Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [133]:

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from google.colab import drive
drive.mount('/content/drive')
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.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
import math
import nltk
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectKBest, chi2
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

In [134]:

```
nltk.download('stopwords')
```

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

Out[134]:

```
True
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [135]:

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

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

Out[135]:

	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 [136]:

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

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

Out[136]:

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

3.1.3. Preprocessing of text

In [0]:

```
# 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 [138]:

```
#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 : 942.4638800000002 seconds
```

In [139]:

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

Out[139]:

	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 [140]:

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

Out[140]:

	ID	Gene	Variation	Class	TEXT
1109	1109	FANCA	S1088F	1	NaN
1277	1277	ARID5B	Truncating Mutations	1	NaN

	ID	Gene	Variation	Class	TEXT
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 [0]:

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

In [142]:

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

Out[142]:

	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 [0]:

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

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

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

In [144]:

```
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])
```

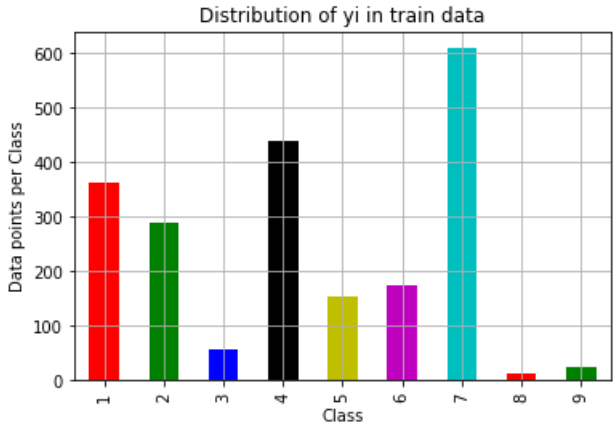
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

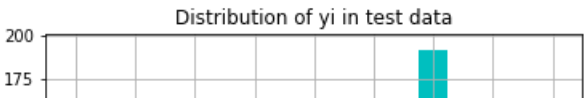
In [145]:

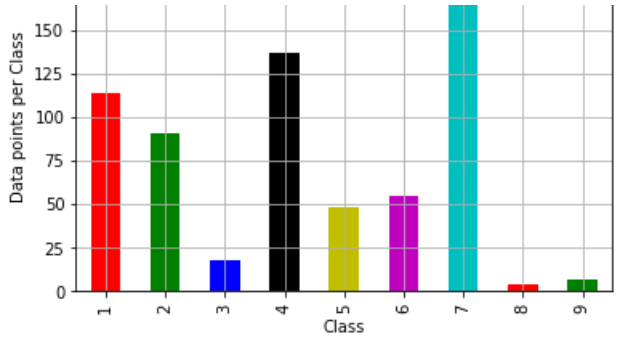
```
# it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution = train_df['Class'].value_counts().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

my_colors = ['r','g','b','k','y','m','c']
train_class_distribution.plot(kind='bar', color = my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
```

```
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.ro
und((train_class_distribution.values[i]/train_df.shape[0]*100), 3), '%)')


print('-'*80)
my_colors = ['r','g','b','k','y','m','c']
test_class_distribution.plot(kind='bar', color = my_colors)
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.rou
nd((test_class_distribution.values[i]/test_df.shape[0]*100), 3), '%)')

print('-'*80)
my_colors = ['r','g','b','k','y','m','c']
cv_class_distribution.plot(kind='bar', color = my_colors)
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_distribution.values[i]/cv_df.shape[0]*100), 3), '%)')
```

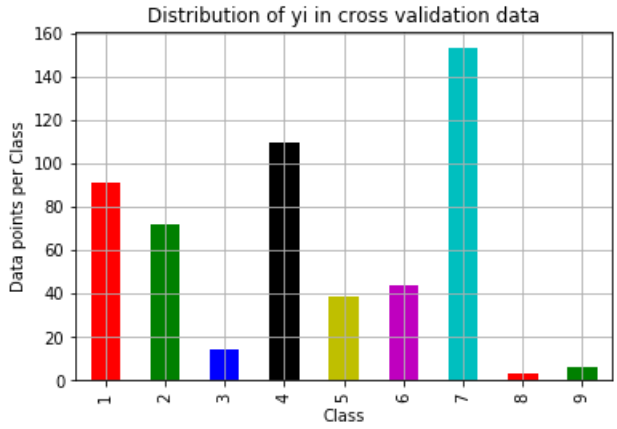



```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298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






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




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

3.2 Prediction using a 'Random' Model

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

In [0]:

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

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

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

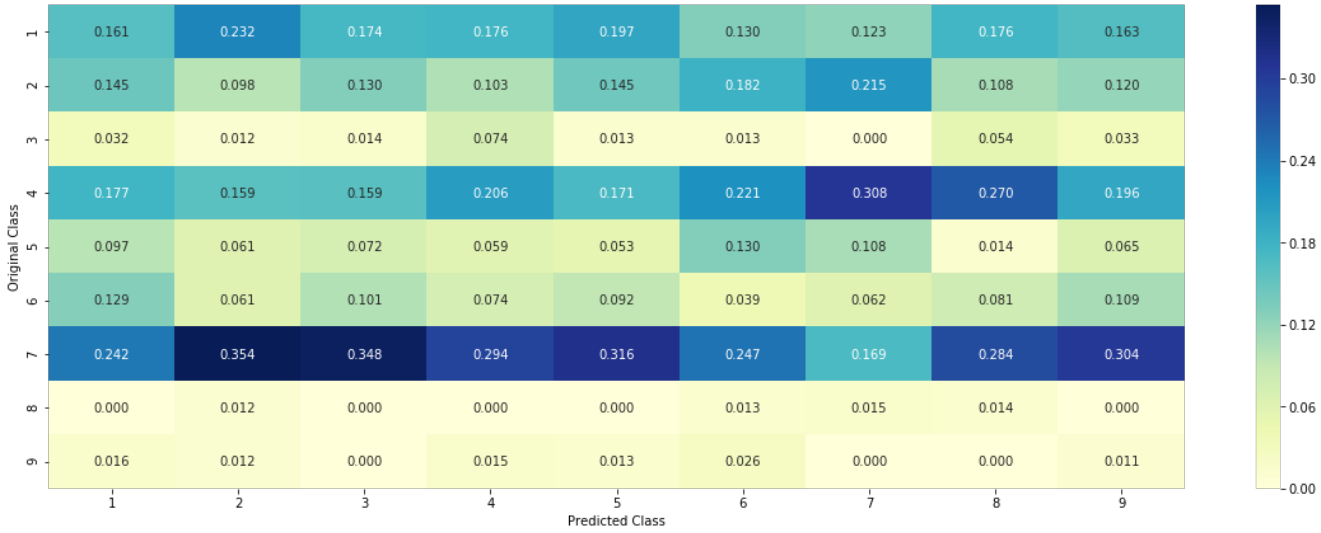
```
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                           [2/3, 4/7]]

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

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

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

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

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

In [147]:

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

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


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

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

```
Log loss on Cross Validation Data using Random Model 2.503383657696682
Log loss on Test Data using Random Model 2.507957961880746
-------------------- Confusion matrix --------------------
```

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



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

3.3 Univariate Analysis

In [0]:

```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
```

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

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

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

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

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

            # cls_cnt.shape[0](numerator) will contain the number of time that particular feature o
ccured 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, 0.0378787878787878, 0.06818181818181818,
0.13636363636363635, 0.25, 0.19318181818181818, 0.0378787878787878, 0.0378787878787878,
0.0378787878787878],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366,
0.27040816326530615, 0.061224489795918366, 0.06632653061224902, 0.051020408163265307, 0.051020408
163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.06818181818181818,
0.06818181818181818, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608,
0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608,
0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917,
0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081
761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295,
0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702,
0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.06666666666666666, 0.17999999999999999, 0.073333333333333334,
0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999,
0.066666666666666666, 0.066666666666666666],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

In [149]:

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

```
Number of Unique Genes : 228
BRCA1      178
TP53       108
EGFR        94
PTEN        87
BRCA2       80
KIT         60
BRAF        53
ERBB2       43
ALK         43
PDGFRA      39
```

```
Name: Gene, dtype: int64
```

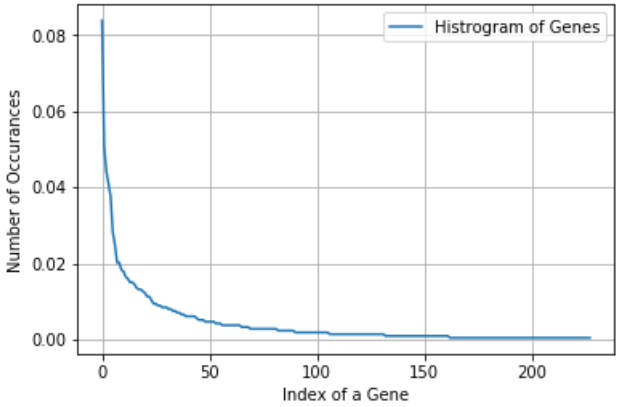
In [150]:

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

```
Ans: There are 228 different categories of genes in the train data, and they are distibuted as follows
```

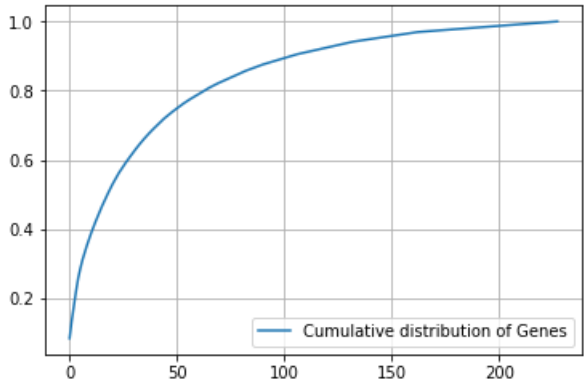
In [151]:

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



In [152]:

```
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 [0]:

```
#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 [154]:

```
print("train_gene_feature_responseCoding is converted feature using respone coding method. The sha
pe of gene feature:", train_gene_feature_responseCoding.shape)
```

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

In [0]:

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

In [156]:

```
train_df['Gene'].head()
```

Out[156]:

```
1689     PMS2
3162     RAF1
1959    NUP93
3013      KIT
1235     PIM1
Name: Gene, dtype: object
```

In [157]:

```
gene_vectorizer.get_feature_names()
```

Out[157]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1b',
 'asxl2',
 'atm',
 'atr',
 'atrx',
 'axl',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcl2l11',
 'bcor',
```

```
'braf',
'brca1',
'brca2',
'brd4',
'brip1',
'btk',
'card11',
'carm1',
'casp8',
'cbl',
'ccnd1',
'ccnd3',
'ccne1',
'cdh1',
'cdk12',
'cdk4',
'cdk6',
'cdk8',
'cdkn1a',
'cdkn1b',
'cdkn2a',
'cdkn2b',
'cdkn2c',
'cebpa',
'chek2',
'cic',
'crebbp',
'ctcf',
'ctla4',
'ctnnb1',
'ddr2',
'dicer1',
'dnmt3a',
'dnmt3b',
'dusp4',
'egfr',
'eif1ax',
'elf3',
'ep300',
'epas1',
'erbb2',
'erbb3',
'erbb4',
'ercc2',
'ercc3',
'ercc4',
'errfi1',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fam58a',
'fanca',
'fancc',
'fat1',
'fbxw7',
'fgf19',
'fgf3',
'fgf4',
'fgfr1',
'fgfr2',
'fgfr3',
'fgfr4',
'flt3',
'foxa1',
'foxl2',
'foxo1',
'foxp1',
'fubp1',
'gata3',
'gli1',
'gnas',
'h3f3a',
'hla',
'hnf1a',
'hras',
```

```
'idh1',
'idh2',
'igf1r',
'ikbke',
'il7r',
'inpp4b',
'jak1',
'jak2',
'jun',
'kdm5a',
'kdm5c',
'kdm6a',
'kdr',
'keap1',
'kit',
'kmt2a',
'kmt2c',
'kmt2d',
'knstrn',
'kras',
'lats2',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
'mdm2',
'mdm4',
'med12',
'mef2b',
'men1',
'met',
'mga',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'mycn',
'myd88',
'ncor1',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',
'nkx2',
'notch1',
'notch2',
'npm1',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'nup93',
'pak1',
'pax8',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pim1',
'pms2',
'pole',
'ppm1d',
'ppp2r1a',
'ppp6c',
'prdm1',
'ptch1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
```

```
 'rab35',
 'rac1',
 'rad21',
 'rad50',
 'rad51b',
 'rad54l',
 'raf1',
 'rara',
 'rasa1',
 'rb1',
 'rbm10',
 'ret',
 'rhoa',
 'rictor',
 'rit1',
 'ros1',
 'runx1',
 'rxra',
 'rybp',
 'sdhb',
 'setd2',
 'sf3b1',
 'shoc2',
 'shq1',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'stat3',
 'stk11',
 'tcf3',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'whsc1l1',
 'xpo1',
 'xrcc2',
 'yap1']
```

In [158]:

```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The sha
pe of gene feature:", train_gene_feature_onehotCoding.shape)
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of g
ene feature: (2124, 228)
```

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

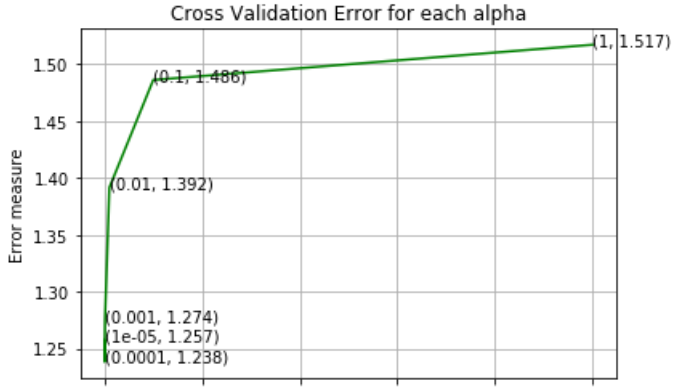
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

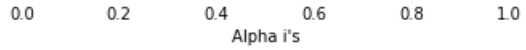
In [159]:

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

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
```

```
"
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=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.clas
ses_, 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.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.256979021765014
For values of alpha =  0.0001 The log loss is: 1.2382024871341475
For values of alpha =  0.001 The log loss is: 1.2744172946131491
For values of alpha =  0.01 The log loss is: 1.3915763444535802
For values of alpha =  0.1 The log loss is: 1.4862963480710547
For values of alpha =  1 The log loss is: 1.5173033106704894
```






```
For values of best alpha =  0.0001 The train log loss is: 0.9775293329552974
For values of best alpha =  0.0001 The cross validation log loss is: 1.2382024871341475
For values of best alpha =  0.0001 The test log loss is: 1.2630728511229796
```

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 [160]:

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

test_coverage=test_df[test_df['Gene'].isin(list(set(train_df['Gene'])))].shape[0]
cv_coverage=cv_df[cv_df['Gene'].isin(list(set(train_df['Gene'])))].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.s
hape[0])*100)
```

```
Q6. How many data points in Test and CV datasets are covered by the  228  genes in train dataset?
Ans
1. In test data 637 out of 665 : 95.78947368421052
2. In cross validation data 512 out of  532 : 96.2406015037594
```

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 [161]:

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

```
Number of Unique Variations : 1936
Truncating_Mutations    58
Amplification           49
Deletion                41
Fusions                 20
Overexpression           3
G12V                     3
K117N                    2
Q61L                     2
G35R                     2
G13D                     2
Name: Variation, dtype: int64
```

In [162]:

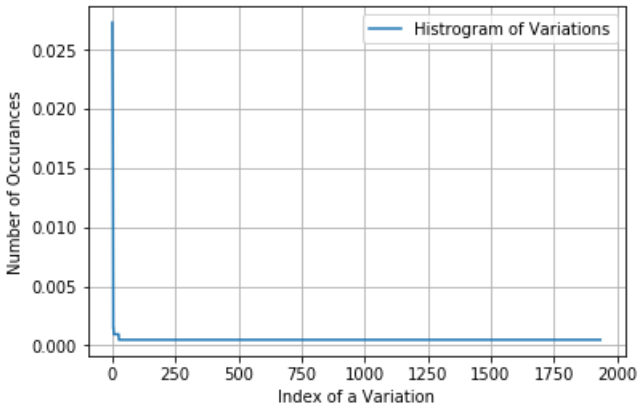
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of variations in the
train data, and they are distibuted as follows",)
```

```
Ans: There are 1936 different categories of variations in the train data, and they are distibuted
as follows
```

In [163]:

```
s = sum(unique_variations.values);
```

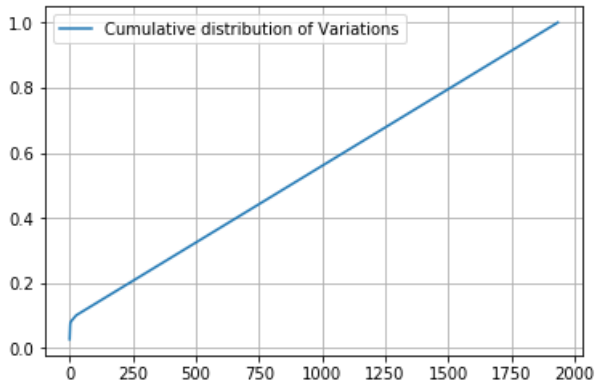
```
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [164]:

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

```
[0.02730697 0.05037665 0.06967985 ... 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 [0]:

```
# 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 [166]:

```
print("train_variation_feature_responseCoding is a converted feature using the response coding met
hod. The shape of Variation feature:", train_variation_feature_responseCoding.shape)
```

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

In [0]:

```
# 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 [168]:

```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding meth
od. The shape of Variation feature:", train_variation_feature_onehotCoding.shape)
```

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The
shape of Variation feature: (2124, 1970)
```

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

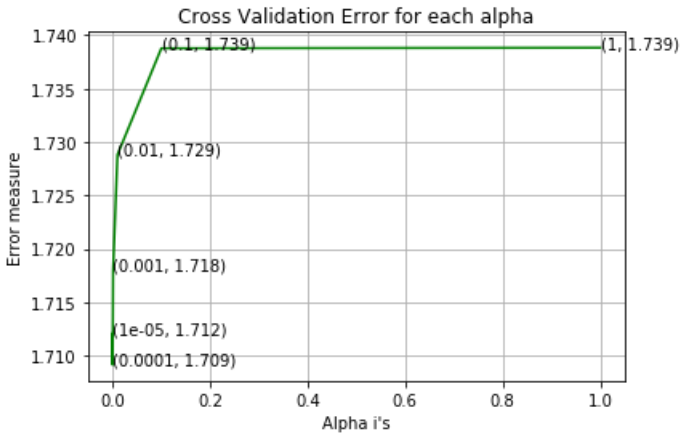
Let's build a model just like the earlier!

In [169]:

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

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# -------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=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.clas
ses_, 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.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.7120290753038556
For values of alpha =  0.0001 The log loss is: 1.7091109519248884
For values of alpha =  0.001 The log loss is: 1.7179579351186323
For values of alpha =  0.01 The log loss is: 1.7288066851664714
For values of alpha =  0.1 The log loss is: 1.738782746727176
For values of alpha =  1 The log loss is: 1.73883448972555
```




```
For values of best alpha =  0.0001 The train log loss is: 0.7471226855670314
For values of best alpha =  0.0001 The cross validation log loss is: 1.7091109519248884
For values of best alpha =  0.0001 The test log loss is: 1.6858307310126224
```

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 [170]:

```
print("Q12. How many data points are covered by total ", unique_variations.shape[0], " genes in te
st and cross validation data sets?")
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.s
hape[0])*100)
```

```
Q12. How many data points are covered by total  1936  genes in test and cross validation data
sets?
Ans
1. In test data 73 out of 665 : 10.977443609022556
2. In cross validation data 57 out of  532 : 10.714285714285714
```

3.2.3 Univariate Analysis on Text Feature

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

In [0]:

```
# 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 [0]:

```
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 [173]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = CountVectorizer(min_df=3)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

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


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

```
Total number of unique words in train data : 53028
```

