Personalized cancer diagnosis

Note:Applying Logistic regression with CountVectorizer Features, including both unigrams and bigrams

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

ID,Text
0||Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity revealed. Previous work has shown that CDK10 silencing increases ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The precise mechanisms by which CDK10 modulates ETS2 activity, and more generally the functions of CDK10, remain elusive. Here we demonstrate that CDK10 is a cyclin-dependent kinase by identifying cyclin M as an activating cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR syndrome, a human developmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malformations. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and in conferring tamoxifen resistance to breast cancer cells. CDK10/cyclin M phosphorylates ETS2 in vitro, and in cells it positively controls ETS2 degradation by the proteasome. ETS2 protein levels are increased in cells derived from a STAR patient, and this increase is attributable to decreased cyclin M levels.

Altogether, our results reveal an additional regulatory mechanism for ETS2, which plays key roles in cancer and development. They also shed light on the molecular mechanisms underlying STAR syndrome.Cyclin-dependent kinases (CDKs) play a pivotal role in the control of a number of fundamental cellular processes (1). The human genome contains 21 genes encoding proteins that can be considered as members of the CDK family owing to their sequence similarity with bona fide CDKs, those known to be activated by cyclins (2). Although discovered almost 20 y ago (3, 4), CDK10 remains one of the two CDKs without an identified cyclin partner. This knowledge gap has largely impeded the exploration of its biological functions. CDK10 can act as a positive cell cycle regulator in some cells (5, 6) or as a tumor suppressor in others (7, 8). CDK10 interacts with the ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity through an unknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK pathway, and induces resistance of MCF7 cells to tamoxifen (6). ...

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
```

```python
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.cross_validation import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

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

```
C:\Users\Admin\Anaconda3\lib\site-packages\sklearn\cross_validation.py:
41: DeprecationWarning: This module was deprecated in version 0.18 in f
avor of the model_selection module into which all the refactored classe
s and functions are moved. Also note that the interface of the new CV i
terators are different from that of this module. This module will be re
moved in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:
```
data = pd.read_csv('training_variants')
print('Number of data points : ', data.shape[0])
print('Number of features : ', data.shape[1])
print('Features : ', data.columns.values)
data.head()
```

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

Out[2]:

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

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

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

3.1.2. Reading Text Data

In [3]:
```
# note the seprator in this file
data_text =pd.read_csv("training_text",sep="\|\|",engine="python",names
=["ID","TEXT"],skiprows=1)
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

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

Out[3]:

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

3.1.3. Preprocessing of text

In [4]:

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


def nlp_preprocessing(total_text, index, column):
    if type(total_text) is not int:
        string = ""
        # replace every special char with space
        total_text = re.sub('[^a-zA-Z0-9\n]', ' ', total_text)
        # replace multiple spaces with single space
        total_text = re.sub('\s+',' ', total_text)
        # converting all the chars into lower-case.
        total_text = total_text.lower()
        for word in total_text.split():
        # if the word is a not a stop word then retain that word from t
he data
            if not word in stop_words:
                string += word + " "

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

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

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 210.62841338088228 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...

	ID	Gene	Variation	Class	TEXT
3	3	CBL	N454D	3	recent evidence demonstrated acquired uniparen...
4	4	CBL	L399V	4	oncogenic mutations monomeric casitas b lineag...

In [7]:
```
result[result.isnull().any(axis=1)]
```

Out[7]:

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

In [8]:
```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result
['Variation']
```

In [9]:
```
result[result['ID']==1109]
```

Out[9]:

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

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

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

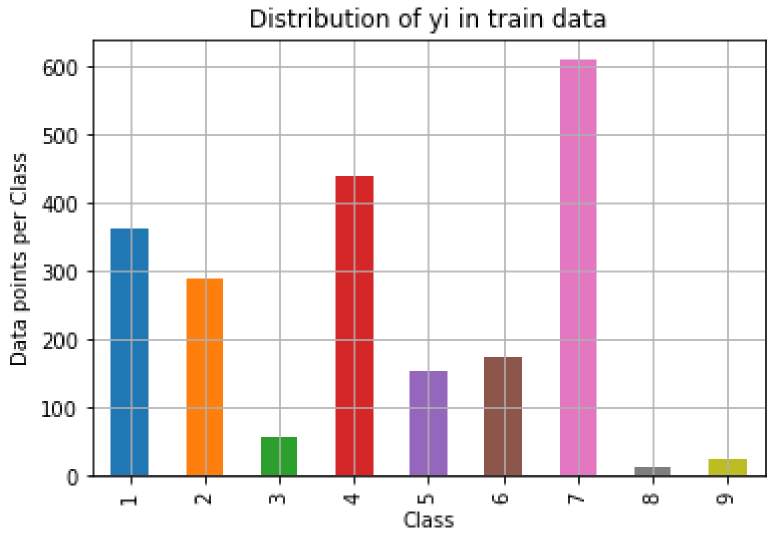
We split the data into train, test and cross validation data sets, preserving the ratio of class distribution in the original data set

```
In [11]: print('Number of data points in train data:', train_df.shape[0])
         print('Number of data points in test data:', test_df.shape[0])
         print('Number of data points in cross validation data:', cv_df.shape[0
         ])
```

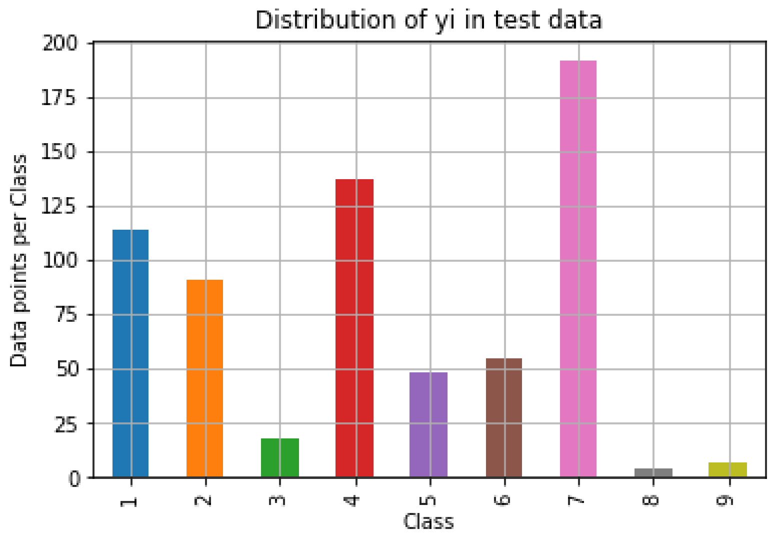
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

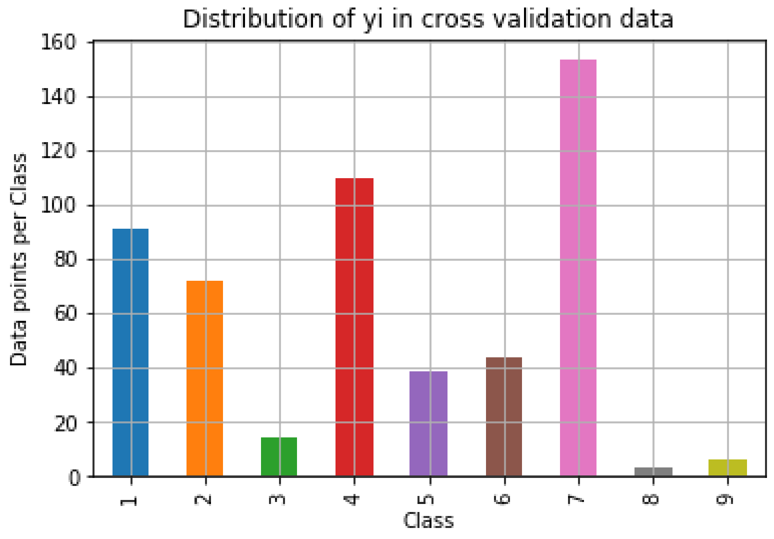
```
In [12]: # it returns a dict, keys as class labels and values as the number of d
         ata points in that class
         train_class_distribution = train_df['Class'].value_counts().sortlevel()
         test_class_distribution = test_df['Class'].value_counts().sortlevel()
         cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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

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


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

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

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

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



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



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



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

3.2 Prediction using a 'Random' Model

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

In [13]:

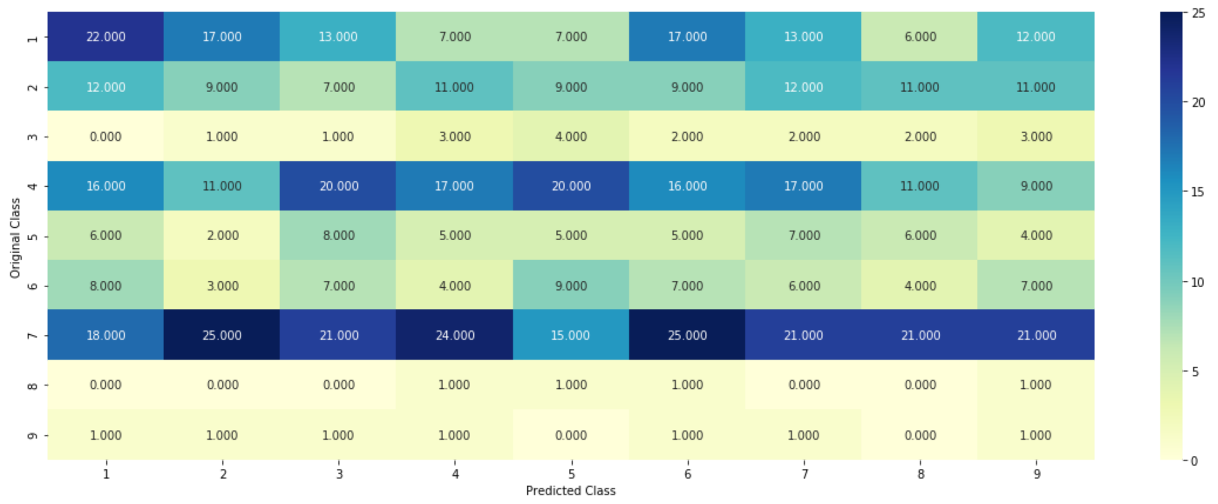
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of cl
ass i are predicted class j
```

