Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

- List item
- List item

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 [ ]:
```
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 import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")
import six
import sys
sys.modules['sklearn.externals.six'] = six
from mlxtend.classifier import StackingClassifier
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
```

In [ ]:
```
!gdown --id 1RmX5_q6D7rzoXD7nPUM_s8rKEf1KVMDi #training_text.zip download
!gdown --id 1bSQrw5WmDqqI8hBcr8Pflzatx4xCT0Ex #training_variants.zip download
```

```
Downloading...
From: https://drive.google.com/uc?id=1RmX5_q6D7rzoXD7nPUM_s8rKEf1KVMDi
To: /content/training_text.zip
100% 63.9M/63.9M [00:00<00:00, 198MB/s]
Downloading...
From: https://drive.google.com/uc?id=1bSQrw5WmDqqI8hBcr8Pflzatx4xCT0Ex
To: /content/training_variants.zip
100% 24.8k/24.8k [00:00<00:00, 17.4MB/s]
```

In [ ]:
```
!unzip training_text.zip
```

In [ ]:
```
!unzip training_variants.zip
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [ ]:
```
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[ ]:

	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 [ ]:
```
# 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[ ]:

	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 [ ]:
```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[ ]:
```
True
```

In [ ]:
```
# 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 [ ]:
```
#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 : 31.118156 seconds
```

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

Out[ ]:

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

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

Out[ ]:

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

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

Out[ ]:

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

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

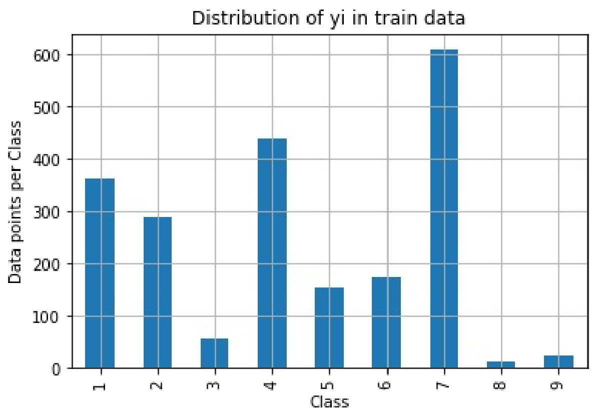
We split the data into train, test and cross validation data sets, preserving the ratio of class distribution in the original data set

In [ ]:
```
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])
```

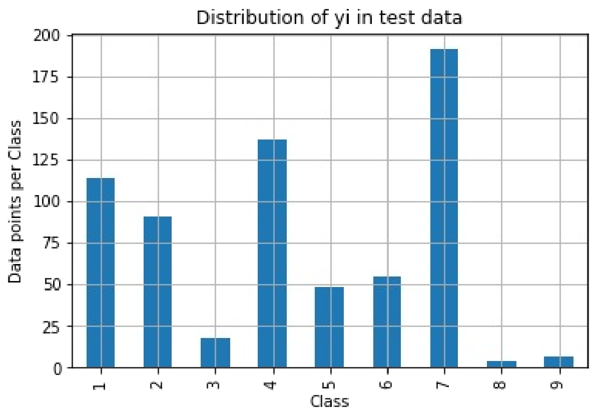
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

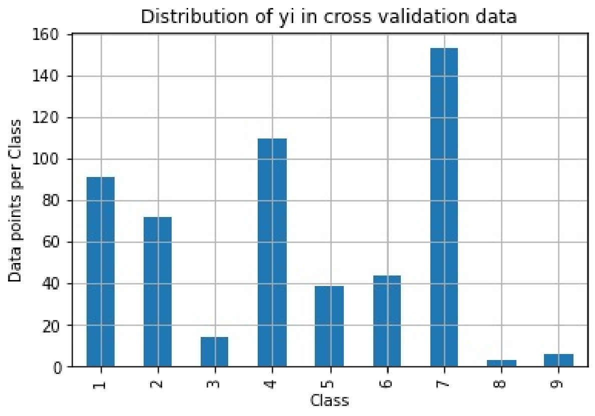
In [ ]:
```
# it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution = train_df['Class'].value_counts().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

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

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


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

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

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

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



```
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

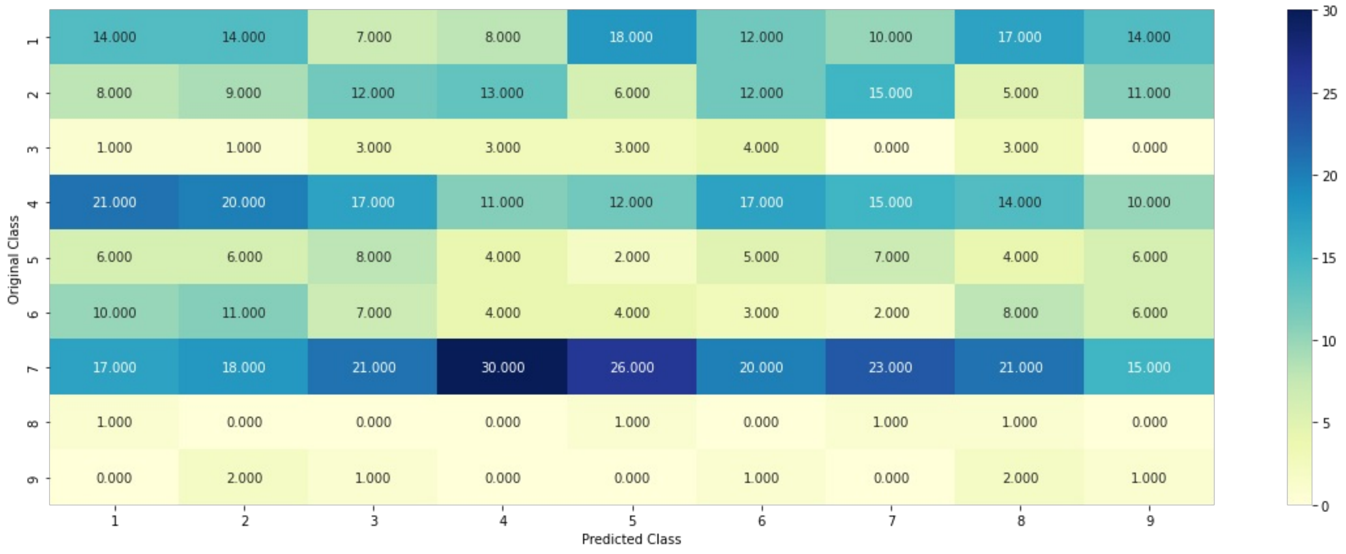
In a 'Random' Model, we generate the NINE class probabilites randomly such that they sum to 1.

In [ ]:

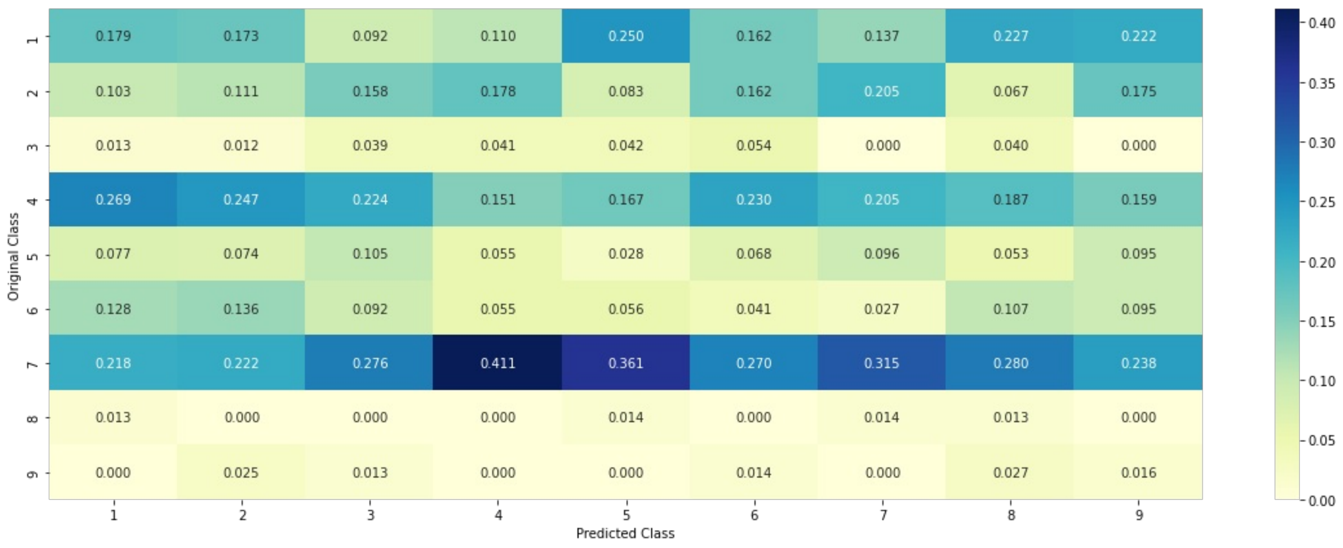
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

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

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

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

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

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

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

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

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

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


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

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

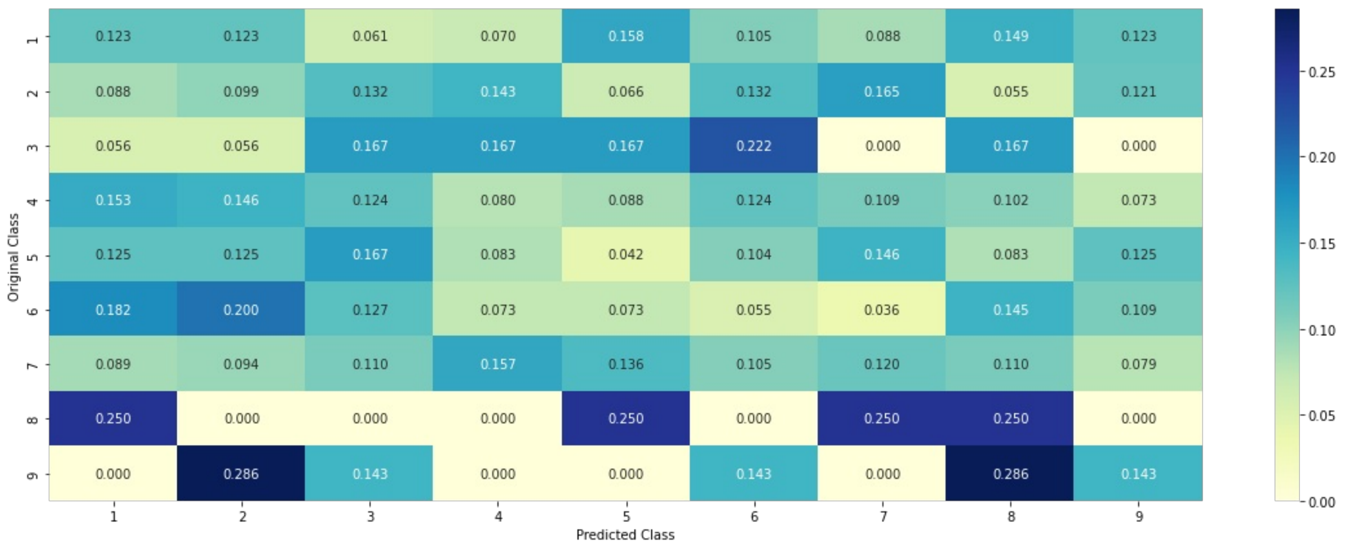
```
Log loss on Cross Validation Data using Random Model 2.4765341763991557
Log loss on Test Data using Random Model 2.4659259282136015
-------------------- Confusion matrix --------------------
```



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



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



3.3 Univariate Analysis

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

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

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

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

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

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177, 0.13636363636363635, 0.25,
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.27040816326530615, 0.061
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177, 0.068181818181818181
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.078787878787878782, 0.1
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.46540880503144655, 0.07
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.072847682119205295, 0.066
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.073333333333333334, 0.09
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

    # gv_fea: Gene_variation feature, it will contain the feature for each feature value in the data
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there in the train data then we w
    # 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?

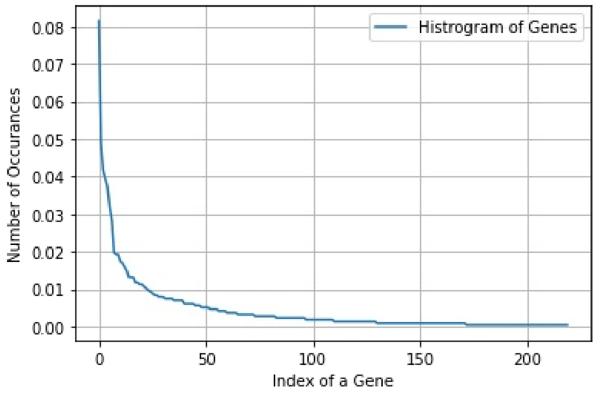
In [ ]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 220
BRCA1     173
TP53      104
PTEN       89
EGFR       84
BRCA2      79
KIT        68
BRAF       60
ERBB2      42
PDGFRA     41
ALK        41
Name: Gene, dtype: int64
```

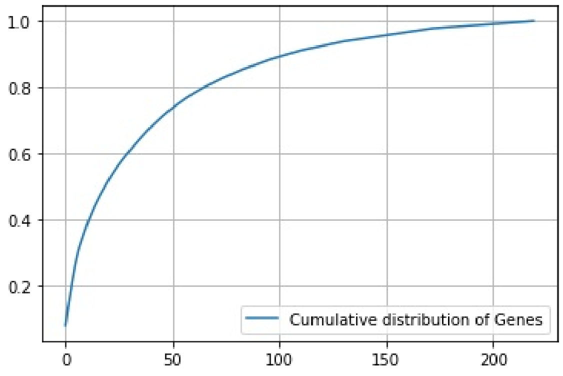
In [ ]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the train data, and they are d
```

```
Ans: There are 220 different categories of genes in the train data, and they are distibuted as follows
```

In [ ]:
```
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 [ ]:
```
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 [ ]:
```
#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 [ ]:
```
print("train_gene_feature_responseCoding is converted feature using respone coding method. The shape of gene fe
```

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

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

Out[ ]:
```
1134     MET
2335    JAK2
2730    BRAF
2750    BRAF
1363    AKT1
Name: Gene, dtype: object
```

In [ ]:
```
gene_vectorizer.get_feature_names()
```
Out[ ]:
```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'arid2',
 'asxl2',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'b2m',
 'bap1',
 'bcl10',
 'bcl2',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdkn1a',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cebpa',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctla4',
 'ctnnb1',
 'ddr2',
 'dicer1',
 'dnmt3a',
 'dnmt3b',
 'egfr',
 'elf3',
 'ep300',
 'epas1',
 'epcam',
 'erbb2',
 'erbb3',
 'erbb4',
 'ercc2',
 'ercc3',
 'ercc4',
 'erg',
 'esr1',
 'etv1',
 'etv6',
 'ewsr1',
 'ezh2',
 'fam58a',
 'fanca',
 'fancc',
 'fat1',
 'fbxw7',
 'fgfr1',
 'fgfr2',
 'fgfr3',
 'flt1',
 'flt3',
 'foxa1',
 'foxl2',
 'foxp1',
 'fubp1',
 'gata3',
 'gnaq',
 'gnas',
 'h3f3a',
 'hla',
 'hnf1a',
 'hras',
 'idh1',
 'idh2',
 'igf1r',
 'ikbke',
 'jak1',
 'jak2',
 'jun',
 'kdm5c',
 'kdm6a',
 'kdr',
 'keap1',
 'kit',
 'kmt2a',
 'kmt2b',
 'kmt2c',
 'kmt2d',
 'knstrn',
 'kras',
 'lats1',
 'lats2',
 'map2k1',
 'map2k2',
 'map2k4',
 'map3k1',
 'mapk1',
 'mdm2',
 'med12',
 'mef2b',
 'met',
 'mga',
 'mlh1',
 'mpl',
 'msh2',
 'msh6',
 'mtor',
 'myc',
 'mycn',
 'myd88',
 'myod1',
 'nf1',
 'nf2',
 'nfe2l2',
 'nfkbia',
 'nkx2',
 'notch1',
 'notch2',
 'npm1',
 'nras',
 'nsd1',
 'ntrk1',
 'ntrk2',
 'ntrk3',
 'nup93',
 'pax8',
 'pbrm1',
 'pdgfra',
 'pdgfrb',
 'pik3ca',
 'pik3cb',
 'pik3cd',
 'pik3r1',
 'pik3r2',
 'pik3r3',
 'pim1',
 'pms2',
 'pole',
 'ppm1d',
 'ppp2r1a',
 'ppp6c',
 'prdm1',
 'ptch1',
 'pten',
 'ptpn11',
 'ptprd',
 'ptprt',
 'rab35',
 'rac1',
 'rad21',
 'rad50',
 'rad51c',
 'raf1',
 'rara',
 'rasa1',
 'rb1',
 'rbm10',
 'ret',
 'rheb',
 'rhoa',
 'rit1',
 'ros1',
 'rras2',
 'runx1',
 'sdhb',
 'setd2',
 'sf3b1',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'srsf2',
 'stat3',
 'stk11',
 'tcf3',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'whsc1',
 'xpo1',
 'xrcc2',
 'yap1']
```

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature:
(2124, 219)
```

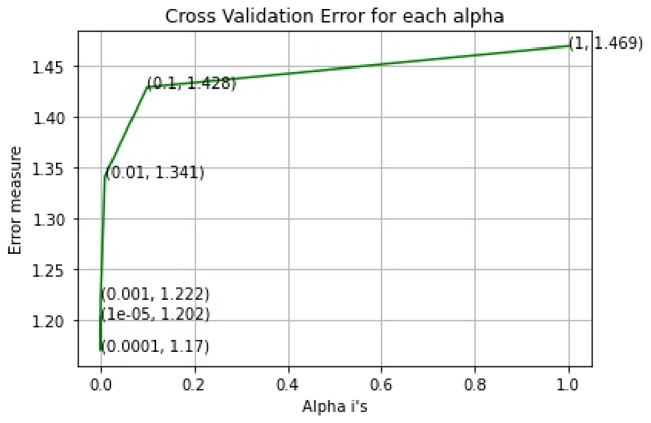
Q4. How good is this gene feature in predicting y_i?

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 [ ]:
```
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.SGDC
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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(y_train, predict_y, l
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(y_cv, pred
predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, lab
```

```
For values of alpha =  1e-05 The log loss is: 1.2019771057579447
For values of alpha =  0.0001 The log loss is: 1.1696293681066592
For values of alpha =  0.001 The log loss is: 1.2215125782552025
For values of alpha =  0.01 The log loss is: 1.3411650392265073
For values of alpha =  0.1 The log loss is: 1.428488435290755
For values of alpha =  1 The log loss is: 1.4690037382597259
```



```
For values of best alpha =  0.0001 The train log loss is: 0.9776547327315706
For values of best alpha =  0.0001 The cross validation log loss is: 1.1696293681066592
For values of best alpha =  0.0001 The test log loss is: 1.2233680150660557
```

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

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  220  genes in train dataset?
Ans
1. In test data 638 out of 665 : 95.93984962406014
2. In cross validation data 510 out of  532 : 95.86466165413535
```

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?

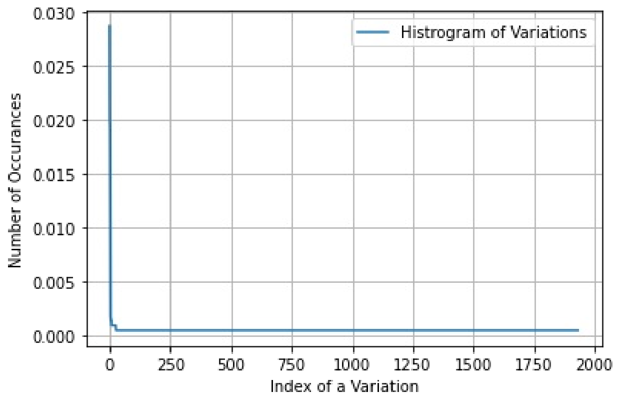
In [ ]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1932
Truncating_Mutations    61
Deletion                43
Amplification           41
Fusions                 24
Overexpression           4
Q61R                     3
G12V                     3
T58I                     3
R173C                    2
M1R                      2
Name: Variation, dtype: int64
```

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

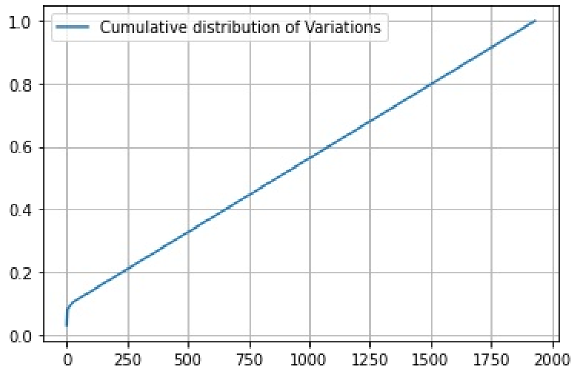
```
Ans: There are 1932 different categories of variations in the train data, and they are distibuted as follows
```

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

```
[0.0287194  0.04896422 0.06826742 ... 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 [ ]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_df))
```

In [ ]:
```
print("train_variation_feature_responseCoding is a converted feature using the response coding method. The shap
```
```
train_variation_feature_responseCoding is a converted feature using the response coding method. The shape of Va
riation feature: (2124, 9)
```

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

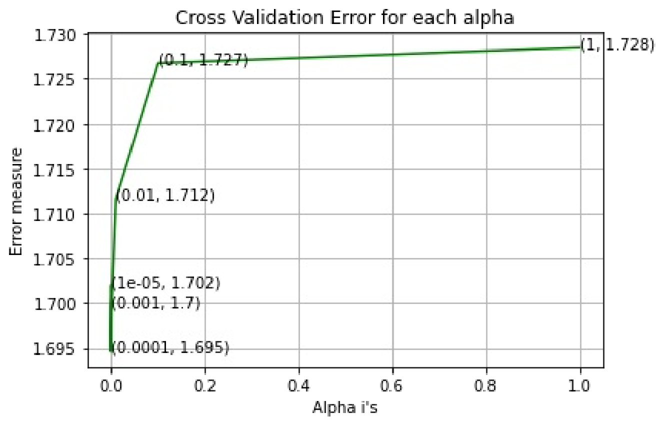
In [ ]:

In [ ]:
```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape
```
```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Var
iation feature: (2124, 1965)
```

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

Let's build a model just like the earlier!

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

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

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

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


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

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

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

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, l
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, pred
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, lab
```
```
For values of alpha =  1e-05 The log loss is: 1.7019089449865779
For values of alpha =  0.0001 The log loss is: 1.6945940959595986
For values of alpha =  0.001 The log loss is: 1.6996136531383927
For values of alpha =  0.01 The log loss is: 1.7115871142321066
For values of alpha =  0.1 The log loss is: 1.7267255831086643
For values of alpha =  1 The log loss is: 1.7284696694282644
```



```
For values of best alpha =  0.0001 The train log loss is: 0.695339821384064
For values of best alpha =  0.0001 The cross validation log loss is: 1.6945940959595986
For values of best alpha =  0.0001 The test log loss is: 1.711085381210239
```

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

```
Q12. How many data points are covered by total  1932  genes in test and cross validation data sets?
Ans
1. In test data 72 out of 665 : 10.827067669172932
2. In cross validation data 58 out of  532 : 10.902255639097744
```

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

In [ ]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = CountVectorizer(min_df=3)
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 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 number of times it occured
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))


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

```
Total number of unique words in train data : 53103
```

In [ ]:
```
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 [ ]:
```
#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 [ ]:
```
# 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(
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_feature_responseCoding.sum(axi
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_responseCoding.sum(axis=1)).
```

