

Personalized Cancer Diagnosis BIGRAM Using Logistic Regression

November 3, 2018

Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

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 contains 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:

```
<li>
training_variants (ID , Gene, Variations, Class)
</li>
<li>
training_text (ID, Text)
</li>
```

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

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 Learning 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 probabilities => 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
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.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\Saurabh\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: "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 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"],
        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 the data
            if not word in stop_words:
                string += word + " "

        data_text[column][index] = string

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

there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
```

Time took for preprocessing the text : 141.0041900200722 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	\
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	

	TEXT
0	cyclin dependent kinases cdks regulate variety...
1	abstract background non small cell lung cancer...
2	abstract background non small cell lung cancer...
3	recent evidence demonstrated acquired uniparen...
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 of output variati
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true,
# split the train data into train and cross validation by maintaining same distributi
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train,
```

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 data points in
         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 y_i in train data')
         plt.grid()
         plt.show()

         # ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.htm
         # -(train_class_distribution.values): the minus sign will give us in decreasing order
         sorted_yi = np.argsort(-train_class_distribution.values)
         for i in sorted_yi:
             print('Number of data points in class', i+1, ':', train_class_distribution.values[i])

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

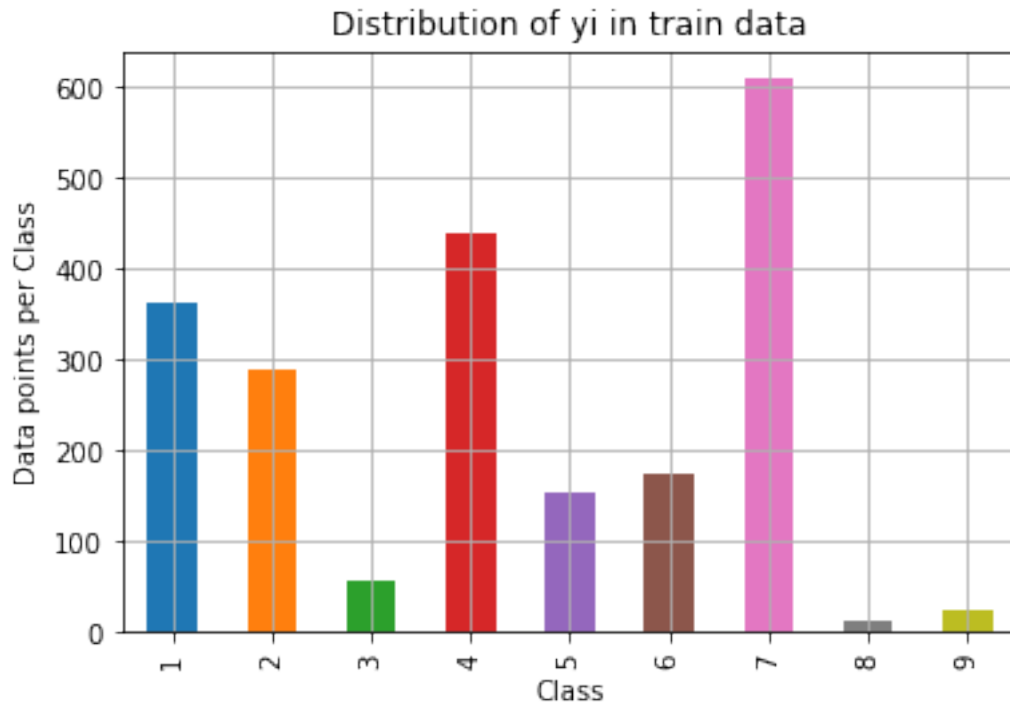
         # ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.htm
         # -(train_class_distribution.values): the minus sign will give us in decreasing order
         sorted_yi = np.argsort(-test_class_distribution.values)
         for i in sorted_yi:
             print('Number of data points in class', i+1, ':', test_class_distribution.values[i])
```

```

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.htm
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ': ', cv_class_distribution.values[i],

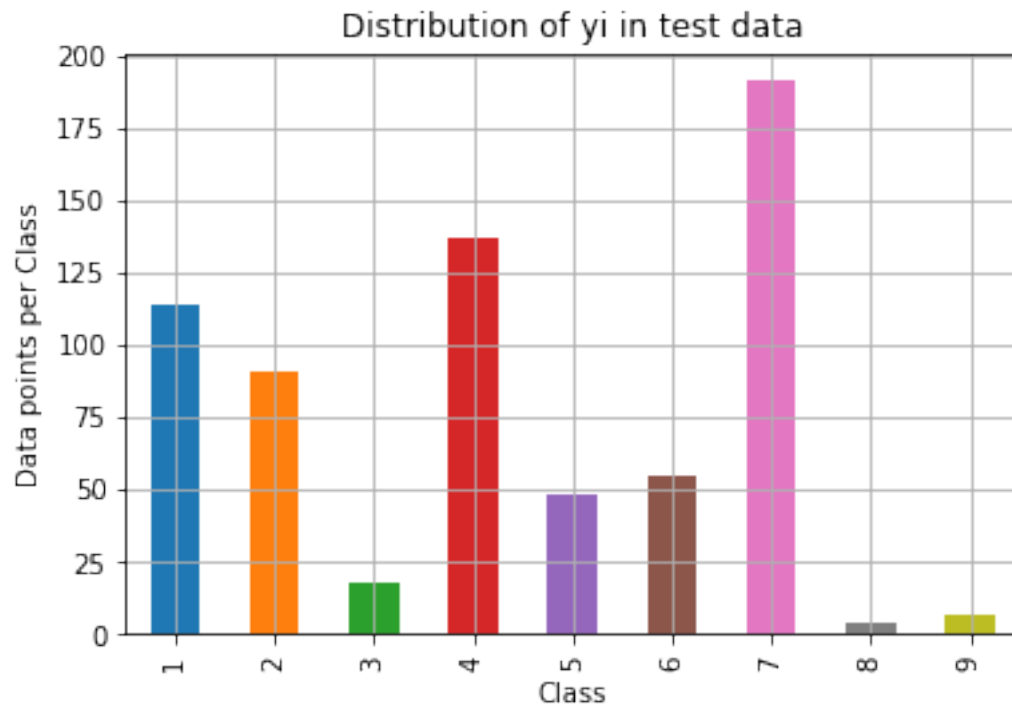
```



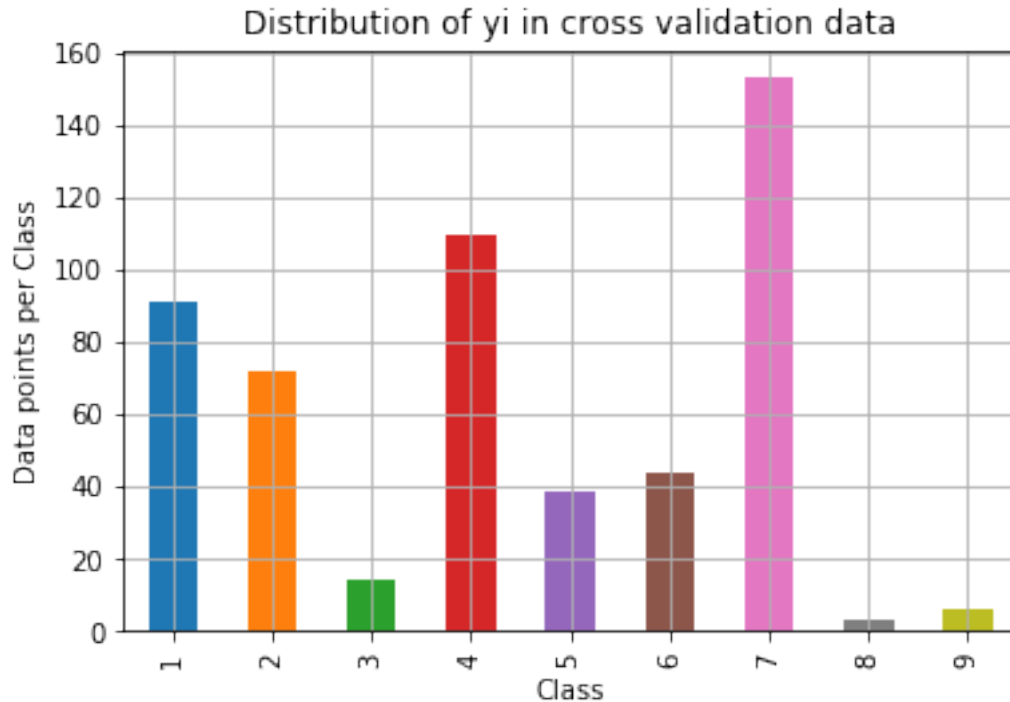
```

