Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

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

from mlxtend.classifier import StackingClassifier

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

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

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

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

Out[2]:

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

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

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

3.1.2. Reading Text Data

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

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

Out[3]:

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

3.1.3. Preprocessing of text

In [4]:
```
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))


def nlp_preprocessing(total_text, index, column):
    if type(total_text) is not int:
        string = ""
        # replace every special char with space
        total_text = re.sub('[^a-zA-Z0-9\n]', ' ', total_text)
        # replace multiple spaces with single space
        total_text = re.sub('\s+',' ', total_text)
        # converting all the chars into lower-case.
        total_text = total_text.lower()

        for word in total_text.split():
        # if the word is a not a stop word then retain that word from
            if not word in stop_words:
                string += word + " "

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

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

```
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 : 227.78651299999999 seconds
```

In [6]:
```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[6]:

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

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

Out[7]:

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

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

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

Out[9]:

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

3.1.4. Test, Train and Cross Validation Split

3.1.4.1. Splitting data into train, test and cross validation (64:20:16)

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

# split the data into test and train by maintaining same distribution 
X_train, test_df, y_train, y_test = train_test_split(result, y_true, s
# split the train data into train and cross validation by maintaining 
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, st
```

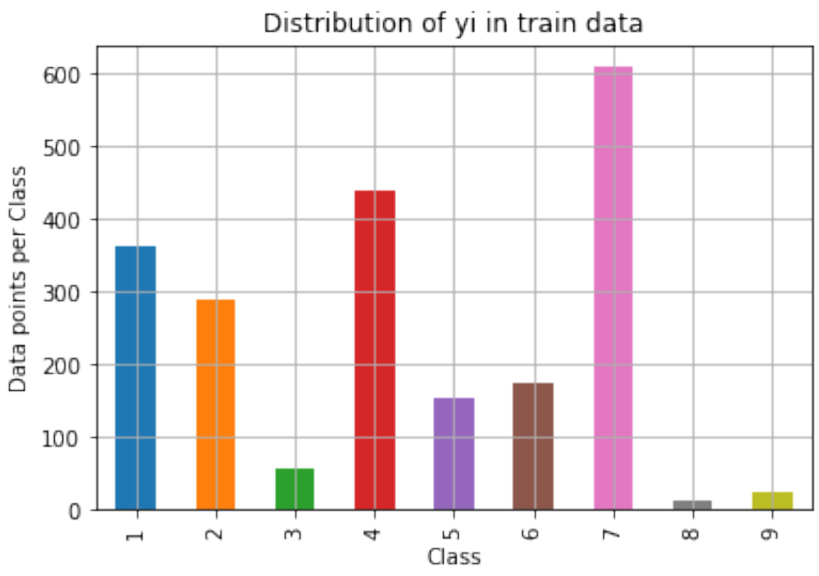
We split the data into train, test and cross validation data sets, preserving the ratio of class distribution in the original data set

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

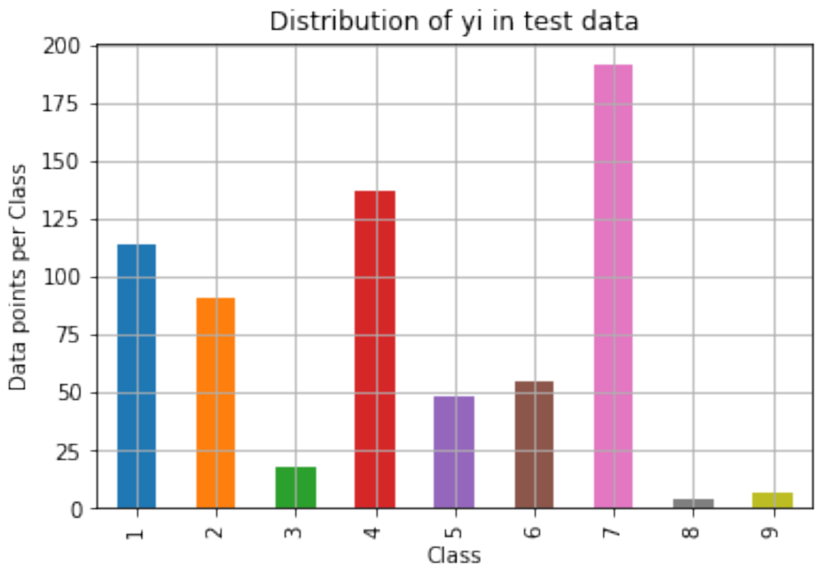
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

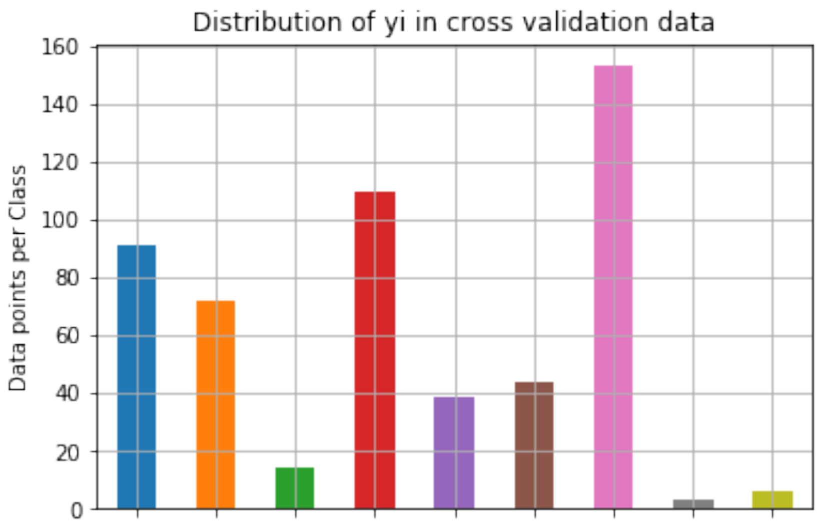
In [12]:
```
# it returns a dict, keys as class labels and values as the number of 
train_class_distribution = train_df['Class'].value_counts().sortlevel(
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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

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


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

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

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

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

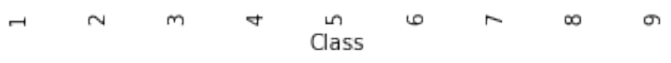


```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298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



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





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

3.2 Prediction using a 'Random' Model

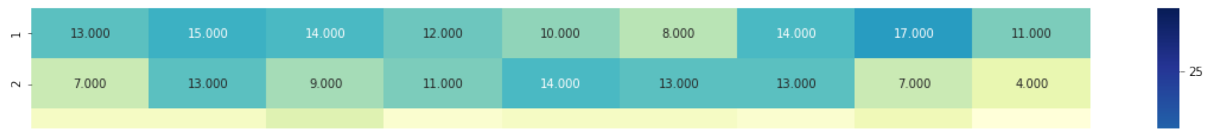
In a 'Random' Model, we generate the NINE class probabilites randomly such that they sum to 1.

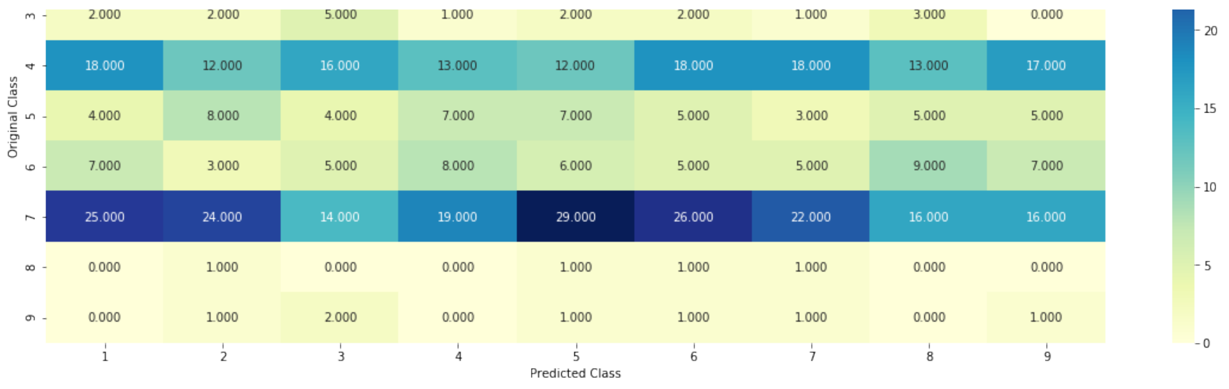
In [13]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of c

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

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 correspo
    # 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 elemen
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 correspo
    # 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))
    plt.figure(figsize=(20,7))
    sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=la
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

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

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

In [14]:

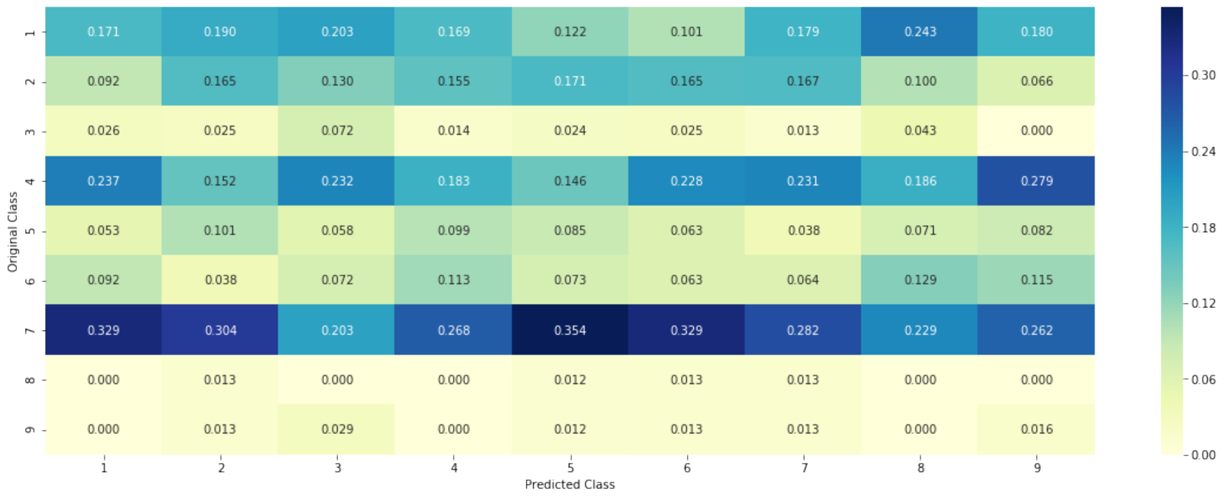
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# 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


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

