1527, 'example': 1
351, 'exhibited': 948, 'exon': 1252, 'exons': 1148, 'expected': 1466, 'experimental': 1138, 'experim
ents': 1643, 'express': 1033, 'expressed': 1822, 'expressing': 1538, 'expression': 2021, 'extracellu
lar': 762, 'f3': 380, 'fact': 1351, 'factor': 1508, 'factors': 1341, 'families': 687, 'family': 1539
, 'features': 1126, 'fgfr2': 111, 'fgfr3': 120, 'fig': 1677, 'figure': 1885, 'figures': 804, 'finall
y': 1221, 'finding': 1188, 'findings': 1522, 'first': 1842, 'fish': 399, 'five': 1487, 'flag': 506,
'flt3': 251, 'fold': 1355, 'followed': 1404, 'following': 1618, 'form': 1457, 'formation': 1321, 'fo
rms': 1227, 'found': 2019, 'four': 1736, 'fraction': 895, 'frame': 978, 'free': 1222, 'frequency': 1
403, 'frequent': 1008, 'frequently': 1335, 'full': 1509, 'function': 1911, 'functional': 1822, 'func
tionally': 906, 'functions': 1301, 'furthermore': 1504, 'fusion': 864, 'fusions': 394, 'gain': 1078,
'gap': 330, 'gefitinib': 240, 'gel': 984, 'gene': 2010, 'general': 1073, 'generated': 1553, 'generat
ion': 1049, 'genes': 1754, 'genetic': 1634, 'genome': 1050, 'genomic': 1467, 'germline': 933, 'gfp':
539, 'gist': 120, 'gists': 102, 'given': 1453, 'go': 1082, 'grade': 525, 'greater': 1061, 'green': 1
137, 'group': 1548, 'groups': 1254, 'growth': 1719, 'gst': 303, 'gtp': 237, 'ha': 508, 'half': 1073,
'harboring': 939, 'helix': 807, 'her2': 269, 'heterozygous': 754, 'hif': 83, 'high': 1917, 'higher':
1719, 'highly': 1710, 'histone': 362, 'history': 727, 'hotspot': 387, 'hotspots': 276, 'hours': 1010
, 'however': 2068, 'hr': 436, 'http': 969, 'human': 1919, 'hydrogen': 531, 'hydrophobic': 665, 'hypo
thesis': 1091, 'ic50': 523, 'identification': 1288, 'identified': 1982, 'identify': 1496, 'ii': 1358
, 'iii': 905, 'il': 616, 'image': 779, 'imatinib': 318, 'impact': 1097, 'impaired': 742, 'important'
: 1821, 'inactivation': 895, 'inactive': 891, 'include': 1350, 'included': 1483, 'including': 1963,
'increase': 1459, 'increased': 1770, 'incubated': 1162, 'indeed': 1225, 'independent': 1802, 'indica
te': 1589, 'indicated': 1769, 'indicates': 1256, 'indicating': 1473, 'individual': 1412, 'individual
s': 924, 'induce': 1074, 'induced': 1520, 'induction': 806, 'information': 1226, 'inhibit': 1017, 'i
nhibited': 1001, 'inhibition': 1392, 'inhibitor': 1425, 'inhibitors': 1303, 'inhibitory': 859, 'init
ial': 1045, 'insertion': 666, 'insertions': 530, 'interact': 887, 'interaction': 1442, 'interactions
': 1233, 'interestingly': 1451, 'interface': 579, 'intermediate': 708, 'introduction': 1695, 'invest
igated': 1182, 'invitrogen': 930, 'involved': 1651, 'involving': 1069, 'isolated': 1169, 'itd': 80,
'jak2': 142, 'kd': 331, 'keap1': 58, 'key': 1278, 'kinase': 1640, 'kinases': 1065, 'kit': 1292, 'kno
ckdown': 431, 'known': 1896, 'kras': 482, 'l858r': 146, 'lack': 1305, 'lane': 473, 'large': 1628, 'l
arger': 1093, 'lead': 1312, 'leading': 1246, 'leads': 1072, 'least': 1652, 'led': 1104, 'left': 982,
'length': 1361, 'lesions': 671, 'less': 1659, 'leukemia': 767, 'level': 1635, 'levels': 1795, 'life'
: 882, 'ligand': 851, 'ligase': 486, 'like': 1405, 'likely': 1824, 'limited': 1219, 'line': 1744, 'l
ines': 1717, 'linked': 993, 'localization': 832, 'located': 1523, 'locus': 754, 'loop':
963, 'loss': 1718, 'low': 1693, 'lower': 1569, 'luciferase': 578, 'lung': 1020, 'lymphoma': 378, 'ly
sates': 1068, 'made': 1077, 'major': 1303, 'majority': 1449, 'malignant': 816, 'mammalian': 934, 'ma
nner': 1019, 'manufacturer': 1090, 'manuscript': 333, 'many': 1617, 'mapk': 556, 'materials': 1364,
'may': 2058, 'mean': 1174, 'measured': 1390, 'mechanism': 1436, 'mechanisms': 1471, 'median': 622, '
mediated': 1491, 'medium': 1178, 'mek': 352, 'mek1': 227, 'melanoma': 635, 'melanomas': 300, 'member
s': 1132, 'membrane': 941, 'met': 558, 'metastatic': 662, 'method': 1219, 'methods': 1831, 'methylat
ion': 338, 'mg': 956, 'mice': 1116, 'might': 1624, 'min': 943, 'missense': 1448, 'ml': 1339, 'mlh1':
104, 'mm': 1081, 'mmr': 99, 'model': 1613, 'models': 1200, 'molecular': 1738, 'molecule': 931, 'mole
cules': 911, 'months': 699, 'moreover': 1304, 'motif': 915, 'mouse': 1382, 'mrna': 1015, 'msh2': 131
, 'mtor': 375, 'multiple': 1715, 'mutagenesis': 1262, 'mutant': 1913, 'mutants': 1627, 'mutated': 15
73, 'mutation': 2046, 'mutational': 1092, 'mutations': 2082, 'myc': 607, 'myeloid': 535, 'na': 306,
'nature': 978, 'negative': 1601, 'neutral': 540, 'new': 1533, 'next': 1411, 'nf1': 178, 'ng': 961, '
nih': 717, 'nm': 964, 'non': 1529, 'none': 1097, 'normal': 1559, 'normalized': 889, 'notch': 97, 'no
tch1': 118, 'note': 1272, 'novel': 1346, 'nras': 339, 'nrf2': 47, 'nsclc': 362, 'nuclear': 959, 'nuc
leotide': 1265, 'null': 615, 'number': 1887, 'numbers': 986, 'observation': 1284, 'observations': 11
87, 'observed': 1983, 'obtained': 1744, 'occur': 1415, 'odds': 236, 'often': 1115, 'oncogene': 862,
'oncogenic': 1222, 'one': 2049, 'open': 960, 'org': 829, 'ovarian': 727, 'overall': 1494, 'overexpre
ssion': 962, 'p110': 172, 'p16': 123, 'p16ink4a': 115, 'p53': 706, 'p85': 135, 'pa': 403, 'page': 10

31, 'panel': 1191, 'part': 1409, 'partial': 922, 'particular': 1291, 'pathogenic': 678, 'pathway': 1418, 'pathways': 1347, 'patient': 1502, 'patients': 1773, 'pattern': 1042, 'pcr': 1595, 'pdgfr': 210, 'pdgfra': 201, 'peptide': 689, 'per': 1377, 'percentage': 914, 'performed': 1823, 'ph': 889, 'phase': 1047, 'phenotype': 1261, 'phosphatase': 700, 'phospho': 800, 'phosphorylated': 1162, 'phosphorylation': 1482, 'pi3k': 630, 'pik3ca': 348, 'plasmid': 1012, 'plates': 988, 'play': 993, 'pocket': 631, 'point': 1531, 'population': 1138, 'position': 1402, 'positions': 986, 'positive': 1618, 'possibility': 1275, 'possible': 1645, 'potential': 1667, 'powerpoint': 611, 'predicted': 1465, 'presence': 1863, 'present': 1822, 'presented': 1044, 'previous': 1756, 'previously': 2000, 'primary': 1549, 'primer': 868, 'primers': 1118, 'prior': 1040, 'process': 909, 'product': 955, 'products': 1082, 'progression': 1142, 'proliferation': 1408, 'promote': 859, 'promoter': 906, 'properties': 1013, 'proportion': 881, 'prostate': 505, 'protein': 2066, 'proteins': 1900, 'provide': 1632, 'provided': 1293, 'pten': 488, 'published': 1225, 'purified': 1119, 'pvhl': 35, 'raf': 392, 'range': 1374, 'rare': 1179, 'ras': 711, 'rate': 1178, 'rates': 876, 'rather': 1116, 'ratio': 1088, 'reaction': 1156, 'rearrangements': 579, 'recent': 1452, 'recently': 1683, 'receptor': 1265, 'receptors': 695, 'recognition': 576, 'recurrent': 779, 'red': 1152, 'reduced': 1564, 'reduction': 1078, 'ref': 801, 'region': 1746, 'regions': 1438, 'regulated': 986, 'regulation': 1214, 'regulatory': 1126, 'related': 1398, 'relative': 1481, 'relatively': 1217, 'relevant': 1147, 'remaining': 1125, 'remains': 1368, 'repair': 730, 'repeat': 605, 'repeats': 457, 'report': 1396, 'reported': 1976, 'reporter': 747, 'reports': 1112, 'represent': 1404, 'representative': 1012, 'require': 946, 'required': 1613, 'research': 1202, 'residue': 1408, 'residues': 1557, 'resistance': 973, 'resistant': 1001, 'respectively': 1844, 'response': 1553, 'responses': 828, 'result': 1772, 'resulted': 1322, 'resulting': 1682, 'results': 2101, 'ret': 185, 'revealed': 1618, 'reverse': 970, 'right': 965, 'ring': 623, 'risk': 893, 'rna': 1177, 'role': 1774, 'ros1': 88, 'rt': 691, 's1': 1109, 's2': 1004, 's3': 843, 'sample': 1400, 'samples': 1655, 'score': 647, 'screening': 978, 'sd': 635, 'sds': 987, 'second': 1619, 'secondary': 1168, 'see': 1374, 'seen': 1189, 'selected': 1433, 'selection': 1085, 'selective': 860, 'sensitive': 1309, 'sensitivity': 1255, 'sequence': 1847, 'sequenced': 1157, 'sequences': 1418, 'sequencing': 1777, 'series': 1122, 'serum': 1144, 'set': 1370, 'seven': 1124, 'several': 1888, 'show': 1723, 'showed': 1822, 'showing': 1361, 'shown': 2037, 'shows': 1279, 'side': 910, 'signal': 1364, 'signaling': 1525, 'signals': 919, 'significance': 1351, 'significant': 1777, 'significantly': 1669, 'similar': 1962, 'similarly': 1253, 'since': 1202, 'single': 1824, 'sirna': 322, 'site': 1878, 'sites': 1540, 'six': 1378, 'size': 1202, 'smad2': 83, 'smad3': 78, 'smad4': 116, 'small': 1784, 'smo': 24, 'somatic': 1290, 'specific': 1938, 'specifically': 1207, 'specificity': 1000, 'specimens': 688, 'spectrum': 910, 'splice': 744, 'splicing': 699, 'spop': 32, 'sporadic': 627, 'src': 334, 'stability': 985, 'stable': 1202, 'stage': 809, 'staining': 1012, 'standard': 1265, 'stat3': 212, 'state': 1053, 'statistical': 1008, 'status': 1247, 'stem': 539, 'still': 1251, 'stimulated': 672, 'stimulation': 690, 'strand': 753, 'strong': 1320, 'strongly': 1249, 'structural': 1322, 'structure': 1569, 'structures': 891, 'studied': 1179, 'studies': 1961, 'study': 1995, 'subjected': 1003, 'subsequent': 1031, 'subset': 1205, 'substitution': 1295, 'substitutions': 1232, 'substrate': 975, 'substrates': 498, 'subunit': 611, 'sufficient': 1138, 'suggest': 1775, 'suggested': 1428, 'suggesting': 1705, 'suggests': 1559, 'supplemental': 358, 'supplementary': 1090, 'support': 1293, 'suppression': 759, 'suppressor': 1109, 'surface': 912, 'survival': 1169, 'syndrome': 852, 'system': 1477, 'table': 1881, 'tables': 747, 'tagged': 861, 'taken': 1306, 'target': 1570, 'targeted': 1204, 'targeting': 1077, 'targets': 1231, 'terminal': 1547, 'terminus': 876, 'test': 1589, 'tested': 1643, 'testing': 890, 'tgf': 224, 'therapeutic': 1122, 'therapy': 1043, 'therefore': 1728, 'three': 1975, 'thus': 1865, 'time': 1482, 'times': 1114, 'tissue': 1315, 'tissues': 962, 'tki': 240, 'tkis': 236, 'together': 1553, 'top': 972, 'total': 1744, 'tp53': 460, 'transactivation': 491, 'transcription': 1339, 'transcriptional': 926, 'transduced': 480, 'transfected': 1461, 'transfection': 1261, 'transformation': 866, 'transformed': 850, 'transforming': 711, 'translocation': 603, 'treated': 1359, 'treatment': 1426, 'trials': 767, 'tsc1': 117, 'tsc2': 111, 'tumor': 1780, 'tumorigenesis': 882, 'tumors': 1660, 'tumour': 441, 'tumours': 311, 'two': 2053, 'type': 2041, 'types': 1393, 'tyrosine': 1156, 'ubiquitin': 489, 'unable': 823, 'unique': 1127, 'university': 971, 'unknown': 1260, 'upon': 1175, 'use': 1465, 'used': 2025, 'using': 2071, 'value': 1148, 'values': 1257, 'variant': 1135, 'variants': 1227, 'various': 1372, 'vector': 1555, 'version': 928, 'versus': 934, 'vhl': 83, 'via': 1140, 'view': 1068, 'vitro': 1728, 'vivo': 1329, 'volume': 849, 'vus': 162, 'weeks': 674, 'well': 2033, 'western': 1296, 'whereas': 1777, 'whether': 1805, 'whole': 1039, 'wild': 1928, 'within': 1865, 'without': 1573, 'would': 1688, 'wt': 1274, 'www': 912, 'years': 1034, 'yeast': 548, 'yet': 1231}

In [35]:

```python
#Top 50 feature names based on the idf score
#I have used the referrence from: https://buhrmann.github.io/tfidf-analysis.html
def get_top_features(tf_idf_obj, feat_names, top_n_features):
    # returns a dataframe with top n features :
    feat_index = np.argsort(tf_idf_obj.idf_)
    top_features = [(feat_names[i], tf_idf_obj.idf_[i]) for i in feat_index[:top_n_features]]
    df_topFeatures = pd.DataFrame(data=top_features, columns = ['Top Words','TF-IDF Value'])
    print(df_topFeatures)

#Top 50 most important features using TFIDF
get_top_features(text_vectorizer, train_text_features, top_n_features=50)
```

```
      Top Words  TF-IDF Value
0          also      1.010407
1       results      1.010883
2     mutations      1.019963
3         using      1.025257
4       however      1.026706
5      analysis      1.027190
6            10      1.027190
7       protein      1.027674
8         cells      1.028157
9          data      1.030581
10          may      1.031551
11          two      1.033983
12          one      1.035932
13     mutation      1.037396
14         cell      1.038374
15         type      1.039842
16        shown      1.041803
17         well      1.043768
18         used      1.047708
19   expression      1.049685
20        found      1.050674
21         gene      1.055140
22           20      1.055637
23           12      1.056135
24   previously      1.060125
25     although      1.061125
26    described      1.061625
27        study      1.062627
28     observed      1.068657
29     activity      1.068657
30           15      1.069161
31   identified      1.069161
32     reported      1.072191
33           14      1.072697
34        three      1.072697
35       domain      1.073710
36    including      1.078789
37      similar      1.079298
38           11      1.079807
39      studies      1.079807
40     addition      1.081848
41     compared      1.083382
42       cancer      1.083893
43          dna      1.087996
44    different      1.089539
45     specific      1.091599
46   associated      1.092115
47   discussion      1.094699
48         wild      1.096770
49        human      1.101447
```

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

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


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

```python
#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)
```

```python
# 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
```

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

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

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

```python
#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()))
```