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 probabilities 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 class i are predicted as class j

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

    # C = [[1, 2],
  
```

```

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

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

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

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

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

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

```

In [14]: # we need to generate 9 numbers and the sum of numbers should be 1
one solution is to generate 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))

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

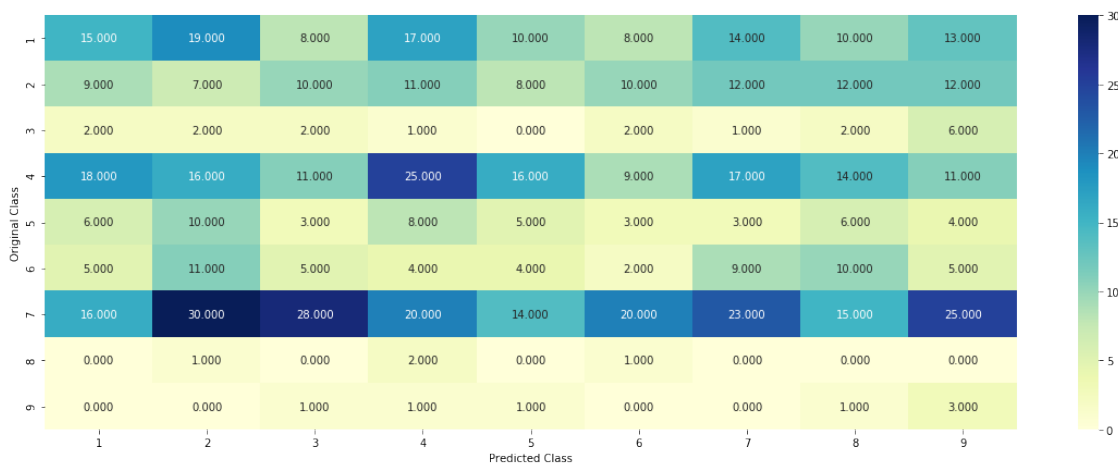
predicted_y =np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y+1)

```

Log loss on Cross Validation Data using Random Model 2.499882224480034

Log loss on Test Data using Random Model 2.468709131763825

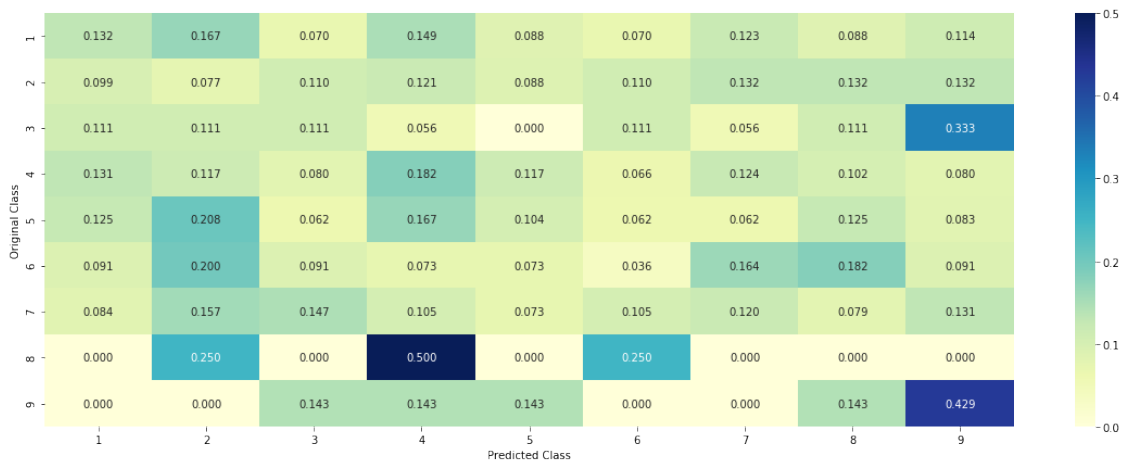
----- Confusion matrix -----



----- Precision matrix (Column 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 feature in train d
# build a vector (1*9) , the first element = (number of times it occurred in class1 +
# gv_dict is like a look up table, for every gene it store a (1*9) representation of
# for a value of feature in df:
# if it is in train data:
```

```

# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# -----

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

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

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

```

```

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

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

# cls_cnt.shape[0](numerator) will contain the number of time that partic
vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    # {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.06818181818181817,
    # 'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366,
    # 'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181
    # 'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608,
    # 'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917,
    # 'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0
    # 'BRAF': [0.066666666666666666, 0.17999999999999999, 0.07333333333333334,
    # ...
    # }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

```

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