```
Log loss on Cross Validation Data using Random Model 2.4784642729411
55
Log loss on Test Data using Random Model 2.5256048601090977
-------------------- Confusion matrix --------------------
```

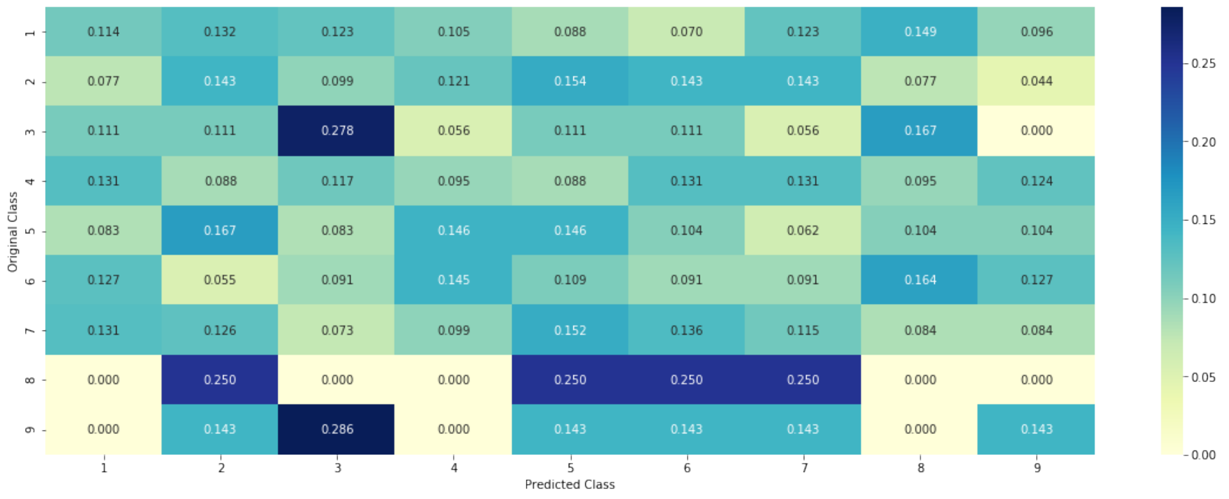




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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



3.3 Univariate Analysis

In [15]:
```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given fea
# build a vector (1*9) , the first element = (number of times it occur
# gv_dict is like a look up table, for every gene it store a (1*9) rep
# 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_
# 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 ar
    gv_dict = dict()

    # denominator will contain the number of time that particular feat
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation
        # vec is 9 diamensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['G
            #         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[

            # cls_cnt.shape[0](numerator) will contain the number of t
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 9

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.06818
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.06122
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625,
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.0606
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.0691
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.07333
    #      ...
    #     }
    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 
    gv_fea = []
    # for every feature values in the given data frame we will check i
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_f
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#             gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea
```

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

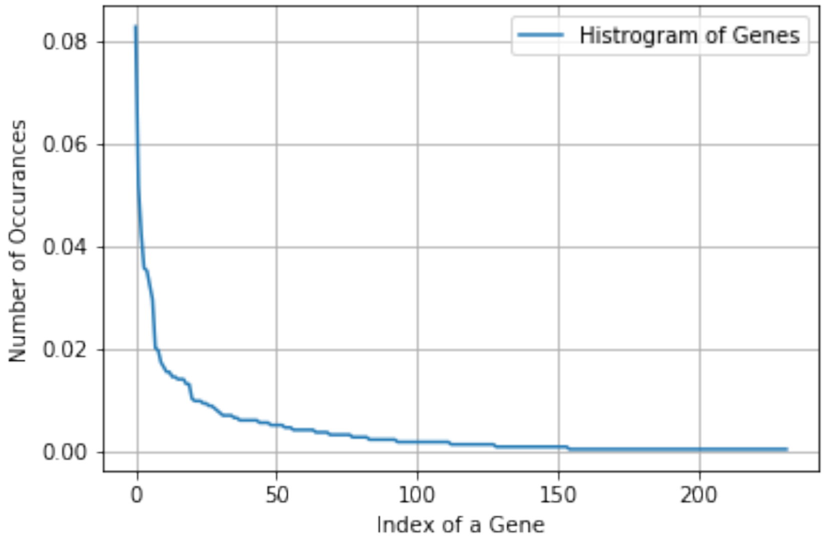
In [16]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 232
BRCA1     176
TP53      110
EGFR       90
PTEN       76
BRCA2      75
KIT        69
BRAF       63
ERBB2      43
ALK        42
PDGFRA     37
Name: Gene, dtype: int64
```

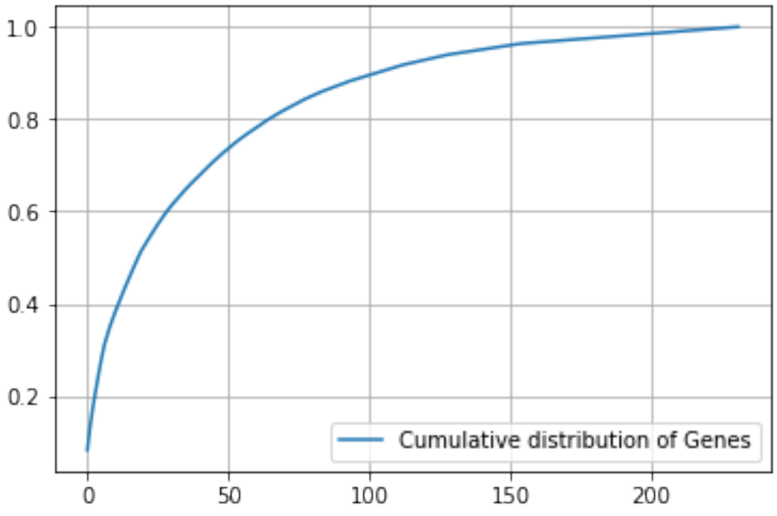
In [17]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories o
```

```
Ans: There are 232 different categories of genes in the train data,
and they are distibuted as follows
```

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



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



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

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

In [21]:
```
print("train_gene_feature_responseCoding is converted feature using re
```

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

In [22]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer()

train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Ge
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene']
```

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

Out[23]:
```
1696      PMS2
2971       KIT
1942    CARD11
371       TP53
13         CBL
Name: Gene, dtype: object
```

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

```
 'stat5',
 'stk11',
 'tcf3',
 'tcf7l2',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tsc1',
 'tsc2',
 'u2af1',
 'vhl',
 'whsc1',
 'whsc1l1',
 'xpo1',
 'xrcc2',
 'yap1']
```

In [25]:
```
print("train_gene_feature_onehotCoding is converted feature using one-
```

```
train_gene_feature_onehotCoding is converted feature using one-hot e
ncoding method. The shape of gene feature: (2124, 232)
```

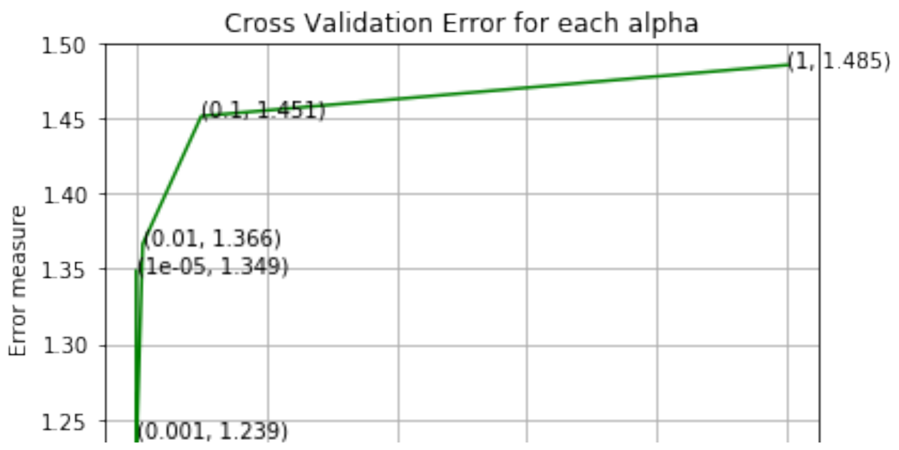
Q4. How good is this gene feature in predicting y_i?