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

```
Counter({1822: 4, 1202: 4, 1178: 4, 1122: 4, 1082: 4, 1078: 4, 1865: 3, 1777: 3, 1744: 3, 1728: 3, 1
618: 3, 1574: 3, 1559: 3, 1553: 3, 1383: 3, 1382: 3, 1374: 3, 1361: 3, 1351: 3, 1350: 3, 1265: 3, 12
53: 3, 1219: 3, 1200: 3, 1179: 3, 1162: 3, 1138: 3, 1090: 3, 1068: 3, 1012: 3, 986: 3, 978: 3, 955:
3, 946: 3, 939: 3, 862: 3, 754: 3, 747: 3, 691: 3, 579: 3, 539: 3, 83: 3, 2067: 2, 1983: 2, 1982: 2,
1975: 2, 1961: 2, 1824: 2, 1795: 2, 1754: 2, 1719: 2, 1715: 2, 1710: 2, 1695: 2, 1643: 2, 1640: 2, 1
617: 2, 1613: 2, 1589: 2, 1575: 2, 1573: 2, 1569: 2, 1555: 2, 1547: 2, 1527: 2, 1523: 2, 1520: 2, 14
83: 2, 1482: 2, 1473: 2, 1465: 2, 1456: 2, 1452: 2, 1451: 2, 1448: 2, 1442: 2, 1438: 2, 1418: 2, 140
9: 2, 1408: 2, 1404: 2, 1398: 2, 1396: 2, 1393: 2, 1377: 2, 1370: 2, 1364: 2, 1339: 2, 1337: 2, 1329
: 2, 1323: 2, 1322: 2, 1309: 2, 1305: 2, 1303: 2, 1295: 2, 1293: 2, 1288: 2, 1262: 2, 1261: 2, 1260:
2, 1247: 2, 1233: 2, 1231: 2, 1227: 2, 1225: 2, 1222: 2, 1189: 2, 1169: 2, 1168: 2, 1156: 2, 1155: 2
, 1148: 2, 1143: 2, 1135: 2, 1126: 2, 1124: 2, 1116: 2, 1109: 2, 1103: 2, 1099: 2, 1097: 2, 1092: 2,
1077: 2, 1074: 2, 1073: 2, 1072: 2, 1053: 2, 1045: 2, 1042: 2, 1033: 2, 1031: 2, 1019: 2, 1013: 2, 1
008: 2, 1001: 2, 1000: 2, 997: 2, 993: 2, 967: 2, 965: 2, 962: 2, 961: 2, 950: 2, 941: 2, 934: 2, 92
8: 2, 914: 2, 912: 2, 910: 2, 906: 2, 895: 2, 891: 2, 889: 2, 888: 2, 882: 2, 876: 2, 859: 2, 852: 2
, 851: 2, 816: 2, 800: 2, 779: 2, 776: 2, 767: 2, 753: 2, 732: 2, 727: 2, 711: 2, 702: 2, 699: 2, 66
6: 2, 635: 2, 630: 2, 611: 2, 587: 2, 548: 2, 399: 2, 362: 2, 311: 2, 276: 2, 240: 2, 236: 2, 178: 2
, 146: 2, 120: 2, 111: 2, 2102: 1, 2101: 1, 2082: 1, 2071: 1, 2068: 1, 2066: 1, 2065: 1, 2060: 1, 20
58: 1, 2053: 1, 2049: 1, 2046: 1, 2044: 1, 2041: 1, 2037: 1, 2033: 1, 2025: 1, 2021: 1, 2019: 1, 201
0: 1, 2009: 1, 2008: 1, 2000: 1, 1998: 1, 1997: 1, 1995: 1, 1976: 1, 1973: 1, 1963: 1, 1962: 1, 1957
: 1, 1954: 1, 1953: 1, 1945: 1, 1942: 1, 1938: 1, 1937: 1, 1932: 1, 1928: 1, 1919: 1, 1918: 1, 1917:
1, 1913: 1, 1912: 1, 1911: 1, 1906: 1, 1900: 1, 1896: 1, 1895: 1, 1890: 1, 1889: 1, 1888: 1, 1887: 1
, 1886: 1, 1885: 1, 1881: 1, 1878: 1, 1873: 1, 1866: 1, 1863: 1, 1859: 1, 1857: 1, 1856: 1, 1852: 1,
1851: 1, 1847: 1, 1844: 1, 1842: 1, 1841: 1, 1838: 1, 1831: 1, 1830: 1, 1825: 1, 1823: 1, 1821: 1, 1
814: 1, 1805: 1, 1802: 1, 1801: 1, 1794: 1, 1785: 1, 1784: 1, 1780: 1, 1778: 1, 1775: 1, 1774: 1, 17
73: 1, 1772: 1, 1770: 1, 1769: 1, 1766: 1, 1756: 1, 1755: 1, 1748: 1, 1746: 1, 1740: 1, 1738: 1, 173
6: 1, 1723: 1, 1718: 1, 1717: 1, 1705: 1, 1699: 1, 1693: 1, 1690: 1, 1688: 1, 1687: 1, 1686: 1, 1683
: 1, 1682: 1, 1680: 1, 1678: 1, 1677: 1, 1674: 1, 1669: 1, 1668: 1, 1667: 1, 1660: 1, 1659: 1, 1655:
1, 1652: 1, 1651: 1, 1649: 1, 1645: 1, 1635: 1, 1634: 1, 1633: 1, 1632: 1, 1631: 1, 1628: 1, 1627: 1
, 1624: 1, 1619: 1, 1615: 1, 1606: 1, 1601: 1, 1595: 1, 1593: 1, 1582: 1, 1570: 1, 1567: 1, 1566: 1,
1564: 1, 1557: 1, 1549: 1, 1548: 1, 1542: 1, 1541: 1, 1540: 1, 1539: 1, 1538: 1, 1533: 1, 1531: 1, 1
529: 1, 1528: 1, 1525: 1, 1522: 1, 1518: 1, 1514: 1, 1513: 1, 1509: 1, 1508: 1, 1504: 1, 1502: 1, 14
98: 1, 1496: 1, 1494: 1, 1491: 1, 1487: 1, 1481: 1, 1478: 1, 1477: 1, 1471: 1, 1467: 1, 1466: 1, 146
1: 1, 1459: 1, 1457: 1, 1455: 1, 1453: 1, 1450: 1, 1449: 1, 1437: 1, 1436: 1, 1433: 1, 1428: 1, 1426
: 1, 1425: 1, 1416: 1, 1415: 1, 1412: 1, 1411: 1, 1410: 1, 1405: 1, 1403: 1, 1402: 1, 1400: 1, 1399:
1, 1392: 1, 1390: 1, 1389: 1, 1378: 1, 1376: 1, 1375: 1, 1372: 1, 1368: 1, 1359: 1, 1358: 1, 1355: 1
, 1347: 1, 1346: 1, 1344: 1, 1341: 1, 1340: 1, 1335: 1, 1321: 1, 1320: 1, 1319: 1, 1315: 1, 1313: 1,
1312: 1, 1310: 1, 1307: 1, 1306: 1, 1304: 1, 1301: 1, 1296: 1, 1292: 1, 1291: 1, 1290: 1, 1284: 1, 1
279: 1, 1278: 1, 1275: 1, 1274: 1, 1272: 1, 1271: 1, 1267: 1, 1259: 1, 1257: 1, 1256: 1, 1255: 1, 12
54: 1, 1252: 1, 1251: 1, 1249: 1, 1246: 1, 1238: 1, 1232: 1, 1228: 1, 1226: 1, 1224: 1, 1221: 1, 121
7: 1, 1214: 1, 1211: 1, 1207: 1, 1205: 1, 1204: 1, 1199: 1, 1191: 1, 1190: 1, 1188: 1, 1187: 1, 1186
: 1, 1185: 1, 1184: 1, 1182: 1, 1177: 1, 1175: 1, 1174: 1, 1165: 1, 1163: 1, 1160: 1, 1157: 1, 1152:
1, 1149: 1, 1147: 1, 1144: 1, 1142: 1, 1140: 1, 1137: 1, 1136: 1, 1133: 1, 1132: 1, 1127: 1, 1125: 1
, 1119: 1, 1118: 1, 1115: 1, 1114: 1, 1112: 1, 1107: 1, 1104: 1, 1093: 1, 1091: 1, 1089: 1, 1088: 1,
1085: 1, 1084: 1, 1083: 1, 1081: 1, 1070: 1, 1069: 1, 1066: 1, 1065: 1, 1064: 1, 1061: 1, 1057: 1, 1
050: 1, 1049: 1, 1047: 1, 1044: 1, 1043: 1, 1041: 1, 1040: 1, 1039: 1, 1037: 1, 1036: 1, 1034: 1, 10
30: 1, 1026: 1, 1020: 1, 1018: 1, 1017: 1, 1016: 1, 1015: 1, 1014: 1, 1010: 1, 1007: 1, 1006: 1, 100
5: 1, 1004: 1, 1003: 1, 994: 1, 990: 1, 988: 1, 987: 1, 985: 1, 984: 1, 982: 1, 977: 1, 976: 1, 975:
1, 974: 1, 973: 1, 972: 1, 971: 1, 970: 1, 969: 1, 966: 1, 964: 1, 963: 1, 960: 1, 959: 1, 956: 1, 9
48: 1, 944: 1, 943: 1, 938: 1, 935: 1, 933: 1, 931: 1, 930: 1, 926: 1, 924: 1, 923: 1, 922: 1, 921:
1, 920: 1, 919: 1, 916: 1, 915: 1, 911: 1, 909: 1, 905: 1, 897: 1, 896: 1, 893: 1, 890: 1, 887: 1, 8
81: 1, 868: 1, 867: 1, 866: 1, 864: 1, 863: 1, 861: 1, 860: 1, 854: 1, 850: 1, 849: 1, 844: 1, 843:
1, 832: 1, 831: 1, 829: 1, 828: 1, 823: 1, 820: 1, 817: 1, 812: 1, 810: 1, 809: 1, 807: 1, 806: 1, 8
04: 1, 802: 1, 801: 1, 796: 1, 790: 1, 787: 1, 775: 1, 771: 1, 762: 1, 761: 1, 759: 1, 751: 1, 744:
1, 742: 1, 741: 1, 730: 1, 718: 1, 717: 1, 715: 1, 712: 1, 708: 1, 706: 1, 704: 1, 700: 1, 696: 1, 6
95: 1, 690: 1, 689: 1, 688: 1, 687: 1, 678: 1, 674: 1, 672: 1, 671: 1, 665: 1, 662: 1, 657: 1, 649:
1, 647: 1, 642: 1, 634: 1, 632: 1, 631: 1, 627: 1, 625: 1, 623: 1, 622: 1, 619: 1, 616: 1, 615: 1, 6
08: 1, 607: 1, 605: 1, 603: 1, 600: 1, 596: 1, 592: 1, 586: 1, 584: 1, 578: 1, 576: 1, 574: 1, 569:
1, 559: 1, 558: 1, 556: 1, 551: 1, 543: 1, 540: 1, 538: 1, 535: 1, 531: 1, 530: 1, 525: 1, 523: 1, 5
21: 1, 520: 1, 518: 1, 511: 1, 508: 1, 506: 1, 505: 1, 503: 1, 498: 1, 491: 1, 489: 1, 488: 1, 486:
1, 482: 1, 480: 1, 477: 1, 473: 1, 460: 1, 457: 1, 444: 1, 443: 1, 441: 1, 440: 1, 436: 1, 431: 1, 4
23: 1, 415: 1, 407: 1, 403: 1, 394: 1, 392: 1, 387: 1, 380: 1, 378: 1, 375: 1, 364: 1, 358: 1, 355:
1, 352: 1, 348: 1, 339: 1, 338: 1, 337: 1, 334: 1, 333: 1, 331: 1, 330: 1, 322: 1, 318: 1, 306: 1, 3
03: 1, 300: 1, 297: 1, 296: 1, 269: 1, 263: 1, 253: 1, 251: 1, 237: 1, 235: 1, 227: 1, 224: 1, 218:
1, 216: 1, 212: 1, 210: 1, 207: 1, 201: 1, 185: 1, 182: 1, 181: 1, 172: 1, 162: 1, 142: 1, 135: 1, 1
31: 1, 130: 1, 123: 1, 119: 1, 118: 1, 117: 1, 116: 1, 115: 1, 104: 1, 102: 1, 99: 1, 97: 1, 88: 1,
80: 1, 78: 1, 76: 1, 58: 1, 47: 1, 35: 1, 32: 1, 24: 1, 14: 1})
```

```python
# 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.SGDCla
ssifier.html
# -------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=N
one,
# 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_tfidf, y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_text_feature_tfidf, y_train)
    predict_y = sig_clf.predict_proba(cv_text_feature_tfidf)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-1
5))

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


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

predict_y = sig_clf.predict_proba(train_text_feature_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, lab
els=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_text_feature_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predic
t_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_text_feature_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, label
s=clf.classes_, eps=1e-15))
```
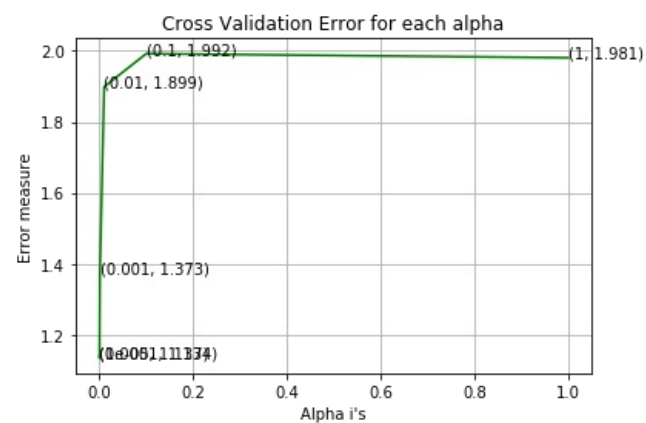
```
For values of alpha =  1e-05 The log loss is: 1.1369260094541893
For values of alpha =  0.0001 The log loss is: 1.1343089839018037
For values of alpha =  0.001 The log loss is: 1.3730813484295175
For values of alpha =  0.01 The log loss is: 1.8987382874674228
For values of alpha =  0.1 The log loss is: 1.9923208744961918
For values of alpha =  1 The log loss is: 1.9807468495377367
```



```
For values of best alpha =  0.0001 The train log loss is: 0.8289618194789619
For values of best alpha =  0.0001 The cross validation log loss is: 1.1343089839018037
For values of best alpha =  0.0001 The test log loss is: 1.107040792952017
```

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

Ans. Yes, it seems like!

In [44]:

```python
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=3, 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 [45]:

```python
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")
```

```
95.5 % of word of test data appeared in train data
94.4 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [46]:

```python
#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)
```

```python
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)
```

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

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

Stacking the three types of features

```
# 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_tfidf = hstack((train_gene_feature_tfidf,train_variation_feature_tfidf))
test_gene_var_tfidf = hstack((test_gene_feature_tfidf,test_variation_feature_tfidf))
cv_gene_var_tfidf = hstack((cv_gene_feature_tfidf,cv_variation_feature_tfidf))

train_x_tfidf = hstack((train_gene_var_tfidf, train_text_feature_tfidf)).tocsr()
train_y = np.array(list(train_df['Class']))

test_x_tfidf = hstack((test_gene_var_tfidf, test_text_feature_tfidf)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_tfidf = hstack((cv_gene_var_tfidf, cv_text_feature_tfidf)).tocsr()
cv_y = np.array(list(cv_df['Class']))


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_variation_feature_responseCoding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variation_feature_responseCoding))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_feature_responseCoding))

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

```
print("TFDIF features representation:\n")
print("(Number of data points * number of features) in train data = ", train_x_tfidf.shape)
print("(Number of data points * number of features) in test data = ", test_x_tfidf.shape)
print("(Number of data points * number of features) in cross validation data =", cv_x_tfidf.shape)
```

```
TFDIF features representation:

(Number of data points * number of features) in train data =  (2124, 3207)
(Number of data points * number of features) in test data =  (665, 3207)
(Number of data points * number of features) in cross validation data = (532, 3207)
```

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

```
 Response encoding features :

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

4.1. Base Line Model

4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

```python
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.n
aive_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.calibra
tion.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_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    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_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)


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

```
for alpha = 1e-05
Log Loss : 1.1493317019449805
for alpha = 0.0001
Log Loss : 1.1486370555375527
for alpha = 0.001
Log Loss : 1.1457345745026473
for alpha = 0.1
Log Loss : 1.1954436504413877
for alpha = 1
Log Loss : 1.2927248208297015
for alpha = 10
Log Loss : 1.4673135413628005
for alpha = 100
Log Loss : 1.4287040154315227
for alpha = 1000
Log Loss : 1.409896370818157
```


Cross Validation Error for each alpha
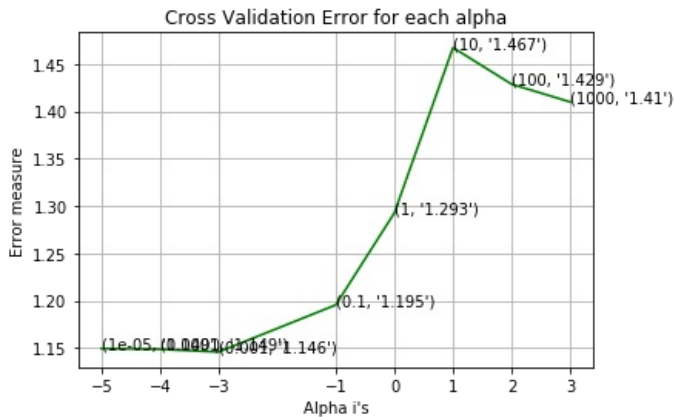
```
For values of best alpha =  0.001 The train log loss is: 0.525322769292468
For values of best alpha =  0.001 The cross validation log loss is: 1.1457345745026473
For values of best alpha =  0.001 The test log loss is: 1.1639006568871844
```

4.1.1.2. Testing the model with best hyper paramters

In [55]:

```python
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.n
aive_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.calibra
tion.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_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
# to avoid rounding error while multiplying probabilites we use log-probability estimates
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_tfidf)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_tfidf.toarray()))
```
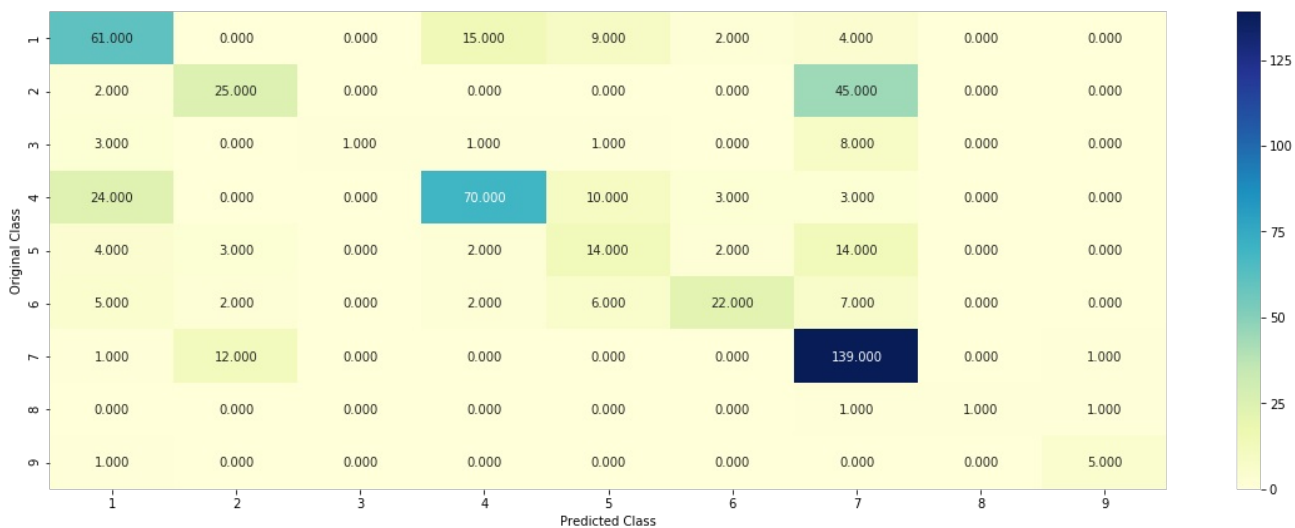
```
Log Loss : 1.1457345745026473
Number of missclassified point : 0.36466165413533835
-------------------- Confusion matrix --------------------
```



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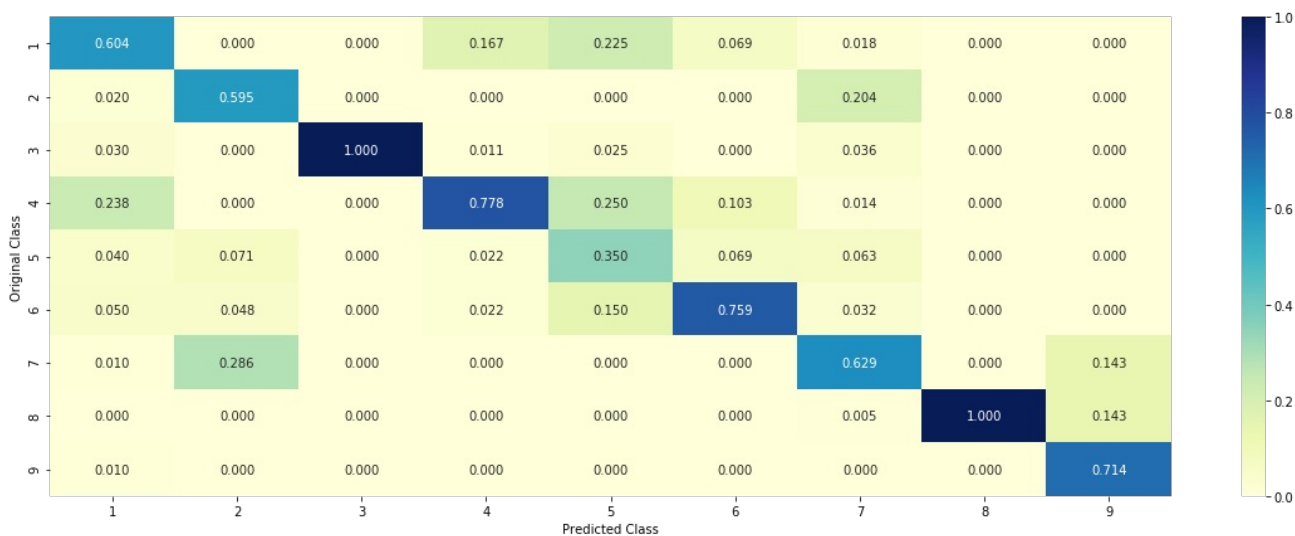


```
-------------------- Recall matrix (Row sum=1) --------------------
```
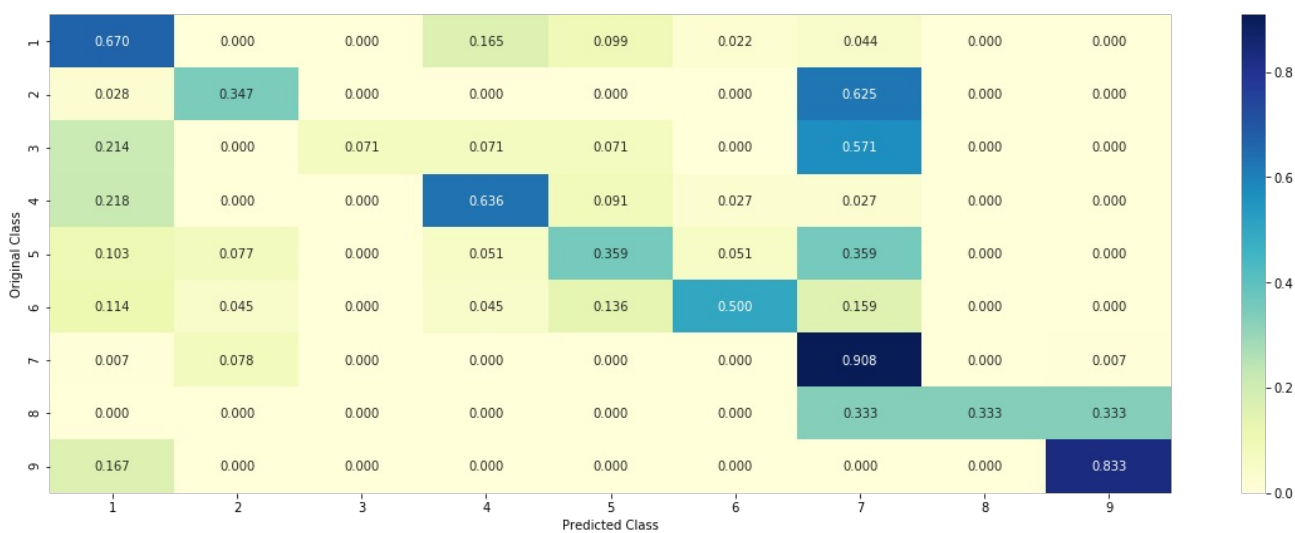

4.1.1.3. Feature Importance, Correctly classified point

```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[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],te
st_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.062  0.0793 0.0104 0.0659 0.0354 0.031  0.71   0.0028 0.0031]]
Actual Class : 7
--------------------------------------------------
16 Text feature [activation] present in test data point [True]
17 Text feature [activated] present in test data point [True]
20 Text feature [kinase] present in test data point [True]
21 Text feature [downstream] present in test data point [True]
22 Text feature [cells] present in test data point [True]
23 Text feature [inhibitor] present in test data point [True]
24 Text feature [expressing] present in test data point [True]
25 Text feature [contrast] present in test data point [True]
26 Text feature [signaling] present in test data point [True]
27 Text feature [factor] present in test data point [True]
28 Text feature [independent] present in test data point [True]
29 Text feature [treatment] present in test data point [True]
30 Text feature [10] present in test data point [True]
31 Text feature [however] present in test data point [True]
32 Text feature [also] present in test data point [True]
36 Text feature [growth] present in test data point [True]
37 Text feature [sensitive] present in test data point [True]
38 Text feature [presence] present in test data point [True]
39 Text feature [treated] present in test data point [True]
40 Text feature [well] present in test data point [True]
41 Text feature [addition] present in test data point [True]
42 Text feature [compared] present in test data point [True]
43 Text feature [constitutive] present in test data point [True]
44 Text feature [activating] present in test data point [True]
45 Text feature [previously] present in test data point [True]
46 Text feature [cell] present in test data point [True]
47 Text feature [mutations] present in test data point [True]
48 Text feature [tyrosine] present in test data point [True]
49 Text feature [shown] present in test data point [True]
50 Text feature [higher] present in test data point [True]
51 Text feature [similar] present in test data point [True]
52 Text feature [receptor] present in test data point [True]
53 Text feature [inhibitors] present in test data point [True]
54 Text feature [potential] present in test data point [True]
55 Text feature [may] present in test data point [True]
56 Text feature [without] present in test data point [True]
57 Text feature [recently] present in test data point [True]
58 Text feature [enhanced] present in test data point [True]
59 Text feature [oncogenic] present in test data point [True]
60 Text feature [showed] present in test data point [True]
61 Text feature [12] present in test data point [True]
62 Text feature [found] present in test data point [True]
63 Text feature [13] present in test data point [True]
64 Text feature [absence] present in test data point [True]
65 Text feature [increased] present in test data point [True]
66 Text feature [constitutively] present in test data point [True]
67 Text feature [suggest] present in test data point [True]
68 Text feature [inhibition] present in test data point [True]
69 Text feature [mutation] present in test data point [True]
70 Text feature [results] present in test data point [True]
71 Text feature [18] present in test data point [True]
72 Text feature [survival] present in test data point [True]
73 Text feature [proliferation] present in test data point [True]
74 Text feature [different] present in test data point [True]
75 Text feature [total] present in test data point [True]
76 Text feature [interestingly] present in test data point [True]
77 Text feature [pathways] present in test data point [True]
78 Text feature [phosphorylation] present in test data point [True]
79 Text feature [clinical] present in test data point [True]
80 Text feature [reported] present in test data point [True]
81 Text feature [discussion] present in test data point [True]
82 Text feature [described] present in test data point [True]
83 Text feature [3b] present in test data point [True]
84 Text feature [20] present in test data point [True]
85 Text feature [although] present in test data point [True]
86 Text feature [mechanism] present in test data point [True]
```

```
87 Text feature [including] present in test data point [True]
88 Text feature [identified] present in test data point [True]
89 Text feature [study] present in test data point [True]
90 Text feature [mutant] present in test data point [True]
91 Text feature [fig] present in test data point [True]
92 Text feature [studies] present in test data point [True]
93 Text feature [patients] present in test data point [True]
94 Text feature [trials] present in test data point [True]
95 Text feature [ligand] present in test data point [True]
96 Text feature [could] present in test data point [True]
97 Text feature [respectively] present in test data point [True]
98 Text feature [using] present in test data point [True]
99 Text feature [serum] present in test data point [True]
Out of the top  100  features  79 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [61]:

```python
test_point_index = 2
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[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],te
st_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.0917 0.0585 0.0136 0.0858 0.5928 0.0416 0.1083 0.0036 0.004 ]]
Actual Class : 1
--------------------------------------------------
5 Text feature [assays] present in test data point [True]
6 Text feature [functional] present in test data point [True]
7 Text feature [variants] present in test data point [True]
8 Text feature [neutral] present in test data point [True]
9 Text feature [brca1] present in test data point [True]
10 Text feature [assay] present in test data point [True]
11 Text feature [variant] present in test data point [True]
12 Text feature [based] present in test data point [True]
16 Text feature [likely] present in test data point [True]
18 Text feature [assess] present in test data point [True]
19 Text feature [introduction] present in test data point [True]
20 Text feature [author] present in test data point [True]
21 Text feature [available] present in test data point [True]
22 Text feature [large] present in test data point [True]
23 Text feature [effect] present in test data point [True]
24 Text feature [data] present in test data point [True]
25 Text feature [embryonic] present in test data point [True]
26 Text feature [missense] present in test data point [True]
27 Text feature [brct] present in test data point [True]
28 Text feature [assessment] present in test data point [True]
29 Text feature [manuscript] present in test data point [True]
30 Text feature [methods] present in test data point [True]
31 Text feature [pa] present in test data point [True]
32 Text feature [results] present in test data point [True]
33 Text feature [specificity] present in test data point [True]
34 Text feature [information] present in test data point [True]
35 Text feature [research] present in test data point [True]
36 Text feature [used] present in test data point [True]
37 Text feature [database] present in test data point [True]
38 Text feature [structural] present in test data point [True]
39 Text feature [general] present in test data point [True]
40 Text feature [classification] present in test data point [True]
41 Text feature [risk] present in test data point [True]
42 Text feature [controls] present in test data point [True]
43 Text feature [remaining] present in test data point [True]
44 Text feature [tested] present in test data point [True]
45 Text feature [pathogenic] present in test data point [True]
46 Text feature [intermediate] present in test data point [True]
47 Text feature [published] present in test data point [True]
48 Text feature [discussion] present in test data point [True]
49 Text feature [comparison] present in test data point [True]
50 Text feature [function] present in test data point [True]
51 Text feature [addition] present in test data point [True]
52 Text feature [although] present in test data point [True]
53 Text feature [table] present in test data point [True]
54 Text feature [activities] present in test data point [True]
55 Text feature [provide] present in test data point [True]
56 Text feature [set] present in test data point [True]
```

```
57 Text feature [classified] present in test data point [True]
58 Text feature [cancer] present in test data point [True]
63 Text feature [nih] present in test data point [True]
64 Text feature [sequence] present in test data point [True]
65 Text feature [defined] present in test data point [True]
66 Text feature [genetic] present in test data point [True]
67 Text feature [sensitivity] present in test data point [True]
68 Text feature [vitro] present in test data point [True]
69 Text feature [several] present in test data point [True]
71 Text feature [protein] present in test data point [True]
72 Text feature [correlation] present in test data point [True]
73 Text feature [strong] present in test data point [True]
74 Text feature [use] present in test data point [True]
75 Text feature [position] present in test data point [True]
76 Text feature [predicted] present in test data point [True]
77 Text feature [method] present in test data point [True]
78 Text feature [functionally] present in test data point [True]
79 Text feature [also] present in test data point [True]
80 Text feature [possible] present in test data point [True]
81 Text feature [another] present in test data point [True]
82 Text feature [known] present in test data point [True]
83 Text feature [however] present in test data point [True]
84 Text feature [type] present in test data point [True]
85 Text feature [wild] present in test data point [True]
86 Text feature [whereas] present in test data point [True]
87 Text feature [analysis] present in test data point [True]
88 Text feature [affect] present in test data point [True]
89 Text feature [corresponding] present in test data point [True]
90 Text feature [three] present in test data point [True]
91 Text feature [example] present in test data point [True]
92 Text feature [displayed] present in test data point [True]
93 Text feature [control] present in test data point [True]
94 Text feature [fact] present in test data point [True]
95 Text feature [previously] present in test data point [True]
96 Text feature [class] present in test data point [True]
97 Text feature [clear] present in test data point [True]
98 Text feature [specifically] present in test data point [True]
Out of the top  100  features  85 are present in query point
```

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibra
tion.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, lab
els=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predic
t_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, predict_y, label
s=clf.classes_, eps=1e-15))
```

```
for alpha = 5
Log Loss : 1.0300430831065184
for alpha = 11
Log Loss : 1.016842277311916
for alpha = 15
Log Loss : 1.0286803826450734
for alpha = 21
Log Loss : 1.0589248958268909
for alpha = 31
Log Loss : 1.0721786691038484
for alpha = 41
Log Loss : 1.0649762873365463
for alpha = 51
Log Loss : 1.062998929548556
for alpha = 99
Log Loss : 1.0842507389031995
```


Cross Validation Error for each alpha
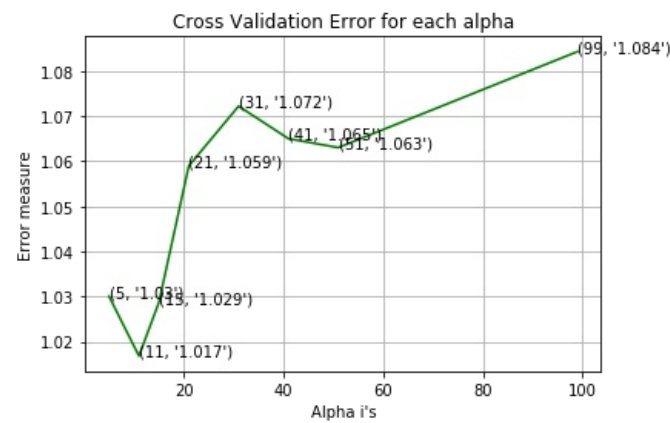
```
For values of best alpha =  11 The train log loss is: 0.6428137993704953
For values of best alpha =  11 The cross validation log loss is: 1.016842277311916
For values of best alpha =  11 The test log loss is: 1.0328097585725124
```

4.2.2. Testing the model with best hyper paramters

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

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

