

# Personalized cancer diagnosis

## Task 2

### 1. Business Problem ¶

#### 1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training\_variants.zip and training\_text.zip from Kaggle.

##### **Context:**

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

##### **Problem statement :**

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

#### 1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

1. <https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25> (<https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25>)

2. <https://www.youtube.com/watch?v=UwbuW7oK8rk> (<https://www.youtube.com/watch?v=UwbuW7oK8rk>)
3. <https://www.youtube.com/watch?v=qxXRKVompl8> (<https://www.youtube.com/watch?v=qxXRKVompl8>)

### 1.3. Real-world/Business objectives and constraints.

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

## 2. Machine Learning Problem Formulation

### 2.1. Data

#### 2.1.1. Data Overview

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

#### 2.1.2. Example Data Point

*training\_variants*

---

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

### ***training\_text***

---

ID, Text

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

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

### **2.2.1. Type of Machine Learning Problem**

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

### **2.2.2. Performance Metric**

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

Metric(s):

- Multi class log-loss
- Confusion matrix

### **2.2.3. Machine Learning Objectives and Constraints**

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

Constraints:

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

## **2.3. Train, CV and Test Datasets**

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

## **3. Exploratory Data Analysis**

```
In [1]: import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.cross_validation import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

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

C:\Users\Himanshu Pc\Anaconda3\lib\site-packages\sklearn\cross\_validation.py:41: DeprecationWarning: This module was de

precated in version 0.18 in favor of the model\_selection module into which all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.

"This module will be removed in 0.20.", DeprecationWarning)

C:\Users\Himanshu Pc\Anaconda3\lib\site-packages\sklearn\ensemble\weight\_boosting.py:29: DeprecationWarning: numpy.core.umath\_tests is an internal NumPy module and should not be imported. It will be removed in a future NumPy release.

```
from numpy.core.umath_tests import inner1d
```

## 3.1. Reading Data

### 3.1.1. Reading Gene and Variation Data

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

Number of data points : 3321

Number of features : 4

Features : ['ID' 'Gene' 'Variation' 'Class']

Out[2]:

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

training/training\_variants is a comma separated file containing the description of the genetic mutations used for training.

Fields are

- **ID** : the id of the row used to link the mutation to the clinical evidence
- **Gene** : the gene where this genetic mutation is located

- **Variation** : the aminoacid change for this mutations
- **Class** : 1-9 the class this genetic mutation has been classified on

### 3.1.2. Reading Text Data

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

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

Out[3]:

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

```
In [4]: import nltk
nltk.download('stopwords')

[nltk_data] Downloading package stopwords to C:\Users\Himanshu
[nltk_data] Pc\AppData\Roaming\nltk_data...
[nltk_data] Package stopwords is already up-to-date!
```

Out[4]: True

### 3.1.3. Preprocessing of text

```

In [5]: # 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 [6]: #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 : 124.60713170000001 seconds

```



```
In [7]: #merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[7]:

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

Out[8]:

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

```
In [10]: result[result['ID']==1109]
```

Out[10]:

	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 [11]: 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 variable 'y_true' [stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true, test_size=0.2)
# split the train data into train and cross validation by maintaining same distribution of output variable 'y_train' [st
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 [12]: 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 [13]: # it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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

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

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

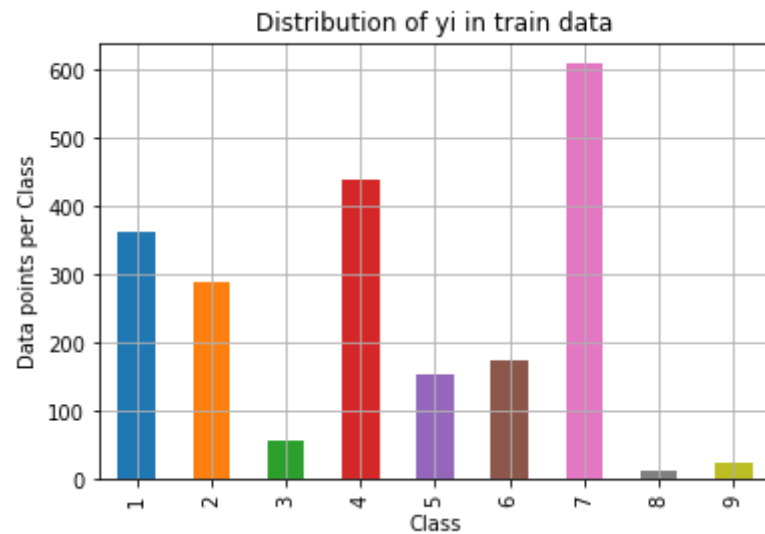
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',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

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()

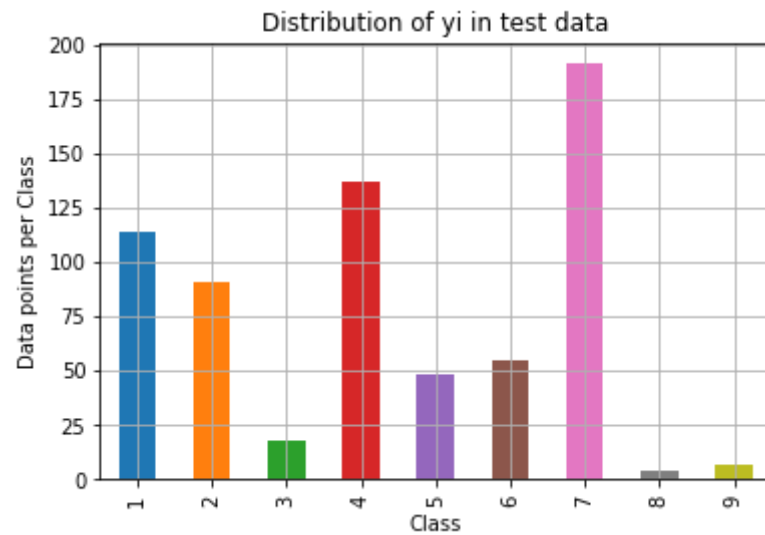
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':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



Number of data points in class 7 : 609 ( 28.672 %)  
 Number of data points in class 4 : 439 ( 20.669 %)

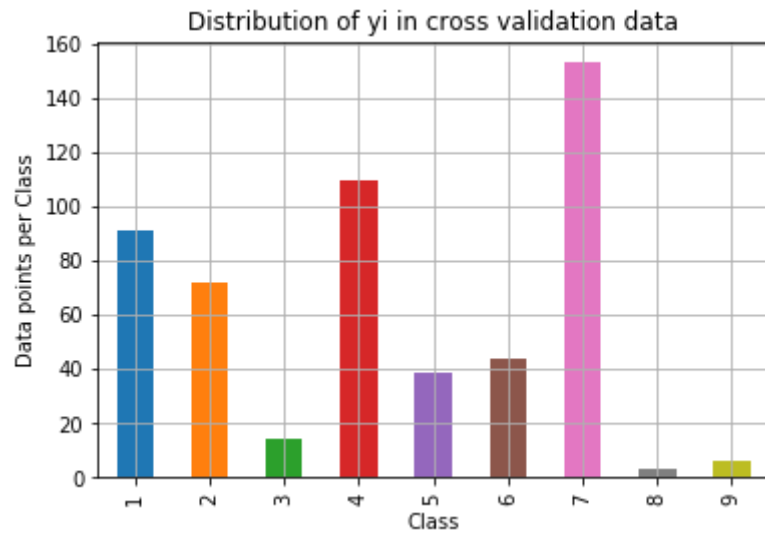
Number of data points in class 1 : 363 ( 17.09 %)  
Number of data points in class 2 : 289 ( 13.606 %)  
Number of data points in class 6 : 176 ( 8.286 %)  
Number of data points in class 5 : 155 ( 7.298 %)  
Number of data points in class 3 : 57 ( 2.684 %)  
Number of data points in class 9 : 24 ( 1.13 %)  
Number of data points in class 8 : 12 ( 0.565 %)

---



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

---



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

### 3.2 Prediction using a 'Random' Model

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



```

In [14]: # 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 corresponds to columns and axis=1 corresponds to rows in two dimensional 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 corresponds to columns and axis=1 corresponds to rows in two dimensional 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 [15]: # we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to generate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

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

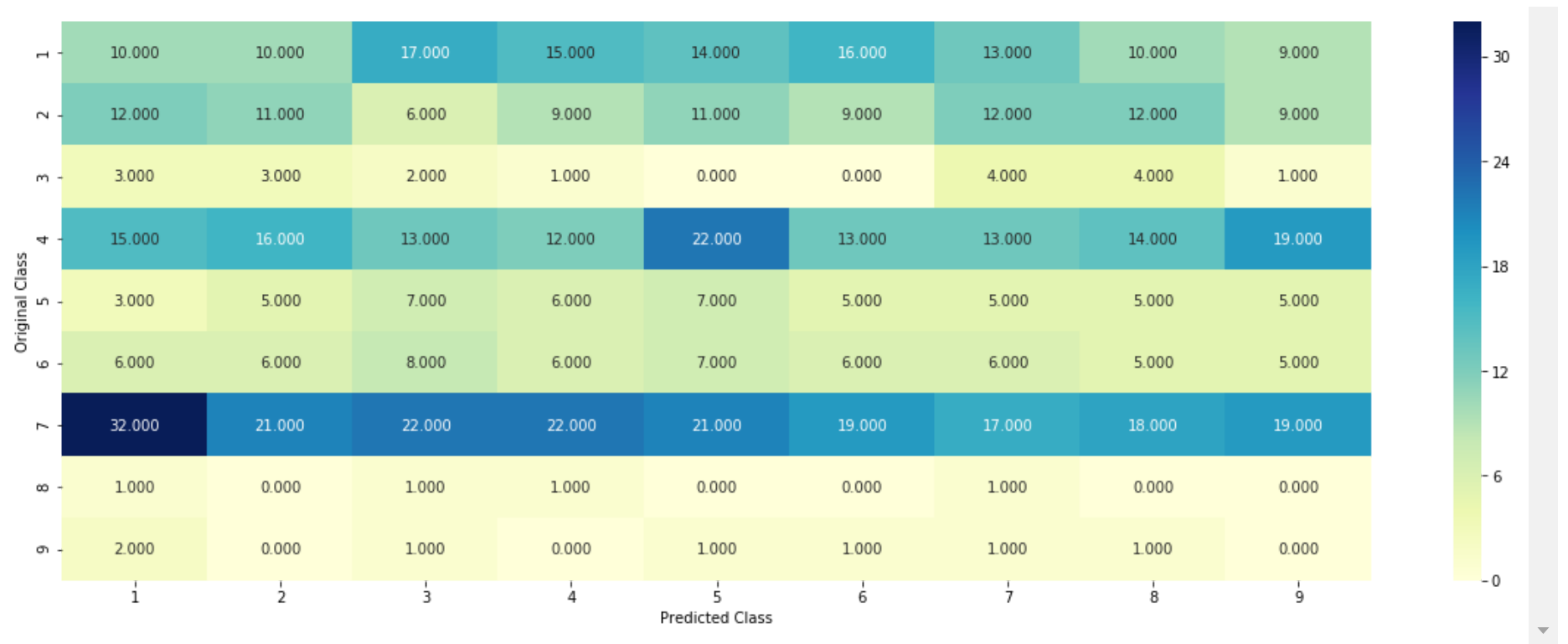
predicted_y =np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y+1)

```

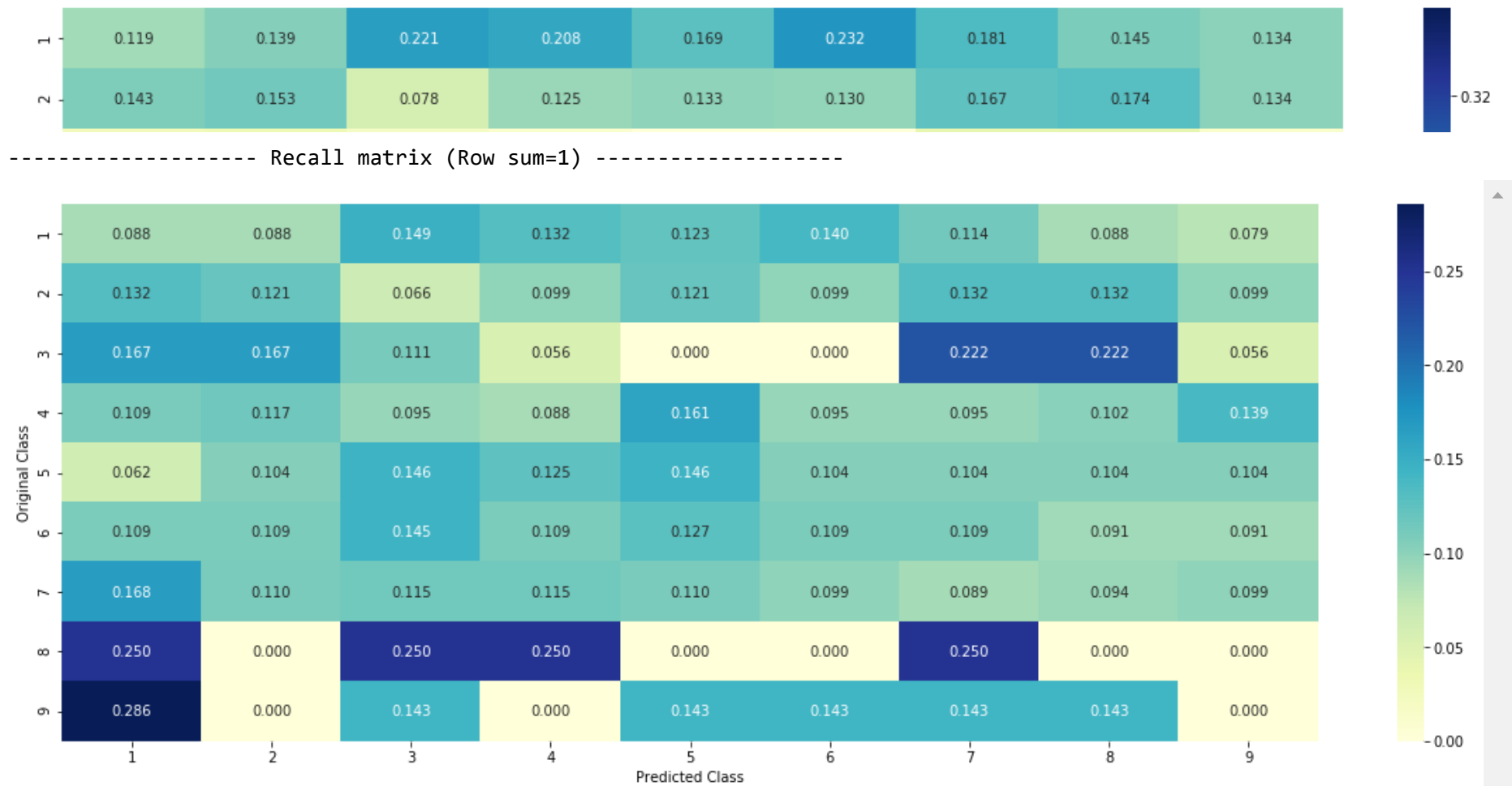
Log loss on Cross Validation Data using Random Model 2.5033153426703136