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

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

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

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

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

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

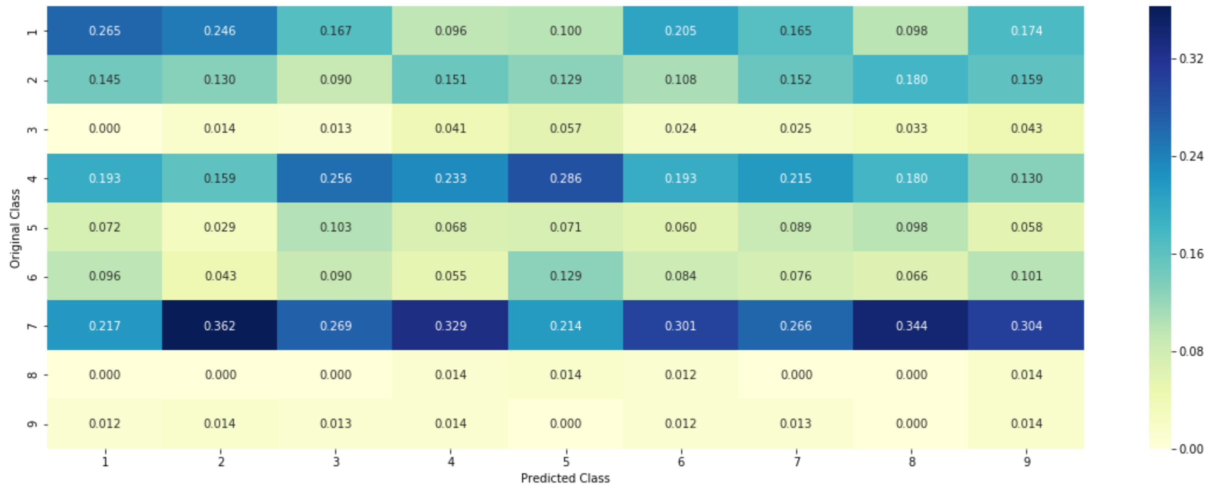
In [14]:

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

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


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

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

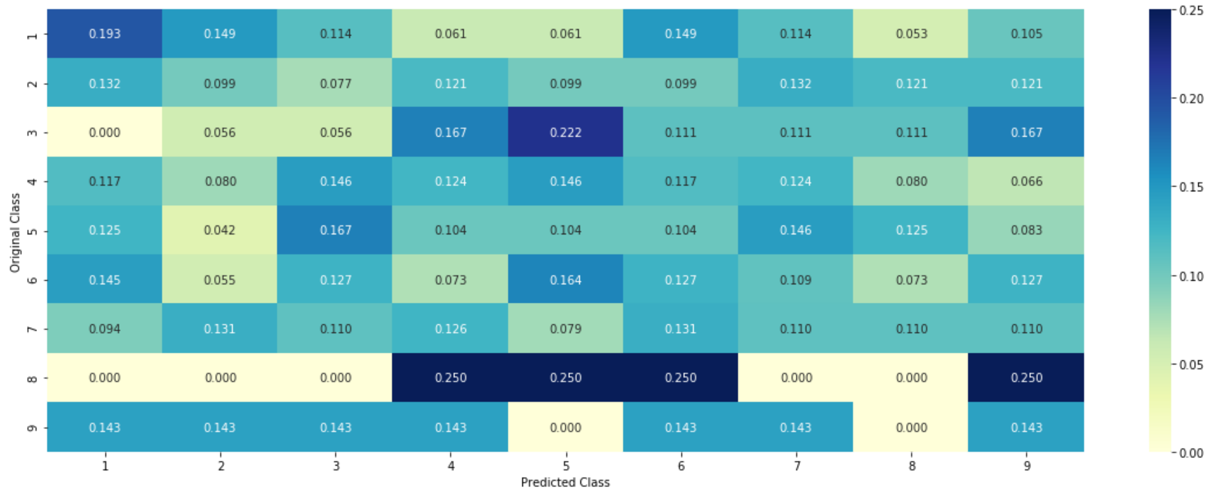
```
Log loss on Cross Validation Data using Random Model 2.5405658663588686
Log loss on Test Data using Random Model 2.471530117488993
-------------------- Confusion matrix --------------------
```



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



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



3.3 Univariate Analysis

In [15]:
```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feat
ure in train data dataframe
# build a vector (1*9) , the first element = (number of times it occure
d in class1 + 10*alpha / number of time it occurred in total data+90*al
pha)
# gv_dict is like a look up table, for every gene it store a (1*9) repr
esentation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_f
ea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

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

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

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

            cls_cnt = train_df.loc[(train_df['Class']==k) & (train_df[f
eature]==i)]
```

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181
818181818177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.0378787
8787878788, 0.03787878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224
489795918366, 0.27040816326530615, 0.061224489795918366, 0.066326530612
244902, 0.051020408163265307, 0.051020408163265307, 0.05612244897959183
7],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625,
 0.068181818181818177, 0.068181818181818177, 0.0625, 0.3465909090909091
2, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.06060
6060606060608, 0.078787878787878782, 0.1393939393939394, 0.345454545454
54546, 0.060606060606060608, 0.060606060606060608, 0.06060606060606060
8],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.06918
2389937106917, 0.46540880503144655, 0.075471698113207544, 0.06289308176
1006289, 0.069182389937106917, 0.062893081761006289, 0.0628930817610062
89],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.0728476
82119205295, 0.072847682119205295, 0.066225165562913912, 0.066225165562
913912, 0.27152317880794702, 0.066225165562913912, 0.06622516556291391
2],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333
333333333334, 0.073333333333333334, 0.093333333333333338, 0.08000000000
0000002, 0.29999999999999999, 0.066666666666666666, 0.066666666666666666
6],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

In [16]:
```
unique_genes = train_df['Gene'].value_counts()
```

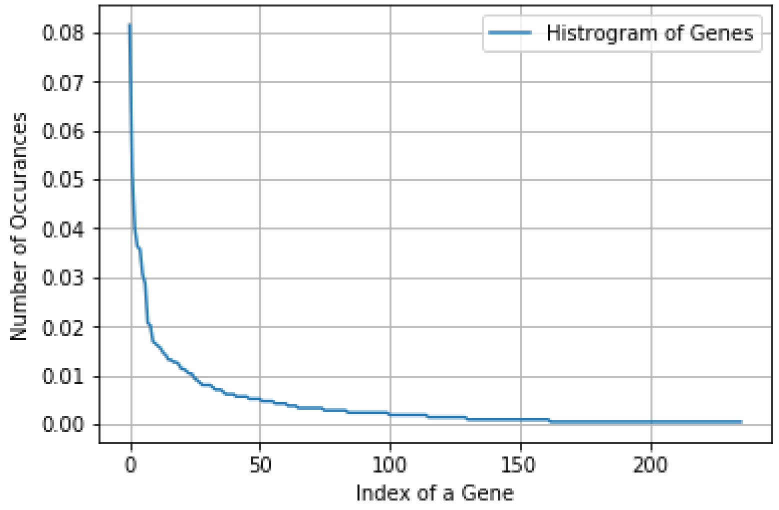
```
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 236
BRCA1     173
TP53      109
EGFR       85
BRCA2      77
PTEN       76
BRAF       65
KIT        61
ERBB2      44
ALK        43
PDGFRA     36
Name: Gene, dtype: int64
```

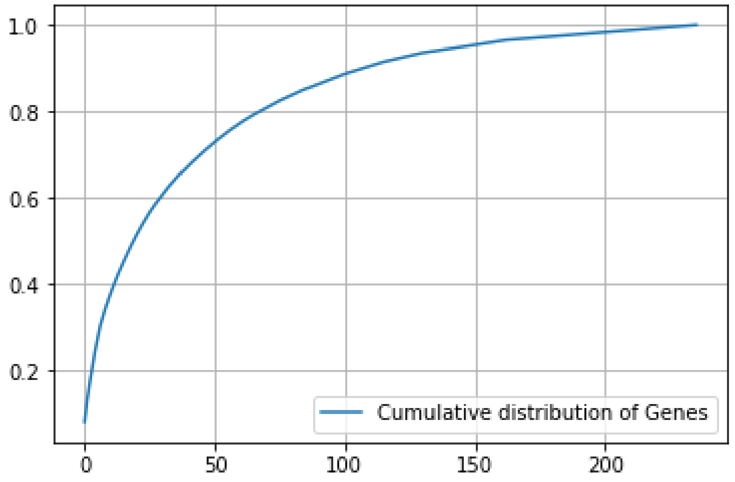
In [17]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of
 genes in the train data, and they are distibuted as follows",)
```

```
Ans: There are 236 different categories of genes in the train data, and
they are distibuted as follows
```

In [18]:
```
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histrogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



```
In [19]: c = np.cumsum(h)
         plt.plot(c,label='Cumulative distribution of Genes')
         plt.grid()
         plt.legend()
         plt.show()
```



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

In [20]:
```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gen
e", train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gen
e", test_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene",
 cv_df))
```

In [21]:
```
print("train_gene_feature_responseCoding is converted feature using res
pone coding method. The shape of gene feature:", train_gene_feature_res
ponseCoding.shape)
```

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

In [22]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer(ngram_range=(1,2))
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_d
f['Gene'])
```

```
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gen
e'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [23]:
```
train_df['Gene'].head()
```

Out[23]:
```
2188      PTEN
2615     BRCA1
1989    MAP2K1
2490     BRCA1
1382     FGFR1
Name: Gene, dtype: object
```

In [24]:
```
gene_vectorizer.get_feature_names()
```

Out[24]:
```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid2',
 'arid5b',
 'asxl1',
 'asxl2',
 'atm',
 'atrx',
 'aurka',
 'axin1',
 'axl',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk6',
 'cdk8',
 'cdkn1a',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cdkn2c',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctnnb1',
 'ddr2',
 'dicer1',
 'dnmt3a',
 'dnmt3b',
 'dusp4',
 'egfr',
 'eif1ax',
 'elf3',
 'ep300',
 'epas1',
 'epcam',
 'erbb2',
 'erbb3',
 'erbb4',
 'ercc2',
 'ercc3',
 'ercc4',
 'erg',
 'errfi1',
 'esr1',
 'etv1',
 'etv6',
 'ewsr1',
 'ezh2',
 'fam58a',
 'fanca',
 'fat1',
 'fbxw7',
 'fgf19',
 'fgf4',
 'fgfr1',
 'fgfr2',
 'fgfr3',
 'fgfr4',
 'flt3',
 'foxa1',
 'foxl2',
 'foxo1',
 'foxp1',
 'fubp1',
 'gata3',
 'gli1',
 'gna11',
 'gnas',
 'h3f3a',
 'hla',
 'hnf1a',
 'hras',
 'idh1',
 'idh2',
 'ikbke',
 'inpp4b',
 'jak1',
 'jak2',
 'jun',
 'kdm5a',
 'kdm5c',
 'kdm6a',
 'kdr',
 'keap1',
 'kit',
 'kmt2b',
 'kmt2c',
 'knstrn',
 'kras',
 'map2k1',
 'map2k2',
 'map2k4',
 'map3k1',
 'mapk1',
 'mdm2',
 'mdm4',
 'med12',
 'mef2b',
 'men1',
 'met',
 'mga',
 'mlh1',
 'mpl',
 'msh2',
 'msh6',
 'mtor',
 'myc',
 'mycn',
 'myd88',
 'myod1',
 'ncor1',
 'nf1',
 'nf2',
 'nfe2l2',
 'nfkbia',
 'nkx2',
 'notch1',
 'notch2',
 'npm1',
 'nras',
 'nsd1',
 'ntrk1',
 'ntrk2',
 'ntrk3',
 'nup93',
 'pak1',
 'pax8',
 'pbrm1',
 'pdgfra',
 'pdgfrb',
 'pik3ca',
 'pik3cb',
 'pik3cd',
 'pik3r1',
 'pik3r2',
 'pik3r3',
 'pim1',
 'pms2',
 'pole',
 'ppp2r1a',
 'ppp6c',
 'prdm1',
 'ptch1',
 'pten',
 'ptpn11',
 'ptprd',
 'ptprt',
 'rab35',
 'rac1',
 'rad21',
 'rad50',
 'rad51c',
 'rad51d',
 'rad54l',
 'raf1',
 'rara',
 'rasa1',
 'rb1',
 'rbm10',
 'ret',
 'rheb',
 'rhoa',
 'rictor',
 'rit1',
 'rnf43',
 'ros1',
 'runx1',
 'rxra',
 'sdhb',
 'sdhc',
 'setd2',
 'sf3b1',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'srsf2',
 'stag2',
 'stat3',
 'stk11',
 'tcf7l2',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'whsc1',
 'whsc1l1',
 'xpo1',
 'xrcc2',
 'yap1']
```