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

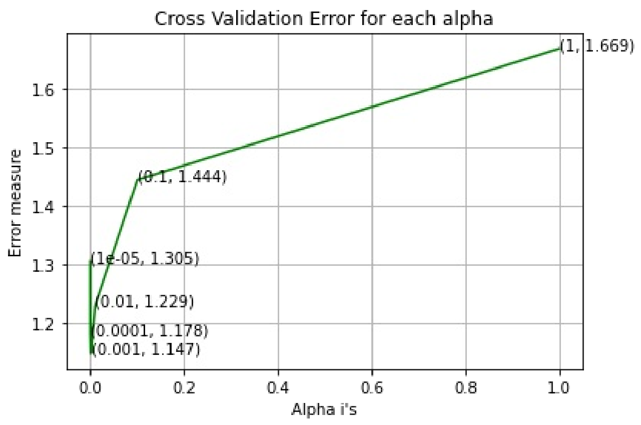
```
Counter({3: 5172, 4: 3630, 5: 3042, 6: 2871, 8: 1989, 7: 1842, 9: 1553, 11: 1417, 10: 1379, 12: 1171, 13: 1081,
15: 1021, 16: 840, 14: 801, 19: 700, 18: 621, 17: 590, 21: 574, 20: 539, 24: 460, 22: 440, 25: 414, 26: 410, 30
: 400, 31: 381, 23: 361, 37: 355, 27: 335, 28: 313, 33: 278, 32: 274, 36: 273, 29: 267, 35: 257, 53: 250, 38: 2
39, 34: 239, 40: 216, 42: 213, 39: 211, 44: 191, 45: 184, 46: 177, 48: 169, 51: 167, 50: 165, 49: 163, 41: 163,
54: 156, 43: 155, 57: 146, 52: 145, 60: 139, 55: 136, 72: 135, 56: 131, 62: 125, 61: 122, 47: 122, 66: 120, 58:
116, 65: 115, 63: 115, 67: 113, 64: 106, 59: 99, 78: 92, 69: 91, 81: 87, 79: 87, 70: 87, 77: 86, 86: 84, 80: 84
, 74: 84, 84: 81, 68: 79, 75: 77, 91: 73, 90: 73, 82: 73, 76: 73, 71: 70, 88: 69, 102: 67, 106: 64, 105: 63, 92
: 62, 87: 62, 108: 61, 100: 59, 96: 59, 94: 59, 73: 59, 83: 58, 89: 57, 85: 56, 114: 54, 93: 54, 103: 53, 98: 5
3, 124: 52, 101: 52, 99: 52, 120: 51, 97: 50, 111: 49, 113: 48, 112: 48, 107: 48, 121: 47, 109: 47, 144: 46, 12
9: 46, 115: 46, 128: 44, 126: 44, 116: 44, 110: 43, 134: 41, 132: 41, 95: 41, 127: 40, 153: 39, 135: 39, 136: 3
8, 125: 38, 143: 37, 130: 37, 187: 36, 180: 36, 146: 36, 165: 35, 157: 35, 150: 35, 160: 34, 117: 34, 166: 33,
137: 33, 122: 33, 104: 33, 155: 32, 133: 32, 192: 31, 147: 31, 142: 31, 123: 31, 173: 30, 162: 30, 140: 30, 118
: 30, 168: 29, 159: 29, 156: 29, 145: 29, 141: 29, 139: 29, 198: 28, 184: 28, 154: 28, 119: 28, 182: 27, 169: 2
7, 148: 27, 197: 26, 138: 26, 218: 25, 171: 25, 172: 24, 170: 24, 167: 24, 164: 24, 131: 24, 202: 23, 174: 23,
265: 22, 234: 22, 216: 22, 152: 22, 149: 22, 246: 21, 215: 21, 207: 21, 203: 21, 195: 21, 190: 21, 189: 21, 179
: 21, 177: 21, 258: 20, 243: 20, 227: 20, 223: 20, 208: 20, 194: 20, 193: 20, 183: 20, 176: 20, 300: 19, 255: 1
9, 241: 19, 228: 19, 224: 19, 196: 19, 186: 19, 175: 19, 163: 19, 161: 19, 158: 19, 320: 18, 263: 18, 257: 18,
242: 18, 229: 18, 219: 18, 213: 18, 210: 18, 200: 18, 199: 18, 188: 18, 178: 18, 334: 17, 232: 17, 212: 17, 204
: 17, 185: 17, 181: 17, 151: 17, 329: 16, 310: 16, 301: 16, 288: 16, 281: 16, 267: 16, 254: 16, 250: 16, 247: 1
6, 230: 16, 222: 16, 209: 16, 205: 16, 407: 15, 283: 15, 264: 15, 220: 15, 206: 15, 201: 15, 306: 14, 294: 14,
290: 14, 278: 14, 274: 14, 240: 14, 239: 14, 238: 14, 237: 14, 233: 14, 231: 14, 226: 14, 214: 14, 191: 14, 363
: 13, 351: 13, 318: 13, 316: 13, 312: 13, 302: 13, 297: 13, 296: 13, 291: 13, 277: 13, 225: 13, 211: 13, 350: 1
2, 324: 12, 313: 12, 292: 12, 260: 12, 259: 12, 235: 12, 221: 12, 435: 11, 429: 11, 389: 11, 379: 11, 378: 11,
339: 11, 327: 11, 314: 11, 303: 11, 299: 11, 289: 11, 286: 11, 285: 11, 282: 11, 279: 11, 276: 11, 275: 11, 272
: 11, 268: 11, 256: 11, 253: 11, 249: 11, 248: 11, 245: 11, 244: 11, 217: 11, 522: 10, 513: 10, 474: 10, 409: 1
0, 401: 10, 397: 10, 396: 10, 395: 10, 388: 10, 347: 10, 338: 10, 336: 10, 335: 10, 325: 10, 322: 10, 321: 10,
319: 10, 317: 10, 252: 10, 251: 10, 610: 9, 524: 9, 463: 9, 457: 9, 446: 9, 424: 9, 420: 9, 414: 9, 406: 9, 387
: 9, 380: 9, 365: 9, 358: 9, 354: 9, 341: 9, 337: 9, 328: 9, 315: 9, 309: 9, 308: 9, 305: 9, 287: 9, 284: 9, 28
0: 9, 266: 9, 261: 9, 864: 8, 671: 8, 649: 8, 648: 8, 604: 8, 590: 8, 547: 8, 489: 8, 473: 8, 470: 8, 452: 8, 4
34: 8, 423: 8, 418: 8, 398: 8, 394: 8, 386: 8, 376: 8, 374: 8, 373: 8, 353: 8, 352: 8, 348: 8, 345: 8, 343: 8,
342: 8, 333: 8, 326: 8, 323: 8, 307: 8, 304: 8, 295: 8, 271: 8, 269: 8, 802: 7, 697: 7, 691: 7, 623: 7, 570: 7,
544: 7, 532: 7, 528: 7, 525: 7, 485: 7, 472: 7, 466: 7, 460: 7, 456: 7, 451: 7, 449: 7, 442: 7, 415: 7, 413: 7,
385: 7, 383: 7, 381: 7, 366: 7, 355: 7, 344: 7, 331: 7, 298: 7, 293: 7, 270: 7, 262: 7, 799: 6, 753: 6, 664: 6,
656: 6, 624: 6, 581: 6, 559: 6, 557: 6, 542: 6, 540: 6, 526: 6, 519: 6, 515: 6, 511: 6, 500: 6, 499: 6, 482: 6,
461: 6, 459: 6, 438: 6, 436: 6, 430: 6, 427: 6, 425: 6, 408: 6, 404: 6, 402: 6, 400: 6, 399: 6, 392: 6, 384: 6,
377: 6, 372: 6, 369: 6, 368: 6, 360: 6, 357: 6, 356: 6, 346: 6, 340: 6, 330: 6, 273: 6, 236: 6, 3491: 5, 1213:
5, 1028: 5, 1016: 5, 920: 5, 907: 5, 857: 5, 813: 5, 812: 5, 785: 5, 781: 5, 763: 5, 750: 5, 740: 5, 729: 5, 70
7: 5, 704: 5, 699: 5, 698: 5, 666: 5, 660: 5, 651: 5, 645: 5, 638: 5, 637: 5, 625: 5, 622: 5, 603: 5, 580: 5, 5
69: 5, 566: 5, 564: 5, 555: 5, 553: 5, 550: 5, 546: 5, 536: 5, 533: 5, 531: 5, 523: 5, 510: 5, 507: 5, 501: 5,
498: 5, 496: 5, 493: 5, 486: 5, 481: 5, 476: 5, 471: 5, 468: 5, 467: 5, 464: 5, 447: 5, 445: 5, 444: 5, 443: 5,
440: 5, 433: 5, 426: 5, 421: 5, 417: 5, 416: 5, 412: 5, 411: 5, 410: 5, 405: 5, 403: 5, 393: 5, 390: 5, 370: 5,
364: 5, 362: 5, 361: 5, 359: 5, 349: 5, 332: 5, 3575: 4, 2922: 4, 2683: 4, 2320: 4, 1996: 4, 1738: 4, 1700: 4,
1543: 4, 1508: 4, 1504: 4, 1358: 4, 1307: 4, 1252: 4, 1237: 4, 1227: 4, 1218: 4, 1217: 4, 1216: 4, 1202: 4, 116
6: 4, 1149: 4, 1127: 4, 1052: 4, 1043: 4, 1026: 4, 972: 4, 968: 4, 952: 4, 948: 4, 940: 4, 939: 4, 925: 4, 902:
4, 896: 4, 894: 4, 893: 4, 875: 4, 869: 4, 865: 4, 858: 4, 851: 4, 820: 4, 814: 4, 808: 4, 798: 4, 774: 4, 765:
4, 761: 4, 759: 4, 756: 4, 752: 4, 751: 4, 738: 4, 731: 4, 719: 4, 710: 4, 703: 4, 701: 4, 694: 4, 693: 4, 683:
4, 676: 4, 670: 4, 654: 4, 653: 4, 652: 4, 650: 4, 640: 4, 630: 4, 627: 4, 621: 4, 619: 4, 617: 4, 612: 4, 611:
4, 606: 4, 601: 4, 598: 4, 596: 4, 586: 4, 583: 4, 579: 4, 575: 4, 568: 4, 567: 4, 560: 4, 558: 4, 556: 4, 549:
4, 545: 4, 543: 4, 541: 4, 539: 4, 538: 4, 535: 4, 534: 4, 516: 4, 514: 4, 512: 4, 509: 4, 508: 4, 506: 4, 503:
4, 495: 4, 494: 4, 487: 4, 484: 4, 477: 4, 469: 4, 462: 4, 455: 4, 422: 4, 419: 4, 382: 4, 375: 4, 4575: 3, 273
1: 3, 2412: 3, 2350: 3, 2306: 3, 2250: 3, 2189: 3, 2168: 3, 2156: 3, 2090: 3, 2043: 3, 2019: 3, 2011: 3, 1956:
3, 1939: 3, 1884: 3, 1871: 3, 1828: 3, 1659: 3, 1647: 3, 1646: 3, 1623: 3, 1613: 3, 1608: 3, 1587: 3, 1572: 3,
1536: 3, 1531: 3, 1511: 3, 1501: 3, 1481: 3, 1479: 3, 1444: 3, 1420: 3, 1407: 3, 1403: 3, 1395: 3, 1387: 3, 136
9: 3, 1366: 3, 1341: 3, 1324: 3, 1281: 3, 1279: 3, 1263: 3, 1261: 3, 1259: 3, 1203: 3, 1190: 3, 1181: 3, 1179:
3, 1176: 3, 1175: 3, 1170: 3, 1164: 3, 1159: 3, 1157: 3, 1154: 3, 1143: 3, 1122: 3, 1109: 3, 1100: 3, 1089: 3,
1088: 3, 1086: 3, 1083: 3, 1066: 3, 1054: 3, 1050: 3, 1048: 3, 1033: 3, 1029: 3, 1014: 3, 1009: 3, 1002: 3, 100
0: 3, 992: 3, 988: 3, 987: 3, 985: 3, 984: 3, 960: 3, 959: 3, 956: 3, 944: 3, 936: 3, 917: 3, 912: 3, 910: 3, 8
98: 3, 895: 3, 891: 3, 890: 3, 888: 3, 873: 3, 853: 3, 852: 3, 843: 3, 839: 3, 838: 3, 834: 3, 833: 3, 831: 3,
828: 3, 827: 3, 824: 3, 821: 3, 819: 3, 815: 3, 810: 3, 807: 3, 805: 3, 797: 3, 796: 3, 794: 3, 792: 3, 791: 3,
789: 3, 787: 3, 786: 3, 779: 3, 775: 3, 773: 3, 770: 3, 760: 3, 755: 3, 745: 3, 739: 3, 736: 3, 733: 3, 723: 3,
720: 3, 717: 3, 716: 3, 714: 3, 709: 3, 708: 3, 706: 3, 705: 3, 696: 3, 686: 3, 685: 3, 684: 3, 682: 3, 678: 3,
672: 3, 667: 3, 659: 3, 658: 3, 657: 3, 646: 3, 644: 3, 643: 3, 642: 3, 641: 3, 639: 3, 635: 3, 632: 3, 629: 3,
626: 3, 616: 3, 615: 3, 609: 3, 607: 3, 600: 3, 594: 3, 591: 3, 588: 3, 587: 3, 578: 3, 572: 3, 571: 3, 565: 3,
563: 3, 562: 3, 561: 3, 554: 3, 548: 3, 530: 3, 529: 3, 520: 3, 505: 3, 502: 3, 497: 3, 492: 3, 488: 3, 483: 3,
480: 3, 475: 3, 458: 3, 454: 3, 453: 3, 450: 3, 448: 3, 439: 3, 432: 3, 371: 3, 367: 3, 311: 3, 17201: 2, 9027:
2, 8096: 2, 6700: 2, 6114: 2, 6009: 2, 5667: 2, 5596: 2, 4410: 2, 4392: 2, 4357: 2, 4353: 2, 4339: 2, 4305: 2,
4279: 2, 4265: 2, 4225: 2, 4215: 2, 4156: 2, 4096: 2, 4050: 2, 3899: 2, 3788: 2, 3770: 2, 3722: 2, 3511: 2, 350
9: 2, 3505: 2, 3488: 2, 3476: 2, 3380: 2, 3347: 2, 3290: 2, 3268: 2, 3261: 2, 3256: 2, 3248: 2, 3241: 2, 3240:
2, 3202: 2, 3198: 2, 3196: 2, 3139: 2, 2964: 2, 2961: 2, 2853: 2, 2825: 2, 2816: 2, 2806: 2, 2771: 2, 2738: 2,
2720: 2, 2708: 2, 2695: 2, 2684: 2, 2660: 2, 2636: 2, 2614: 2, 2590: 2, 2574: 2, 2491: 2, 2476: 2, 2473: 2, 246
8: 2, 2449: 2, 2439: 2, 2438: 2, 2433: 2, 2431: 2, 2409: 2, 2402: 2, 2369: 2, 2349: 2, 2343: 2, 2337: 2, 2330:
2, 2322: 2, 2301: 2, 2298: 2, 2265: 2, 2257: 2, 2236: 2, 2234: 2, 2211: 2, 2195: 2, 2188: 2, 2174: 2, 2167: 2,
2125: 2, 2116: 2, 2093: 2, 2079: 2, 2073: 2, 2070: 2, 2067: 2, 2065: 2, 2061: 2, 2058: 2, 2039: 2, 2034: 2, 203
3: 2, 2027: 2, 2018: 2, 2009: 2, 1997: 2, 1991: 2, 1970: 2, 1967: 2, 1964: 2, 1944: 2, 1937: 2, 1928: 2, 1916:
2, 1913: 2, 1908: 2, 1906: 2, 1895: 2, 1883: 2, 1877: 2, 1875: 2, 1870: 2, 1869: 2, 1868: 2, 1860: 2, 1859: 2,
1856: 2, 1838: 2, 1836: 2, 1825: 2, 1815: 2, 1814: 2, 1813: 2, 1807: 2, 1802: 2, 1799: 2, 1796: 2, 1793: 2, 179
2: 2, 1790: 2, 1773: 2, 1770: 2, 1749: 2, 1740: 2, 1729: 2, 1724: 2, 1723: 2, 1714: 2, 1706: 2, 1704: 2, 1682:
2, 1681: 2, 1668: 2, 1648: 2, 1638: 2, 1634: 2, 1622: 2, 1621: 2, 1614: 2, 1606: 2, 1605: 2, 1601: 2, 1598: 2,
1596: 2, 1592: 2, 1586: 2, 1584: 2, 1567: 2, 1564: 2, 1552: 2, 1546: 2, 1542: 2, 1539: 2, 1523: 2, 1521: 2, 151
9: 2, 1518: 2, 1515: 2, 1495: 2, 1486: 2, 1474: 2, 1471: 2, 1464: 2, 1447: 2, 1442: 2, 1440: 2, 1435: 2, 1434:
2, 1431: 2, 1425: 2, 1422: 2, 1418: 2, 1417: 2, 1406: 2, 1401: 2, 1400: 2, 1398: 2, 1396: 2, 1389: 2, 1385: 2,
1375: 2, 1370: 2, 1368: 2, 1356: 2, 1355: 2, 1354: 2, 1349: 2, 1343: 2, 1342: 2, 1336: 2, 1333: 2, 1329: 2, 132
7: 2, 1326: 2, 1323: 2, 1320: 2, 1318: 2, 1313: 2, 1312: 2, 1305: 2, 1298: 2, 1296: 2, 1294: 2, 1287: 2, 1286:
2, 1285: 2, 1280: 2, 1277: 2, 1276: 2, 1275: 2, 1271: 2, 1269: 2, 1254: 2, 1253: 2, 1248: 2, 1242: 2, 1241: 2,
1239: 2, 1235: 2, 1234: 2, 1229: 2, 1228: 2, 1226: 2, 1225: 2, 1222: 2, 1205: 2, 1204: 2, 1200: 2, 1192: 2, 118
9: 2, 1186: 2, 1174: 2, 1172: 2, 1171: 2, 1151: 2, 1147: 2, 1142: 2, 1136: 2, 1130: 2, 1128: 2, 1121: 2, 1115:
2, 1114: 2, 1111: 2, 1105: 2, 1104: 2, 1101: 2, 1084: 2, 1079: 2, 1077: 2, 1076: 2, 1074: 2, 1058: 2, 1057: 2,
1055: 2, 1047: 2, 1042: 2, 1039: 2, 1032: 2, 1031: 2, 1027: 2, 1020: 2, 1018: 2, 1011: 2, 1005: 2, 999: 2, 996:
2, 995: 2, 993: 2, 991: 2, 981: 2, 980: 2, 978: 2, 975: 2, 967: 2, 966: 2, 965: 2, 964: 2, 963: 2, 951: 2, 950:
2, 943: 2, 942: 2, 941: 2, 935: 2, 933: 2, 932: 2, 928: 2, 924: 2, 919: 2, 905: 2, 904: 2, 892: 2, 887: 2, 884:
2, 882: 2, 881: 2, 880: 2, 878: 2, 877: 2, 876: 2, 874: 2, 872: 2, 870: 2, 868: 2, 866: 2, 863: 2, 861: 2, 855:
2, 854: 2, 850: 2, 846: 2, 845: 2, 832: 2, 830: 2, 829: 2, 825: 2, 823: 2, 818: 2, 817: 2, 816: 2, 809: 2, 803:
2, 801: 2, 800: 2, 793: 2, 780: 2, 778: 2, 776: 2, 772: 2, 769: 2, 768: 2, 767: 2, 764: 2, 757: 2, 754: 2, 749:
2, 747: 2, 743: 2, 741: 2, 737: 2, 732: 2, 730: 2, 727: 2, 726: 2, 724: 2, 722: 2, 718: 2, 715: 2, 712: 2, 700:
2, 695: 2, 690: 2, 689: 2, 688: 2, 687: 2, 680: 2, 679: 2, 675: 2, 673: 2, 669: 2, 668: 2, 665: 2, 663: 2, 655:
2, 647: 2, 636: 2, 634: 2, 631: 2, 620: 2, 614: 2, 605: 2, 599: 2, 597: 2, 593: 2, 589: 2, 585: 2, 576: 2, 574:
2, 551: 2, 537: 2, 521: 2, 518: 2, 504: 2, 491: 2, 490: 2, 479: 2, 478: 2, 465: 2, 437: 2, 431: 2, 428: 2, 391:
2, 153902: 1, 118327: 1, 80423: 1, 69224: 1, 68076: 1, 65974: 1, 65764: 1, 63176: 1, 62544: 1, 56436: 1, 55250:
1, 49694: 1, 48988: 1, 47897: 1, 46756: 1, 44827: 1, 42643: 1, 42453: 1, 41999: 1, 41489: 1, 41074: 1, 40551: 1
, 40169: 1, 39942: 1, 38566: 1, 38430: 1, 36660: 1, 36288: 1, 35930: 1, 34321: 1, 34205: 1, 33876: 1, 33633: 1,
33321: 1, 32593: 1, 31895: 1, 29644: 1, 28694: 1, 27984: 1, 27644: 1, 26727: 1, 26556: 1, 26343: 1, 25973: 1, 2
5949: 1, 25468: 1, 24846: 1, 24612: 1, 24542: 1, 24270: 1, 24262: 1, 24089: 1, 23986: 1, 22574: 1, 22396: 1, 22
288: 1, 21914: 1, 21809: 1, 21652: 1, 21641: 1, 21274: 1, 21183: 1, 20429: 1, 20375: 1, 20003: 1, 19840: 1, 197
47: 1, 19738: 1, 19621: 1, 19573: 1, 19166: 1, 19117: 1, 18873: 1, 18580: 1, 18546: 1, 18379: 1, 18288: 1, 1821
3: 1, 18205: 1, 18192: 1, 18138: 1, 18095: 1, 17947: 1, 17910: 1, 17776: 1, 17754: 1, 17400: 1, 17300: 1, 17272
: 1, 17158: 1, 17146: 1, 16966: 1, 16785: 1, 16740: 1, 16576: 1, 16335: 1, 16328: 1, 16220: 1, 15928: 1, 15916:
1, 15879: 1, 15836: 1, 15796: 1, 15551: 1, 15524: 1, 15489: 1, 15467: 1, 15387: 1, 15035: 1, 15016: 1, 15000: 1
, 14922: 1, 14652: 1, 14637: 1, 14501: 1, 14390: 1, 14343: 1, 13989: 1, 13893: 1, 13876: 1, 13772: 1, 13681: 1,
13601: 1, 13569: 1, 13484: 1, 13463: 1, 13400: 1, 13310: 1, 13276: 1, 13248: 1, 13175: 1, 13148: 1, 13085: 1, 1
3070: 1, 12957: 1, 12778: 1, 12776: 1, 12726: 1, 12723: 1, 12722: 1, 12683: 1, 12651: 1, 12555: 1, 12460: 1, 12
415: 1, 12405: 1, 12353: 1, 12340: 1, 12336: 1, 12328: 1, 12327: 1, 12321: 1, 12319: 1, 12281: 1, 12250: 1, 122
42: 1, 12226: 1, 12225: 1, 12166: 1, 12161: 1, 12028: 1, 12010: 1, 11999: 1, 11991: 1, 11985: 1, 11943: 1, 1192
```

```
9: 1, 11900: 1, 11740: 1, 11726: 1, 11702: 1, 11693: 1, 11624: 1, 11608: 1, 11517: 1, 11500: 1, 11469: 1, 11405
: 1, 11334: 1, 11041: 1, 11012: 1, 10960: 1, 10940: 1, 10893: 1, 10842: 1, 10812: 1, 10649: 1, 10613: 1, 10609:
1, 10598: 1, 10555: 1, 10490: 1, 10417: 1, 10351: 1, 10333: 1, 10307: 1, 10281: 1, 10256: 1, 10185: 1, 10174: 1
, 10172: 1, 10152: 1, 10116: 1, 10109: 1, 10015: 1, 9993: 1, 9962: 1, 9884: 1, 9880: 1, 9824: 1, 9823: 1, 9783:
1, 9603: 1, 9585: 1, 9582: 1, 9560: 1, 9445: 1, 9429: 1, 9416: 1, 9412: 1, 9385: 1, 9317: 1, 9305: 1, 9272: 1,
9185: 1, 9172: 1, 9138: 1, 9114: 1, 9036: 1, 9024: 1, 9010: 1, 8972: 1, 8933: 1, 8911: 1, 8871: 1, 8844: 1, 882
5: 1, 8823: 1, 8802: 1, 8789: 1, 8744: 1, 8737: 1, 8727: 1, 8640: 1, 8538: 1, 8533: 1, 8498: 1, 8481: 1, 8461:
1, 8458: 1, 8437: 1, 8429: 1, 8387: 1, 8377: 1, 8371: 1, 8360: 1, 8335: 1, 8293: 1, 8291: 1, 8279: 1, 8254: 1,
8253: 1, 8231: 1, 8227: 1, 8175: 1, 8070: 1, 8049: 1, 8036: 1, 8019: 1, 8011: 1, 8002: 1, 8001: 1, 7991: 1, 794
5: 1, 7929: 1, 7903: 1, 7866: 1, 7816: 1, 7815: 1, 7808: 1, 7784: 1, 7769: 1, 7766: 1, 7759: 1, 7748: 1, 7737:
1, 7729: 1, 7703: 1, 7667: 1, 7641: 1, 7637: 1, 7623: 1, 7600: 1, 7596: 1, 7554: 1, 7544: 1, 7490: 1, 7485: 1,
7473: 1, 7446: 1, 7357: 1, 7332: 1, 7328: 1, 7326: 1, 7319: 1, 7308: 1, 7305: 1, 7290: 1, 7260: 1, 7257: 1, 724
5: 1, 7243: 1, 7222: 1, 7219: 1, 7217: 1, 7210: 1, 7179: 1, 7177: 1, 7114: 1, 7096: 1, 7075: 1, 7068: 1, 7050:
1, 7048: 1, 7017: 1, 7000: 1, 6936: 1, 6929: 1, 6920: 1, 6912: 1, 6871: 1, 6862: 1, 6819: 1, 6800: 1, 6799: 1,
6791: 1, 6755: 1, 6745: 1, 6737: 1, 6730: 1, 6729: 1, 6720: 1, 6713: 1, 6689: 1, 6671: 1, 6638: 1, 6633: 1, 662
9: 1, 6595: 1, 6590: 1, 6566: 1, 6559: 1, 6552: 1, 6551: 1, 6524: 1, 6481: 1, 6475: 1, 6458: 1, 6448: 1, 6435:
1, 6410: 1, 6408: 1, 6407: 1, 6378: 1, 6374: 1, 6365: 1, 6361: 1, 6360: 1, 6357: 1, 6353: 1, 6323: 1, 6296: 1,
6295: 1, 6269: 1, 6238: 1, 6221: 1, 6194: 1, 6172: 1, 6151: 1, 6149: 1, 6139: 1, 6125: 1, 6109: 1, 6100: 1, 609
9: 1, 6085: 1, 6065: 1, 6058: 1, 6046: 1, 6042: 1, 6041: 1, 6039: 1, 6022: 1, 6003: 1, 6002: 1, 5964: 1, 5954:
1, 5946: 1, 5933: 1, 5908: 1, 5907: 1, 5900: 1, 5867: 1, 5844: 1, 5825: 1, 5817: 1, 5811: 1, 5769: 1, 5760: 1,
5753: 1, 5752: 1, 5749: 1, 5748: 1, 5739: 1, 5700: 1, 5693: 1, 5690: 1, 5666: 1, 5641: 1, 5632: 1, 5615: 1, 560
1: 1, 5587: 1, 5567: 1, 5549: 1, 5546: 1, 5545: 1, 5532: 1, 5517: 1, 5514: 1, 5500: 1, 5468: 1, 5449: 1, 5417:
1, 5383: 1, 5375: 1, 5363: 1, 5333: 1, 5325: 1, 5304: 1, 5302: 1, 5300: 1, 5299: 1, 5286: 1, 5273: 1, 5269: 1,
5259: 1, 5258: 1, 5254: 1, 5250: 1, 5245: 1, 5228: 1, 5223: 1, 5214: 1, 5211: 1, 5199: 1, 5195: 1, 5194: 1, 519
3: 1, 5097: 1, 5094: 1, 5092: 1, 5090: 1, 5089: 1, 5077: 1, 5058: 1, 5053: 1, 5035: 1, 5031: 1, 5030: 1, 5025:
1, 5017: 1, 5013: 1, 5010: 1, 5008: 1, 5006: 1, 4996: 1, 4986: 1, 4973: 1, 4971: 1, 4945: 1, 4943: 1, 4941: 1,
4935: 1, 4933: 1, 4928: 1, 4920: 1, 4899: 1, 4887: 1, 4881: 1, 4874: 1, 4861: 1, 4858: 1, 4850: 1, 4847: 1, 484
3: 1, 4820: 1, 4808: 1, 4791: 1, 4780: 1, 4778: 1, 4773: 1, 4772: 1, 4771: 1, 4767: 1, 4747: 1, 4735: 1, 4734:
1, 4720: 1, 4719: 1, 4705: 1, 4687: 1, 4682: 1, 4681: 1, 4675: 1, 4660: 1, 4652: 1, 4646: 1, 4621: 1, 4574: 1,
4567: 1, 4538: 1, 4529: 1, 4510: 1, 4509: 1, 4508: 1, 4504: 1, 4497: 1, 4496: 1, 4486: 1, 4473: 1, 4464: 1, 446
0: 1, 4444: 1, 4434: 1, 4430: 1, 4421: 1, 4419: 1, 4412: 1, 4407: 1, 4395: 1, 4386: 1, 4385: 1, 4380: 1, 4377:
1, 4358: 1, 4344: 1, 4343: 1, 4341: 1, 4306: 1, 4298: 1, 4285: 1, 4283: 1, 4281: 1, 4278: 1, 4274: 1, 4264: 1,
4252: 1, 4249: 1, 4236: 1, 4232: 1, 4200: 1, 4193: 1, 4174: 1, 4161: 1, 4145: 1, 4144: 1, 4143: 1, 4134: 1, 412
5: 1, 4123: 1, 4116: 1, 4112: 1, 4110: 1, 4103: 1, 4102: 1, 4094: 1, 4089: 1, 4087: 1, 4073: 1, 4067: 1, 4065:
1, 4061: 1, 4048: 1, 4047: 1, 4042: 1, 4027: 1, 4003: 1, 4000: 1, 3999: 1, 3989: 1, 3979: 1, 3975: 1, 3963: 1,
3961: 1, 3960: 1, 3947: 1, 3946: 1, 3942: 1, 3941: 1, 3940: 1, 3939: 1, 3932: 1, 3929: 1, 3928: 1, 3916: 1, 391
4: 1, 3910: 1, 3907: 1, 3906: 1, 3893: 1, 3890: 1, 3888: 1, 3880: 1, 3875: 1, 3873: 1, 3855: 1, 3850: 1, 3849:
1, 3845: 1, 3832: 1, 3820: 1, 3818: 1, 3815: 1, 3801: 1, 3799: 1, 3796: 1, 3786: 1, 3778: 1, 3766: 1, 3765: 1,
3762: 1, 3744: 1, 3743: 1, 3738: 1, 3733: 1, 3732: 1, 3725: 1, 3724: 1, 3721: 1, 3712: 1, 3708: 1, 3701: 1, 369
7: 1, 3694: 1, 3693: 1, 3690: 1, 3684: 1, 3675: 1, 3674: 1, 3669: 1, 3668: 1, 3667: 1, 3666: 1, 3659: 1, 3656:
1, 3645: 1, 3635: 1, 3629: 1, 3622: 1, 3618: 1, 3613: 1, 3605: 1, 3603: 1, 3597: 1, 3594: 1, 3588: 1, 3583: 1,
3581: 1, 3570: 1, 3569: 1, 3568: 1, 3564: 1, 3559: 1, 3558: 1, 3553: 1, 3546: 1, 3544: 1, 3535: 1, 3534: 1, 353
3: 1, 3527: 1, 3522: 1, 3513: 1, 3512: 1, 3510: 1, 3507: 1, 3501: 1, 3493: 1, 3490: 1, 3484: 1, 3479: 1, 3477:
1, 3471: 1, 3469: 1, 3458: 1, 3449: 1, 3448: 1, 3445: 1, 3442: 1, 3439: 1, 3436: 1, 3433: 1, 3432: 1, 3427: 1,
3423: 1, 3414: 1, 3408: 1, 3404: 1, 3402: 1, 3397: 1, 3391: 1, 3374: 1, 3373: 1, 3371: 1, 3361: 1, 3348: 1, 334
4: 1, 3341: 1, 3339: 1, 3336: 1, 3332: 1, 3324: 1, 3323: 1, 3318: 1, 3317: 1, 3305: 1, 3288: 1, 3283: 1, 3276:
1, 3275: 1, 3269: 1, 3262: 1, 3251: 1, 3237: 1, 3232: 1, 3229: 1, 3228: 1, 3226: 1, 3222: 1, 3214: 1, 3212: 1,
3210: 1, 3209: 1, 3208: 1, 3207: 1, 3194: 1, 3189: 1, 3185: 1, 3177: 1, 3162: 1, 3156: 1, 3152: 1, 3148: 1, 314
3: 1, 3137: 1, 3134: 1, 3133: 1, 3132: 1, 3131: 1, 3122: 1, 3118: 1, 3100: 1, 3099: 1, 3082: 1, 3079: 1, 3078:
1, 3073: 1, 3067: 1, 3064: 1, 3057: 1, 3056: 1, 3053: 1, 3052: 1, 3046: 1, 3045: 1, 3039: 1, 3037: 1, 3032: 1,
3030: 1, 3020: 1, 3019: 1, 3016: 1, 3014: 1, 3013: 1, 3003: 1, 3001: 1, 2992: 1, 2988: 1, 2987: 1, 2986: 1, 298
1: 1, 2970: 1, 2968: 1, 2958: 1, 2955: 1, 2949: 1, 2948: 1, 2941: 1, 2940: 1, 2935: 1, 2927: 1, 2924: 1, 2923:
1, 2916: 1, 2904: 1, 2896: 1, 2887: 1, 2886: 1, 2865: 1, 2854: 1, 2841: 1, 2839: 1, 2836: 1, 2835: 1, 2829: 1,
2824: 1, 2823: 1, 2810: 1, 2791: 1, 2790: 1, 2782: 1, 2776: 1, 2775: 1, 2763: 1, 2761: 1, 2755: 1, 2748: 1, 274
7: 1, 2743: 1, 2741: 1, 2740: 1, 2734: 1, 2725: 1, 2721: 1, 2707: 1, 2706: 1, 2672: 1, 2670: 1, 2667: 1, 2666:
1, 2659: 1, 2656: 1, 2655: 1, 2652: 1, 2647: 1, 2646: 1, 2644: 1, 2643: 1, 2635: 1, 2634: 1, 2633: 1, 2626: 1,
2625: 1, 2620: 1, 2619: 1, 2616: 1, 2607: 1, 2606: 1, 2605: 1, 2598: 1, 2596: 1, 2595: 1, 2594: 1, 2592: 1, 258
2: 1, 2580: 1, 2572: 1, 2569: 1, 2568: 1, 2565: 1, 2564: 1, 2562: 1, 2561: 1, 2558: 1, 2554: 1, 2552: 1, 2550:
1, 2546: 1, 2540: 1, 2535: 1, 2534: 1, 2533: 1, 2532: 1, 2530: 1, 2523: 1, 2522: 1, 2516: 1, 2513: 1, 2510: 1,
2503: 1, 2502: 1, 2499: 1, 2489: 1, 2487: 1, 2484: 1, 2482: 1, 2478: 1, 2477: 1, 2472: 1, 2471: 1, 2467: 1, 245
4: 1, 2452: 1, 2450: 1, 2444: 1, 2436: 1, 2434: 1, 2430: 1, 2429: 1, 2428: 1, 2427: 1, 2422: 1, 2416: 1, 2414:
1, 2411: 1, 2400: 1, 2398: 1, 2395: 1, 2392: 1, 2389: 1, 2384: 1, 2382: 1, 2379: 1, 2374: 1, 2373: 1, 2371: 1,
2370: 1, 2367: 1, 2362: 1, 2356: 1, 2355: 1, 2348: 1, 2333: 1, 2332: 1, 2328: 1, 2315: 1, 2313: 1, 2299: 1, 229
6: 1, 2293: 1, 2290: 1, 2289: 1, 2288: 1, 2283: 1, 2281: 1, 2276: 1, 2274: 1, 2253: 1, 2249: 1, 2247: 1, 2244:
1, 2238: 1, 2233: 1, 2230: 1, 2229: 1, 2221: 1, 2219: 1, 2218: 1, 2216: 1, 2207: 1, 2206: 1, 2203: 1, 2193: 1,
2192: 1, 2184: 1, 2180: 1, 2178: 1, 2177: 1, 2173: 1, 2169: 1, 2165: 1, 2161: 1, 2160: 1, 2155: 1, 2154: 1, 215
3: 1, 2151: 1, 2150: 1, 2143: 1, 2142: 1, 2139: 1, 2135: 1, 2134: 1, 2133: 1, 2131: 1, 2130: 1, 2126: 1, 2124:
1, 2121: 1, 2118: 1, 2114: 1, 2107: 1, 2106: 1, 2105: 1, 2104: 1, 2102: 1, 2097: 1, 2085: 1, 2084: 1, 2083: 1,
2080: 1, 2074: 1, 2072: 1, 2071: 1, 2069: 1, 2068: 1, 2064: 1, 2059: 1, 2057: 1, 2055: 1, 2052: 1, 2051: 1, 204
7: 1, 2042: 1, 2040: 1, 2035: 1, 2028: 1, 2023: 1, 2022: 1, 2020: 1, 2017: 1, 2015: 1, 2010: 1, 2006: 1, 2003:
1, 2000: 1, 1989: 1, 1985: 1, 1983: 1, 1976: 1, 1974: 1, 1972: 1, 1971: 1, 1963: 1, 1962: 1, 1960: 1, 1958: 1,
1955: 1, 1954: 1, 1952: 1, 1947: 1, 1946: 1, 1943: 1, 1942: 1, 1940: 1, 1934: 1, 1932: 1, 1929: 1, 1927: 1, 192
6: 1, 1918: 1, 1914: 1, 1910: 1, 1904: 1, 1901: 1, 1898: 1, 1897: 1, 1891: 1, 1886: 1, 1882: 1, 1874: 1, 1872:
1, 1866: 1, 1864: 1, 1862: 1, 1861: 1, 1857: 1, 1853: 1, 1851: 1, 1849: 1, 1846: 1, 1845: 1, 1840: 1, 1837: 1,
1835: 1, 1834: 1, 1831: 1, 1830: 1, 1827: 1, 1818: 1, 1816: 1, 1812: 1, 1798: 1, 1794: 1, 1789: 1, 1787: 1, 178
4: 1, 1782: 1, 1775: 1, 1774: 1, 1771: 1, 1769: 1, 1768: 1, 1760: 1, 1759: 1, 1758: 1, 1757: 1, 1751: 1, 1750:
1, 1747: 1, 1746: 1, 1742: 1, 1739: 1, 1734: 1, 1733: 1, 1732: 1, 1731: 1, 1730: 1, 1727: 1, 1722: 1, 1721: 1,
1719: 1, 1718: 1, 1713: 1, 1712: 1, 1711: 1, 1709: 1, 1708: 1, 1703: 1, 1701: 1, 1699: 1, 1698: 1, 1697: 1, 169
4: 1, 1693: 1, 1692: 1, 1688: 1, 1685: 1, 1683: 1, 1678: 1, 1676: 1, 1675: 1, 1673: 1, 1672: 1, 1670: 1, 1667:
1, 1663: 1, 1658: 1, 1656: 1, 1654: 1, 1651: 1, 1650: 1, 1645: 1, 1644: 1, 1640: 1, 1637: 1, 1632: 1, 1629: 1,
1628: 1, 1626: 1, 1625: 1, 1624: 1, 1617: 1, 1611: 1, 1610: 1, 1609: 1, 1595: 1, 1594: 1, 1589: 1, 1582: 1, 158
0: 1, 1579: 1, 1578: 1, 1577: 1, 1576: 1, 1575: 1, 1574: 1, 1570: 1, 1569: 1, 1565: 1, 1561: 1, 1560: 1, 1558:
1, 1557: 1, 1556: 1, 1551: 1, 1550: 1, 1549: 1, 1547: 1, 1545: 1, 1541: 1, 1538: 1, 1535: 1, 1534: 1, 1524: 1,
1522: 1, 1520: 1, 1516: 1, 1506: 1, 1503: 1, 1499: 1, 1498: 1, 1497: 1, 1496: 1, 1494: 1, 1493: 1, 1492: 1, 148
9: 1, 1488: 1, 1483: 1, 1478: 1, 1469: 1, 1468: 1, 1467: 1, 1465: 1, 1463: 1, 1462: 1, 1460: 1, 1456: 1, 1450:
1, 1449: 1, 1443: 1, 1437: 1, 1427: 1, 1424: 1, 1423: 1, 1421: 1, 1419: 1, 1414: 1, 1413: 1, 1411: 1, 1410: 1,
1409: 1, 1408: 1, 1404: 1, 1402: 1, 1397: 1, 1393: 1, 1386: 1, 1383: 1, 1380: 1, 1373: 1, 1372: 1, 1371: 1, 136
7: 1, 1364: 1, 1363: 1, 1360: 1, 1357: 1, 1353: 1, 1352: 1, 1351: 1, 1350: 1, 1348: 1, 1347: 1, 1346: 1, 1345:
1, 1335: 1, 1334: 1, 1332: 1, 1331: 1, 1330: 1, 1325: 1, 1322: 1, 1321: 1, 1317: 1, 1316: 1, 1311: 1, 1309: 1,
1306: 1, 1304: 1, 1303: 1, 1302: 1, 1300: 1, 1297: 1, 1295: 1, 1290: 1, 1284: 1, 1283: 1, 1282: 1, 1274: 1, 127
0: 1, 1267: 1, 1266: 1, 1265: 1, 1264: 1, 1260: 1, 1258: 1, 1257: 1, 1255: 1, 1249: 1, 1246: 1, 1245: 1, 1244:
1, 1233: 1, 1231: 1, 1230: 1, 1224: 1, 1223: 1, 1215: 1, 1214: 1, 1211: 1, 1210: 1, 1208: 1, 1207: 1, 1201: 1,
1198: 1, 1197: 1, 1196: 1, 1195: 1, 1194: 1, 1191: 1, 1188: 1, 1187: 1, 1185: 1, 1184: 1, 1182: 1, 1180: 1, 117
7: 1, 1169: 1, 1168: 1, 1167: 1, 1163: 1, 1160: 1, 1153: 1, 1152: 1, 1148: 1, 1146: 1, 1141: 1, 1140: 1, 1139:
1, 1137: 1, 1133: 1, 1132: 1, 1129: 1, 1125: 1, 1124: 1, 1119: 1, 1118: 1, 1117: 1, 1113: 1, 1110: 1, 1108: 1,
1107: 1, 1106: 1, 1103: 1, 1102: 1, 1098: 1, 1097: 1, 1096: 1, 1094: 1, 1092: 1, 1091: 1, 1087: 1, 1085: 1, 108
2: 1, 1080: 1, 1075: 1, 1073: 1, 1071: 1, 1070: 1, 1068: 1, 1067: 1, 1064: 1, 1061: 1, 1059: 1, 1056: 1, 1053:
1, 1051: 1, 1046: 1, 1044: 1, 1040: 1, 1025: 1, 1024: 1, 1023: 1, 1022: 1, 1017: 1, 1012: 1, 1004: 1, 1003: 1,
1001: 1, 998: 1, 997: 1, 994: 1, 990: 1, 989: 1, 986: 1, 983: 1, 979: 1, 977: 1, 976: 1, 974: 1, 973: 1, 971: 1
, 969: 1, 962: 1, 961: 1, 955: 1, 954: 1, 949: 1, 945: 1, 937: 1, 934: 1, 931: 1, 930: 1, 929: 1, 927: 1, 926:
1, 922: 1, 921: 1, 916: 1, 915: 1, 914: 1, 913: 1, 908: 1, 906: 1, 903: 1, 900: 1, 899: 1, 897: 1, 889: 1, 886:
1, 883: 1, 879: 1, 871: 1, 867: 1, 862: 1, 859: 1, 856: 1, 842: 1, 841: 1, 837: 1, 836: 1, 811: 1, 806: 1, 795:
1, 790: 1, 788: 1, 783: 1, 782: 1, 777: 1, 771: 1, 766: 1, 762: 1, 748: 1, 746: 1, 744: 1, 742: 1, 734: 1, 725:
1, 721: 1, 702: 1, 692: 1, 677: 1, 674: 1, 662: 1, 661: 1, 628: 1, 618: 1, 602: 1, 595: 1, 592: 1, 584: 1, 582:
1, 577: 1, 552: 1, 527: 1, 517: 1, 441: 1})
```