Log loss on Test Data using Random Model 2.540669705320987

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



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



### 3.3 Univariate Analysis

```

In [16]: # 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 occurrences of given feature in train data dataframe
# build a vector (1*9) , the first element = (number of times it occurred in class1 + 10*alpha / number of times it occurred)
# gv_dict is like a look up table, for every gene it stores 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] to 'gv_fea'
# return 'gv_fea'
# -----

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

```

```

# Q61L                                3
# S222D                                2
# P130S                                2
# ...
# }
value_count = train_df[feature].value_counts()

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

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

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

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    # {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177, 0.13636363636363635, 0.25, 0.193181
    # 'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.27040816326530615, 0.061224489795
    # 'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177, 0.068181818181818177, 0.062

```

```

#      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.078787878787878782, 0.1393939393
#      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.46540880503144655, 0.07547169811
#      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.072847682119205295, 0.066225165562
#      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.07333333333333334, 0.09333333333
#      ...
#      }
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 will add t
# 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 calculate the probability of a feature belongs to any particular class, we apply laplace smoothing

- $(\text{numerator} + 10 \cdot \alpha) / (\text{denominator} + 90 \cdot \alpha)$

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

Number of Unique Genes : 231

BRCA1 169

EGFR 101

TP53 99

PTEN 86

BRCA2 80

KIT 65

BRAF 56

ALK 42

ERBB2 39

FGFR2 39

Name: Gene, dtype: int64

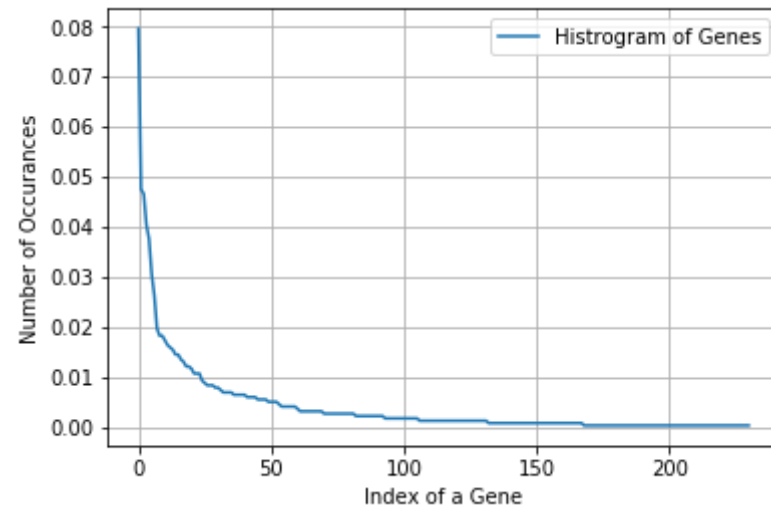
```
In [18]: print("Ans: There are", unique_genes.shape[0] , "different categories of genes in the train data, and they are distributed
```



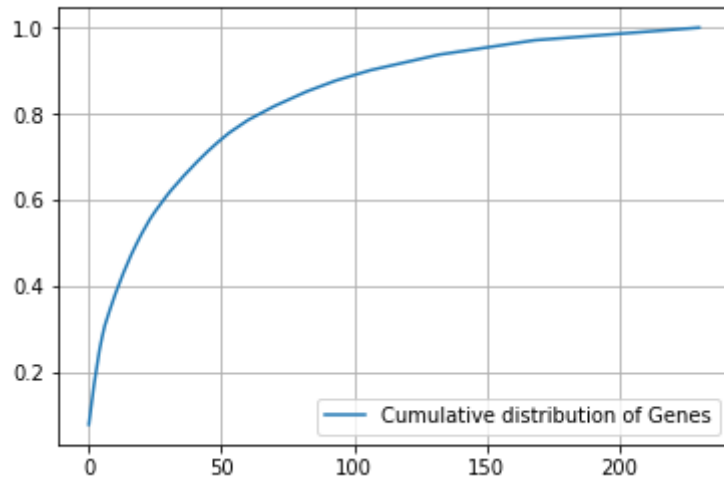
Ans: There are 231 different categories of genes in the train data, and they are distributed as follows



```
In [19]: s = sum(unique_genes.values);  
h = unique_genes.values/s;  
plt.plot(h, label="Histogram of Genes")  
plt.xlabel('Index of a Gene')  
plt.ylabel('Number of Occurances')  
plt.legend()  
plt.grid()  
plt.show()
```



```
In [20]: 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 [21]: #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 [22]: print("train_gene_feature_responseCoding is converted feature using response coding method. The shape of gene feature:",
train_gene_feature_responseCoding is converted feature using response coding method. The shape of gene feature: (2124,
9)
```

```
In [23]: # one-hot encoding of Gene feature.
from sklearn.feature_extraction.text import TfidfVectorizer
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

```
In [24]: train_df['Gene'].head()
```

```
Out[24]: 205      EGFR
908      PDGFRA
3069     NOTCH1
694      CDKN2B
502      TP53
Name: Gene, dtype: object
```

```
In [25]: gene_vectorizer.get_feature_names()
```

```
Out[25]: ['abl1',
          'acvr1',
          'ago2',
          'akt1',
          'akt2',
          'akt3',
          'alk',
          'apc',
          'ar',
          'araf',
          'arid1b',
          'arid2',
          'arid5b',
          'asxl1',
          'atm',
          'atrx',
          'aurka',
          'axl',
          'b2m',
          'b2m1']
```

```
In [26]: print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature:",
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature: (2124, 231)
```

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

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

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

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

cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_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-15))

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

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

```

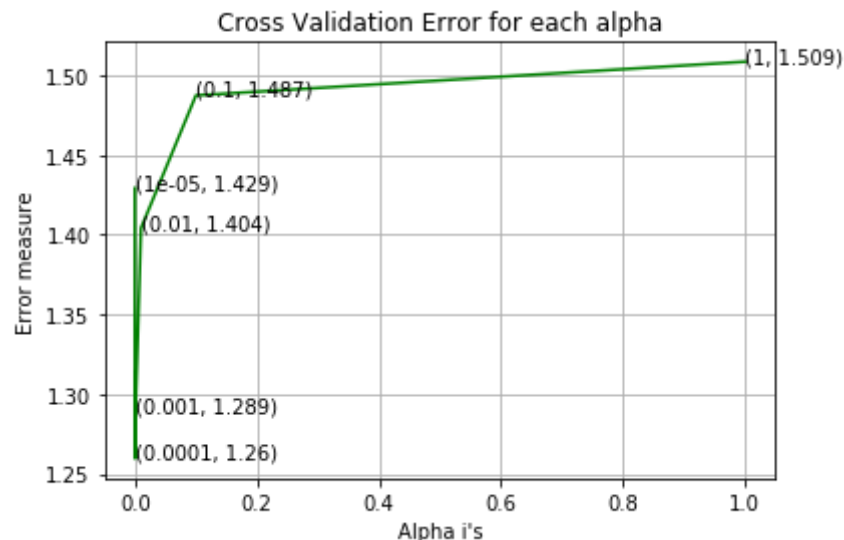
```

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

predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.c
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, predict_y, la
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, labels=clf.c

```

For values of alpha = 1e-05 The log loss is: 1.429228899186377  
 For values of alpha = 0.0001 The log loss is: 1.2597402994982916  
 For values of alpha = 0.001 The log loss is: 1.2885303197883522  
 For values of alpha = 0.01 The log loss is: 1.404154679357312  
 For values of alpha = 0.1 The log loss is: 1.487374202847152  
 For values of alpha = 1 The log loss is: 1.5085111816306274



For values of best alpha = 0.0001 The train log loss is: 1.0444280458539423  
For values of best alpha = 0.0001 The cross validation log loss is: 1.2597402994982916  
For values of best alpha = 0.0001 The test log loss is: 1.195465777441181

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

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 231 genes in train dataset?

Ans

1. In test data 645 out of 665 : 96.99248120300751
2. In cross validation data 515 out of 532 : 96.80451127819549

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

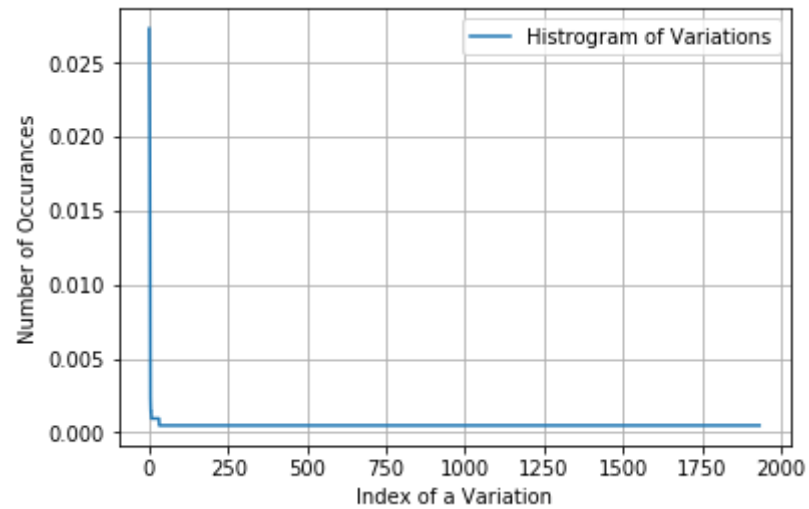
```
Number of Unique Variations : 1933
Truncating_Mutations      58
Deletion                  48
Amplification             39
Fusions                   17
Overexpression            5
Q61H                      3
G12V                      3
A146T                     2
G12S                      2
Y64A                      2
Name: Variation, dtype: int64
```

```
In [30]: print("Ans: There are", unique_variations.shape[0] , "different categories of variations in the train data, and they are
```

Ans: There are 1933 different categories of variations in the train data, and they are distributed as follows

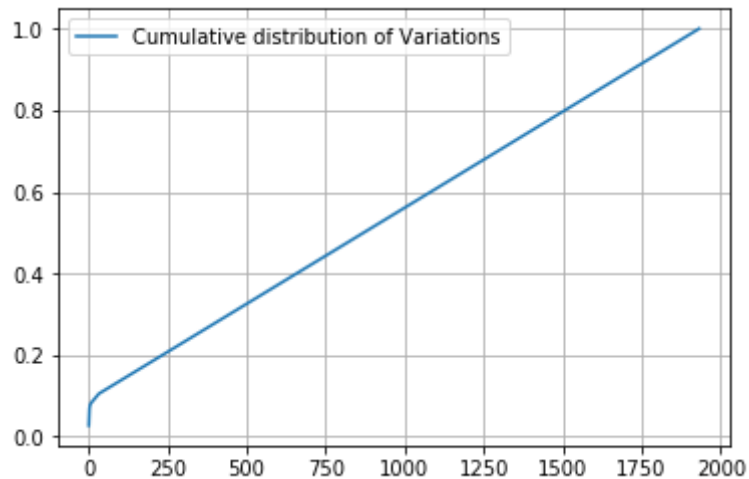


```
In [31]: s = sum(unique_variations.values);  
h = unique_variations.values/s;  
plt.plot(h, label="Histogram of Variations")  
plt.xlabel('Index of a Variation')  
plt.ylabel('Number of Occurances')  
plt.legend()  
plt.grid()  
plt.show()
```



```
In [32]: c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()

[0.02730697 0.04990584 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 [33]: # 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 [34]: print("train_variation_feature_responseCoding is a converted feature using the response coding method. The shape of Vari
```

train\_variation\_feature\_responseCoding is a converted feature using the response coding method. The shape of Variation feature: (2124, 9)

```
In [35]: # 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 [36]: print("train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Varia
```

train\_variation\_feature\_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Variation feature: (2124, 1964)

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

Let's build a model just like the earlier!

```
In [37]: alpha = [10 ** x for x in range(-5, 1)]
```

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

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

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

cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_variation_feature_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-15))

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

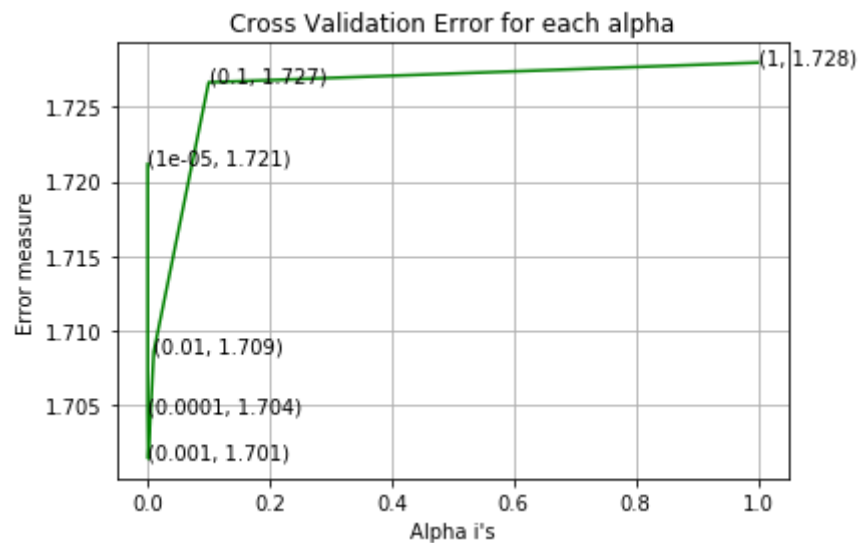
```

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

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

```

For values of alpha = 1e-05 The log loss is: 1.7211525515730757  
 For values of alpha = 0.0001 The log loss is: 1.7044985632354883  
 For values of alpha = 0.001 The log loss is: 1.7014228937147464  
 For values of alpha = 0.01 The log loss is: 1.708586770029349  
 For values of alpha = 0.1 The log loss is: 1.7266369180037033  
 For values of alpha = 1 The log loss is: 1.7279579451890903



For values of best alpha = 0.001 The train log loss is: 1.0714262196625013  
For values of best alpha = 0.001 The cross validation log loss is: 1.7014228937147464  
For values of best alpha = 0.001 The test log loss is: 1.6934979089250732

**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 [38]: print("Q12. How many data points are covered by total ", unique_variations.shape[0], " genes in test and cross validation")
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 1933 genes in test and cross validation data sets?

Ans

1. In test data 70 out of 665 : 10.526315789473683
2. In cross validation data 59 out of 532 : 11.090225563909774

### 3.2.3 Univariate Analysis on Text Feature

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

```
In [39]: # 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 [40]: 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 [41]: # building a CountVectorizer with all the words that occurred minimum 3 times in train data
# Reference - https://buhrmann.github.io/tfidf-analysis.html
text_vectorizer = TfidfVectorizer(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

final_fts = train_text_feature_onehotCoding.toarray().sum(axis=0)
top_1k_fts = np.argsort(final_fts)[::-1][0:1000]

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it occurred
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 : 52855



```

In [42]: dict_list = []
# dict_list =[] contains 9 dictionaries each corresponds to a class
for i in range(1,10):
    cls_text = train_df[train_df['Class']==i]
    # build a word dict based on the words in that class
    dict_list.append(extract_dictionary_paddle(cls_text))
    # append it to dict_list

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

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

```

```

In [43]: #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 [44]: # https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.T/train_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_responseCoding.sum(axis=1)).T