In [25]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature:", train_gene_feature_onehotCoding.shape)
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature: (2124, 235)
```

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

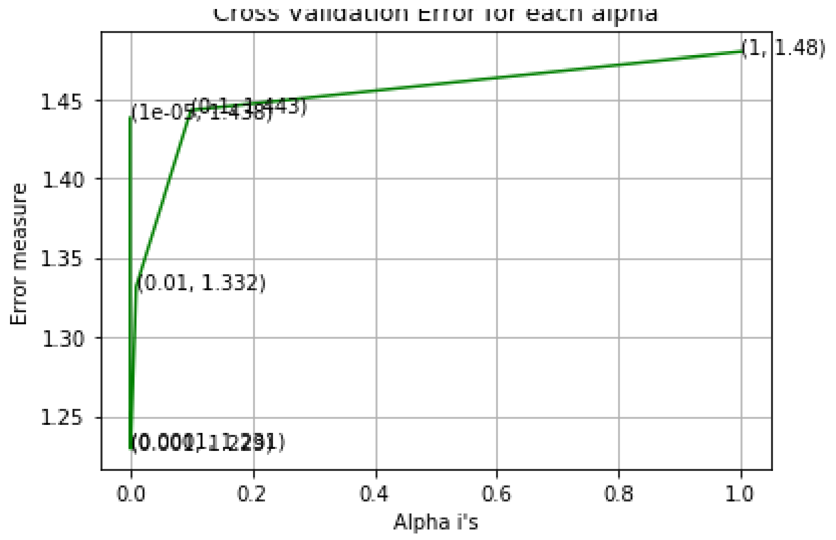
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [26]:
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

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

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

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



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

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

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

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

```
For values of alpha =  1e-05 The log loss is: 1.4379984484703494
For values of alpha =  0.0001 The log loss is: 1.2313183726512797
For values of alpha =  0.001 The log loss is: 1.2293958555824849
For values of alpha =  0.01 The log loss is: 1.331603233528802
For values of alpha =  0.1 The log loss is: 1.44296342332063
For values of alpha =  1 The log loss is: 1.4796564716509595
```

Cross Validation Error for each alpha



```
For values of best alpha =  0.001 The train log loss is: 1.097575223051
893
For values of best alpha =  0.001 The cross validation log loss is: 1.2
293958555824849
For values of best alpha =  0.001 The test log loss is: 1.2219718500779
153
```

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 [27]:
```
print("Q6. How many data points in Test and CV datasets are covered by
 the ", unique_genes.shape[0], " genes in train dataset?")

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  23
6  genes in train dataset?
Ans
1. In test data 653 out of 665 : 98.19548872180451
2. In cross validation data 508 out of  532 : 95.48872180451127
```

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?

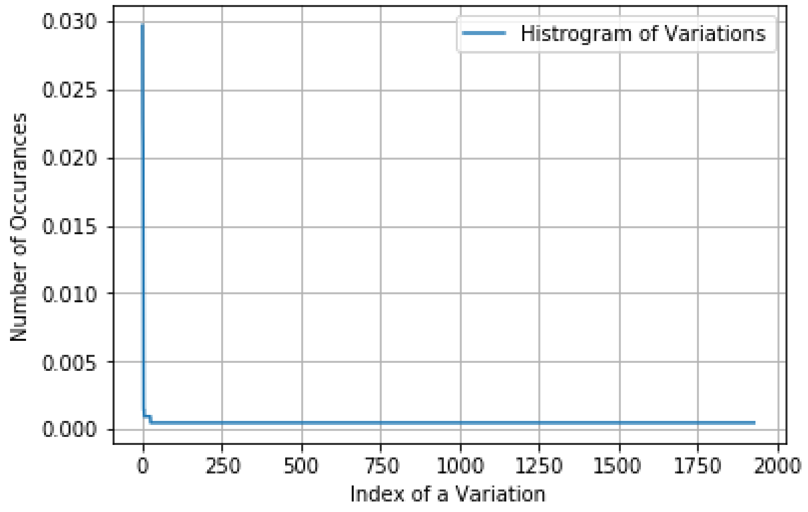
In [28]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1927
Truncating_Mutations    63
Amplification           48
Deletion                44
Fusions                 22
Overexpression           3
G12V                     3
T167A                    2
Y42C                     2
E17K                     2
R170W                    2
Name: Variation, dtype: int64
```

In [29]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categori
es of variations in the train data, and they are distibuted as follows"
,)
```

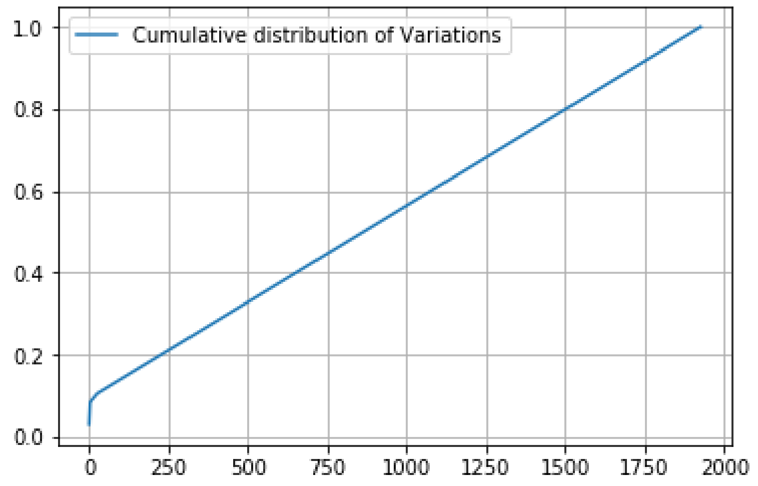
```
Ans: There are 1927 different categories of variations in the train dat
a, and they are distibuted as follows
```

In [30]:
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [31]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.02966102 0.05225989 0.07297552 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [32]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
 "Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
"Variation", test_df))
# cross validation gene feature
```

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

In [35]:
```
print("train_variation_feature_responseCoding is a converted feature us
ing the response coding method. The shape of Variation feature:", train
_variation_feature_responseCoding.shape)
```

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

In [36]:
```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer(ngram_range=(1,2))
train_variation_feature_onehotCoding = variation_vectorizer.fit_transfo
rm(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(te
st_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_d
f['Variation'])
```

In [37]:
```
print("train_variation_feature_onehotEncoded is converted feature using
 the onne-hot encoding method. The shape of Variation feature:", train_
variation_feature_onehotCoding.shape)
```

```
train_variation_feature_onehotEncoded is converted feature using the on
ne-hot encoding method. The shape of Variation feature: (2124, 2061)
```

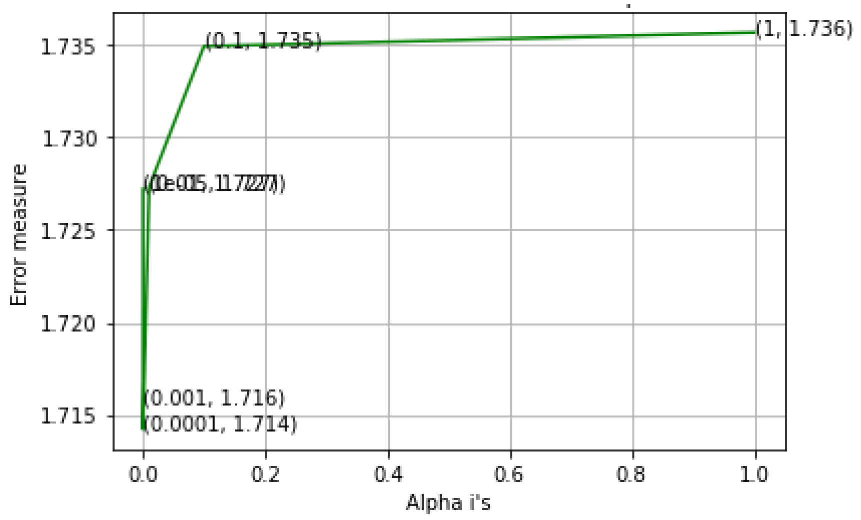
Q10. How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

In [38]:
```
alpha = [10 ** x for x in range(-5, 1)]

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

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

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



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

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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.7272085769039183
For values of alpha =  0.0001 The log loss is: 1.7142124463972064
For values of alpha =  0.001 The log loss is: 1.7155971034526885
For values of alpha =  0.01 The log loss is: 1.7272898519203104
For values of alpha =  0.1 The log loss is: 1.7349137710077112
For values of alpha =  1 The log loss is: 1.7356324515181336
```

Cross Validation Error for each alpha



```
For values of best alpha =  0.0001 The train log loss is: 0.72677226253
20597
For values of best alpha =  0.0001 The cross validation log loss is: 1.
7142124463972064
For values of best alpha =  0.0001 The test log loss is: 1.718488751630
7388
```

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

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

In [39]:
```
print("Q12. How many data points are covered by total ", unique_variati
ons.shape[0], " genes in test and cross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Vari
ation'])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'
])))].shape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0],
":",(test_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[
0],":" ,(cv_coverage/cv_df.shape[0])*100)
```

```
Q12. How many data points are covered by total  1927  genes in test and
cross validation data sets?
Ans
1. In test data 68 out of 665 : 10.225563909774436
2. In cross validation data 53 out of  532 : 9.962406015037594
```

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 [40]:
```
# cls_text is a data frame
# for every row in data fram consider the 'TEXT'
# split the words by space
# make a dict with those words
# increment its count whenever we see that word

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

In [41]:
```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(t
otal_dict.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_pr
ob/len(row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