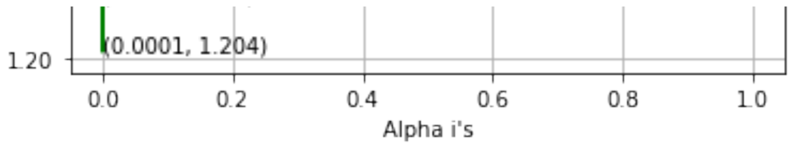
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/moc
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
# 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 Stoch
# 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
    clf.fit(train_gene_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.clas
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_arra
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',
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
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
```

```
For values of alpha =  1e-05 The log loss is: 1.3486976473069794
For values of alpha =  0.0001 The log loss is: 1.2044061213620791
For values of alpha =  0.001 The log loss is: 1.2386371953072366
For values of alpha =  0.01 The log loss is: 1.3664524263279412
For values of alpha =  0.1 The log loss is: 1.45140562181427
For values of alpha =  1 The log loss is: 1.4852922142539582
```





```
For values of best alpha =  0.0001 The train log loss is: 1.01650002
9590596
For values of best alpha =  0.0001 The cross validation log loss is:
1.2044061213620791
For values of best alpha =  0.0001 The test log loss is: 1.257102828
1365928
```

Q5. Is the Gene feature stable across all the data sets (Test, Train, Cross validation)?

Ans. Yes, it is. Otherwise, the CV and Test errors would be significantly more than train error.

In [27]:
```
print("Q6. How many data points in Test and CV datasets are covered by

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

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

```
Q6. How many data points in Test and CV datasets are covered by the
232  genes in train dataset?
Ans
1. In test data 640 out of 665 : 96.2406015037594
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?

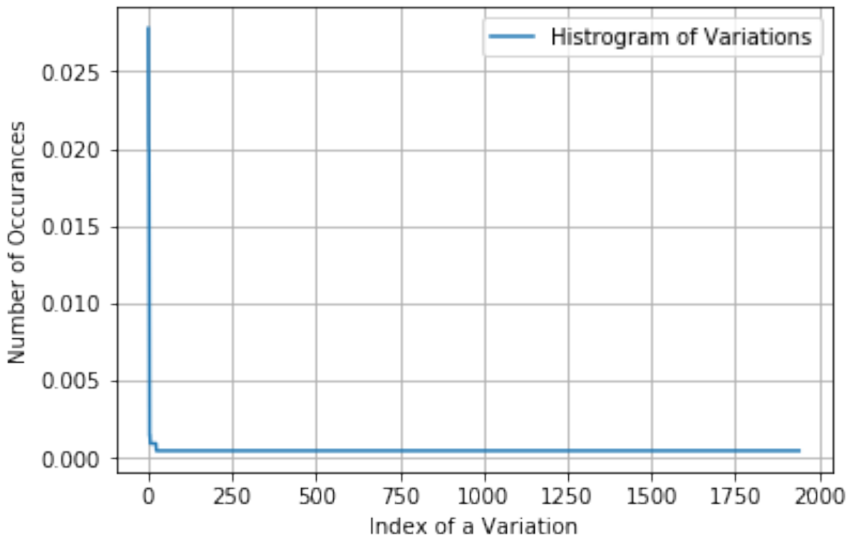
In [28]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1942
Truncating_Mutations    59
Deletion                43
Amplification           42
Fusions                 20
G12V                     3
E17K                     3
G35R                     2
F384L                    2
Overexpression           2
G12D                     2
Name: Variation, dtype: int64
```

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

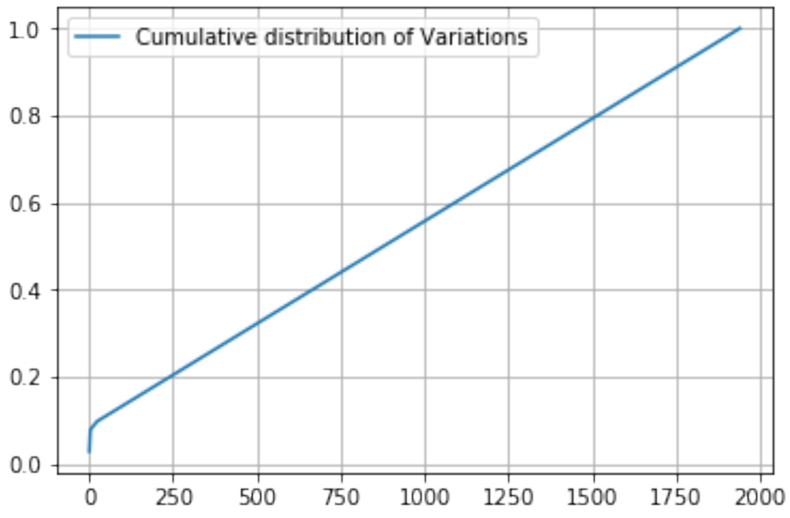
```
Ans: There are 1942 different categories of variations in the train
data, and they are distibuted as follows
```

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



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

```
[0.02777778 0.0480226  0.06779661 ... 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/ (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/)

1. One hot Encoding
2. Response coding

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

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

In [33]:
```
print("train_variation_feature_responseCoding is a converted feature us
```

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

In [34]:
```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()

train_variation_feature_onehotCoding = variation_vectorizer.fit_transfo
test_variation_feature_onehotCoding = variation_vectorizer.transform(te
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_v
```

In [35]:
```
print("train_variation_feature_onehotEncoded is converted feature using
```

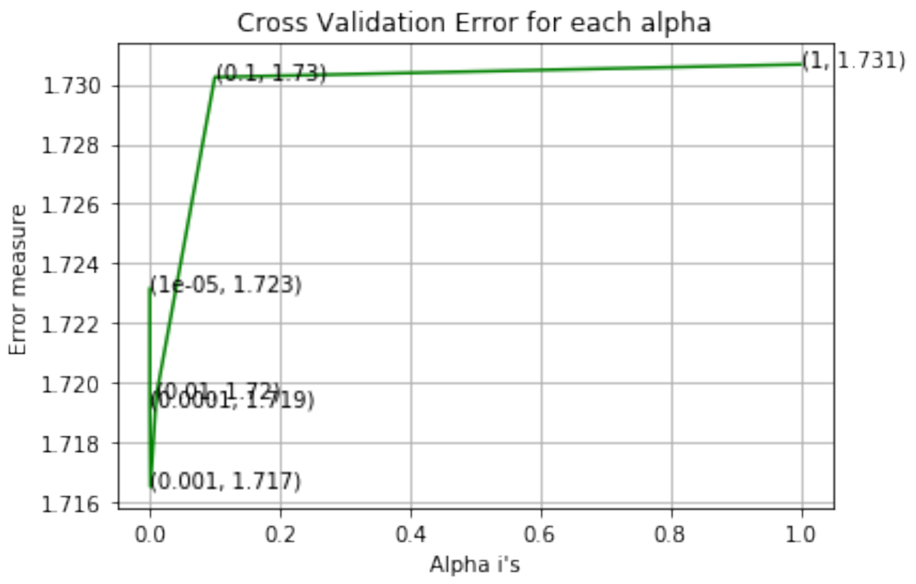
```
train_variation_feature_onehotEncoded is converted feature using the
onne-hot encoding method. The shape of Variation feature: (2124, 197
4)
```

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

Let's build a model just like the earlier!

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
# 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 Stoc
# 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
    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_onehotCodin

    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.clas
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_arra
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',
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
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross val
predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log
```

```
For values of alpha =  1e-05 The log loss is: 1.723149654408076
For values of alpha =  0.0001 The log loss is: 1.719284826787999
For values of alpha =  0.001 The log loss is: 1.716513194600769 7
For values of alpha =  0.01 The log loss is: 1.7195567183490503
For values of alpha =  0.1 The log loss is: 1.730246298152995
For values of alpha =  1 The log loss is: 1.7306740533710212
```



```
For values of best alpha =  0.001 The train log loss is: 1.074285508
1393898
For values of best alpha =  0.001 The cross validation log loss is:
```

```
1.7165131946007697
For values of best alpha =  0.001 The test log loss is: 1.6910460060
205412
```

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

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

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

```
Q12. How many data points are covered by total  1942  genes in test
and cross validation data sets?
Ans
1. In test data 75 out of 665 : 11.278195488721805
2. In cross validation data 64 out of  532 : 12.030075187969924
```

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 [38]:
```
# cls_text is a data frame
# for every row in data fram consider the 'TEXT'
# split the words by space
# make a dict with those words
# increment its count whenever we see that word

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

In [39]:
```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(
            text_feature_responseCoding[row_index][i] = math.exp(sum_p
            row_index += 1
    return text_feature_responseCoding
```

In [40]:
```
# building a CountVectorizer with all the words that occured minimum 3
text_vectorizer = TfidfVectorizer(max_features=2000)

train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_
# 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 an
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 nu
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_coun


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

```
Total number of unique words in train data : 2000
```

In [41]:
```
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 [42]:
```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
test_text_feature_responseCoding  = get_text_responsecoding(test_df)
cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)
```

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

In [44]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCo

# we use the same vectorizer that was trained on train data
test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TE
# don't forget to normalize every feature
test_text_feature_onehotCoding = normalize(test_text_feature_onehotCod

# 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,
```

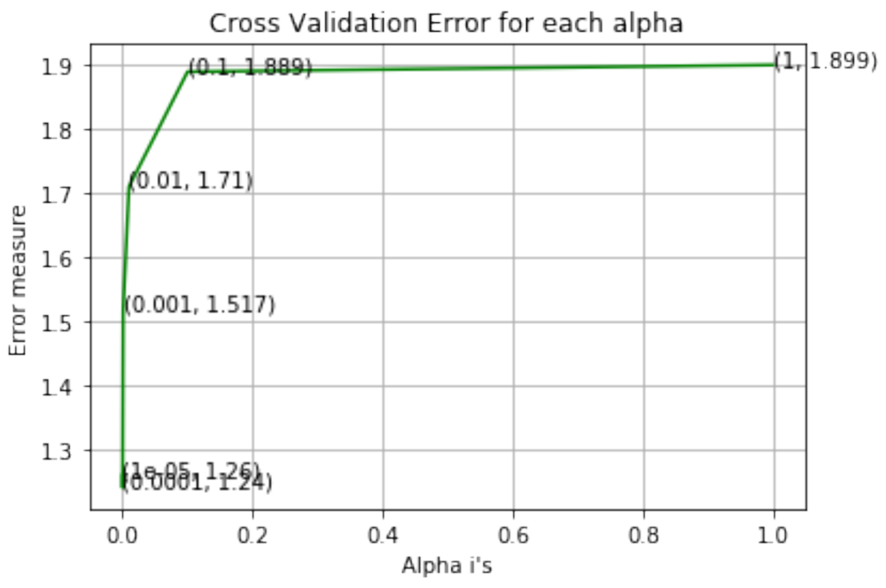
In [45]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