In [174]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer_bigrams = CountVectorizer(min_df=5, ngram_range = (1, 2))
train_text_feature_onehotCoding_bigrams = text_vectorizer_bigrams.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features_bigrams= text_vectorizer_bigrams.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of featu
res) vector
train_text_fea_counts_bigrams = train_text_feature_onehotCoding_bigrams.sum(axis=0).A1
```

```
# zip(list(text_features),text_fea_counts) will zip a word with its number of times it occured
text_fea_dict_bigrams = dict(zip(list(train_text_features_bigrams),train_text_fea_counts_bigrams))


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

```
Total number of unique words in train data : 475947
```

In [175]:

```
# building a Tfidf Vectorizer with all the words that occured minimum 3 times in train data
text_vectorizer_tfidf = TfidfVectorizer(max_features = 10000)
train_text_feature_tfidf = text_vectorizer_tfidf.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer_tfidf.get_feature_names()

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

```
Total number of unique words in train data : 10000
```

In [0]:

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

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


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

In [0]:

```
#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 [0]:

```
# 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 [0]:

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

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

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

In [0]:

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

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

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

In [0]:

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

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

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

In [0]:

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

In [183]:

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

```
Counter({3: 5319, 4: 3977, 5: 2624, 6: 2501, 9: 2086, 7: 1947, 8: 1859, 10: 1512, 12: 1349, 11: 106
5, 13: 985, 14: 970, 18: 879, 16: 835, 15: 771, 20: 584, 17: 554, 24: 507, 19: 505, 22: 457, 21: 4
55, 27: 418, 23: 406, 25: 376, 32: 361, 28: 356, 42: 348, 26: 337, 30: 332, 54: 314, 36: 309, 29:
280, 33: 270, 31: 249, 34: 244, 35: 234, 40: 205, 39: 205, 41: 200, 46: 199, 37: 190, 45: 182, 48:
178, 38: 176, 43: 169, 44: 168, 52: 154, 50: 152, 47: 150, 57: 147, 56: 147, 49: 147, 51: 137, 63:
128, 53: 127, 55: 126, 64: 121, 61: 115, 60: 115, 58: 114, 59: 112, 70: 107, 62: 106, 65: 105, 68:
104, 66: 103, 74: 99, 72: 98, 67: 97, 71: 94, 69: 92, 92: 91, 76: 89, 78: 88, 81: 87, 73: 84, 75:
83, 80: 82, 91: 74, 97: 73, 86: 73, 82: 73, 108: 71, 85: 71, 84: 70, 99: 69, 87: 68, 93: 67, 79:
67, 77: 66, 88: 64, 89: 63, 98: 60, 94: 60, 90: 60, 100: 59, 96: 58, 102: 57, 110: 53, 104: 52, 1
15: 51, 105: 51, 95: 51, 117: 50, 83: 50, 120: 47, 111: 47, 145: 46, 138: 46, 114: 46, 103: 46, 12
8: 45, 112: 45, 126: 44, 123: 44, 109: 44, 130: 42, 101: 42, 124: 41, 119: 41, 106: 41, 129: 40, 1
39: 39, 132: 39, 136: 38, 125: 38, 121: 38, 141: 37, 133: 37, 140: 36, 134: 36, 127: 36, 107: 36,
148: 35, 122: 35, 118: 35, 204: 33, 135: 33, 116: 33, 113: 33, 166: 32, 151: 32, 144: 32, 157: 31,
147: 31, 146: 31, 142: 30, 172: 29, 171: 29, 154: 29, 131: 29, 168: 28, 162: 28, 152: 28, 164: 27,
149: 27, 232: 26, 161: 26, 160: 26, 155: 26, 143: 26, 137: 26, 190: 25, 188: 25, 165: 25, 153: 25,
150: 25, 219: 24, 212: 24, 208: 24, 193: 24, 186: 24, 184: 24, 173: 24, 235: 23, 222: 23, 214: 23,
213: 23, 207: 23, 202: 23, 180: 23, 179: 23, 176: 23, 209: 22, 206: 22, 198: 22, 175: 22, 174: 22,
159: 22, 273: 21, 210: 21, 200: 21, 183: 21, 181: 21, 169: 21, 156: 21, 272: 20, 263: 20, 258: 20,
238: 20, 230: 20, 221: 20, 217: 20, 197: 20, 196: 20, 189: 20, 177: 20, 248: 19, 205: 19, 170: 19,
167: 19, 241: 18, 237: 18, 234: 18, 201: 18, 182: 18, 297: 17, 295: 17, 292: 17, 282: 17, 245: 17,
226: 17, 223: 17, 203: 17, 195: 17, 194: 17, 187: 17, 178: 17, 316: 16, 290: 16, 254: 16, 227: 16,
211: 16, 199: 16, 192: 16, 191: 16, 185: 16, 158: 16, 341: 15, 308: 15, 300: 15, 287: 15, 277: 15,
271: 15, 268: 15, 260: 15, 252: 15, 249: 15, 231: 15, 220: 15, 215: 15, 163: 15, 416: 14, 354: 14,
348: 14, 296: 14, 265: 14, 242: 14, 224: 14, 396: 13, 382: 13, 367: 13, 335: 13, 330: 13, 276: 13,
```

266: 13, 253: 13, 244: 13, 240: 13, 239: 13, 233: 13, 218: 13, 418: 12, 342: 12, 337: 12, 332: 12,
286: 12, 285: 12, 278: 12, 270: 12, 267: 12, 228: 12, 531: 11, 385: 11, 365: 11, 362: 11, 353: 11,
345: 11, 336: 11, 310: 11, 304: 11, 303: 11, 302: 11, 298: 11, 291: 11, 289: 11, 288: 11, 274: 11,
264: 11, 259: 11, 256: 11, 243: 11, 637: 10, 504: 10, 482: 10, 457: 10, 433: 10, 383: 10, 370: 10,
358: 10, 355: 10, 349: 10, 343: 10, 327: 10, 324: 10, 322: 10, 320: 10, 311: 10, 307: 10, 305: 10,
301: 10, 293: 10, 283: 10, 279: 10, 255: 10, 251: 10, 250: 10, 247: 10, 536: 9, 515: 9, 498: 9, 44
5: 9, 439: 9, 438: 9, 424: 9, 400: 9, 398: 9, 395: 9, 391: 9, 386: 9, 366: 9, 351: 9, 346: 9, 333:
9, 331: 9, 328: 9, 325: 9, 323: 9, 314: 9, 309: 9, 299: 9, 294: 9, 281: 9, 269: 9, 261: 9, 246: 9,
225: 9, 762: 8, 567: 8, 558: 8, 519: 8, 505: 8, 499: 8, 452: 8, 435: 8, 432: 8, 426: 8, 425: 8,
423: 8, 407: 8, 390: 8, 389: 8, 381: 8, 375: 8, 373: 8, 344: 8, 318: 8, 317: 8, 257: 8, 236: 8,
216: 8, 955: 7, 763: 7, 747: 7, 725: 7, 621: 7, 576: 7, 522: 7, 514: 7, 507: 7, 495: 7, 484: 7,
478: 7, 475: 7, 472: 7, 471: 7, 470: 7, 466: 7, 460: 7, 459: 7, 446: 7, 437: 7, 429: 7, 414: 7,
404: 7, 388: 7, 379: 7, 378: 7, 374: 7, 371: 7, 360: 7, 356: 7, 352: 7, 339: 7, 338: 7, 329: 7,
321: 7, 315: 7, 313: 7, 312: 7, 306: 7, 284: 7, 280: 7, 229: 7, 1152: 6, 1051: 6, 986: 6, 850: 6,
810: 6, 798: 6, 722: 6, 717: 6, 700: 6, 697: 6, 660: 6, 646: 6, 640: 6, 629: 6, 626: 6, 622: 6,
594: 6, 589: 6, 570: 6, 563: 6, 545: 6, 544: 6, 541: 6, 511: 6, 510: 6, 502: 6, 500: 6, 494: 6,
468: 6, 465: 6, 461: 6, 431: 6, 427: 6, 409: 6, 403: 6, 393: 6, 392: 6, 377: 6, 372: 6, 369: 6,
368: 6, 361: 6, 359: 6, 350: 6, 340: 6, 334: 6, 319: 6, 1288: 5, 1273: 5, 1171: 5, 1159: 5, 991: 5,
987: 5, 966: 5, 965: 5, 952: 5, 914: 5, 913: 5, 897: 5, 867: 5, 855: 5, 854: 5, 805: 5, 801: 5,
787: 5, 769: 5, 746: 5, 734: 5, 721: 5, 712: 5, 709: 5, 704: 5, 682: 5, 679: 5, 678: 5, 676: 5,
650: 5, 645: 5, 643: 5, 635: 5, 630: 5, 617: 5, 615: 5, 614: 5, 612: 5, 608: 5, 605: 5, 600: 5,
596: 5, 591: 5, 580: 5, 571: 5, 555: 5, 552: 5, 540: 5, 539: 5, 535: 5, 534: 5, 533: 5, 532: 5,
529: 5, 527: 5, 526: 5, 518: 5, 517: 5, 516: 5, 513: 5, 501: 5, 496: 5, 488: 5, 487: 5, 486: 5,
485: 5, 480: 5, 476: 5, 473: 5, 467: 5, 464: 5, 462: 5, 456: 5, 455: 5, 451: 5, 449: 5, 448: 5,
422: 5, 413: 5, 410: 5, 406: 5, 402: 5, 401: 5, 394: 5, 387: 5, 380: 5, 364: 5, 363: 5, 326: 5,
275: 5, 262: 5, 2338: 4, 2124: 4, 2008: 4, 1860: 4, 1782: 4, 1626: 4, 1543: 4, 1409: 4, 1339: 4, 12
64: 4, 1261: 4, 1234: 4, 1187: 4, 1158: 4, 1130: 4, 1108: 4, 1096: 4, 1064: 4, 1047: 4, 1036: 4, 10
33: 4, 1023: 4, 973: 4, 909: 4, 892: 4, 880: 4, 863: 4, 840: 4, 832: 4, 831: 4, 821: 4, 818: 4,
817: 4, 804: 4, 803: 4, 799: 4, 790: 4, 788: 4, 781: 4, 772: 4, 760: 4, 748: 4, 741: 4, 739: 4,
726: 4, 719: 4, 674: 4, 668: 4, 659: 4, 654: 4, 652: 4, 644: 4, 642: 4, 641: 4, 634: 4, 620: 4,
618: 4, 611: 4, 610: 4, 604: 4, 603: 4, 602: 4, 587: 4, 578: 4, 574: 4, 566: 4, 556: 4, 554: 4,
553: 4, 537: 4, 525: 4, 493: 4, 490: 4, 483: 4, 479: 4, 477: 4, 458: 4, 454: 4, 450: 4, 443: 4,
436: 4, 428: 4, 421: 4, 411: 4, 399: 4, 397: 4, 376: 4, 357: 4, 347: 4, 3528: 3, 3210: 3, 2721: 3,
2512: 3, 2497: 3, 2426: 3, 1988: 3, 1987: 3, 1953: 3, 1929: 3, 1907: 3, 1904: 3, 1881: 3, 1868: 3,
1856: 3, 1841: 3, 1833: 3, 1818: 3, 1808: 3, 1781: 3, 1754: 3, 1681: 3, 1665: 3, 1594: 3, 1590: 3,
1584: 3, 1569: 3, 1565: 3, 1551: 3, 1527: 3, 1524: 3, 1470: 3, 1434: 3, 1423: 3, 1416: 3, 1407: 3,
1377: 3, 1372: 3, 1338: 3, 1324: 3, 1320: 3, 1318: 3, 1307: 3, 1305: 3, 1302: 3, 1299: 3, 1275: 3,
1271: 3, 1269: 3, 1263: 3, 1256: 3, 1245: 3, 1230: 3, 1217: 3, 1206: 3, 1185: 3, 1164: 3, 1157: 3,
1154: 3, 1151: 3, 1150: 3, 1131: 3, 1124: 3, 1118: 3, 1114: 3, 1112: 3, 1088: 3, 1086: 3, 1084: 3,
1068: 3, 1058: 3, 1052: 3, 1018: 3, 1013: 3, 1007: 3, 1004: 3, 1001: 3, 996: 3, 979: 3, 959: 3, 956
: 3, 951: 3, 946: 3, 943: 3, 942: 3, 940: 3, 931: 3, 916: 3, 904: 3, 888: 3, 876: 3, 871: 3, 864:
3, 860: 3, 857: 3, 852: 3, 849: 3, 847: 3, 844: 3, 830: 3, 826: 3, 824: 3, 822: 3, 820: 3, 815: 3,
809: 3, 807: 3, 793: 3, 789: 3, 785: 3, 780: 3, 775: 3, 768: 3, 767: 3, 764: 3, 758: 3, 754: 3,
752: 3, 751: 3, 750: 3, 749: 3, 745: 3, 742: 3, 729: 3, 727: 3, 724: 3, 715: 3, 708: 3, 707: 3,
705: 3, 703: 3, 696: 3, 695: 3, 694: 3, 693: 3, 687: 3, 686: 3, 685: 3, 683: 3, 673: 3, 665: 3,
664: 3, 663: 3, 657: 3, 638: 3, 632: 3, 631: 3, 627: 3, 625: 3, 616: 3, 613: 3, 597: 3, 595: 3,
592: 3, 588: 3, 584: 3, 583: 3, 581: 3, 575: 3, 569: 3, 565: 3, 564: 3, 562: 3, 551: 3, 528: 3,
524: 3, 521: 3, 512: 3, 509: 3, 508: 3, 503: 3, 497: 3, 492: 3, 489: 3, 469: 3, 463: 3, 453: 3,
447: 3, 444: 3, 441: 3, 440: 3, 434: 3, 430: 3, 420: 3, 417: 3, 412: 3, 14147: 2, 9723: 2, 9032:
2, 8068: 2, 6931: 2, 6448: 2, 6378: 2, 6025: 2, 5823: 2, 5694: 2, 5163: 2, 5148: 2, 5048: 2, 4996:
2, 4948: 2, 4918: 2, 4418: 2, 4327: 2, 4189: 2, 4187: 2, 3975: 2, 3827: 2, 3747: 2, 3704: 2, 3650:
2, 3603: 2, 3594: 2, 3562: 2, 3543: 2, 3486: 2, 3477: 2, 3469: 2, 3465: 2, 3453: 2, 3448: 2, 3403:
2, 3359: 2, 3353: 2, 3341: 2, 3270: 2, 3269: 2, 3200: 2, 3197: 2, 3177: 2, 3043: 2, 3004: 2, 2901:
2, 2877: 2, 2827: 2, 2810: 2, 2752: 2, 2708: 2, 2670: 2, 2665: 2, 2647: 2, 2636: 2, 2623: 2, 2618:
2, 2615: 2, 2605: 2, 2604: 2, 2598: 2, 2593: 2, 2569: 2, 2560: 2, 2533: 2, 2531: 2, 2479: 2, 2442:
2, 2436: 2, 2415: 2, 2407: 2, 2380: 2, 2370: 2, 2361: 2, 2351: 2, 2348: 2, 2347: 2, 2296: 2, 2273:
2, 2272: 2, 2260: 2, 2247: 2, 2226: 2, 2225: 2, 2213: 2, 2160: 2, 2146: 2, 2127: 2, 2121: 2, 2119:
2, 2102: 2, 2081: 2, 2070: 2, 2068: 2, 2050: 2, 2042: 2, 2029: 2, 2028: 2, 1997: 2, 1991: 2, 1989:
2, 1976: 2, 1967: 2, 1955: 2, 1944: 2, 1938: 2, 1927: 2, 1922: 2, 1918: 2, 1902: 2, 1901: 2, 1899:
2, 1887: 2, 1883: 2, 1879: 2, 1862: 2, 1861: 2, 1854: 2, 1846: 2, 1838: 2, 1834: 2, 1815: 2, 1804:
2, 1800: 2, 1794: 2, 1763: 2, 1756: 2, 1740: 2, 1737: 2, 1731: 2, 1730: 2, 1721: 2, 1709: 2, 1685:
2, 1684: 2, 1680: 2, 1676: 2, 1654: 2, 1646: 2, 1644: 2, 1637: 2, 1631: 2, 1629: 2, 1624: 2, 1619:
2, 1609: 2, 1608: 2, 1605: 2, 1603: 2, 1593: 2, 1591: 2, 1587: 2, 1586: 2, 1585: 2, 1583: 2, 1549:
2, 1548: 2, 1547: 2, 1545: 2, 1541: 2, 1538: 2, 1537: 2, 1530: 2, 1526: 2, 1525: 2, 1522: 2, 1519:
2, 1517: 2, 1511: 2, 1506: 2, 1498: 2, 1494: 2, 1490: 2, 1486: 2, 1482: 2, 1471: 2, 1463: 2, 1461:
2, 1457: 2, 1443: 2, 1436: 2, 1432: 2, 1421: 2, 1415: 2, 1411: 2, 1408: 2, 1401: 2, 1400: 2, 1399:
2, 1396: 2, 1386: 2, 1380: 2, 1379: 2, 1373: 2, 1370: 2, 1369: 2, 1362: 2, 1361: 2, 1360: 2, 1357:
2, 1356: 2, 1350: 2, 1345: 2, 1315: 2, 1308: 2, 1304: 2, 1303: 2, 1296: 2, 1291: 2, 1290: 2, 1286:
2, 1284: 2, 1283: 2, 1281: 2, 1280: 2, 1274: 2, 1262: 2, 1257: 2, 1251: 2, 1249: 2, 1248: 2, 1246:
2, 1244: 2, 1240: 2, 1239: 2, 1235: 2, 1228: 2, 1227: 2, 1226: 2, 1220: 2, 1219: 2, 1216: 2, 1214:
2, 1208: 2, 1207: 2, 1199: 2, 1198: 2, 1190: 2, 1189: 2, 1180: 2, 1178: 2, 1177: 2, 1174: 2, 1170:
2, 1167: 2, 1155: 2, 1153: 2, 1145: 2, 1137: 2, 1136: 2, 1128: 2, 1125: 2, 1121: 2, 1109: 2, 1107:
2, 1106: 2, 1104: 2, 1102: 2, 1095: 2, 1092: 2, 1089: 2, 1074: 2, 1071: 2, 1070: 2, 1067: 2, 1060:
2, 1044: 2, 1043: 2, 1041: 2, 1039: 2, 1038: 2, 1032: 2, 1029: 2, 1028: 2, 1021: 2, 1009: 2, 1006:
2, 1000: 2, 999: 2, 993: 2, 990: 2, 980: 2, 975: 2, 974: 2, 972: 2, 970: 2, 969: 2, 968: 2, 967:
2, 964: 2, 962: 2, 961: 2, 958: 2, 957: 2, 953: 2, 948: 2, 947: 2, 945: 2, 936: 2, 934: 2, 932: 2,
930: 2, 928: 2, 926: 2, 923: 2, 921: 2, 919: 2, 918: 2, 917: 2, 907: 2, 906: 2, 905: 2, 903: 2,

896: 2, 893: 2, 890: 2, 885: 2, 882: 2, 879: 2, 878: 2, 874: 2, 870: 2, 865: 2, 861: 2, 856: 2,
853: 2, 851: 2, 842: 2, 841: 2, 837: 2, 834: 2, 829: 2, 825: 2, 819: 2, 814: 2, 811: 2, 806: 2,
797: 2, 791: 2, 779: 2, 778: 2, 777: 2, 771: 2, 770: 2, 766: 2, 759: 2, 744: 2, 740: 2, 737: 2,
716: 2, 706: 2, 702: 2, 698: 2, 690: 2, 670: 2, 669: 2, 666: 2, 661: 2, 658: 2, 656: 2, 655: 2,
649: 2, 633: 2, 624: 2, 623: 2, 619: 2, 609: 2, 601: 2, 599: 2, 598: 2, 590: 2, 585: 2, 579: 2,
568: 2, 560: 2, 559: 2, 550: 2, 548: 2, 543: 2, 530: 2, 520: 2, 506: 2, 491: 2, 481: 2, 419: 2,
415: 2, 408: 2, 405: 2, 153874: 1, 116704: 1, 79834: 1, 68160: 1, 66697: 1, 66613: 1, 66349: 1, 64
127: 1, 62557: 1, 57838: 1, 54729: 1, 49332: 1, 49162: 1, 47522: 1, 46751: 1, 44510: 1, 44035: 1,
42523: 1, 41870: 1, 41411: 1, 40975: 1, 40860: 1, 40546: 1, 39095: 1, 38669: 1, 38110: 1, 37492: 1
, 36716: 1, 36290: 1, 36236: 1, 34517: 1, 34316: 1, 33838: 1, 33148: 1, 31830: 1, 31748: 1, 29592:
1, 28137: 1, 27253: 1, 26874: 1, 26249: 1, 26134: 1, 25988: 1, 25674: 1, 25183: 1, 24761: 1, 24750
: 1, 24562: 1, 24487: 1, 24359: 1, 24031: 1, 23647: 1, 23131: 1, 22781: 1, 22456: 1, 22389: 1, 222
85: 1, 22258: 1, 22058: 1, 21405: 1, 21041: 1, 20888: 1, 20589: 1, 20361: 1, 20291: 1, 19708: 1, 1
9700: 1, 19676: 1, 19528: 1, 19151: 1, 18884: 1, 18814: 1, 18624: 1, 18425: 1, 18373: 1, 18369: 1,
18098: 1, 18009: 1, 18007: 1, 17897: 1, 17858: 1, 17818: 1, 17628: 1, 17548: 1, 17541: 1, 17430: 1