In [42]:
```
# building a CountVectorizer with all the words that occured minimum 3
 times in train data
text_vectorizer = CountVectorizer(min_df=3,max_features=1000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_d
f['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

# zip(list(text_features),text_fea_counts) will zip a word with its num
ber of times it occured
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_count
s))


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

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

In [43]:
```
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = train_df[train_df['Class']==i]
    # build a word dict based on the words in that class
    dict_list.append(extract_dictionary_paddle(cls_text))
    # append it to dict_list

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


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

In [44]:
```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
test_text_feature_responseCoding  = get_text_responsecoding(test_df)
cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)
```

In [45]:
```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.
T/train_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/
test_text_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_t
ext_feature_responseCoding.sum(axis=1)).T
```

In [46]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCo
ding, axis=0)

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

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

In [47]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x:
 x[1] , reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

In [48]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({4142: 4, 3040: 3, 2614: 3, 7186: 2, 6461: 2, 6076: 2, 5815: 2,
5071: 2, 4987: 2, 4981: 2, 4923: 2, 4897: 2, 4829: 2, 4708: 2, 4369: 2,
4319: 2, 4311: 2, 4095: 2, 4080: 2, 3922: 2, 3803: 2, 3795: 2, 3759: 2,
3742: 2, 3719: 2, 3685: 2, 3632: 2, 3563: 2, 3540: 2, 3538: 2, 3516: 2,
3471: 2, 3450: 2, 3447: 2, 3360: 2, 3316: 2, 3280: 2, 3272: 2, 3216: 2,
3165: 2, 3158: 2, 3154: 2, 3124: 2, 3103: 2, 3086: 2, 3078: 2, 2984: 2,
2975: 2, 2967: 2, 2927: 2, 2919: 2, 2904: 2, 2837: 2, 2808: 2, 2699: 2,
2696: 2, 2686: 2, 2684: 2, 2662: 2, 151443: 1, 119487: 1, 80543: 1, 684
31: 1, 67808: 1, 65674: 1, 64633: 1, 64630: 1, 62117: 1, 56003: 1, 5389
5: 1, 50324: 1, 48901: 1, 48300: 1, 47408: 1, 44297: 1, 42382: 1, 4227
2: 1, 41917: 1, 41293: 1, 40975: 1, 40798: 1, 39850: 1, 38776: 1, 3818
8: 1, 37823: 1, 37321: 1, 37200: 1, 36444: 1, 36441: 1, 34617: 1, 3458
8: 1, 33495: 1, 33121: 1, 31672: 1, 31554: 1, 29592: 1, 28322: 1, 2816
7: 1, 26804: 1, 26375: 1, 26048: 1, 26033: 1, 24979: 1, 24852: 1, 2479
2: 1, 24653: 1, 24322: 1, 24133: 1, 24044: 1, 23851: 1, 23672: 1, 2352
2: 1, 22662: 1, 22455: 1, 22078: 1, 21720: 1, 21492: 1, 21423: 1, 2115
9: 1, 20737: 1, 20302: 1, 20147: 1, 19993: 1, 19525: 1, 19466: 1, 1945
6: 1, 19382: 1, 19282: 1, 19239: 1, 19077: 1, 19072: 1, 18817: 1, 1875
2: 1, 18655: 1, 18647: 1, 18263: 1, 18199: 1, 18194: 1, 18143: 1, 1800
6: 1, 17897: 1, 17885: 1, 17750: 1, 17743: 1, 17720: 1, 17556: 1, 1742
1: 1, 17303: 1, 17207: 1, 17105: 1, 17040: 1, 16976: 1, 16780: 1, 1668
5: 1, 16667: 1, 16490: 1, 16149: 1, 16137: 1, 16020: 1, 15841: 1, 1575
5: 1, 15745: 1, 15653: 1, 15606: 1, 15455: 1, 15398: 1, 15385: 1, 1537
0: 1, 15264: 1, 15224: 1, 15186: 1, 14962: 1, 14864: 1, 14592: 1, 1458
2: 1, 14552: 1, 14430: 1, 14400: 1, 14399: 1, 14309: 1, 13978: 1, 1392
```

```
9: 1, 13892: 1, 13774: 1, 13690: 1, 13654: 1, 13514: 1, 13495: 1, 1330
2: 1, 13287: 1, 13155: 1, 13132: 1, 13078: 1, 13069: 1, 13022: 1, 1288
6: 1, 12764: 1, 12676: 1, 12633: 1, 12592: 1, 12591: 1, 12584: 1, 1257
0: 1, 12528: 1, 12496: 1, 12417: 1, 12415: 1, 12346: 1, 12310: 1, 1229
5: 1, 12280: 1, 12257: 1, 12242: 1, 12194: 1, 12173: 1, 12116: 1, 1209
5: 1, 12065: 1, 12055: 1, 12039: 1, 11998: 1, 11993: 1, 11949: 1, 1193
0: 1, 11927: 1, 11804: 1, 11783: 1, 11754: 1, 11744: 1, 11718: 1, 1168
5: 1, 11651: 1, 11643: 1, 11629: 1, 11556: 1, 11485: 1, 11414: 1, 1127
2: 1, 11260: 1, 11167: 1, 11127: 1, 11098: 1, 11075: 1, 11067: 1, 1083
4: 1, 10784: 1, 10636: 1, 10575: 1, 10574: 1, 10523: 1, 10498: 1, 1048
8: 1, 10480: 1, 10468: 1, 10398: 1, 10396: 1, 10288: 1, 10204: 1, 1017
7: 1, 10162: 1, 10155: 1, 10143: 1, 10046: 1, 10036: 1, 9954: 1, 9945:
1, 9930: 1, 9882: 1, 9850: 1, 9808: 1, 9807: 1, 9792: 1, 9653: 1, 9652:
1, 9636: 1, 9617: 1, 9608: 1, 9582: 1, 9574: 1, 9571: 1, 9498: 1, 9412:
1, 9393: 1, 9361: 1, 9354: 1, 9322: 1, 9255: 1, 9219: 1, 9216: 1, 9199:
1, 9186: 1, 9171: 1, 9151: 1, 9072: 1, 9069: 1, 9059: 1, 9056: 1, 8998:
1, 8987: 1, 8973: 1, 8921: 1, 8824: 1, 8810: 1, 8799: 1, 8758: 1, 8730:
1, 8673: 1, 8646: 1, 8614: 1, 8575: 1, 8563: 1, 8548: 1, 8517: 1, 8509:
1, 8415: 1, 8377: 1, 8375: 1, 8358: 1, 8326: 1, 8305: 1, 8264: 1, 8202:
1, 8198: 1, 8193: 1, 8154: 1, 8151: 1, 8143: 1, 8133: 1, 8126: 1, 8117:
1, 8056: 1, 8052: 1, 8023: 1, 7988: 1, 7987: 1, 7981: 1, 7941: 1, 7924:
1, 7890: 1, 7856: 1, 7840: 1, 7832: 1, 7818: 1, 7817: 1, 7803: 1, 7794:
1, 7771: 1, 7727: 1, 7720: 1, 7698: 1, 7675: 1, 7631: 1, 7591: 1, 7550:
1, 7544: 1, 7524: 1, 7498: 1, 7488: 1, 7471: 1, 7464: 1, 7419: 1, 7403:
1, 7384: 1, 7296: 1, 7275: 1, 7265: 1, 7263: 1, 7246: 1, 7238: 1, 7215:
1, 7191: 1, 7180: 1, 7121: 1, 7101: 1, 7099: 1, 7064: 1, 7059: 1, 7042:
1, 7037: 1, 7033: 1, 7032: 1, 7030: 1, 7025: 1, 7020: 1, 7018: 1, 7000:
1, 6999: 1, 6988: 1, 6984: 1, 6975: 1, 6967: 1, 6949: 1, 6942: 1, 6936:
1, 6927: 1, 6912: 1, 6888: 1, 6881: 1, 6847: 1, 6792: 1, 6728: 1, 6697:
1, 6673: 1, 6650: 1, 6633: 1, 6622: 1, 6620: 1, 6614: 1, 6611: 1, 6603:
1, 6547: 1, 6544: 1, 6539: 1, 6520: 1, 6505: 1, 6497: 1, 6479: 1, 6469:
1, 6455: 1, 6449: 1, 6439: 1, 6423: 1, 6413: 1, 6404: 1, 6374: 1, 6370:
1, 6351: 1, 6319: 1, 6318: 1, 6306: 1, 6277: 1, 6271: 1, 6255: 1, 6252:
1, 6249: 1, 6229: 1, 6212: 1, 6208: 1, 6207: 1, 6199: 1, 6172: 1, 6165:
1, 6155: 1, 6137: 1, 6134: 1, 6094: 1, 6088: 1, 6082: 1, 6075: 1, 6072:
1, 6067: 1, 6044: 1, 6027: 1, 6004: 1, 5930: 1, 5920: 1, 5910: 1, 5901:
1, 5884: 1, 5860: 1, 5855: 1, 5853: 1, 5829: 1, 5808: 1, 5803: 1, 5794:
1, 5790: 1, 5789: 1, 5788: 1, 5786: 1, 5776: 1, 5775: 1, 5762: 1, 5747:
1, 5743: 1, 5698: 1, 5661: 1, 5656: 1, 5650: 1, 5647: 1, 5614: 1, 5610:
```