```

```
In [45]: # don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding[:,top_1k_fts], 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[:,top_1k_fts], 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[:,top_1k_fts], axis=0)
```

```
In [46]: print( train_text_feature_onehotCoding.shape)
print(test_text_feature_onehotCoding.shape)
print(cv_text_feature_onehotCoding.shape)
```

```
(2124, 1000)
(665, 1000)
(532, 1000)
```

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

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

```
Counter({0.017934441704394233: 313, 0.03431923769297222: 236, 0.011277558733376377: 231, 0.021309456155699007: 223,  
0.0841860290653762: 205, 0.3330221744812404: 154, 0.21510343232332796: 130, 0.014686548022060701: 127, 0.064152006164  
95627: 124, 0.017812897527400954: 116, 0.041400945757954326: 115, 0.06128211560731485: 110, 0.04418970107945162: 108,  
0.10509540770516547: 105, 0.06274877899729857: 102, 0.024487265016113324: 99, 0.032221395984043125: 98, 0.03483613619  
1600996: 93, 0.12950399536473373: 88, 0.025804193053244665: 88, 0.024117879426990736: 85, 0.015961806558021575: 85,  
0.03381623805999512: 82, 0.03215890625802947: 79, 0.02036686040588048: 78, 0.01115145943432505: 77, 0.043573153047941  
245: 76, 0.016359378755454686: 73, 0.026152865605715482: 68, 0.05720855812992253: 67, 0.04469764785297529: 64, 0.0435  
74115196151765: 64, 0.022990101662539: 64, 0.021949053656975435: 63, 0.15160712160314466: 62, 0.06614444729389045: 6  
0, 0.023910720062271067: 58, 0.01552513137008564: 58, 0.05401913450010369: 57, 0.02780203761708643: 56, 0.01399416217  
5609483: 56, 0.052601197650032995: 55, 0.04481499037704162: 55, 0.03171720644353171: 55, 0.027340599777748106: 55, 0.  
024550098058064664: 54, 0.022555117466752754: 54, 0.021265368078569647: 54, 0.014629839776652552: 54, 0.2077386615768  
0368: 52, 0.01870657143726904: 52, 0.015052592829190923: 52, 0.08446940365284634: 50, 0.027010650099231974: 50, 0.018  
165035483886347: 50, 0.01644586509562593: 50, 0.08603103451105196: 49, 0.01261264032088736: 49, 0.028692883835239793:  
48, 0.026748627077242824: 48, 0.022349781372801136: 48, 0.0089618106233534: 48, 0.07497441518051895: 47, 0.0328358359  
4839404: 47, 0.020512952104763258: 47, 0.14949390809041957: 46, 0.04850833632982919: 46, 0.03450100540935392: 46, 0.0  
1882456254654479: 46, 0.06288839907932976: 45, 0.04446368688952694: 45, 0.020867228858467087: 45, 0.05033884348217182  
4: 43, 0.04930146185047533: 43, 0.03235191530506621: 43, 0.05537090061264073: 41, 0.04690428725963047: 41, 0.01963306  
102242494: 41, 0.00993121284994268: 41, 0.030123166780268818: 40, 0.0265432290929526: 40, 0.015523038963677195: 39,  
0.19574012359113718: 38, 0.054113779635093846: 38, 0.04731678568050374: 38, 0.034058318109026944: 38, 0.0326976726845
```

```

In [49]: # Train a Logistic regression+Calibration model using text features which are on-hot encoded
alpha = [10 ** x for x in range(-5, 1)]

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

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

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

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

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

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

```

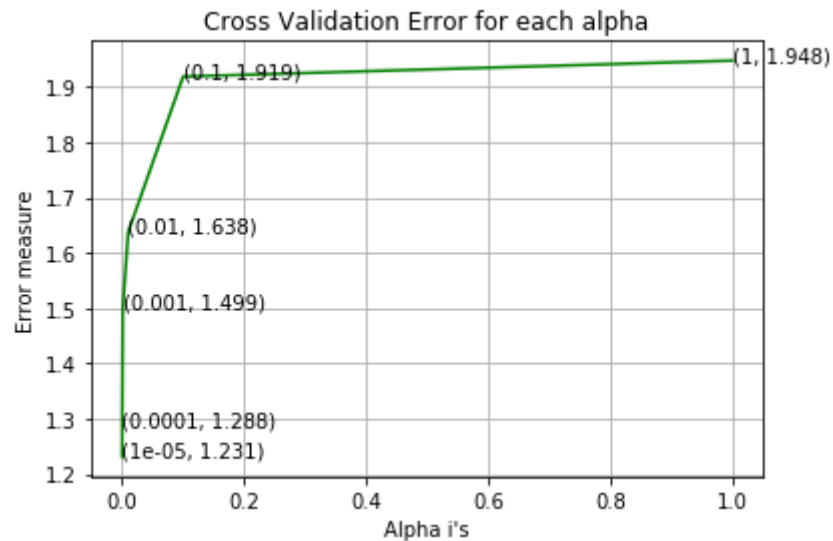
```

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

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(y_cv, predict_y, la
predict_y = sig_clf.predict_proba(test_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y, labels=clf.c

```

For values of alpha = 1e-05 The log loss is: 1.2312050568184227  
 For values of alpha = 0.0001 The log loss is: 1.287891737144318  
 For values of alpha = 0.001 The log loss is: 1.4986687471426732  
 For values of alpha = 0.01 The log loss is: 1.637970223360734  
 For values of alpha = 0.1 The log loss is: 1.9189685168583874  
 For values of alpha = 1 The log loss is: 1.9478998340502305



For values of best alpha = 1e-05 The train log loss is: 0.8052349094757469  
 For values of best alpha = 1e-05 The cross validation log loss is: 1.2312050568184227  
 For values of best alpha = 1e-05 The test log loss is: 1.0905987111534998

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

**Ans.** Yes, it seems like!

```
In [50]: def get_intersec_text(df):
          df_text_vec = CountVectorizer(min_df=3)
          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 [51]: len1, len2 = get_intersec_text(test_df)
          print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train data")
          len1, len2 = get_intersec_text(cv_df)
          print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in train data")
```

```
96.545 % of word of test data appeared in train data
98.08 % of word of Cross Validation appeared in train data
```

## 4. Machine Learning Models

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

*#Misc. functionns for ML models*

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

    # for calculating log_loss we willl provide the array of 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 [53]: **def** report\_log\_loss(train\_x, train\_y, test\_x, test\_y, clf):

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

```

In [54]: # 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 [{}]" .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 [{}]" .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 [{}]" .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 [55]: *# 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']))

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

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

```
In [56]: 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, 3195)
(number of data points * number of features) in test data = (665, 3195)
(number of data points * number of features) in cross validation data = (532, 3195)
```

```
In [57]: 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 [58]: # find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive\_bayes
# -----
# 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.applidaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algorithm-1/
# -----

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.Ca
# -----
# 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.applidaicourse.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 probabillites we use log-probability estimates
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

```

```

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

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

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

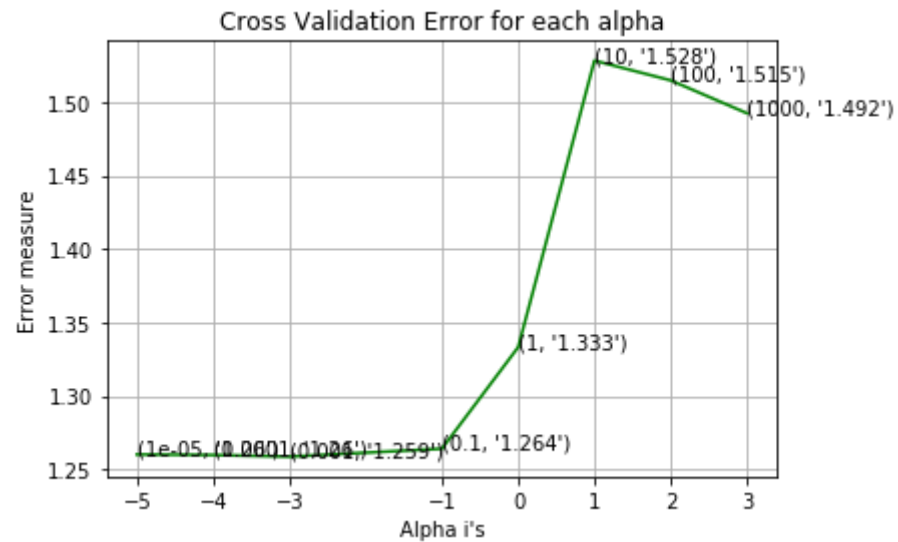
```

```

for alpha = 1e-05
Log Loss : 1.2602538209854242
for alpha = 0.0001
Log Loss : 1.259920383687711
for alpha = 0.001
Log Loss : 1.2587969584012375
for alpha = 0.1
Log Loss : 1.2641875870596706
for alpha = 1
Log Loss : 1.333203286285047
for alpha = 10
Log Loss : 1.5279693911229684
for alpha = 100

```

Log Loss : 1.5145462970878025  
for alpha = 1000  
Log Loss : 1.4922075790521818



For values of best alpha = 0.001 The train log loss is: 0.535997354839793  
For values of best alpha = 0.001 The cross validation log loss is: 1.2587969584012375  
For values of best alpha = 0.001 The test log loss is: 1.146580618792004

#### 4.1.1.2. Testing the model with best hyper paramters

```

In [59]: # find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive\_bayes
# -----
# 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.applidaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algorithm-1/
# -----

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.Ca
# -----
# 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
# -----

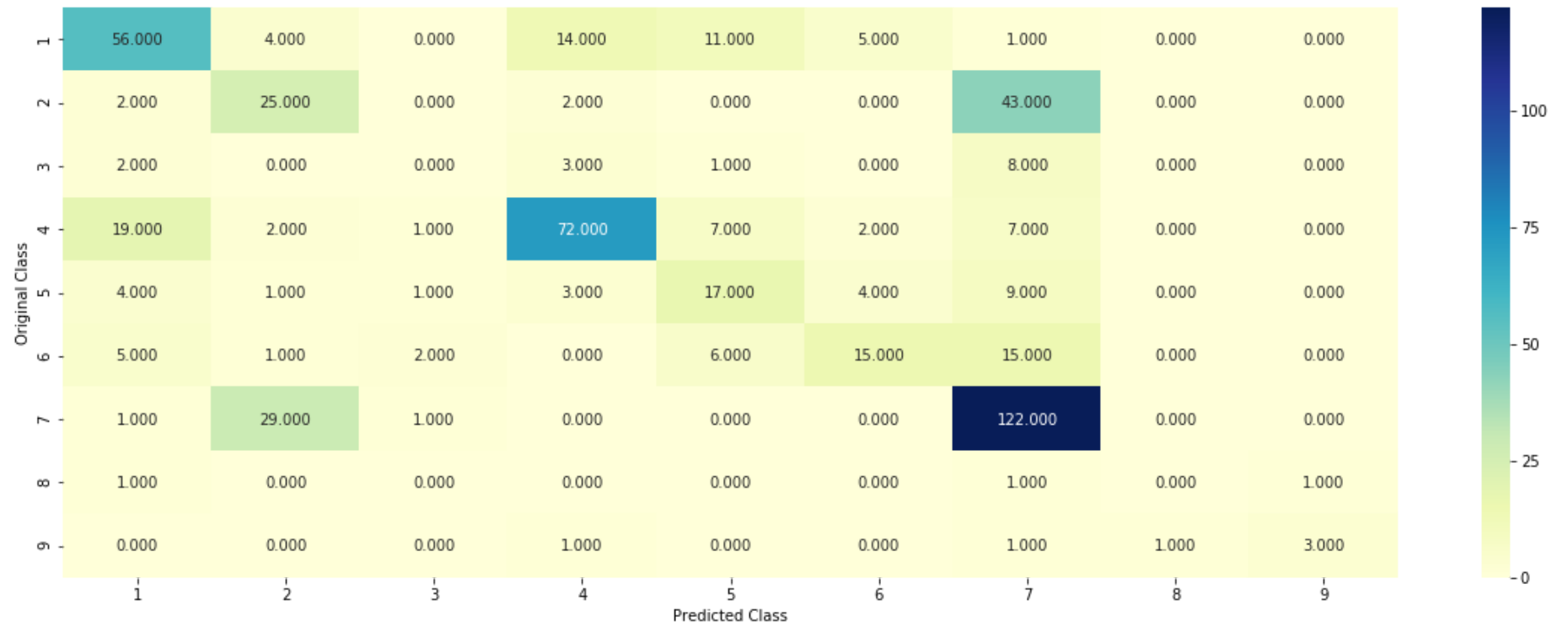
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabillites we 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.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))

```

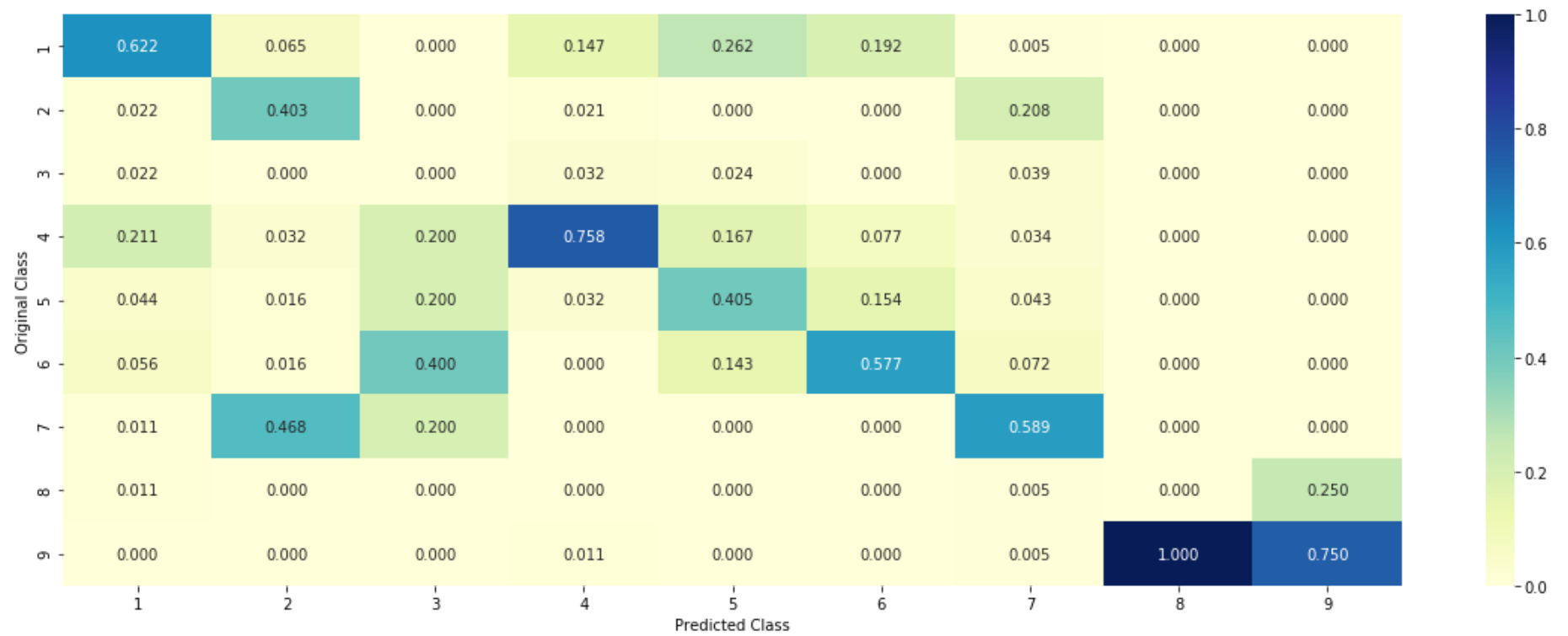
Log Loss : 1.2587969584012375

Number of missclassified point : 0.41729323308270677

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



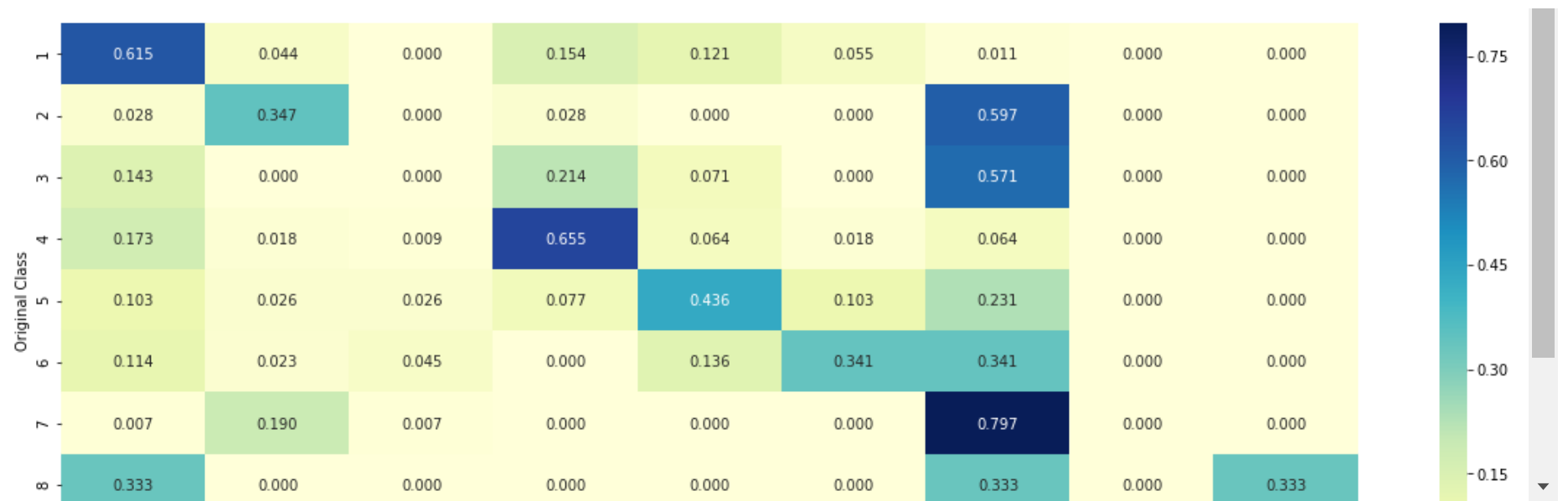
----- Precision matrix (Column Sum=1) -----



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







#### 4.1.1.3. Feature Importance, Correctly classified point