, 17408: 1, 17389: 1, 17378: 1, 17322: 1, 17293: 1, 17228: 1, 16916: 1, 16849: 1, 16836: 1, 16780:
1, 16623: 1, 16494: 1, 16298: 1, 16203: 1, 16021: 1, 15958: 1, 15869: 1, 15792: 1, 15577: 1, 15545
: 1, 15385: 1, 15321: 1, 15313: 1, 15167: 1, 15015: 1, 14964: 1, 14898: 1, 14777: 1, 14716: 1, 146
38: 1, 14583: 1, 14347: 1, 14264: 1, 13886: 1, 13755: 1, 13707: 1, 13609: 1, 13550: 1, 13463: 1, 1
3355: 1, 13317: 1, 13271: 1, 13222: 1, 13128: 1, 13030: 1, 13010: 1, 12996: 1, 12946: 1, 12883: 1,
12838: 1, 12791: 1, 12788: 1, 12741: 1, 12726: 1, 12671: 1, 12661: 1, 12647: 1, 12486: 1, 12471: 1
, 12413: 1, 12410: 1, 12404: 1, 12379: 1, 12333: 1, 12277: 1, 12263: 1, 12252: 1, 12241: 1, 12214:
1, 12206: 1, 12176: 1, 12139: 1, 12124: 1, 12116: 1, 12110: 1, 12005: 1, 11998: 1, 11972: 1, 11966
: 1, 11957: 1, 11900: 1, 11811: 1, 11673: 1, 11658: 1, 11610: 1, 11568: 1, 11467: 1, 11453: 1, 113
48: 1, 11273: 1, 11127: 1, 11123: 1, 11116: 1, 11062: 1, 10984: 1, 10968: 1, 10963: 1, 10861: 1, 1
0823: 1, 10822: 1, 10818: 1, 10753: 1, 10665: 1, 10540: 1, 10492: 1, 10461: 1, 10342: 1, 10336: 1,
10316: 1, 10291: 1, 10289: 1, 10260: 1, 10257: 1, 10222: 1, 10205: 1, 10175: 1, 10163: 1, 9925: 1,
9888: 1, 9884: 1, 9881: 1, 9799: 1, 9761: 1, 9686: 1, 9599: 1, 9589: 1, 9533: 1, 9519: 1, 9491: 1,
9462: 1, 9460: 1, 9429: 1, 9423: 1, 9403: 1, 9393: 1, 9292: 1, 9273: 1, 9269: 1, 9217: 1, 9208: 1,
9197: 1, 9187: 1, 9131: 1, 9108: 1, 9076: 1, 9022: 1, 8997: 1, 8951: 1, 8928: 1, 8919: 1, 8886: 1,
8884: 1, 8848: 1, 8845: 1, 8812: 1, 8784: 1, 8670: 1, 8659: 1, 8583: 1, 8572: 1, 8479: 1, 8475: 1,
8392: 1, 8374: 1, 8351: 1, 8348: 1, 8325: 1, 8311: 1, 8303: 1, 8295: 1, 8290: 1, 8278: 1, 8261: 1,
8234: 1, 8197: 1, 8119: 1, 8083: 1, 8070: 1, 8069: 1, 8042: 1, 8036: 1, 8027: 1, 8022: 1, 8017: 1,
7972: 1, 7969: 1, 7943: 1, 7938: 1, 7935: 1, 7917: 1, 7876: 1, 7875: 1, 7855: 1, 7769: 1, 7767: 1,
7738: 1, 7701: 1, 7630: 1, 7622: 1, 7621: 1, 7595: 1, 7579: 1, 7572: 1, 7543: 1, 7527: 1, 7524: 1,
7522: 1, 7510: 1, 7508: 1, 7503: 1, 7501: 1, 7406: 1, 7363: 1, 7356: 1, 7327: 1, 7326: 1, 7277: 1,
7256: 1, 7244: 1, 7223: 1, 7222: 1, 7205: 1, 7174: 1, 7164: 1, 7131: 1, 7126: 1, 7119: 1, 7112: 1,
7105: 1, 7094: 1, 7081: 1, 7050: 1, 7031: 1, 7012: 1, 6996: 1, 6954: 1, 6945: 1, 6932: 1, 6929: 1,
6901: 1, 6891: 1, 6868: 1, 6867: 1, 6866: 1, 6839: 1, 6822: 1, 6806: 1, 6775: 1, 6758: 1, 6755: 1,
6742: 1, 6728: 1, 6708: 1, 6679: 1, 6678: 1, 6674: 1, 6659: 1, 6630: 1, 6629: 1, 6627: 1, 6593: 1,
6571: 1, 6562: 1, 6561: 1, 6560: 1, 6552: 1, 6514: 1, 6512: 1, 6463: 1, 6425: 1, 6418: 1, 6357: 1,
6353: 1, 6349: 1, 6337: 1, 6305: 1, 6283: 1, 6265: 1, 6264: 1, 6231: 1, 6218: 1, 6212: 1, 6196: 1,
6188: 1, 6185: 1, 6179: 1, 6148: 1, 6141: 1, 6136: 1, 6123: 1, 6122: 1, 6083: 1, 6051: 1, 6048: 1,
6032: 1, 6022: 1, 6019: 1, 6009: 1, 5995: 1, 5979: 1, 5974: 1, 5967: 1, 5960: 1, 5948: 1, 5932: 1,
5923: 1, 5862: 1, 5798: 1, 5778: 1, 5776: 1, 5764: 1, 5763: 1, 5762: 1, 5749: 1, 5748: 1, 5735: 1,
5728: 1, 5685: 1, 5652: 1, 5637: 1, 5633: 1, 5616: 1, 5608: 1, 5606: 1, 5585: 1, 5575: 1, 5572: 1,
5570: 1, 5564: 1, 5562: 1, 5553: 1, 5548: 1, 5542: 1, 5527: 1, 5513: 1, 5507: 1, 5498: 1, 5488: 1,
5478: 1, 5469: 1, 5463: 1, 5454: 1, 5451: 1, 5430: 1, 5425: 1, 5394: 1, 5370: 1, 5351: 1, 5344: 1,
5331: 1, 5326: 1, 5319: 1, 5316: 1, 5308: 1, 5304: 1, 5302: 1, 5300: 1, 5285: 1, 5258: 1, 5230: 1,
5199: 1, 5175: 1, 5158: 1, 5153: 1, 5149: 1, 5112: 1, 5083: 1, 5077: 1, 5063: 1, 5059: 1, 5058: 1,
5041: 1, 5034: 1, 5028: 1, 5023: 1, 5010: 1, 5002: 1, 4971: 1, 4967: 1, 4965: 1, 4960: 1, 4954: 1,
4946: 1, 4931: 1, 4924: 1, 4922: 1, 4921: 1, 4915: 1, 4913: 1, 4902: 1, 4899: 1, 4894: 1, 4852: 1,
4845: 1, 4844: 1, 4841: 1, 4828: 1, 4827: 1, 4825: 1, 4821: 1, 4820: 1, 4818: 1, 4814: 1, 4804: 1,
4790: 1, 4789: 1, 4788: 1, 4775: 1, 4766: 1, 4758: 1, 4757: 1, 4737: 1, 4723: 1, 4713: 1, 4706: 1,
4695: 1, 4685: 1, 4659: 1, 4645: 1, 4632: 1, 4629: 1, 4627: 1, 4623: 1, 4613: 1, 4593: 1, 4574: 1,
4569: 1, 4551: 1, 4549: 1, 4536: 1, 4530: 1, 4527: 1, 4526: 1, 4525: 1, 4496: 1, 4486: 1, 4476: 1,
4471: 1, 4470: 1, 4458: 1, 4453: 1, 4448: 1, 4423: 1, 4406: 1, 4399: 1, 4390: 1, 4389: 1, 4388: 1,
4383: 1, 4382: 1, 4380: 1, 4378: 1, 4376: 1, 4374: 1, 4371: 1, 4369: 1, 4366: 1, 4363: 1, 4362: 1,
4354: 1, 4349: 1, 4342: 1, 4315: 1, 4309: 1, 4307: 1, 4287: 1, 4275: 1, 4273: 1, 4272: 1, 4266: 1,
4262: 1, 4255: 1, 4249: 1, 4246: 1, 4243: 1, 4228: 1, 4217: 1, 4206: 1, 4197: 1, 4175: 1, 4171: 1,
4169: 1, 4163: 1, 4154: 1, 4152: 1, 4147: 1, 4145: 1, 4139: 1, 4134: 1, 4133: 1, 4128: 1, 4122: 1,
4118: 1, 4109: 1, 4102: 1, 4099: 1, 4096: 1, 4089: 1, 4082: 1, 4078: 1, 4074: 1, 4070: 1, 4065: 1,
4059: 1, 4058: 1, 4055: 1, 4048: 1, 4041: 1, 4037: 1, 4013: 1, 4012: 1, 4004: 1, 4000: 1, 3994: 1,
3993: 1, 3986: 1, 3980: 1, 3977: 1, 3972: 1, 3959: 1, 3953: 1, 3940: 1, 3935: 1, 3931: 1, 3929: 1,
3924: 1, 3921: 1, 3902: 1, 3893: 1, 3892: 1, 3891: 1, 3884: 1, 3875: 1, 3866: 1, 3863: 1, 3860: 1,
3857: 1, 3848: 1, 3844: 1, 3842: 1, 3834: 1, 3823: 1, 3814: 1, 3801: 1, 3797: 1, 3781: 1, 3770: 1,
3756: 1, 3755: 1, 3749: 1, 3748: 1, 3745: 1, 3737: 1, 3733: 1, 3719: 1, 3716: 1, 3711: 1, 3703: 1,
3701: 1, 3696: 1, 3681: 1, 3678: 1, 3671: 1, 3670: 1, 3663: 1, 3661: 1, 3653: 1, 3643: 1, 3640: 1,
3625: 1, 3611: 1, 3609: 1, 3605: 1, 3593: 1, 3591: 1, 3589: 1, 3588: 1, 3584: 1, 3579: 1, 3573: 1,
3566: 1, 3559: 1, 3556: 1, 3553: 1, 3552: 1, 3551: 1, 3544: 1, 3539: 1, 3533: 1, 3529: 1, 3522: 1,
3516: 1, 3514: 1, 3508: 1, 3505: 1, 3501: 1, 3494: 1, 3485: 1, 3476: 1, 3470: 1, 3468: 1, 3466: 1,
3464: 1, 3435: 1, 3434: 1, 3426: 1, 3425: 1, 3410: 1, 3406: 1, 3400: 1, 3399: 1, 3393: 1, 3392: 1,
3390: 1, 3385: 1, 3380: 1, 3370: 1, 3369: 1, 3368: 1, 3367: 1, 3365: 1, 3361: 1, 3358: 1, 3352: 1,
3350: 1, 3342: 1, 3336: 1, 3328: 1, 3327: 1, 3322: 1, 3317: 1, 3315: 1, 3313: 1, 3312: 1, 3310: 1,
3303: 1, 3301: 1, 3299: 1, 3292: 1, 3288: 1, 3273: 1, 3260: 1, 3258: 1, 3255: 1, 3247: 1, 3242: 1,
3228: 1, 3222: 1, 3216: 1, 3213: 1, 3202: 1, 3192: 1, 3191: 1, 3186: 1, 3179: 1, 3176: 1, 3162: 1,
3157: 1, 3156: 1, 3152: 1, 3151: 1, 3147: 1, 3145: 1, 3144: 1, 3140: 1, 3138: 1, 3137: 1, 3135: 1,

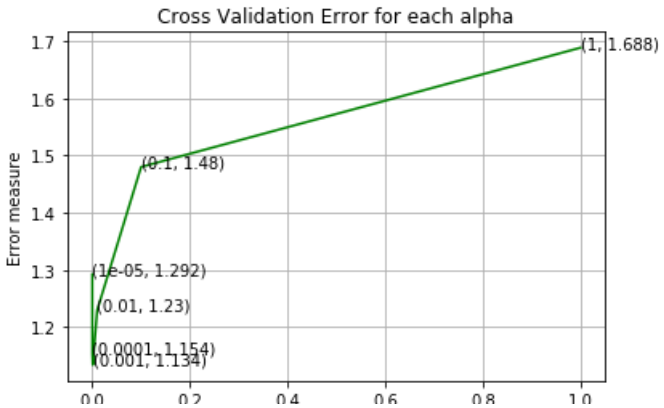
```
3133: 1, 3131: 1, 3130: 1, 3124: 1, 3109: 1, 3108: 1, 3107: 1, 3106: 1, 3104: 1, 3102: 1, 3096: 1,
3095: 1, 3085: 1, 3083: 1, 3078: 1, 3076: 1, 3069: 1, 3063: 1, 3057: 1, 3055: 1, 3046: 1, 3038: 1,
3037: 1, 3036: 1, 3030: 1, 3027: 1, 3025: 1, 3013: 1, 3007: 1, 3002: 1, 2992: 1, 2989: 1, 2988: 1,
2980: 1, 2978: 1, 2971: 1, 2957: 1, 2949: 1, 2946: 1, 2945: 1, 2944: 1, 2943: 1, 2925: 1, 2914: 1,
2913: 1, 2911: 1, 2907: 1, 2906: 1, 2904: 1, 2897: 1, 2895: 1, 2894: 1, 2893: 1, 2891: 1, 2890: 1,
2889: 1, 2888: 1, 2872: 1, 2865: 1, 2860: 1, 2853: 1, 2852: 1, 2851: 1, 2848: 1, 2845: 1, 2841: 1,
2839: 1, 2830: 1, 2824: 1, 2823: 1, 2818: 1, 2804: 1, 2797: 1, 2787: 1, 2785: 1, 2783: 1, 2782: 1,
2774: 1, 2772: 1, 2771: 1, 2768: 1, 2766: 1, 2763: 1, 2759: 1, 2758: 1, 2755: 1, 2740: 1, 2732: 1,
2727: 1, 2724: 1, 2723: 1, 2720: 1, 2691: 1, 2690: 1, 2688: 1, 2685: 1, 2679: 1, 2676: 1, 2675: 1,
2656: 1, 2654: 1, 2653: 1, 2651: 1, 2649: 1, 2648: 1, 2646: 1, 2645: 1, 2643: 1, 2635: 1, 2634: 1,
2631: 1, 2630: 1, 2621: 1, 2606: 1, 2599: 1, 2596: 1, 2592: 1, 2587: 1, 2584: 1, 2583: 1, 2582: 1,
2577: 1, 2575: 1, 2574: 1, 2565: 1, 2564: 1, 2557: 1, 2554: 1, 2547: 1, 2538: 1, 2528: 1, 2519: 1,
2518: 1, 2516: 1, 2514: 1, 2511: 1, 2505: 1, 2503: 1, 2502: 1, 2499: 1, 2492: 1, 2490: 1, 2487: 1,
2486: 1, 2480: 1, 2477: 1, 2471: 1, 2468: 1, 2463: 1, 2461: 1, 2455: 1, 2452: 1, 2449: 1, 2445: 1,
2444: 1, 2443: 1, 2439: 1, 2438: 1, 2437: 1, 2434: 1, 2429: 1, 2423: 1, 2421: 1, 2412: 1, 2406: 1,
2405: 1, 2395: 1, 2391: 1, 2388: 1, 2381: 1, 2378: 1, 2371: 1, 2366: 1, 2363: 1, 2346: 1, 2343: 1,
2342: 1, 2341: 1, 2337: 1, 2333: 1, 2332: 1, 2327: 1, 2325: 1, 2324: 1, 2323: 1, 2319: 1, 2315: 1,
2314: 1, 2308: 1, 2287: 1, 2285: 1, 2274: 1, 2271: 1, 2269: 1, 2268: 1, 2263: 1, 2261: 1, 2259: 1,
2257: 1, 2256: 1, 2249: 1, 2248: 1, 2242: 1, 2238: 1, 2235: 1, 2233: 1, 2232: 1, 2231: 1, 2230: 1,
2229: 1, 2228: 1, 2220: 1, 2219: 1, 2214: 1, 2209: 1, 2207: 1, 2206: 1, 2204: 1, 2199: 1, 2197: 1,
2196: 1, 2188: 1, 2184: 1, 2182: 1, 2180: 1, 2179: 1, 2178: 1, 2177: 1, 2172: 1, 2171: 1, 2170: 1,
2169: 1, 2165: 1, 2164: 1, 2162: 1, 2159: 1, 2157: 1, 2153: 1, 2152: 1, 2151: 1, 2138: 1, 2137: 1,
2135: 1, 2129: 1, 2128: 1, 2120: 1, 2111: 1, 2106: 1, 2101: 1, 2099: 1, 2095: 1, 2091: 1, 2089: 1,
2088: 1, 2087: 1, 2085: 1, 2083: 1, 2082: 1, 2080: 1, 2077: 1, 2076: 1, 2072: 1, 2071: 1, 2067: 1,
2066: 1, 2065: 1, 2062: 1, 2061: 1, 2057: 1, 2056: 1, 2053: 1, 2046: 1, 2044: 1, 2043: 1, 2041: 1,
2039: 1, 2032: 1, 2030: 1, 2027: 1, 2024: 1, 2022: 1, 2020: 1, 2019: 1, 2013: 1, 2012: 1, 2006: 1,
2005: 1, 2003: 1, 2002: 1, 1998: 1, 1994: 1, 1992: 1, 1982: 1, 1979: 1, 1978: 1, 1975: 1, 1974: 1,
1972: 1, 1970: 1, 1969: 1, 1968: 1, 1966: 1, 1963: 1, 1962: 1, 1959: 1, 1958: 1, 1956: 1, 1947: 1,
1945: 1, 1943: 1, 1942: 1, 1941: 1, 1934: 1, 1930: 1, 1926: 1, 1919: 1, 1917: 1, 1910: 1, 1906: 1,
1900: 1, 1898: 1, 1892: 1, 1890: 1, 1889: 1, 1886: 1, 1885: 1, 1877: 1, 1871: 1, 1865: 1, 1863: 1,
1859: 1, 1855: 1, 1847: 1, 1844: 1, 1839: 1, 1837: 1, 1831: 1, 1830: 1, 1826: 1, 1823: 1, 1817: 1,
1814: 1, 1813: 1, 1812: 1, 1807: 1, 1806: 1, 1805: 1, 1801: 1, 1798: 1, 1797: 1, 1795: 1, 1793: 1,
1792: 1, 1791: 1, 1780: 1, 1779: 1, 1778: 1, 1777: 1, 1776: 1, 1770: 1, 1768: 1, 1766: 1, 1765: 1,
1762: 1, 1758: 1, 1755: 1, 1753: 1, 1752: 1, 1748: 1, 1747: 1, 1746: 1, 1745: 1, 1741: 1, 1739: 1,
1736: 1, 1735: 1, 1729: 1, 1724: 1, 1720: 1, 1719: 1, 1717: 1, 1713: 1, 1712: 1, 1710: 1, 1708: 1,
1707: 1, 1706: 1, 1702: 1, 1696: 1, 1695: 1, 1693: 1, 1691: 1, 1689: 1, 1688: 1, 1679: 1, 1675: 1,
1672: 1, 1661: 1, 1660: 1, 1659: 1, 1655: 1, 1650: 1, 1649: 1, 1642: 1, 1641: 1, 1640: 1, 1639: 1,
1638: 1, 1636: 1, 1632: 1, 1628: 1, 1627: 1, 1623: 1, 1621: 1, 1617: 1, 1612: 1, 1607: 1, 1602: 1,
1601: 1, 1600: 1, 1592: 1, 1582: 1, 1581: 1, 1580: 1, 1578: 1, 1577: 1, 1575: 1, 1573: 1, 1572: 1,
1571: 1, 1570: 1, 1567: 1, 1566: 1, 1560: 1, 1559: 1, 1556: 1, 1555: 1, 1544: 1, 1542: 1, 1540: 1,
1536: 1, 1535: 1, 1534: 1, 1529: 1, 1515: 1, 1505: 1, 1501: 1, 1496: 1, 1493: 1, 1492: 1, 1491: 1,
1488: 1, 1485: 1, 1484: 1, 1483: 1, 1481: 1, 1479: 1, 1477: 1, 1474: 1, 1473: 1, 1466: 1, 1465: 1,
1464: 1, 1459: 1, 1458: 1, 1448: 1, 1444: 1, 1441: 1, 1440: 1, 1438: 1, 1437: 1, 1431: 1, 1430: 1,
1428: 1, 1427: 1, 1426: 1, 1424: 1, 1422: 1, 1417: 1, 1414: 1, 1413: 1, 1412: 1, 1410: 1, 1404: 1,
1398: 1, 1397: 1, 1391: 1, 1387: 1, 1385: 1, 1384: 1, 1383: 1, 1382: 1, 1381: 1, 1376: 1, 1375: 1,
1368: 1, 1367: 1, 1366: 1, 1365: 1, 1354: 1, 1353: 1, 1351: 1, 1348: 1, 1347: 1, 1346: 1, 1342: 1,
1337: 1, 1335: 1, 1333: 1, 1331: 1, 1329: 1, 1327: 1, 1326: 1, 1325: 1, 1322: 1, 1319: 1, 1317: 1,
1316: 1, 1313: 1, 1310: 1, 1301: 1, 1300: 1, 1297: 1, 1295: 1, 1294: 1, 1293: 1, 1287: 1, 1282: 1,
1279: 1, 1277: 1, 1276: 1, 1272: 1, 1270: 1, 1268: 1, 1267: 1, 1266: 1, 1258: 1, 1255: 1, 1254: 1,
1252: 1, 1250: 1, 1247: 1, 1243: 1, 1242: 1, 1241: 1, 1237: 1, 1236: 1, 1233: 1, 1231: 1, 1229: 1,
1222: 1, 1218: 1, 1210: 1, 1202: 1, 1197: 1, 1194: 1, 1192: 1, 1191: 1, 1188: 1, 1186: 1, 1183: 1,
1181: 1, 1175: 1, 1169: 1, 1168: 1, 1166: 1, 1165: 1, 1161: 1, 1149: 1, 1148: 1, 1146: 1, 1144: 1,
1143: 1, 1142: 1, 1141: 1, 1140: 1, 1139: 1, 1138: 1, 1134: 1, 1133: 1, 1132: 1, 1129: 1, 1126: 1,
1123: 1, 1116: 1, 1111: 1, 1110: 1, 1098: 1, 1094: 1, 1093: 1, 1087: 1, 1083: 1, 1080: 1, 1075: 1,
1073: 1, 1069: 1, 1066: 1, 1065: 1, 1062: 1, 1059: 1, 1056: 1, 1055: 1, 1054: 1, 1053: 1, 1050: 1,
1049: 1, 1040: 1, 1035: 1, 1034: 1, 1031: 1, 1030: 1, 1027: 1, 1026: 1, 1025: 1, 1024: 1, 1019: 1,
1017: 1, 1015: 1, 1014: 1, 1011: 1, 1005: 1, 1003: 1, 1002: 1, 998: 1, 992: 1, 988: 1, 985: 1, 983:
1, 982: 1, 981: 1, 978: 1, 976: 1, 971: 1, 963: 1, 960: 1, 954: 1, 950: 1, 949: 1, 944: 1, 937: 1,
929: 1, 920: 1, 915: 1, 912: 1, 910: 1, 908: 1, 902: 1, 901: 1, 899: 1, 895: 1, 894: 1, 886: 1,
884: 1, 883: 1, 881: 1, 877: 1, 869: 1, 868: 1, 866: 1, 862: 1, 859: 1, 848: 1, 845: 1, 843: 1,
838: 1, 835: 1, 828: 1, 816: 1, 813: 1, 812: 1, 808: 1, 802: 1, 800: 1, 796: 1, 795: 1, 792: 1,
786: 1, 784: 1, 782: 1, 776: 1, 774: 1, 773: 1, 765: 1, 761: 1, 757: 1, 756: 1, 755: 1, 753: 1,
736: 1, 735: 1, 732: 1, 731: 1, 730: 1, 728: 1, 723: 1, 720: 1, 714: 1, 711: 1, 710: 1, 701: 1,
699: 1, 689: 1, 688: 1, 684: 1, 681: 1, 680: 1, 677: 1, 675: 1, 671: 1, 667: 1, 662: 1, 653: 1,
651: 1, 648: 1, 647: 1, 636: 1, 628: 1, 607: 1, 606: 1, 593: 1, 586: 1, 582: 1, 577: 1, 572: 1,
561: 1, 557: 1, 549: 1, 547: 1, 542: 1, 538: 1, 523: 1, 474: 1, 384: 1})
```

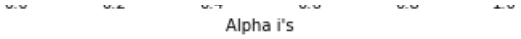
In [184]:

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

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
```

```
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=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.clas
ses_, eps=1e-15))

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.2922541945829085
For values of alpha =  0.0001 The log loss is: 1.1542268338881476
For values of alpha =  0.001 The log loss is: 1.1341464216868673
For values of alpha =  0.01 The log loss is: 1.229662594818684
For values of alpha =  0.1 The log loss is: 1.479781409115675
For values of alpha =  1 The log loss is: 1.6880482470315228
```






```
For values of best alpha =  0.001 The train log loss is: 0.6685825372825236
For values of best alpha =  0.001 The cross validation log loss is: 1.1341464216868673
For values of best alpha =  0.001 The test log loss is: 1.2484046737170897
```