```
Log loss : 1.016842277311916
Number of mis-classified points : 0.36278195488721804
------------------- Confusion matrix -------------------
```
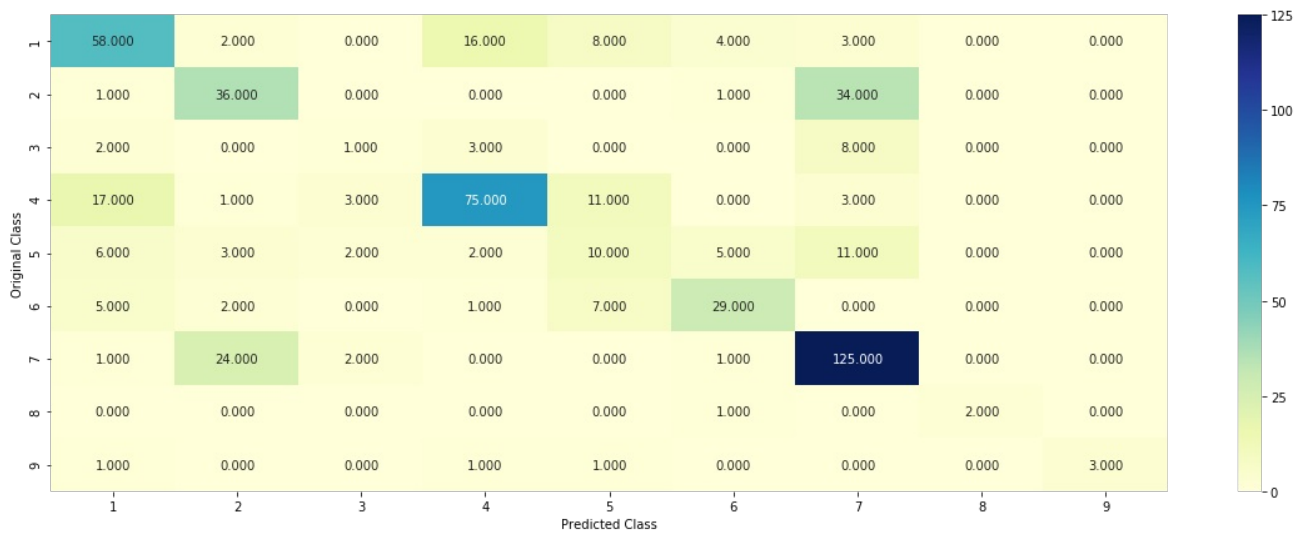
-------------------- Precision matrix (Columm Sum=1) --------------------
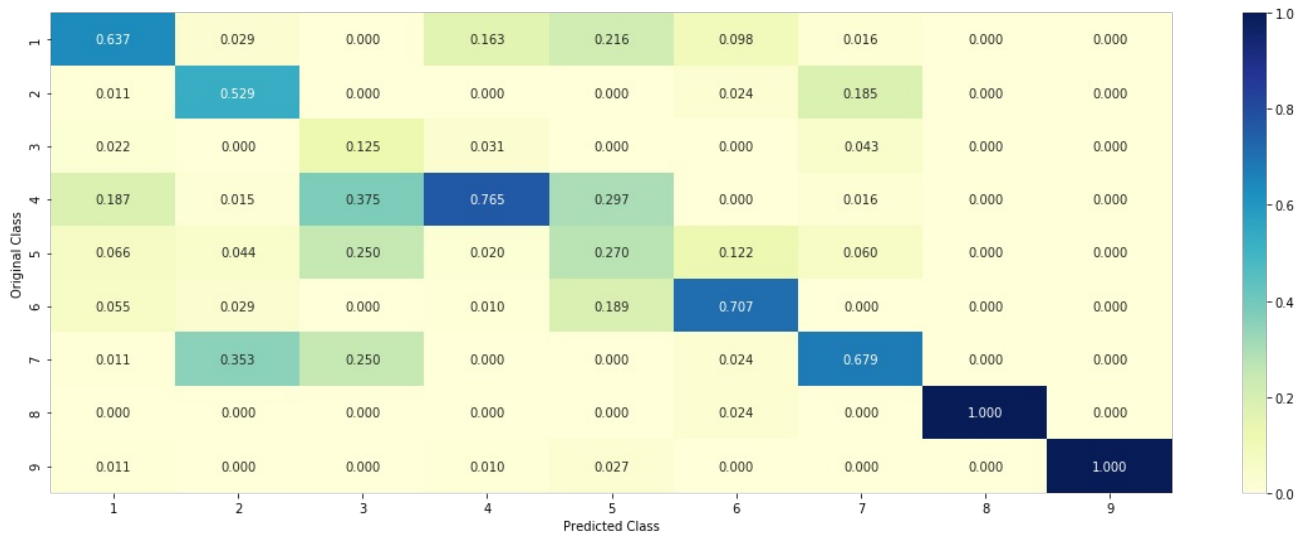
-------------------- Recall matrix (Row sum=1) --------------------
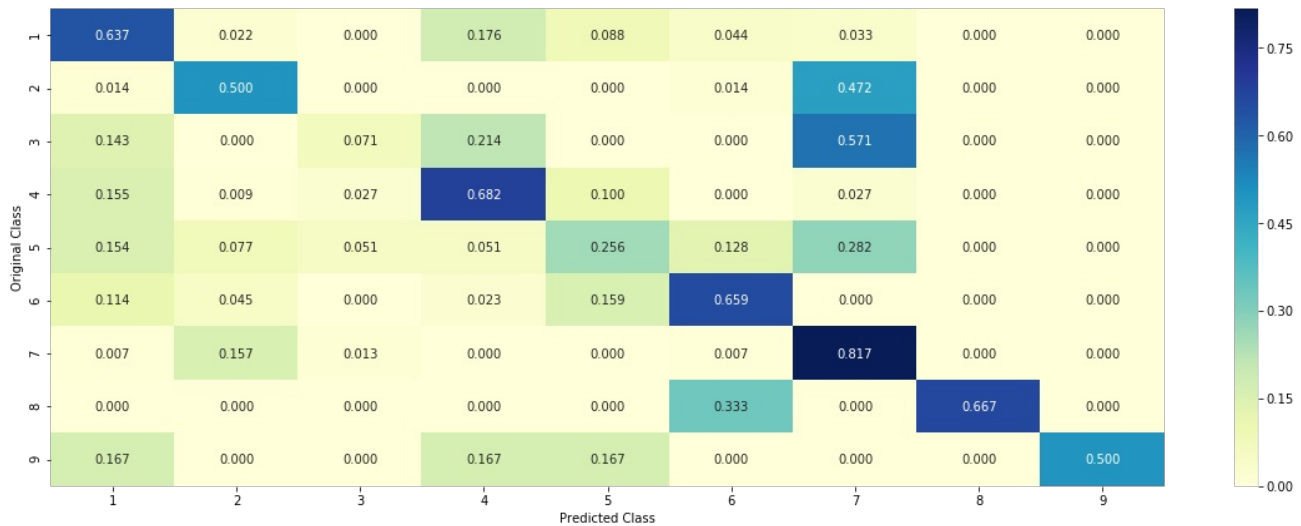
4.2.3.Sample Query point -1

```
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[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 ",alpha[best_alpha]," nearest neighbours of the test points belongs to classes",train_y[neighbors[1][0
]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 7
The  11  nearest neighbours of the test points belongs to classes [7 7 7 7 5 7 7 2 7 2 2]
Fequency of nearest points : Counter({7: 7, 2: 3, 5: 1})
```

4.2.4. Sample Query Point-2

```
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

#Correction @5:32
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 belongs to classe
s",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 4
Actual Class : 4
The K value for knn is 11 and the nearest neighbours of the test points belongs to classes [4 4 4 1
4 4 4 4 1 4 4]
Fequency of nearest points : Counter({4: 9, 1: 2})
```

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

```python
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDCla
ssifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=N
one,
# 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.calibra
tion.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_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    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_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)

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

```
for alpha = 1e-06
Log Loss : 1.157478261526367
for alpha = 1e-05
Log Loss : 1.097320254982882
for alpha = 0.0001
Log Loss : 1.0219203812363364
for alpha = 0.001
Log Loss : 1.0235774358881025
for alpha = 0.01
Log Loss : 1.169715223917661
for alpha = 0.1
Log Loss : 1.565443655599046
for alpha = 1
Log Loss : 1.7086009180237496
for alpha = 10
Log Loss : 1.7262410391864038
for alpha = 100
Log Loss : 1.7278988939672637
```


Cross Validation Error for each alpha
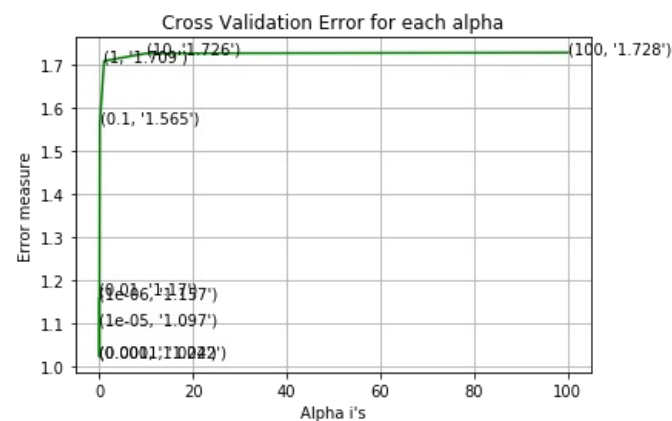
```
For values of best alpha =  0.0001 The train log loss is: 0.4417916218658447
For values of best alpha =  0.0001 The cross validation log loss is: 1.0219203812363364
For values of best alpha =  0.0001 The test log loss is: 0.9781512523620731
```

4.3.1.2. Testing the model with best hyper paramters

In [67]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDCla
ssifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=N
one,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0
.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

```
Log loss : 1.0219203812363364
Number of mis-classified points : 0.34022556390977443
-------------------- Confusion matrix --------------------
```
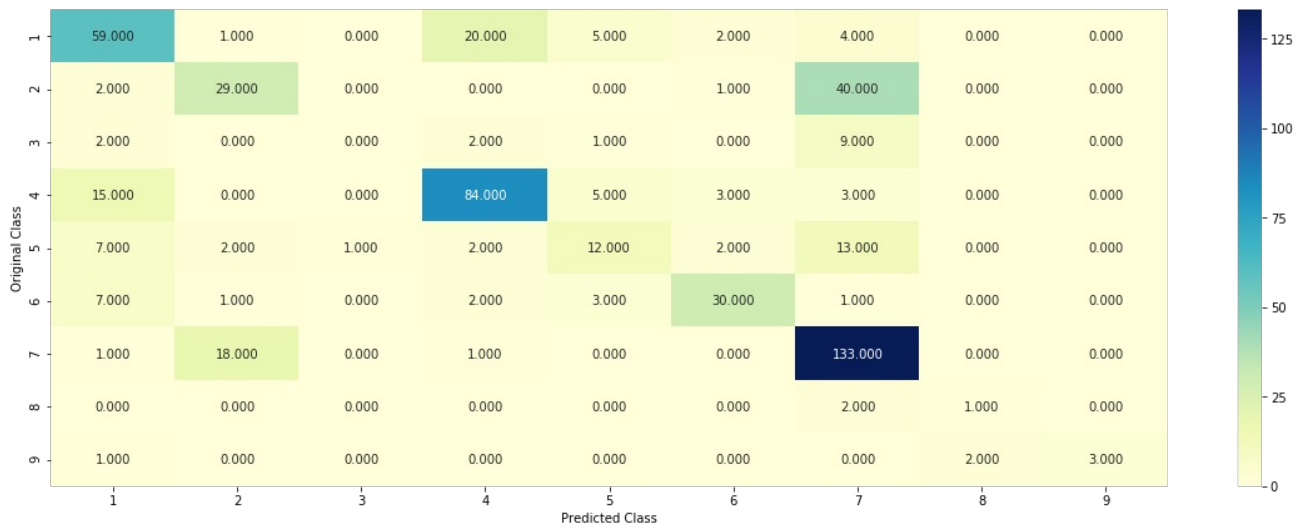
-------------------- Precision matrix (Columm Sum=1) --------------------
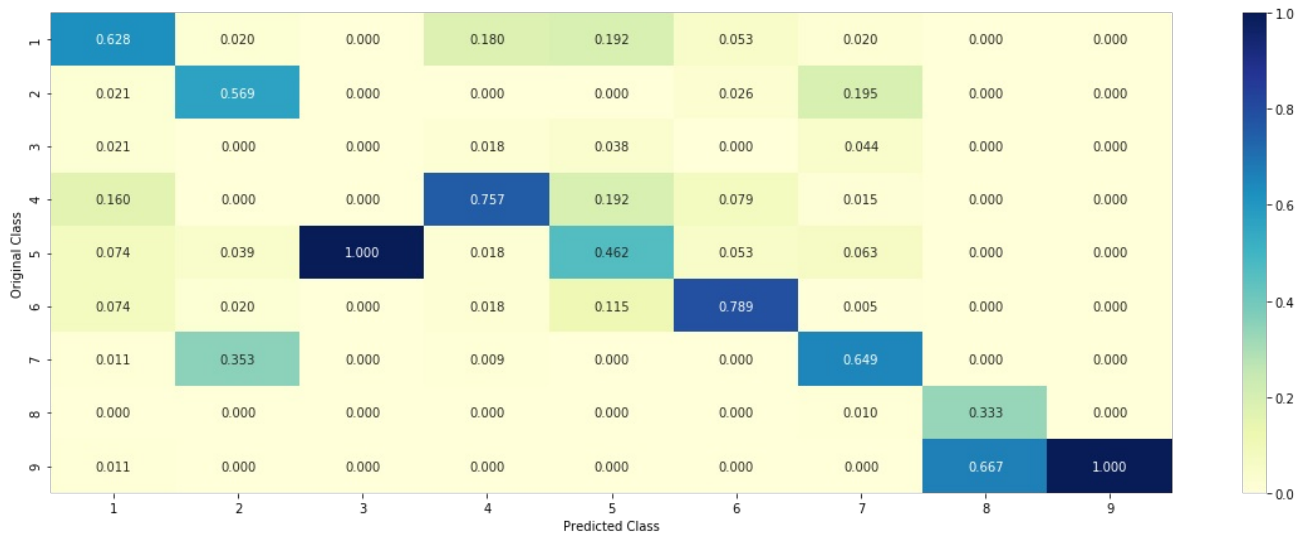
-------------------- Recall matrix (Row sum=1) --------------------
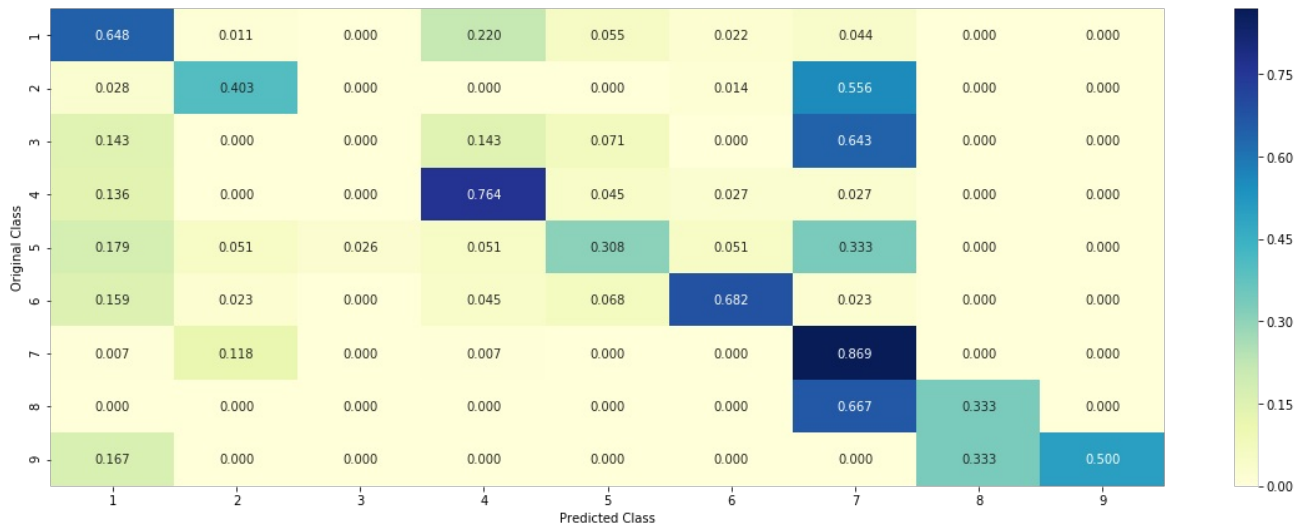
4.3.1.3. Feature Importance

```python
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_tfidf.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']))
```

4.3.1.3.1. Correctly Classified point

```python
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_tfidf,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[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: [[0.0236 0.0893 0.0016 0.0072 0.0326 0.0091 0.833  0.0019 0.0018]]
Actual Class : 7
-------------------------------------------------
7 Text feature [oncogene] present in test data point [True]
8 Text feature [activation] present in test data point [True]
9 Text feature [ligand] present in test data point [True]
11 Text feature [enhanced] present in test data point [True]
14 Text feature [factor] present in test data point [True]
15 Text feature [transformed] present in test data point [True]
25 Text feature [downstream] present in test data point [True]
28 Text feature [overexpression] present in test data point [True]
35 Text feature [constitutive] present in test data point [True]
37 Text feature [activated] present in test data point [True]
54 Text feature [lung] present in test data point [True]
57 Text feature [egf] present in test data point [True]
68 Text feature [oncogenic] present in test data point [True]
73 Text feature [inhibited] present in test data point [True]
90 Text feature [transforming] present in test data point [True]
102 Text feature [activate] present in test data point [True]
106 Text feature [advanced] present in test data point [True]
108 Text feature [bp] present in test data point [True]
118 Text feature [leading] present in test data point [True]
132 Text feature [signaling] present in test data point [True]
137 Text feature [receptors] present in test data point [True]
139 Text feature [observations] present in test data point [True]
144 Text feature [regulated] present in test data point [True]
147 Text feature [tyrosine] present in test data point [True]
148 Text feature [derived] present in test data point [True]
155 Text feature [approximately] present in test data point [True]
179 Text feature [membrane] present in test data point [True]
190 Text feature [concentrations] present in test data point [True]
201 Text feature [presence] present in test data point [True]
202 Text feature [genomic] present in test data point [True]
209 Text feature [properties] present in test data point [True]
210 Text feature [factors] present in test data point [True]
211 Text feature [distinct] present in test data point [True]
214 Text feature [000] present in test data point [True]
230 Text feature [long] present in test data point [True]
250 Text feature [pathways] present in test data point [True]
252 Text feature [days] present in test data point [True]
256 Text feature [s3] present in test data point [True]
257 Text feature [her2] present in test data point [True]
259 Text feature [epithelial] present in test data point [True]
262 Text feature [inhibitor] present in test data point [True]
277 Text feature [size] present in test data point [True]
284 Text feature [3b] present in test data point [True]
301 Text feature [position] present in test data point [True]
306 Text feature [phosphorylated] present in test data point [True]
313 Text feature [free] present in test data point [True]
320 Text feature [serum] present in test data point [True]
333 Text feature [2a] present in test data point [True]
336 Text feature [gfp] present in test data point [True]
342 Text feature [colony] present in test data point [True]
347 Text feature [medium] present in test data point [True]
349 Text feature [fold] present in test data point [True]
356 Text feature [activating] present in test data point [True]
357 Text feature [increased] present in test data point [True]
372 Text feature [malignant] present in test data point [True]
373 Text feature [day] present in test data point [True]
387 Text feature [expressing] present in test data point [True]
392 Text feature [transformation] present in test data point [True]
399 Text feature [culture] present in test data point [True]
407 Text feature [pdgfr] present in test data point [True]
431 Text feature [mechanisms] present in test data point [True]
432 Text feature [sensitive] present in test data point [True]
445 Text feature [ras] present in test data point [True]
466 Text feature [occur] present in test data point [True]
473 Text feature [25] present in test data point [True]
497 Text feature [positive] present in test data point [True]
498 Text feature [specimens] present in test data point [True]
499 Text feature [added] present in test data point [True]
Out of the top  500  features  68 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