```
In [16]: unique_genes = train_df['Gene'].value_counts()
         print('Number of Unique Genes :', unique_genes.shape[0])
         # the top 10 genes that occurred most
         print(unique_genes.head(10))
```

Number of Unique Genes : 240

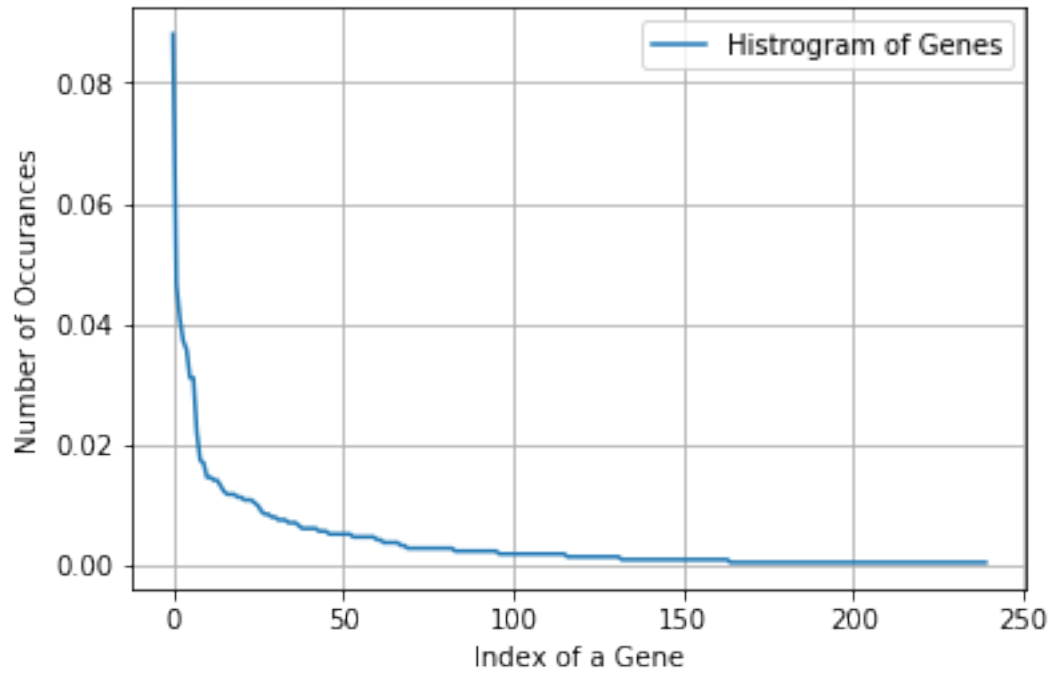
BRCA1	187
TP53	99
EGFR	87
PTEN	79
BRCA2	76
KIT	66
BRAF	66
ALK	47
PDGFRA	37
ERBB2	36

Name: Gene, dtype: int64

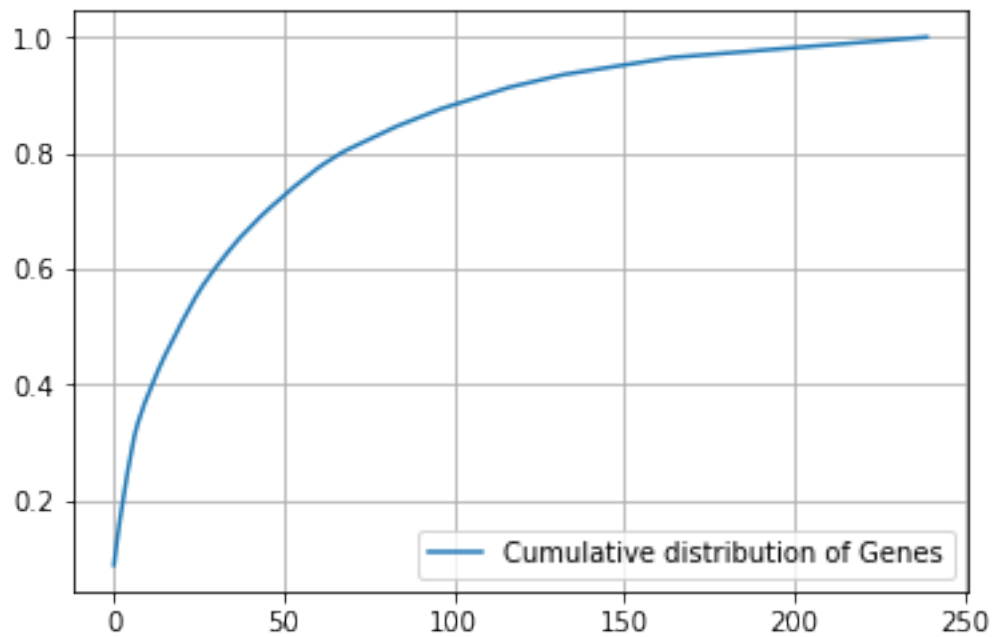
```
In [17]: print("Ans: There are", unique_genes.shape[0] , "different categories of genes in the t
```

Ans: There are 240 different categories of genes in the train data, and they are distributed as

```
In [18]: s = sum(unique_genes.values);
         h = unique_genes.values/s;
         plt.plot(h, label="Histogram 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/>

One hot Encoding

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, "Gene", train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
```

```
In [21]: print("train_gene_feature_responseCoding is converted feature using response coding method")
train_gene_feature_responseCoding is converted feature using response coding method. The shape of train_gene_feature_responseCoding is (1936, 1)
```

```
In [22]: # one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

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

```
Out[23]: 1936    CARD11
2703    BRAF
1070    ACVR1
1828    PPP2R1A
317     ROS1
Name: Gene, dtype: object
```

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

```
Out[24]: ['abl1',
'acvr1',
'ago2',
'akt1',
'akt2',
```

'akt3',
'alk',
'apc',
'ar',
'araf',
'arid1a',
'arid1b',
'arid2',
'arid5b',
'asx11',
'asx12',
'atm',
'atr',
'atrx',
'aurka',
'aurkb',
'axin1',
'b2m',
'bap1',
'bcl10',
'bcl2',
'bcl2l11',
'bcor',
'braf',
'brca1',
'brca2',
'brd4',
'brip1',
'btk',
'card11',
'carm1',
'casp8',
'cbl',
'ccnd1',
'ccnd2',
'ccnd3',
'ccne1',
'cdh1',
'cdk12',
'cdk4',
'cdk8',
'cdkn1a',
'cdkn1b',
'cdkn2a',
'cdkn2b',
'cdkn2c',
'cebpa',
'chek2',

'cic',
'crebbp',
'ctcf',
'ctla4',
'ctnnb1',
'ddr2',
'dicer1',
'dnmt3a',
'dnmt3b',
'dusp4',
'egfr',
'elf3',
'ep300',
'epas1',
'epcam',
'erbb2',
'erbb3',
'erbb4',
'ercc2',
'ercc3',
'ercc4',
'erg',
'errfi1',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fam58a',
'fanca',
'fancc',
'fat1',
'fbxw7',
'fgfr1',
'fgfr2',
'fgfr3',
'fgfr4',
'flt3',
'foxa1',
'foxl2',
'foxo1',
'fubp1',
'gata3',
'gnaq',
'gnas',
'h3f3a',
'hla',
'hnf1a',

'hras',
'idh1',
'idh2',
'igf1r',
'ikzf1',
'il7r',
'jak1',
'jak2',
'jun',
'kdm5a',
'kdm5c',
'kdm6a',
'kdr',
'keap1',
'kit',
'kmt2a',
'kmt2b',
'kmt2c',
'kmt2d',
'knstrn',
'kras',
'lats1',
'lats2',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
'mapk1',
'mdm4',
'med12',
'mef2b',
'men1',
'met',
'mga',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'mycn',
'myd88',
'myod1',
'ncor1',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',

'nkx2',
'notch1',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'nup93',
'pak1',
'pax8',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pik3r3',
'pim1',
'pms1',
'pms2',
'pole',
'ppm1d',
'ppp2r1a',
'ppp6c',
'prdm1',
'ptch1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rab35',
'rac1',
'rad21',
'rad50',
'rad51d',
'rad54l',
'raf1',
'rara',
'rasa1',
'rb1',
'rbm10',
'ret',
'rheb',
'rhoa',
'rictor',
'rit1',

```
'rnf43',  
'ros1',  
'rras2',  
'runx1',  
'rxra',  
'sdhb',  
'sdhc',  
'setd2',  
'sf3b1',  
'shq1',  
'smad2',  
'smad3',  
'smad4',  
'smarca4',  
'smarcb1',  
'smo',  
'sos1',  
'sox9',  
'spop',  
'src',  
'srsf2',  
'stag2',  
'stat3',  
'stk11',  
'tcf3',  
'tcf7l2',  
'tert',  
'tet1',  
'tet2',  
'tgfbr1',  
'tgfbr2',  
'tmprss2',  
'tp53',  
'tp53bp1',  
'tsc1',  
'tsc2',  
'u2af1',  
'vhl',  
'whsc1l1',  
'xpo1',  
'xrcc2',  
'yap1']
```

```
In [25]: print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method")  
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of train_gene_feature_onehotCoding is (1000, 40)
```

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/
# -----
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=optimal,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_gene_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-10))
    print('For values of alpha = ', i, "The log loss is:", log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-10))