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

Ans. Yes, it seems like!

In [0]:

```
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 [186]:

```
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")
```

```
32.26 % of word of test data appeared in train data
37.106 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [0]:

```
#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 [0]:

```
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 [0]:

```
# 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_imp_feature_names(indices, text, gene, var, no_features, text_vectorizer = None, features =
None):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    if(text_vectorizer == 'bag of words'):
      text_count_vec = CountVectorizer(min_df=3)
    elif(text_vectorizer == 'tf-idf'):
      text_count_vec = TfidfVectorizer(max_features = 1000)

    gene_vec = gene_count_vec.fit(train_df['Gene'])
    var_vec  = var_count_vec.fit(train_df['Variation'])
    text_vec = text_count_vec.fit(train_df['TEXT'])

    fea1_len = len(gene_vec.get_feature_names())
    fea2_len = len(var_count_vec.get_feature_names())

    word_present = 0
    for i,v in enumerate(indices):
        if (v < fea1_len):
          if(features is None):
            word = gene_vec.get_feature_names()[v]
          else:
            word = features[v]
            yes_no = True if word == gene else False
            if yes_no:
                word_present += 1
                print(i, "Gene feature [{}] present in test data point [{}]".format(word,yes_no))
        elif (v < fea1_len+fea2_len):
          if(features is None):
            word = var_vec.get_feature_names()[v-(fea1_len)]
          else:
            word = features[v]
            yes_no = True if word == var else False
            if yes_no:
                word_present += 1
                print(i, "variation feature [{}] present in test data point [{}]".format(word,yes_r
o))
        else:
          if(features is None):
            word = text_vec.get_feature_names()[v-(fea1_len+fea2_len)]
          else:
            word = features[v]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
                print(i, "Text feature [{}] present in test data point [{}]".format(word,yes_no))

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

Stacking all possible types of features

In [0]:

```
# 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)).tocs
r()
train_x_onehotCoding_bigrams = hstack((train_gene_var_onehotCoding,
```

```
train_text_feature_onehotCoding_bigrams)).tocsr()
train_x_onehotCoding_tfidf = hstack((train_gene_var_onehotCoding, train_text_feature_tfidf)).tocsr
()
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_onehotCoding)).tocsr()
test_x_onehotCoding_bigrams = hstack((test_gene_var_onehotCoding,
test_text_feature_onehotCoding_bigrams)).tocsr()
test_x_onehotCoding_tfidf = hstack((test_gene_var_onehotCoding, test_text_feature_tfidf)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).tocsr()
cv_x_onehotCoding_bigrams = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding_bigrams
)).tocsr()
cv_x_onehotCoding_tfidf = hstack((cv_gene_var_onehotCoding, cv_text_feature_tfidf)).tocsr()
cv_y = np.array(list(cv_df['Class']))


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

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

train_x_responseCoding_tfidf = hstack((train_gene_var_responseCoding, train_text_feature_tfidf)).t
ocsr()
test_x_responseCoding_tfidf = hstack((test_gene_var_responseCoding, test_text_feature_tfidf)).tocs
r()
cv_x_responseCoding_tfidf = hstack((cv_gene_var_responseCoding, cv_text_feature_tfidf)).tocsr()
```

Selecting best features from total features

In [0]:

```
features = gene_vectorizer.get_feature_names() + variation_vectorizer.get_feature_names() +
text_vectorizer_tfidf.get_feature_names()
selecting_vectorizer = SelectKBest(chi2, k = 2500).fit(train_x_onehotCoding_tfidf, train_y)
train_x_onehotCoding_tfidf  = selecting_vectorizer.transform(train_x_onehotCoding_tfidf)
cv_x_onehotCoding_tfidf = selecting_vectorizer.transform(cv_x_onehotCoding_tfidf)
test_x_onehotCoding_tfidf = selecting_vectorizer.transform(test_x_onehotCoding_tfidf)
selected_features_indices_tfidf = selecting_vectorizer.get_support(indices = True);
selected_features_tfidf = np.array(features)[selected_features_indices_tfidf]
```

In [0]:

```
features = gene_vectorizer.get_feature_names() + variation_vectorizer.get_feature_names() +
text_vectorizer_bigrams.get_feature_names()
selecting_vectorizer = SelectKBest(chi2, k = 8000).fit(train_x_onehotCoding_bigrams, train_y)
train_x_onehotCoding_bigrams  = selecting_vectorizer.transform(train_x_onehotCoding_bigrams)
cv_x_onehotCoding_bigrams = selecting_vectorizer.transform(cv_x_onehotCoding_bigrams)
test_x_onehotCoding_bigrams = selecting_vectorizer.transform(test_x_onehotCoding_bigrams)
selected_features_indices_bigrams = selecting_vectorizer.get_support(indices = True);
selected_features_bigrams = np.array(features)[selected_features_indices_bigrams]
```

In [196]:

```
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, 55226)
(number of data points * number of features) in test data =  (665, 55226)
```

```
(number of data points * number of features) in cross validation data = (532, 55226)
```

In [197]:

```
print("One hot encoding(text-bigrams) features :")
print("(number of data points * number of features) in train data = ",
train_x_onehotCoding_bigrams.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding_bigrams.
shape)
print("(number of data points * number of features) in cross validation data =",
cv_x_onehotCoding_bigrams.shape)
```

```
One hot encoding(text-bigrams) features :
(number of data points * number of features) in train data =  (2124, 8000)
(number of data points * number of features) in test data =  (665, 8000)
(number of data points * number of features) in cross validation data = (532, 8000)
```

In [198]:

```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCoding.shap
e)
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)
```

In [199]:

```
print("One hot encoding and text tf-idf features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding_tfidf.
shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding_tfidf.sh
ape)
print("(number of data points * number of features) in cross validation data =",
cv_x_onehotCoding_tfidf.shape)
```

```
One hot encoding and text tf-idf features :
(number of data points * number of features) in train data =  (2124, 2500)
(number of data points * number of features) in test data =  (665, 2500)
(number of data points * number of features) in cross validation data = (532, 2500)
```

In [200]:

```
print("Response coding and text tf-idf features :")
print("(number of data points * number of features) in train data = ",
train_x_responseCoding_tfidf.shape)
print("(number of data points * number of features) in test data = ", test_x_responseCoding_tfidf.
shape)
print("(number of data points * number of features) in cross validation data =",
cv_x_responseCoding_tfidf.shape)
```

```
Response coding and text tf-idf features :
(number of data points * number of features) in train data =  (2124, 10018)
(number of data points * number of features) in test data =  (665, 10018)
(number of data points * number of features) in cross validation data = (532, 10018)
```

4.1. Base Line Model

4.1.1. Naive Bayes

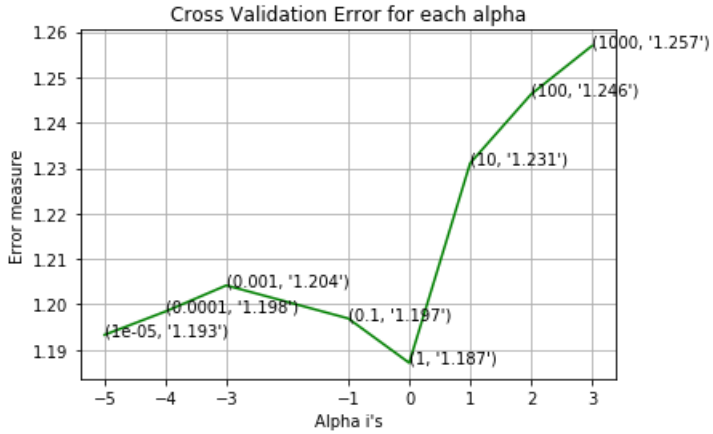
4.1.1.1. Hyper parameter tuning

Hyper parameter tuning

In [202]:

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

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


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


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

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


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


predict_y = sig_clf.predict_proba(train_x_onehotCoding_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-05
Log Loss : 1.193287134681665
for alpha = 0.0001
Log Loss : 1.1984327150422425
for alpha = 0.001
Log Loss : 1.2041809213417811
for alpha = 0.1
Log Loss : 1.1969011405896488
for alpha = 1
Log Loss : 1.1871286002220895
for alpha = 10
Log Loss : 1.2310793353342964
for alpha = 100
Log Loss : 1.2462540578985362
for alpha = 1000
Log Loss : 1.2569322252685253
```




```
For values of best alpha =  1 The train log loss is: 0.9314902323432036
For values of best alpha =  1 The cross validation log loss is: 1.1871286002220895
For values of best alpha =  1 The test log loss is: 1.239769865795945
```

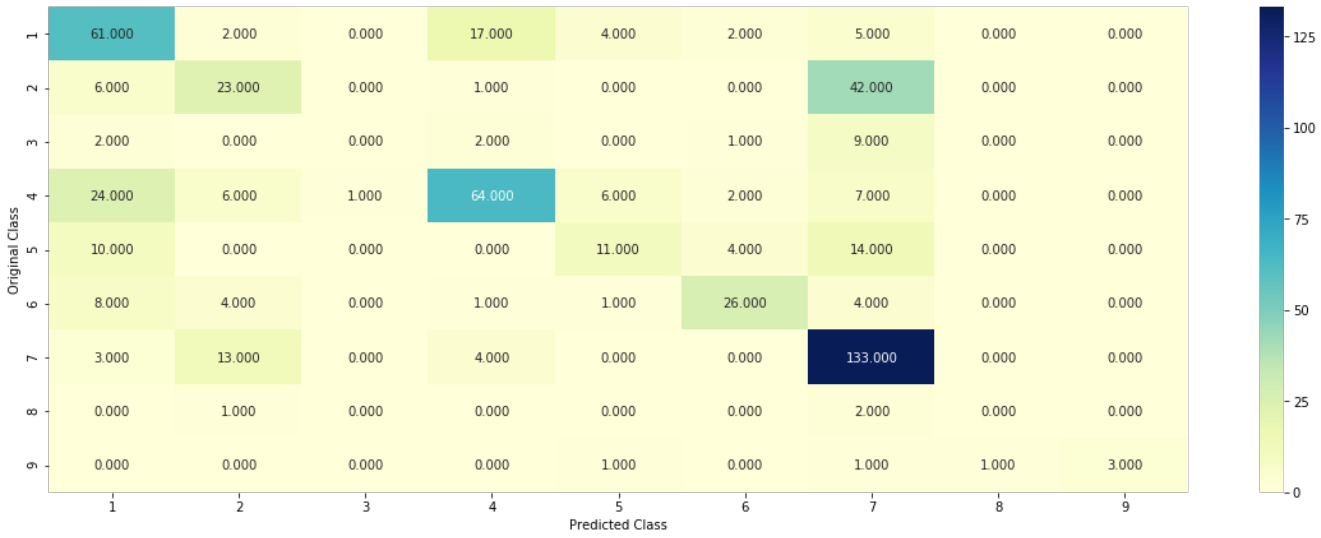
4.1.1.2. Testing the model with best hyper paramters

In [203]:

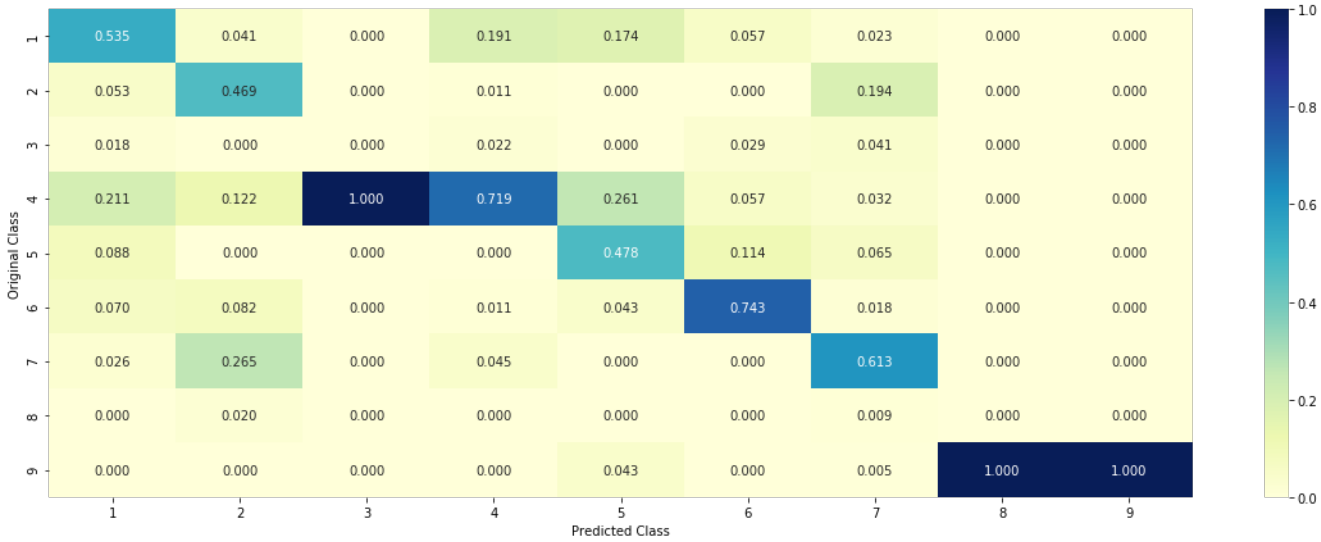
```
# find more about Multinomial Naive base function here http://scikit-
learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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

clf = MultinomialNB(alpha=alpha[best_alpha])
```

```
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding_tfidf, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding_tfidf)
# to avoid rounding error while multiplying probabilites we use log-probability estimates
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :",
np.count_nonzero((sig_clf.predict(cv_x_onehotCoding_tfidf)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding_tfidf.toarray()))
```