```
In [60]: 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(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],test_df['
```

Predicted Class : 6

Predicted Class Probabilities: [[0.0559 0.0392 0.0101 0.0625 0.031 0.7082 0.0868 0.0029 0.0033]]

Actual Class : 6

```
-----
37 Text feature [08] present in test data point [True]
56 Text feature [013] present in test data point [True]
62 Text feature [000] present in test data point [True]
93 Text feature [104brca2] present in test data point [True]
94 Text feature [111] present in test data point [True]
Out of the top 100 features 5 are present in query point
```

#### 4.1.1.4. Feature Importance, Incorrectly classified point

```
In [61]: test_point_index = 201
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(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],test_df['
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0755 0.1005 0.0133 0.0829 0.1504 0.454  0.1149 0.0039 0.0045]]
Actual Class : 1
-----
56 Text feature [013] present in test data point [True]
94 Text feature [111] present in test data point [True]
Out of the top 100 features 2 are present in query point
```

## 4.2. K Nearest Neighbour Classification

### 4.2.1. Hyper parameter tuning

```

In [62]: # find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier
# -----
# 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-geometric-intu
#-----

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV
# -----
# 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 probabilitites we use log-probability estimates

```

```

    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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

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

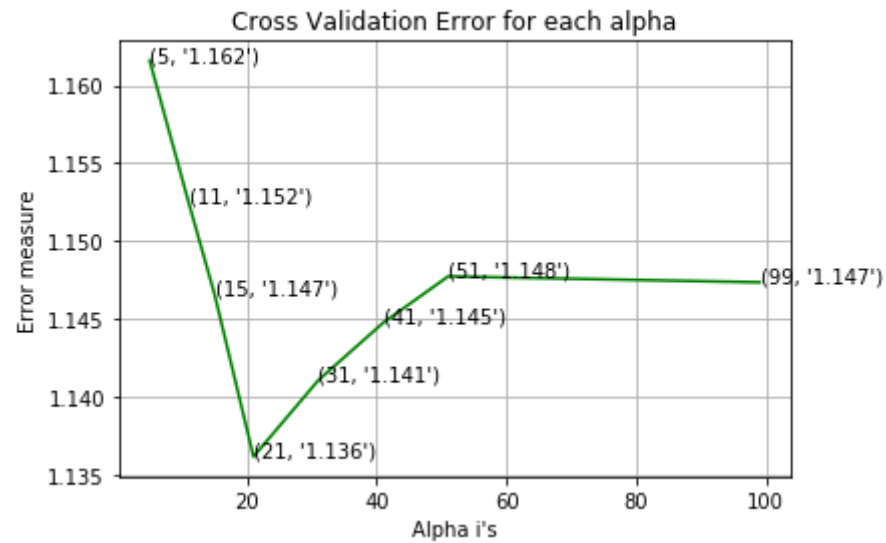
```

```

for alpha = 5
Log Loss : 1.1615810560746531
for alpha = 11
Log Loss : 1.1524895332763279
for alpha = 15
Log Loss : 1.1465831664444128
for alpha = 21
Log Loss : 1.1361695729523122
for alpha = 31
Log Loss : 1.14107467847058
for alpha = 41
Log Loss : 1.1447647885569086
for alpha = 51
Log Loss : 1.1477360662180307

```

for alpha = 99  
Log Loss : 1.1473573847031386



For values of best alpha = 21 The train log loss is: 0.7495357428161341  
For values of best alpha = 21 The cross validation log loss is: 1.1361695729523122  
For values of best alpha = 21 The test log loss is: 1.028719811612548

#### **4.2.2. Testing the model with best hyper paramters**

```
In [63]: # find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier
# -----
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

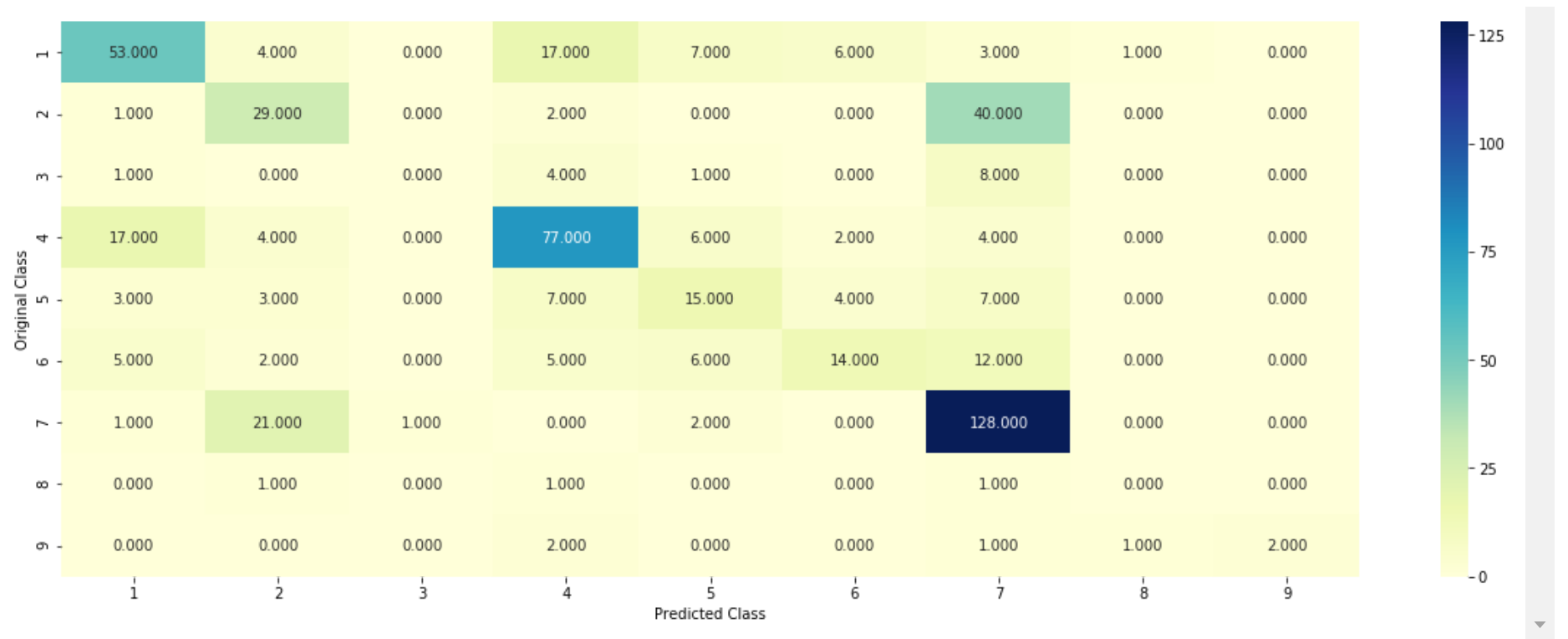
# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-neighbors-geometric-intu
#-----
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 : 1.1361695729523122

Number of mis-classified points : 0.40225563909774437

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





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



### 4.2.3. Sample Query point -1

```
In [74]: 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 = 200
predicted_cls = sig_clf.predict(test_x_responseCoding[0].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), alpha[best_alpha])
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to classes",train_y[neighbors[1][0]])
print("Frequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

Predicted Class : 1

Actual Class : 1

The 21 nearest neighbours of the test points belongs to classes [1 1 1 1 1 1 6 1 1 1 1 1 1 1 1 6 1 1 1 1]

Frequency of nearest points : Counter({1: 19, 6: 2})

### 4.2.4. Sample Query Point-2

```
In [75]: 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 classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

Predicted Class : 1

Actual Class : 4

the k value for knn is 21 and the nearest neighbours of the test points belongs to classes [1 4 1 1 1 4 1 1 1 1 1 1 4 1 1 1 1 1 4 4]

Fequency of nearest points : Counter({1: 16, 4: 5})

## 4.3. Logistic Regression

### 4.3.1. With Class balancing

#### 4.3.1.1. Hyper paramter tuning

In [76]:

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

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

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

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.Ca
# -----
# 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 probabilities we use log-probability estimates
print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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

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

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

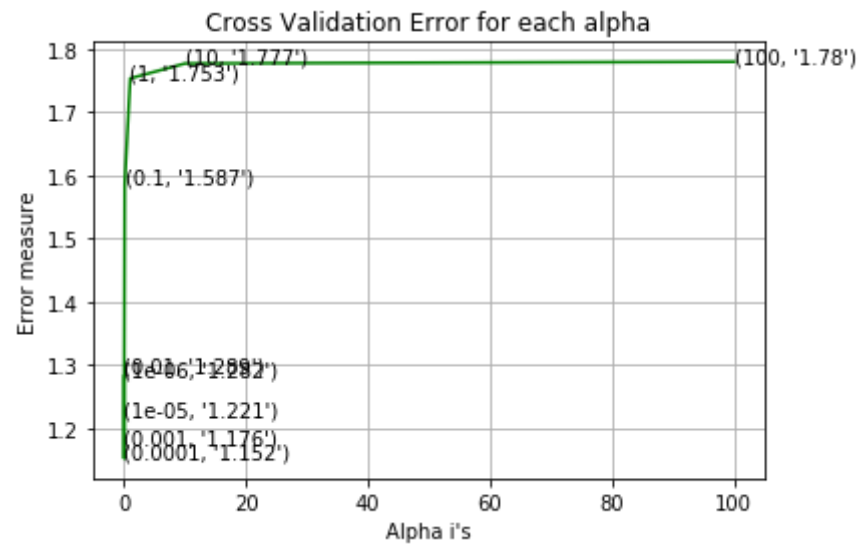
```

```

for alpha = 1e-06
Log Loss : 1.2820795654430093
for alpha = 1e-05
Log Loss : 1.2207100064099214
for alpha = 0.0001
Log Loss : 1.1520352125506361
for alpha = 0.001
Log Loss : 1.1756087391832577
for alpha = 0.01
Log Loss : 1.2886861334650623
for alpha = 0.1
Log Loss : 1.5866921789363262
for alpha = 1
Log Loss : 1.7534743668053976

```

```
for alpha = 10
Log Loss : 1.777078295217653
for alpha = 100
Log Loss : 1.7798465959611116
```



```
For values of best alpha = 0.0001 The train log loss is: 0.4379416661335392
For values of best alpha = 0.0001 The cross validation log loss is: 1.1520352125506361
For values of best alpha = 0.0001 The test log loss is: 0.9954243573188598
```

#### 4.3.1.2. Testing the model with best hyper paramters

```
In [77]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

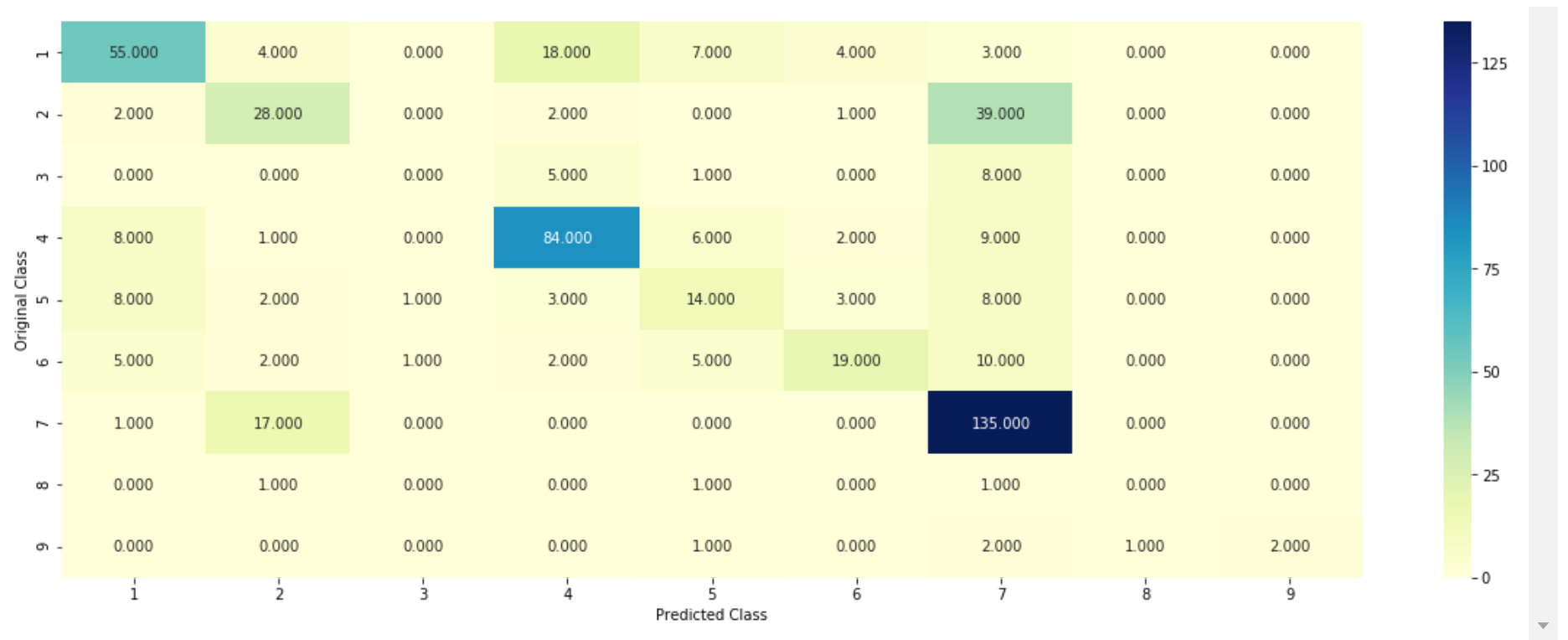
#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1/
#-----
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,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