```
In [75]:
test_point_index = 4
no_feature = 559
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[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],te
st_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.5547 0.0644 0.0038 0.1776 0.041  0.126  0.0295 0.0007 0.0021]]
Actual Class : 4
--------------------------------------------------
210 Text feature [region] present in test data point [True]
218 Text feature [calculated] present in test data point [True]
252 Text feature [age] present in test data point [True]
257 Text feature [early] present in test data point [True]
270 Text feature [cohort] present in test data point [True]
274 Text feature [screening] present in test data point [True]
285 Text feature [families] present in test data point [True]
298 Text feature [39] present in test data point [True]
319 Text feature [chain] present in test data point [True]
334 Text feature [dna] present in test data point [True]
337 Text feature [position] present in test data point [True]
338 Text feature [frequency] present in test data point [True]
345 Text feature [59] present in test data point [True]
346 Text feature [hr] present in test data point [True]
348 Text feature [defined] present in test data point [True]
350 Text feature [46] present in test data point [True]
369 Text feature [17] present in test data point [True]
370 Text feature [following] present in test data point [True]
372 Text feature [colorectal] present in test data point [True]
373 Text feature [next] present in test data point [True]
376 Text feature [furthermore] present in test data point [True]
381 Text feature [msh2] present in test data point [True]
382 Text feature [coding] present in test data point [True]
384 Text feature [general] present in test data point [True]
385 Text feature [19] present in test data point [True]
388 Text feature [2012] present in test data point [True]
391 Text feature [criteria] present in test data point [True]
394 Text feature [derived] present in test data point [True]
399 Text feature [pathogenic] present in test data point [True]
400 Text feature [sequencing] present in test data point [True]
418 Text feature [individual] present in test data point [True]
422 Text feature [yet] present in test data point [True]
423 Text feature [rather] present in test data point [True]
424 Text feature [even] present in test data point [True]
426 Text feature [moreover] present in test data point [True]
428 Text feature [function] present in test data point [True]
430 Text feature [cause] present in test data point [True]
431 Text feature [nucleotide] present in test data point [True]
433 Text feature [carcinoma] present in test data point [True]
435 Text feature [exon] present in test data point [True]
437 Text feature [dependent] present in test data point [True]
441 Text feature [breast] present in test data point [True]
444 Text feature [ovarian] present in test data point [True]
449 Text feature [cancers] present in test data point [True]
451 Text feature [diagnosis] present in test data point [True]
458 Text feature [whole] present in test data point [True]
460 Text feature [like] present in test data point [True]
461 Text feature [carrying] present in test data point [True]
464 Text feature [mean] present in test data point [True]
465 Text feature [genome] present in test data point [True]
466 Text feature [performed] present in test data point [True]
469 Text feature [possible] present in test data point [True]
470 Text feature [type] present in test data point [True]
471 Text feature [analyzed] present in test data point [True]
476 Text feature [42] present in test data point [True]
477 Text feature [six] present in test data point [True]
478 Text feature [strong] present in test data point [True]
482 Text feature [established] present in test data point [True]
485 Text feature [carcinomas] present in test data point [True]
488 Text feature [directly] present in test data point [True]
490 Text feature [either] present in test data point [True]
491 Text feature [reverse] present in test data point [True]
498 Text feature [mlh1] present in test data point [True]
500 Text feature [population] present in test data point [True]
501 Text feature [many] present in test data point [True]
503 Text feature [obtained] present in test data point [True]
```

```
504 Text feature [contact] present in test data point [True]
507 Text feature [reduced] present in test data point [True]
509 Text feature [24] present in test data point [True]
513 Text feature [previous] present in test data point [True]
518 Text feature [reported] present in test data point [True]
519 Text feature [large] present in test data point [True]
520 Text feature [important] present in test data point [True]
534 Text feature [within] present in test data point [True]
537 Text feature [one] present in test data point [True]
539 Text feature [therefore] present in test data point [True]
540 Text feature [table] present in test data point [True]
543 Text feature [protein] present in test data point [True]
545 Text feature [related] present in test data point [True]
546 Text feature [relative] present in test data point [True]
547 Text feature [affected] present in test data point [True]
548 Text feature [indicated] present in test data point [True]
550 Text feature [revealed] present in test data point [True]
553 Text feature [16] present in test data point [True]
554 Text feature [statistical] present in test data point [True]
555 Text feature [years] present in test data point [True]
558 Text feature [deletions] present in test data point [True]
Out of the top  559  features  87 are present in query point
```

4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

```python
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDCla
ssifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=N
one,
# 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.calibra
tion.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_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    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_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)

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

```
for alpha = 1e-06
Log Loss : 1.1392628976163706
for alpha = 1e-05
Log Loss : 1.1449260678627562
for alpha = 0.0001
Log Loss : 1.075402944052435
for alpha = 0.001
Log Loss : 1.1106112143720728
for alpha = 0.01
Log Loss : 1.3305397709621658
for alpha = 0.1
Log Loss : 1.7369695554968776
for alpha = 1
Log Loss : 1.8284725747506534
```


Cross Validation Error for each alpha
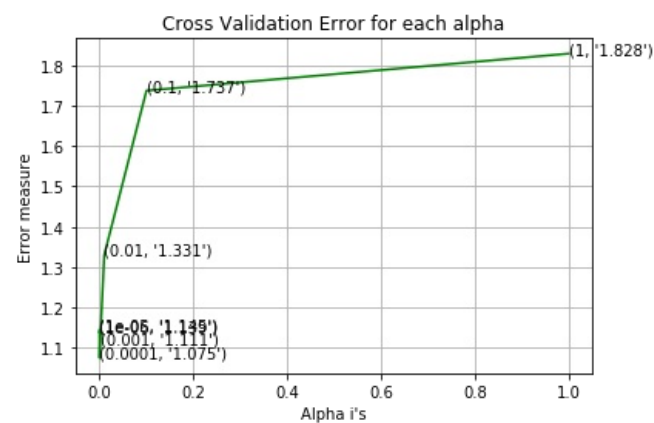
```
For values of best alpha =  0.0001 The train log loss is: 0.4330698941130909
For values of best alpha =  0.0001 The cross validation log loss is: 1.075402944052435
For values of best alpha =  0.0001 The test log loss is: 1.0054637386066378
```

4.3.2.2. Testing model with best hyper parameters

In [77]:

```python
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDCla
ssifier.html
# -------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=N
one,
# 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_tfidf, train_y, cv_x_tfidf, cv_y, clf)
```

```
Log loss : 1.075402944052435
Number of mis-classified points : 0.35150375939849626
------------------- Confusion matrix -------------------
```

Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	58.000	1.000	0.000	22.000	4.000	2.000	4.000	0.000	0.000
2	2.000	26.000	0.000	0.000	0.000	1.000	43.000	0.000	0.000
3	2.000	0.000	0.000	3.000	0.000	0.000	9.000	0.000	0.000
4	16.000	0.000	0.000	84.000	4.000	3.000	3.000	0.000	0.000
5	7.000	2.000	0.000	3.000	10.000	2.000	15.000	0.000	0.000
6	7.000	1.000	0.000	3.000	2.000	30.000	1.000	0.000	0.000
7	1.000	16.000	0.000	1.000	0.000	0.000	135.000	0.000	0.000
8	0.000	0.000	0.000	1.000	0.000	0.000	2.000	0.000	0.000
9	2.000	0.000	0.000	1.000	0.000	0.000	1.000	0.000	2.000

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

Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.611	0.022		0.186	0.200	0.053	0.019		0.000
2	0.021	0.565		0.000	0.000	0.026	0.202		0.000
3	0.021	0.000		0.025	0.000	0.000	0.042		0.000
4	0.168	0.000		0.712	0.200	0.079	0.014		0.000
5	0.074	0.043		0.025	0.500	0.053	0.070		0.000
6	0.074	0.022		0.025	0.100	0.789	0.005		0.000
7	0.011	0.348		0.008	0.000	0.000	0.634		0.000
8	0.000	0.000		0.008	0.000	0.000	0.009		0.000
9	0.021	0.000		0.008	0.000	0.000	0.005		1.000

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

Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.637	0.011	0.000	0.242	0.044	0.022	0.044	0.000	0.000
2	0.028	0.361	0.000	0.000	0.000	0.014	0.597	0.000	0.000
3	0.143	0.000	0.000	0.214	0.000	0.000	0.643	0.000	0.000
4	0.145	0.000	0.000	0.764	0.036	0.027	0.027	0.000	0.000
5	0.179	0.051	0.000	0.077	0.256	0.051	0.385	0.000	0.000
6	0.159	0.023	0.000	0.068	0.045	0.682	0.023	0.000	0.000
7	0.007	0.105	0.000	0.007	0.000	0.000	0.882	0.000	0.000
8	0.000	0.000	0.000	0.333	0.000	0.000	0.667	0.000	0.000
9	0.333	0.000	0.000	0.167	0.000	0.000	0.167	0.000	0.333

4.3.2.3. Feature Importance, Correctly Classified point

```python
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_tfidf,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[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)
```
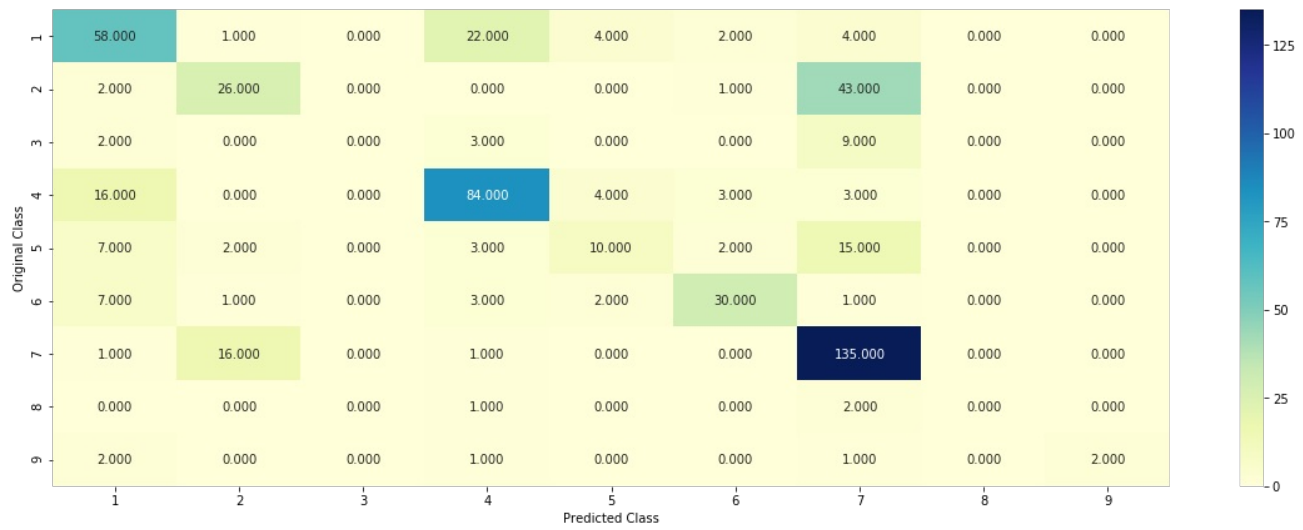
```
Predicted Class : 7
Predicted Class Probabilities: [[2.370e-02 8.090e-02 1.200e-03 7.700e-03 2.760e-02 8.100e-03 8.503e-
01
  1.000e-04 3.000e-04]]
Actual Class : 7
-------------------------------------------------
11 Text feature [enhanced] present in test data point [True]
19 Text feature [oncogene] present in test data point [True]
21 Text feature [factor] present in test data point [True]
23 Text feature [activation] present in test data point [True]
24 Text feature [ligand] present in test data point [True]
25 Text feature [overexpression] present in test data point [True]
30 Text feature [downstream] present in test data point [True]
32 Text feature [transformed] present in test data point [True]
68 Text feature [activated] present in test data point [True]
80 Text feature [lung] present in test data point [True]
91 Text feature [constitutive] present in test data point [True]
96 Text feature [inhibited] present in test data point [True]
103 Text feature [oncogenic] present in test data point [True]
104 Text feature [bp] present in test data point [True]
127 Text feature [advanced] present in test data point [True]
131 Text feature [egf] present in test data point [True]
137 Text feature [leading] present in test data point [True]
144 Text feature [membrane] present in test data point [True]
146 Text feature [regulated] present in test data point [True]
148 Text feature [derived] present in test data point [True]
149 Text feature [activate] present in test data point [True]
154 Text feature [properties] present in test data point [True]
159 Text feature [her2] present in test data point [True]
169 Text feature [observations] present in test data point [True]
178 Text feature [approximately] present in test data point [True]
192 Text feature [distinct] present in test data point [True]
211 Text feature [genomic] present in test data point [True]
212 Text feature [s3] present in test data point [True]
215 Text feature [transforming] present in test data point [True]
224 Text feature [presence] present in test data point [True]
242 Text feature [concentrations] present in test data point [True]
254 Text feature [000] present in test data point [True]
258 Text feature [signaling] present in test data point [True]
270 Text feature [receptors] present in test data point [True]
275 Text feature [epithelial] present in test data point [True]
296 Text feature [tyrosine] present in test data point [True]
303 Text feature [3b] present in test data point [True]
307 Text feature [2a] present in test data point [True]
308 Text feature [factors] present in test data point [True]
309 Text feature [free] present in test data point [True]
312 Text feature [inhibitor] present in test data point [True]
321 Text feature [size] present in test data point [True]
325 Text feature [long] present in test data point [True]
327 Text feature [phosphorylated] present in test data point [True]
330 Text feature [increased] present in test data point [True]
343 Text feature [days] present in test data point [True]
350 Text feature [position] present in test data point [True]
354 Text feature [ras] present in test data point [True]
356 Text feature [pathways] present in test data point [True]
361 Text feature [fold] present in test data point [True]
383 Text feature [colony] present in test data point [True]
397 Text feature [gfp] present in test data point [True]
402 Text feature [positive] present in test data point [True]
414 Text feature [25] present in test data point [True]
421 Text feature [sensitive] present in test data point [True]
427 Text feature [activating] present in test data point [True]
438 Text feature [medium] present in test data point [True]
440 Text feature [occur] present in test data point [True]
444 Text feature [pdgfr] present in test data point [True]
449 Text feature [culture] present in test data point [True]
455 Text feature [serum] present in test data point [True]
458 Text feature [added] present in test data point [True]
467 Text feature [day] present in test data point [True]
468 Text feature [high] present in test data point [True]
470 Text feature [malignant] present in test data point [True]
472 Text feature [strongly] present in test data point [True]
475 Text feature [progression] present in test data point [True]
481 Text feature [expressing] present in test data point [True]
488 Text feature [mechanisms] present in test data point [True]
Out of the top  500  features  69 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

```python
test_point_index = 4
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[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],te
st_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[5.589e-01 6.410e-02 6.200e-03 1.774e-01 4.130e-02 1.152e-01 3.660e-
02
  1.000e-04 2.000e-04]]
Actual Class : 4
-------------------------------------------------
209 Text feature [region] present in test data point [True]
236 Text feature [calculated] present in test data point [True]
248 Text feature [screening] present in test data point [True]
258 Text feature [age] present in test data point [True]
267 Text feature [early] present in test data point [True]
271 Text feature [39] present in test data point [True]
276 Text feature [cohort] present in test data point [True]
283 Text feature [families] present in test data point [True]
322 Text feature [chain] present in test data point [True]
327 Text feature [46] present in test data point [True]
330 Text feature [position] present in test data point [True]
337 Text feature [dna] present in test data point [True]
339 Text feature [59] present in test data point [True]
340 Text feature [hr] present in test data point [True]
345 Text feature [frequency] present in test data point [True]
348 Text feature [defined] present in test data point [True]
357 Text feature [msh2] present in test data point [True]
358 Text feature [17] present in test data point [True]
360 Text feature [next] present in test data point [True]
361 Text feature [2012] present in test data point [True]
362 Text feature [furthermore] present in test data point [True]
364 Text feature [following] present in test data point [True]
369 Text feature [19] present in test data point [True]
378 Text feature [pathogenic] present in test data point [True]
383 Text feature [yet] present in test data point [True]
388 Text feature [general] present in test data point [True]
390 Text feature [colorectal] present in test data point [True]
391 Text feature [nucleotide] present in test data point [True]
393 Text feature [criteria] present in test data point [True]
397 Text feature [sequencing] present in test data point [True]
403 Text feature [coding] present in test data point [True]
409 Text feature [individual] present in test data point [True]
411 Text feature [rather] present in test data point [True]
412 Text feature [derived] present in test data point [True]
413 Text feature [cause] present in test data point [True]
419 Text feature [even] present in test data point [True]
420 Text feature [dependent] present in test data point [True]
423 Text feature [carcinoma] present in test data point [True]
426 Text feature [analyzed] present in test data point [True]
427 Text feature [function] present in test data point [True]
430 Text feature [moreover] present in test data point [True]
432 Text feature [ovarian] present in test data point [True]
436 Text feature [exon] present in test data point [True]
437 Text feature [42] present in test data point [True]
440 Text feature [cancers] present in test data point [True]
445 Text feature [either] present in test data point [True]
448 Text feature [possible] present in test data point [True]
449 Text feature [strong] present in test data point [True]
451 Text feature [diagnosis] present in test data point [True]
452 Text feature [like] present in test data point [True]
453 Text feature [directly] present in test data point [True]
458 Text feature [established] present in test data point [True]
464 Text feature [six] present in test data point [True]
466 Text feature [breast] present in test data point [True]
472 Text feature [whole] present in test data point [True]
473 Text feature [mlh1] present in test data point [True]
474 Text feature [performed] present in test data point [True]
478 Text feature [reduced] present in test data point [True]
482 Text feature [mean] present in test data point [True]
483 Text feature [previous] present in test data point [True]
485 Text feature [genome] present in test data point [True]
486 Text feature [contact] present in test data point [True]
489 Text feature [important] present in test data point [True]
491 Text feature [many] present in test data point [True]
499 Text feature [type] present in test data point [True]
Out of the top  500  features  65 are present in query point
```

4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

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

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

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



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibra
tion.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_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

```
for C = 1e-05
Log Loss : 1.0992913078105087
for C = 0.0001
Log Loss : 1.0749694386284763
for C = 0.001
Log Loss : 1.0764693086284474
for C = 0.01
Log Loss : 1.3102137267928047
for C = 0.1
Log Loss : 1.6018193764838544
for C = 1
Log Loss : 1.727755050303273
for C = 10
Log Loss : 1.7277547278778702
for C = 100
Log Loss : 1.727754790523812
```
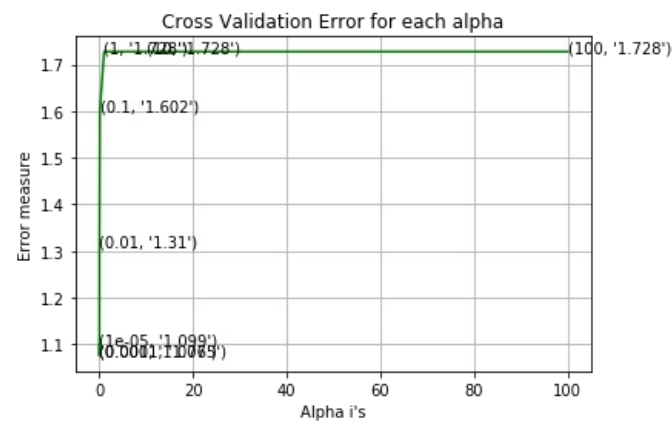


```
For values of best alpha =  0.0001 The train log loss is: 0.5047416233425432
For values of best alpha =  0.0001 The cross validation log loss is: 1.0749694386284763
For values of best alpha =  0.0001 The test log loss is: 1.0419275369526866
```

4.4.2. Testing model with best hyper parameters

In [83]:

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

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

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