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

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

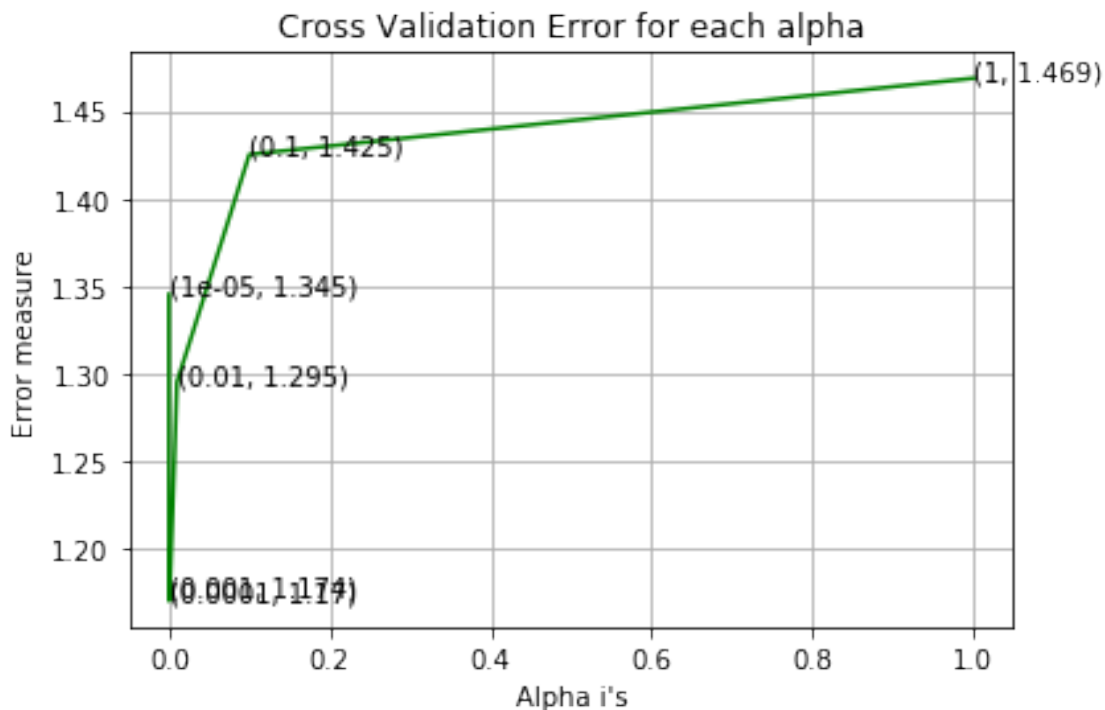


```

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(train_gene_feature_onehotCoding, predict_y))
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(cv_gene_feature_onehotCoding, predict_y))
predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(test_gene_feature_onehotCoding, predict_y))

```

For values of alpha = 1e-05 The log loss is: 1.3453319764674827
 For values of alpha = 0.0001 The log loss is: 1.1704048945574013
 For values of alpha = 0.001 The log loss is: 1.1735003576467538
 For values of alpha = 0.01 The log loss is: 1.295236930826065
 For values of alpha = 0.1 The log loss is: 1.425193970339485
 For values of alpha = 1 The log loss is: 1.4687040767862063



For values of best alpha = 0.0001 The train log loss is: 1.028354063497471
 For values of best alpha = 0.0001 The cross validation log loss is: 1.1704048945574013
 For values of best alpha = 0.0001 The test log loss is: 1.2218908056420403

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_g

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

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

Q6. How many data points in Test and CV datasets are covered by the 240 genes in train dataset?
Ans

1. In test data 645 out of 665 : 96.99248120300751
2. In cross validation data 523 out of 532 : 98.30827067669173

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 occurred most
print(unique_variations.head(10))
```

Number of Unique Variations : 1923

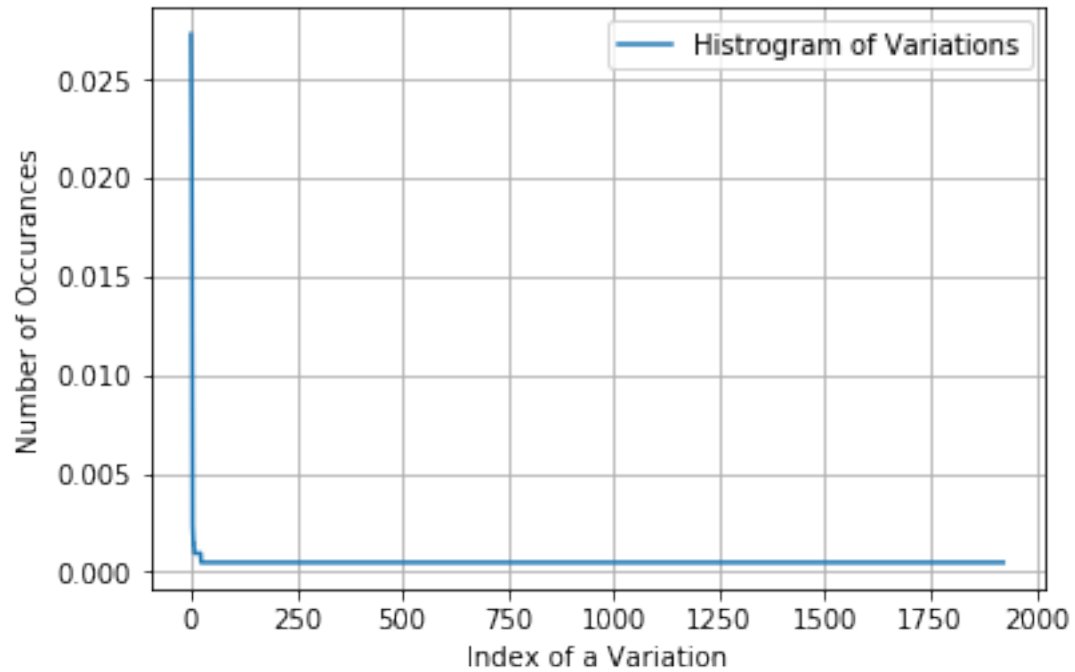
Truncating_Mutations	58
Deletion	52
Amplification	46
Fusions	22
Overexpression	5
G12V	4
Q61R	3
Q61H	3
E330K	2
R170W	2

Name: Variation, dtype: int64

```
In [29]: print("Ans: There are", unique_variations.shape[0] ,"different categories of variation
```

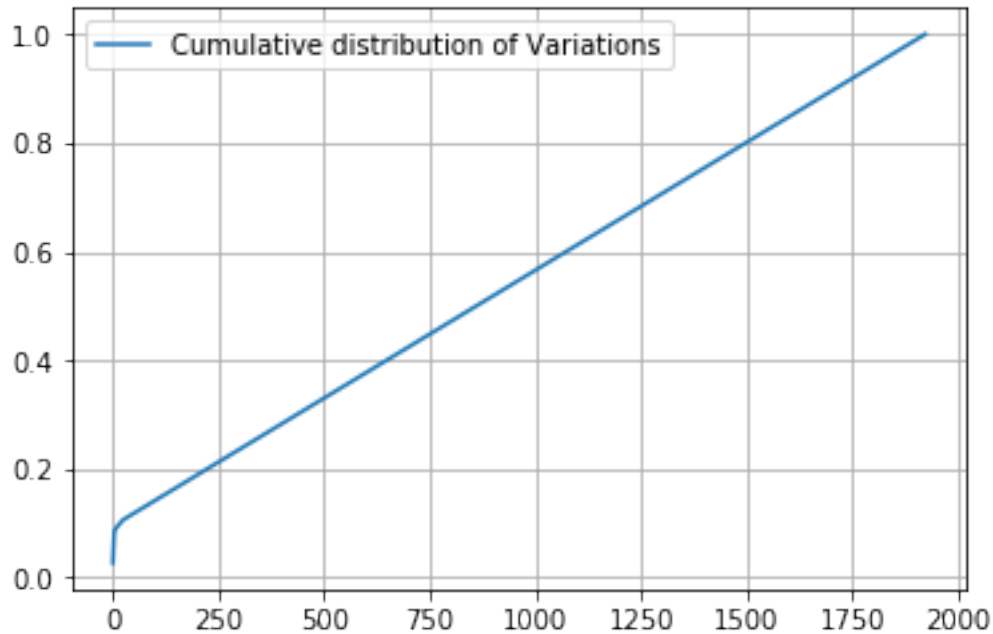
Ans: There are 1923 different categories of variations in the train data, and they are distributed as follows:

```
In [30]: s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histogram 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()

[0.02730697 0.05178908 0.07344633 ... 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/>

One hot Encoding

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", t
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", t
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_c

In [33]: print("train_variation_feature_responseCoding is a converted feature using the response
train_variation_feature_responseCoding is a converted feature using the response coding method

In [34]: # one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['V
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variati
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

```
In [35]: print("train_variation_feature_onehotEncoded is converted feature using the onne-hot e
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method.
```

Q10. How good is this Variation feature in predicting y_i ?
Let's build a model just like the earlier!

```
In [36]: alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
# -----
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

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

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

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

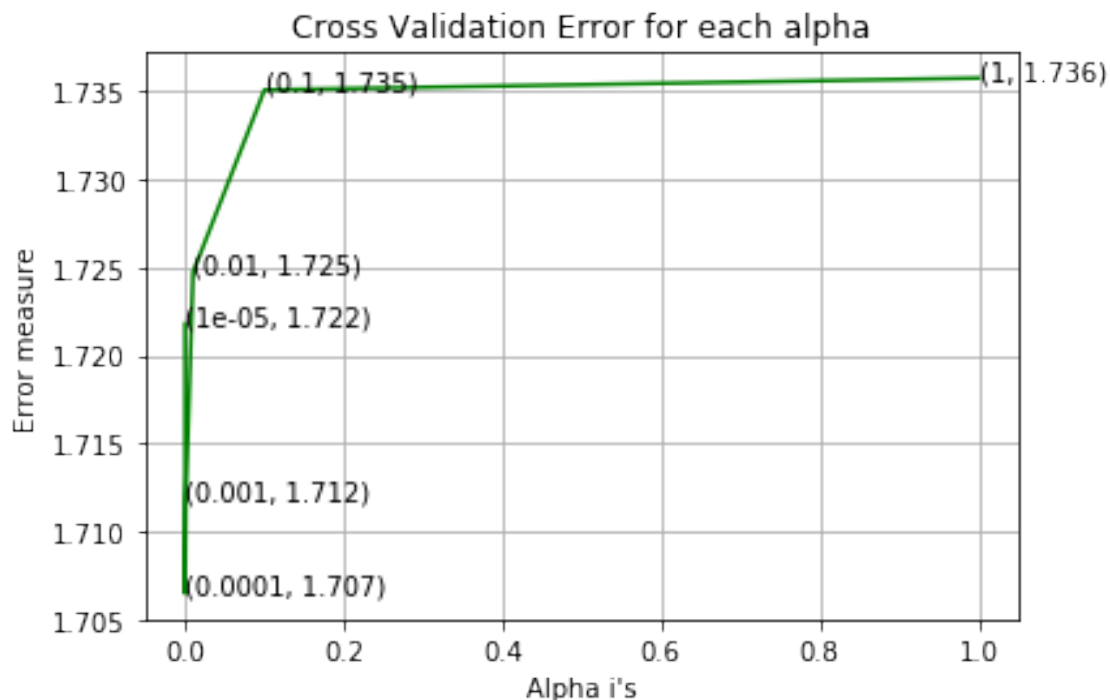
```

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

predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y))

```

For values of alpha = 1e-05 The log loss is: 1.7217932051555664
 For values of alpha = 0.0001 The log loss is: 1.706513132970686
 For values of alpha = 0.001 The log loss is: 1.7118556776699465
 For values of alpha = 0.01 The log loss is: 1.724707740166199
 For values of alpha = 0.1 The log loss is: 1.7350386294951128
 For values of alpha = 1 The log loss is: 1.735710604632296



For values of best alpha = 0.0001 The train log loss is: 0.7612026919194473
 For values of best alpha = 0.0001 The cross validation log loss is: 1.706513132970686

For values of best alpha = 0.0001 The test log loss is: 1.7252122115943214

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

Q12. How many data points are covered by total 1923 genes in test and cross validation data sets?

Ans

1. In test data 65 out of 665 : 9.774436090225564

2. In cross validation data 51 out of 532 : 9.586466165413533

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 predicting y_i?
5. Is the text feature stable across train, test and CV datasets?

```
In [38]: # cls_text is a data frame
          # for every row in data 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 [39]: import math
          #https://stackoverflow.com/a/1602964
          def get_text_responsecoding(df):
              text_feature_responseCoding = np.zeros((df.shape[0],9))
              for i in range(0,9):
                  row_index = 0
                  for index, row in df.iterrows():
                      sum_prob = 0
                      for word in row['TEXT'].split():
                          sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_dict.get(word,0)+10)))
```

```

        text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEXT']))
        row_index += 1
    return text_feature_responseCoding

```

```

In [40]: # building a CountVectorizer with all the words that occurred minimum 3 times in train
text_vectorizer = CountVectorizer( ngram_range=(1,2))
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

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

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

```

Total number of unique words in train data : 2421442

```

In [41]: dict_list = []
# dict_list=[] contains 9 dictionaries each corresponds to a class
for i in range(1,10):
    cls_text = train_df[train_df['Class']==i]
    # build a word dict based on the words in that class
    dict_list.append(extract_dictionary_paddle(cls_text))
    # append it to dict_list

# dict_list[i] is build on i'th class text data
# total_dict is build 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 [42]: #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 [43]: # 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_f
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_feat
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_re

In [44]: # don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

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

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

In [45]: #https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , reverse
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))

In [46]: # Number of words for a given frequency.
print(Counter(sorted_text_occur))

Counter({1: 1144341, 2: 387104, 3: 197001, 4: 132650, 6: 73691, 5: 71848, 7: 44489, 9: 41039, 8

In [47]: # Train a Logistic regression+Calibration model using text features whicha re on-hot
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
# -----
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