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

```
2408721251: 1, 45.75694966969335: 1, 45.13965712959869: 1, 44.564841
548845706: 1, 44.061792280193245: 1, 43.53457901707379: 1, 43.365560
64784127: 1, 43.08353556802731: 1, 41.88095288646154: 1, 41.19097471
996031: 1, 41.03339838107955: 1, 38.65037102158741: 1, 38.1173159434
1287: 1, 37.10028076630142: 1, 36.997393534270785: 1, 36.75229170928
03: 1, 36.25821081665058: 1, 35.95969268730166: 1, 35.92728169532430
4: 1, 35.7395953575918: 1, 35.708140972975414: 1, 35.1079207767532:
1, 35.01221432754632: 1, 34.91991266477593: 1, 34.85631491516496: 1,
34.79395856428282: 1, 34.56609178780437: 1, 34.29463307343136: 1, 34
.14448767831047: 1, 34.10546951916053: 1, 34.04627271376938: 1, 33.2
1799981671358: 1, 33.13649025437092: 1, 32.58829438311439: 1, 32.563
55990607419: 1, 32.21039734042745: 1, 32.189253109921104: 1, 31.3979
74847771152: 1, 30.84230740003412: 1, 30.381640263719913: 1, 30.1928
16649961248: 1, 30.145876031005148: 1, 30.06046020743467: 1, 29.7589
63243364242: 1, 29.66715655490641: 1, 29.52220228415448: 1, 29.39418
6216844943: 1, 29.387271357845254: 1, 29.079248508819212: 1, 28.7566
57987468774: 1, 28.651869375048477: 1, 28.6206760313264: 1, 28.44393
5941308823: 1, 28.285976907120983: 1, 28.17425293186685: 1, 28.13789
3630441702: 1, 28.07574077967449: 1, 27.62685155694277: 1, 27.546427
```

In [47]:
```python
# Train a Logistic regression+Calibration model using text features whic
lpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modu
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, lea
# 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 Stocha
# predict(X)    Predict class labels for samples in X.

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


v_log_error_array=[]
or i in alpha:
   clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=
   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.class
   print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv,
:ig, ax = plt.subplots()
ix.plot(alpha, cv_log_error_array,c='g')
:or 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
›lt.grid()
›lt.title("Cross Validation Error for each alpha")
›lt.xlabel("Alpha i's")
›lt.ylabel("Error measure")
›lt.show()


›est_alpha = np.argmin(cv_log_error_array)
:lf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', r
:lf.fit(train_text_feature_onehotCoding, y_train)
;ig_clf = CalibratedClassifierCV(clf, method="sigmoid")
;ig_clf.fit(train_text_feature_onehotCoding, y_train)

›redict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
›rint('For values of best alpha = ', alpha[best_alpha], "The train log l
›redict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
›rint('For values of best alpha = ', alpha[best_alpha], "The cross valic
›redict_y = sig_clf.predict_proba(test_text_feature_onehotCoding)
›rint('For values of best alpha = ', alpha[best_alpha], "The test log lc
```

```
For values of alpha =  1e-05 The log loss is: 1.25965476748488
For values of alpha =  0.0001 The log loss is: 1.2404946005763091
For values of alpha =  0.001 The log loss is: 1.5169307500945592
For values of alpha =  0.01 The log loss is: 1.709542174626329
For values of alpha =  0.1 The log loss is: 1.8885282681499767
For values of alpha =  1 The log loss is: 1.8991341447880266
```



```
For values of best alpha =  0.0001 The train log loss is: 0.69856901
43046264
For values of best alpha =  0.0001 The cross validation log loss is:
1.2404946005763091
For values of best alpha =  0.0001 The test log loss is: 1.224876352
```

```
2414252
```

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

Ans. Yes, it seems like!

In [48]:
```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(max_features=2000)

    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_cou
    len1 = len(set(df_text_features))
    len2 = len(set(train_text_features) & set(df_text_features))
    return len1,len2
```

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

```
93.65 % of word of test data appeared in train data
92.75 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [50]:
```
#Data preparation for ML models.

#Misc. functionns for ML models


def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y,
    clf.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 probabili
    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_
    plot_confusion_matrix(test_y, pred_y)
```

In [51]:
```
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

In [52]:
```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer()

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

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

    word_present = 0
    for i,v in enumerate(indices):
        if (v < fea1_len):
            word = gene_vec.get_feature_names()[v]
            yes_no = True if word == gene else False
            if yes_no:
                word_present += 1
                print(i, "Gene feature [{}] present in test data point
        elif (v < fea1_len+fea2_len):
            word = var_vec.get_feature_names()[v-(fea1_len)]
            yes_no = True if word == var else False
            if yes_no:
                word_present += 1
                print(i, "variation feature [{}] present in test data 
        else:
            word = text_vec.get_feature_names()[v-(fea1_len+fea2_len)]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
                print(i, "Text feature [{}] present in test data point

    print("Out of the top ",no_features," features ", word_present, "a
```

Stacking the three types of features

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

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

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,t
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,tes
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_vari

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

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

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


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseC
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCod
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding

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

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

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124,
4206)
(number of data points * number of features) in test data =  (665, 4
206)
(number of data points * number of features) in cross validation dat
a = (532, 4206)
```

In [55]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ",
print("(number of data points * number of features) in test data = ",
print("(number of data points * number of features) in cross validatio
```

```
 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, 2
7)
(number of data points * number of features) in cross validation dat
a = (532, 27)
```

4.1. Base Line Model

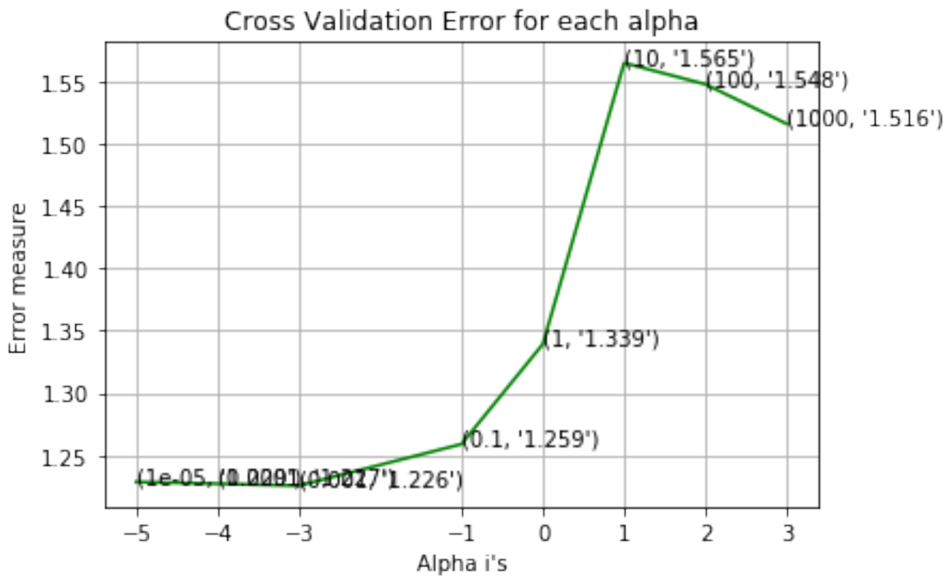
4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [56]:
```
# find more about Multinomial Naive base function here http://scikit-l
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_p

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


# find more about CalibratedClassifierCV here at http://scikit-learn.o
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, meth
#
# 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
# ------------------------


alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf
    # to avoid rounding error while multiplying probabilites we use lo
    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_
plt.grid()
plt.xticks(np.log10(alpha))
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


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


predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross val
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log
```

```
for alpha = 1e-05
Log Loss : 1.2288648750049873
for alpha = 0.0001
Log Loss : 1.2274568489092157
for alpha = 0.001
Log Loss : 1.2258580821921827
for alpha = 0.1
Log Loss : 1.2591399159997683
for alpha = 1
Log Loss : 1.3392901107942885
for alpha = 10
Log Loss : 1.564666451680673
for alpha = 100
Log Loss : 1.5475917423207854
```

```
for alpha = 1000
Log Loss : 1.5158279806124462
```