In [ ]:

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

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

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

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


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

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

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, l
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, pred
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, lab
```

```
For values of alpha =  1e-05 The log loss is: 1.3045830009353085
For values of alpha =  0.0001 The log loss is: 1.1783669700355435
For values of alpha =  0.001 The log loss is: 1.1471860490937813
For values of alpha =  0.01 The log loss is: 1.229025518395817
For values of alpha =  0.1 The log loss is: 1.4438999859721926
For values of alpha =  1 The log loss is: 1.668752549862227
```



```
For values of best alpha =  0.001 The train log loss is: 0.6658233660396363
For values of best alpha =  0.001 The cross validation log loss is: 1.1471860490937813
For values of best alpha =  0.001 The test log loss is: 1.1653990866176018
```

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

Ans. Yes, it seems like!

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

```
95.634 % of word of test data appeared in train data
98.474 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

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

#Misc. functionns for ML models


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

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

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

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

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

Stacking the three types of features

In [ ]:

```python
# merging gene, variance and text features

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

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_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_feature_onehotCoding)).tocsr()
test_y = np.array(list(test_df['Class']))

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

In [ ]:

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

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 55287)
(number of data points * number of features) in test data =  (665, 55287)
(number of data points * number of features) in cross validation data = (532, 55287)
```

In [ ]:

```python
train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_variation_feature_responseCo
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variation_feature_responseCodin
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_feature_responseCoding))

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

In [ ]:

```python
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 = ", test_x_responseCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_responseCoding.shape)
```

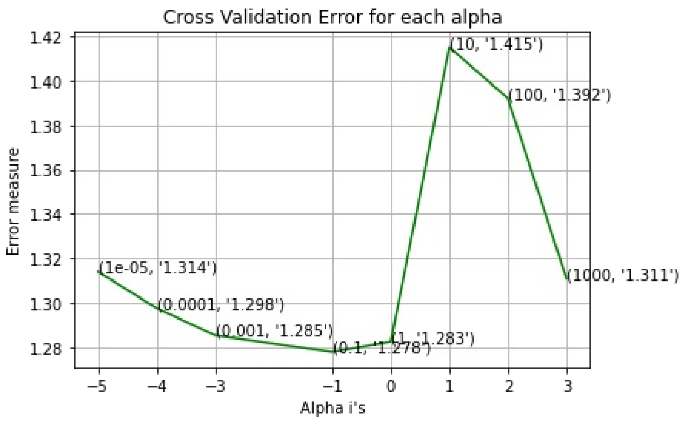
```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 27)
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data = (532, 27)
```

4.1. Base Line Model

4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

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

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


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


alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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


predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, l
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, pred
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, lab
for alpha = 1e-05
Log Loss : 1.3140614997769275
for alpha = 0.0001
Log Loss : 1.2976659520368992
for alpha = 0.001
Log Loss : 1.28546563072638
for alpha = 0.1
Log Loss : 1.2780039500859768
for alpha = 1
Log Loss : 1.2826357503908201
for alpha = 10
Log Loss : 1.4149663487125177
for alpha = 100
Log Loss : 1.3919799612343695
for alpha = 1000
Log Loss : 1.3109442581490653
```