cv_log_error_array=[]

```

```

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

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

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

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

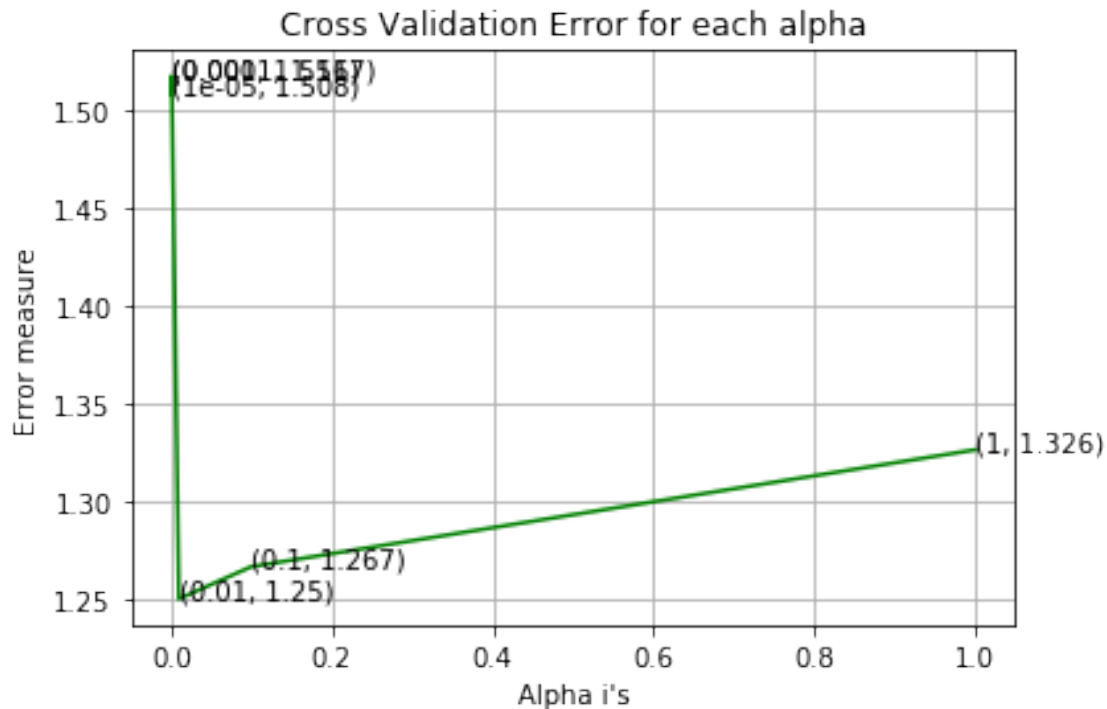
predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best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

```

```

For values of alpha = 1e-05 The log loss is: 1.5080944324194516
For values of alpha = 0.0001 The log loss is: 1.5170912609817575
For values of alpha = 0.001 The log loss is: 1.5164963753173664
For values of alpha = 0.01 The log loss is: 1.2503090714851124
For values of alpha = 0.1 The log loss is: 1.2667502887105992
For values of alpha = 1 The log loss is: 1.3264129806086336

```



For values of best alpha = 0.01 The train log loss is: 0.7583840272572777

For values of best alpha = 0.01 The cross validation log loss is: 1.2503090714851124

For values of best alpha = 0.01 The test log loss is: 1.2816471885933836

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

Ans. Yes, it seems like!

```
In [48]: def get_intersec_text(df):
df_text_vec = CountVectorizer()
df_text_fea = df_text_vec.fit_transform(df['TEXT'])
df_text_features = df_text_vec.get_feature_names()

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

```
In [49]: len1, len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train data")
len1, len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in train data")
```

79.119 % of word of test data appeared in train data
83.227 % of word of Cross Validation appeared in train data

4. Machine Learning Models

In [50]: *#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 will provide the array of probabilities belongs to
    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.size)
    plot_confusion_matrix(test_y, pred_y)
```

```
In [51]: 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 [52]: *# this function will be used just for naive bayes*
for the given indices, we will print the name of the features
and we will check whether the feature present in the test point text or not

```
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3)

    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(w
elif (v < fea1_len+fea2_len):
    word = var_vec.get_feature_names()[v-(fea1_len)]
    yes_no = True if word == var else False
    if yes_no:
        word_present += 1
        print(i, "variation feature [{}] present in test data point [{}].form
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(w

print("Out of the top ",no_features," features ", word_present, "are present in q

```

Stacking the three types of features

In [53]: *# merging gene, variance and text features*

```

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

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_fe
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_c

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

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

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

train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_var
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_varia
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_f

```

```

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature_re
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_re
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_response

```

```

In [54]: print("One hot encoding features :")
         print("(number of data points * number of features) in train data = ", train_x_onehotC
         print("(number of data points * number of features) in test data = ", test_x_onehotCo
         print("(number of data points * number of features) in cross validation data =", cv_x

```

```

One hot encoding features :
(number of data points * number of features) in train data = (2124, 2423633)
(number of data points * number of features) in test data = (665, 2423633)
(number of data points * number of features) in cross validation data = (532, 2423633)

```

```

In [55]: print(" Response encoding features :")
         print("(number of data points * number of features) in train data = ", train_x_respon
         print("(number of data points * number of features) in test data = ", test_x_response
         print("(number of data points * number of features) in cross validation data =", cv_x

```

```

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

```

- 4.1. Base Line Model
- 4.3. Logistic Regression
 - 4.3.1. With Class balancing
 - 4.3.1.1. Hyper paramter tuning

```

In [56]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
         # -----
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
         # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
         # class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

         # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modul

```

```

# -----
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv=
#
# 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', ra
    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=
    # to avoid rounding error while multiplying probabilities we use log-probability e
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', l
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_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
predict_y = sig_clf.predict_proba(test_x_onehotCoding)

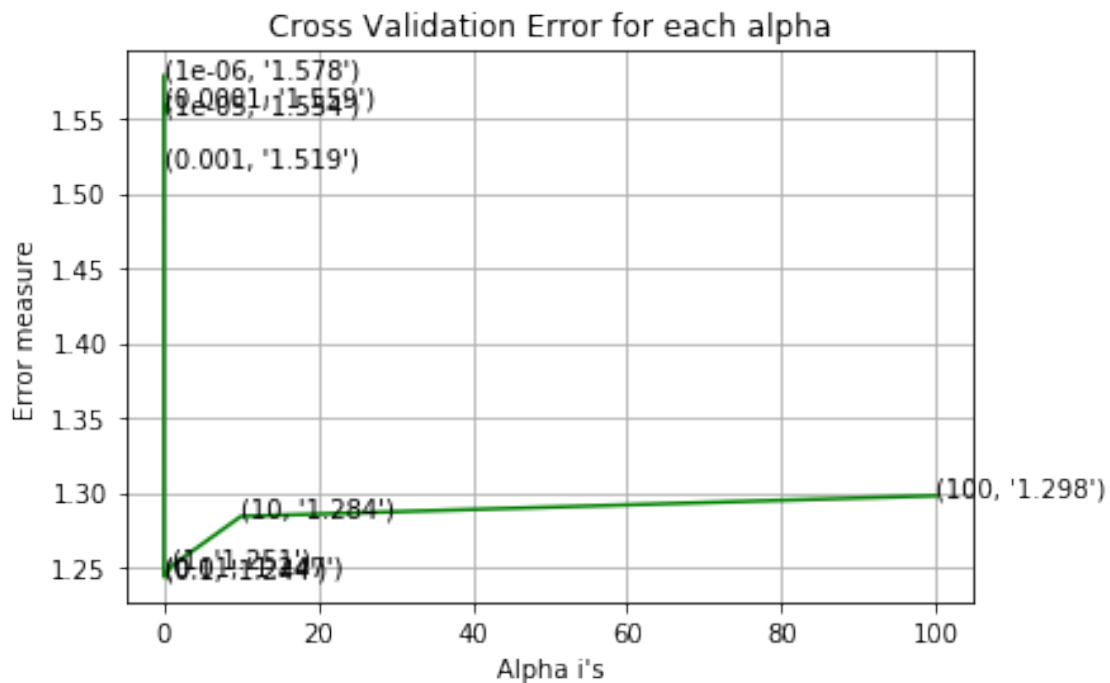
```

```

print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss)

for alpha = 1e-06
Log Loss : 1.5784519314044232
for alpha = 1e-05
Log Loss : 1.5535798236832523
for alpha = 0.0001
Log Loss : 1.559455365580777
for alpha = 0.001
Log Loss : 1.5192035170017437
for alpha = 0.01
Log Loss : 1.2466296046331007
for alpha = 0.1
Log Loss : 1.2436575994185388
for alpha = 1
Log Loss : 1.2510976945323962
for alpha = 10
Log Loss : 1.2842330822023926
for alpha = 100
Log Loss : 1.2979756288721849