```
Log loss : 1.0749694386284763
Number of mis-classified points : 0.35526315789473684
------------------- Confusion matrix -------------------
```
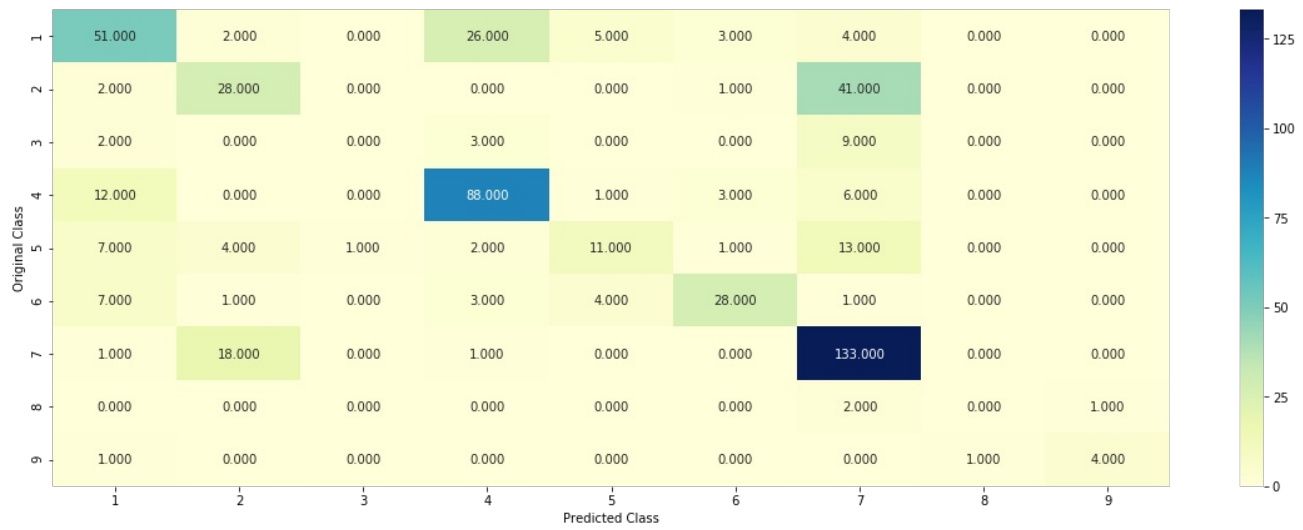
-------------------- Precision matrix (Columm Sum=1) --------------------

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

4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

```python
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_tfidf,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[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],te
st_df['Variation'].iloc[test_point_index], no_feature)
```
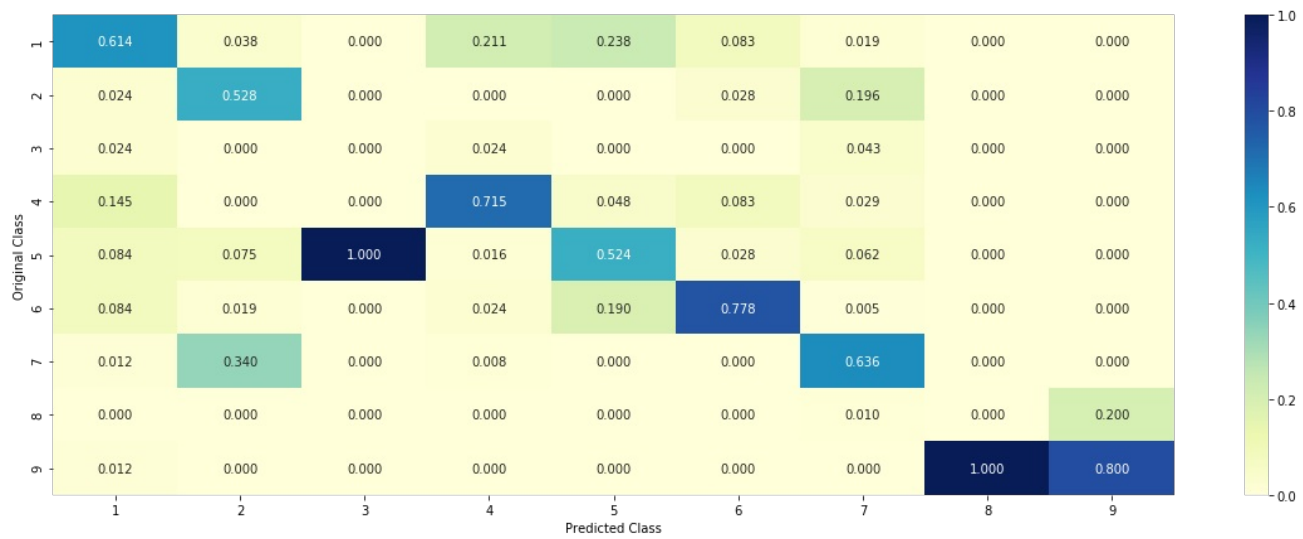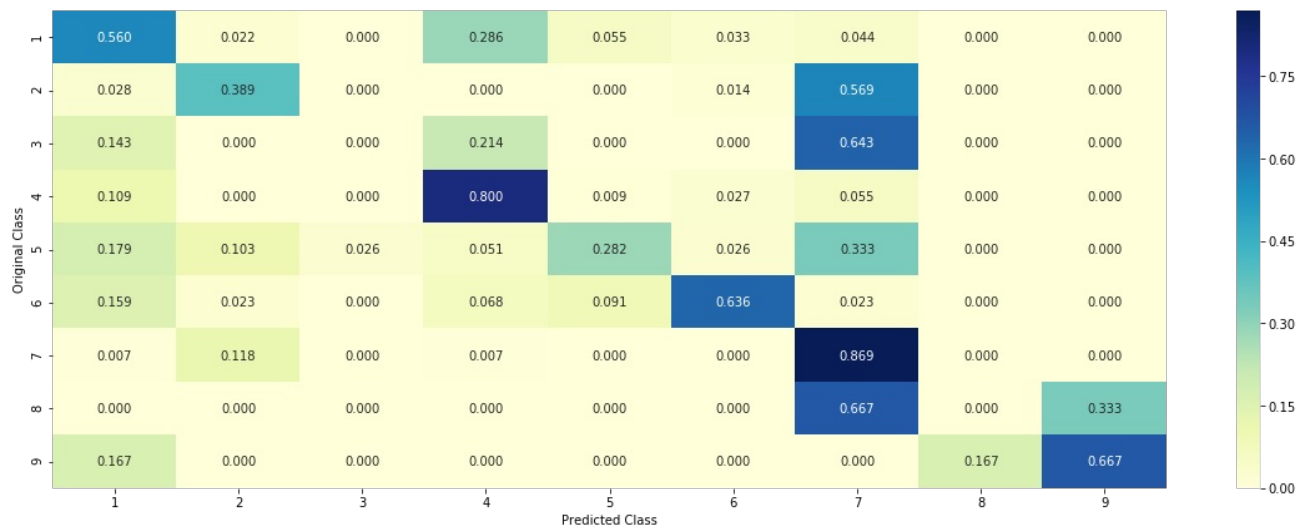
```python
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_tfidf,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[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],te
st_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.038  0.0529 0.0022 0.0175 0.0295 0.0221 0.8328 0.0017 0.0034]]
Actual Class : 7
--------------------------------------------------
25 Text feature [overexpression] present in test data point [True]
26 Text feature [enhanced] present in test data point [True]
27 Text feature [downstream] present in test data point [True]
30 Text feature [lung] present in test data point [True]
31 Text feature [factor] present in test data point [True]
41 Text feature [derived] present in test data point [True]
217 Text feature [oncogene] present in test data point [True]
219 Text feature [activation] present in test data point [True]
220 Text feature [bp] present in test data point [True]
221 Text feature [ligand] present in test data point [True]
224 Text feature [s3] present in test data point [True]
225 Text feature [transformed] present in test data point [True]
226 Text feature [activated] present in test data point [True]
229 Text feature [free] present in test data point [True]
230 Text feature [advanced] present in test data point [True]
232 Text feature [egf] present in test data point [True]
233 Text feature [her2] present in test data point [True]
234 Text feature [properties] present in test data point [True]
235 Text feature [signals] present in test data point [True]
236 Text feature [genomic] present in test data point [True]
239 Text feature [size] present in test data point [True]
240 Text feature [activate] present in test data point [True]
242 Text feature [membrane] present in test data point [True]
247 Text feature [epithelial] present in test data point [True]
252 Text feature [position] present in test data point [True]
254 Text feature [inhibited] present in test data point [True]
255 Text feature [leading] present in test data point [True]
259 Text feature [presence] present in test data point [True]
261 Text feature [phosphorylated] present in test data point [True]
264 Text feature [long] present in test data point [True]
265 Text feature [000] present in test data point [True]
270 Text feature [observations] present in test data point [True]
272 Text feature [fold] present in test data point [True]
273 Text feature [ras] present in test data point [True]
276 Text feature [increased] present in test data point [True]
277 Text feature [added] present in test data point [True]
278 Text feature [oncogenic] present in test data point [True]
280 Text feature [difference] present in test data point [True]
281 Text feature [regulated] present in test data point [True]
282 Text feature [locus] present in test data point [True]
283 Text feature [constitutive] present in test data point [True]
284 Text feature [approximately] present in test data point [True]
287 Text feature [culture] present in test data point [True]
289 Text feature [distinct] present in test data point [True]
291 Text feature [transforming] present in test data point [True]
292 Text feature [3b] present in test data point [True]
295 Text feature [pathways] present in test data point [True]
297 Text feature [medium] present in test data point [True]
298 Text feature [inhibitor] present in test data point [True]
299 Text feature [mean] present in test data point [True]
300 Text feature [positive] present in test data point [True]
302 Text feature [made] present in test data point [True]
303 Text feature [signaling] present in test data point [True]
304 Text feature [days] present in test data point [True]
306 Text feature [factors] present in test data point [True]
307 Text feature [day] present in test data point [True]
312 Text feature [elevated] present in test data point [True]
314 Text feature [colonies] present in test data point [True]
315 Text feature [colony] present in test data point [True]
316 Text feature [2a] present in test data point [True]
318 Text feature [gfp] present in test data point [True]
320 Text feature [cancers] present in test data point [True]
321 Text feature [insertions] present in test data point [True]
324 Text feature [receptors] present in test data point [True]
326 Text feature [12] present in test data point [True]
327 Text feature [manner] present in test data point [True]
328 Text feature [deletion] present in test data point [True]
329 Text feature [box] present in test data point [True]
330 Text feature [25] present in test data point [True]
331 Text feature [24] present in test data point [True]
332 Text feature [could] present in test data point [True]
333 Text feature [sensitive] present in test data point [True]
334 Text feature [total] present in test data point [True]
335 Text feature [additional] present in test data point [True]
336 Text feature [egfr] present in test data point [True]
Out of the top  500  features  75 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [88]:

```python
test_point_index = 4
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[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],te
st_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3428 0.0621 0.0251 0.1479 0.0624 0.2779 0.0785 0.0006 0.0026]]
Actual Class : 4
-------------------------------------------------
208 Text feature [msh2] present in test data point [True]
210 Text feature [screening] present in test data point [True]
213 Text feature [calculated] present in test data point [True]
215 Text feature [39] present in test data point [True]
216 Text feature [early] present in test data point [True]
221 Text feature [dependent] present in test data point [True]
225 Text feature [region] present in test data point [True]
232 Text feature [2012] present in test data point [True]
239 Text feature [frequency] present in test data point [True]
352 Text feature [next] present in test data point [True]
355 Text feature [age] present in test data point [True]
356 Text feature [59] present in test data point [True]
361 Text feature [46] present in test data point [True]
363 Text feature [chain] present in test data point [True]
371 Text feature [strong] present in test data point [True]
375 Text feature [position] present in test data point [True]
376 Text feature [directly] present in test data point [True]
383 Text feature [19] present in test data point [True]
384 Text feature [furthermore] present in test data point [True]
385 Text feature [hr] present in test data point [True]
388 Text feature [yet] present in test data point [True]
390 Text feature [like] present in test data point [True]
392 Text feature [colorectal] present in test data point [True]
393 Text feature [derived] present in test data point [True]
394 Text feature [moreover] present in test data point [True]
396 Text feature [either] present in test data point [True]
400 Text feature [related] present in test data point [True]
401 Text feature [pathogenic] present in test data point [True]
410 Text feature [established] present in test data point [True]
411 Text feature [sporadic] present in test data point [True]
413 Text feature [alone] present in test data point [True]
415 Text feature [reduced] present in test data point [True]
416 Text feature [general] present in test data point [True]
417 Text feature [criteria] present in test data point [True]
423 Text feature [previous] present in test data point [True]
424 Text feature [six] present in test data point [True]
425 Text feature [tumors] present in test data point [True]
426 Text feature [rather] present in test data point [True]
427 Text feature [figures] present in test data point [True]
428 Text feature [17] present in test data point [True]
434 Text feature [cohort] present in test data point [True]
443 Text feature [individual] present in test data point [True]
444 Text feature [fig] present in test data point [True]
445 Text feature [indeed] present in test data point [True]
451 Text feature [interestingly] present in test data point [True]
455 Text feature [indicated] present in test data point [True]
459 Text feature [nucleotide] present in test data point [True]
460 Text feature [families] present in test data point [True]
463 Text feature [germline] present in test data point [True]
467 Text feature [performed] present in test data point [True]
468 Text feature [carcinoma] present in test data point [True]
471 Text feature [non] present in test data point [True]
479 Text feature [contact] present in test data point [True]
480 Text feature [defined] present in test data point [True]
481 Text feature [therefore] present in test data point [True]
484 Text feature [breast] present in test data point [True]
486 Text feature [exon] present in test data point [True]
488 Text feature [mean] present in test data point [True]
489 Text feature [reverse] present in test data point [True]
492 Text feature [sites] present in test data point [True]
493 Text feature [24] present in test data point [True]
494 Text feature [compared] present in test data point [True]
496 Text feature [analyses] present in test data point [True]
497 Text feature [dna] present in test data point [True]
498 Text feature [ovarian] present in test data point [True]
Out of the top  500  features  65 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [89]:

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

```python
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrea
se=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=Fa
lse,
# 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-con
struction-2/
# --------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibra
tion.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_tfidf, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_tfidf, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
        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(bes
t_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.1910873218549298
for n_estimators = 100 and max depth =  10
Log Loss : 1.2494663802096775
for n_estimators = 200 and max depth =  5
Log Loss : 1.1790728075227173
for n_estimators = 200 and max depth =  10
Log Loss : 1.2398753182436049
for n_estimators = 500 and max depth =  5
Log Loss : 1.178292849055564
for n_estimators = 500 and max depth =  10
Log Loss : 1.234873535891077
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1783582054731865
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2301240455793543
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1768825719540634
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2297195716471416
For values of best estimator =  2000 The train log loss is: 0.8522890057699066
For values of best estimator =  2000 The cross validation log loss is: 1.1768825719540634
For values of best estimator =  2000 The test log loss is: 1.1753792914576762
```

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

In [90]:

```python
# --------------------------------
# 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_decrea
se=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=Fa
lse,
# 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-con
struction-2/
# --------------------------------

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(bes
t_alpha%2)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_tfidf, train_y,cv_x_tfidf,cv_y, clf)
```

```
Log loss : 1.1768825719540632
Number of mis-classified points : 0.41353383458646614
------------------ Confusion matrix --------------------
```



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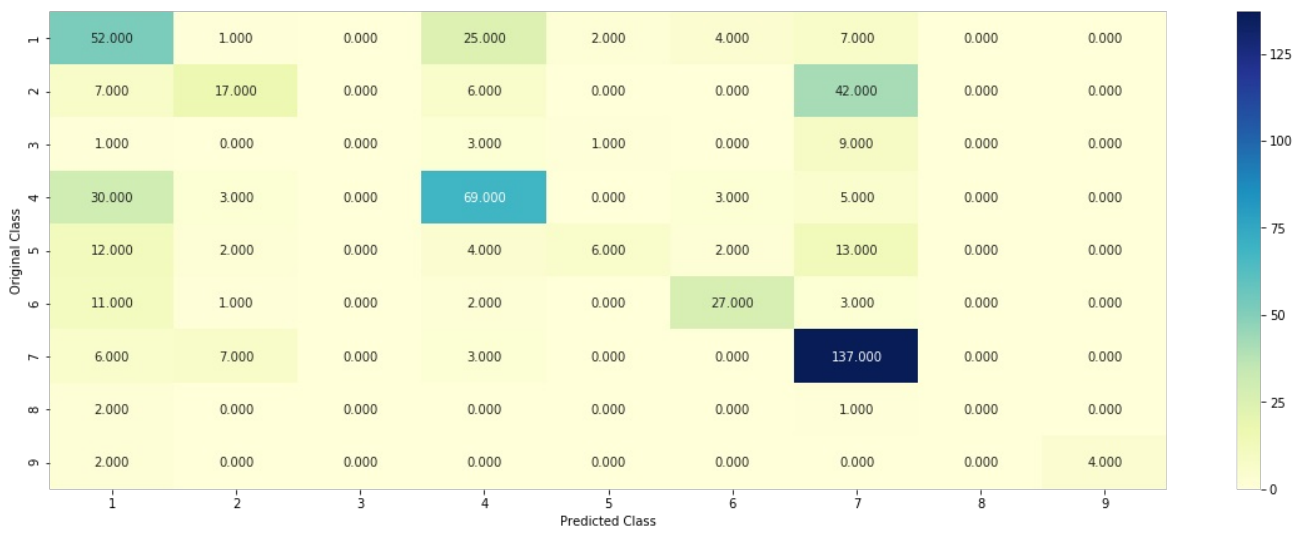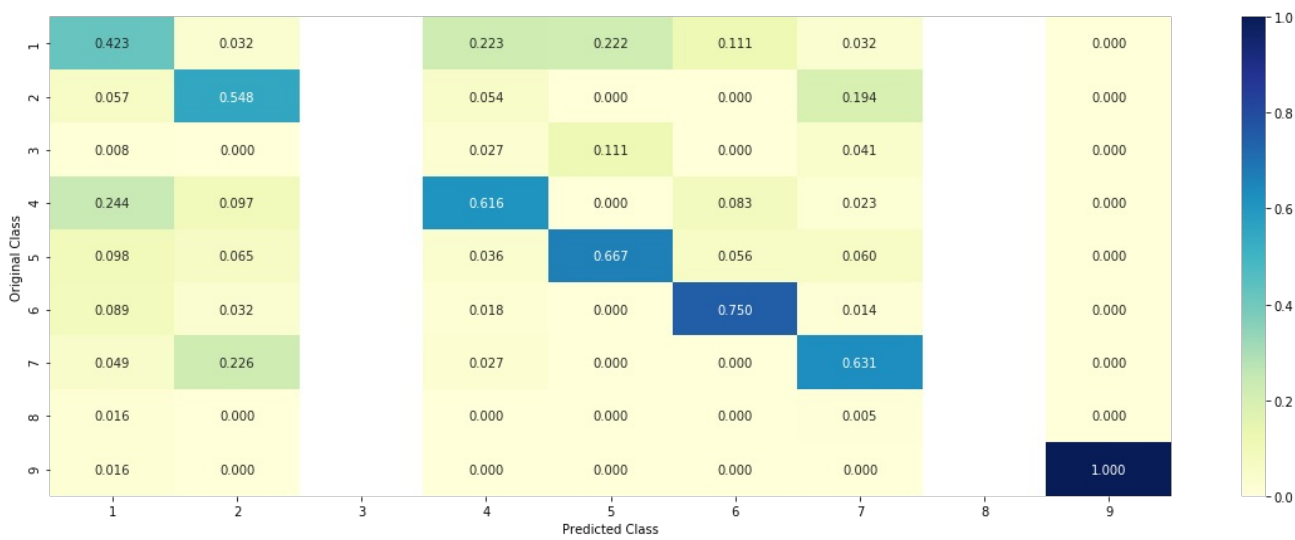