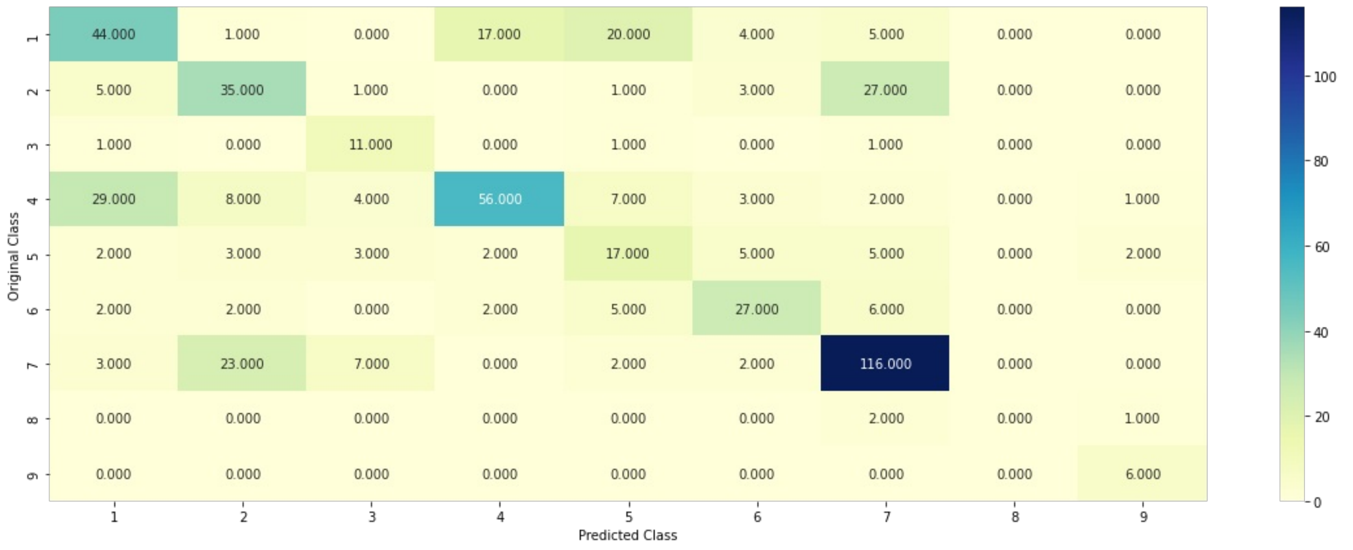
```
For values of best alpha =  0.1 The train log loss is: 0.8809531037897241
For values of best alpha =  0.1 The cross validation log loss is: 1.2780039500859768
For values of best alpha =  0.1 The test log loss is: 1.27941274096128
```

4.1.1.2. Testing the model with best hyper paramters

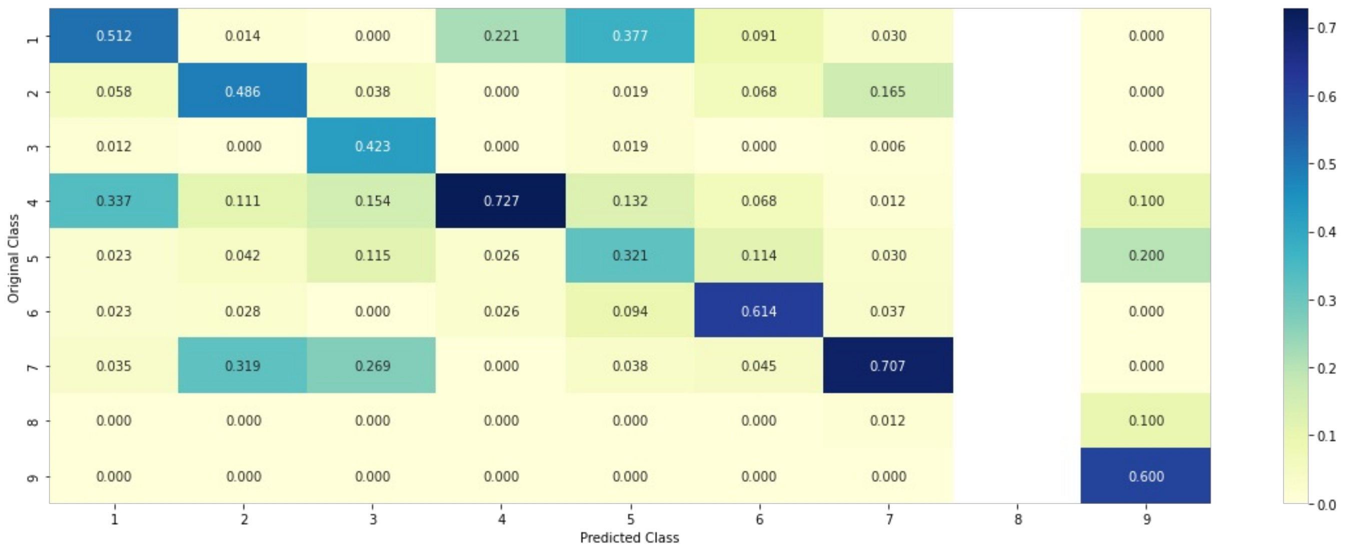
In [ ]:

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

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


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

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

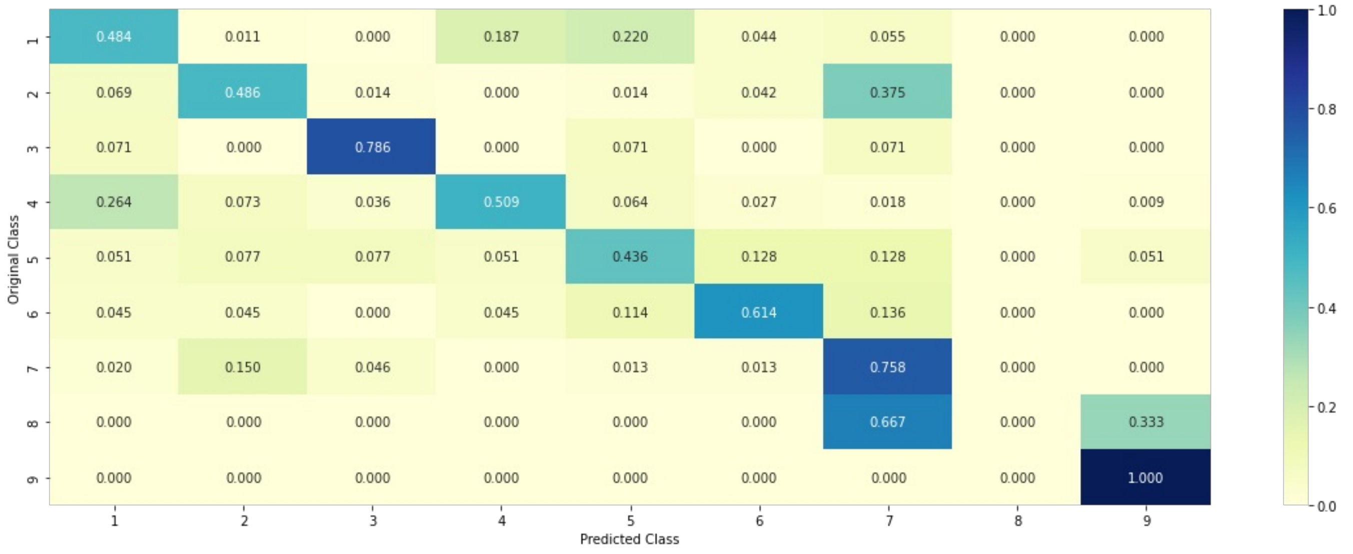
```
Log Loss : 1.2780039500859768
Number of missclassified point : 0.41353383458646614
-------------------- Confusion matrix --------------------
```



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



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



4.1.1.3. Feature Importance, Correctly classified point

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0654 0.0697 0.0115 0.652  0.0327 0.031  0.1304 0.0039 0.0034]]
Actual Class : 7
--------------------------------------------------
10 Text feature [function] present in test data point [True]
12 Text feature [protein] present in test data point [True]
15 Text feature [activity] present in test data point [True]
16 Text feature [acid] present in test data point [True]
17 Text feature [functional] present in test data point [True]
18 Text feature [abrogate] present in test data point [True]
19 Text feature [proteins] present in test data point [True]
21 Text feature [pten] present in test data point [True]
22 Text feature [amino] present in test data point [True]
26 Text feature [retained] present in test data point [True]
28 Text feature [deviation] present in test data point [True]
29 Text feature [ability] present in test data point [True]
30 Text feature [phosphatase] present in test data point [True]
31 Text feature [whereas] present in test data point [True]
32 Text feature [results] present in test data point [True]
35 Text feature [related] present in test data point [True]
37 Text feature [suppressor] present in test data point [True]
38 Text feature [determined] present in test data point [True]
40 Text feature [catalytic] present in test data point [True]
42 Text feature [tensin] present in test data point [True]
44 Text feature [correspond] present in test data point [True]
47 Text feature [critical] present in test data point [True]
48 Text feature [type] present in test data point [True]
51 Text feature [transfection] present in test data point [True]
52 Text feature [affect] present in test data point [True]
56 Text feature [indicate] present in test data point [True]
59 Text feature [indicated] present in test data point [True]
62 Text feature [generated] present in test data point [True]
63 Text feature [indicates] present in test data point [True]
64 Text feature [wild] present in test data point [True]
66 Text feature [purified] present in test data point [True]
67 Text feature [determine] present in test data point [True]
68 Text feature [scanning] present in test data point [True]
69 Text feature [functions] present in test data point [True]
70 Text feature [loss] present in test data point [True]
71 Text feature [transfected] present in test data point [True]
76 Text feature [therefore] present in test data point [True]
77 Text feature [important] present in test data point [True]
79 Text feature [although] present in test data point [True]
80 Text feature [associated] present in test data point [True]
81 Text feature [see] present in test data point [True]
84 Text feature [putative] present in test data point [True]
89 Text feature [either] present in test data point [True]
92 Text feature [average] present in test data point [True]
93 Text feature [terminal] present in test data point [True]
94 Text feature [expressed] present in test data point [True]
Out of the top  100  features  46 are present in query point
```

In [ ]:

```
test_df['TEXT'].iloc[test_point_index]
```

Out[ ]:

```
'tumor suppressor gene phosphatase tensin homolog deleted chromosome 10 pten encodes 403 amino acid protein lip
id protein phosphatase activity li et al 1997 li sun 1997 myers et al 1997 steck et al 1997 antagonizes phospho
inositide 3 kinase pi3k akt signaling pathway dephosphorylating membrane localized phosphatidylinositol 3 4 5 t
risphosphate pip3 maehama dixon 1998 1999 functional inactivation pten results constitutive upregulation pi3k a
kt signaling promotes cell cycle progression cellular survival enhanced protein synthesis cell migration tumor
induced angiogenesis li sun 1998 sun et al 1999 vogt 2001 wen et al 2001 su et al 2003 leslie et al 2005 pten c
ontains n terminal phosphatidylinositol 4 5 bisphosphate pip2 binding motif followed phosphatase domain ptp ca2
independent c2 domain c terminal tail leslie downes 2004 pten primarily cytoplasmic binding pten plasma membran
e mediated combination cationic patches located n terminal pip2 binding motif r11 k13 r14 r15 phosphatase domai
n r161 k163 k164 along contributions c2 domain das et al 2003 leslie downes 2004 pten also detected nucleus det
erminants nuclear cytoplasmic shuttling well understood lian di cristofano 2005 pten lacks classical nuclear lo
calization signal nls however recent study identified nls like sequences pten mediate nuclear entry major vault
protein mvp chung et al 2005 recent studies suggest nuclear pten mediates growth suppression independent downre
gulating akt whereas cytoplasmic pten mediates apoptosis chung eng 2005 liu et al 2005 studies document importa
nce cellular localization regulation pten function examined highly conserved region pten situated near n termin
al pip2 binding motif amino acids 19 25 dgfdldl germ line somatic mutations within region indicate plays critic
al role tumor suppressor function pten bonneau longy 2000 generated amino acid substitution mutant f21a found l
ocalized predominantly cell nucleus examination naturally occurring mutants within region indicates region requ
ired cytoplasmic localization hence refer region cytoplasmic localization signal cls mutations within cls lead
nuclear localization also result loss pten growth suppressing activity preserving lipid phosphatase activity da
ta define elements within pten sequence determine cellular localization suggest cytoplasmic plasma membrane loc
alization essential growth regulatory activities pten top page results f21a mutation abolishes growth suppressi
ve properties pten attempt determine biological significance conserved amino acids 19 25 pten point mutant f21a
generated figure 1 established u87mg cell lines stably expressing either empty vector pcdna3 wild type pten pte
n wt f21a pten mutant pten f21a catalytically inactive pten pten c124s u87mg glioblastoma cells carry frame shi
ft deletion pten gene hence pten null protein level making cell line suitable study pten function li et al 1997
stable expression wild type pten u87mg cells leads dramatic reduction growth cells whereas expression catalytic
ally inactive pten c124s affect growth u87mg cells furnari et al 1997 li sun 1998 examined growth rate various
u87mg cell lines found growth rates pten wt pten c124s expressing cells consistent previous findings figure 2b
li sun 1998 expression pten wt suppresses growth u87mg cells comparison cells transfected empty vector whereas
```

expression pten c124s induce growth suppression growth rates two different pten f21a cell lines similar pten c1
24s cells suggesting f21a mutation abolishes growth suppressive properties pten figure 2b figure 1 figure 1 unf
ortunately unable provide accessible alternative text require assistance access image please contact help natur
e com author multiple sequence alignment exon 1 amino acids 1 26 pten protein diverse species lipid pip2 bindin
g motif amino acids 6 15 underlined corresponding consensus sequence shown parenthesis residues corresponding c
ls amino acids 19 25 underlined amino acids required cytoplasmic localization shown green germline somatic muta
tions within region indicated arrows full figure legend 70k figure 2 figure 2 unfortunately unable provide acce
ssible alternative text require assistance access image please contact help nature com author morphology growth
rate u87mg pten cell lines morphology stably transfected u87mg pten cell lines photographs taken times 40 magni
fication b growth rates stably transfected u87mg pten cell lines data shown correspond average cell number stan
dard deviation full figure legend 113k parental u87mg cells expressing pten c124s grow aggregates elongated app
earance morphological patterns associated oncogenic transformation li sun 1998 stable expression wild type pten
u87mg induces flattened cell shape non refractile appearance li sun 1998 obtained similar results empty vector
pten c124s expressing u87mg cells elongated refractile whereas pten wt expressing cells flat adherent pten f21a
cells however flattened cell shape elongated refractile heavily aggregated pten c124s cells figure 2a f21a muta
tion abolishes ability pten downregulate pi3k akt signaling expression wild type pten u87mg cells antagonizes p
i3k akt signaling li sun 1998 ramaswamy et al 1999 sun et al 1999 effect dependent upon pip3 phosphatase activi
ty pten myers et al 1998 ramaswamy et al 1999 loss function leads upregulation pi3k akt signaling hence promote
s enhanced proliferation survival increased protein synthesis determined effect expressing f21a mutant pi3k akt
signaling figure 3a similar pten c124s expressing u87mg cells proliferating pten f21a cells unable downregulate
levels activated ser473 phosphorylated akt whereas activated akt undetectable pten wt cells pten f21a cells als
o unable downregulate pdgf stimulated activation akt figure 3b surprisingly expression f21a mutant affect downs
tream target rapamycin tor signaling whereas expression wild type pten downregulates levels phosphorylated p70
s6k phosphorylated 4e bp1 f21a c124s mutants effect p70 s6k 4e bp1 phosphorylation figure 3c consistent growth
rate cells p27 induced expression wild type pten f21a c124s pten mutants figure 3c expected treatment cells ly2
94002 pi3k inhibitor also increases p27 levels downregulates phosphorylation p70 s6k 4e bp1 furthermore treatme
nt cells rapamycin mtor specific inhibitor also downregulates p70 s6k 4e bp1 phosphorylation inability pten f21
a mutant abrogate pi3k akt signaling attenuate cell growth suggests mutation interferes replication suppressing
properties pten u87mg cells figure 3 figure 3 unfortunately unable provide accessible alternative text require
assistance access image please contact help nature com author pten f21a mutant fails downregulate pi3k akt sign
aling akt phosphorylation status stably transfected u87mg cell lines b basal level activated akt downregulated
wild type pten f21a mutant c f21a mutation abolishes ability pten downregulate pi3k akt signaling full figure l
egend 114k f21a mutant retains wild type ptdins 3 4 5 p3 phosphatase activity vitro biological activity f21a mu
tant u87mg cells resembles phosphatase inactive c124s tested catalytic activity pten f21a his6 tagged pten wt p
ten f21a pten c124s proteins expressed bacteria purified ni2 nta agarose slurry catalytic activity assayed wate
r soluble ptdins 3 4 5 p3 f21a mutant protein phosphatase activity similar wild type pten protein indicating f2
1a mutation affect lipid phosphatase activity figure 4a inability pten f21a mutant protein attenuate growth dow
nregulate pi3k akt signaling must therefore reflect impairment function distinct lipid phosphatase activity fig
ure 4 figure 4 unfortunately unable provide accessible alternative text require assistance access image please
contact help nature com author pten f21a retains wild type lipid phosphatase activity localizes nucleus plasma
membrane targeting restores function phosphatase activity his6 tagged pten fusion proteins assayed water solubl
e ptdins 3 4 5 p3 using malachite green phosphate assay data shown correspond average amount phosphate released
standard deviation b localization egfp tagged pten proteins u87mg cells images shown taken times 40 magnificati
on c localization egfp tagged pten proteins hek 293 cells images shown taken times 40 magnification effect myri
stylated pten egfp myr wt myr f21a myr c124s akt phosphorylation status e localization egfp tagged myristylated
pten myr wt myr f21a transiently transfected hek 293 cells full figure legend 294k f21a mutation localizes pten
nucleus n terminal pip2 binding motif pten required efficient localization plasma membrane das et al 2003 walke
r et al 2004 pten k13e somatic mutation identified glioblastoma located within n terminal pip2 binding motif ca
uses inefficient membrane targeting vitro whole cells duerr et al 1998 walker et al 2004 pten k13e retains wild
type lipid phosphatase activity vitro fails inhibit cell proliferation prevent akt phosphorylation expressed u8
7mg cells membrane targeting pten k13e insertion myristylation signal restores pten function reconstituted u87m
g cells biological consequences k13e mutation similar caused f21a mutation retains wild type lipid phosphatase
activity vitro impaired ability attenuate cell growth f21a mutation near n terminal pip2 binding motif figure 1
appeared likely mutation also interferes targeting pten lipid membranes order test effects f21a mutation pten l
ocalization generated c terminal enhanced green fluorescence protein egfp tagged pten wt f21a c124s u87mg hek 2
93 cells transiently transfected examined 48 h post transfection fluorescence microscopy figure 4b c confocal m
icroscopy data shown pten wt pten c124s show diffuse cytoplasmic nuclear localization consistent previous obser
vations figure 4b c furnari et al 1997 li sun 1997 das et al 2003 surprisingly pten f21a mutant localizes predo
minantly nucleus u87mg hek 293 cells figure 4b c nuclear localization pten f21a confirmed 4 6 diamidino 2 pheny
lindole dapi staining shown overlay egfp dapi images group cells figure 4b c bottom panel order test possible d
ominance f21a mutation examined double mutant pten f21a c124s single mutant pten c124s localizes primarily cyto
plasm pten f21a c124s double mutant localizes nucleus indicating aberrant nuclear localization induced f21a mut
ation dominant independent pten phosphatase activity figure 4c also examined localization egfp tagged pten k13e
previously examined mutant localizes primarily cytoplasm cells mutant excluded nucleus figure 5a data support c
onclusion mechanisms loss growth suppressive functions k13e versus f21a mutant distinct results suggest pten k1
3e impairs membrane binding recruitment whereas pten f21a retained nucleus two distinct mechanisms loss tumor s
uppressor activity affect lipid phosphatase activity figure 5 figure 5 unfortunately unable provide accessible
alternative text require assistance access image please contact help nature com author nuclear localization wil
d type lipid phosphatase activity mutation cls mutations within cls lead pten nuclear localization hek 293 cell
s images shown taken times 40 magnification b mutations within csl retain wild type lipid phosphatase activity
data shown correspond average amount phosphate released standard deviation full figure legend 105k targeting f2
1a mutant plasma membrane restores ability downregulate akt nuclear localization prevents pten f21a regulating
pi3k akt signaling targeting mutant protein plasma membrane restore ability downregulate akt localization depen
dent restoration function would also consistent finding f21a mutant retains lipid phosphatase activity vitro f2
1a mutant targeted plasma membrane n terminal myristylation signal myristylated f21a protein effective myristyl
ated pten wt reducing levels phosphorylated akt figure 4d result also suggests f21a function lipid phosphatase
cellular environment fluorescence microscopy transiently transfected cells confirmed localization myristylated
f21a protein plasma membrane figure 4e however cell lines stably express myristylated pten wt myristylated f21a
protein could established presumably strong growth suppressive effect pten constitutively localized plasma memb
rane germline somatic mutations surrounding f21a n terminal pip2 binding motif lead nuclear localization pten r
etaining wild type lipid phosphatase activity f21a mutant encountered spontaneous naturally occurring mutation
pten properties may unique without relevance human disease therefore examined disease related germline somatic
mutations surrounding f21 site compared mutations within n terminal pip2 binding motif table 1 determined cellu
lar localization vitro lipid phosphatase activity mutants within highly conserved amino acids 20 25 gfdldl incl
ude germline mutation d24y associated bannayan riley ruvalcaba syndrome brrs somatic cancer associated mutation
s g20e l23f d24n d24 l25 deletion pten expressed egfp glutathione transferase gst tagged fusion proteins pten g
20e pten l23f pten d24n data shown pten d24y localize primarily nucleus hek 293 cells figure 5a suggesting regi
on pten plays important role maintenance cytoplasmic localization pten deletion mutant d24 l25 shows partial re
covery cytoplasmic localization data shown hand somatic cancer associated mutants within n terminal pip2 bindin

g motif s10n k13e r15s y16c show cytoplasmic localization transiently expressed hek 293 cells figure 5a data sh
own observations suggest distinct control pten localization juxtaposed regions refer table 1 table 1 n terminal
germline somatic mutants pten table 1 n terminal germline somatic mutants pten unfortunately unable provide acc
essible alternative text require assistance access image please contact help nature com author full table deter
mined lipid phosphatase activity germline somatic mutations region defined residues 20 25 gst tagged pten mutan
t proteins expressed purified glutathione sepharose 4b slurry catalytic activity assayed water soluble pip3 maj
ority n terminal mutants retain lipid phosphatase activity vitro comparable wild type pten figure 5b table 1 re
sults suggest loss pten growth suppressive activity mutants due loss lipid phosphatase activity probably relate
d nuclear localization analysis n terminal mutants supports conclusion amino acids 20 25 human pten important d
eterminants cytoplasmic localization define amino acids required cellular compartmentalization pten alanine mut
ational scanning region performed results compared data germline somatic mutants alanine scanning mutants inclu
ded q17a e18a d19a d22a l25a t26a table 2 q17a e18a mutants exhibit cytoplasmic localization like wild type pro
tein d19a mutant predominantly nuclear data shown table 2 nuclear localization also observed l25a mutant wherea
s t26a mutant cytoplasmic data shown d22a mutant primarily cytoplasmic contrast mutants region d19a g20a f21a l
23f nuclear amino acid 22 may therefore essential cytoplasmic localization d22a mutant like mutants region wild
type lipid phosphatase activity vitro contrast nuclear pten able reduce levels phosphorylated akt transient tra
nsfection data shown data confirm important role amino acids 19 25 determining cellular localization pten amino
acids g20 f21 l23 d24 particular critical cannot mutated without loss cytoplasmic localization table 2 summary
n terminal pten alanine mutants table 2 summary n terminal pten alanine mutants unfortunately unable provide ac
cessible alternative text require assistance access image please contact help nature com author full table soma
tic pten mutants localized nucleus show loss growth suppressive activity u87mg cells order assess growth suppre
ssive functions naturally occurring pten mutants localize nucleus generated u87mg cell lines stably expressing
cancer associated mutants pten g20e l23f d24n mutants exhibit strong nuclear localization refer figure 5a data
shown d24n also retain wild type lipid phosphatase activity examined phosphorylation status akt figure 6a proli
feration rates figure 6b anchorage dependent growth figure 6c mutant expressing cells consistent observations p
ten f21a mutant g20e l23f d24n unable reduce phosphorylated levels akt figure 6a furthermore mutations result l
oss growth suppressive properties pten proliferation rates similar empty vector pten c124s expressing cells fig
ure 6b colony formation soft agar determined measure anchorage independent growth hallmark transformed cells wi
ld type pten greatly reduces ability u87mg cells grow soft agar whereas effect seen pten c124s li sun 1998 used
wild type c124s mutant proteins positive negative controls figure 6c cells expressing pten f21a pten d24n form
agar colonies efficiency cells expressing pten c124s anchorage independent growth cells expressing g20e l23f mu
tants reduced compared vector control significantly colony formation pten wt expressing cells colony sizes sign
ificantly different vector control observations indicate somatic mutations near f21 residue leading nuclear loc
alization also induce loss growth inhibiting activity figure 6 figure 6 unfortunately unable provide accessible
alternative text require assistance access image please contact help nature com author loss growth suppressive
activity mutation cls akt phosphorylation status u87mg cells transfected pten mutants b growth rates u87mg pten
cells transfected pten mutants data shown correspond average cell number standard deviation growth u87mg pten g
20e 3 cells significantly enhanced u87mg pten wt 3 cells p 0 01228 c pten mutants within cls reduce anchorage i
ndependent growth data shown correspond average number colonies standard deviation 2 weeks number colonies form
ed u87mg pten g20e 1 g20e 3 l23f 4 l23f 6 cell lines significantly higher u87mg pten wt 3 cells p values 0 0003
7 0 00092 00034 0 0018 respectively representative image soft agar colonies formed various somatic u87mg pten c
ell lines taken times 4 magnification 2 weeks seeding full figure legend 151k top page discussion cellular comp
artmentalization regulates function many tumor suppressor proteins fabbro henderson 2003 pten localize cytoplas
m nucleus little known role nuclear pten tumor suppression lian di cristofano 2005 generated pten f21a mutation
part investigation highly conserved region consisting residues 19 25 dgfdldl human pten pten f21a mutant repres
entative mutations region defines short stretch amino acids critical determinant pten cellular localization fun
ction pten f21a mutant fails suppress growth u87mg glioblastoma cells lost ability downregulate phosphorylation
akt two points pten f21a similar loss function mutant pten c124s lacks lipid phosphatase activity enzymatic fun
ction mediates cellular growth control tumor suppression however contrast pten c124s pten f21a mutant retains l
ipid phosphatase activity localized nucleus nuclear localization also seen germ line somatic alanine scanning m
utants map region d19a g20e f21a l23f d24n d24y l25a somatic mutations nearby pip2 binding motif k31e r15s surr
ounding sequences s10n y16c alter cytoplasmic localization pten t26a mutation also retains cytoplasmic localiza
tion within region amino acids 19 25 residue 22 mutated without affecting cytoplasmic localization propose refe
r region cls without implying specific mechanism action mechanism determining cellular localization pten fully
understood recent study describes hypothetical nls pten mediate nuclear import mvp chung et al 2005 sequence in
itiates possible export nucleus identified conserved region encompassing amino acids 19 25 pten could considere
d candidate nuclear export signal resembles canonical nuclear export signal contains closely spaced leucines hy
drophobic amino acids dgfdldl however tests leptomycin b suggest sequence functions nuclear export crm1 exporti
n would major export receptor see supplementary information alternatively amino acids 19 25 pten could contain
cytoplasmic retention signal whose function destroyed mutations region mechanism putative cytoplasmic retention
also remains investigated mutations within cls clearly show region plays crucial role nucleocytoplasmic shuttli
ng pten change cellular compartmentalization likely affect function pten mutations cls cause nuclear localizati
on induce similar changes characteristic activities pten changes include inability mutant protein attenuate tra
nsformed morphology growth rates u87mg cells ability wild type pten reduce anchorage independent growth also ab
olished mutations cls interference akt signaling previously identified cancer derived somatic mutations cls sho
w loss tumor suppressor function observations document lack growth suppressive activities correlated nuclear lo
calization protein yet cases lipid phosphatase activity retained mutant proteins directing mutant protein plasm
a membrane help myristylation signal restores akt regulatory activity lost nuclear localization hypothesize nuc
lear pten still retains lipid phosphatase activity unable carry critical functions essential growth suppressive
activity although observed correlation nuclear localization loss growth controlling activity pten null u87mg ce
lls data derived germline somatic mutations suggest connection cellular compartmentalization function pten may
broader validity observations conclusions complete agreement work vazquez co workers pten membrane interactions
vazquez devreotes 2006 vazquez et al 2006 recent studies attempted assign specific functions nuclear pten pten
targeted nucleus addition n terminal nls found inhibit growth pten null u251mg glioblastoma cells inactivated p
70 s6k without downregulating akt liu et al 2005b observations apparent contradiction findings cls mutants u87m
g cells possible discrepancy due differences cell type culture conditions factor potential importance p53 mutat
ed u251mg cells wild type u87mg van meir et al 1994 pten interacts p53 regulates p53 levels differences wild ty
pe mutated p53 interactions studied detail mayo et al 2002 trotman pandolfi 2003 cellular growth conditions ano
ther possible cause divergent observations studies cited u251mg cells serum starved whereas present investigati
on u87mg cells cultivated 10 fetal bovine serum fbs preliminary observations pten tagged n terminal nls suggest
portion protein retained cytoplasm could also explain discrepant observations cell growth different study assig
ns nuclear pten ability downregulate cyclin d1 interfere activating phosphorylation mitogen activated protein k
inase mcf7 breast cancer cells chung eng 2005 however contrast u87mg cells mcf7 cells express wild type pten co
uld conceivably contribute observed effects substantial additional work needed resolve conflicts clarify functi
ons nuclear pten paper revision gil et al 2006 published extensive study nuclear cytoplasmic distribution pten
data suggest sequences contiguous pip2 binding motif also mediate nuclear localization context truncated versio
n pten amino acids 1 375 assign proapoptotic role nuclear pten contradiction previous observations supported pr
oapoptotic role cytoplasmic pten chung eng 2005 gil et al hypothesize pten exists two conformations nls masked
cytoplasmic nls unmasked nuclear states dictated nuclear exclusion motifs within ptp c2 c terminal domains nucl