```



```

For values of best alpha = 0.1 The train log loss is: 0.6345753364951522
For values of best alpha = 0.1 The cross validation log loss is: 1.2436575994185388
For values of best alpha = 0.1 The test log loss is: 1.2313731434870756

```


4.3.1.2. Testing the model with best hyper paramters

```
In [57]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated,
# -----
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=optimal,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

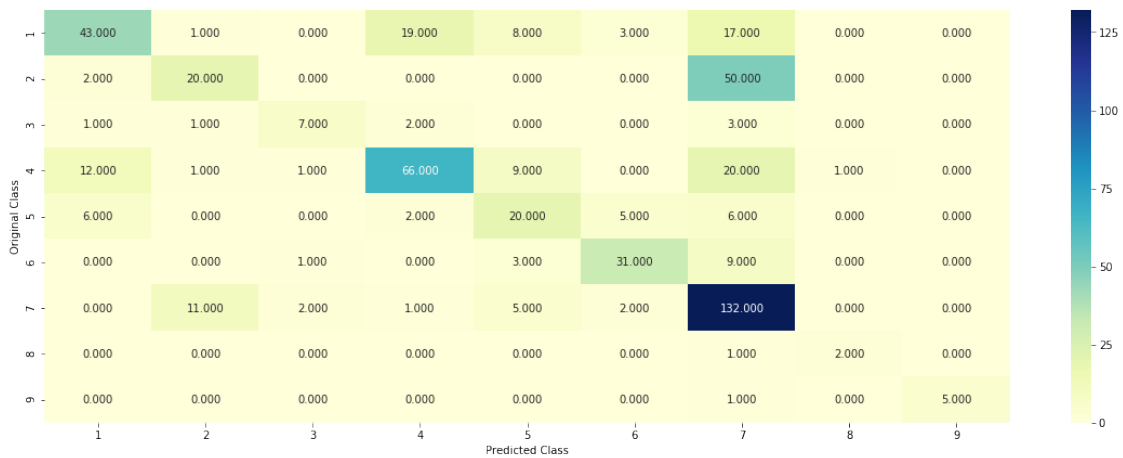
#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons,
#-----

clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', l1_ratio=0.15,
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y)
```

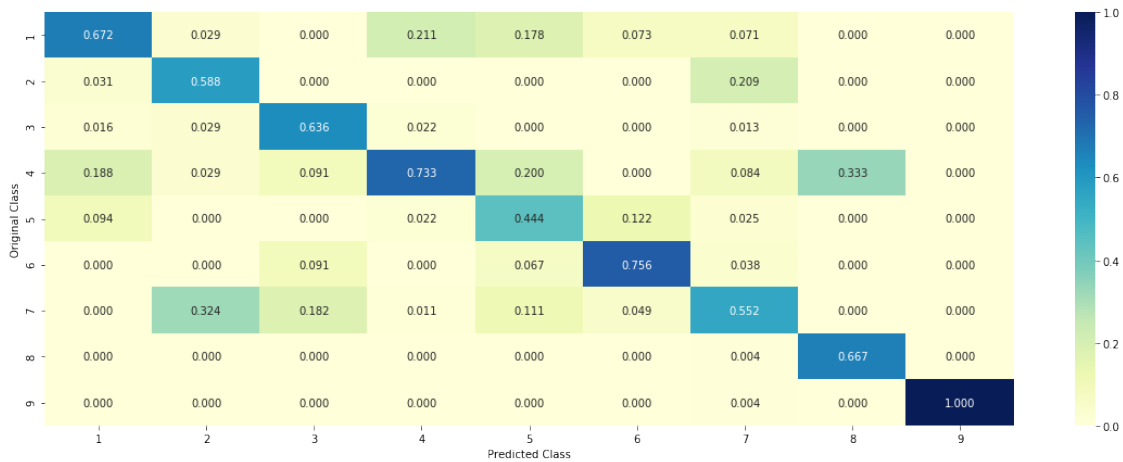
Log loss : 1.2436575994185388

Number of mis-classified points : 0.38721804511278196

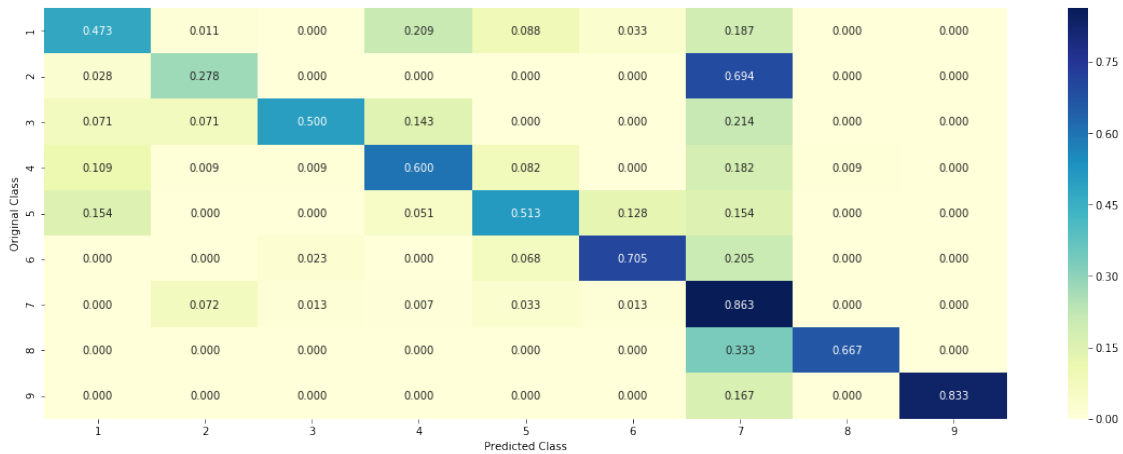
----- Confusion matrix -----



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



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



4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

```
In [58]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
# -----
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=optimal,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

```

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

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# -----
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv=5)
#
# 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")

```

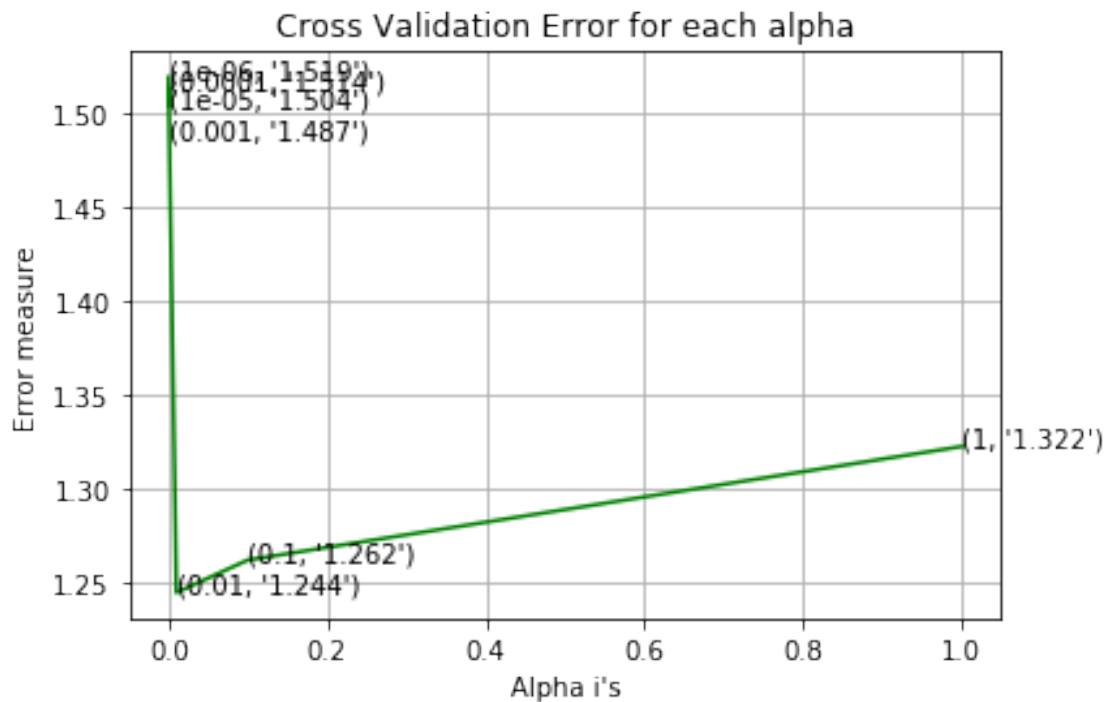
```