Log loss : 1.1520352125506361

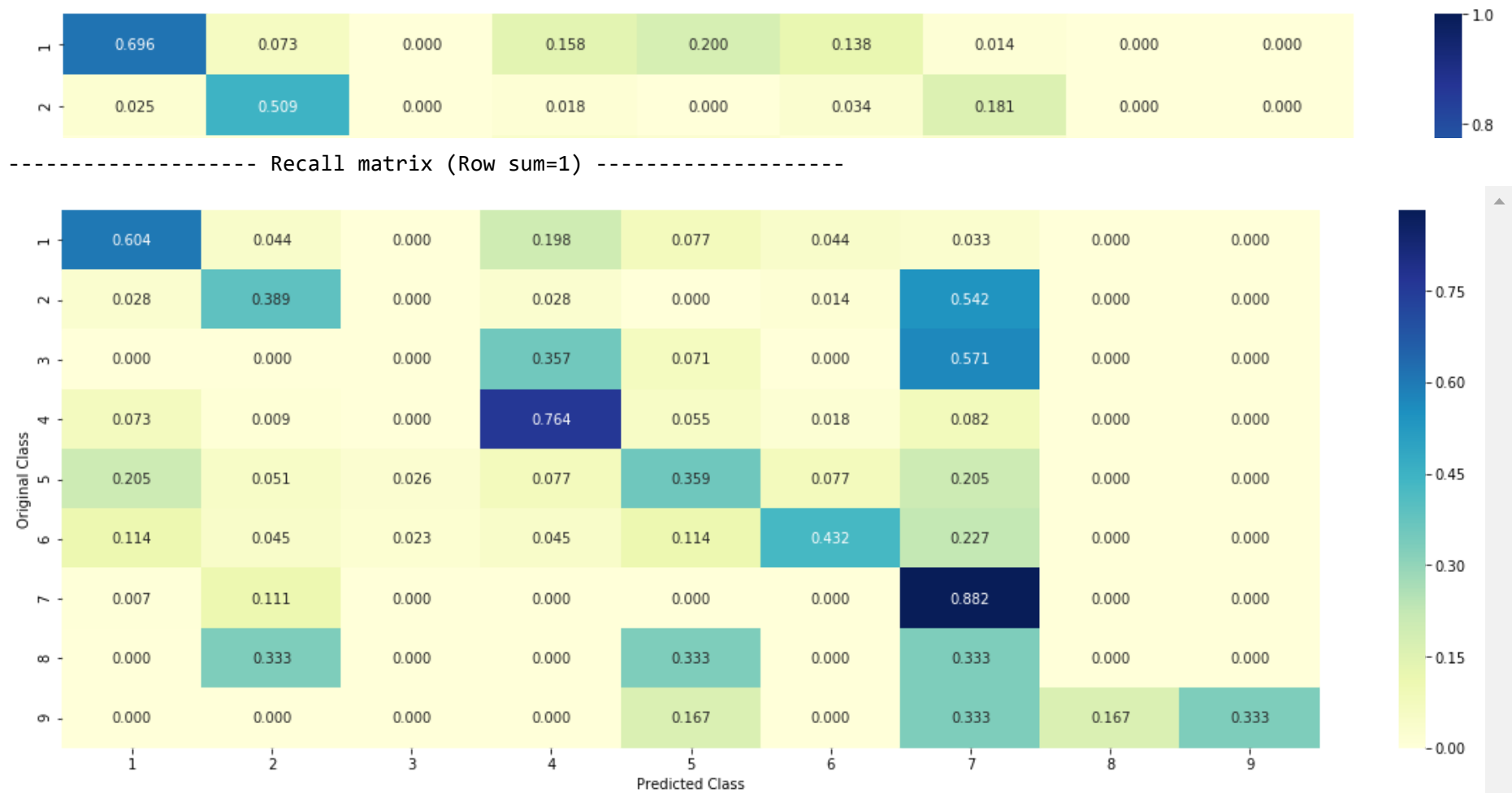
Number of mis-classified points : 0.36654135338345867

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





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



#### 4.3.1.3. Feature Importance

```

In [78]: 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 important features are present in our query point")
print("-"*50)
print("The features that are most important 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 [79]: # 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(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],test_df['
```

```
Predicted Class : 6
Predicted Class Probabilities: [[9.000e-03 5.600e-03 2.000e-04 3.400e-03 3.770e-02 9.432e-01 4.000e-04
 4.000e-04 1.000e-04]]
Actual Class : 6
-----
11 Text feature [049] present in test data point [True]
418 Text feature [001] present in test data point [True]
Out of the top 500 features 2 are present in query point
```

#### 4.3.1.3.2. Incorrectly Classified point

```
In [80]: 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(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],test_df['
```

```
Predicted Class : 1
Predicted Class Probabilities: [[5.812e-01 7.220e-02 5.000e-04 3.369e-01 2.300e-03 3.800e-03 1.400e-03
 1.500e-03 2.000e-04]]
Actual Class : 4
-----
19 Text feature [117] present in test data point [True]
153 Text feature [1087] present in test data point [True]
Out of the top 500 features 2 are present in query point
```

### 4.3.2. Without Class balancing

#### 4.3.2.1. Hyper paramter tuning

```

In [81]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.SGDClassifier
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

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

```

```

    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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

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

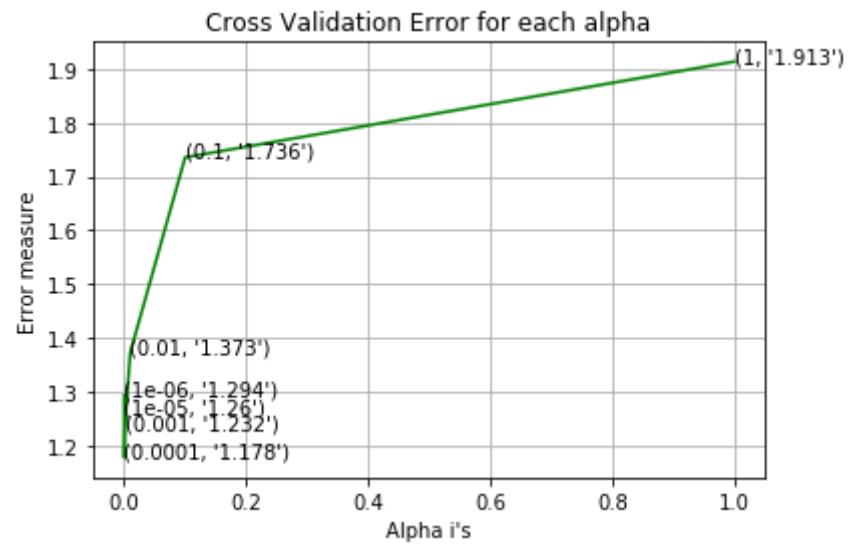
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The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

```

```

for alpha = 1e-06
Log Loss : 1.293690472637134
for alpha = 1e-05
Log Loss : 1.260347713730887
for alpha = 0.0001
Log Loss : 1.1779713526755606
for alpha = 0.001
Log Loss : 1.2322725633450609
for alpha = 0.01
Log Loss : 1.3731366566674625
for alpha = 0.1
Log Loss : 1.7356180965445045
for alpha = 1
Log Loss : 1.9134224113398473

```



For values of best alpha = 0.0001 The train log loss is: 0.4295583799747194

For values of best alpha = 0.0001 The cross validation log loss is: 1.1779713526755606

For values of best alpha = 0.0001 The test log loss is: 1.016897811677835

#### 4.3.2.2. Testing model with best hyper parameters



```
In [82]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

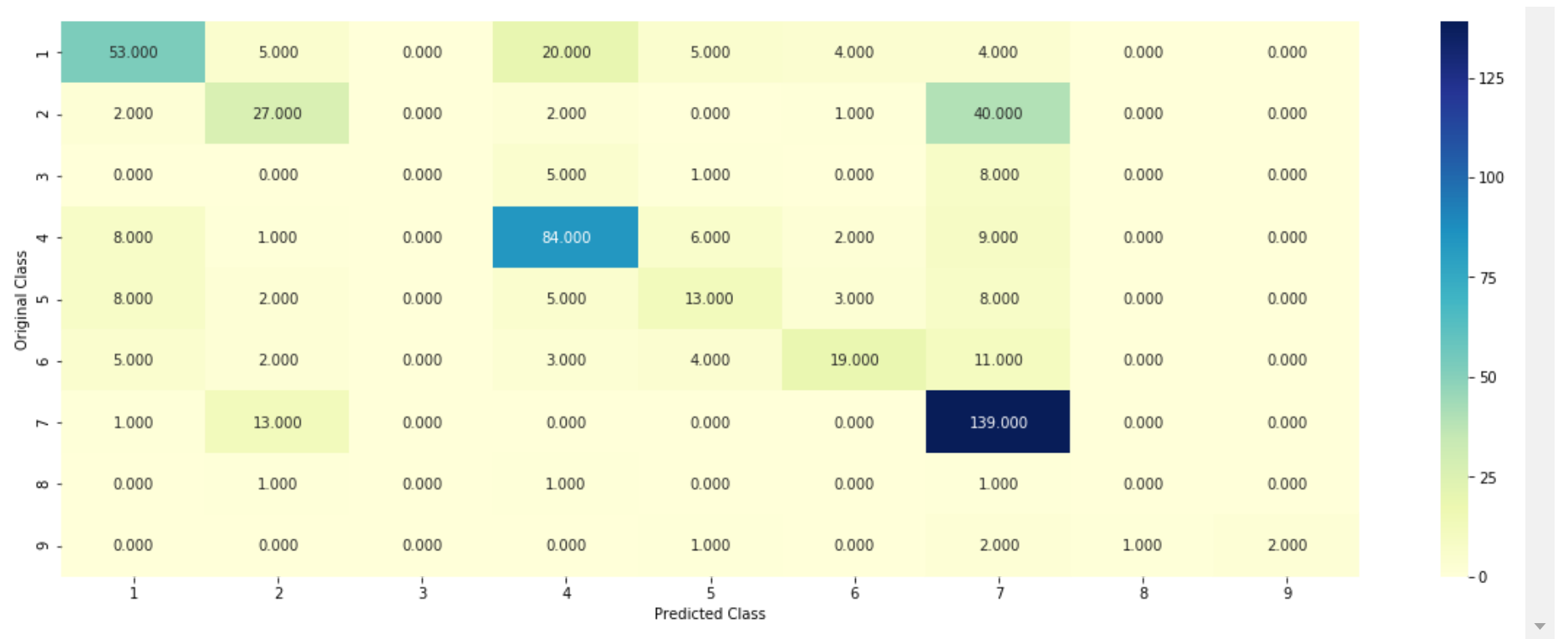
#-----
# video link:
#-----

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 : 1.1779713526755606

Number of mis-classified points : 0.36654135338345867

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



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



#### 4.3.2.3. Feature Importance, Correctly Classified point

```
In [83]: 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(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],test_df['
```

```
Predicted Class : 6
Predicted Class Probabilities: [[9.900e-03 6.100e-03 1.000e-04 4.200e-03 3.730e-02 9.415e-01 7.000e-04
 0.000e+00 0.000e+00]]
Actual Class : 6
-----
20 Text feature [001] present in test data point [True]
117 Text feature [049] present in test data point [True]
352 Text feature [08] 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 [84]: 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(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],test_df['
```

```
Predicted Class : 1
Predicted Class Probabilities: [[5.849e-01 6.770e-02 2.000e-04 3.394e-01 2.100e-03 3.100e-03 2.500e-03
 1.000e-04 0.000e+00]]
Actual Class : 4
-----
11 Text feature [117] present in test data point [True]
427 Text feature [1087] present in test data point [True]
Out of the top 500 features 2 are present in query point
```

## 4.4. Linear Support Vector Machines

### 4.4.1. Hyper paramter tuning

```

In [85]: # read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/generated/skle

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

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

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.Ca
# -----
# 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=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```

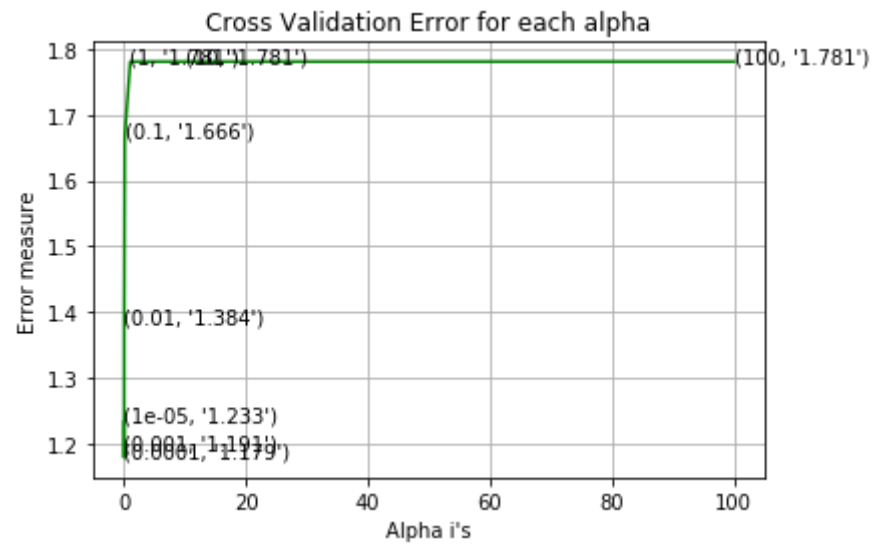
```

for C = 1e-05
Log Loss : 1.2332430422408749
for C = 0.0001
Log Loss : 1.178949205535011
for C = 0.001
Log Loss : 1.1907205174128939
for C = 0.01
Log Loss : 1.38372327382502
for C = 0.1
Log Loss : 1.66649482119244
for C = 1
Log Loss : 1.7805139670746548
for C = 10
Log Loss : 1.7806004756908558

```

for C = 100

Log Loss : 1.7805861473405644



For values of best alpha = 0.0001 The train log loss is: 0.4843472074024008

For values of best alpha = 0.0001 The cross validation log loss is: 1.178949205535011