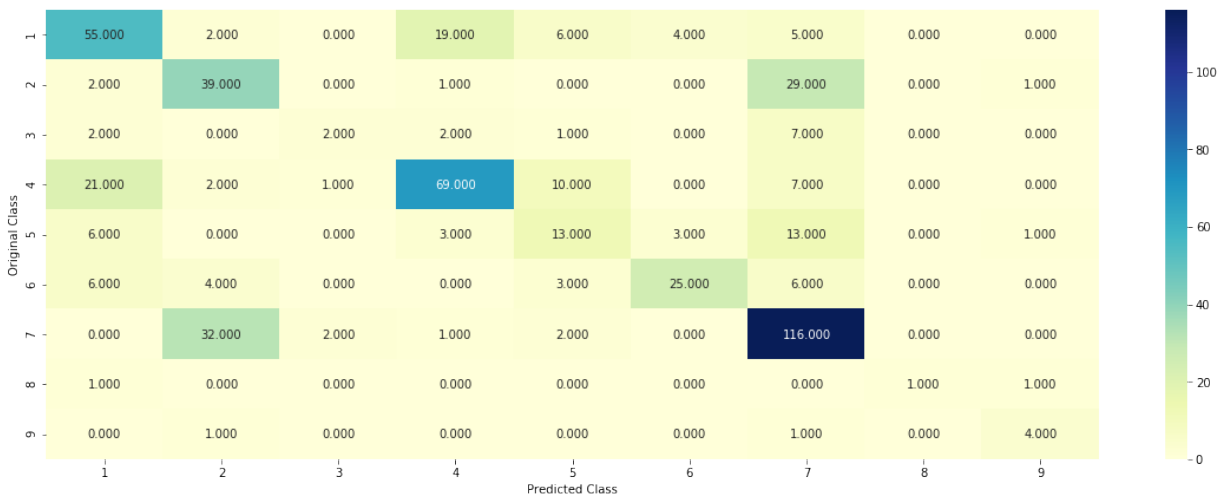


```
For values of best alpha =  0.001 The train log loss is: 0.546164501
9634253
For values of best alpha =  0.001 The cross validation log loss is:
1.2258580821921827
For values of best alpha =  0.001 The test log loss is: 1.2696236791
341742
```

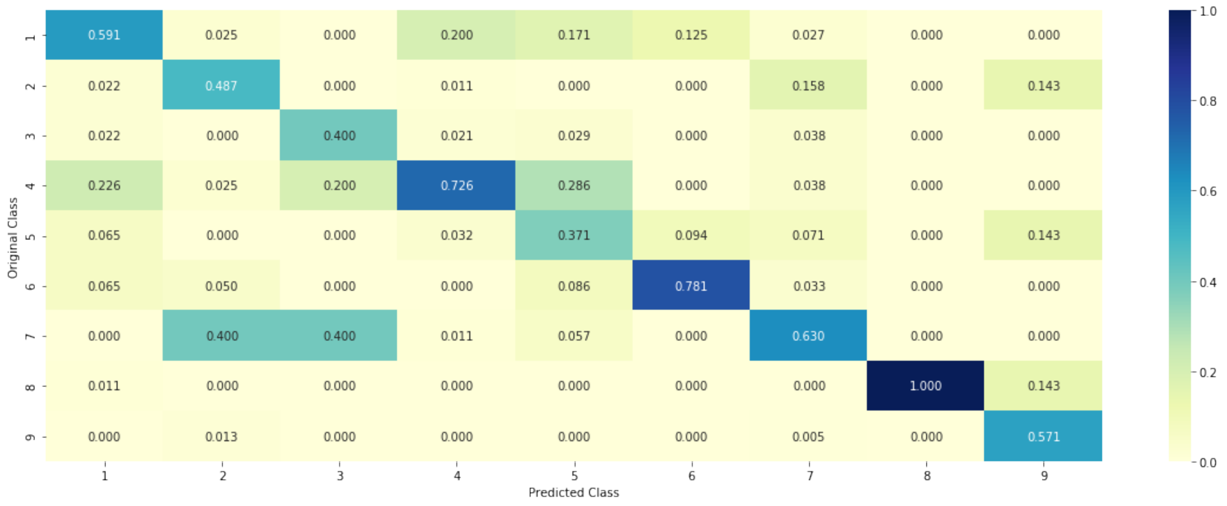
4.1.1.2. Testing the model with best hyper paramters

In [57]:
```
# find more about Multinomial Naive base function here http://scikit-l
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_p

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


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

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

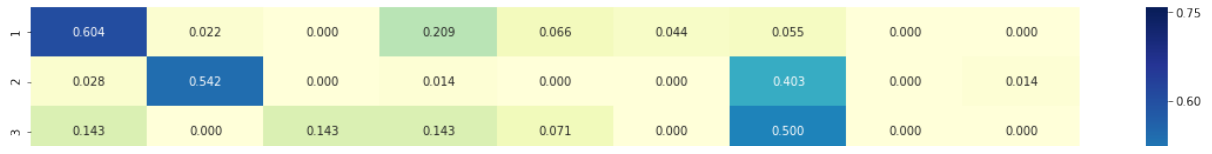
```
Log Loss : 1.2258580821921827
Number of missclassified point : 0.39097744360902253
-------------------- Confusion matrix --------------------
```

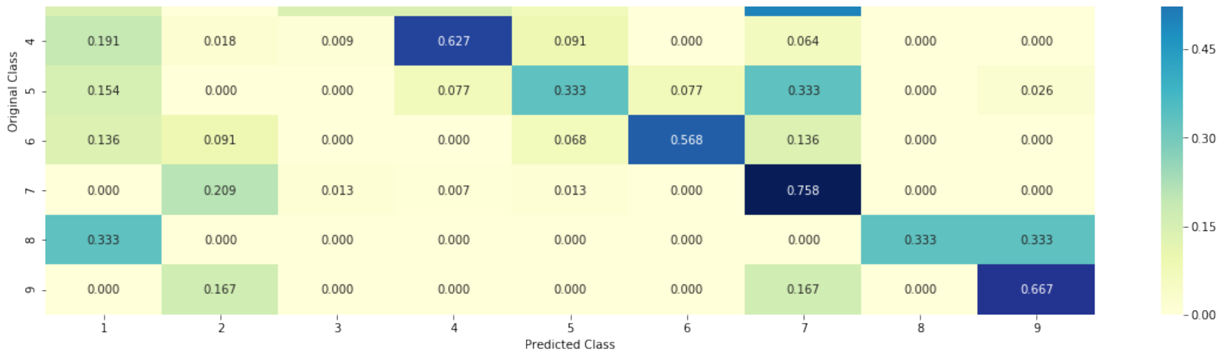


```
-------------------- Precision matrix (Columm Sum=1) ---------------
-----
```



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





4.1.1.3. Feature Importance, Correctly classified point

In [60]:
```
test_point_index = 5
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.6329 0.0472 0.0134 0.1339 0.0357
0.0378 0.0907 0.0049 0.0034]]
Actual Class : 1
--------------------------------------------------
56 Text feature [006] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [61]:
```
test_point_index = 25
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
```

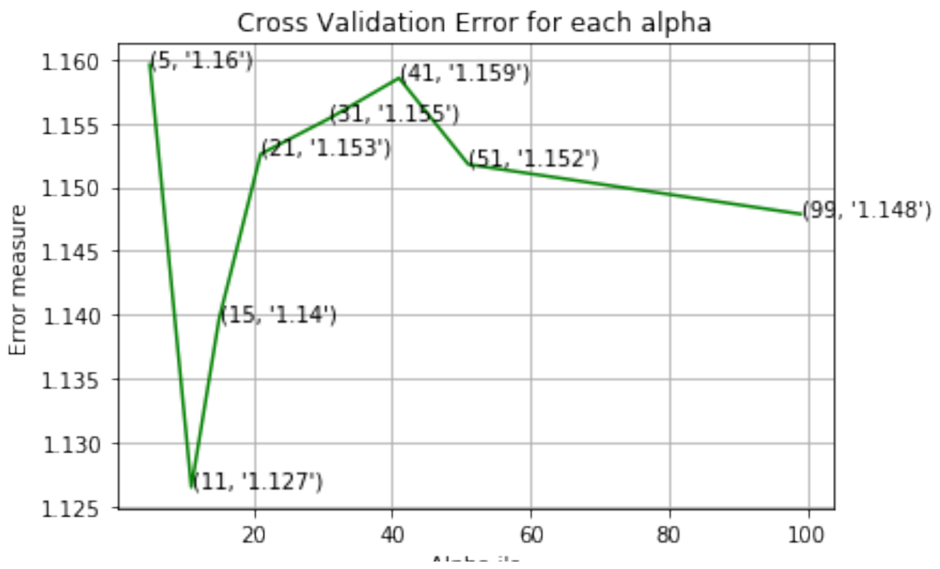
```
Predicted Class : 4
Predicted Class Probabilities: [[0.1994 0.0452 0.0129 0.5769 0.0344
0.0363 0.087  0.0047 0.0033]]
Actual Class : 1
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [62]:
```
find more about KNeighborsClassifier() here http://scikit-learn.org/sta
-------------------------
default parameter
KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto'
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 valu
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-on
------------------------------------


find more about CalibratedClassifierCV here at http://scikit-learn.org/
----------------------------
default paramters
sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=

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:
------------------------------------



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

g, ax = plt.subplots()
.plot(alpha, cv_log_error_array,c='g')
r i, txt in enumerate(np.round(cv_log_error_array,3)):
 ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
t.grid()
t.title("Cross Validation Error for each alpha")
t.xlabel("Alpha i's")
t.ylabel("Error measure")
t.show()


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

edict_y = sig_clf.predict_proba(train_x_responseCoding)
int('For values of best alpha = ', alpha[best_alpha], "The train log lo
edict_y = sig_clf.predict_proba(cv_x_responseCoding)
int('For values of best alpha = ', alpha[best_alpha], "The cross valida
edict_y = sig_clf.predict_proba(test_x_responseCoding)
int('For values of best alpha = ', alpha[best_alpha], "The test log los
```