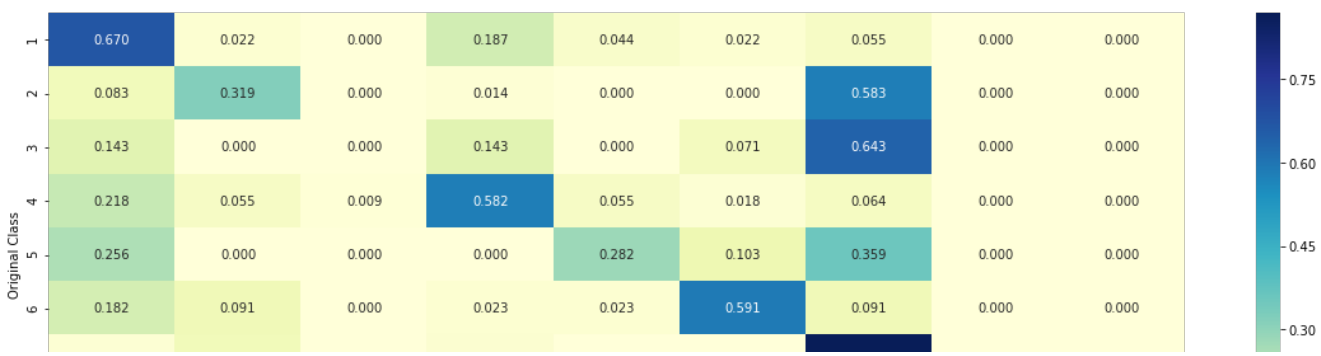
```
Log Loss : 1.1871286002220895
Number of missclassified point : 0.3966165413533835
-------------------- Confusion matrix --------------------
```

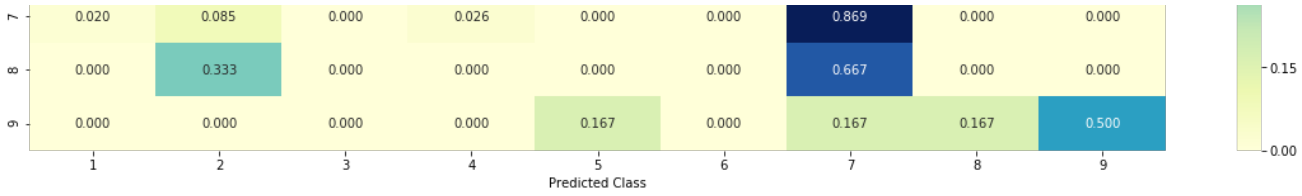



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





4.1.1.3. Feature Importance, Correctly classified point

In [209]:

```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding_tfidf[t
est_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imp_feature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature, text_vectorizer = 'tf-idf', features =
selected_features_tfidf)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.346  0.0383 0.0207 0.4909 0.0437 0.035  0.0176 0.0041 0.0037]]
Actual Class : 4
--------------------------------------------------
34 Text feature [suppressor] present in test data point [True]
44 Text feature [yeast] present in test data point [True]
82 Text feature [yeasts] present in test data point [True]
Out of the top  100  features  3 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [210]:

```
test_point_index = 44
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding_tfidf[t
est_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imp_feature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature, text_vectorizer = 'tf-idf', features =
selected_features_tfidf)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0351 0.0575 0.0208 0.0574 0.044  0.0351 0.7421 0.0042 0.0038]]
Actual Class : 6
--------------------------------------------------
20 Text feature [tyrosine] present in test data point [True]
21 Text feature [treated] present in test data point [True]
22 Text feature [treatment] present in test data point [True]
31 Text feature [trials] present in test data point [True]
41 Text feature [therapy] present in test data point [True]
46 Text feature [trial] present in test data point [True]
Out of the top  100  features  6 are present in query point
```

4.2. K Nearest Neighbour Classification

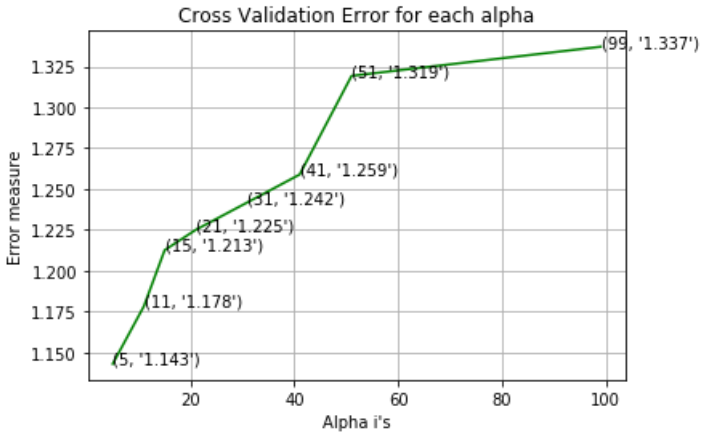
4.2.1. Hyper parameter tuning

4.2.1. Hyper parameter tuning

In [211]:

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

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


# find more about CalibratedClassifierCV here at http://scikit-
learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=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_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_responseCoding_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding_tfidf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

predict_y = sig_clf.predict_proba(train_x_responseCoding_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
for alpha = 5
Log Loss : 1.142976579765318
for alpha = 11
Log Loss : 1.1782121496025155
for alpha = 15
Log Loss : 1.2125702283717694
for alpha = 21
Log Loss : 1.2250233288098236
for alpha = 31
Log Loss : 1.2419131943338444
for alpha = 41
Log Loss : 1.2588636296085611
for alpha = 51
Log Loss : 1.319266683539817
for alpha = 99
Log Loss : 1.337093664360213
```



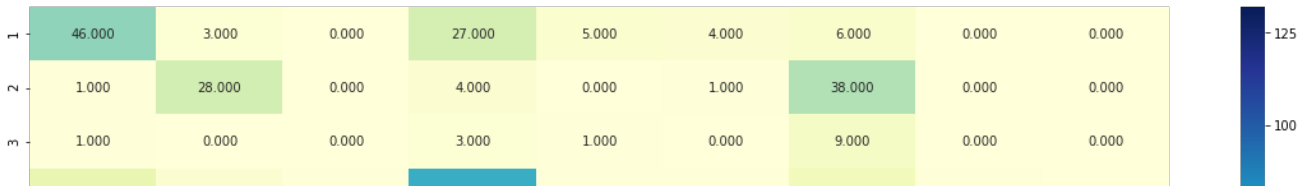

```
For values of best alpha =  5 The train log loss is: 0.8628220610241009
For values of best alpha =  5 The cross validation log loss is: 1.142976579765318
For values of best alpha =  5 The test log loss is: 1.259241306147807
```

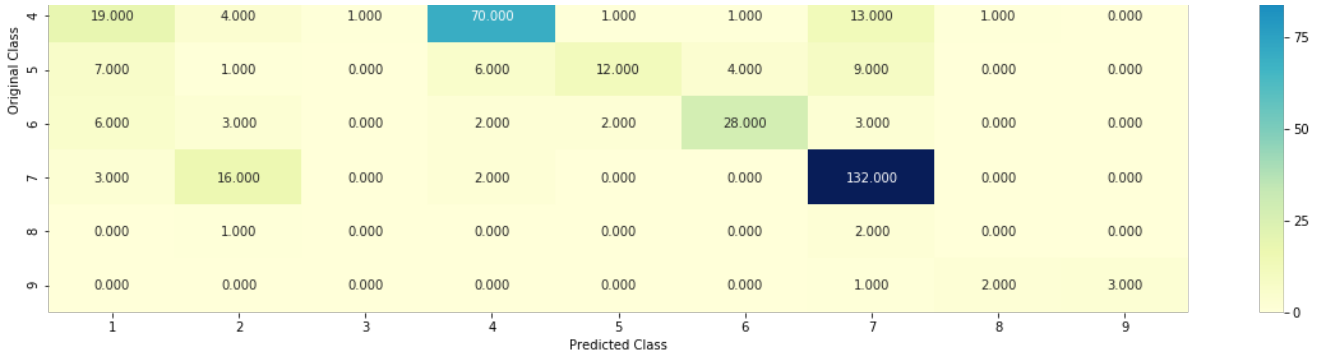
4.2.2. Testing the model with best hyper paramters

In [212]:

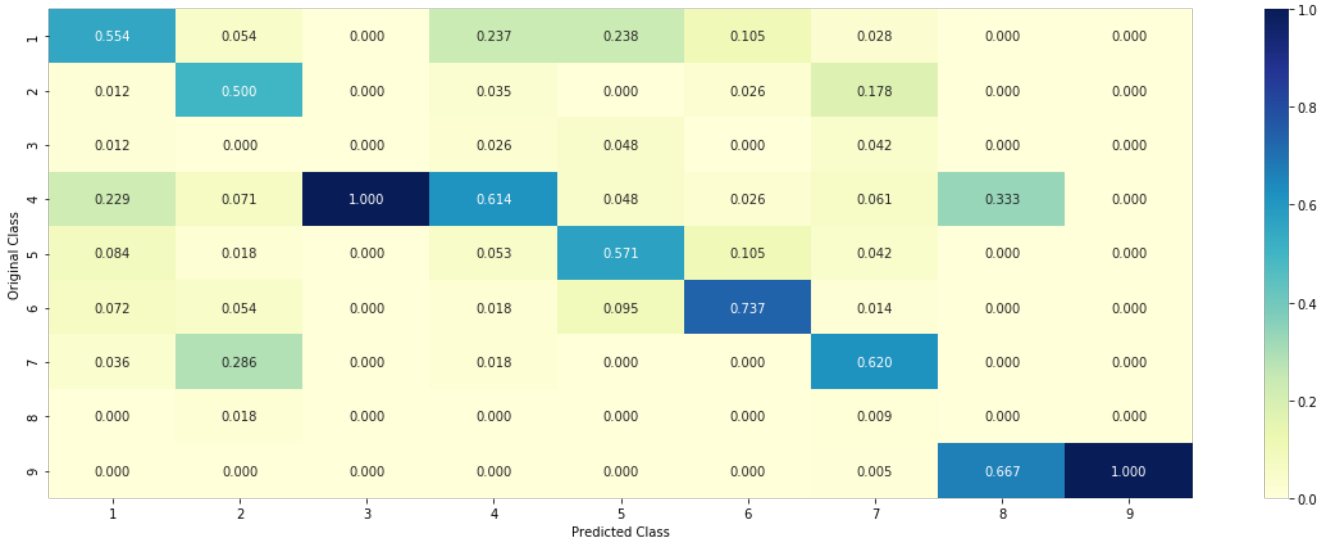
```
# find more about KNeighborsClassifier() here http://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski',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

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

```
Log loss : 1.142976579765318
Number of mis-classified points : 0.40037593984962405
-------------------- Confusion matrix --------------------
```

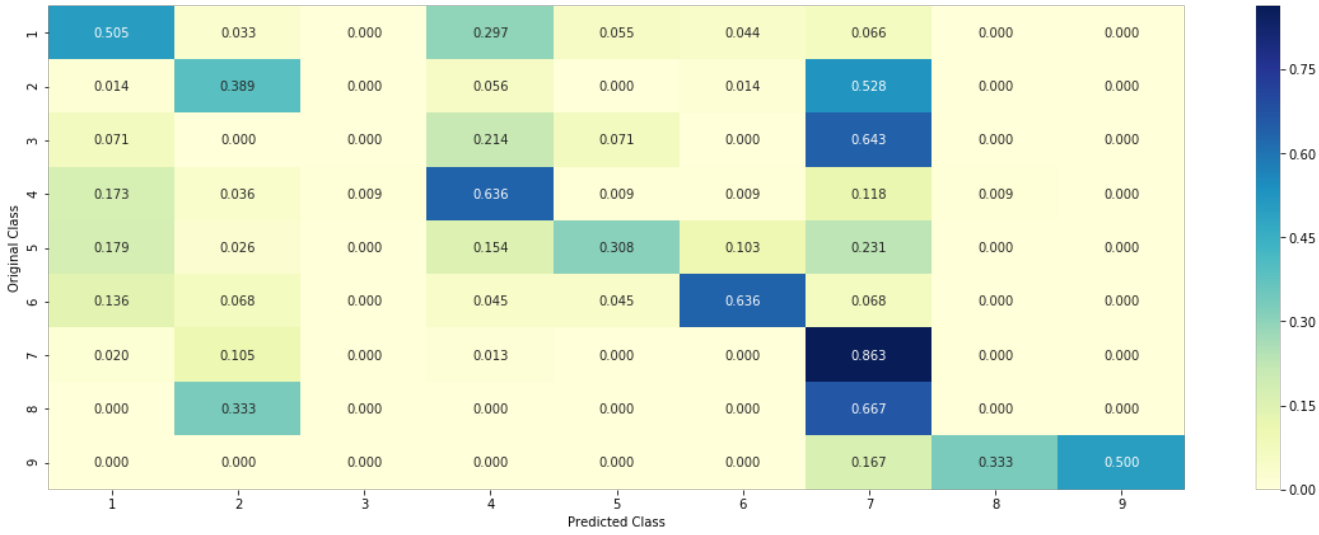




---------- Precision matrix (Columm Sum=1) ----------



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



4.2.3.Sample Query point -1

In [213]:

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

test_point_index = 0
predicted_cls = sig_clf.predict(test_x_responseCoding_tfidf[0].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
```

```
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding_tfidf[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 : 7
Actual Class : 2
The  5  nearest neighbours of the test points belongs to classes [7 7 7 7 7]
Frequency of nearest points : Counter({7: 5})
```

4.2.4. Sample Query Point-2

In [214]:

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

test_point_index = 5

predicted_cls = sig_clf.predict(test_x_responseCoding_tfidf[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_tfidf[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 be
longs to classes",train_y[neighbors[1][0]])
print("Frequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 2
Actual Class : 2
the k value for knn is 5 and the nearest neighbours of the test points belongs to classes [2 2 7 2
6]
Frequency of nearest points : Counter({2: 3, 7: 1, 6: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [0]:

```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=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-in
tuition-1/
#-------------------------------


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

```
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=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:
#-------------------------------------

tuned_parameters = {'alpha': [10 ** x for x in range(-6, 3)], 'penalty': ['l1', 'l2']}

clf = SGDClassifier(class_weight='balanced', loss='log', random_state=42)
# sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
grid_search_clf = GridSearchCV(clf, tuned_parameters, cv = 5);
grid_search_clf.fit(train_x_onehotCoding_tfidf, train_y)

print("Best hyper parameters: ", grid_search_clf.best_params_)

clf = grid_search_clf.best_estimator_
clf.fit(train_x_onehotCoding_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding_tfidf, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding_tfidf)
print('For values of best alpha = ', grid_search_clf.best_params_['alpha'], "The train log loss is
:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding_tfidf)
print('For values of best alpha = ', grid_search_clf.best_params_['alpha'], "The cross validation
log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding_tfidf)
print('For values of best alpha = ', grid_search_clf.best_params_['alpha'], "The test log loss is:
",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
Best hyper parameters:  {'alpha': 0.0001, 'penalty': 'l2'}
For values of best alpha =  0.0001 The train log loss is: 0.5600723800807138
For values of best alpha =  0.0001 The cross validation log loss is: 0.980519584054448
For values of best alpha =  0.0001 The test log loss is: 0.9897085513635472
```

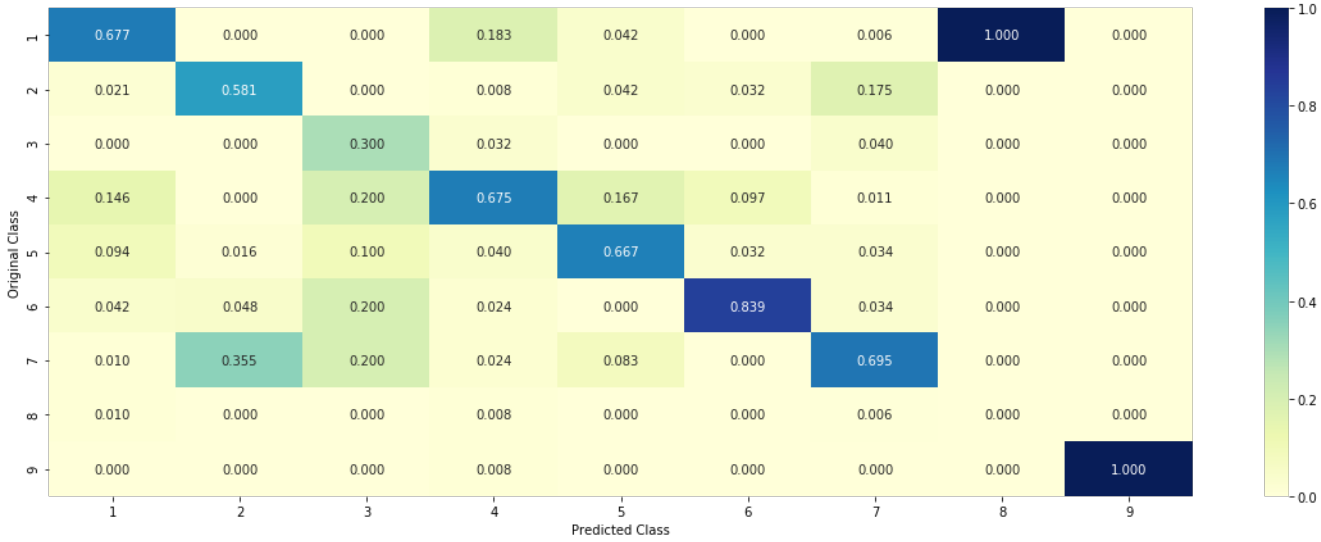
4.3.1.2. Testing the model with best hyper paramters

In [0]:

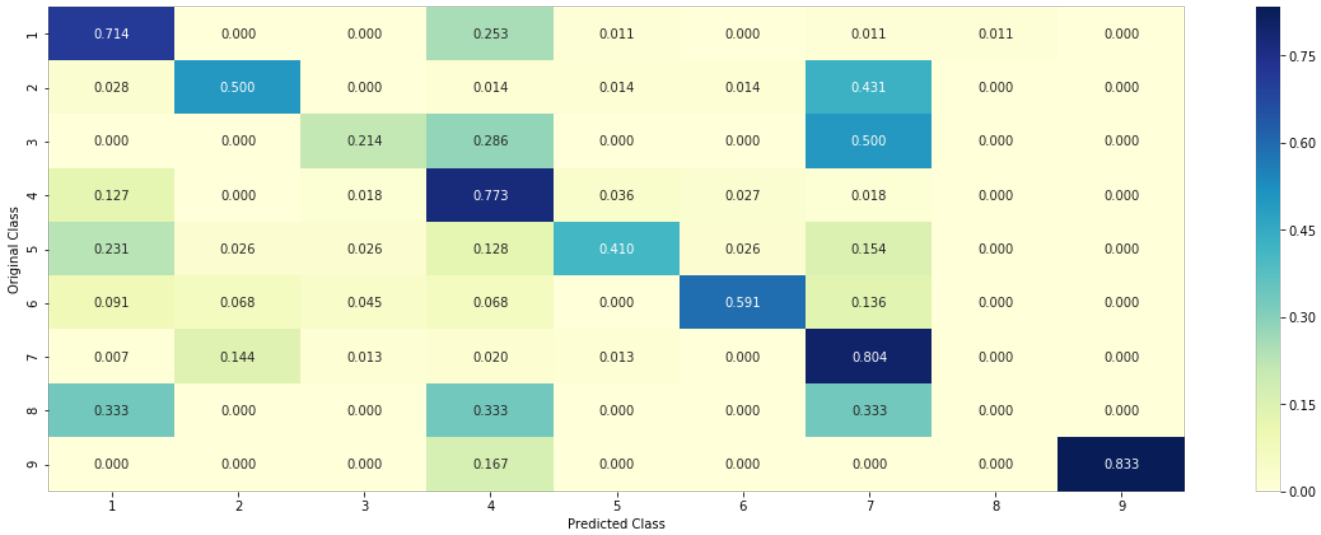
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# 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-in
tuition-1/
#------------------------------
clf = grid_search_clf.best_estimator_
predict_and_plot_confusion_matrix(train_x_onehotCoding_tfidf, train_y, cv_x_onehotCoding_tfidf, cv
_y, clf)
```

```
Log loss : 0.980519584054448
Number of mis-classified points : 0.325187969924812
-------------------- Confusion matrix --------------------
```