For values of best alpha = 0.0001 The test log loss is: 1.053816098758451

#### 4.4.2. Testing model with best hyper parameters

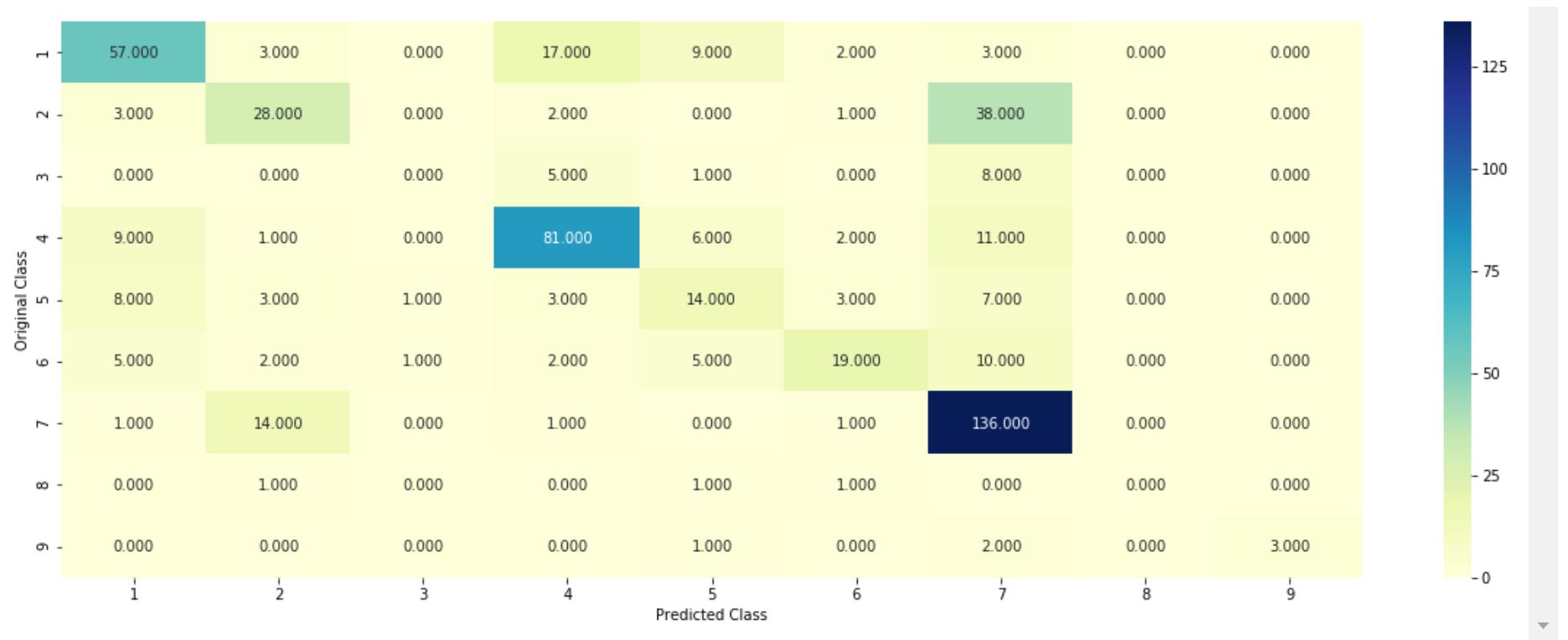


In [86]: *# read more about support vector machines with linear kernals here [```
# -----  
# default parameters  
# SVC\(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,  
# cache\_size=200, class\_weight=None, verbose=False, max\_iter=-1, decision\_function\_shape='ovr', random\_state=None\)  
  
# Some of methods of SVM\(\)  
# fit\(X, y, \[sample\_weight\]\)    Fit the SVM model according to the given training data.  
# predict\(X\)    Perform classification on samples in X.  
# -----  
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/mathematical-derivation-copy-8/  
# -----  
  
# clf = SVC\(C=alpha\[best\_alpha\],kernel='Linear',probability=True, class\_weight='balanced'\)  
clf = SGDClassifier\(alpha=alpha\[best\_alpha\], penalty='l2', loss='hinge', random\_state=42,class\_weight='balanced'\)  
predict\_and\_plot\_confusion\_matrix\(train\_x\_onehotCoding, train\_y,cv\_x\_onehotCoding,cv\_y, clf\)
```](http://scikit-learn.org/stable/modules/generated/skle</a></i></p></div><div data-bbox=)*

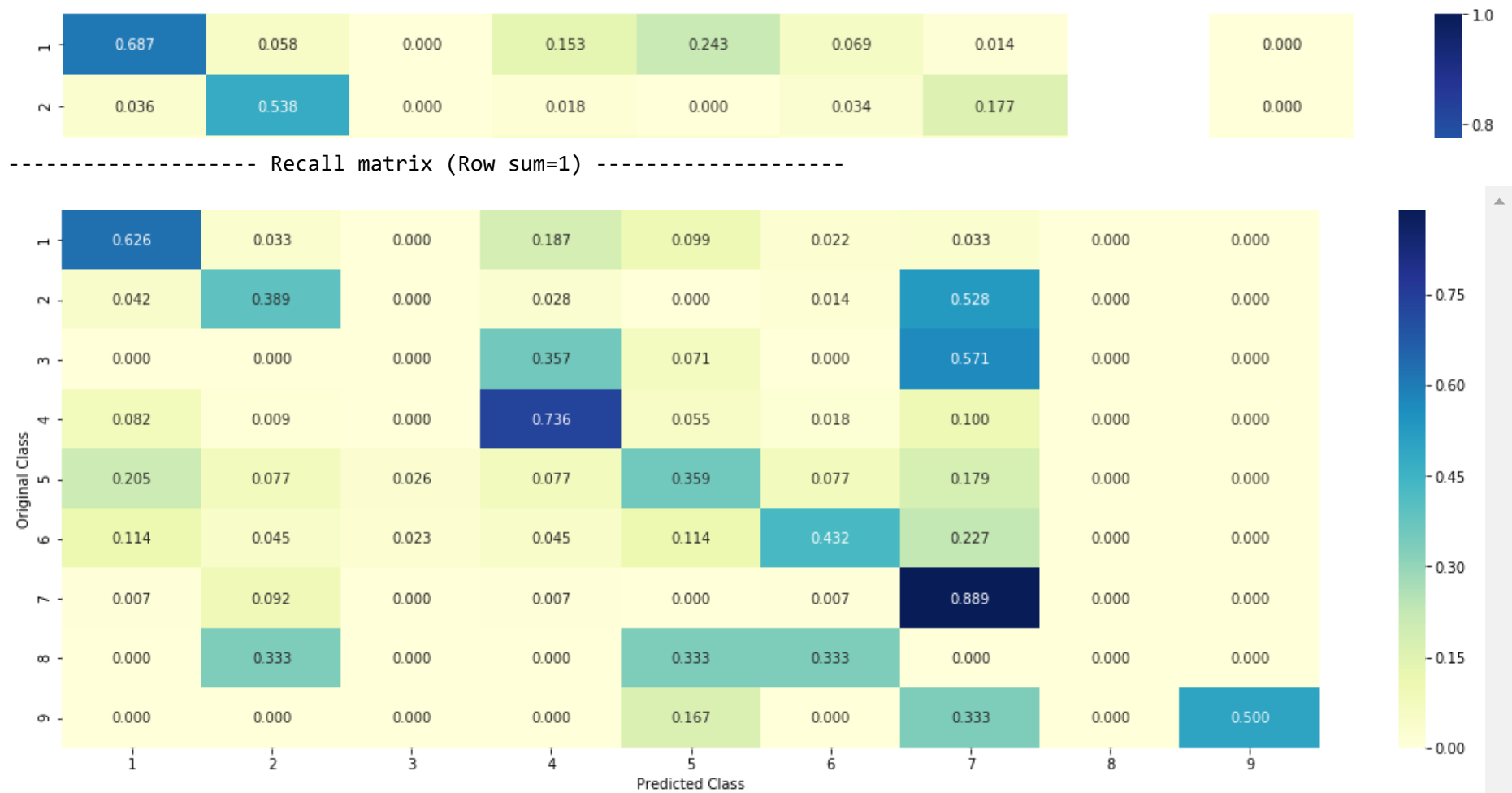
Log loss : 1.178949205535011

Number of mis-classified points : 0.36466165413533835

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



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



### 4.3.3. Feature Importance

#### 4.3.3.1. For Correctly classified point

```
In [87]: 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(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],test_df['
```

```
Predicted Class : 6
Predicted Class Probabilities: [[2.390e-02 5.540e-02 2.100e-03 4.420e-02 6.660e-02 7.986e-01 8.600e-03
 5.000e-04 1.000e-04]]
Actual Class : 6
-----
Out of the top 500 features 0 are present in query point
```

#### 4.3.3.2. For Incorrectly classified point

```
In [88]: 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(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],test_df['
```

```
Predicted Class : 1
Predicted Class Probabilities: [[6.882e-01 1.022e-01 1.100e-03 1.860e-01 1.140e-02 6.300e-03 3.800e-03
 6.000e-04 4.000e-04]]
Actual Class : 4
-----
Out of the top 500 features 0 are present in query point
```

## 4.5 Random Forest Classifier

### 4.5.1. Hyper paramter tuning (With One hot Encoding)

```

In [89]: # -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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

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

alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)

```

```

clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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

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

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

```

```

for n_estimators = 100 and max depth = 5
Log Loss : 1.2550943058706716
for n_estimators = 100 and max depth = 10
Log Loss : 1.2652870179708249
for n_estimators = 200 and max depth = 5
Log Loss : 1.249200406837101
for n_estimators = 200 and max depth = 10
Log Loss : 1.2592712924638831

```

```
for n_estimators = 500 and max depth = 5
Log Loss : 1.2482148129895727
for n_estimators = 500 and max depth = 10
Log Loss : 1.2553227549412975
for n_estimators = 1000 and max depth = 5
Log Loss : 1.248393302898599
for n_estimators = 1000 and max depth = 10
Log Loss : 1.2524934816011564
for n_estimators = 2000 and max depth = 5
Log Loss : 1.2491871099410945
for n_estimators = 2000 and max depth = 10
Log Loss : 1.2520829537059068
For values of best estimator = 500 The train log loss is: 0.8764422331679262
For values of best estimator = 500 The cross validation log loss is: 1.2482148129895727
For values of best estimator = 500 The test log loss is: 1.1677440324629056
```

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



```
In [90]: # -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

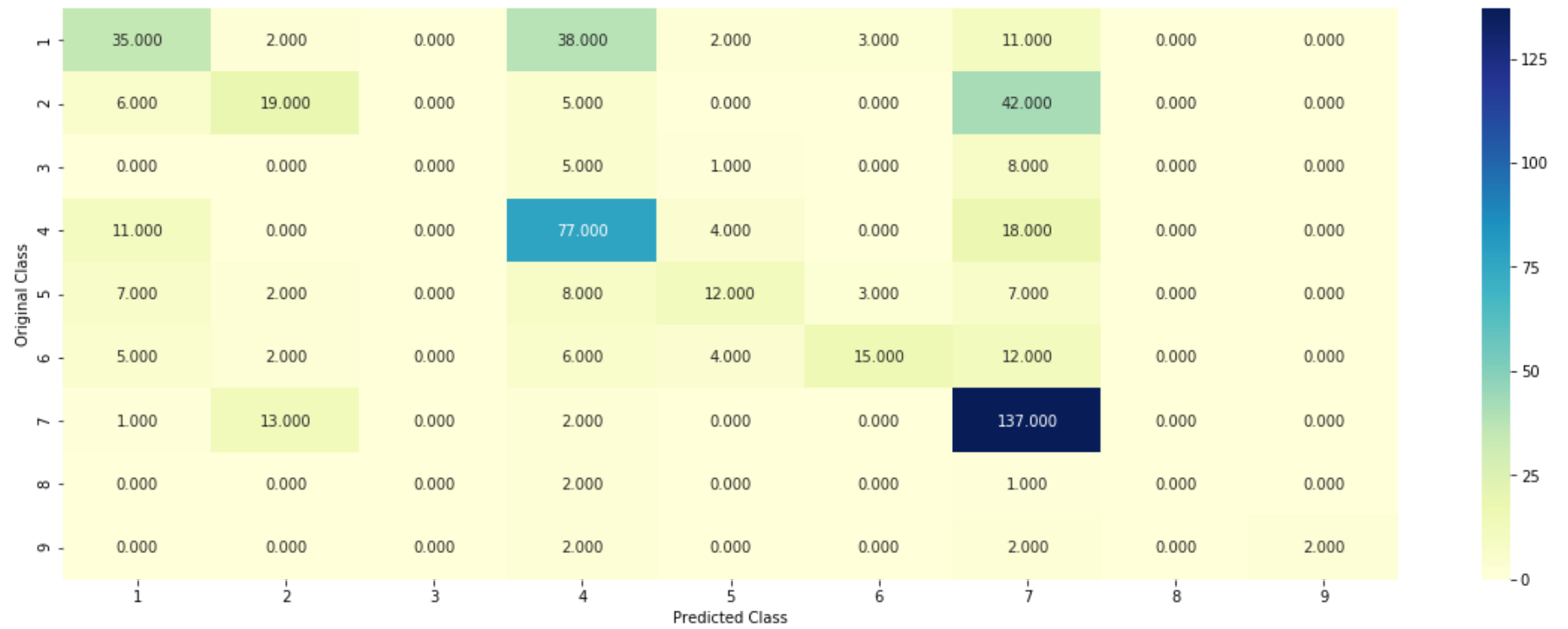
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-constructi
# -----