```
for alpha = 5
Log Loss : 1.1595853675437158
for alpha = 11
Log Loss : 1.1265417444885677
for alpha = 15
Log Loss : 1.139685597047045
for alpha = 21
Log Loss : 1.1525809224707366
for alpha = 31
Log Loss : 1.1553896544442446
for alpha = 41
Log Loss : 1.1585244996386168
for alpha = 51
Log Loss : 1.1517724523759485
for alpha = 99
Log Loss : 1.1478957568123944
```



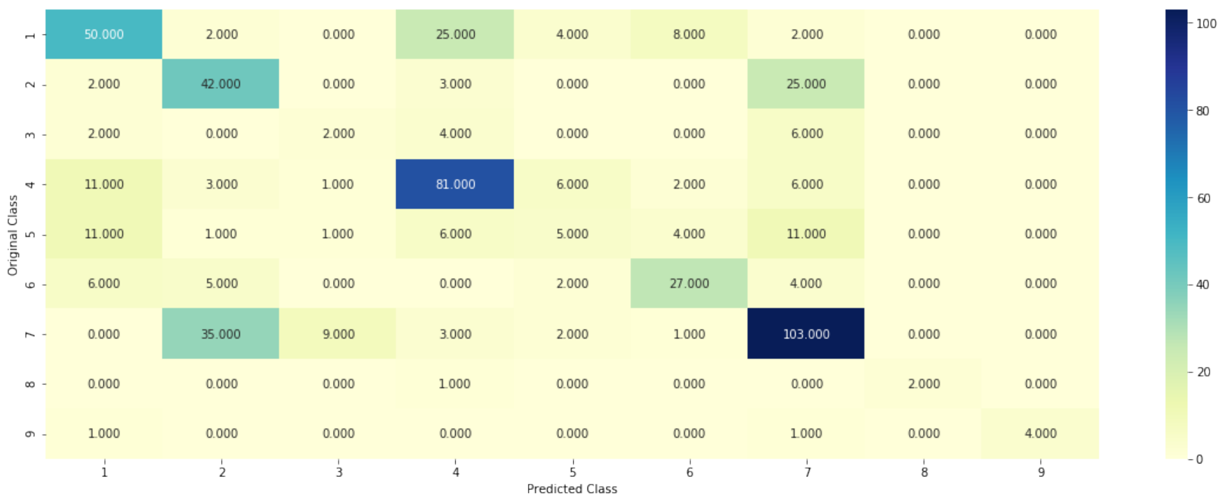
Alpha i's

```
For values of best alpha =  11 The train log loss is: 0.635487220687
4764
For values of best alpha =  11 The cross validation log loss is: 1.1
265417444885677
For values of best alpha =  11 The test log loss is: 1.1033467903220
922
```

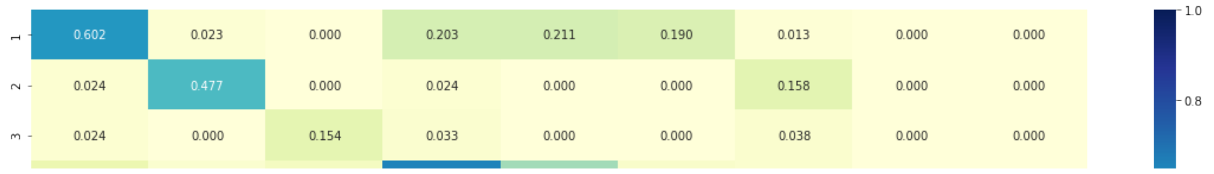
4.2.2. Testing the model with best hyper paramters

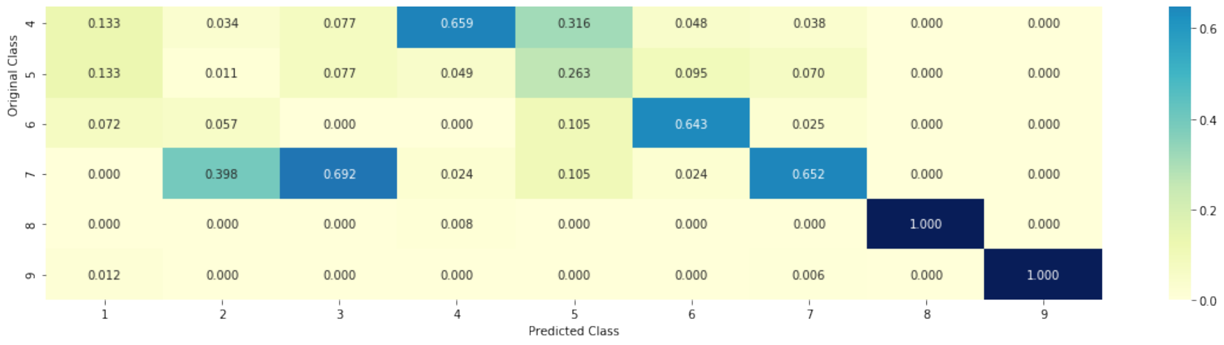
In [63]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='au
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target va
# 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-
#-------------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_
```

```
Log loss : 1.1265417444885677
Number of mis-classified points : 0.40601503759398494
-------------------- Confusion matrix --------------------
```

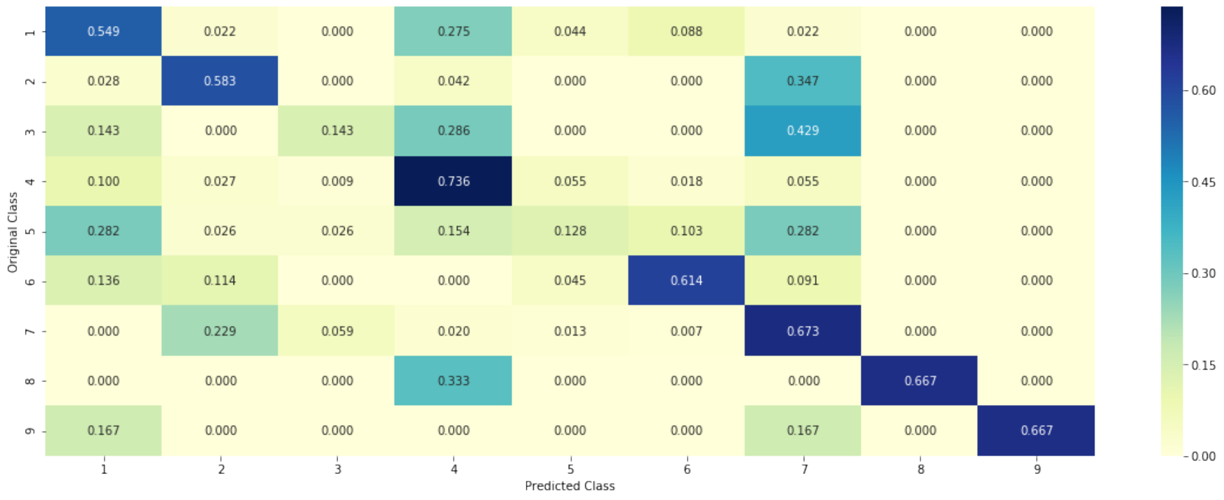


```
-------------------- Precision matrix (Columm Sum=1) ---------------
-----
```





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



4.2.3.Sample Query point -1

In [65]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 6
predicted_cls = sig_clf.predict(test_x_responseCoding[0].reshape(1,-1)
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].res
print("The ",alpha[best_alpha]," nearest neighbours of the test points
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 4
Actual Class : 4
The  11  nearest neighbours of the test points belongs to classes [4
4 5 5 7 7 7 4 2 2 2]
Fequency of nearest points : Counter({4: 3, 7: 3, 2: 3, 5: 2})
```

4.2.4. Sample Query Point-2

In [66]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 25

predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].res
print("the k value for knn is",alpha[best_alpha],"and the nearest neig
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]])
```

```
Predicted Class : 4
Actual Class : 1
the k value for knn is 11 and the nearest neighbours of the test poi
nts belongs to classes [4 4 4 4 4 4 4 4 4 4 4]
Fequency of nearest points : Counter({4: 11})
```

4.3. Logistic Regression

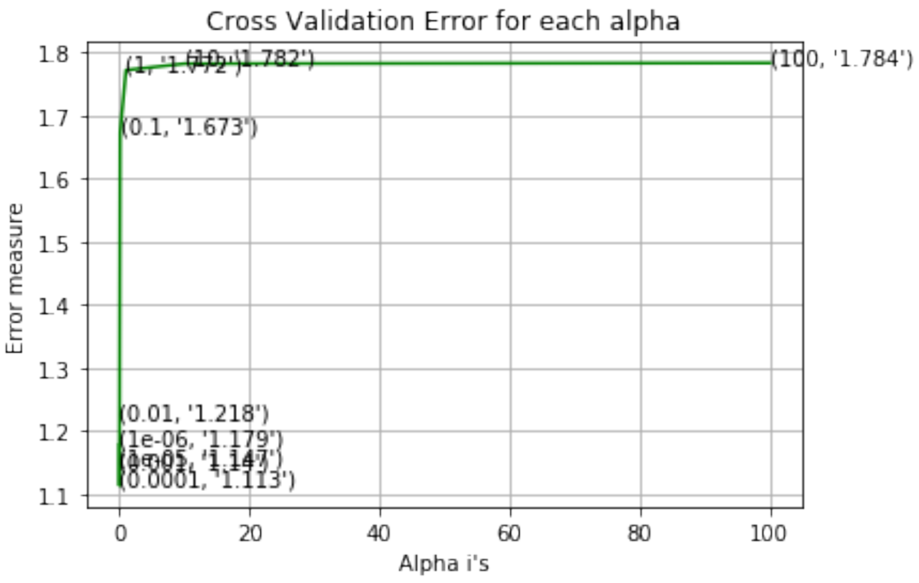
4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [67]:
```

# read more about SGDClassifier() at http://scikit-learn.org/stable/mo
# -------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, l
# 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 Stoc
# predict(X)    Predict class labels for samples in X.

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


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

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

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross val
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log 
```

```
for alpha = 1e-06
Log Loss : 1.1785175841081825
for alpha = 1e-05
Log Loss : 1.1467130108686638
for alpha = 0.0001
Log Loss : 1.1134290450021926
for alpha = 0.001
Log Loss : 1.140117936670911
```

```
for alpha = 0.01
Log Loss : 1.217832995709949
for alpha = 0.1
Log Loss : 1.673285647195903
for alpha = 1
Log Loss : 1.7722010859004989
for alpha = 10
Log Loss : 1.7824093883933187
for alpha = 100
Log Loss : 1.7835552070401708
```