```
------------------ Recall matrix (Row sum=1) --------------------
```
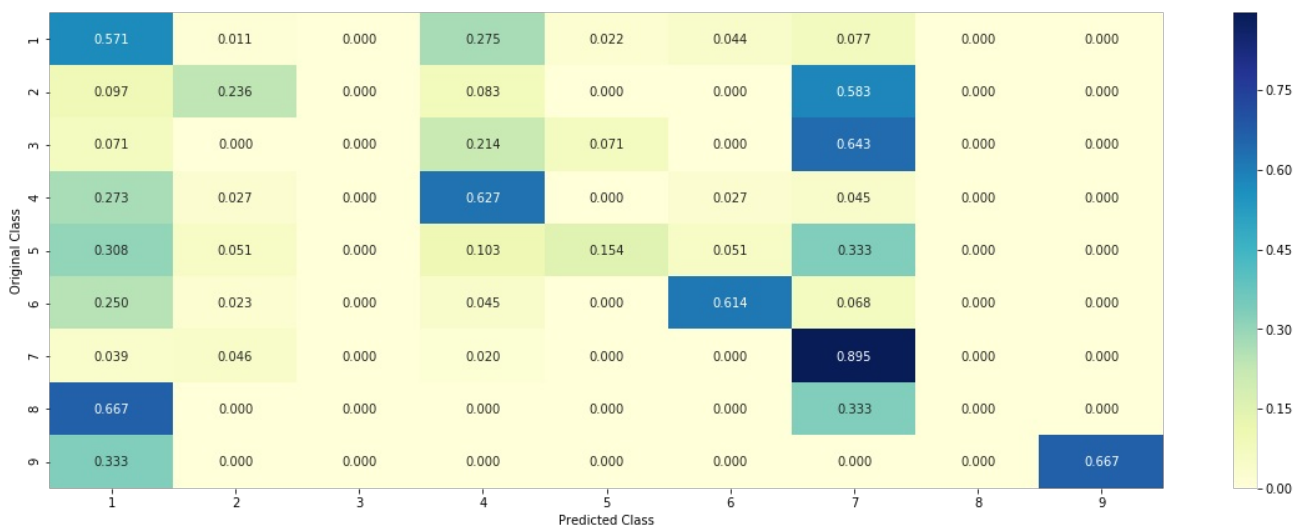

4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

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

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[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: [[0.0383 0.1428 0.0232 0.0342 0.05   0.0457 0.657  0.0039 0.0048]]
Actual Class : 7
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [tyrosine] present in test data point [True]
3 Text feature [activation] present in test data point [True]
4 Text feature [suppressor] present in test data point [True]
5 Text feature [treatment] present in test data point [True]
6 Text feature [function] present in test data point [True]
7 Text feature [inhibitors] present in test data point [True]
8 Text feature [constitutive] present in test data point [True]
9 Text feature [activated] present in test data point [True]
10 Text feature [phosphorylation] present in test data point [True]
11 Text feature [receptor] present in test data point [True]
12 Text feature [missense] present in test data point [True]
13 Text feature [inhibitor] present in test data point [True]
14 Text feature [loss] present in test data point [True]
15 Text feature [therapy] present in test data point [True]
16 Text feature [protein] present in test data point [True]
17 Text feature [deleterious] present in test data point [True]
18 Text feature [oncogenic] present in test data point [True]
20 Text feature [stability] present in test data point [True]
21 Text feature [variants] present in test data point [True]
22 Text feature [treated] present in test data point [True]
23 Text feature [f3] present in test data point [True]
24 Text feature [pten] present in test data point [True]
25 Text feature [trials] present in test data point [True]
27 Text feature [cells] present in test data point [True]
28 Text feature [cell] present in test data point [True]
29 Text feature [proteins] present in test data point [True]
30 Text feature [ligand] present in test data point [True]
32 Text feature [signaling] present in test data point [True]
34 Text feature [functional] present in test data point [True]
35 Text feature [neutral] present in test data point [True]
36 Text feature [inhibited] present in test data point [True]
37 Text feature [expression] present in test data point [True]
38 Text feature [constitutively] present in test data point [True]
39 Text feature [therapeutic] present in test data point [True]
41 Text feature [transforming] present in test data point [True]
42 Text feature [patients] present in test data point [True]
43 Text feature [ba] present in test data point [True]
44 Text feature [growth] present in test data point [True]
46 Text feature [drug] present in test data point [True]
48 Text feature [inhibition] present in test data point [True]
52 Text feature [kinases] present in test data point [True]
55 Text feature [akt] present in test data point [True]
56 Text feature [classified] present in test data point [True]
60 Text feature [activate] present in test data point [True]
63 Text feature [downstream] present in test data point [True]
64 Text feature [phosphatase] present in test data point [True]
66 Text feature [response] present in test data point [True]
67 Text feature [predicted] present in test data point [True]
68 Text feature [advanced] present in test data point [True]
69 Text feature [57] present in test data point [True]
71 Text feature [proliferation] present in test data point [True]
75 Text feature [receptors] present in test data point [True]
76 Text feature [tagged] present in test data point [True]
78 Text feature [clinical] present in test data point [True]
79 Text feature [based] present in test data point [True]
81 Text feature [expected] present in test data point [True]
82 Text feature [factor] present in test data point [True]
83 Text feature [sequence] present in test data point [True]
84 Text feature [sensitivity] present in test data point [True]
85 Text feature [information] present in test data point [True]
87 Text feature [survival] present in test data point [True]
88 Text feature [catalytic] present in test data point [True]
89 Text feature [ring] present in test data point [True]
90 Text feature [oncogene] present in test data point [True]
91 Text feature [variant] present in test data point [True]
94 Text feature [dna] present in test data point [True]
97 Text feature [stimulation] present in test data point [True]
98 Text feature [effective] present in test data point [True]
99 Text feature [phospho] present in test data point [True]
Out of the top  100  features  71 are present in query point
```

4.5.3.2. Inorrectly Classified point

```
test_point_index = 4
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[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 : 6
Predicted Class Probabilities: [[0.2939 0.062  0.0152 0.1811 0.0851 0.3123 0.0379 0.0067 0.0057]]
Actuall Class : 4
--------------------------------------------------
6 Text feature [function] present in test data point [True]
12 Text feature [missense] present in test data point [True]
14 Text feature [loss] present in test data point [True]
16 Text feature [protein] present in test data point [True]
17 Text feature [deleterious] present in test data point [True]
28 Text feature [cell] present in test data point [True]
31 Text feature [repair] present in test data point [True]
34 Text feature [functional] present in test data point [True]
42 Text feature [patients] present in test data point [True]
53 Text feature [pathogenic] present in test data point [True]
56 Text feature [classified] present in test data point [True]
57 Text feature [efficacy] present in test data point [True]
61 Text feature [ovarian] present in test data point [True]
69 Text feature [57] present in test data point [True]
78 Text feature [clinical] present in test data point [True]
79 Text feature [based] present in test data point [True]
81 Text feature [expected] present in test data point [True]
85 Text feature [information] present in test data point [True]
92 Text feature [history] present in test data point [True]
94 Text feature [dna] present in test data point [True]
96 Text feature [splice] present in test data point [True]
Out of the top  100  features  21 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

```
# ---------------------------------
# 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_decrea
se=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=Fa
lse,
# 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-con
struction-2/
# ---------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibra
tion.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:
#-----------------------------------
```

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

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

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

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.052207985924622
for n_estimators = 10 and max depth =  3
Log Loss : 1.5767312616593399
for n_estimators = 10 and max depth =  5
Log Loss : 1.2546402153863823
for n_estimators = 10 and max depth =  10
Log Loss : 1.7416277218527452
for n_estimators = 50 and max depth =  2
Log Loss : 1.6311445330087273
for n_estimators = 50 and max depth =  3
Log Loss : 1.3423302619422708
for n_estimators = 50 and max depth =  5
Log Loss : 1.2407255928374676
for n_estimators = 50 and max depth =  10
Log Loss : 1.6444106206341713
for n_estimators = 100 and max depth =  2
Log Loss : 1.4928391368877068
for n_estimators = 100 and max depth =  3
Log Loss : 1.3465007249200969
for n_estimators = 100 and max depth =  5
Log Loss : 1.2494869593842424
for n_estimators = 100 and max depth =  10
Log Loss : 1.6762647488526141
for n_estimators = 200 and max depth =  2
Log Loss : 1.52713356521443
for n_estimators = 200 and max depth =  3
Log Loss : 1.3469399875929744
for n_estimators = 200 and max depth =  5
Log Loss : 1.2945424581406002
for n_estimators = 200 and max depth =  10
Log Loss : 1.6991623826518398
for n_estimators = 500 and max depth =  2
Log Loss : 1.5352605109366642
for n_estimators = 500 and max depth =  3
Log Loss : 1.383103764079479
for n_estimators = 500 and max depth =  5
Log Loss : 1.2926885781745625
for n_estimators = 500 and max depth =  10
Log Loss : 1.7192727544291841
for n_estimators = 1000 and max depth =  2
Log Loss : 1.5336170161883964
for n_estimators = 1000 and max depth =  3
Log Loss : 1.4040668330643817
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2956633910463777
for n_estimators = 1000 and max depth =  10
Log Loss : 1.707107323765178
For values of best alpha =  50 The train log loss is: 0.05978967122651576
For values of best alpha =  50 The cross validation log loss is: 1.2407255928374679
For values of best alpha =  50 The test log loss is: 1.2572802465323227
```

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

```python
# ---------------------------------
# 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_decrea
se=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=Fa
lse,
# 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-con
struction-2/
# ---------------------------------

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

```
Log loss : 1.240725592837468
Number of mis-classified points : 0.4191729323308271
------------------- Confusion matrix --------------------
```



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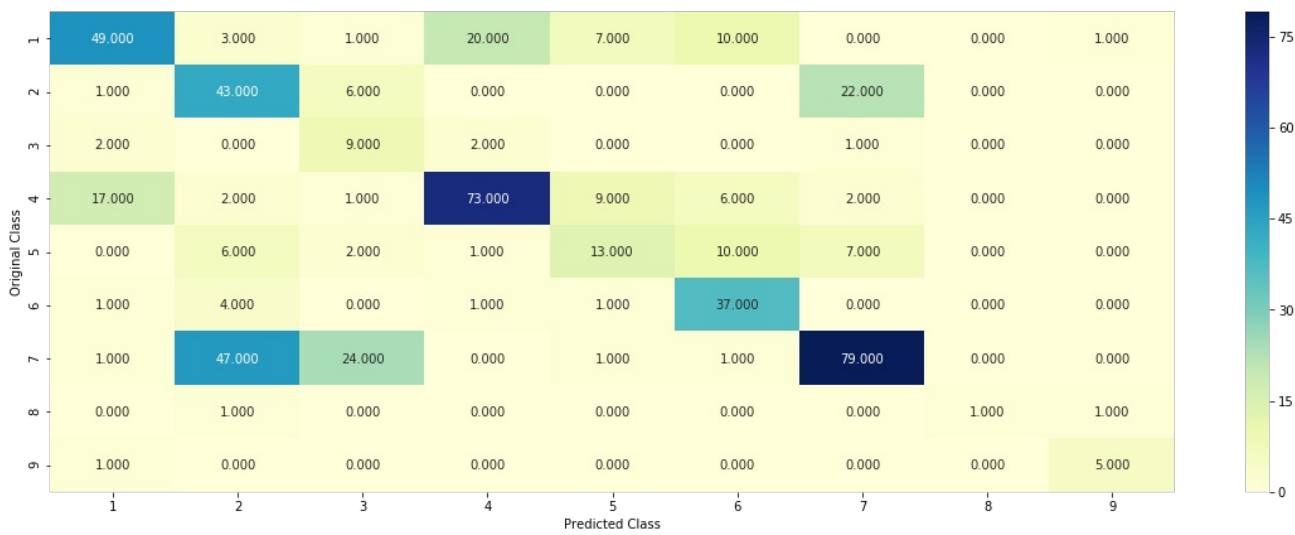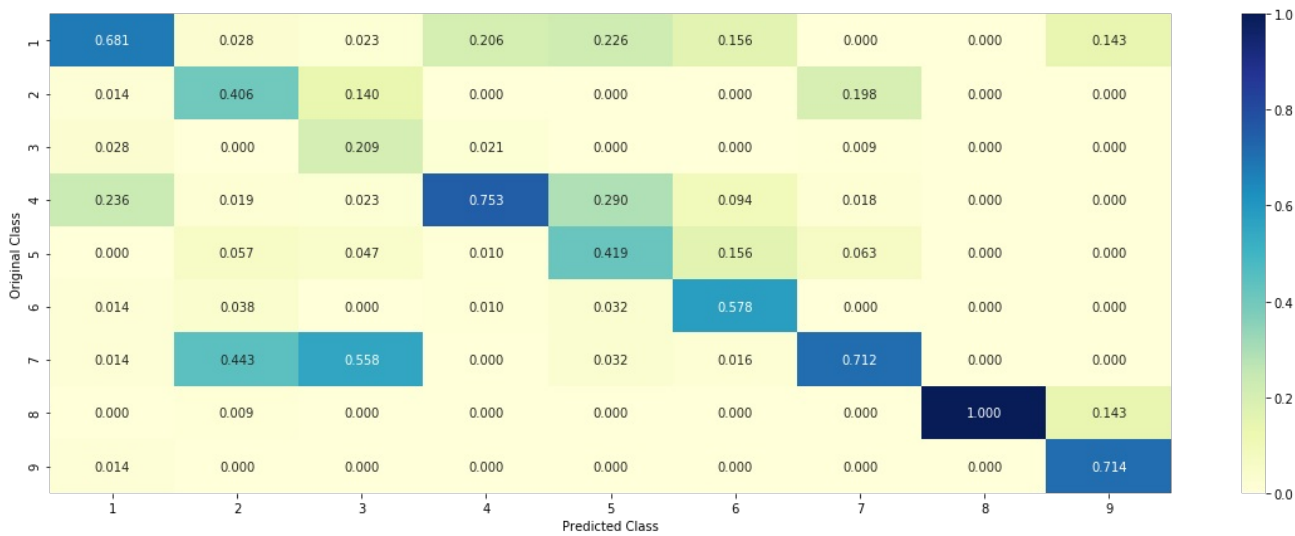


```
------------------- Recall matrix (Row sum=1) --------------------
```
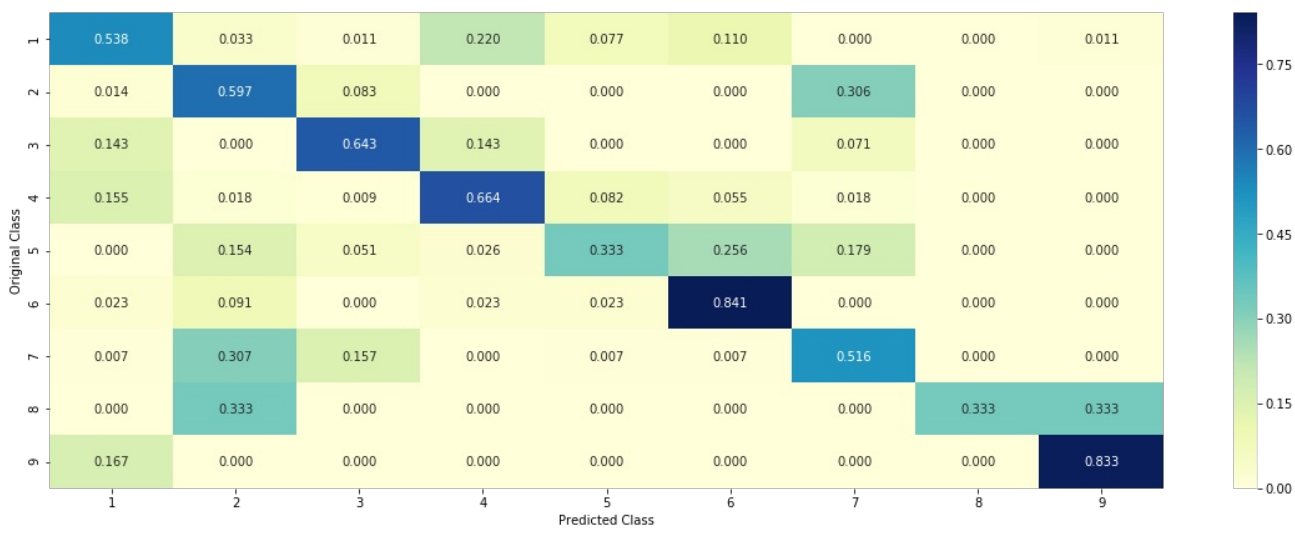

4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

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


test_point_index = 1
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index].re
shape(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: [[0.0202 0.2562 0.2539 0.0386 0.0367 0.0543 0.3061 0.0182 0.0157]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