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

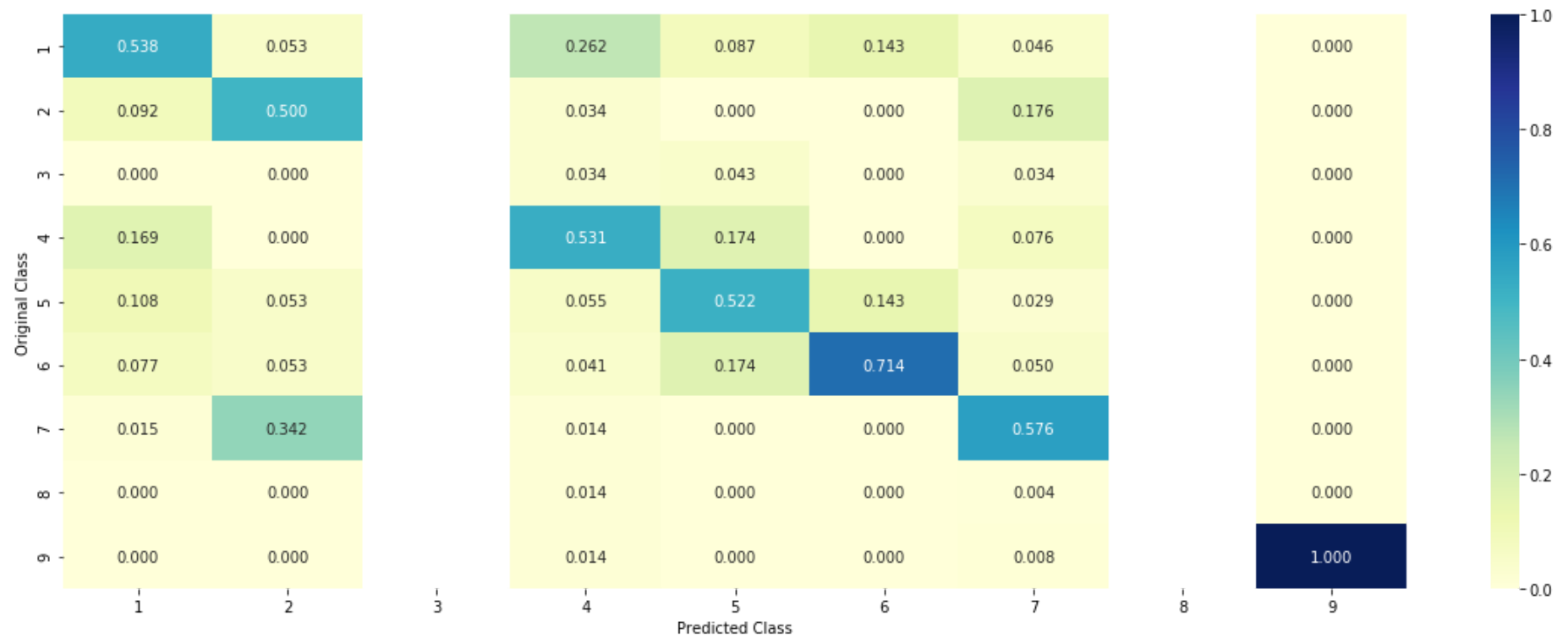
Log loss : 1.2482148129895727

Number of mis-classified points : 0.4417293233082707

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

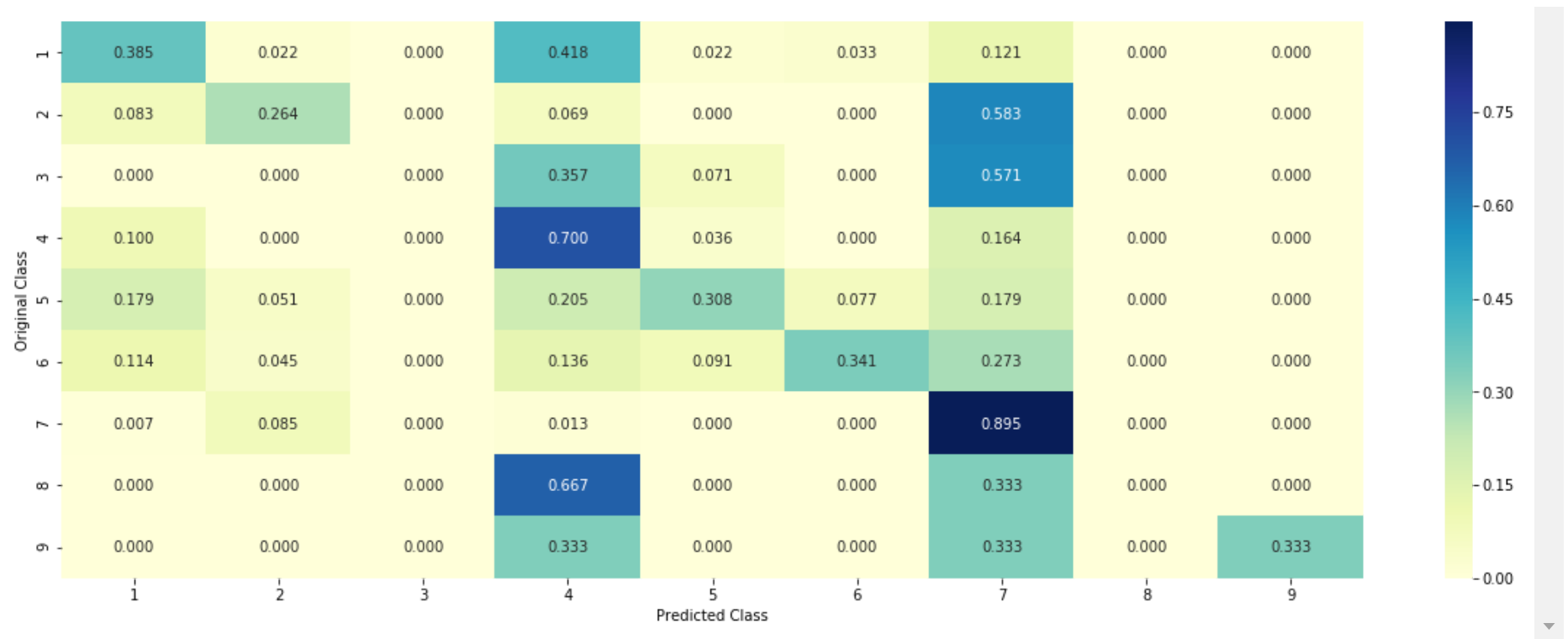


----- Precision matrix (Column Sum=1) -----



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





### 4.5.3. Feature Importance

#### 4.5.3.1. Correctly Classified point

```
In [91]: # test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_alpha
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

Predicted Class : 6

Predicted Class Probabilities: [[7.200e-03 4.100e-03 6.400e-03 5.700e-03 2.051e-01 7.632e-01 7.400e-03  
7.000e-04 2.000e-04]]

Actual Class : 6

```
-----
0 Text feature [0001] present in test data point [True]
1 Text feature [01] present in test data point [True]
12 Text feature [13] present in test data point [True]
30 Text feature [000] present in test data point [True]
66 Text feature [045] present in test data point [True]
95 Text feature [013] present in test data point [True]
Out of the top 100 features 6 are present in query point
```

#### 4.5.3.2. Inorrectly Classified point

```
In [92]: 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(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index])
```

Predicted Class : 1

Predicted Class Probabilities: [[0.3821 0.0905 0.0196 0.2457 0.0902 0.0929 0.0635 0.0085 0.0068]]

Actual Class : 4

-----  
0 Text feature [0001] present in test data point [True]

1 Text feature [01] present in test data point [True]

12 Text feature [13] present in test data point [True]

30 Text feature [000] present in test data point [True]

Out of the top 100 features 4 are present in query point

### 4.5.3. Hyper paramter tuning (With Response Coding)

```

In [93]: # -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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

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

alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)

```

```

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

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

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

```

```

for n_estimators = 10 and max depth = 2
Log Loss : 2.1396158620410772
for n_estimators = 10 and max depth = 3
Log Loss : 1.8253984221775816
for n_estimators = 10 and max depth = 5
Log Loss : 1.5932937108315584
for n_estimators = 10 and max depth = 10

```



```
Log Loss : 2.032961196490442
for n_estimators = 50 and max depth = 2
Log Loss : 1.8302465804253756
for n_estimators = 50 and max depth = 3
Log Loss : 1.5203722322484539
for n_estimators = 50 and max depth = 5
Log Loss : 1.3724465559765668
for n_estimators = 50 and max depth = 10
Log Loss : 1.7741076862080802
for n_estimators = 100 and max depth = 2
Log Loss : 1.6955573829864947
for n_estimators = 100 and max depth = 3
Log Loss : 1.5217293921762525
for n_estimators = 100 and max depth = 5
Log Loss : 1.3402852840550363
for n_estimators = 100 and max depth = 10
Log Loss : 1.7182186018638175
for n_estimators = 200 and max depth = 2
Log Loss : 1.6927941073979533
for n_estimators = 200 and max depth = 3
Log Loss : 1.5384308692905582
for n_estimators = 200 and max depth = 5
Log Loss : 1.4051315958754826
for n_estimators = 200 and max depth = 10
Log Loss : 1.6980856739016488
for n_estimators = 500 and max depth = 2
Log Loss : 1.747196585304519
for n_estimators = 500 and max depth = 3
Log Loss : 1.6116286845091206
for n_estimators = 500 and max depth = 5
Log Loss : 1.4188527723369866
for n_estimators = 500 and max depth = 10
Log Loss : 1.692914155032921
for n_estimators = 1000 and max depth = 2
Log Loss : 1.734259102836383
for n_estimators = 1000 and max depth = 3
Log Loss : 1.606952961052826
for n_estimators = 1000 and max depth = 5
Log Loss : 1.4315171523662278
for n_estimators = 1000 and max depth = 10
Log Loss : 1.6424980650145842
For values of best alpha = 100 The train log loss is: 0.056154254021325845
```

For values of best alpha = 100 The cross validation log loss is: 1.3402852840550366  
For values of best alpha = 100 The test log loss is: 1.2483371340759994

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

```

In [94]: # -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# class_weight=None)

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

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

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

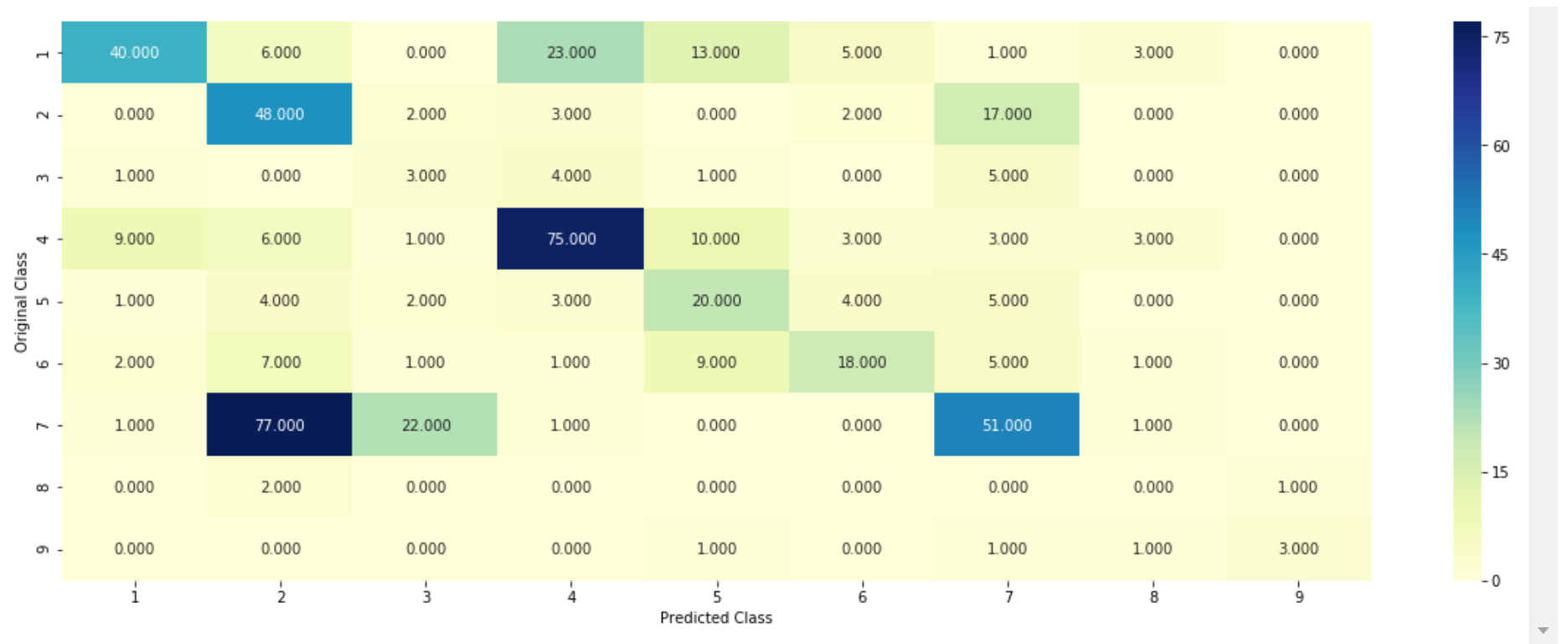
clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimators=alpha[int(best_alpha/4)], criterion='g
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCoding,cv_y, clf)

```

Log loss : 1.3402852840550366

Number of mis-classified points : 0.5150375939849624

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



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



## 4.5.5. Feature Importance

### 4.5.5.1. Correctly Classified point



```

In [95]: clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=max_depth[int(best_alpha/4)],
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].reshape(1,-1)).ravel(), 4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")

```

```

Predicted Class : 6
Predicted Class Probabilities: [[0.0102 0.0031 0.0129 0.0079 0.135  0.8169 0.0027 0.0053 0.006  ]]
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
Text is important feature
Text is important feature
Gene is important feature

```

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

#### **4.5.5.2. Incorrectly Classified point**



```

In [96]: test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index].reshape(1,-1)).ravel(), 5))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")

```

Predicted Class : 1

Predicted Class Probabilities: [[0.6685 0.002 0.0017 0.3165 0.001 0.0026 0.0019 0.0033 0.0025]]

Actual Class : 4

```

-----
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
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Variation is important feature

```

Text is important feature  
Text is important feature  
Text is important feature  
Gene is important feature  
Gene is important feature  
Gene is important feature

## **4.7 Stack the models**

### **4.7.1 testing with hyper parameter tuning**

```

In [97]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.SGDClassifier
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-----
# video link: https://www.applidaicourse.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/generated/sklearn.svm.LinearSVC
# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

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

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

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

```

```

# predict_proba (X) Perform classification on samples in X.

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

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

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_probabilities=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifier : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))))
    log_error = log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error

```

```
Logistic Regression : Log Loss: 1.18
Support vector machines : Log Loss: 1.78
Naive Bayes : Log Loss: 1.26
```

```
-----
Stacking Classifier : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifier : for the value of alpha: 0.001000 Log Loss: 2.032
Stacking Classifier : for the value of alpha: 0.010000 Log Loss: 1.514
Stacking Classifier : for the value of alpha: 0.100000 Log Loss: 1.226
Stacking Classifier : for the value of alpha: 1.000000 Log Loss: 1.487
Stacking Classifier : for the value of alpha: 10.000000 Log Loss: 1.970
```

#### **4.7.2 testing the model with the best hyper parameters**

```
In [98]: lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probabilities=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