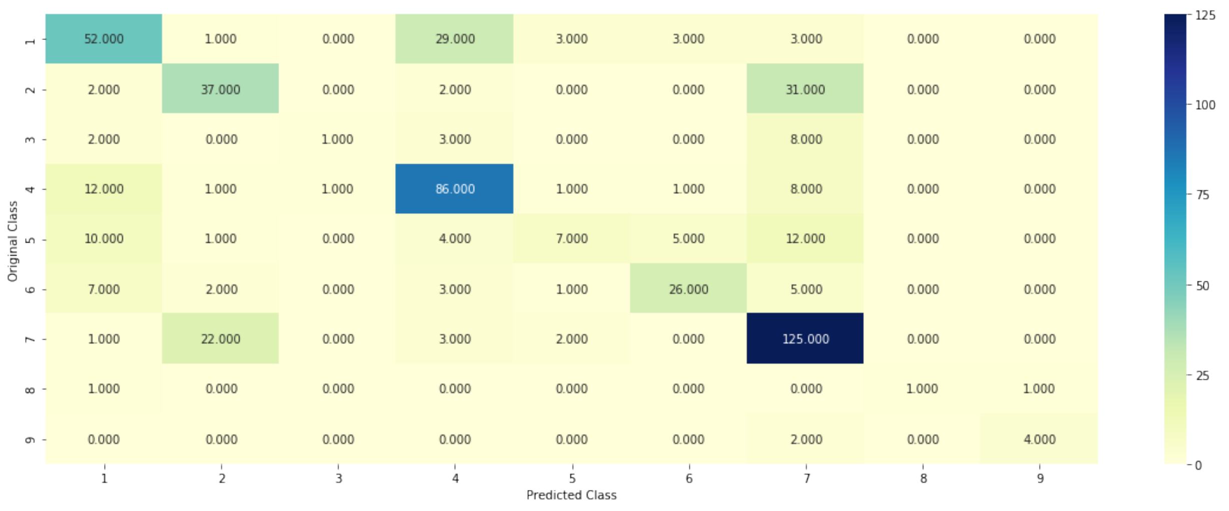


```
For values of best alpha =  0.0001 The train log loss is: 0.40614006
743241177
For values of best alpha =  0.0001 The cross validation log loss is:
1.1134290450021926
For values of best alpha =  0.0001 The test log loss is: 1.093404514
5654252
```

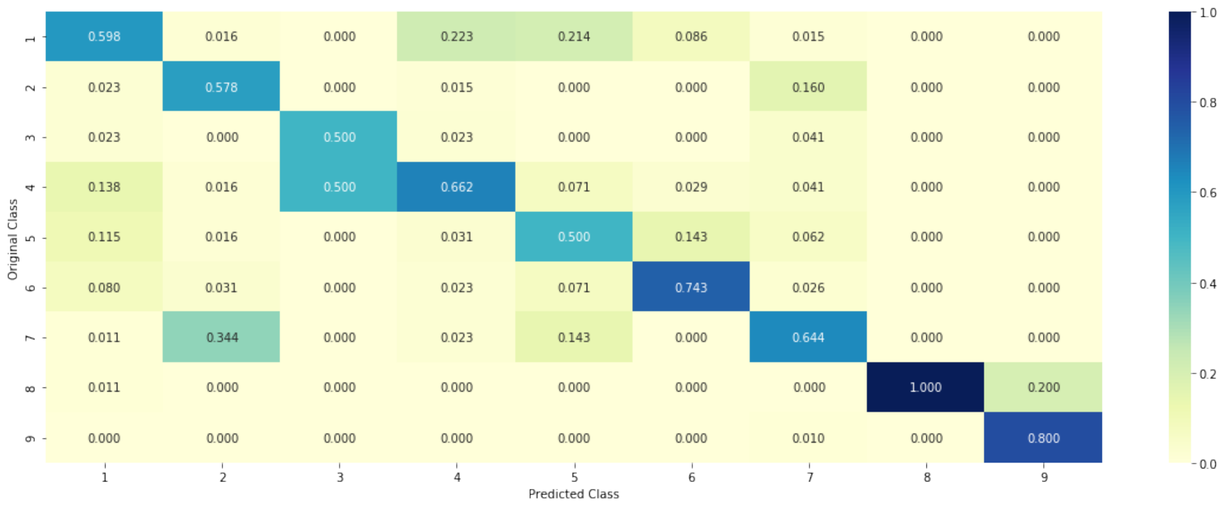
4.3.1.2. Testing the model with best hyper paramters

In [68]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mo
# -------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, l
# 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 Stoc
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course
#-------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], 
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_
```

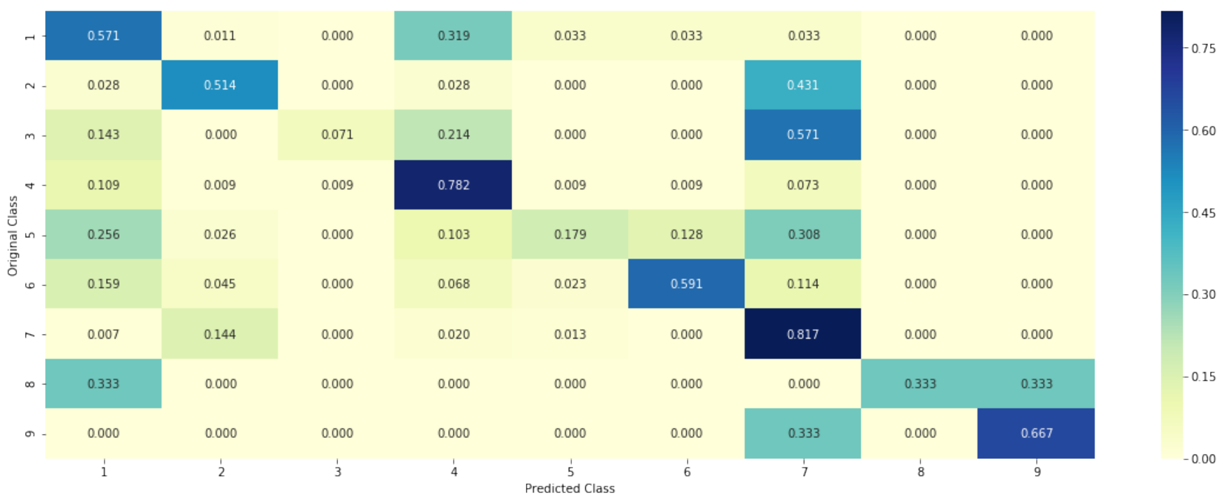
```
Log loss : 1.1134290450021926
Number of mis-classified points : 0.36278195488721804
-------------------- Confusion matrix --------------------
```



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



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



4.3.1.3. Feature Importance

In [69]:
```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_feature
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our que
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Pre
```

4.3.1.3.1. Correctly Classified point

In [71]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], 
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 25
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.4701 0.0099 0.0138 0.4612 0.0122
0.0071 0.0123 0.0068 0.0066]]
Actual Class : 1
--------------------------------------------------
361 Text feature [000548] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [75]:
```
test_point_index = 45
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
```

```
Predicted Class : 1
Predicted Class Probabilities: [[9.264e-01 3.100e-03 6.000e-04 5.000
e-02 3.000e-03 5.000e-04 1.370e-02
  1.800e-03 8.000e-04]]
Actual Class : 5
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

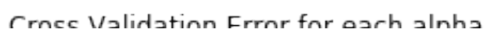
4.3.2. Without Class balancing

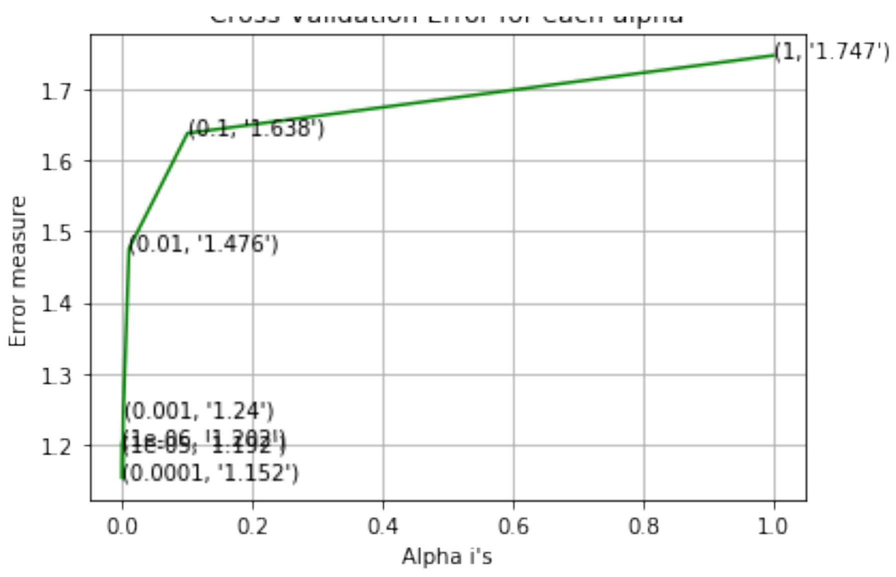
4.3.2.1. Hyper paramter tuning

In [76]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mo
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, l
# 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 Stoc
# predict(X)    Predict class labels for samples in X.