```
test_point_index = 4
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].re
shape(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 : 1
Predicted Class Probabilities: [[0.3614 0.0141 0.1261 0.0999 0.1552 0.1981 0.0065 0.0153 0.0234]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDCla
ssifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=N
one,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0
.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/generat
ed/sklearn.svm.SVC.html
# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
# cache size=200, class weight=None, verbose=False, max iter=-1, decision function shape='ovr', random state=None
```

```python
#  cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None
)

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


# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/generat
ed/sklearn.ensemble.RandomForestClassifier.html
# -------------------------------
# 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_decrea
se=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=Fa
lse,
# 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-con
struction-2/
# -------------------------------


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

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


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

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

```
Logistic Regression :  Log Loss: 1.03
Support vector machines : Log Loss: 1.73
Naive Bayes : Log Loss: 1.15
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.177
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.029
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.484
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.133
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.328
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.781
```

4.7.2 testing the model with the best hyper parameters

```python
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True)
sclf.fit(train_x_tfidf, train_y)

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

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

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

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

```
Log loss (train) on the stacking classifier : 0.5338470622848455
Log loss (CV) on the stacking classifier : 1.1325718256271302
Log loss (test) on the stacking classifier : 1.1358570042252494
Number of missclassified point : 0.37293233082706767
-------------------- Confusion matrix --------------------
```



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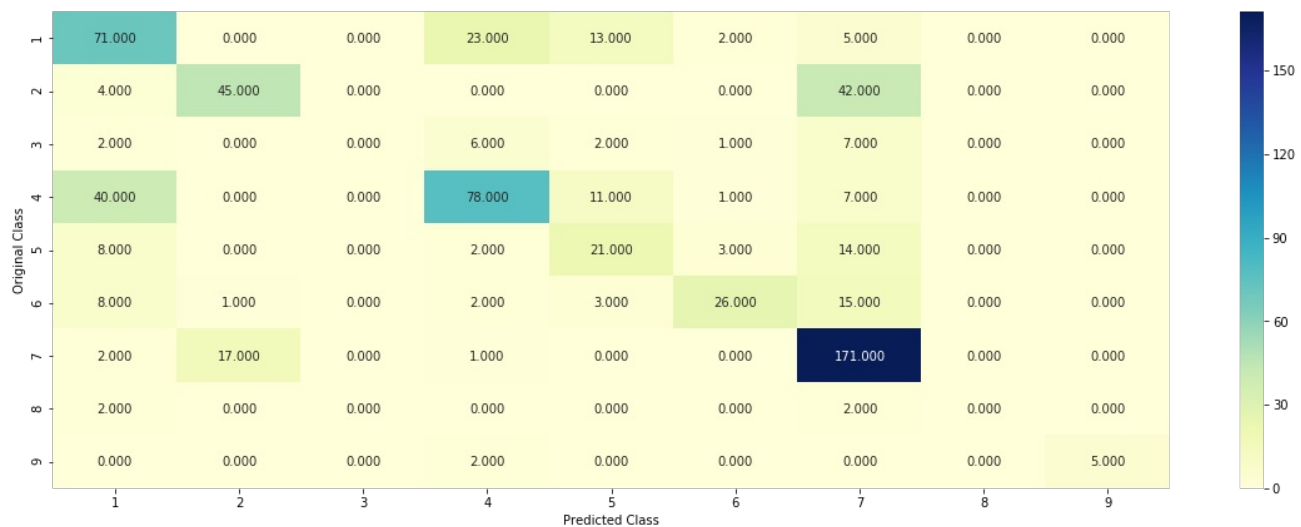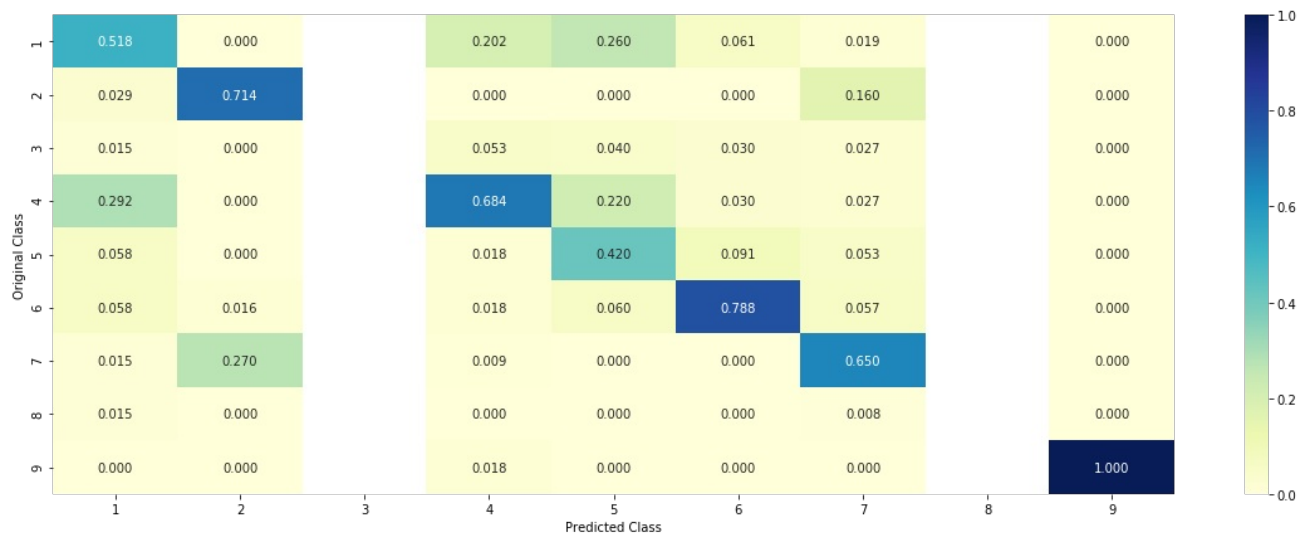


```
-------------------- Recall matrix (Row sum=1) --------------------
```
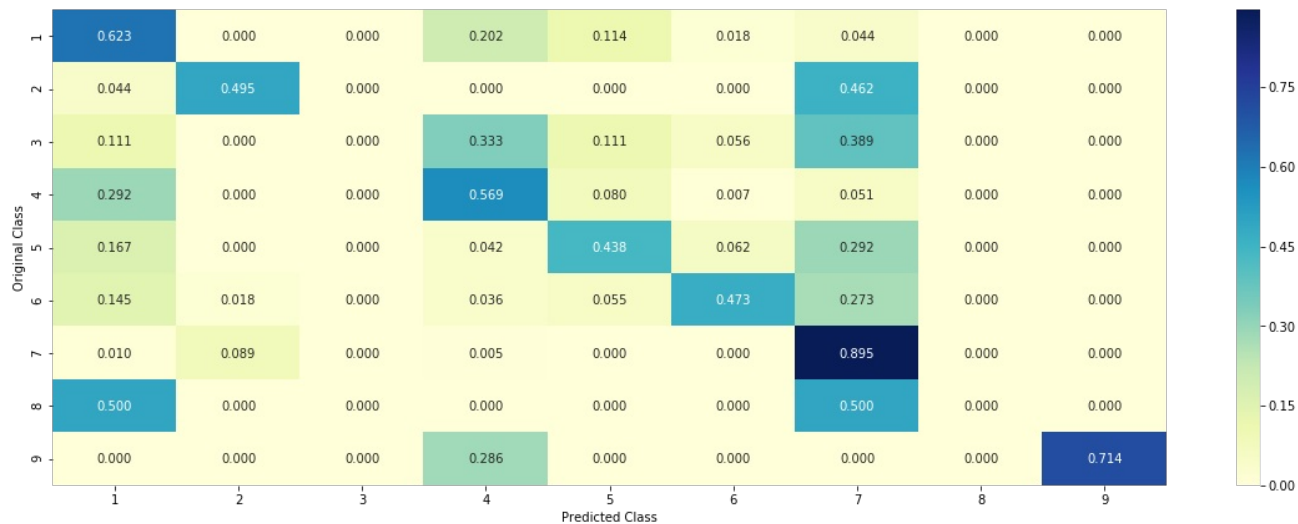
4.7.3 Maximum Voting classifier

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

```
Log loss (train) on the VotingClassifier : 0.8326888022807533
Log loss (CV) on the VotingClassifier : 1.1553390107726362
Log loss (test) on the VotingClassifier : 1.1732260715931448
Number of missclassified point : 0.3744360902255639
-------------------- Confusion matrix --------------------
```
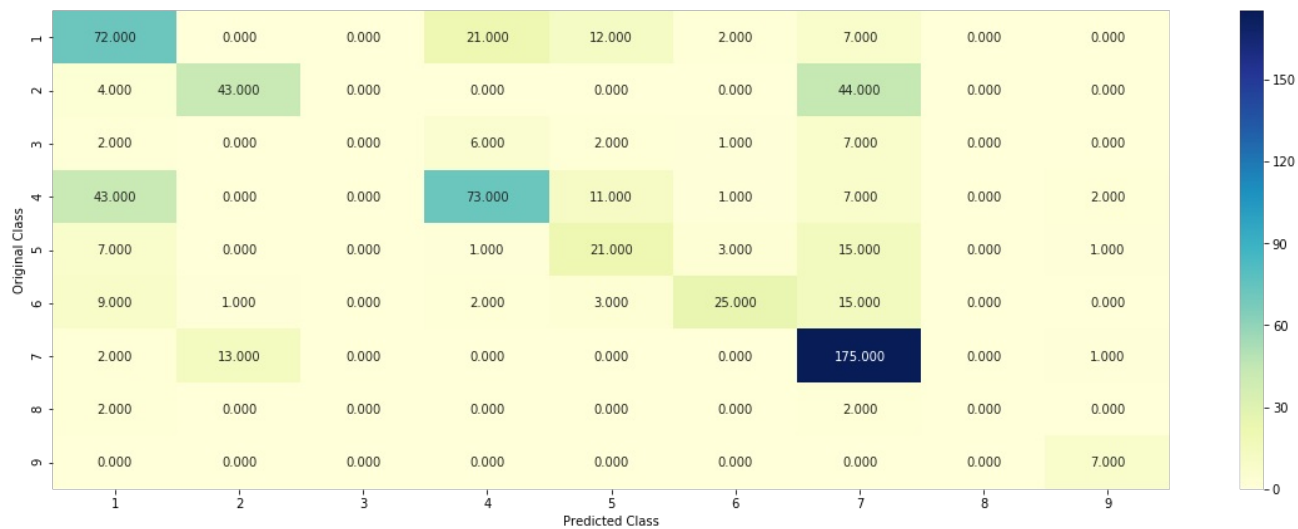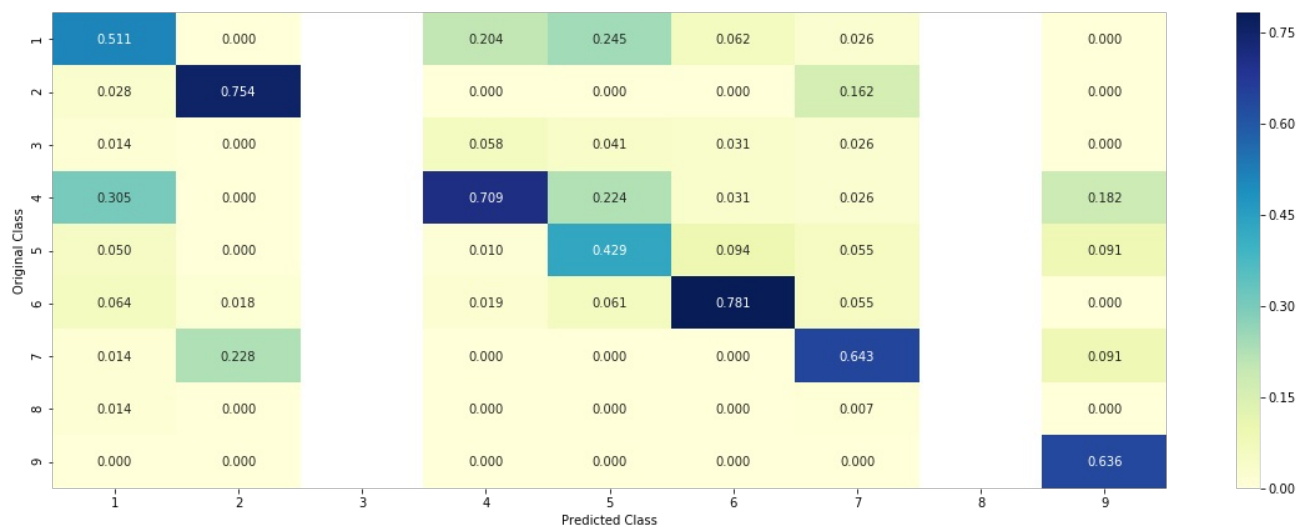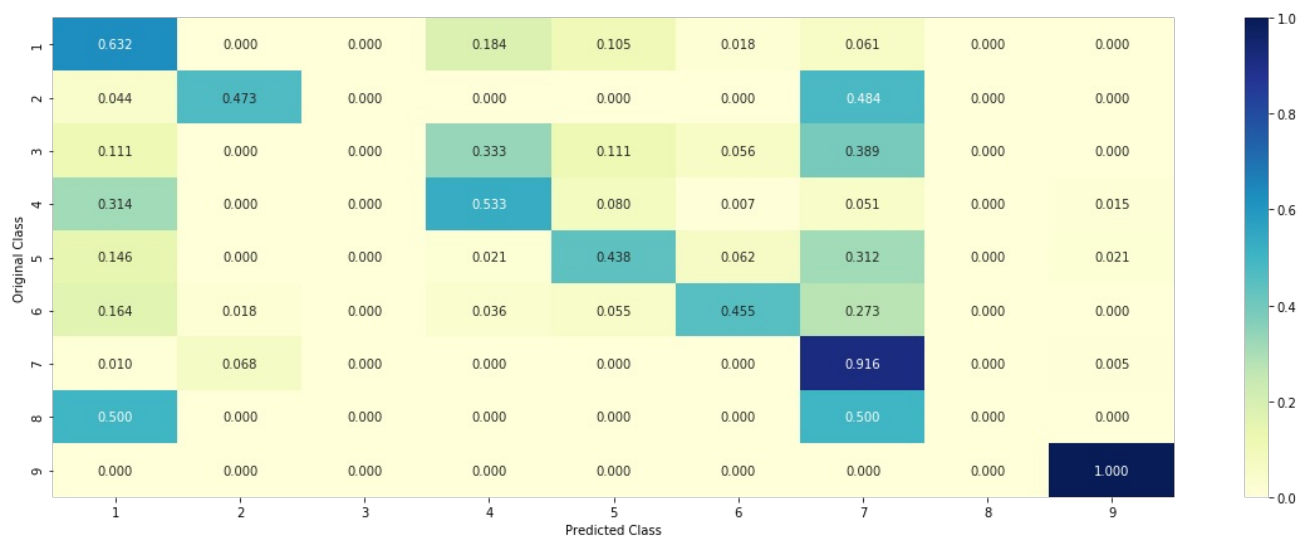


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```

Original Class	1	2	3	4	5	6	7	8	9
1	0.511	0.000		0.204	0.245	0.062	0.026		0.000
2	0.028	0.754		0.000	0.000	0.000	0.162		0.000
3	0.014	0.000		0.058	0.041	0.031	0.026		0.000
4	0.305	0.000		0.709	0.224	0.031	0.026		0.182
5	0.050	0.000		0.010	0.429	0.094	0.055		0.091
6	0.064	0.018		0.019	0.061	0.781	0.055		0.000
7	0.014	0.228		0.000	0.000	0.000	0.643		0.091
8	0.014	0.000		0.000	0.000	0.000	0.007		0.000
9	0.000	0.000		0.000	0.000	0.000	0.000		0.636

Predicted Class

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

Original Class	1	2	3	4	5	6	7	8	9
1	0.632	0.000	0.000	0.184	0.105	0.018	0.061	0.000	0.000
2	0.044	0.473	0.000	0.000	0.000	0.000	0.484	0.000	0.000
3	0.111	0.000	0.000	0.333	0.111	0.056	0.389	0.000	0.000
4	0.314	0.000	0.000	0.533	0.080	0.007	0.051	0.000	0.015
5	0.146	0.000	0.000	0.021	0.438	0.062	0.312	0.000	0.021
6	0.164	0.018	0.000	0.036	0.055	0.455	0.273	0.000	0.000
7	0.010	0.068	0.000	0.000	0.000	0.000	0.916	0.000	0.005
8	0.500	0.000	0.000	0.000	0.000	0.000	0.500	0.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000

Predicted Class

What we did so far?

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

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

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

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

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

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

Now, it's time for use to see what are the top most frequently occurring words for a given class. I have used word clouds for this purpose. The word clouds can straight away give us relevant keywords which helps us understand what type of words are mots common for every class. For example we see keywords like "breast cancer", "lung cancer", "ovarian cancer", "gene mutation", "tyrosie

kinase", "egfr mutation","amino acids" and so on, belonging to each of the classes. Here I have given just a few examples. We also see that there are some classes which talks about some particular variation/mutant type more frequently than others.

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

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

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

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

Typically for high dimensional data, logistic regression and linear SVMs work very well. We can use KNN, Naive Bayes for lower dimensional data representations. In a seperate experiment we can also try models like XGBoost and LightGBM to extract features. Anyway, as the first step we need to encode the text feature into numerical data. How? We have two ways - a simple one hot encoding of the text corpus using Bag of Words approach and a response coding approach. We will choose the appropriate featurization based on the ML model we use. For this problem of multi-class classification with categorical features, one-hot encoding is better for Logistic regression while response coding is better for Random Forests.

Very Important note about response coding: Avoiding response leakage.

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

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

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

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

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

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

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

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

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

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

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

Comparing model performances:

TF-IDF Feature representations:

In [19]:

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

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

BOW (bigrams) Feature representations:

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

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

print(table)
```

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

Feature Engineering Results:

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

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

print(table)
```

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