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_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_

for alpha = 1e-06
Log Loss : 1.5193190857429708
for alpha = 1e-05
Log Loss : 1.5040860029908616
for alpha = 0.0001
Log Loss : 1.513729626382753
for alpha = 0.001
Log Loss : 1.4873377786310866
for alpha = 0.01
Log Loss : 1.244217093326909
for alpha = 0.1
Log Loss : 1.261604297188673
for alpha = 1
Log Loss : 1.3222064826458313

```



For values of best alpha = 0.01 The train log loss is: 0.7396707300768426
 For values of best alpha = 0.01 The cross validation log loss is: 1.244217093326909
 For values of best alpha = 0.01 The test log loss is: 1.2815031233797223

4.3.2.2. Testing model with best hyper parameters

```
In [59]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated,
# -----
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=optimal,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

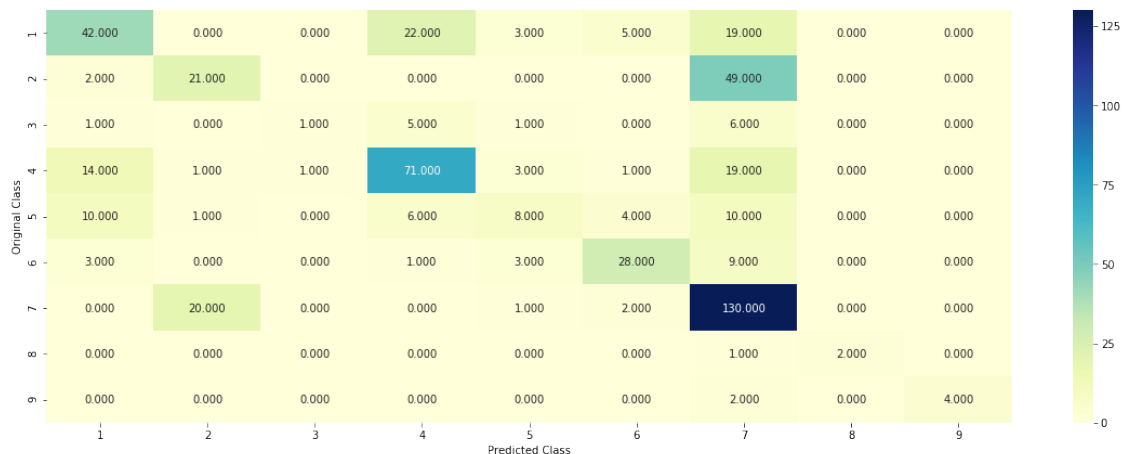
#-----
# video link:
#-----

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

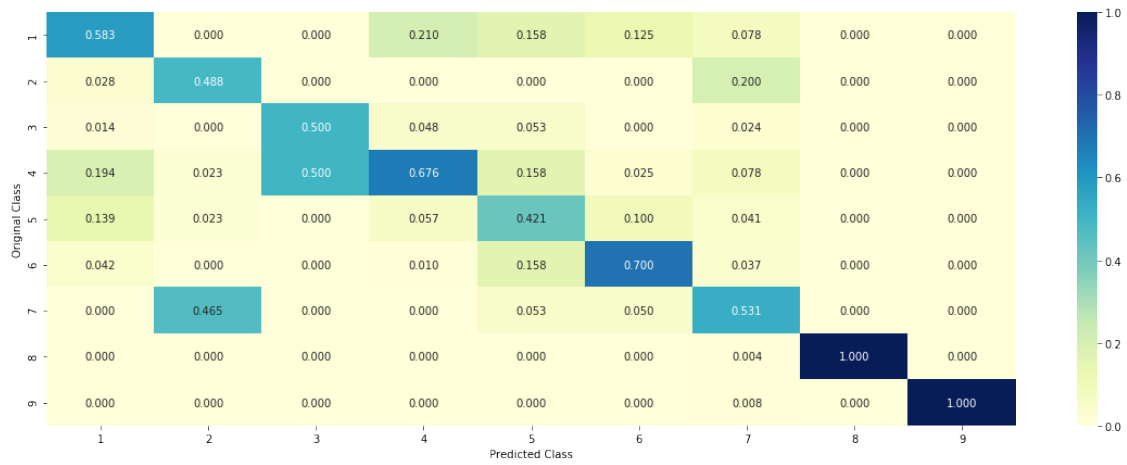
Log loss : 1.244217093326909

Number of mis-classified points : 0.42293233082706766

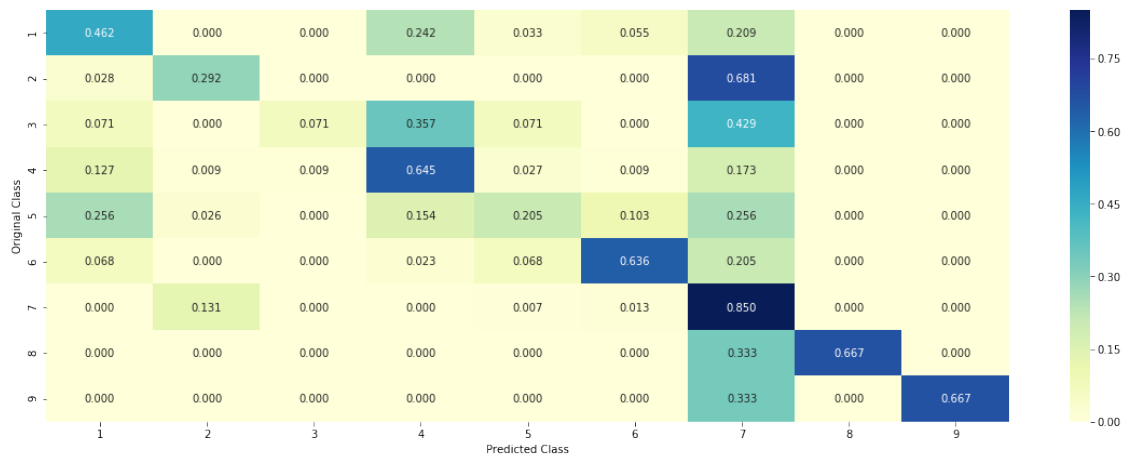
----- Confusion matrix -----



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



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



Conclusion : So here we easily seen that by using BIGRAM the los-loss remains the same but the miss-classified points increases. In the BIGRAM the result is not good as we seen in UNIGRAM