```
ear localization domain within pip2 binding motif context model cls defines region pten like c terminal nuclear
exclusion motif induces strong nuclear localization mutated suggesting mutations might cause switch nls unmaske
d conformation combined data studies position n terminus pten pip2 binding domain nuclear localization domain a
djacent cls region critical region determines nucleocytoplasmic localization also binds plasma membrane summary
defined residues 19 25 pten cls essential cytoplasmic localization mutations cls preserve lipid phosphatase act
ivity induce nuclear localization concomitant loss growth regulatory functions '
```

In [ ]:
```
no_feature
```

Out[ ]:
```
100
```

In [ ]:
```
test_df['Gene'].iloc[test_point_index]
```

Out[ ]:
```
'PTEN'
```

In [ ]:
```
test_df['Variation'].iloc[test_point_index]
```

Out[ ]:
```
'R15S'
```

In [ ]:
```
clf.coef_.shape
```

Out[ ]:
```
(9, 55287)
```

In [ ]:
```
indices=np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
indices[0]
```

Out[ ]:
```
array([    0, 29483, 29482, 29480, 29475, 29474, 29471, 29470, 29466,
       29463, 29461, 29458, 29456, 29455, 29454, 29453, 29451, 29449,
       29448, 29446, 29444, 29443, 29441, 29436, 29435, 29433, 29432,
       29431, 29484, 29430, 29486, 29499, 29574, 29572, 29560, 29557,
       29554, 29549, 29548, 29544, 29536, 29535, 29533, 29532, 29530,
       29529, 29527, 29526, 29525, 29524, 29518, 29517, 29514, 29513,
       29512, 29510, 29509, 29508, 29504, 29497, 29429, 29428, 29427,
       29333, 29332, 29330, 29329, 29327, 29326, 29312, 29308, 29307,
       29306, 29305, 29301, 29300, 29298, 29296, 29292, 29291, 29290,
       29287, 29284, 29282, 29281, 29280, 29275, 29273, 29267, 29262,
       29334, 29335, 29338, 29341, 29426, 29425, 29424, 29422, 29417,
       29415])
```

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

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

In [ ]:
```
for i in range(10):
  test_point_index = i
  no_feature = 100
  predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
  print("Predicted Class :", predicted_cls[0])
  print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index])
  print("Actual Class :", test_y[test_point_index])
  indices=np.argsort(-1*abs(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
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0654 0.0697 0.0115 0.652  0.0327 0.031  0.1304 0.0039 0.0034]]
Actual Class : 4
--------------------------------------------------
Out of the top  100  features  0 are present in query point
Predicted Class : 4
Predicted Class Probabilities: [[0.0654 0.0697 0.0115 0.652  0.0327 0.031  0.1304 0.0039 0.0034]]
Actual Class : 7
--------------------------------------------------
Out of the top  100  features  0 are present in query point
Predicted Class : 4
Predicted Class Probabilities: [[0.1579 0.0715 0.0119 0.5514 0.0336 0.0321 0.1342 0.004  0.0034]]
Actual Class : 1
--------------------------------------------------
Out of the top  100  features  0 are present in query point
Predicted Class : 2
Predicted Class Probabilities: [[0.2126 0.2816 0.0155 0.1233 0.0443 0.0421 0.2706 0.0054 0.0046]]
Actual Class : 8
--------------------------------------------------
Out of the top  100  features  0 are present in query point
Predicted Class : 7
Predicted Class Probabilities: [[0.0734 0.0784 0.0129 0.1025 0.0367 0.0348 0.653  0.0044 0.0038]]
Actual Class : 2
--------------------------------------------------
Out of the top  100  features  0 are present in query point
Predicted Class : 1
Predicted Class Probabilities: [[0.578  0.0786 0.013  0.1031 0.0369 0.0351 0.1471 0.0044 0.0038]]
Actual Class : 1
--------------------------------------------------
Out of the top  100  features  0 are present in query point
Predicted Class : 1
Predicted Class Probabilities: [[0.578  0.0786 0.013  0.1031 0.0369 0.0351 0.1471 0.0044 0.0038]]
Actual Class : 1
--------------------------------------------------
Out of the top  100  features  0 are present in query point
Predicted Class : 7
Predicted Class Probabilities: [[0.0824 0.3405 0.0146 0.1133 0.0406 0.0386 0.3606 0.0049 0.0044]]
Actual Class : 7
--------------------------------------------------
Out of the top  100  features  0 are present in query point
Predicted Class : 1
Predicted Class Probabilities: [[0.5666 0.0791 0.013  0.1124 0.0371 0.0354 0.1481 0.0045 0.0038]]
Actual Class : 4
--------------------------------------------------
Out of the top  100  features  0 are present in query point
Predicted Class : 6
Predicted Class Probabilities: [[0.0748 0.1118 0.0132 0.1038 0.0376 0.4606 0.1897 0.0045 0.004 ]]
Actual Class : 2
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [ ]:

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0711 0.0755 0.0126 0.0977 0.0351 0.5596 0.1401 0.0043 0.0039]]
Actual Class : 6
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

In [ ]:

In [ ]:

4.2. K Nearest Neighbour Classification

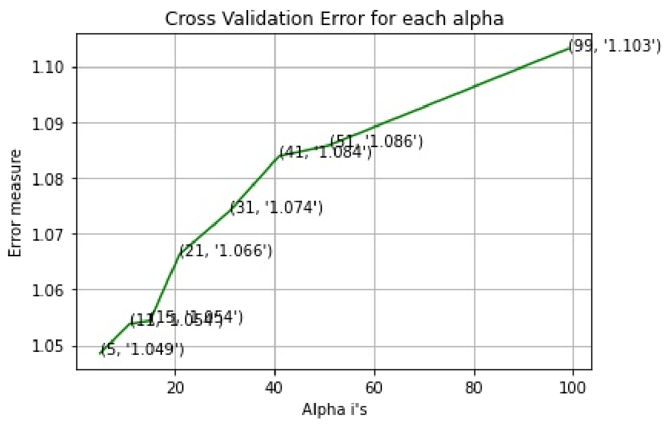
4.2.1. Hyper parameter tuning

In [ ]:

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

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


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


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

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


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

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, l
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, pred
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, lab
```

```
for alpha = 5
Log Loss : 1.0485068580062582
for alpha = 11
Log Loss : 1.0538278702819512
for alpha = 15
Log Loss : 1.0543503022652165
for alpha = 21
Log Loss : 1.0663004139138115
for alpha = 31
Log Loss : 1.0739775156280884
for alpha = 41
Log Loss : 1.083895893756924
for alpha = 51
Log Loss : 1.0858016231189966
for alpha = 99
Log Loss : 1.103126701759781
```



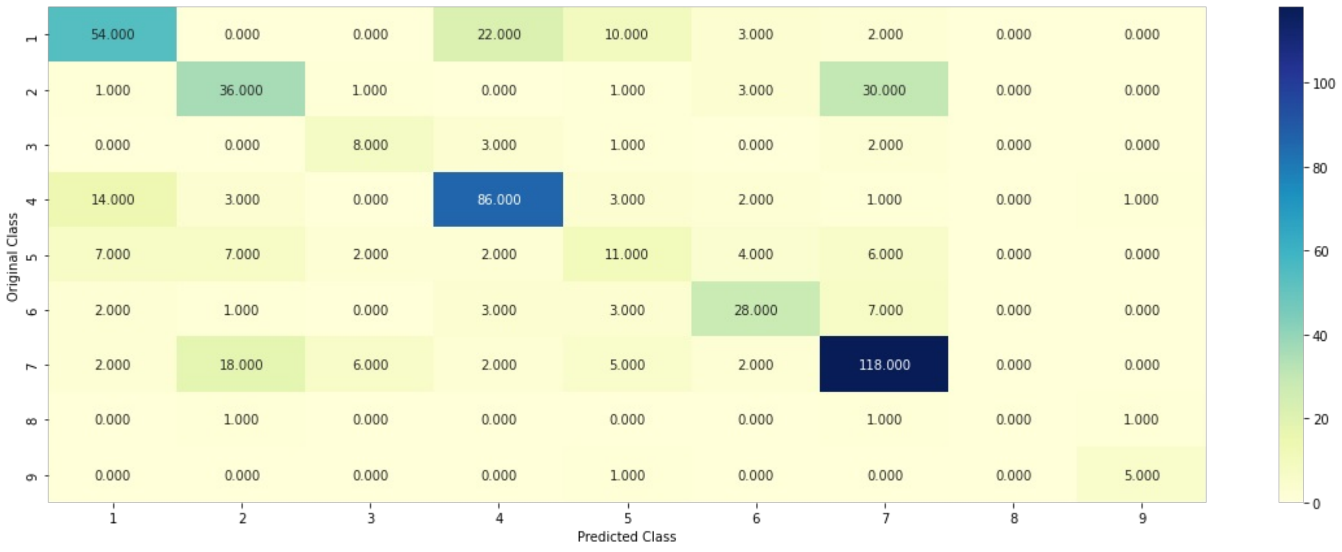
```
For values of best alpha =  5 The train log loss is: 0.450894904504374755
For values of best alpha =  5 The cross validation log loss is: 1.0485068580062582
For values of best alpha =  5 The test log loss is: 1.0695622744171402
```

4.2.2. Testing the model with best hyper paramters

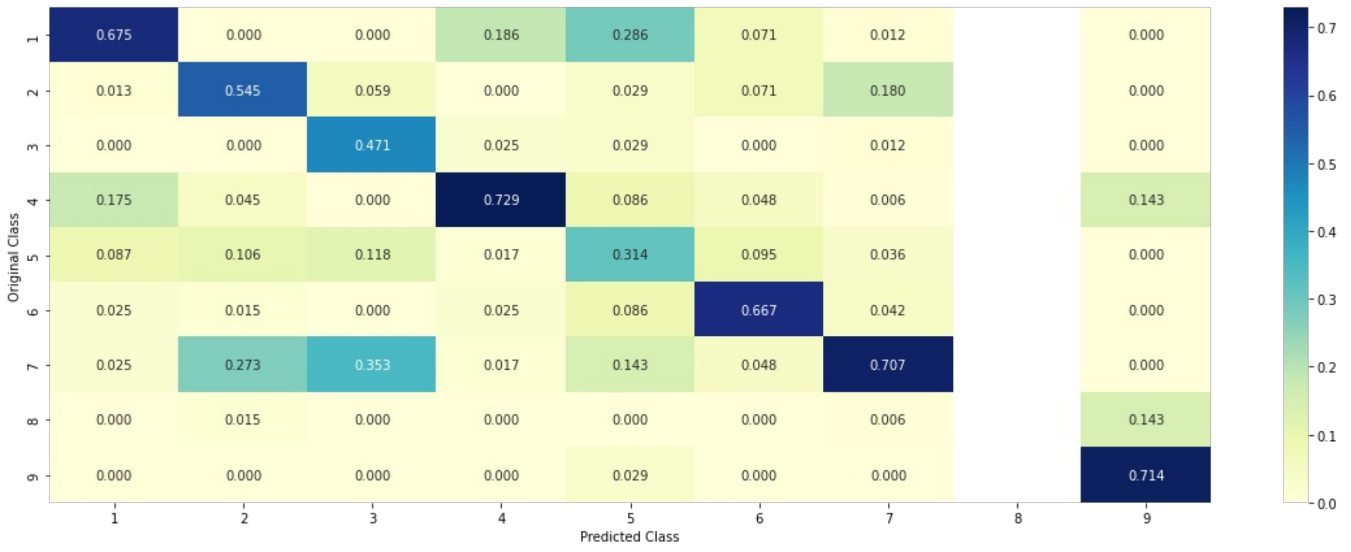
In [ ]:

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

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

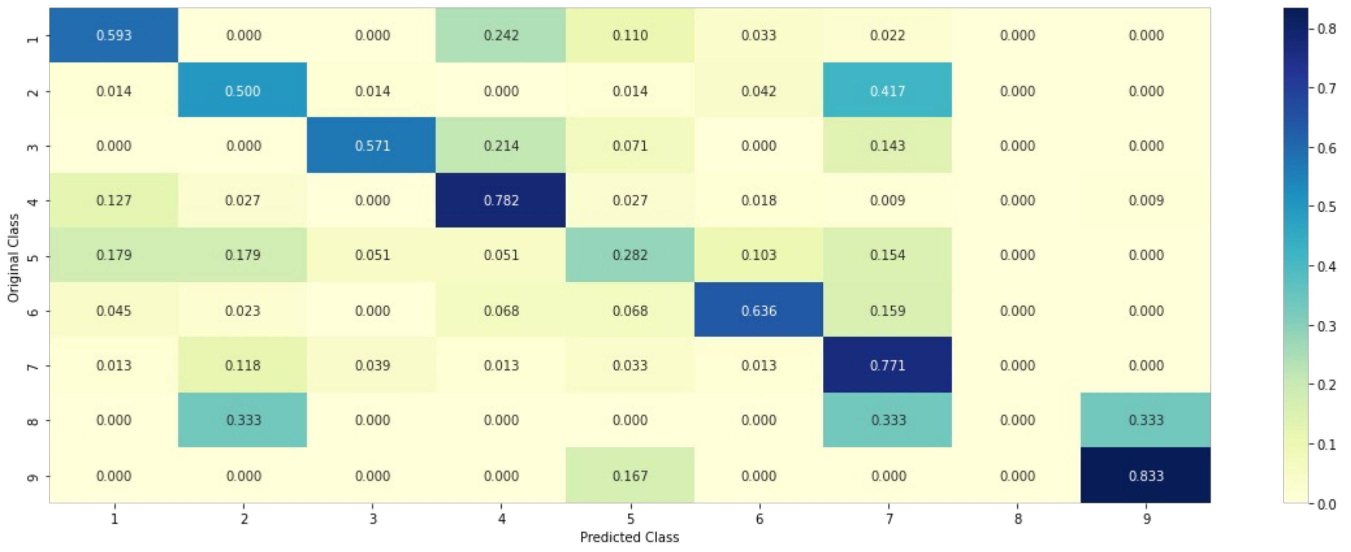
```
Log loss : 1.0485068580062582
Number of mis-classified points : 0.34962406015037595
-------------------- Confusion matrix --------------------
```



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



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



4.2.3.Sample Query point -1

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

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

```
Predicted Class : 4
Actual Class : 7
The  5  nearest neighbours of the test points belongs to classes [4 4 4 4 4]
Fequency of nearest points : Counter({4: 5})
```

4.2.4. Sample Query Point-2

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

        test_point_index = 100

        predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
```

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

```
Predicted Class : 6
Actual Class : 6
the k value for knn is 5 and the nearest neighbours of the test points belongs to classes [6 6 6 6 6]
Fequency of nearest points : Counter({6: 5})
```

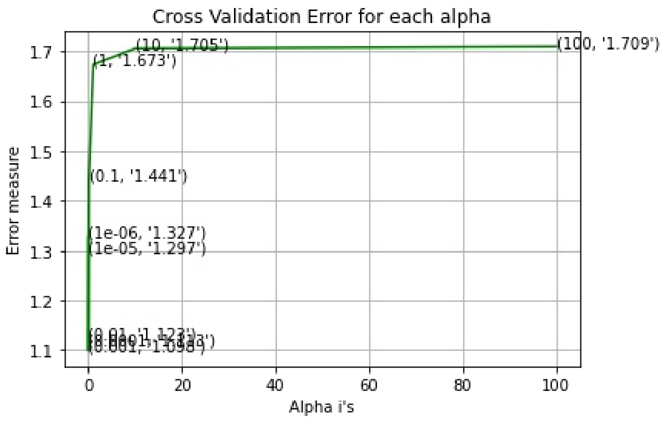
4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [ ]:

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

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

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


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

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

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, l
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, pred
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, lab
for alpha = 1e-06
Log Loss : 1.326760209961165
for alpha = 1e-05
Log Loss : 1.2969007800970345
for alpha = 0.0001
Log Loss : 1.1125552645658294
for alpha = 0.001
Log Loss : 1.0984880478285448
for alpha = 0.01
Log Loss : 1.1230912128287551
for alpha = 0.1
Log Loss : 1.4413381298984405
for alpha = 1
Log Loss : 1.6729751069150949
for alpha = 10
Log Loss : 1.705499507577818
for alpha = 100
Log Loss : 1.7090966790457829
```

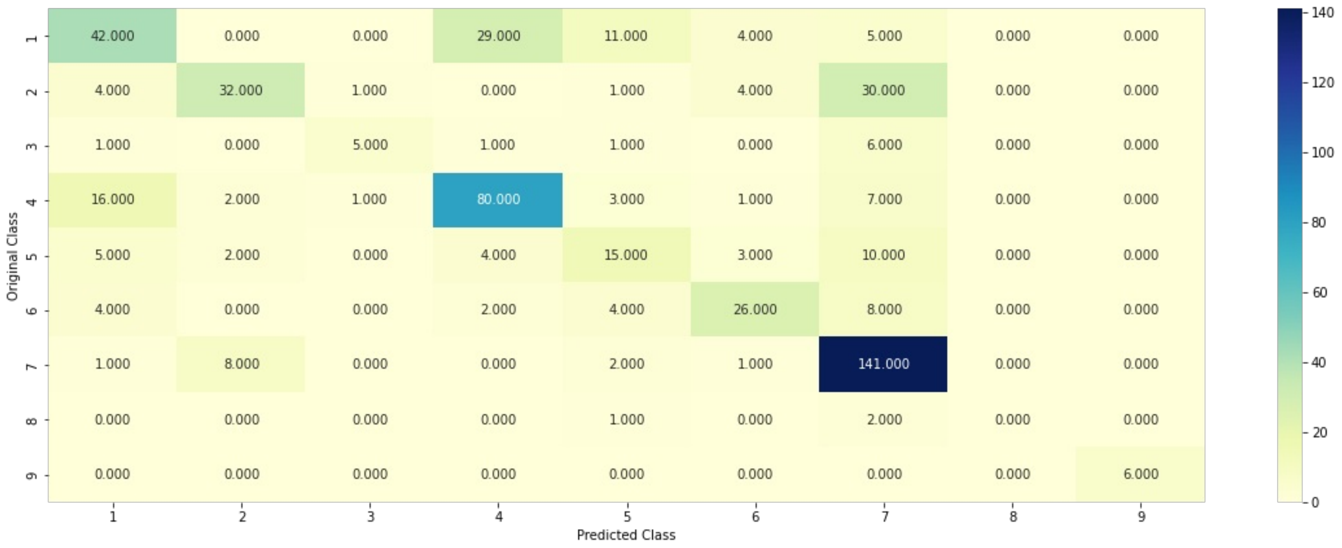


```
For values of best alpha =  0.001 The train log loss is: 0.5251646646890683
For values of best alpha =  0.001 The cross validation log loss is: 1.0984880478285448
For values of best alpha =  0.001 The test log loss is: 1.1062623242332932
```

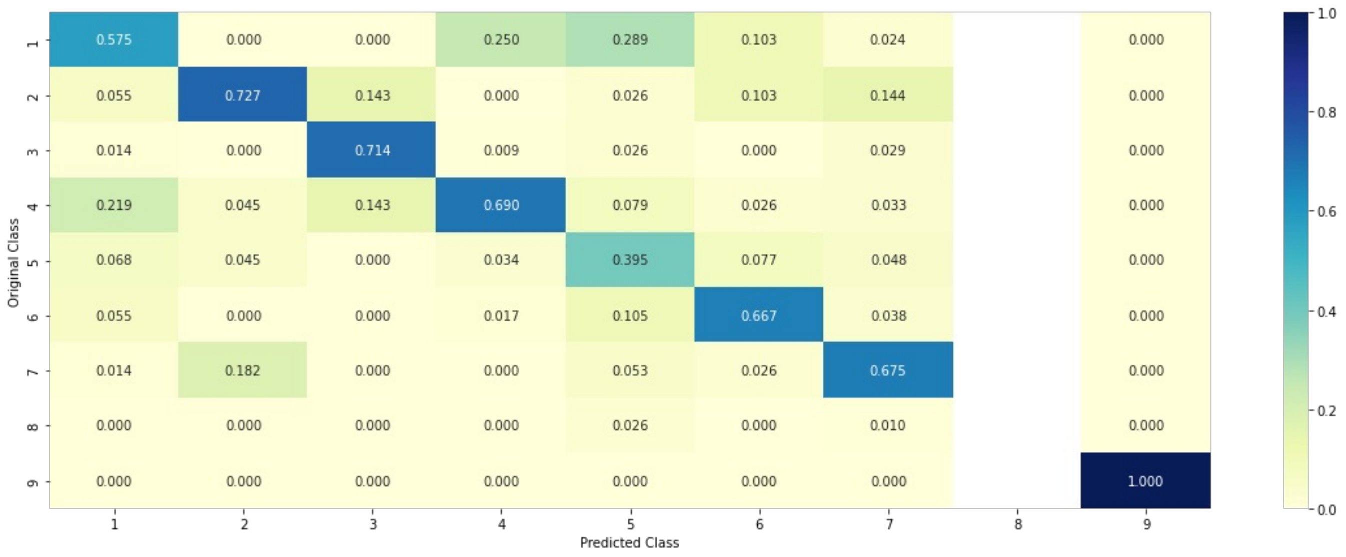
4.3.1.2. Testing the model with best hyper paramters

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

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

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

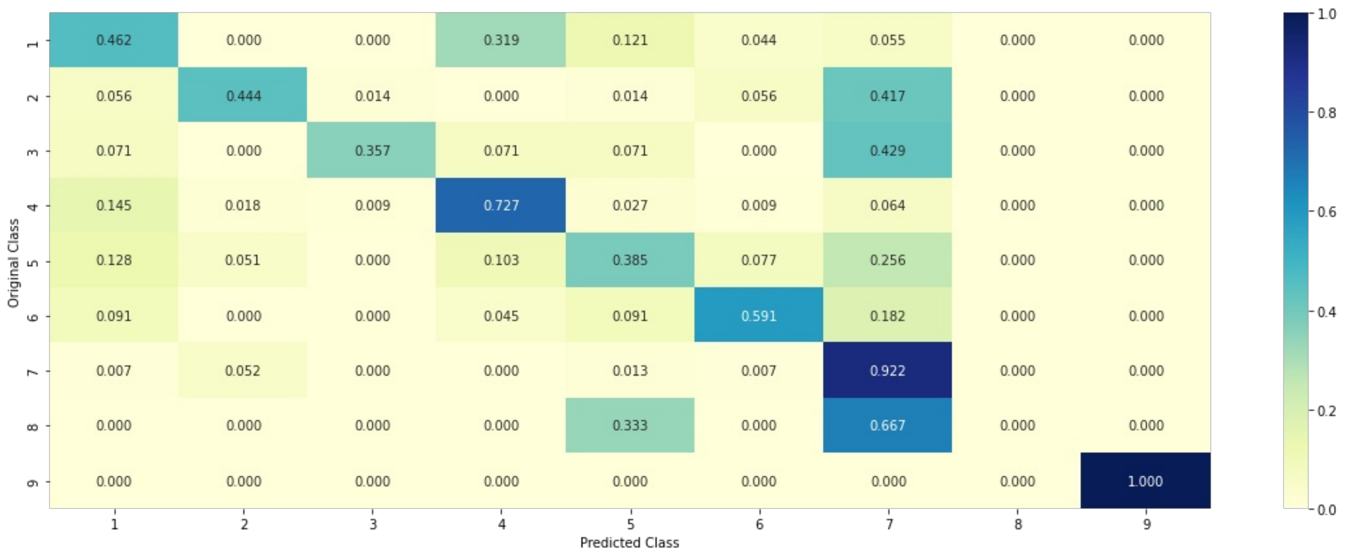
```
Log loss : 1.0984880478285448
Number of mis-classified points : 0.34774436090225563
-------------------- Confusion matrix --------------------
```



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



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



4.3.1.3. Feature Importance

```
In [ ]: def get_imp_feature_names(text, indices, removed_ind = []):
            word_present = 0
            tabulte_list = []
            incresingorder_ind = 0
            for i in indices:
                if i < train_gene_feature_onehotCoding.shape[1]:
                    tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
                elif i< 18:
                    tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
                if ((i > 17) & (i not in removed_ind)) :
                    word = train_text_features[i]
                    yes_no = True if word in text.split() else False
                    if yes_no:
                        word_present += 1
                    tabulte_list.append([incresingorder_ind,train_text_features[i], yes_no])
                incresingorder_ind += 1
            print(word_present, "most importent features are present in our query point")
            print("-"*50)
            print("The features that are most importent of the ",predicted_cls[0]," class:")
            print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or Not']))
```

4.3.1.3.1. Correctly Classified point

```
In [ ]: # from tabulate import tabulate
        clf = SGDClassifier(class_weight='balanced', 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(-1*abs(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],
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0206 0.0287 0.0055 0.8692 0.0142 0.0093 0.0448 0.0046 0.0032]]
Actual Class : 7
--------------------------------------------------
179 Text feature [activating] present in test data point [True]
207 Text feature [suppressor] present in test data point [True]
403 Text feature [riley] present in test data point [True]
450 Text feature [determinants] present in test data point [True]
484 Text feature [agar] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [ ]:
```
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(-1*abs(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],
```

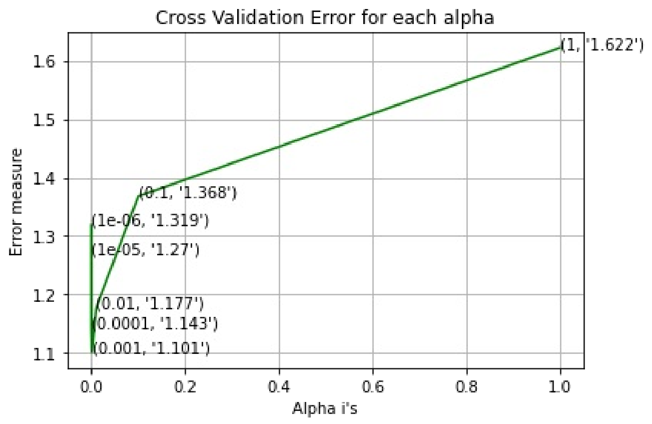
```
Predicted Class : 6
Predicted Class Probabilities: [[1.90e-03 1.23e-02 1.70e-03 1.70e-03 3.30e-03 9.66e-01 9.10e-03 3.10e-03
  9.00e-04]]
Actual Class : 6
--------------------------------------------------
271 Text feature [mutants] present in test data point [True]
340 Text feature [3a] present in test data point [True]
350 Text feature [3b] present in test data point [True]
379 Text feature [affecting] present in test data point [True]
399 Text feature [expressing] present in test data point [True]
409 Text feature [described] present in test data point [True]
429 Text feature [4a] present in test data point [True]
445 Text feature [assays] present in test data point [True]
450 Text feature [weakened] present in test data point [True]
459 Text feature [figure] present in test data point [True]
Out of the top  500  features  10 are present in query point
```

4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

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

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

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



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

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

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, l
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, pred
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, lab
```

```
for alpha = 1e-06
Log Loss : 1.3193643509907436
for alpha = 1e-05
Log Loss : 1.2703909395463335
for alpha = 0.0001
Log Loss : 1.1429802230719204
for alpha = 0.001
Log Loss : 1.1012093614904157
for alpha = 0.01
Log Loss : 1.177021876437383
for alpha = 0.1
Log Loss : 1.368063059507093
for alpha = 1
Log Loss : 1.6219908620557617
```

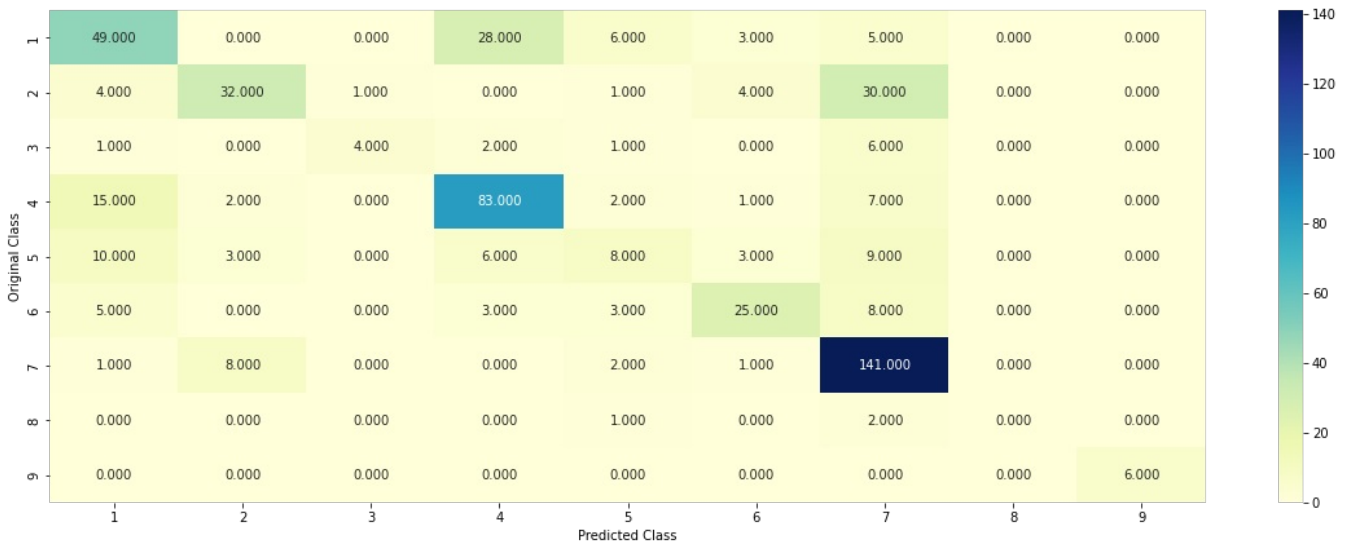


```
For values of best alpha =  0.001 The train log loss is: 0.5275928752281991
For values of best alpha =  0.001 The cross validation log loss is: 1.1012093614904157
For values of best alpha =  0.001 The test log loss is: 1.107728663036323
```

4.3.2.2. Testing model with best hyper parameters

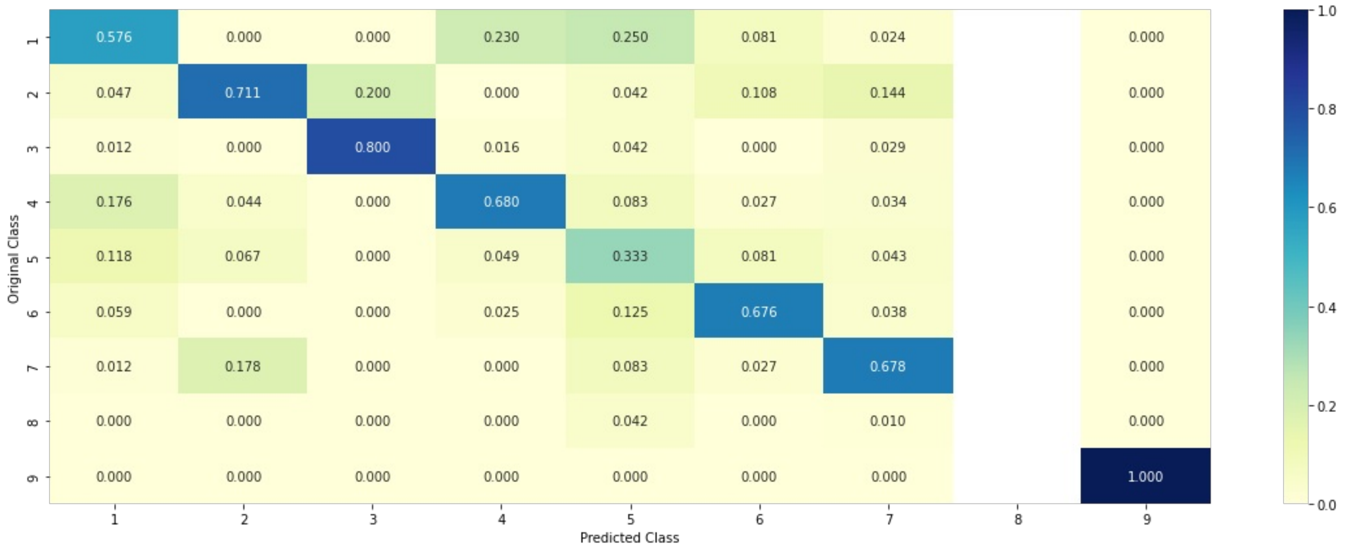
In [ ]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDC
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t
# 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, clf)
```

```
Log loss : 1.1012093614904157
Number of mis-classified points : 0.3458646616541353
-------------------- Confusion matrix --------------------
```



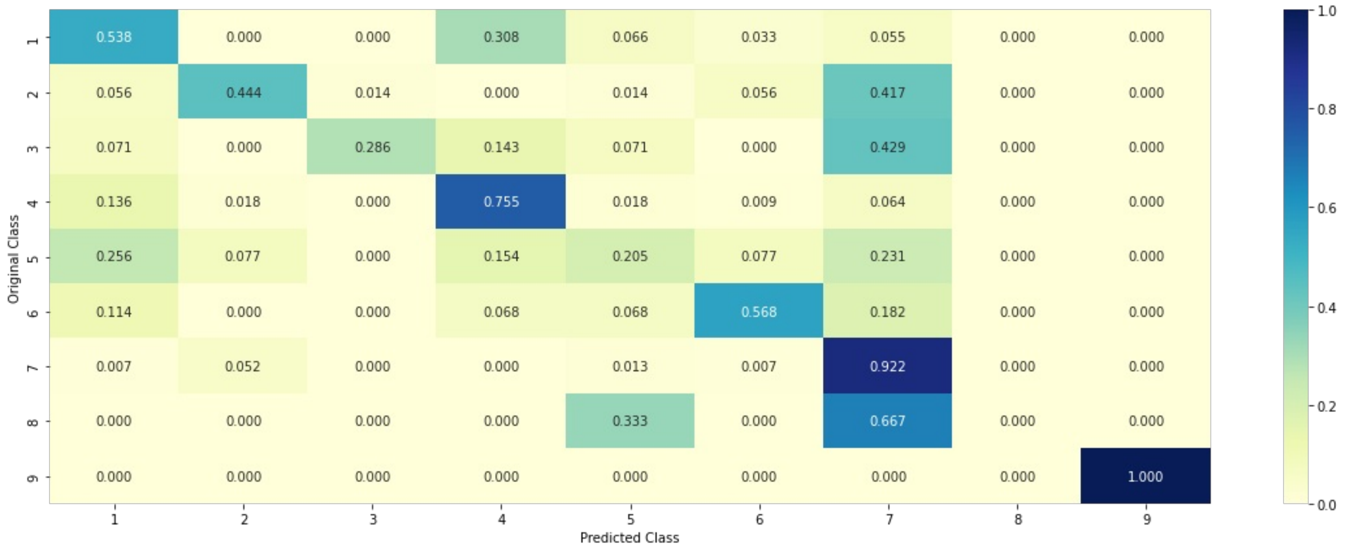
	1	2	3	4	5	6	7	8	9
1	49.000	0.000	0.000	28.000	6.000	3.000	5.000	0.000	0.000
2	4.000	32.000	1.000	0.000	1.000	4.000	30.000	0.000	0.000
3	1.000	0.000	4.000	2.000	1.000	0.000	6.000	0.000	0.000
4	15.000	2.000	0.000	83.000	2.000	1.000	7.000	0.000	0.000
5	10.000	3.000	0.000	6.000	8.000	3.000	9.000	0.000	0.000
6	5.000	0.000	0.000	3.000	3.000	25.000	8.000	0.000	0.000
7	1.000	8.000	0.000	0.000	2.000	1.000	141.000	0.000	0.000
8	0.000	0.000	0.000	0.000	1.000	0.000	2.000	0.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	6.000

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



	1	2	3	4	5	6	7	8	9
1	0.576	0.000	0.000	0.230	0.250	0.081	0.024		0.000
2	0.047	0.711	0.200	0.000	0.042	0.108	0.144		0.000
3	0.012	0.000	0.800	0.016	0.042	0.000	0.029		0.000
4	0.176	0.044	0.000	0.680	0.083	0.027	0.034		0.000
5	0.118	0.067	0.000	0.049	0.333	0.081	0.043		0.000
6	0.059	0.000	0.000	0.025	0.125	0.676	0.038		0.000
7	0.012	0.178	0.000	0.000	0.083	0.027	0.678		0.000
8	0.000	0.000	0.000	0.000	0.042	0.000	0.010		0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000		1.000

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



	1	2	3	4	5	6	7	8	9
1	0.538	0.000	0.000	0.308	0.066	0.033	0.055	0.000	0.000
2	0.056	0.444	0.014	0.000	0.014	0.056	0.417	0.000	0.000
3	0.071	0.000	0.286	0.143	0.071	0.000	0.429	0.000	0.000
4	0.136	0.018	0.000	0.755	0.018	0.009	0.064	0.000	0.000
5	0.256	0.077	0.000	0.154	0.205	0.077	0.231	0.000	0.000
6	0.114	0.000	0.000	0.068	0.068	0.568	0.182	0.000	0.000
7	0.007	0.052	0.000	0.000	0.013	0.007	0.922	0.000	0.000
8	0.000	0.000	0.000	0.000	0.333	0.000	0.667	0.000	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000

4.3.2.3. Feature Importance, Correctly Classified point

In [ ]:

```
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(-1*abs(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],
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.021  0.0292 0.0064 0.8658 0.0143 0.0093 0.0482 0.0043 0.0015]]
Actual Class : 7
--------------------------------------------------
260 Text feature [suppressor] present in test data point [True]
351 Text feature [activating] present in test data point [True]
465 Text feature [cultivated] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [ ]:
```
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(-1*abs(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],
```

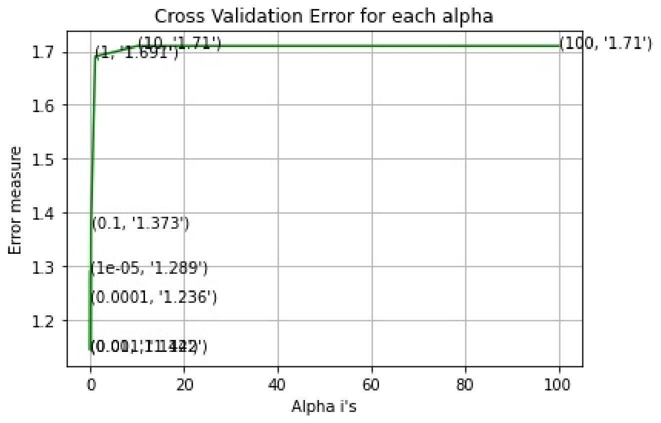
```
Predicted Class : 6
Predicted Class Probabilities: [[2.300e-03 1.300e-02 1.700e-03 2.200e-03 3.200e-03 9.635e-01 1.120e-02
  2.900e-03 2.000e-04]]
Actual Class : 6
--------------------------------------------------
239 Text feature [mutants] present in test data point [True]
311 Text feature [3b] present in test data point [True]
312 Text feature [3a] present in test data point [True]
323 Text feature [affecting] present in test data point [True]
348 Text feature [expressing] present in test data point [True]
352 Text feature [described] present in test data point [True]
389 Text feature [4a] present in test data point [True]
400 Text feature [figure] present in test data point [True]
403 Text feature [previously] present in test data point [True]
407 Text feature [assays] present in test data point [True]
417 Text feature [weakened] present in test data point [True]
433 Text feature [mutations] present in test data point [True]
438 Text feature [sequenced] present in test data point [True]
443 Text feature [findings] present in test data point [True]
446 Text feature [anti] present in test data point [True]
488 Text feature [domain] present in test data point [True]
Out of the top  500  features  16 are present in query point
```

4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [ ]:
```
# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/gener

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=No

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



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

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

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


best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=
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, l
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, pred
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, lab
```