```
1, 5608: 1, 5604: 1, 5587: 1, 5562: 1, 5557: 1, 5547: 1, 5541: 1, 5525:
1, 5508: 1, 5497: 1, 5488: 1, 5466: 1, 5448: 1, 5447: 1, 5445: 1, 5444:
1, 5413: 1, 5411: 1, 5406: 1, 5400: 1, 5399: 1, 5394: 1, 5388: 1, 5360:
1, 5335: 1, 5310: 1, 5305: 1, 5300: 1, 5298: 1, 5290: 1, 5275: 1, 5260:
1, 5243: 1, 5242: 1, 5226: 1, 5211: 1, 5203: 1, 5181: 1, 5173: 1, 5163:
1, 5153: 1, 5134: 1, 5125: 1, 5122: 1, 5121: 1, 5114: 1, 5112: 1, 5101:
1, 5091: 1, 5082: 1, 5063: 1, 5056: 1, 5054: 1, 5030: 1, 5023: 1, 5020:
1, 5014: 1, 4978: 1, 4964: 1, 4957: 1, 4939: 1, 4934: 1, 4896: 1, 4889:
1, 4880: 1, 4872: 1, 4870: 1, 4868: 1, 4865: 1, 4845: 1, 4835: 1, 4832:
1, 4830: 1, 4816: 1, 4807: 1, 4804: 1, 4791: 1, 4765: 1, 4758: 1, 4747:
1, 4745: 1, 4739: 1, 4738: 1, 4736: 1, 4735: 1, 4733: 1, 4716: 1, 4712:
1, 4701: 1, 4660: 1, 4654: 1, 4631: 1, 4621: 1, 4605: 1, 4604: 1, 4600:
1, 4576: 1, 4566: 1, 4555: 1, 4550: 1, 4533: 1, 4520: 1, 4501: 1, 4480:
1, 4476: 1, 4475: 1, 4459: 1, 4457: 1, 4454: 1, 4449: 1, 4446: 1, 4440:
1, 4438: 1, 4433: 1, 4415: 1, 4408: 1, 4394: 1, 4375: 1, 4372: 1, 4365:
1, 4356: 1, 4346: 1, 4344: 1, 4341: 1, 4337: 1, 4312: 1, 4307: 1, 4297:
1, 4294: 1, 4290: 1, 4286: 1, 4273: 1, 4268: 1, 4255: 1, 4238: 1, 4237:
1, 4230: 1, 4226: 1, 4222: 1, 4216: 1, 4213: 1, 4208: 1, 4201: 1, 4197:
1, 4175: 1, 4167: 1, 4165: 1, 4164: 1, 4160: 1, 4159: 1, 4141: 1, 4140:
1, 4136: 1, 4134: 1, 4124: 1, 4121: 1, 4120: 1, 4114: 1, 4111: 1, 4110:
1, 4107: 1, 4101: 1, 4090: 1, 4089: 1, 4075: 1, 4071: 1, 4068: 1, 4063:
1, 4061: 1, 4050: 1, 4047: 1, 4046: 1, 4029: 1, 4027: 1, 4009: 1, 3996:
1, 3990: 1, 3978: 1, 3965: 1, 3956: 1, 3952: 1, 3946: 1, 3944: 1, 3928:
1, 3924: 1, 3921: 1, 3916: 1, 3908: 1, 3903: 1, 3899: 1, 3896: 1, 3895:
1, 3892: 1, 3882: 1, 3866: 1, 3857: 1, 3856: 1, 3854: 1, 3851: 1, 3836:
1, 3833: 1, 3823: 1, 3820: 1, 3819: 1, 3798: 1, 3785: 1, 3781: 1, 3769:
1, 3763: 1, 3754: 1, 3753: 1, 3750: 1, 3749: 1, 3740: 1, 3739: 1, 3735:
1, 3724: 1, 3718: 1, 3717: 1, 3708: 1, 3695: 1, 3694: 1, 3688: 1, 3668:
1, 3667: 1, 3666: 1, 3664: 1, 3658: 1, 3657: 1, 3649: 1, 3648: 1, 3647:
1, 3633: 1, 3629: 1, 3628: 1, 3627: 1, 3620: 1, 3613: 1, 3605: 1, 3604:
1, 3591: 1, 3583: 1, 3579: 1, 3578: 1, 3570: 1, 3569: 1, 3567: 1, 3566:
1, 3562: 1, 3559: 1, 3551: 1, 3549: 1, 3548: 1, 3529: 1, 3527: 1, 3520:
1, 3518: 1, 3511: 1, 3502: 1, 3500: 1, 3497: 1, 3495: 1, 3494: 1, 3491:
1, 3490: 1, 3489: 1, 3477: 1, 3476: 1, 3468: 1, 3456: 1, 3446: 1, 3438:
1, 3431: 1, 3429: 1, 3423: 1, 3419: 1, 3414: 1, 3411: 1, 3405: 1, 3401:
1, 3399: 1, 3398: 1, 3397: 1, 3395: 1, 3388: 1, 3382: 1, 3374: 1, 3372:
1, 3364: 1, 3362: 1, 3350: 1, 3347: 1, 3343: 1, 3342: 1, 3334: 1, 3333:
1, 3329: 1, 3324: 1, 3322: 1, 3317: 1, 3313: 1, 3311: 1, 3307: 1, 3301:
1, 3300: 1, 3298: 1, 3289: 1, 3277: 1, 3275: 1, 3266: 1, 3265: 1, 3262:
```

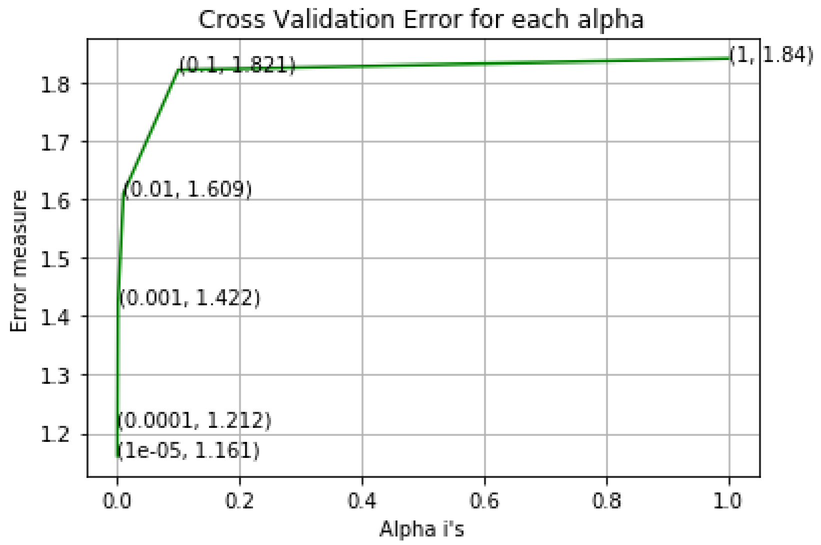
```
1, 3259: 1, 3248: 1, 3240: 1, 3239: 1, 3238: 1, 3236: 1, 3232: 1, 3230:
1, 3226: 1, 3225: 1, 3223: 1, 3206: 1, 3200: 1, 3199: 1, 3194: 1, 3182:
1, 3181: 1, 3178: 1, 3175: 1, 3163: 1, 3149: 1, 3135: 1, 3132: 1, 3127:
1, 3109: 1, 3105: 1, 3104: 1, 3092: 1, 3090: 1, 3088: 1, 3087: 1, 3079:
1, 3076: 1, 3074: 1, 3073: 1, 3071: 1, 3064: 1, 3061: 1, 3060: 1, 3059:
1, 3055: 1, 3054: 1, 3049: 1, 3048: 1, 3042: 1, 3036: 1, 3034: 1, 3030:
1, 3028: 1, 3024: 1, 3022: 1, 3020: 1, 3016: 1, 3015: 1, 3011: 1, 3009:
1, 3004: 1, 3003: 1, 2986: 1, 2985: 1, 2982: 1, 2978: 1, 2977: 1, 2970:
1, 2968: 1, 2961: 1, 2948: 1, 2946: 1, 2940: 1, 2935: 1, 2933: 1, 2930:
1, 2929: 1, 2920: 1, 2917: 1, 2915: 1, 2912: 1, 2901: 1, 2897: 1, 2888:
1, 2887: 1, 2881: 1, 2877: 1, 2872: 1, 2871: 1, 2858: 1, 2856: 1, 2855:
1, 2848: 1, 2844: 1, 2842: 1, 2836: 1, 2831: 1, 2827: 1, 2826: 1, 2823:
1, 2822: 1, 2813: 1, 2800: 1, 2799: 1, 2780: 1, 2765: 1, 2759: 1, 2751:
1, 2747: 1, 2736: 1, 2731: 1, 2730: 1, 2724: 1, 2723: 1, 2721: 1, 2717:
1, 2716: 1, 2715: 1, 2707: 1, 2705: 1, 2675: 1, 2669: 1, 2668: 1, 2663:
1, 2659: 1, 2658: 1, 2652: 1, 2651: 1, 2648: 1, 2647: 1, 2641: 1, 2634:
1, 2633: 1, 2632: 1, 2630: 1, 2628: 1, 2625: 1, 2621: 1, 2620: 1, 2615:
1, 2613: 1, 2612: 1, 2608: 1, 2606: 1, 2605: 1, 2604: 1, 2603: 1})
```

In [49]:

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

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

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

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

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


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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.1609811576088083
For values of alpha =  0.0001 The log loss is: 1.212451943689073
For values of alpha =  0.001 The log loss is: 1.4216373926725325
For values of alpha =  0.01 The log loss is: 1.6092031649384446
For values of alpha =  0.1 The log loss is: 1.8210514666597388
For values of alpha =  1 The log loss is: 1.839505424450917
```



```
For values of best alpha =  1e-05 The train log loss is: 0.886330259449
93
For values of best alpha =  1e-05 The cross validation log loss is: 1.1
609811576088083
For values of best alpha =  1e-05 The test log loss is: 1.1593656891610
78
```

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

Ans. Yes, it seems like!

In [50]:
```
def get_intersec_text(df):
    df_text_vec = CountVectorizer(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_coun
ts))
    len1 = len(set(df_text_features))
    len2 = len(set(train_text_features) & set(df_text_features))
    return len1,len2
```

In [51]:
```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in
 train data")
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appe
ared in train data")
```

```
95.1 % of word of test data appeared in train data
94.2 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [52]:
```
#Data preparation for ML models.

#Misc. functionns for ML models


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

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

In [53]:
```
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

In [55]:
```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text
 or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer(ngram_range=(1,2))
    var_count_vec = CountVectorizer(ngram_range=(1,2))
    text_count_vec = CountVectorizer(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 p
oint [{}]".format(word,yes_no))
        else:
            word = text_vec.get_feature_names()[v-(fea1_len+fea2_len)]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
                print(i, "Text feature [{}] present in test data point
 [{}]".format(word,yes_no))

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

Stacking the three types of features

In [56]:
```
# merging gene, variance and text features

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

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,t
rain_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,tes
t_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_vari
ation_feature_onehotCoding))

train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_
feature_onehotCoding)).tocsr()
train_y = np.array(list(train_df['Class']))

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

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


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

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

In [57]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ",
train_x_onehotCoding.shape)
```

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

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 32
96)
(number of data points * number of features) in test data =  (665, 329
6)
(number of data points * number of features) in cross validation data =
(532, 3296)
```

In [58]:

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

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 2
7)
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data =
(532, 27)
```