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



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



4.3.1.3.1. Correctly Classified point

In [218]:

```
# from tabulate import tabulate
clf = grid_search_clf.best_estimator_
clf.fit(train_x_onehotCoding_tfidf,train_y)
```

```
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding_tfidf[t
est_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imp_feature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature, text_vectorizer = 'tf-idf', features =
selected_features_tfidf)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.4396 0.0137 0.0091 0.5032 0.0127 0.0106 0.0034 0.0044 0.0033]]
Actual Class : 4
--------------------------------------------------
111 Text feature [yeast] present in test data point [True]
117 Text feature [suppressor] present in test data point [True]
184 Text feature [subcellular] present in test data point [True]
307 Text feature [yeasts] present in test data point [True]
331 Text feature [take] present in test data point [True]
420 Text feature [subtle] present in test data point [True]
498 Text feature [tetramerization] present in test data point [True]
Out of the top  500  features  7 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [219]:

```
# from tabulate import tabulate
clf = grid_search_clf.best_estimator_
clf.fit(train_x_onehotCoding_tfidf,train_y)
test_point_index = 44
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding_tfidf[t
est_point_index]),2))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imp_feature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature, text_vectorizer = 'tf-idf', features =
selected_features_tfidf)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.01 0.14 0.   0.01 0.03 0.01 0.78 0.   0.  ]]
Actual Class : 6
--------------------------------------------------
22 Text feature [upregulate] present in test data point [True]
76 Text feature [tyrosine] present in test data point [True]
200 Text feature [trial] present in test data point [True]
299 Text feature [therapy] present in test data point [True]
364 Text feature [treated] present in test data point [True]
379 Text feature [trials] present in test data point [True]
418 Text feature [subdivided] present in test data point [True]
Out of the top  500  features  7 are present in query point
```

4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

In [226]:

```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
```

```
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=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-in
tuition-1/
#-------------------------------


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

tuned_parameters = {'alpha': [10 ** x for x in range(-6, 3)], 'penalty': ['l1', 'l2']}

clf = SGDClassifier(loss='log', random_state=42)
# sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
grid_search_clf = GridSearchCV(clf, tuned_parameters, cv = 5);
grid_search_clf.fit(train_x_onehotCoding_tfidf, train_y)

print("Best hyper parameters: ", grid_search_clf.best_params_)

clf = grid_search_clf.best_estimator_
clf.fit(train_x_onehotCoding_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding_tfidf, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding_tfidf)
print('For values of best alpha = ', grid_search_clf.best_params_['alpha'], "The train log loss is
:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding_tfidf)
print('For values of best alpha = ', grid_search_clf.best_params_['alpha'], "The cross validation
log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding_tfidf)
print('For values of best alpha = ', grid_search_clf.best_params_['alpha'], "The test log loss is:
",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
Best hyper parameters:  {'alpha': 0.0001, 'penalty': 'l2'}
For values of best alpha =  0.0001 The train log loss is: 0.5138000938877464
For values of best alpha =  0.0001 The cross validation log loss is: 1.0219577843153778
For values of best alpha =  0.0001 The test log loss is: 1.0785331096126887
```

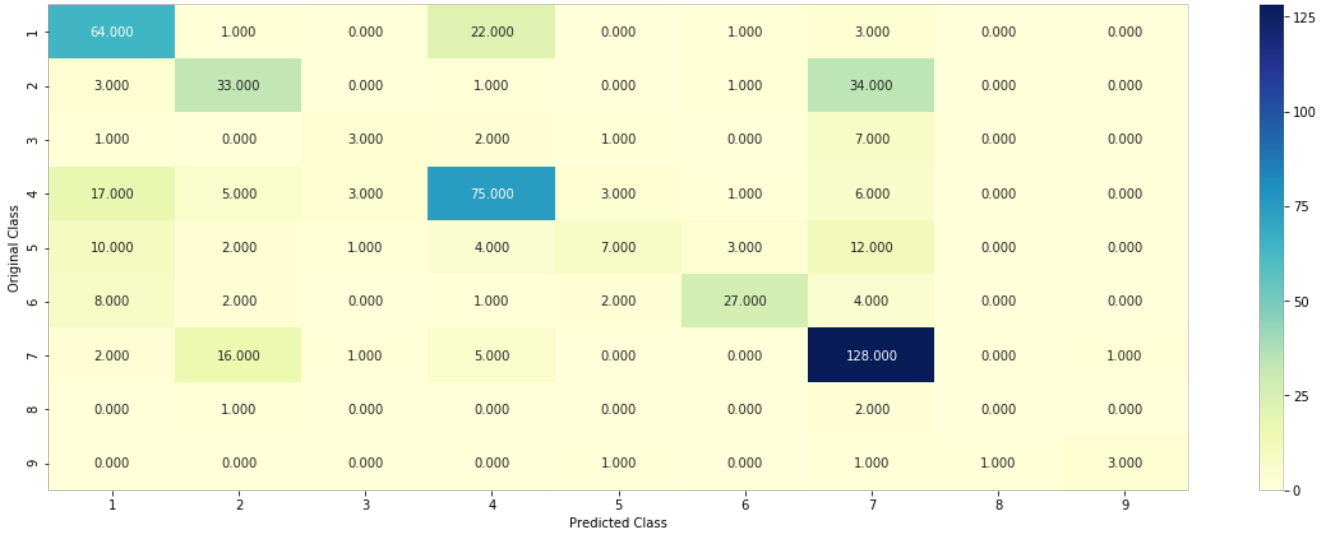
4.3.1.2. Testing the model with best hyper paramters

In [227]:

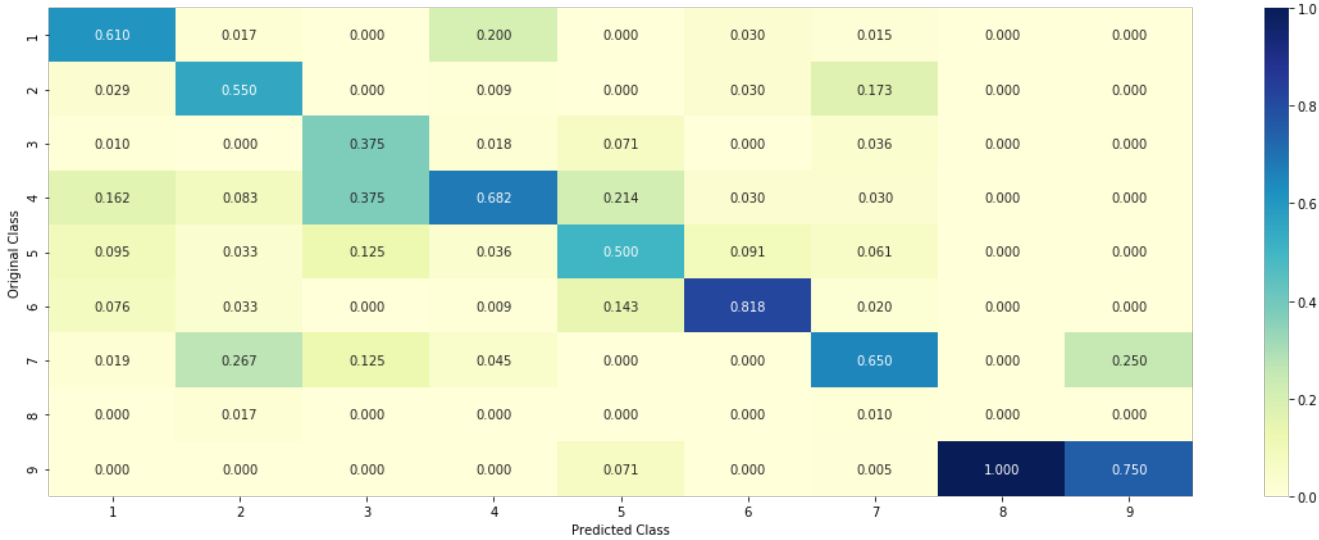
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# 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 = grid_search_clf.best_estimator_
predict_and_plot_confusion_matrix(train_x_onehotCoding_tfidf, train_y, cv_x_onehotCoding_tfidf, cv_y, clf)
```

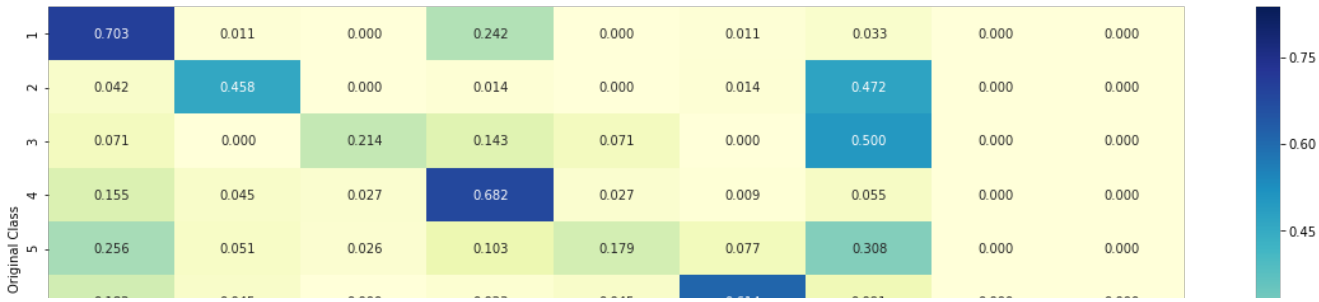
```
Log loss : 1.0219577843153778
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```

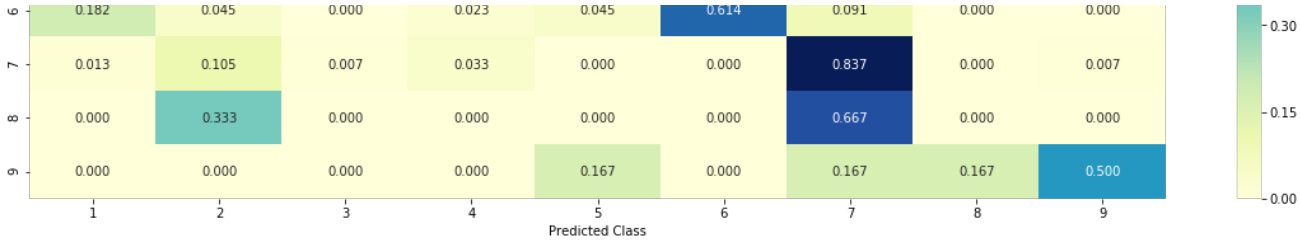



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





4.3.1.3.1. Correctly Classified point

In [228]:

```
# from tabulate import tabulate
clf = grid_search_clf.best_estimator_
clf.fit(train_x_onehotCoding_tfidf,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding_tfidf[t
est_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imp_feature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature, text_vectorizer = 'tf-idf', features =
selected_features_tfidf)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.4335 0.0148 0.0071 0.5159 0.0118 0.0097 0.0035 0.0019 0.0019]]
Actual Class : 4
--------------------------------------------------
114 Text feature [suppressor] present in test data point [True]
122 Text feature [yeast] present in test data point [True]
212 Text feature [subcellular] present in test data point [True]
218 Text feature [yeasts] present in test data point [True]
328 Text feature [take] present in test data point [True]
411 Text feature [tetramerization] present in test data point [True]
498 Text feature [uses] present in test data point [True]
Out of the top  500  features  7 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [229]:

```
# from tabulate import tabulate
clf = grid_search_clf.best_estimator_
clf.fit(train_x_onehotCoding_tfidf,train_y)
test_point_index = 44
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding_tfidf[t
est_point_index]),2))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imp_feature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature, text_vectorizer = 'tf-idf', features =
selected_features_tfidf)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.01 0.15 0.   0.01 0.03 0.01 0.78 0.   0.  ]]
Actual Class : 6
--------------------------------------------------
20 Text feature [upregulate] present in test data point [True]
80 Text feature [tyrosine] present in test data point [True]
217 Text feature [trial] present in test data point [True]
```

```
349 Text feature [therapy] present in test data point [True]
394 Text feature [subdivided] present in test data point [True]
437 Text feature [treated] present in test data point [True]
462 Text feature [trials] present in test data point [True]
492 Text feature [threshold] present in test data point [True]
Out of the top  500  features  8 are present in query point
```

4.3.2 Logistic Regression(Text feature containing uni-grams and bi-grams)

In [230]:

```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=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-in
tuition-1/
#-------------------------------


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

tuned_parameters = {'alpha': [10 ** x for x in range(-6, 3)], 'penalty': ['l1', 'l2']}

clf = SGDClassifier(class_weight='balanced', loss='log', random_state=42)
# sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
grid_search_clf = GridSearchCV(clf, tuned_parameters, cv = 5);
grid_search_clf.fit(train_x_onehotCoding_bigrams, train_y)

print("Best hyper parameters: ", grid_search_clf.best_params_)

clf = grid_search_clf.best_estimator_
clf.fit(train_x_onehotCoding_bigrams, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding_bigrams, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding_bigrams)
print('For values of best alpha = ', grid_search_clf.best_params_['alpha'], "The train log loss is
:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding_bigrams)
print('For values of best alpha = ', grid_search_clf.best_params_['alpha'], "The cross validation
log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding_bigrams)
print('For values of best alpha = ', grid_search_clf.best_params_['alpha'], "The test log loss is:
",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

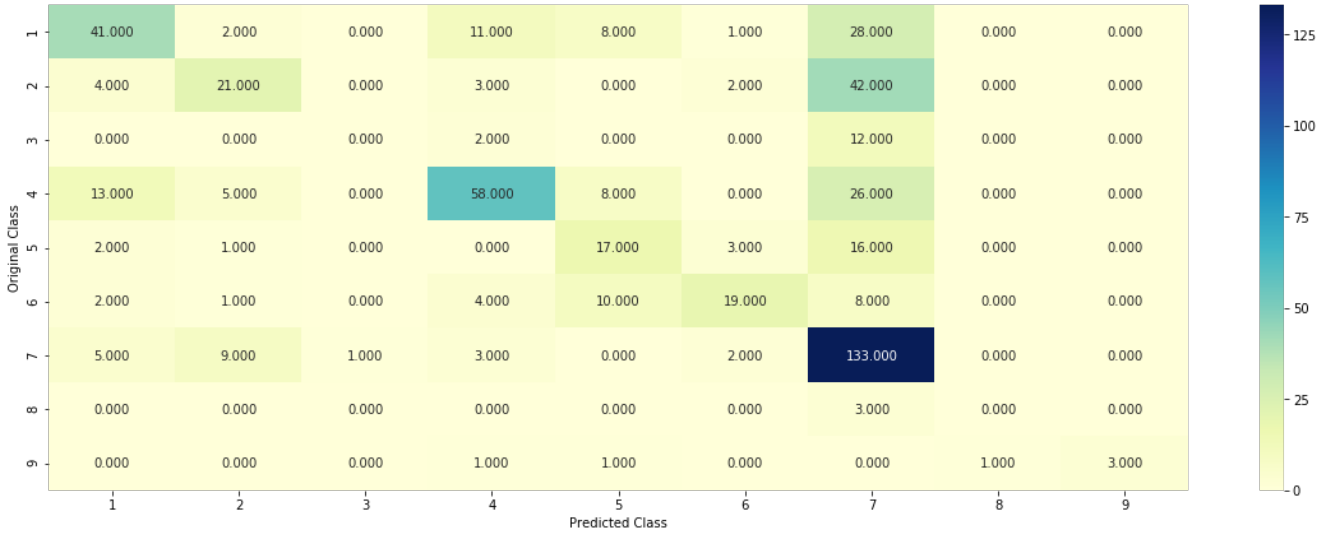
```
Best hyper parameters:  {'alpha': 0.0001, 'penalty': 'l2'}
For values of best alpha =  0.0001 The train log loss is: 0.9804169326243453
For values of best alpha =  0.0001 The cross validation log loss is: 1.2645839978810003
For values of best alpha =  0.0001 The test log loss is: 1.273986111753843
```

4.3.1.2. Testing the model with best hyper paramters

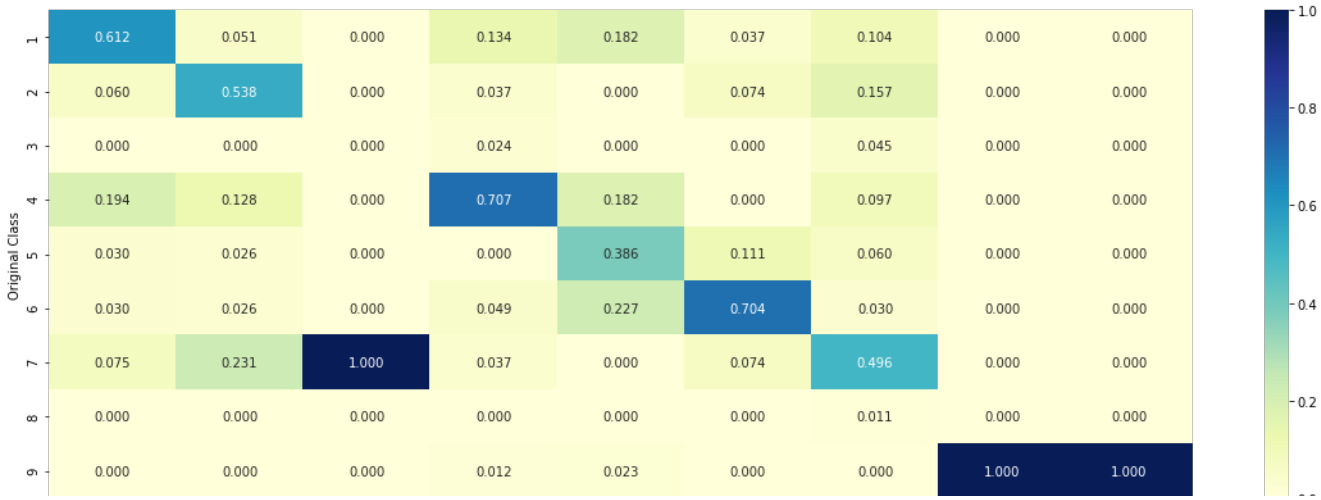
In [231]:

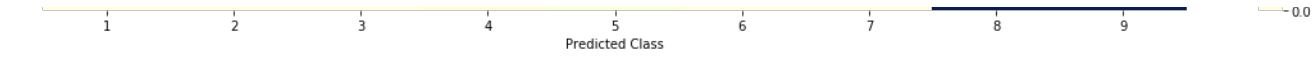
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# 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-in
tuition-1/
#-------------------------------
clf = grid_search_clf.best_estimator_
predict_and_plot_confusion_matrix(train_x_onehotCoding_bigrams, train_y, cv_x_onehotCoding_bigrams
, cv_y, clf)
```

```
Log loss : 1.2645839978810003
Number of mis-classified points : 0.45112781954887216
-------------------- Confusion matrix --------------------
```

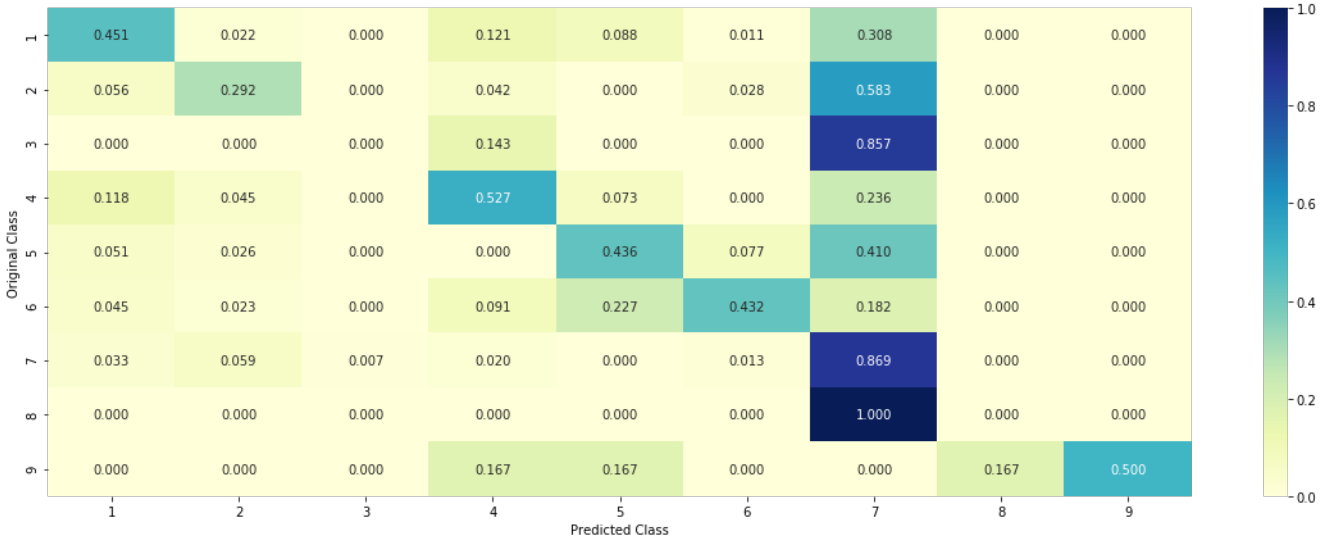



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



4.3.1.3.1. Correctly Classified point

In [232]:

```
# from tabulate import tabulate
clf = grid_search_clf.best_estimator_
clf.fit(train_x_onehotCoding_bigrams,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_bigrams[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding_bigrams
[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imp_feature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature, text_vectorizer = 'bag of words', features =
selected_features_bigrams)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1683 0.03   0.0109 0.7569 0.0057 0.008  0.0057 0.005  0.0094]]
Actual Class : 4
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [233]:

```
# from tabulate import tabulate
clf = grid_search_clf.best_estimator_
clf.fit(train_x_onehotCoding_bigrams,train_y)
test_point_index = 44
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_bigrams[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding_bigrams
[test_point_index]),2))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imp_feature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
```

```
.iloc[test_point_index], no_feature, text_vectorizer = 'bag of words', features =
selected_features_bigrams)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.12 0.11 0.02 0.15 0.05 0.05 0.48 0.   0.01]]
Actual Class : 6
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [234]:

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

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, t
ol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', ra
ndom_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/
# --------------------------------



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

tuned_parameters = {'alpha': [10 ** x for x in range(-5, 3)], 'penalty': ['l1', 'l2']}

clf = SGDClassifier(class_weight = 'balanced', loss='hinge', random_state=42)
# sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
grid_search_clf = GridSearchCV(clf, tuned_parameters, cv = 5);
grid_search_clf.fit(train_x_onehotCoding_tfidf, train_y)

print("Best hyper parameters: ", grid_search_clf.best_params_)

clf = grid_search_clf.best_estimator_
clf.fit(train_x_onehotCoding_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding_tfidf, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding_tfidf)
print('For values of best alpha = ', grid_search_clf.best_params_['alpha'], "The train log loss is
:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding_tfidf)
print('For values of best alpha = ', grid_search_clf.best_params_['alpha'], "The cross validation
log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding_tfidf)
print('For values of best alpha = ', grid_search_clf.best_params_['alpha'], "The test log loss is:
",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
,log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

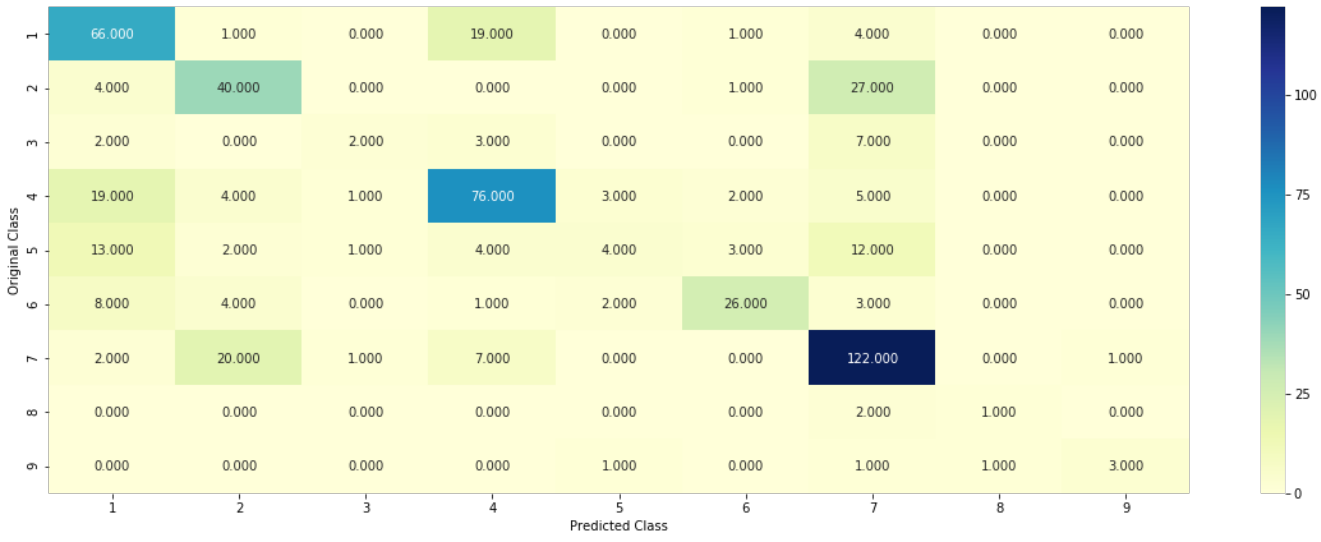
```
Best hyper parameters:  {'alpha': 0.0001, 'penalty': 'l2'}
For values of best alpha =  0.0001 The train log loss is: 0.5786804276002244
For values of best alpha =  0.0001 The cross validation log loss is: 1.1424234017897488
For values of best alpha =  0.0001 The test log loss is: 1.2141933028945058
```

4.4.2. Testing model with best hyper parameters

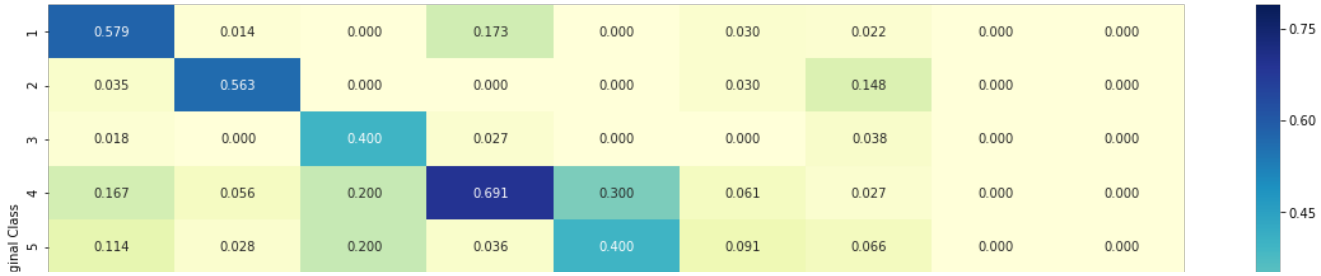
In [235]:

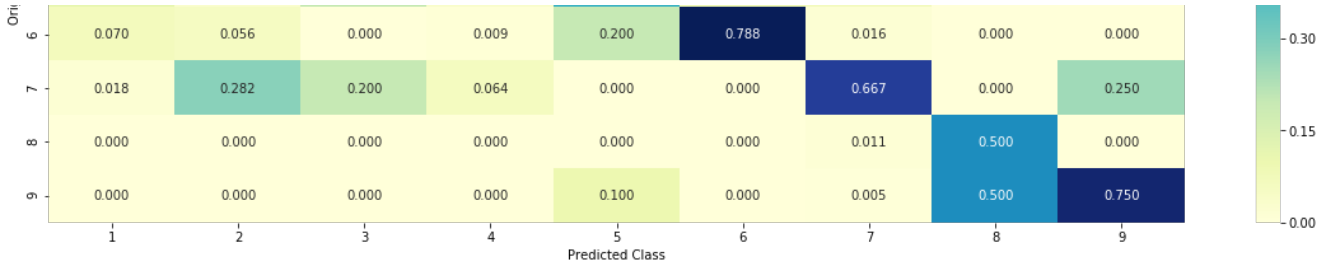
```
# read more about support vector machines with linear kernals here http://scikit-
learn.org/stable/modules/generated/sklearn.svm.SVC.html

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, t
ol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', ra
ndom_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 = grid_search_clf.best_estimator_
predict_and_plot_confusion_matrix(train_x_onehotCoding_tfidf, train_y, cv_x_onehotCoding_tfidf, cv
_y, clf)
```

```
Log loss : 1.1424234017897488
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```

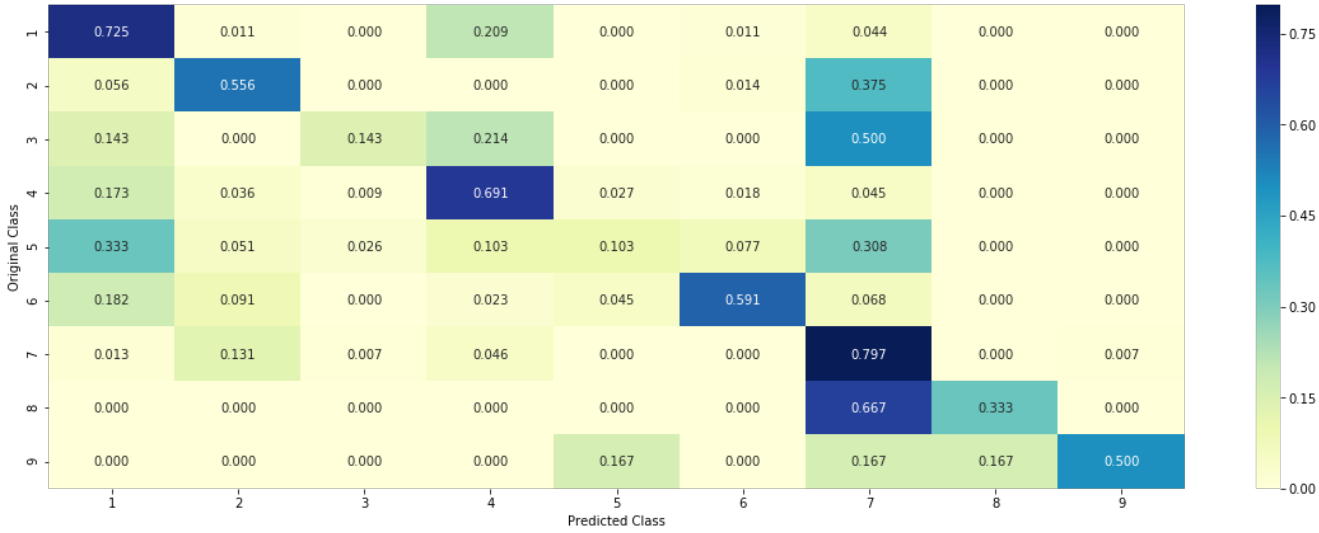



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```






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



4.3.3.1. For Correctly classified point

In [236]:

```
clf = grid_search_clf.best_estimator_
clf.fit(train_x_onehotCoding_tfidf,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding_tfidf[t
est_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imp_feature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature, text_vectorizer = 'tf-idf', features =
selected_features_tfidf)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3597 0.0252 0.0285 0.4535 0.0515 0.0417 0.0296 0.0055 0.0049]]
Actual Class : 4
--------------------------------------------------
96 Text feature [yeast] present in test data point [True]
185 Text feature [suppressor] present in test data point [True]
218 Text feature [therapy] present in test data point [True]
244 Text feature [take] present in test data point [True]
288 Text feature [subtle] present in test data point [True]
309 Text feature [tetramerization] present in test data point [True]
337 Text feature [yeasts] present in test data point [True]
382 Text feature [useful] present in test data point [True]
494 Text feature [uses] present in test data point [True]
Out of the top  500  features  9 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [237]:

```
clf = grid_search_clf.best_estimator_
clf.fit(train_x_onehotCoding_tfidf,train_y)
test_point_index = 3
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding_tfidf[t
est_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imp_feature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature, text_vectorizer = 'tf-idf', features =
selected_features_tfidf)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0677 0.0872 0.0097 0.0496 0.0332 0.0224 0.7219 0.0038 0.0046]]
Actual Class : 7
--------------------------------------------------
21 Text feature [upregulate] present in test data point [True]
97 Text feature [useful] present in test data point [True]
170 Text feature [tyrosine] present in test data point [True]
265 Text feature [vx] present in test data point [True]
342 Text feature [tables] present in test data point [True]
383 Text feature [stata] present in test data point [True]
408 Text feature [treated] present in test data point [True]
416 Text feature [subtle] present in test data point [True]
423 Text feature [therapy] present in test data point [True]
482 Text feature [threshold] present in test data point [True]
Out of the top  500  features  10 are present in query point
```

In [0]:

```
for test_point_index in range(0, 100):
  predicted_cls = sig_clf.predict(test_x_onehotCoding_tfidf[test_point_index])
  actual_cls = test_y[test_point_index]
  if(int(actual_cls) != int(predicted_cls)):
    break;
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [239]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_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-fores
t-and-their-construction-2/
```

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


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

# find more about hyperparameter tuning random forests at
https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-lear
n-28d2aa77dd74
#-----------------------------------------
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 5)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

clf = RandomForestClassifier(criterion='gini', random_state=42, n_jobs=-1)
# sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
random_search_clf = RandomizedSearchCV(estimator = clf, param_distributions = random_grid, cv = 3,
n_iter = 50, verbose = 2, random_state = 42, n_jobs = -1);
random_search_clf.fit(train_x_onehotCoding_tfidf, train_y)
```

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed: 10.8min
[Parallel(n_jobs=-1)]: Done 150 out of 150 | elapsed: 32.5min finished
```

Out[239]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    n_estimators='warn',
                                                    n_jobs=-1, oob_scor...
                   iid='warn', n_iter=50, n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [10, 20, 30, 40, 50, 60,
                                                      70, 80, 90, 100, 110,
                                                      None]
```

```
                                                    None],
                                'max_features': ['auto', 'sqrt'],
                                'min_samples_leaf': [1, 2, 4],
                                'min_samples_split': [2, 5, 10],
                                'n_estimators': [200, 650, 1100, 1550,
                                                 2000]},
                   pre_dispatch='2*n_jobs', random_state=42, refit=True,
                   return_train_score=False, scoring=None, verbose=2)
```

In [240]:

```
print("Best hyper parameters: ", random_search_clf.best_params_)

clf = random_search_clf.best_estimator_
clf.fit(train_x_onehotCoding_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding_tfidf, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding_tfidf)
print("The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding_tfidf)
print("The cross validation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15)
)
predict_y = sig_clf.predict_proba(test_x_onehotCoding_tfidf)
print("The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

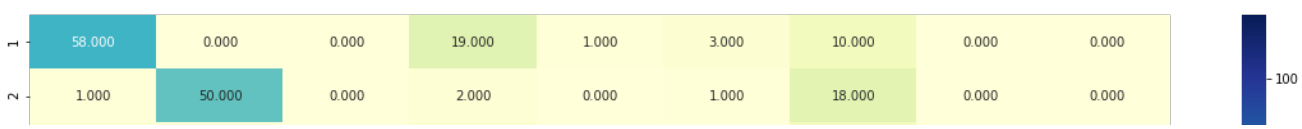
```
Best hyper parameters:  {'n_estimators': 2000, 'min_samples_split': 5, 'min_samples_leaf': 1,
'max_features': 'auto', 'max_depth': None, 'bootstrap': False}
The train log loss is: 0.47213490724777146
The cross validation log loss is: 1.1883918149034804
The test log loss is: 1.215832853740422
```

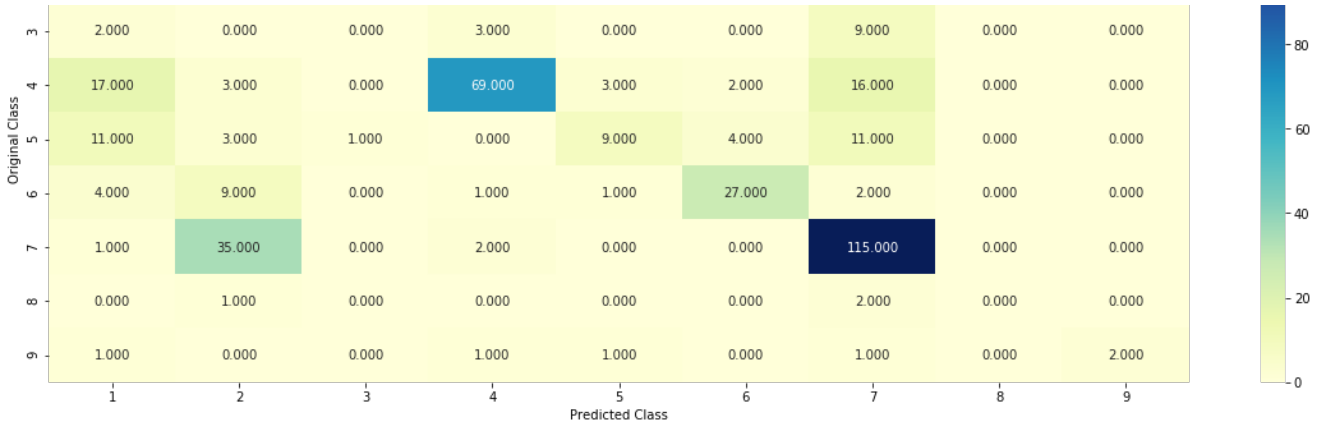
4.5.2. Testing model with best hyper parameters (One Hot Encoding & Tf-Idf)

In [241]:

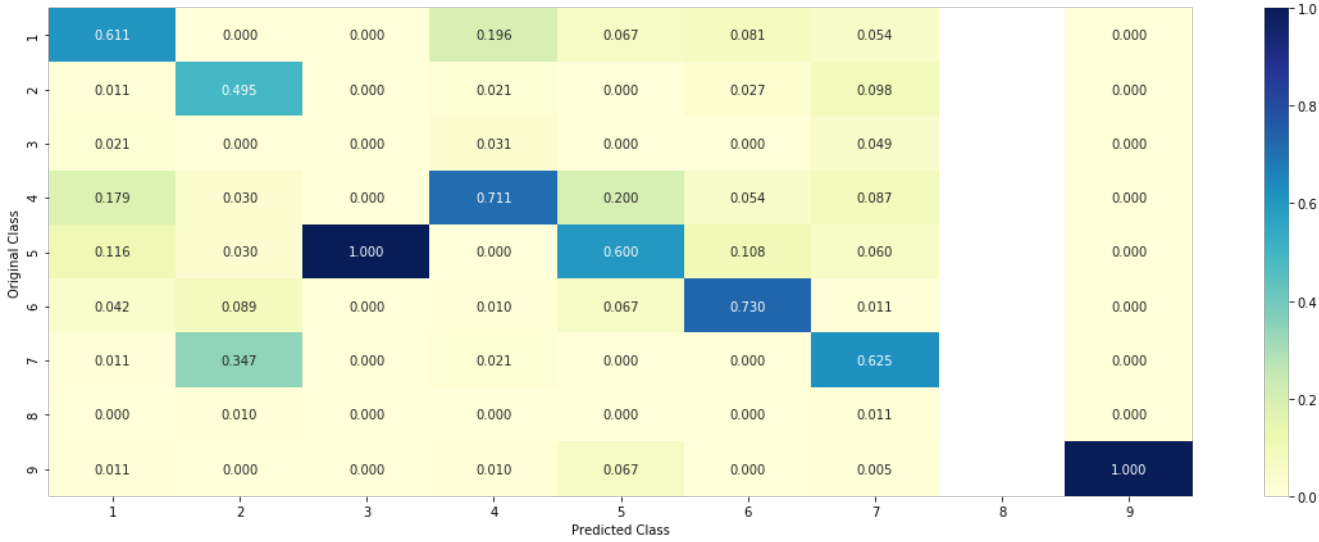
```
# ------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, 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-fores
t-and-their-construction-2/
# ------------------------------

clf = random_search_clf.best_estimator_
predict_and_plot_confusion_matrix(train_x_onehotCoding_tfidf, train_y,cv_x_onehotCoding_tfidf,cv_y
, clf)
```

```
Log loss : 1.1883918149034804
Number of mis-classified points : 0.37969924812030076
-------------------- Confusion matrix --------------------
```