```
for C = 1e-05
Log Loss : 1.289145220024082
for C = 0.0001
Log Loss : 1.23572675787792
for C = 0.001
Log Loss : 1.142489271341791
for C = 0.01
Log Loss : 1.142443611656499
for C = 0.1
Log Loss : 1.3733401474425544
for C = 1
Log Loss : 1.6905870906848457
for C = 10
Log Loss : 1.709718391233839
for C = 100
Log Loss : 1.70971956660859
```



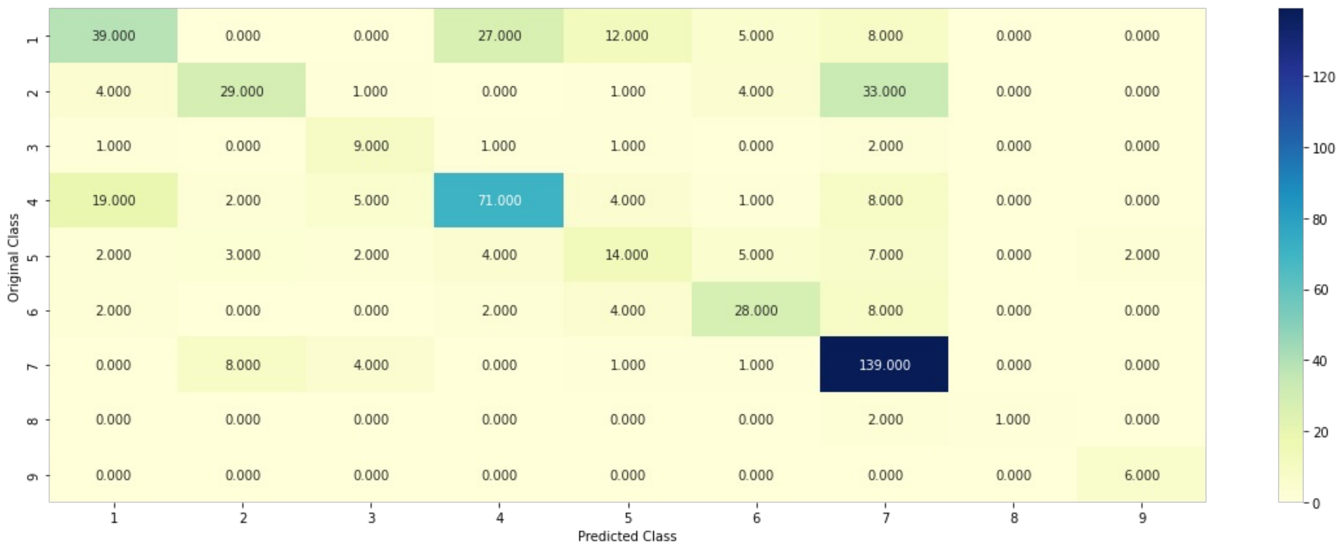
```
For values of best alpha =  0.01 The train log loss is: 0.7571255525740147
For values of best alpha =  0.01 The cross validation log loss is: 1.142443611656499
For values of best alpha =  0.01 The test log loss is: 1.171790348003692
```

4.4.2. Testing model with best hyper parameters

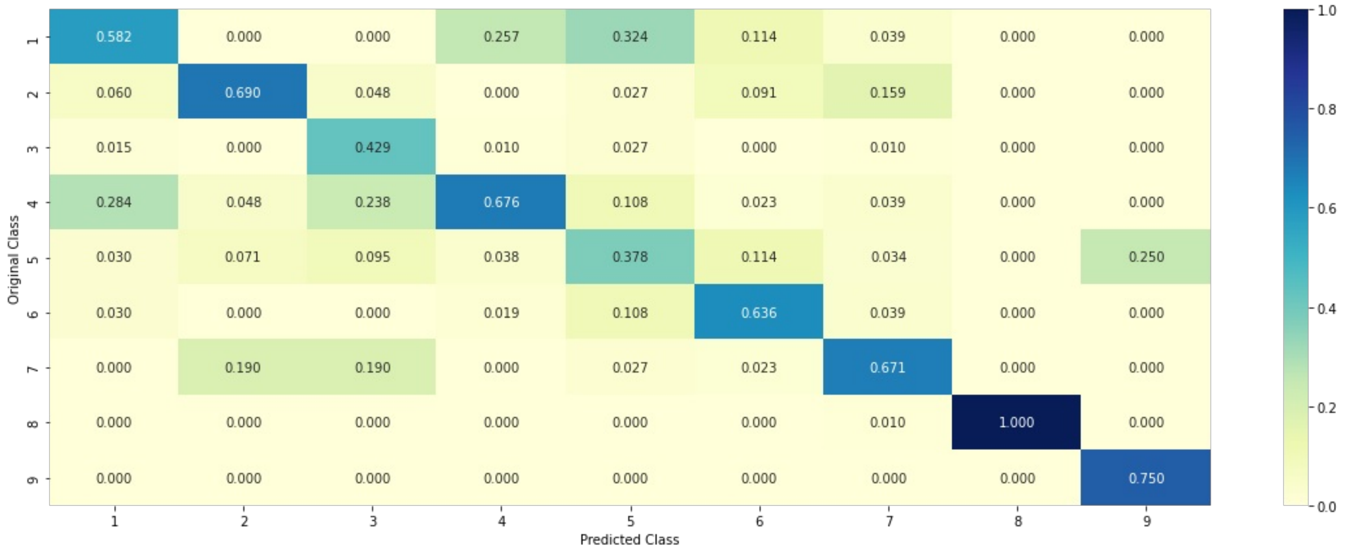
```
In [ ]: # read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/gener

        # --------------------------------
        # default parameters
        # SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
        # cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=No

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


        # clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='balanced')
```

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

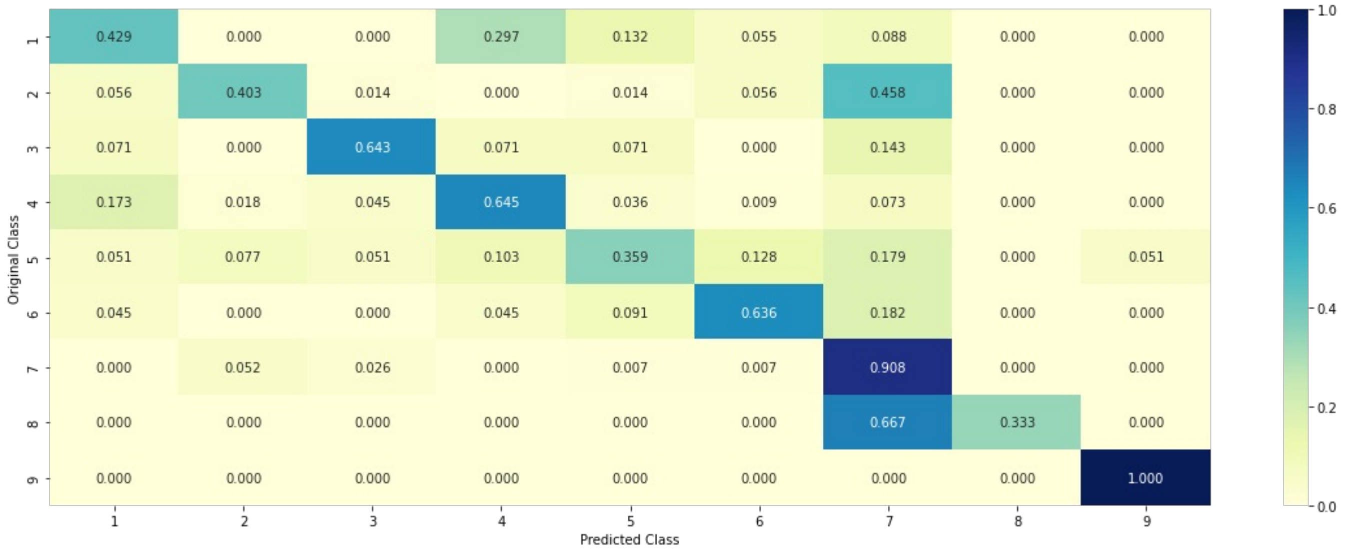
```
Log loss : 1.142443611656499
Number of mis-classified points : 0.3684210526315789
-------------------- Confusion matrix --------------------
```



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



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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0496 0.0455 0.0082 0.7373 0.0278 0.0186 0.1066 0.0035 0.003 ]]
Actual Class : 7
--------------------------------------------------
124 Text feature [suppressor] present in test data point [True]
184 Text feature [riley] present in test data point [True]
352 Text feature [pten] present in test data point [True]
403 Text feature [microscopy] present in test data point [True]
468 Text feature [determinants] present in test data point [True]
478 Text feature [dapi] present in test data point [True]
479 Text feature [ruvalcaba] present in test data point [True]
482 Text feature [bannayan] present in test data point [True]
Out of the top  500  features  8 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [ ]:

```
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(-1*abs(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],
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0322 0.0429 0.007  0.0342 0.0244 0.7905 0.0636 0.0029 0.0022]]
Actual Class : 6
--------------------------------------------------
221 Text feature [weakened] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [ ]:

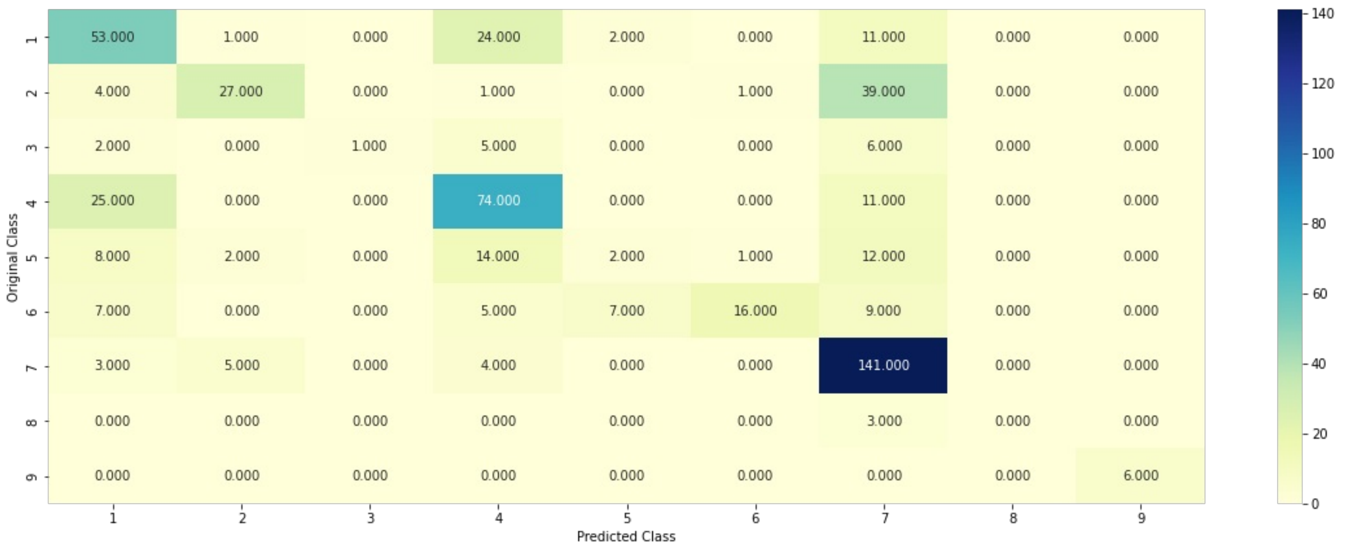
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decr
# min_impurity_split=None,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

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

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

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


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

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

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

best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(b
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log loss is:",log_loss(y_train, p
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validation log loss is:",log_loss
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss is:",log_loss(y_test, pre
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2466921329487382
for n_estimators = 100 and max depth =  10
Log Loss : 1.1807728571039329
for n_estimators = 200 and max depth =  5
Log Loss : 1.2331334291493237
for n_estimators = 200 and max depth =  10
Log Loss : 1.172574974013667
for n_estimators = 500 and max depth =  5
Log Loss : 1.2250243375392598
for n_estimators = 500 and max depth =  10
Log Loss : 1.1672393949842135
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2263050502722521
for n_estimators = 1000 and max depth =  10
Log Loss : 1.163245563203369
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2236713663860606
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1603032267997027
For values of best estimator =  2000 The train log loss is: 0.685826535198577
For values of best estimator =  2000 The cross validation log loss is: 1.1603032267997027
For values of best estimator =  2000 The test log loss is: 1.1573690955880913
```

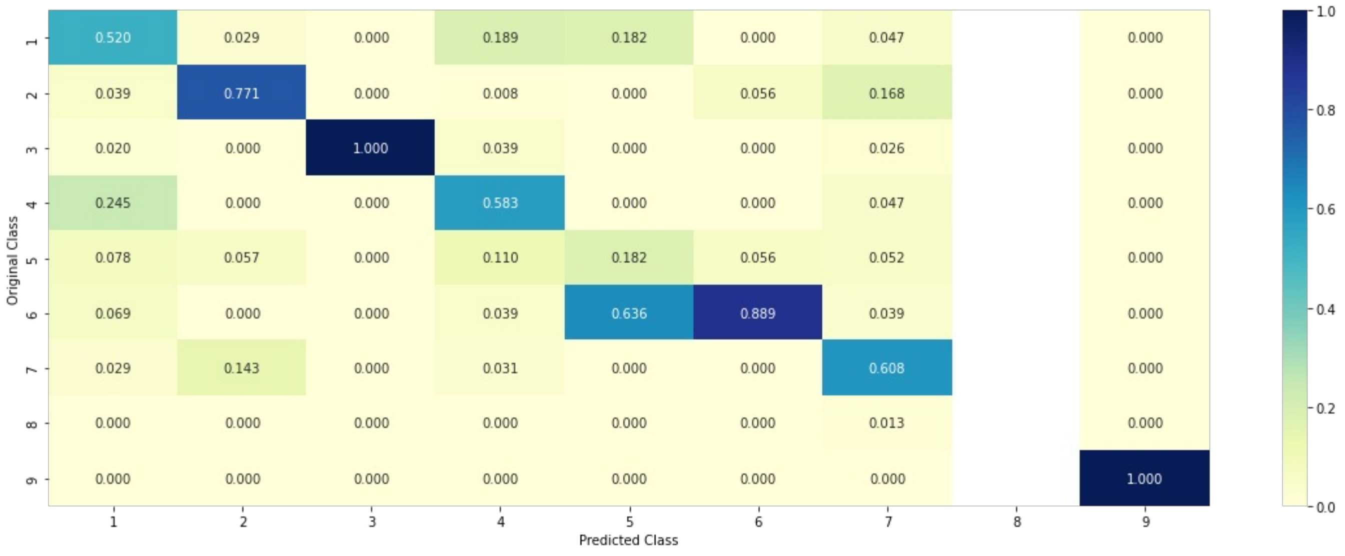
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

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

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

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

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

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(b
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

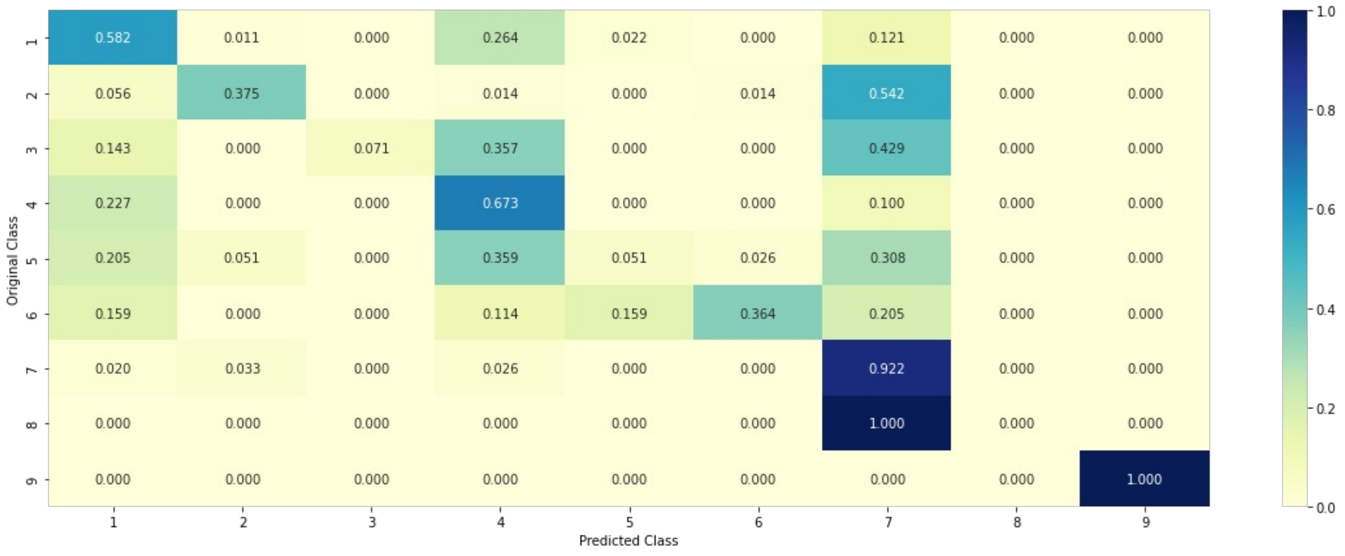
```
Log loss : 1.1603032267997027
Number of mis-classified points : 0.39849624060150374
-------------------- Confusion matrix --------------------
```



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



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [ ]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(b
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

test_point_index = 1
```

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.106  0.0498 0.0204 0.6337 0.0441 0.0378 0.0954 0.0057 0.0071]]
Actual Class : 7
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
3 Text feature [activation] present in test data point [True]
6 Text feature [phosphorylation] present in test data point [True]
7 Text feature [function] present in test data point [True]
8 Text feature [constitutive] present in test data point [True]
9 Text feature [treatment] present in test data point [True]
10 Text feature [activated] present in test data point [True]
11 Text feature [signaling] present in test data point [True]
12 Text feature [downstream] present in test data point [True]
13 Text feature [growth] present in test data point [True]
16 Text feature [oncogenic] present in test data point [True]
17 Text feature [receptor] present in test data point [True]
18 Text feature [inhibitor] present in test data point [True]
19 Text feature [cells] present in test data point [True]
21 Text feature [loss] present in test data point [True]
24 Text feature [akt] present in test data point [True]
26 Text feature [suppressor] present in test data point [True]
29 Text feature [constitutively] present in test data point [True]
44 Text feature [proliferation] present in test data point [True]
46 Text feature [functional] present in test data point [True]
52 Text feature [protein] present in test data point [True]
53 Text feature [cell] present in test data point [True]
58 Text feature [expressing] present in test data point [True]
68 Text feature [phosphatase] present in test data point [True]
70 Text feature [nuclear] present in test data point [True]
73 Text feature [expression] present in test data point [True]
75 Text feature [pten] present in test data point [True]
77 Text feature [serum] present in test data point [True]
84 Text feature [starved] present in test data point [True]
85 Text feature [lines] present in test data point [True]
87 Text feature [proteins] present in test data point [True]
Out of the top  100  features  32 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [ ]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_poi
Predicted Class : 6
Predicted Class Probabilities: [[0.0262 0.0321 0.0171 0.0316 0.0336 0.7837 0.0661 0.0052 0.0044]]
Actuall Class : 6
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [inhibitors] present in test data point [True]
3 Text feature [activation] present in test data point [True]
4 Text feature [tyrosine] present in test data point [True]
6 Text feature [phosphorylation] present in test data point [True]
8 Text feature [constitutive] present in test data point [True]
9 Text feature [treatment] present in test data point [True]
10 Text feature [activated] present in test data point [True]
11 Text feature [signaling] present in test data point [True]
13 Text feature [growth] present in test data point [True]
14 Text feature [therapeutic] present in test data point [True]
15 Text feature [kinases] present in test data point [True]
16 Text feature [oncogenic] present in test data point [True]
17 Text feature [receptor] present in test data point [True]
18 Text feature [inhibitor] present in test data point [True]
19 Text feature [cells] present in test data point [True]
20 Text feature [therapy] present in test data point [True]
21 Text feature [loss] present in test data point [True]
22 Text feature [treated] present in test data point [True]
28 Text feature [stability] present in test data point [True]
29 Text feature [constitutively] present in test data point [True]
35 Text feature [imatinib] present in test data point [True]
39 Text feature [variants] present in test data point [True]
42 Text feature [ic50] present in test data point [True]
43 Text feature [drug] present in test data point [True]
44 Text feature [proliferation] present in test data point [True]
45 Text feature [resistance] present in test data point [True]
46 Text feature [functional] present in test data point [True]
47 Text feature [defective] present in test data point [True]
49 Text feature [trials] present in test data point [True]
50 Text feature [patients] present in test data point [True]
51 Text feature [inhibition] present in test data point [True]
52 Text feature [protein] present in test data point [True]
53 Text feature [cell] present in test data point [True]
55 Text feature [efficacy] present in test data point [True]
57 Text feature [egfr] present in test data point [True]
58 Text feature [expressing] present in test data point [True]
61 Text feature [activate] present in test data point [True]
64 Text feature [mek] present in test data point [True]
67 Text feature [inhibited] present in test data point [True]
68 Text feature [phosphatase] present in test data point [True]
72 Text feature [oncogene] present in test data point [True]
73 Text feature [expression] present in test data point [True]
78 Text feature [il] present in test data point [True]
79 Text feature [clinical] present in test data point [True]
81 Text feature [ligand] present in test data point [True]
85 Text feature [lines] present in test data point [True]
87 Text feature [proteins] present in test data point [True]
89 Text feature [sensitive] present in test data point [True]
90 Text feature [phospho] present in test data point [True]
93 Text feature [variant] present in test data point [True]
Out of the top  100  features  52 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [ ]:

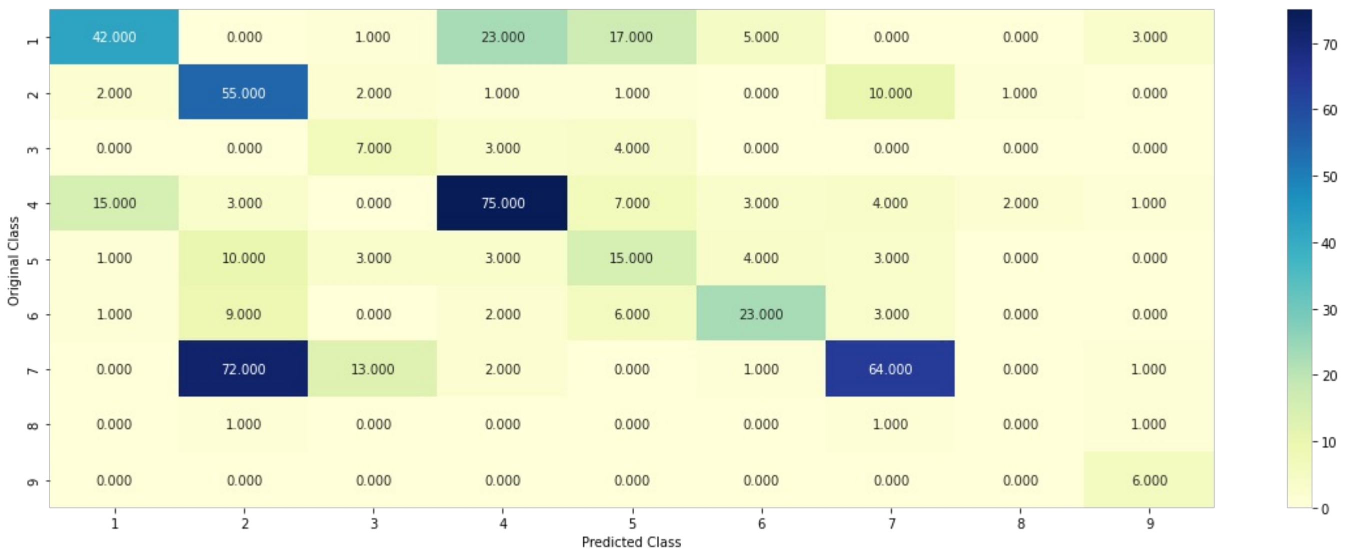
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decr
# min_impurity_split=None,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