4.1. Base Line Model

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

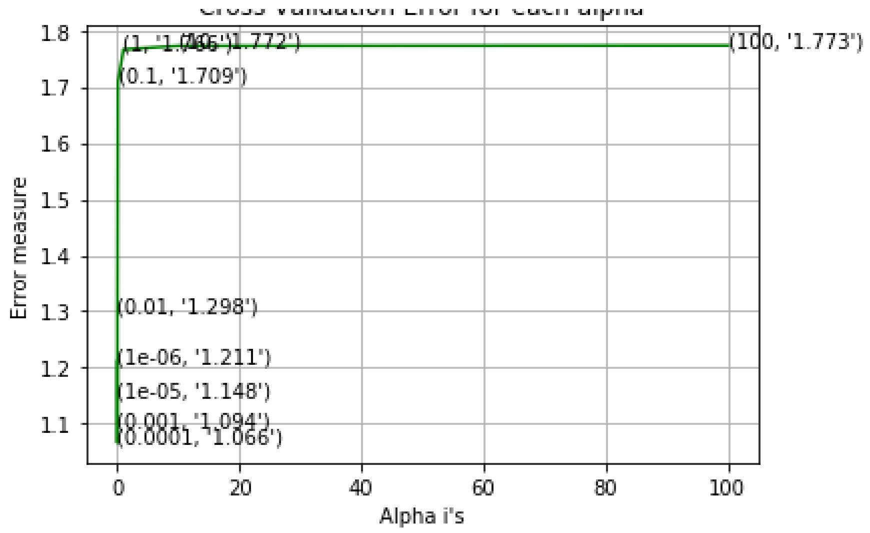
In [59]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



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

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

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


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

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

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

```
for alpha = 1e-06
Log Loss : 1.211310400566089
for alpha = 1e-05
Log Loss : 1.1477911961389067
for alpha = 0.0001
Log Loss : 1.0659397153244263
for alpha = 0.001
Log Loss : 1.0939899046956856
for alpha = 0.01
Log Loss : 1.2983319997281053
for alpha = 0.1
Log Loss : 1.709250548590752
for alpha = 1
Log Loss : 1.7662130131793063
for alpha = 10
Log Loss : 1.7722434722354805
for alpha = 100
Log Loss : 1.7729281218791404
```

Cross Validation Error for each alpha



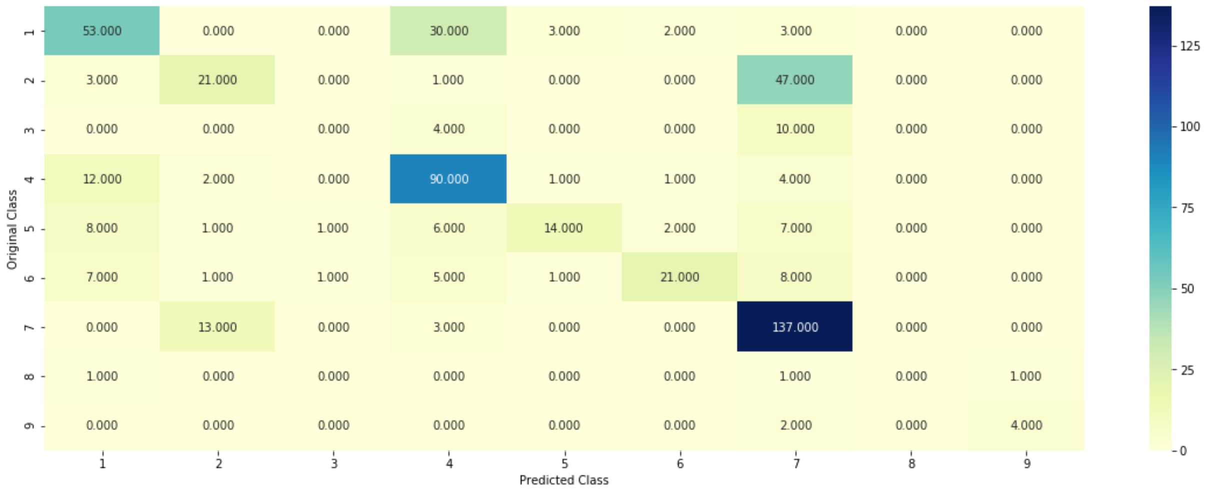
```
For values of best alpha =  0.0001 The train log loss is: 0.42213506881
127827
For values of best alpha =  0.0001 The cross validation log loss is: 1.
0659397153244263
For values of best alpha =  0.0001 The test log loss is: 0.995273354535
8166
```

4.3.1.2. Testing the model with best hyper paramters

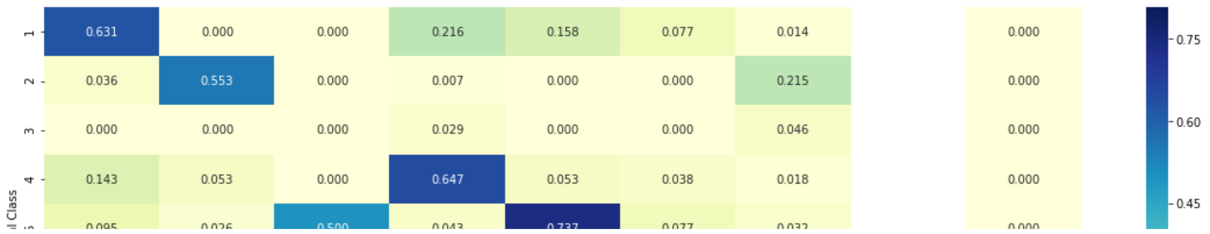
In [60]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

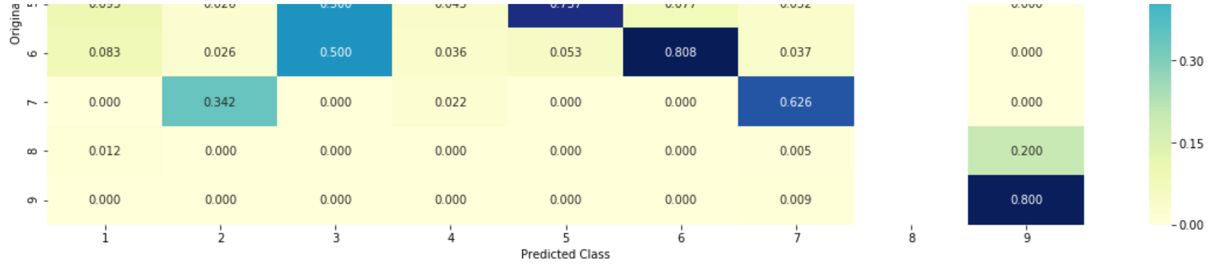
```
# predict(X)    Predict class labels for samples in X.

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

```
Log loss : 1.0659397153244263
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```

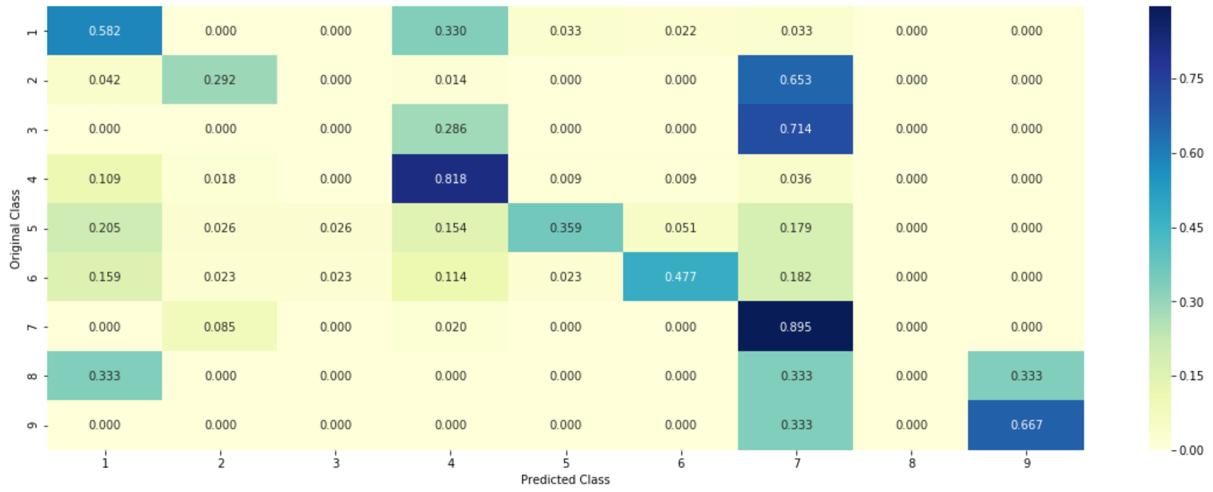


```
-------------------- Precision matrix (Columm Sum=1) ------------------
--
```





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



4.3.1.3. Feature Importance

```
In [61]: def get_imp_feature_names(text, indices, removed_ind = []):
             word_present = 0
             tabulte_list = []
             incresingorder_ind = 0
             for i in indices:
                 if i < train_gene_feature_onehotCoding.shape[1]:
                     tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
                 elif i< 18:
                     tabulte_list.append([incresingorder_ind,"Variation", "Yes"
         ])
```

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

4.3.1.3.1. Correctly Classified point

In [62]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.000e+00 3.000e-04 0.000e+00 1.800e-0
3 0.000e+00 0.000e+00 9.978e-01
```

```
  0.000e+00 0.000e+00]]
Actual Class : 7
--------------------------------------------------
12 Text feature [downstream] present in test data point [True]
23 Text feature [extracellular] present in test data point [True]
33 Text feature [akt] present in test data point [True]
43 Text feature [s3] present in test data point [True]
46 Text feature [transformed] present in test data point [True]
48 Text feature [constitutive] present in test data point [True]
53 Text feature [somatic] present in test data point [True]
56 Text feature [insertion] present in test data point [True]
63 Text feature [isoforms] present in test data point [True]
64 Text feature [inhibited] present in test data point [True]
67 Text feature [concentrations] present in test data point [True]
71 Text feature [nucleotide] present in test data point [True]
75 Text feature [transforming] present in test data point [True]
81 Text feature [elevated] present in test data point [True]
87 Text feature [activated] present in test data point [True]
88 Text feature [carcinomas] present in test data point [True]
108 Text feature [oncogene] present in test data point [True]
119 Text feature [enhanced] present in test data point [True]
132 Text feature [high] present in test data point [True]
134 Text feature [ligand] present in test data point [True]
144 Text feature [colony] present in test data point [True]
146 Text feature [positive] present in test data point [True]
147 Text feature [lung] present in test data point [True]
152 Text feature [3b] present in test data point [True]
169 Text feature [expressing] present in test data point [True]
170 Text feature [phospho] present in test data point [True]
174 Text feature [cancers] present in test data point [True]
176 Text feature [approximately] present in test data point [True]
177 Text feature [codon] present in test data point [True]
183 Text feature [derived] present in test data point [True]
191 Text feature [receptors] present in test data point [True]
230 Text feature [malignant] present in test data point [True]
235 Text feature [constitutively] present in test data point [True]
246 Text feature [epithelial] present in test data point [True]
251 Text feature [egf] present in test data point [True]
256 Text feature [day] present in test data point [True]
```