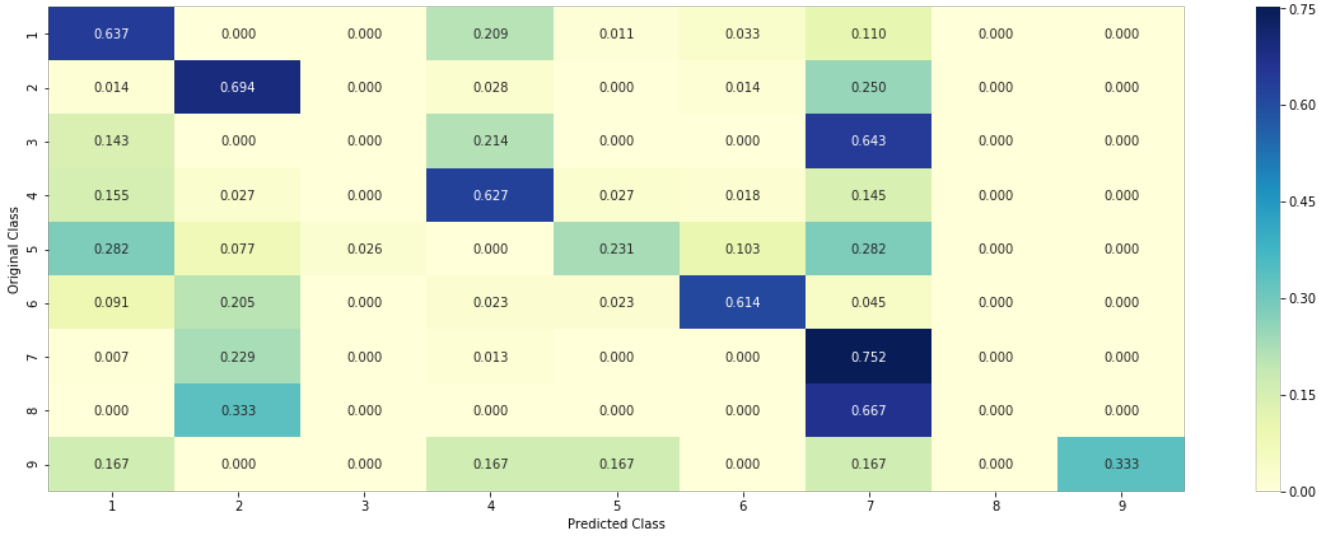




---------- Precision matrix (Columm Sum=1) ----------



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [242]:

```
# test_point_index = 10
clf = random_search_clf.best_estimator_
clf.fit(train_x_onehotCoding,tfidf, train_y)
```

```
clf.fit(train_x_onehotCoding_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding_tfidf, train_y)

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding_tfidf[t
est_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_imp_feature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
.iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature, text_vectorizer =
'tf-idf', features = selected_features_tfidf)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2744 0.0716 0.0241 0.428  0.0664 0.0464 0.0748 0.0061 0.0081]]
Actual Class : 4
--------------------------------------------------
7 Text feature [suppressor] present in test data point [True]
11 Text feature [therapy] present in test data point [True]
35 Text feature [yeast] present in test data point [True]
60 Text feature [transcriptional] present in test data point [True]
71 Text feature [useful] present in test data point [True]
Out of the top  100  features  5 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [243]:

```
test_point_index = 5
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding_tfidf[t
est_point_index]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_imp_feature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
.iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature, text_vectorizer =
'tf-idf', features = selected_features_tfidf)
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0663 0.5177 0.018  0.0642 0.0433 0.0304 0.2501 0.0048 0.0052]]
Actuall Class : 2
--------------------------------------------------
2 Text feature [tyrosine] present in test data point [True]
8 Text feature [treatment] present in test data point [True]
11 Text feature [therapy] present in test data point [True]
93 Text feature [trial] present in test data point [True]
Out of the top  100  features  4 are present in query point
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [244]:

```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=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/
#-------------------------------


# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
# ----------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

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


# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
# ----------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_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-construction-2/
# ----------------------------------


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

clf2 = SGDClassifier(alpha=0.0001, penalty='l2', loss='hinge', class_weight='balanced', random_state=0)
clf2.fit(train_x_onehotCoding_tfidf, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


clf3 = KNeighborsClassifier(n_neighbors=5)
clf3.fit(train_x_onehotCoding_tfidf, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")

sig_clf1.fit(train_x_onehotCoding_tfidf, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehotCoding_tfidf))))
sig_clf2.fit(train_x_onehotCoding_tfidf, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(cv_x_onehotCoding_tfidf))))
```

```
sig_clf3.fit(train_x_onehotCoding_tfidf, train_y)
print("K Neighbors : Log Loss: %0.2f" % (log_loss(cv_y,
sig_clf3.predict_proba(cv_x_onehotCoding_tfidf))))
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_p
robas=True)
    sclf.fit(train_x_onehotCoding_tfidf, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sc
lf.predict_proba(cv_x_onehotCoding_tfidf))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding_tfidf))
    if best_alpha > log_error:
        best_alpha = log_error
```

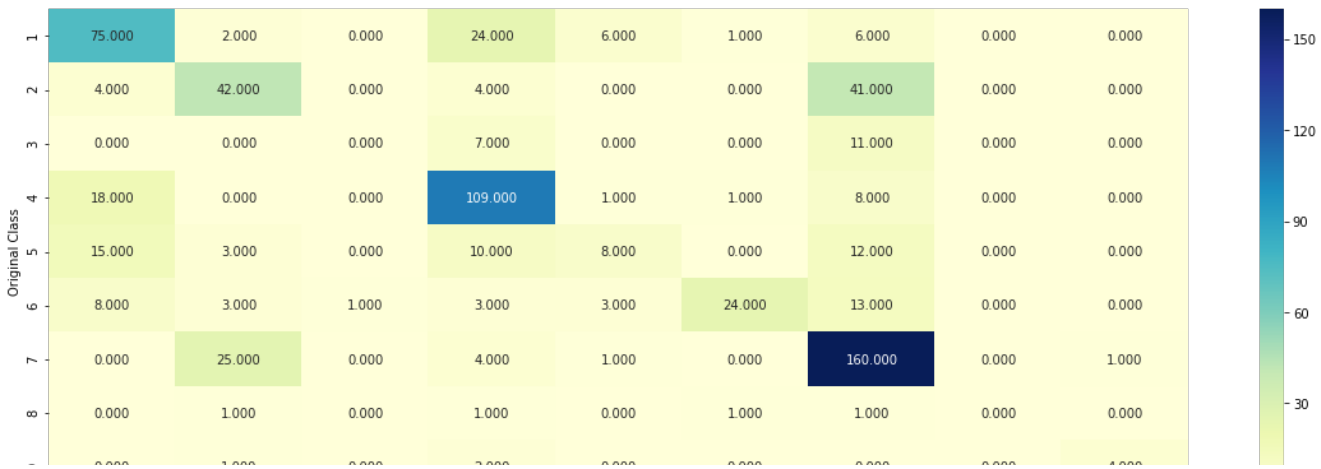
```
Logistic Regression :  Log Loss: 1.06
Support vector machines : Log Loss: 1.13
K Neighbors : Log Loss: 1.15
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.175
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.011
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.449
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.085
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.140
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.398
```

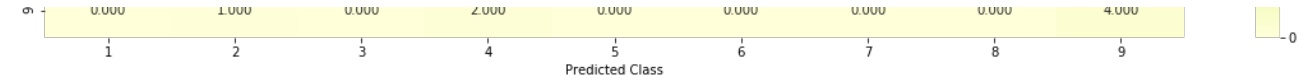
4.7.2 testing the model with the best hyper parameters

In [245]:

```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba
s=True)
sclf.fit(train_x_onehotCoding_tfidf, train_y)

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

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

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

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

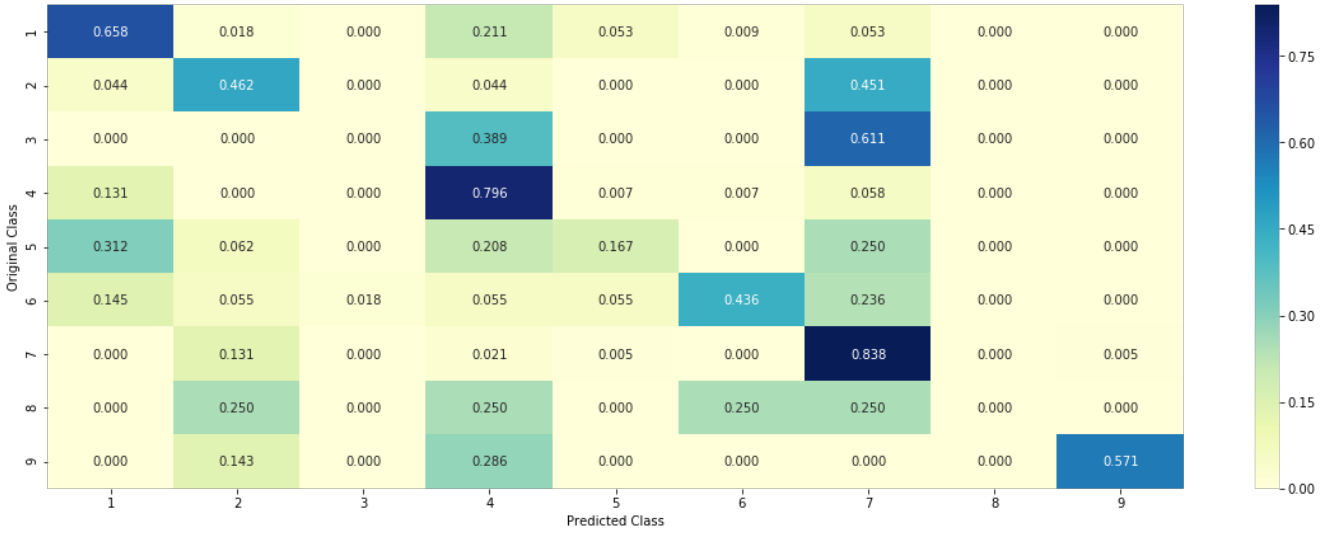
```
Log loss (train) on the stacking classifier : 0.617766195961976
Log loss (CV) on the stacking classifier : 1.0854352829952434
Log loss (test) on the stacking classifier : 1.139876562880946
Number of missclassified point : 0.36541353383458647
-------------------- Confusion matrix --------------------
```






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


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



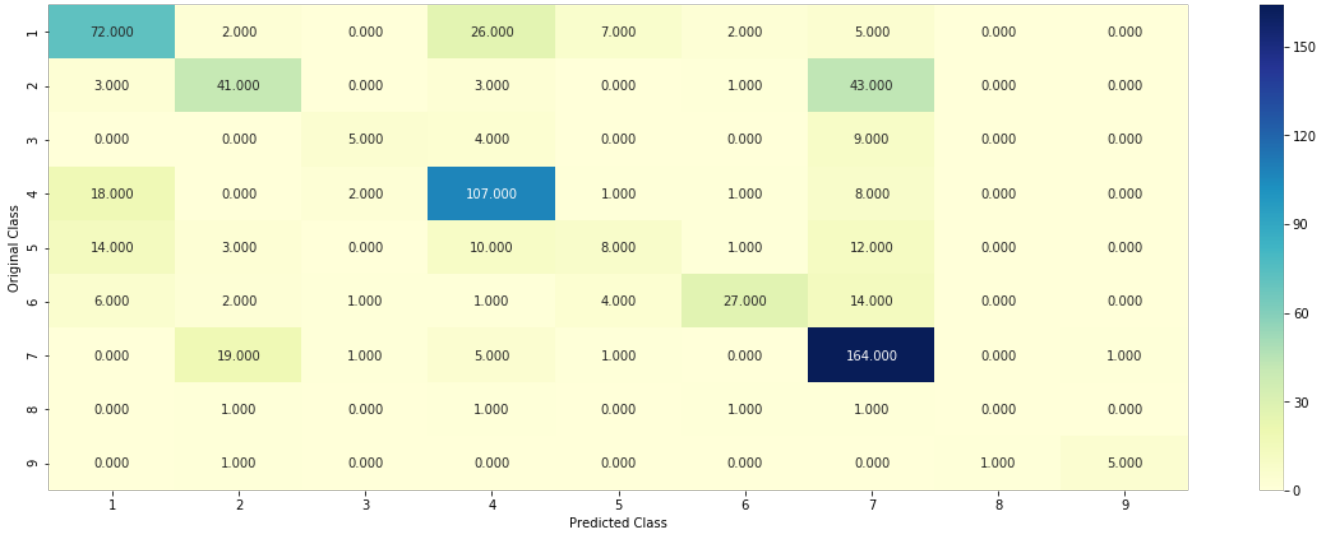
4.7.3 Maximum Voting classifier

In [246]:

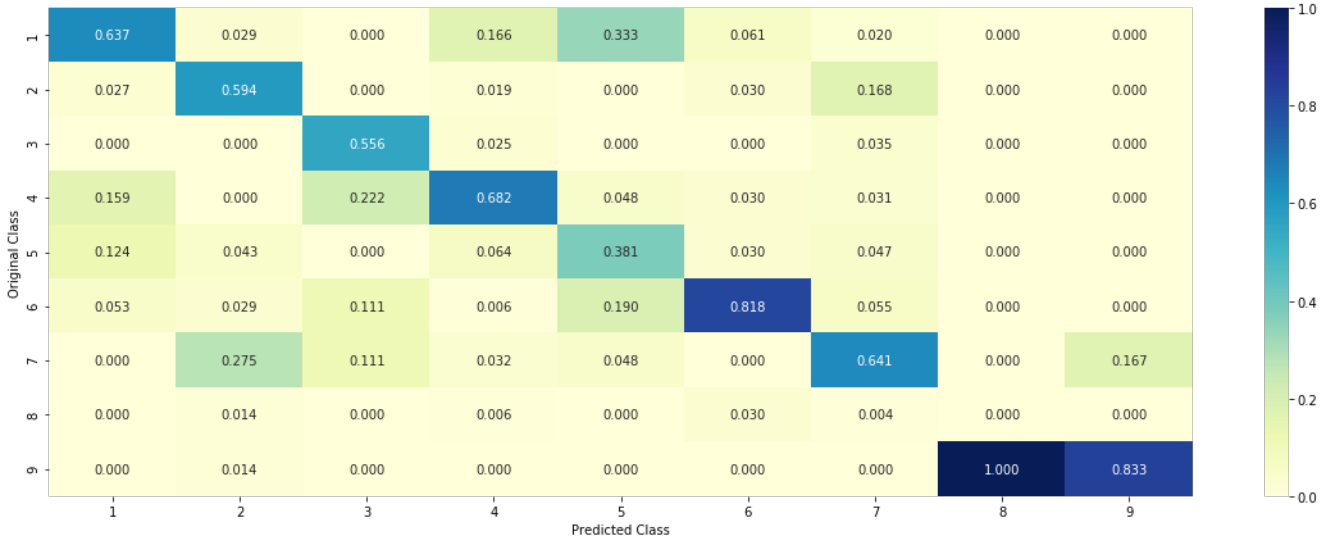
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('knn', sig_clf3)], voting
='soft')
vclf.fit(train_x_onehotCoding_tfidf, train_y)
print("Log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

```
Log loss (train) on the VotingClassifier : 0.6758677314583923
Log loss (CV) on the VotingClassifier : 1.0355280073013928
Log loss (test) on the VotingClassifier : 1.0929847915537725
```

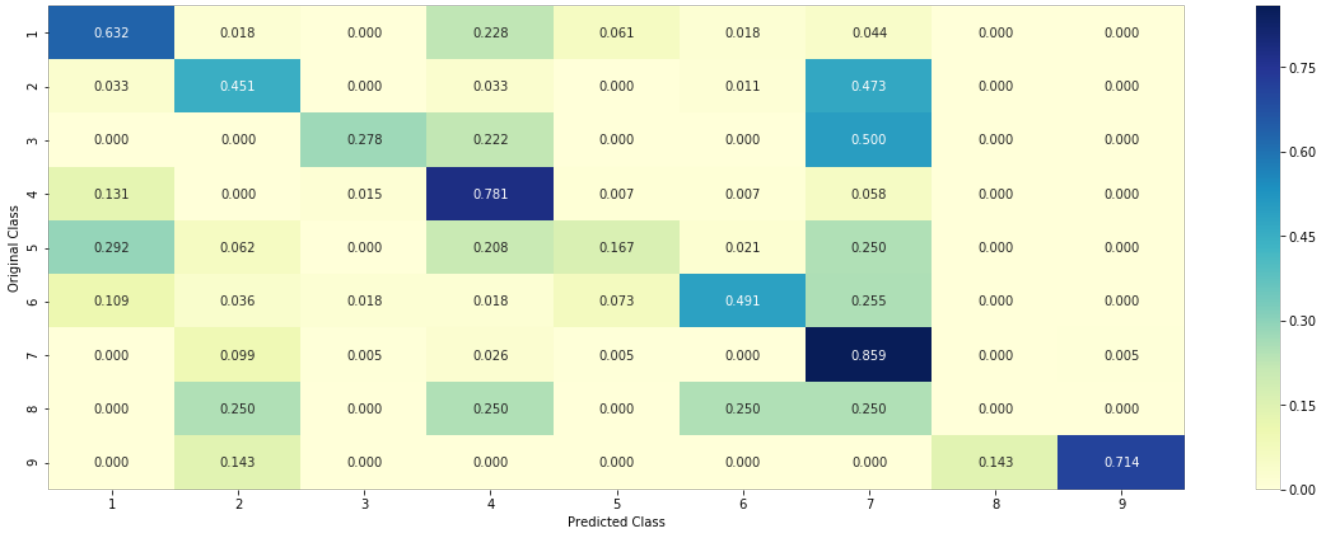
```
Number of missclassified point : 0.3548872180451128
--------------------- Confusion matrix ---------------------
```




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




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



5. Results

In [0]:

```
resultsDataFrame = pd.DataFrame(columns = ['Model', 'Vectorization Technique', 'Cross-Validation Lo
g-loss', 'Test Log-loss', 'Misclassification%']);
models = ['Naive Bayes', 'K-NN', 'Logistic Regression', 'Logistic Regression', 'Logistic
Regression', 'Linear SVM', 'Random Forests', 'Stacking Classifier', 'Majority Vote Classifier']
vectorization_technique = ['tf-idf', 'response coding & tf-idf', 'tf-idf(balanced data)', 'tf-idf(
imbalanced data)', 'unigrams & bigrams', 'tf-idf', 'tf-idf', 'tf-idf', 'tf-idf']
cv_loss = [1.187, 1.142, 0.98, 1.02, 1.264, 1.142, 1.188, 1.13, 1.035]
test_loss = [1.239, 1.259, 0.989, 1.07, 1.273, 1.214, 1.215, 1.15, 1.092]
misclassification = [39.66, 40.03, 32.51, 36.09, 45.11, 36.09, 37.96, 36.54, 35.48]
for i in range(len(models)):
  resultsDataFrame = resultsDataFrame.append({'Model': models[i], 'Vectorization Technique': vector
ization_technique[i], 'Cross-Validation Log-loss': cv_loss[i], 'Test Log-loss': test_loss[i],
'Misclassification%': misclassification[i]}, ignore_index = True)
```

In [254]:

```
resultsDataFrame
```

Out[254]:

	Model	Vectorization Technique	Cross-Validation Log-loss	Test Log-loss	Misclassification%
0	Naive Bayes	tf-idf	1.187	1.239	39.66
1	K-NN	response coding & tf-idf	1.142	1.259	40.03
2	Logistic Regression	tf-idf(balanced data)	0.980	0.989	32.51
3	Logistic Regression	tf-idf(imbalanced data)	1.020	1.070	36.09
4	Logistic Regression	unigrams & bigrams	1.264	1.273	45.11
5	Linear SVM	tf-idf	1.142	1.214	36.09
6	Random Forests	tf-idf	1.188	1.215	37.96
7	Stacking Classifier	tf-idf	1.130	1.150	36.54
8	Majority Vote Classifier	tf-idf	1.035	1.092	35.48

6. Conclusions

1. From exploratory data analysis we can see that three features plays very important role in predicting class of data and of them text is the most important feature.
2. There are some better algorithms like xgboost which we didn't use as feature interpretation is difficult in that.
3. It seems that of all models the models built using logistic regression are better than others.
4. All models we got are far better than a random model and so we can use them in live.
5. All models built are not that over-fitted or under-fitted except model built by random forests which is highly overfitted.
6. It seems like when we increase n-grams we can build even better model.
7. Atlast, the model I suggest to use is model built with logistic regression on balanced data vectorized using bag of words & tf-idf. It resulted in less log-loss value when compared with all other models.