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

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

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


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

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

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

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log loss is:",log_loss(y_train, predi
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross validation log loss is:",log_loss(y_c
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log loss is:",log_loss(y_test, predict
for n_estimators = 10 and max depth =  2
Log Loss : 2.1390831914062893
for n_estimators = 10 and max depth =  3
Log Loss : 1.6292419654115449
for n_estimators = 10 and max depth =  5
Log Loss : 1.3844312406895536
for n_estimators = 10 and max depth =  10
Log Loss : 1.5428893198160947
for n_estimators = 50 and max depth =  2
Log Loss : 1.727722673449383
for n_estimators = 50 and max depth =  3
Log Loss : 1.4561993253415668
for n_estimators = 50 and max depth =  5
Log Loss : 1.3661555179512317
for n_estimators = 50 and max depth =  10
Log Loss : 1.6174398096515306
for n_estimators = 100 and max depth =  2
Log Loss : 1.5861809494935444
for n_estimators = 100 and max depth =  3
Log Loss : 1.4632225810116772
for n_estimators = 100 and max depth =  5
Log Loss : 1.3034549953632524
for n_estimators = 100 and max depth =  10
Log Loss : 1.6044658513456833
for n_estimators = 200 and max depth =  2
Log Loss : 1.5709937689210594
for n_estimators = 200 and max depth =  3
Log Loss : 1.4458395540684004
for n_estimators = 200 and max depth =  5
Log Loss : 1.3560872207512635
for n_estimators = 200 and max depth =  10
Log Loss : 1.6197442228356262
for n_estimators = 500 and max depth =  2
Log Loss : 1.6185581568309115
for n_estimators = 500 and max depth =  3
Log Loss : 1.5110008792646619
for n_estimators = 500 and max depth =  5
Log Loss : 1.3906505190731293
for n_estimators = 500 and max depth =  10
Log Loss : 1.6412378672265269
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6055583753980949
for n_estimators = 1000 and max depth =  3
Log Loss : 1.516460266675378
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3913266813009235
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6359416965111886
For values of best alpha =  100 The train log loss is: 0.07414488996302426
For values of best alpha =  100 The cross validation log loss is: 1.3034549953632524
For values of best alpha =  100 The test log loss is: 1.2621020216620713
```

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

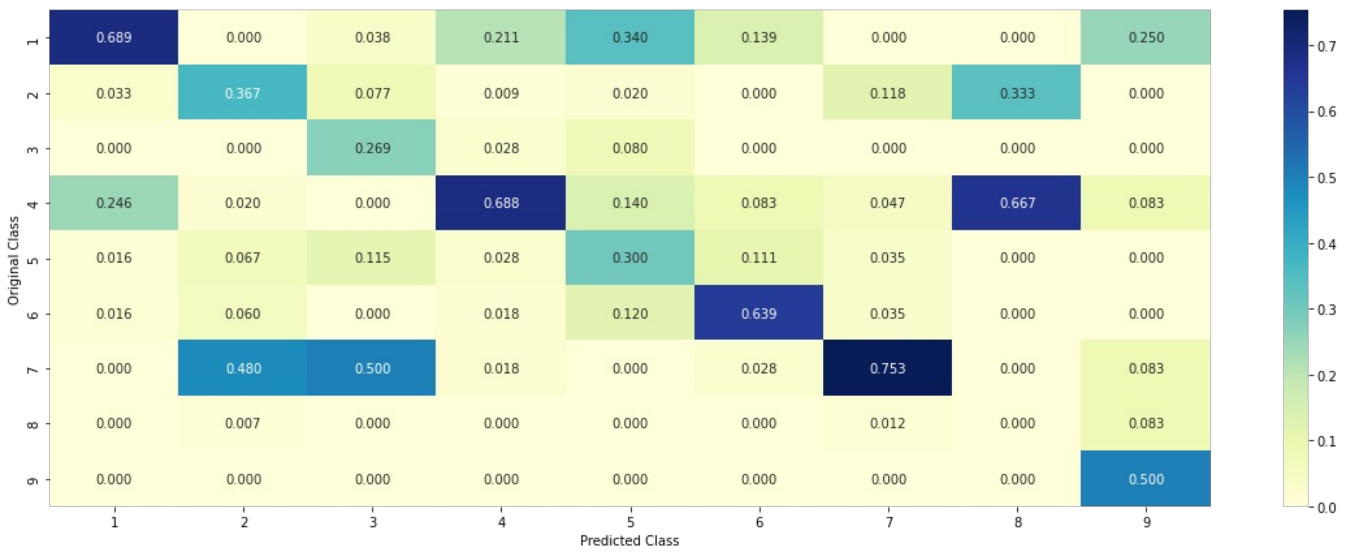
In [ ]:

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

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

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

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

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimators=alpha[int(best_alpha/4)], cri
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCoding,cv_y, clf)
```

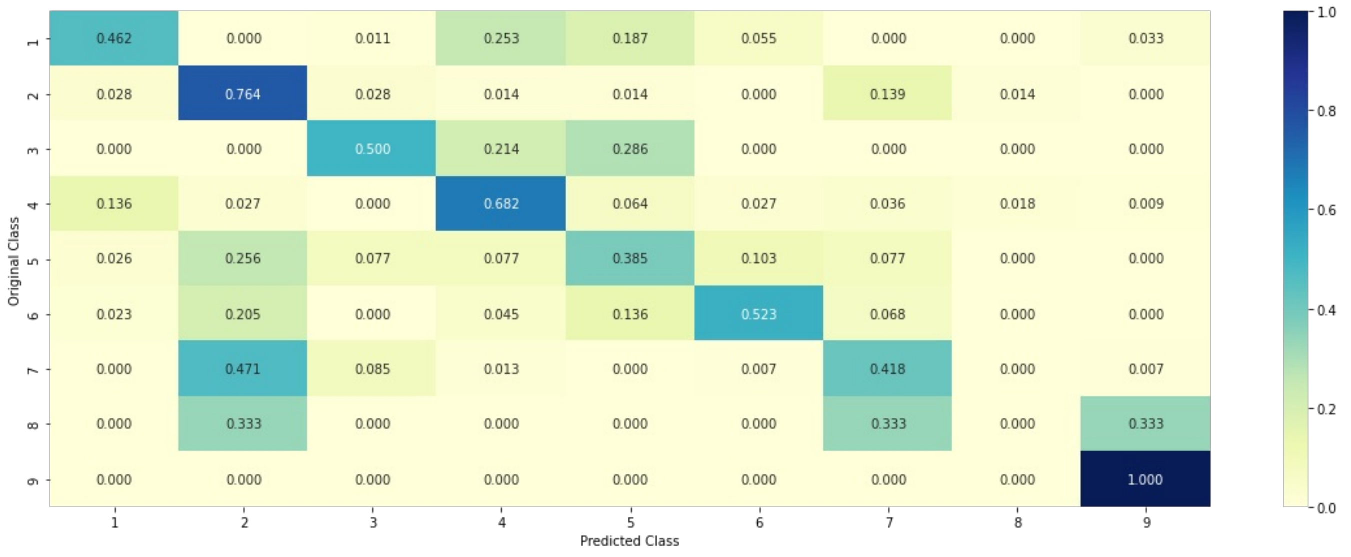
```
Log loss : 1.3034549953632522
Number of mis-classified points : 0.4605263157894737
-------------------- Confusion matrix --------------------
```



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



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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

```
In [ ]: clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=max_depth[int(b
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```
sig_clf.fit(train_x_responseCoding, train_y)


test_point_index = 1
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index].
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.097  0.0422 0.22   0.4938 0.0289 0.0422 0.0146 0.0236 0.0378]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [ ]:

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

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0317 0.1353 0.1648 0.0294 0.0461 0.2676 0.2447 0.0561 0.0243]]
Actual Class : 6
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

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

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

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


# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/gener
# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=No

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


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

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

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

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

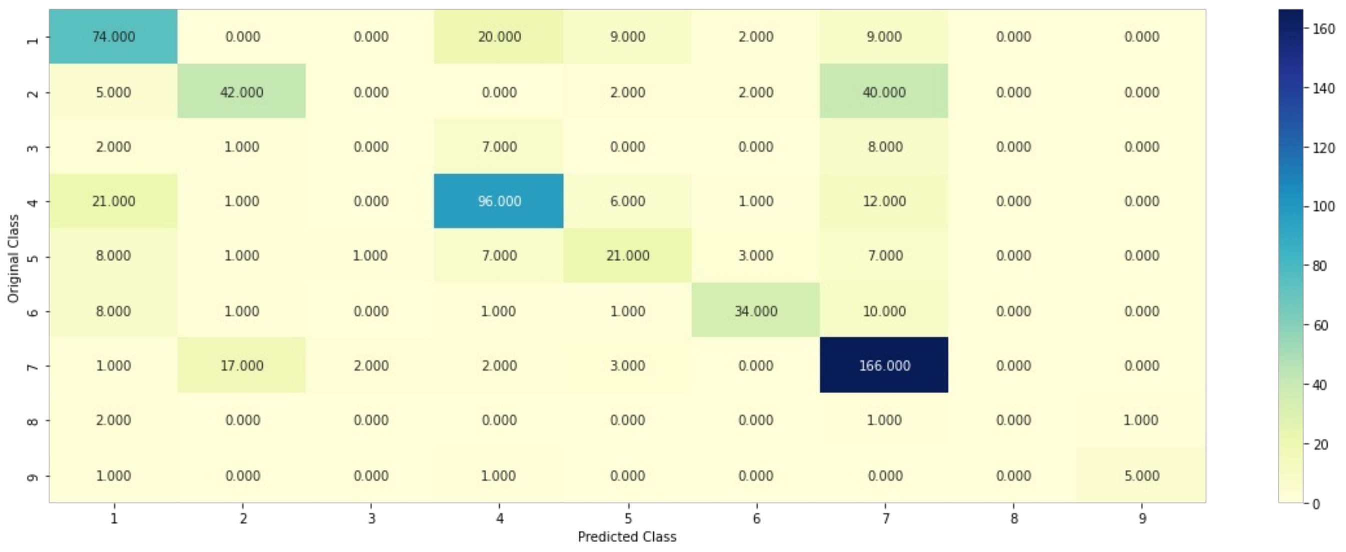
```
clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced', random_state=0)
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

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


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

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

```
Logistic Regression :  Log Loss: 1.10
Support vector machines : Log Loss: 1.69
Naive Bayes : Log Loss: 1.29
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.819
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.730
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.351
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.193
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.481
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.859
```

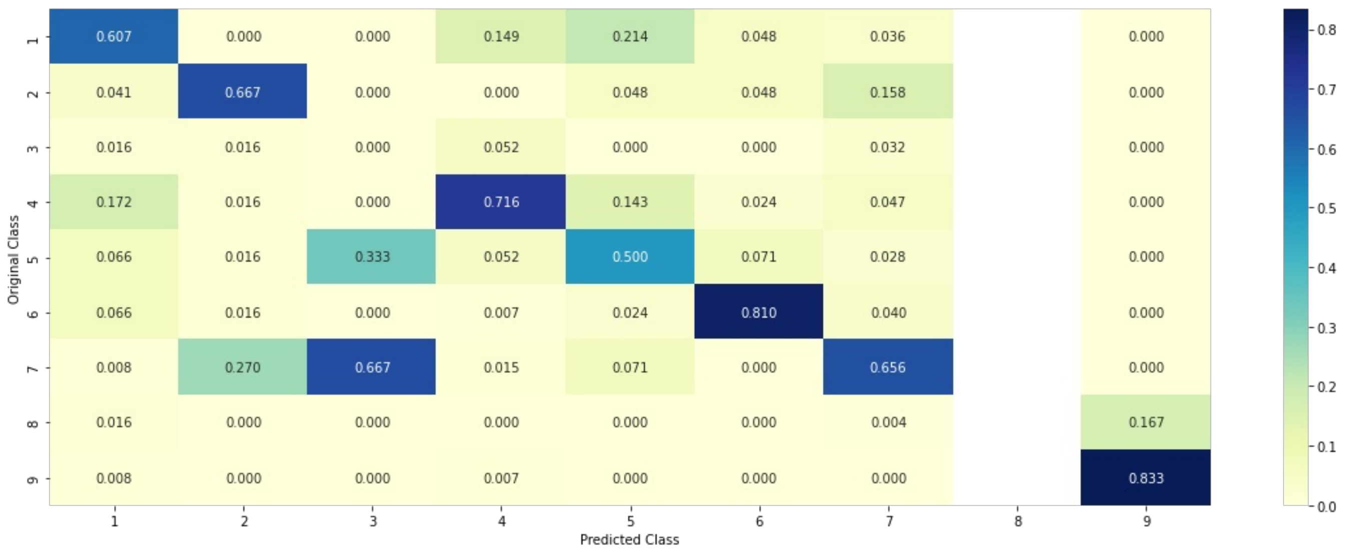
4.7.2 testing the model with the best hyper parameters

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

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

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

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

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

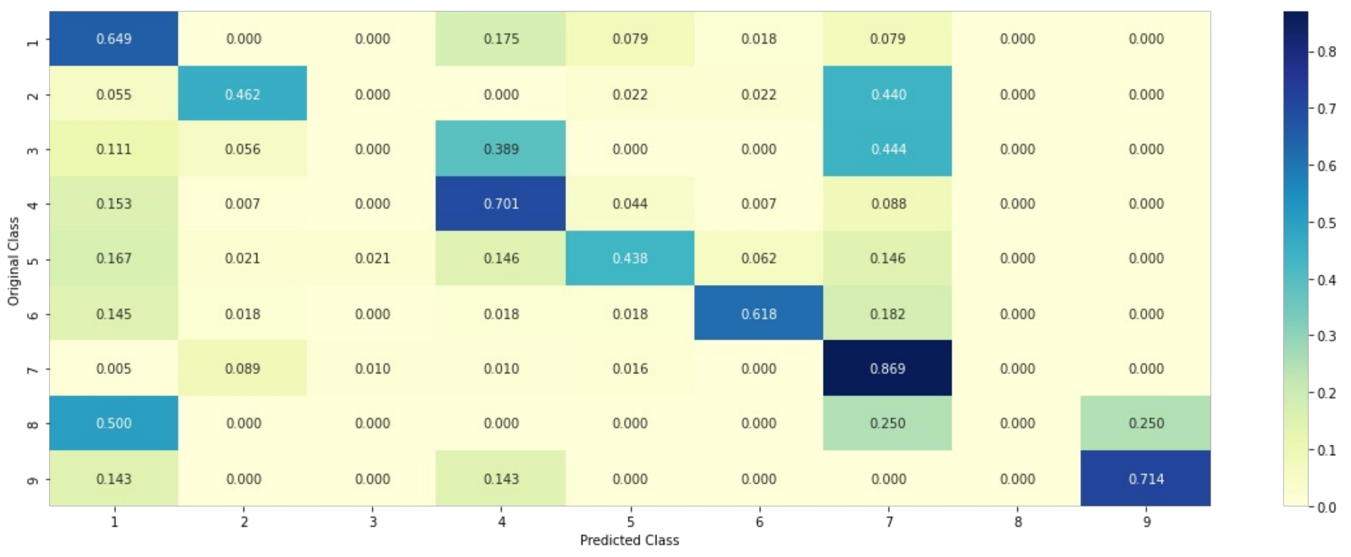
```
Log loss (train) on the stacking classifier : 0.4955889217045466
Log loss (CV) on the stacking classifier : 1.193027432270816
Log loss (test) on the stacking classifier : 1.1589190884598037
Number of missclassified point : 0.34135338345864663
-------------------- Confusion matrix --------------------
```



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



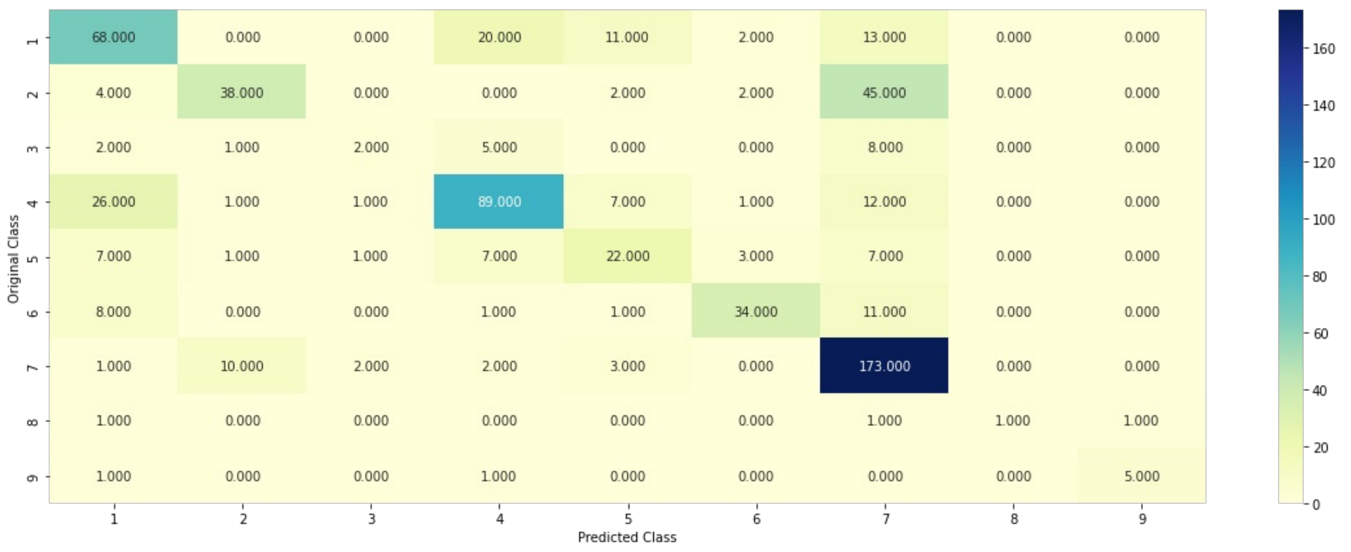
```
-------------------- Recall matrix (Row sum=1) --------------------
```



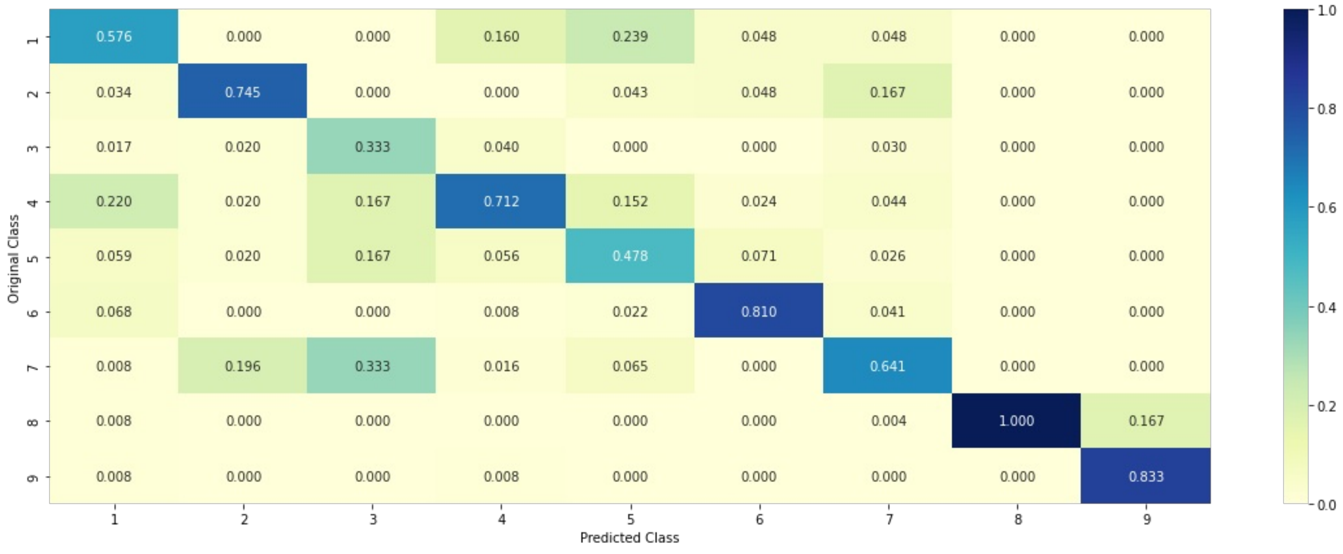
4.7.3 Maximum Voting classifier

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

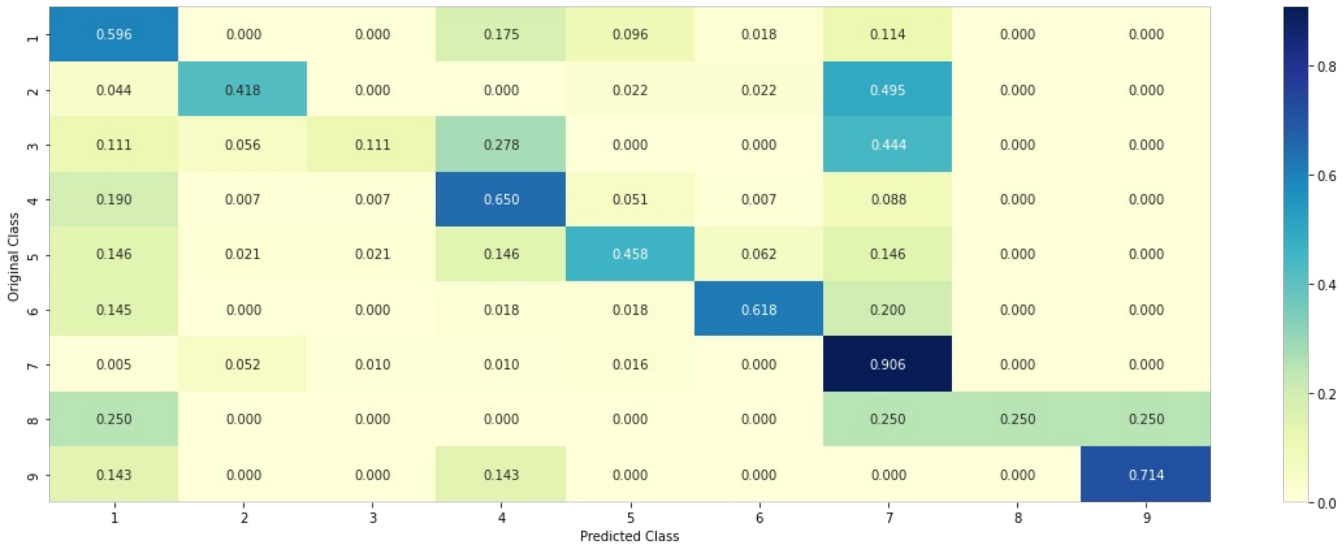
```
Log loss (train) on the VotingClassifier : 0.8736835715673336
Log loss (CV) on the VotingClassifier : 1.1885011748181689
Log loss (test) on the VotingClassifier : 1.20523929306732
Number of missclassified point : 0.35037593984962406
-------------------- Confusion matrix --------------------
```



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



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



5. Assignments

1. Apply All the models with tf-idf features (Replace CountVectorizer with tfidfVectorizer and run the same cells)
2. Instead of using all the words in the dataset, use only the top 1000 words based of tf-idf values
3. Apply Logistic regression with CountVectorizer Features, including both unigrams and bigrams
4. Try any of the feature engineering techniques discussed in the course to reduce the CV and test log-loss to a value less than 1.0