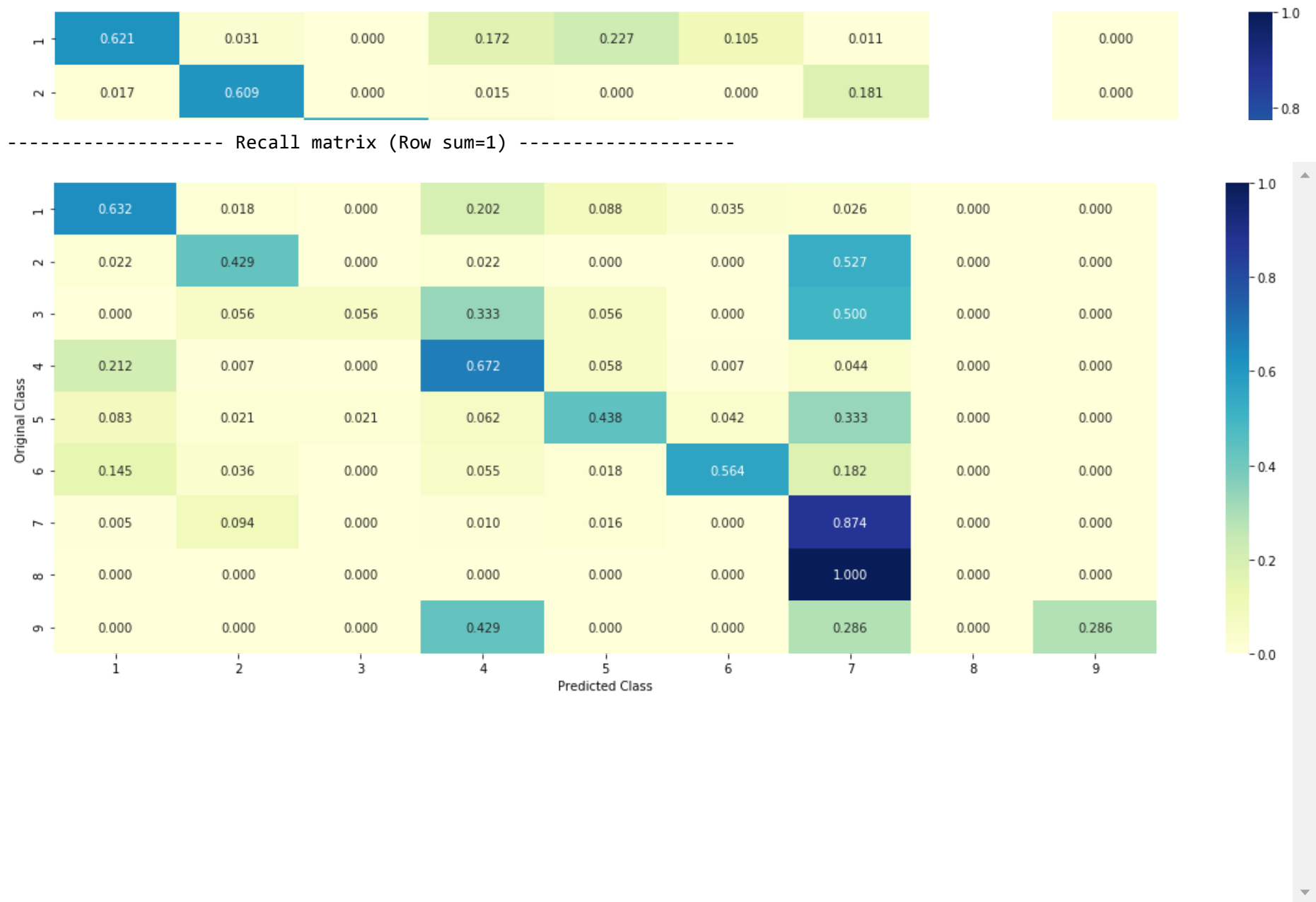
log_error = log_loss(test_y, sclf.predict_proba(test_x_onehotCoding))
print("Log loss (test) on the stacking classifier :",log_error)

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

```
Log loss (train) on the stacking classifier : 0.5446597903382286
Log loss (CV) on the stacking classifier : 1.2262893401386683
Log loss (test) on the stacking classifier : 1.1351665562235762
Number of missclassified point : 0.3609022556390977
----- Confusion matrix -----
```



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



#### 4.7.3 Maximum Voting classifier



```
In [99]: #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.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

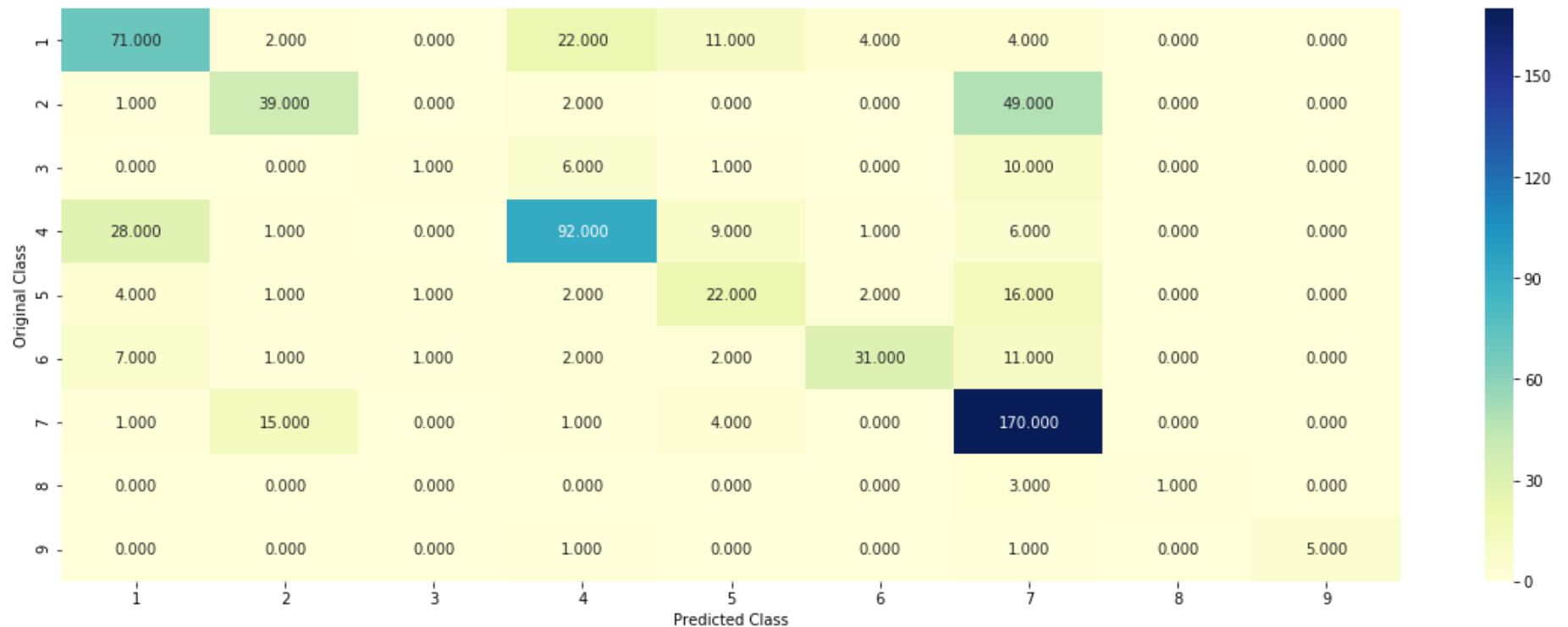
Log loss (train) on the VotingClassifier : 0.8226161251503965

Log loss (CV) on the VotingClassifier : 1.241163129516119

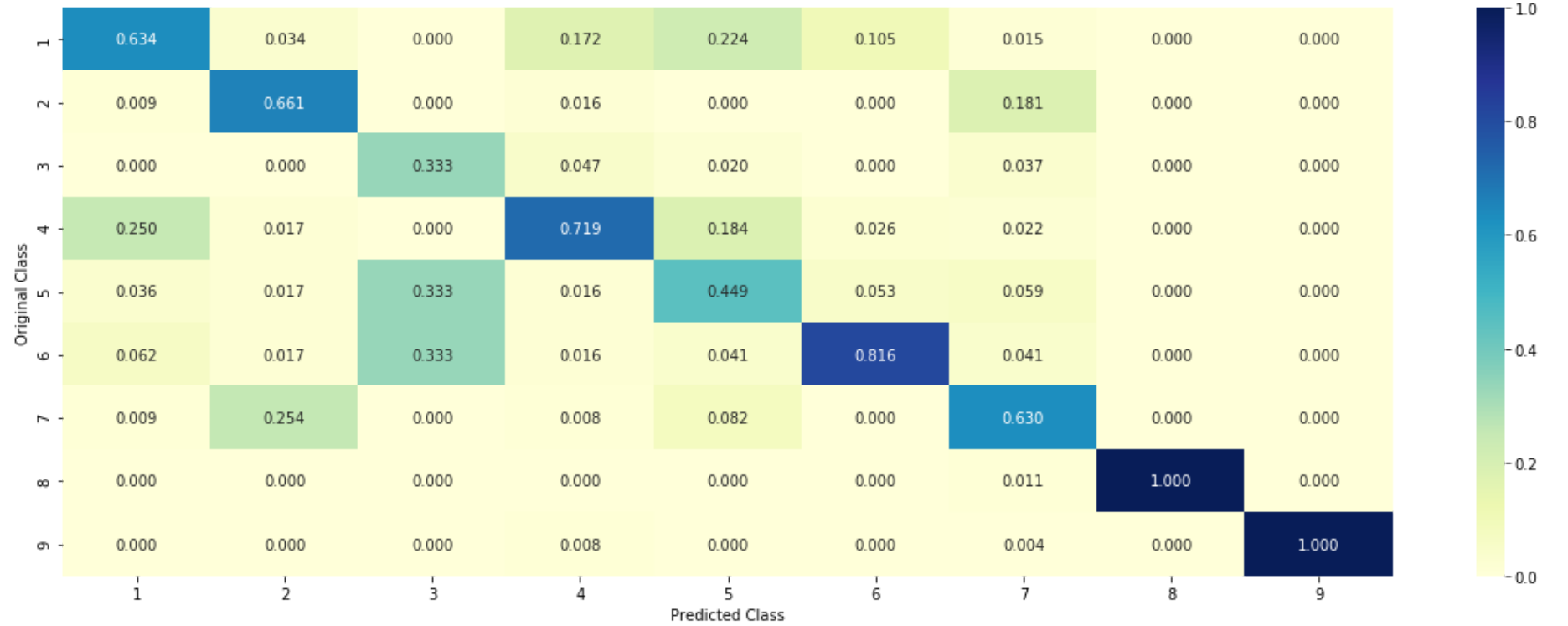
Log loss (test) on the VotingClassifier : 1.1418488103339375

Number of missclassified point : 0.35037593984962406

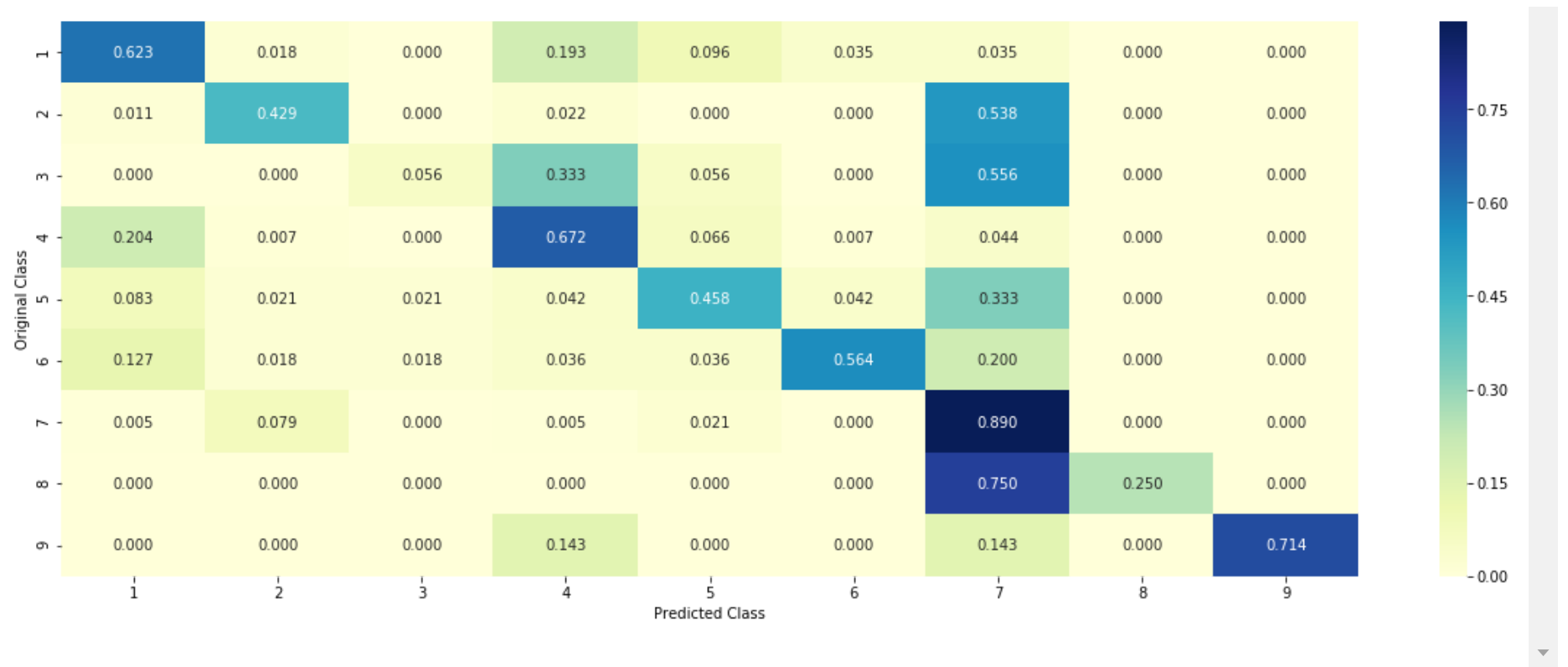
----- Confusion matrix -----



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



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



## Comparing the scores from all the above models

Performance of various model with TFIDF Vectorizer & Top 1000 features

```
In [102]: # http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable
x = PrettyTable()
x.field_names = ["Model-Name", "Train loss", "CV loss", "Test Loss", "% Misclassified"]
x.add_row(["Naive Bayes", "0.535", "1.258", "1.146", "41.72"])
x.add_row(["KNN", "0.749", "1.136", "1.028", "40.22"])
x.add_row(["LR with class Balancing", "0.437", "1.152", "0.995", "36.65"])
x.add_row(["LR without class Balancing", "0.429", "1.117", "1.016", "36.65"])
x.add_row(["Linear SVM", "0.484", "1.178", "1.053", "36.46"])
x.add_row(["RF(One Hot Encoding)", "0.896", "1.248", "1.167", "44.17"])
x.add_row(["RF(Response coding)", "0.05", "1.340", "1.248", "51.50"])
x.add_row(["Stacking Classifier", "0.544", "1.226", "1.135", "36.09"])
x.add_row(["Max voting classifier", "0.822", "1.241", "1.141", "35.03"])
print(x)
```

| Model-Name                 | Train loss | CV loss | Test Loss | % Misclassified |
|----------------------------|------------|---------|-----------|-----------------|
| Naive Bayes                | 0.535      | 1.258   | 1.146     | 41.72           |
| KNN                        | 0.749      | 1.136   | 1.028     | 40.22           |
| LR with class Balancing    | 0.437      | 1.152   | 0.995     | 36.65           |
| LR without class Balancing | 0.429      | 1.117   | 1.016     | 36.65           |
| Linear SVM                 | 0.484      | 1.178   | 1.053     | 36.46           |
| RF(One Hot Encoding)       | 0.896      | 1.248   | 1.167     | 44.17           |
| RF(Response coding)        | 0.05       | 1.340   | 1.248     | 51.50           |
| Stacking Classifier        | 0.544      | 1.226   | 1.135     | 36.09           |
| Max voting classifier      | 0.822      | 1.241   | 1.141     | 35.03           |

## Conclusion / Observations :-

1. We got the minimum test loss with Logistic Regression(with class balancing) which is 1.078 but the percentage of misclassified points are on the higher side being 36.65%.
2. We got the minimum misclassified points with Maximum voting classifier with 35.03% with test log loss of 1.14.

3. Random Forest with Response coding gave the maximum misclassified points with 51.50% of misclassified points. Also there is severe overfitting with this model where the train loss is 0.05 while the test loss is 1.248 which is highest amongst all.
4. So for Task-2 where we used TFIDF vectorizer with top 1000 words we got minimum mis-classified points with Max voting classifier but this model is not very interpretable.