```
277 Text feature [free] present in test data point [True]
283 Text feature [2a] present in test data point [True]
293 Text feature [specimens] present in test data point [True]
303 Text feature [initial] present in test data point [True]
305 Text feature [bone] present in test data point [True]
316 Text feature [3a] present in test data point [True]
317 Text feature [carcinoma] present in test data point [True]
327 Text feature [express] present in test data point [True]
328 Text feature [adenocarcinoma] present in test data point [True]
330 Text feature [locus] present in test data point [True]
332 Text feature [common] present in test data point [True]
334 Text feature [survival] present in test data point [True]
338 Text feature [position] present in test data point [True]
351 Text feature [activation] present in test data point [True]
355 Text feature [fig] present in test data point [True]
369 Text feature [provided] present in test data point [True]
370 Text feature [nm] present in test data point [True]
374 Text feature [factor] present in test data point [True]
379 Text feature [sensitive] present in test data point [True]
383 Text feature [her2] present in test data point [True]
384 Text feature [wt] present in test data point [True]
393 Text feature [long] present in test data point [True]
395 Text feature [2b] present in test data point [True]
399 Text feature [measured] present in test data point [True]
402 Text feature [induction] present in test data point [True]
408 Text feature [pathways] present in test data point [True]
432 Text feature [regulated] present in test data point [True]
436 Text feature [mutants] present in test data point [True]
445 Text feature [distinct] present in test data point [True]
446 Text feature [overexpression] present in test data point [True]
450 Text feature [basal] present in test data point [True]
452 Text feature [effective] present in test data point [True]
455 Text feature [regions] present in test data point [True]
460 Text feature [51] present in test data point [True]
462 Text feature [phosphorylated] present in test data point [True]
472 Text feature [melanomas] present in test data point [True]
473 Text feature [versus] present in test data point [True]
474 Text feature [jak2] present in test data point [True]
479 Text feature [leading] present in test data point [True]
```

```
480 Text feature [tyrosine] present in test data point [True]
482 Text feature [rate] present in test data point [True]
491 Text feature [bp] present in test data point [True]
492 Text feature [gain] present in test data point [True]
Out of the top  500  features  79 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [63]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[1.170e-01 2.622e-01 1.000e-04 5.700e-0
3 1.100e-03 1.300e-03 6.109e-01
  1.500e-03 1.000e-04]]
Actual Class : 2
--------------------------------------------------
12 Text feature [downstream] present in test data point [True]
23 Text feature [extracellular] present in test data point [True]
33 Text feature [akt] present in test data point [True]
46 Text feature [transformed] present in test data point [True]
48 Text feature [constitutive] present in test data point [True]
53 Text feature [somatic] present in test data point [True]
67 Text feature [concentrations] present in test data point [True]
71 Text feature [nucleotide] present in test data point [True]
75 Text feature [transforming] present in test data point [True]
81 Text feature [elevated] present in test data point [True]
87 Text feature [activated] present in test data point [True]
```

```
88 Text feature [carcinomas] present in test data point [True]
108 Text feature [oncogene] present in test data point [True]
119 Text feature [enhanced] present in test data point [True]
132 Text feature [high] present in test data point [True]
146 Text feature [positive] present in test data point [True]
147 Text feature [lung] present in test data point [True]
152 Text feature [3b] present in test data point [True]
169 Text feature [expressing] present in test data point [True]
170 Text feature [phospho] present in test data point [True]
174 Text feature [cancers] present in test data point [True]
176 Text feature [approximately] present in test data point [True]
177 Text feature [codon] present in test data point [True]
183 Text feature [derived] present in test data point [True]
235 Text feature [constitutively] present in test data point [True]
256 Text feature [day] present in test data point [True]
277 Text feature [free] present in test data point [True]
283 Text feature [2a] present in test data point [True]
289 Text feature [prostate] present in test data point [True]
293 Text feature [specimens] present in test data point [True]
303 Text feature [initial] present in test data point [True]
316 Text feature [3a] present in test data point [True]
317 Text feature [carcinoma] present in test data point [True]
327 Text feature [express] present in test data point [True]
328 Text feature [adenocarcinoma] present in test data point [True]
330 Text feature [locus] present in test data point [True]
332 Text feature [common] present in test data point [True]
334 Text feature [survival] present in test data point [True]
338 Text feature [position] present in test data point [True]
351 Text feature [activation] present in test data point [True]
355 Text feature [fig] present in test data point [True]
369 Text feature [provided] present in test data point [True]
374 Text feature [factor] present in test data point [True]
379 Text feature [sensitive] present in test data point [True]
384 Text feature [wt] present in test data point [True]
393 Text feature [long] present in test data point [True]
395 Text feature [2b] present in test data point [True]
399 Text feature [measured] present in test data point [True]
402 Text feature [induction] present in test data point [True]
408 Text feature [pathways] present in test data point [True]
```

```
432 Text feature [regulated] present in test data point [True]
436 Text feature [mutants] present in test data point [True]
445 Text feature [distinct] present in test data point [True]
446 Text feature [overexpression] present in test data point [True]
450 Text feature [basal] present in test data point [True]
452 Text feature [effective] present in test data point [True]
455 Text feature [regions] present in test data point [True]
460 Text feature [51] present in test data point [True]
462 Text feature [phosphorylated] present in test data point [True]
472 Text feature [melanomas] present in test data point [True]
473 Text feature [versus] present in test data point [True]
482 Text feature [rate] present in test data point [True]
Out of the top  500  features  62 are present in query point
```

4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

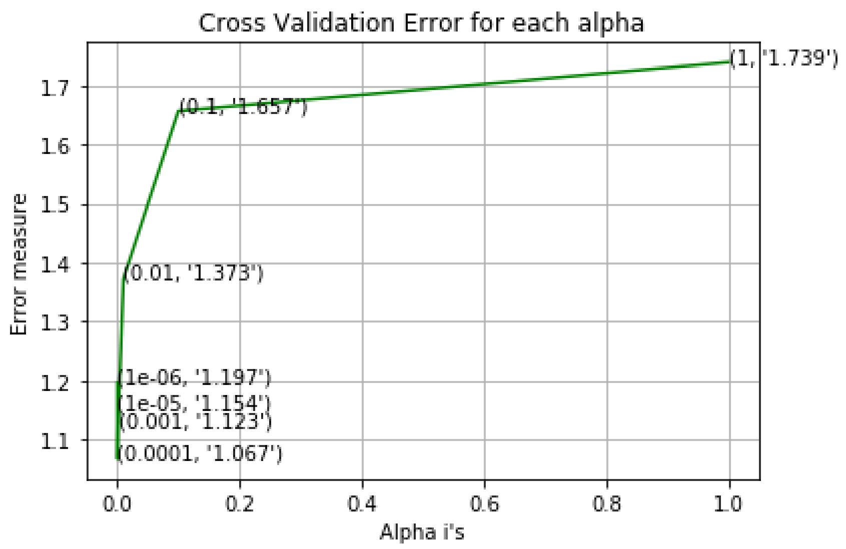
In [64]:

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

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

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

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



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

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

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.1965592340943971
for alpha = 1e-05
Log Loss : 1.1539293301588411
for alpha = 0.0001
Log Loss : 1.0672609467055778
for alpha = 0.001
Log Loss : 1.1226791058638195
for alpha = 0.01
Log Loss : 1.372531512169629
for alpha = 0.1
Log Loss : 1.6565007985068247
for alpha = 1
Log Loss : 1.7389934711034953
```

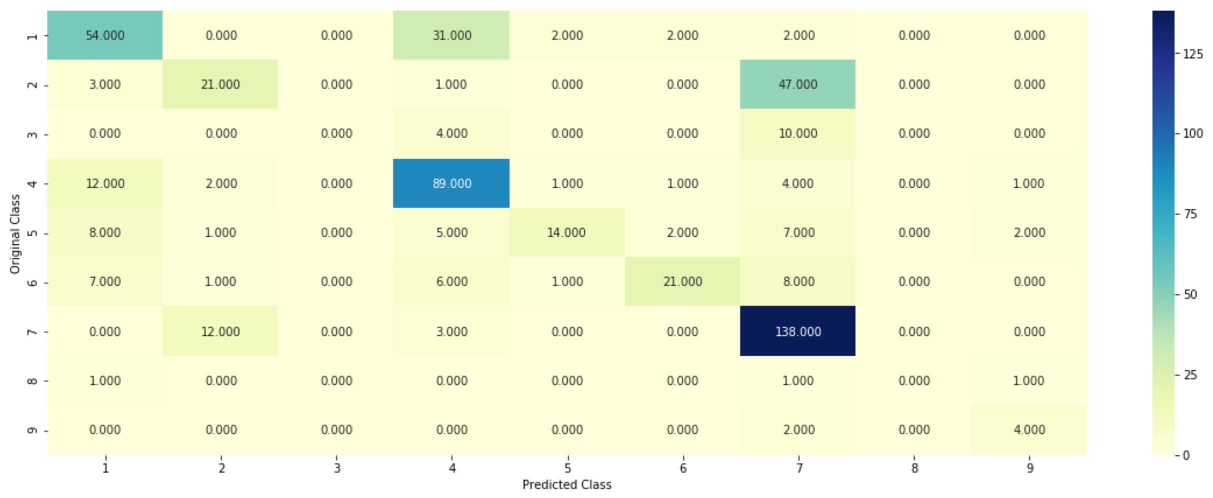


```
For values of best alpha =  0.0001 The train log loss is: 0.41618714740
493373
For values of best alpha =  0.0001 The cross validation log loss is: 1.
0672609467055778
For values of best alpha =  0.0001 The test log loss is: 1.007674785556
6217
```

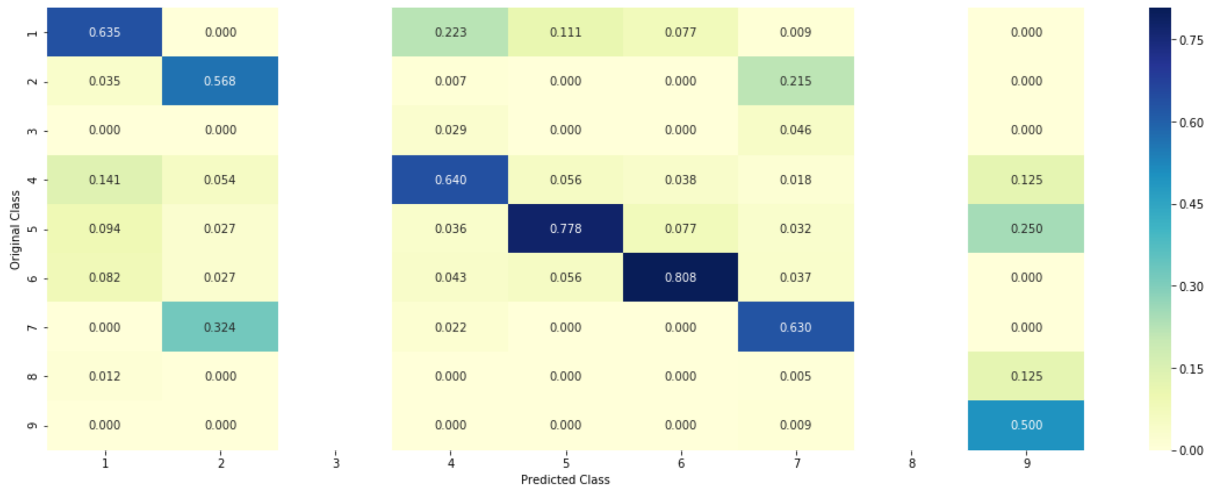
4.3.2.2. Testing model with best hyper parameters

In [65]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

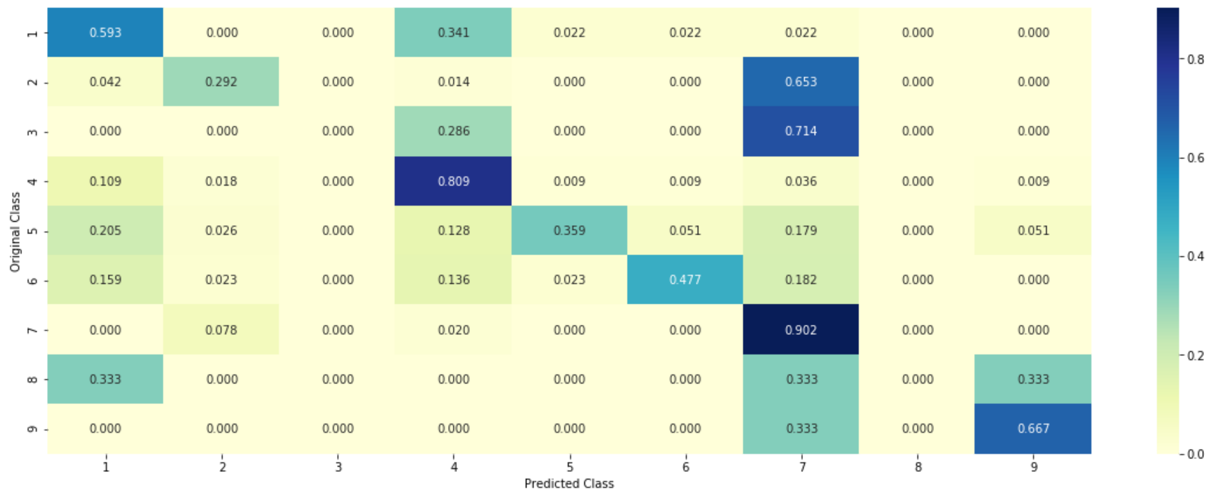
```
Log loss : 1.0672609467055778
Number of mis-classified points : 0.35902255639097747
-------------------- Confusion matrix --------------------
```



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



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



4.3.2.3. Feature Importance, Correctly Classified point

```
In [66]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
         random_state=42)
         clf.fit(train_x_onehotCoding,train_y)
```

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.000e+00 3.000e-04 0.000e+00 1.600e-0
3 0.000e+00 0.000e+00 9.981e-01
  0.000e+00 0.000e+00]]
Actual Class : 7
--------------------------------------------------
21 Text feature [downstream] present in test data point [True]
40 Text feature [extracellular] present in test data point [True]
59 Text feature [s3] present in test data point [True]
73 Text feature [akt] present in test data point [True]
81 Text feature [somatic] present in test data point [True]
86 Text feature [transformed] present in test data point [True]
104 Text feature [isoforms] present in test data point [True]
107 Text feature [nucleotide] present in test data point [True]
115 Text feature [insertion] present in test data point [True]
128 Text feature [carcinomas] present in test data point [True]
131 Text feature [concentrations] present in test data point [True]
144 Text feature [elevated] present in test data point [True]
149 Text feature [inhibited] present in test data point [True]
182 Text feature [activated] present in test data point [True]
186 Text feature [transforming] present in test data point [True]
189 Text feature [high] present in test data point [True]
193 Text feature [codon] present in test data point [True]
194 Text feature [constitutive] present in test data point [True]
196 Text feature [colony] present in test data point [True]
223 Text feature [enhanced] present in test data point [True]
226 Text feature [positive] present in test data point [True]
```

```
230 Text feature [cancers] present in test data point [True]
233 Text feature [ligand] present in test data point [True]
235 Text feature [3b] present in test data point [True]
244 Text feature [2a] present in test data point [True]
251 Text feature [lung] present in test data point [True]
261 Text feature [locus] present in test data point [True]
265 Text feature [derived] present in test data point [True]
284 Text feature [egf] present in test data point [True]
287 Text feature [expressing] present in test data point [True]
294 Text feature [epithelial] present in test data point [True]
300 Text feature [oncogene] present in test data point [True]
301 Text feature [phospho] present in test data point [True]
318 Text feature [receptors] present in test data point [True]
331 Text feature [malignant] present in test data point [True]
335 Text feature [position] present in test data point [True]
339 Text feature [3a] present in test data point [True]
340 Text feature [approximately] present in test data point [True]
343 Text feature [initial] present in test data point [True]
347 Text feature [common] present in test data point [True]
348 Text feature [day] present in test data point [True]
355 Text feature [fig] present in test data point [True]
359 Text feature [bone] present in test data point [True]
362 Text feature [specimens] present in test data point [True]
363 Text feature [regions] present in test data point [True]
369 Text feature [wt] present in test data point [True]
375 Text feature [distinct] present in test data point [True]
380 Text feature [long] present in test data point [True]
382 Text feature [overexpression] present in test data point [True]
398 Text feature [carcinoma] present in test data point [True]
399 Text feature [provided] present in test data point [True]
403 Text feature [phosphorylated] present in test data point [True]
415 Text feature [adenocarcinoma] present in test data point [True]
416 Text feature [2b] present in test data point [True]
419 Text feature [regulated] present in test data point [True]
421 Text feature [factor] present in test data point [True]
432 Text feature [pathways] present in test data point [True]
433 Text feature [free] present in test data point [True]
435 Text feature [nm] present in test data point [True]
453 Text feature [1998] present in test data point [True]
```

```
454 Text feature [constitutively] present in test data point [True]
459 Text feature [her2] present in test data point [True]
460 Text feature [survival] present in test data point [True]
462 Text feature [measured] present in test data point [True]
468 Text feature [mutants] present in test data point [True]
472 Text feature [express] present in test data point [True]
473 Text feature [membrane] present in test data point [True]
475 Text feature [basal] present in test data point [True]
478 Text feature [rate] present in test data point [True]
480 Text feature [induction] present in test data point [True]
481 Text feature [regulatory] present in test data point [True]
484 Text feature [examined] present in test data point [True]
485 Text feature [activation] present in test data point [True]
486 Text feature [2004] present in test data point [True]
497 Text feature [sensitive] present in test data point [True]
498 Text feature [gain] present in test data point [True]
499 Text feature [coding] present in test data point [True]
Out of the top  500  features  77 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [67]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[1.202e-01 2.555e-01 2.000e-04 5.300e-0
3 1.000e-03 1.800e-03 6.145e-01
  1.500e-03 0.000e+00]]
```

```
Actual Class : 2
--------------------------------------------------
21 Text feature [downstream] present in test data point [True]
40 Text feature [extracellular] present in test data point [True]
73 Text feature [akt] present in test data point [True]
81 Text feature [somatic] present in test data point [True]
86 Text feature [transformed] present in test data point [True]
107 Text feature [nucleotide] present in test data point [True]
128 Text feature [carcinomas] present in test data point [True]
131 Text feature [concentrations] present in test data point [True]
144 Text feature [elevated] present in test data point [True]
182 Text feature [activated] present in test data point [True]
186 Text feature [transforming] present in test data point [True]
189 Text feature [high] present in test data point [True]
193 Text feature [codon] present in test data point [True]
194 Text feature [constitutive] present in test data point [True]
223 Text feature [enhanced] present in test data point [True]
226 Text feature [positive] present in test data point [True]
230 Text feature [cancers] present in test data point [True]
235 Text feature [3b] present in test data point [True]
244 Text feature [2a] present in test data point [True]
245 Text feature [prostate] present in test data point [True]
251 Text feature [lung] present in test data point [True]
261 Text feature [locus] present in test data point [True]
265 Text feature [derived] present in test data point [True]
287 Text feature [expressing] present in test data point [True]
300 Text feature [oncogene] present in test data point [True]
301 Text feature [phospho] present in test data point [True]
335 Text feature [position] present in test data point [True]
339 Text feature [3a] present in test data point [True]
340 Text feature [approximately] present in test data point [True]
343 Text feature [initial] present in test data point [True]
347 Text feature [common] present in test data point [True]
348 Text feature [day] present in test data point [True]
355 Text feature [fig] present in test data point [True]
362 Text feature [specimens] present in test data point [True]
363 Text feature [regions] present in test data point [True]
369 Text feature [wt] present in test data point [True]
375 Text feature [distinct] present in test data point [True]
```

```
380 Text feature [long] present in test data point [True]
382 Text feature [overexpression] present in test data point [True]
398 Text feature [carcinoma] present in test data point [True]
399 Text feature [provided] present in test data point [True]
403 Text feature [phosphorylated] present in test data point [True]
415 Text feature [adenocarcinoma] present in test data point [True]
416 Text feature [2b] present in test data point [True]
419 Text feature [regulated] present in test data point [True]
421 Text feature [factor] present in test data point [True]
432 Text feature [pathways] present in test data point [True]
433 Text feature [free] present in test data point [True]
454 Text feature [constitutively] present in test data point [True]
460 Text feature [survival] present in test data point [True]
462 Text feature [measured] present in test data point [True]
468 Text feature [mutants] present in test data point [True]
472 Text feature [express] present in test data point [True]
473 Text feature [membrane] present in test data point [True]
475 Text feature [basal] present in test data point [True]
478 Text feature [rate] present in test data point [True]
480 Text feature [induction] present in test data point [True]
484 Text feature [examined] present in test data point [True]
485 Text feature [activation] present in test data point [True]
497 Text feature [sensitive] present in test data point [True]
499 Text feature [coding] present in test data point [True]
Out of the top  500  features  61 are present in query point
```

Conclusion

In [1]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["S.No","Model","Train logloss","Cv logloss","Test logl
oss", "Misclassified error"]

x.add_row(["1","Logistic Regression(with balancing)","0.422","1.065",
"0.995","0.36"])
x.add_row(["2","Logistic regression(without balancing)","0.416","1.067"
,"1.007","0.35"])


print(x)
```

```
+------+----------------------------------------+--------------+------
------+-------------+--------------------+
| S.No |                 Model                  | Train logloss | Cv lo
gloss | Test logloss | Misclassified error |
+------+----------------------------------------+--------------+------
------+-------------+--------------------+
|  1   |  Logistic Regression(with balancing)   |     0.422     |   1.0
65    |    0.995     |         0.36        |
|  2   | Logistic regression(without balancing) |     0.416     |   1.0
67    |    1.007     |         0.35        |
+------+----------------------------------------+--------------+------
------+-------------+--------------------+
```