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



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

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

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

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross val
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log
```

```
for alpha = 1e-06
Log Loss : 1.2021637425196516
for alpha = 1e-05
Log Loss : 1.1918963652906986
for alpha = 0.0001
Log Loss : 1.151757108358039
for alpha = 0.001
Log Loss : 1.2397090256742735
for alpha = 0.01
Log Loss : 1.4755417553797656
for alpha = 0.1
Log Loss : 1.63805933836213
for alpha = 1
Log Loss : 1.7473823990507653
```



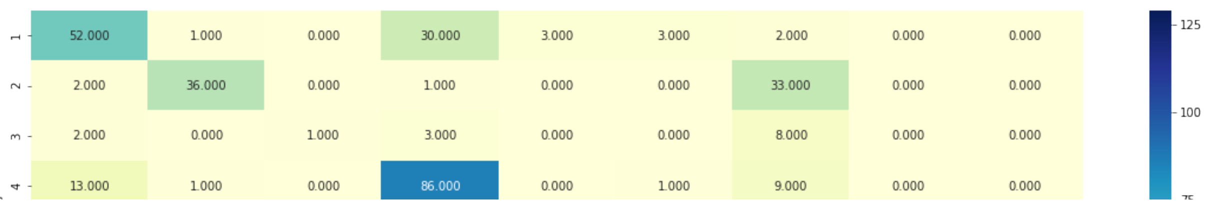


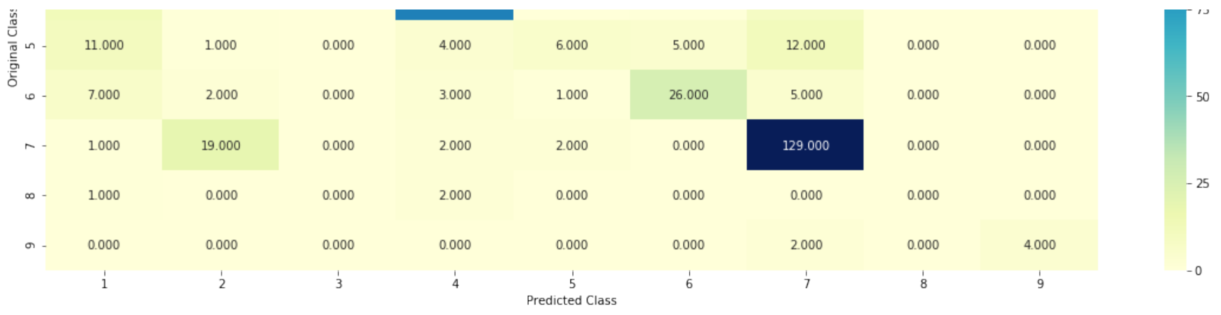
```
For values of best alpha =  0.0001 The train log loss is: 0.39568927
861704284
For values of best alpha =  0.0001 The cross validation log loss is:
1.151757108358039
For values of best alpha =  0.0001 The test log loss is: 1.126364453
5984654
```

4.3.2.2. Testing model with best hyper parameters

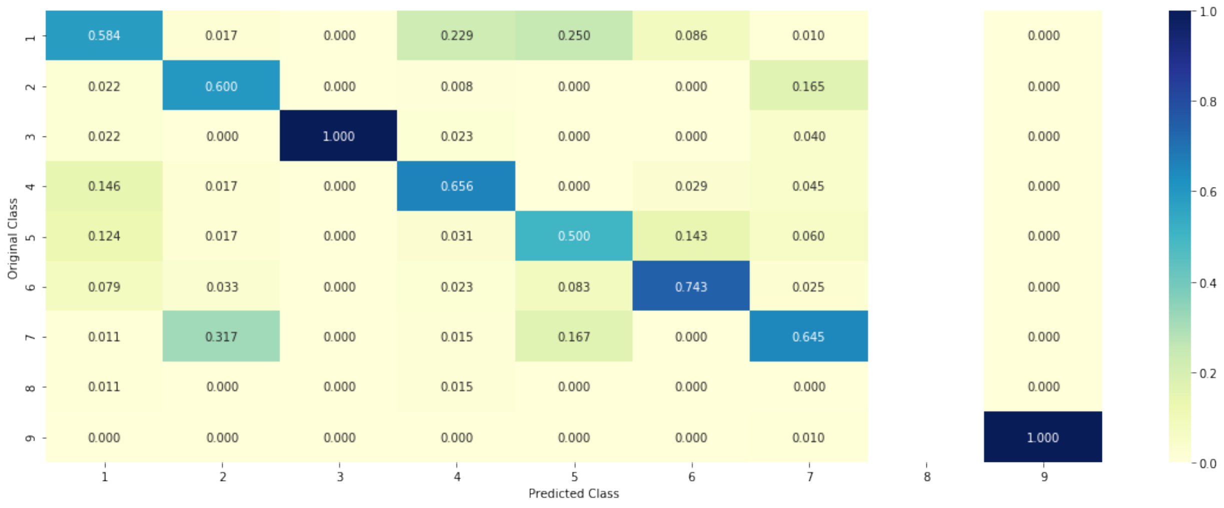
In [77]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_
```

```
Log loss : 1.151757108358039
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```

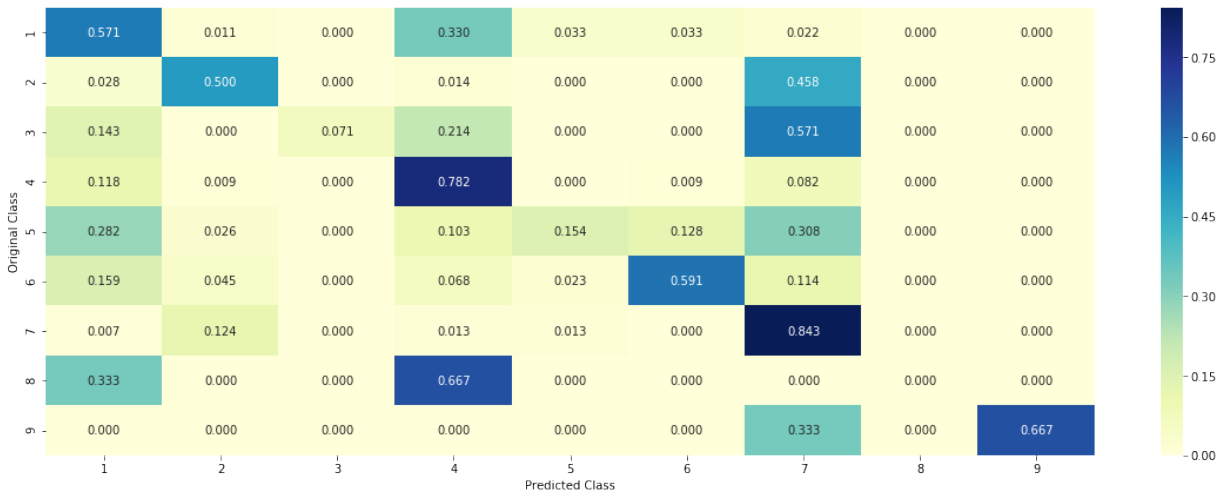




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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



4.3.2.3. Feature Importance, Correctly Classified point

In [80]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 5
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
```

```
Predicted Class : 1
Predicted Class Probabilities: [[7.414e-01 2.240e-02 1.300e-03 1.443
e-01 4.300e-03 2.300e-03 8.000e-02
  3.500e-03 6.000e-04]]
Actual Class : 1
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [81]:
```
test_point_index = 30
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0692 0.0124 0.0034 0.8856 0.0071
0.0108 0.0061 0.0033 0.0022]]
Actual Class : 1
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

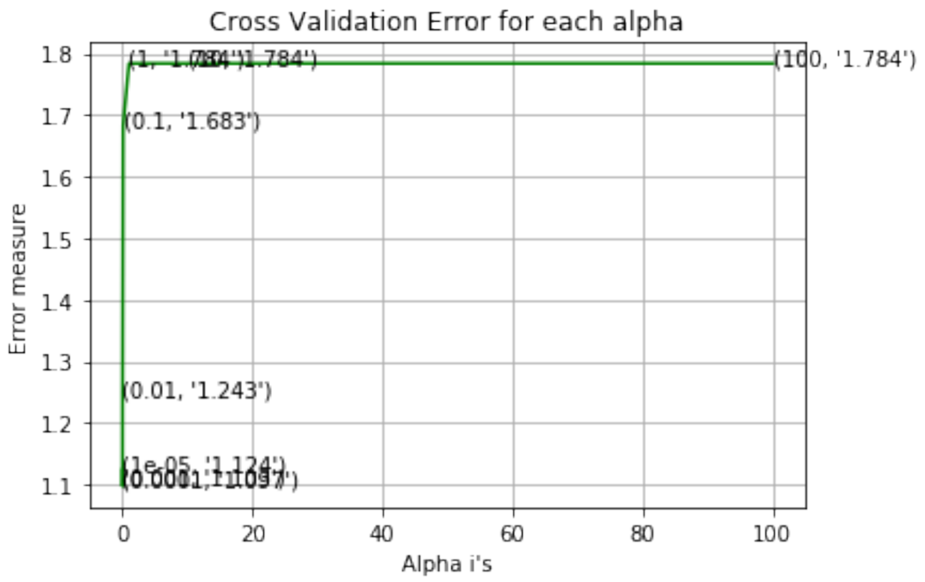
4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [82]:
```
# read more about support vector machines with linear kernals here htt

# ---------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinkin
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decis

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



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

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

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


best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanc
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], 
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross val
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log
```

```
for C = 1e-05
Log Loss : 1.1235670533233313
for C = 0.0001
Log Loss : 1.097110069926837
for C = 0.001
Log Loss : 1.102500685498744
for C = 0.01
Log Loss : 1.2432997429119372
for C = 0.1
Log Loss : 1.6829093740897547
for C = 1
Log Loss : 1.783808859241266
for C = 10
Log Loss : 1.78380889088798
for C = 100
Log Loss : 1.7838088805867707
```

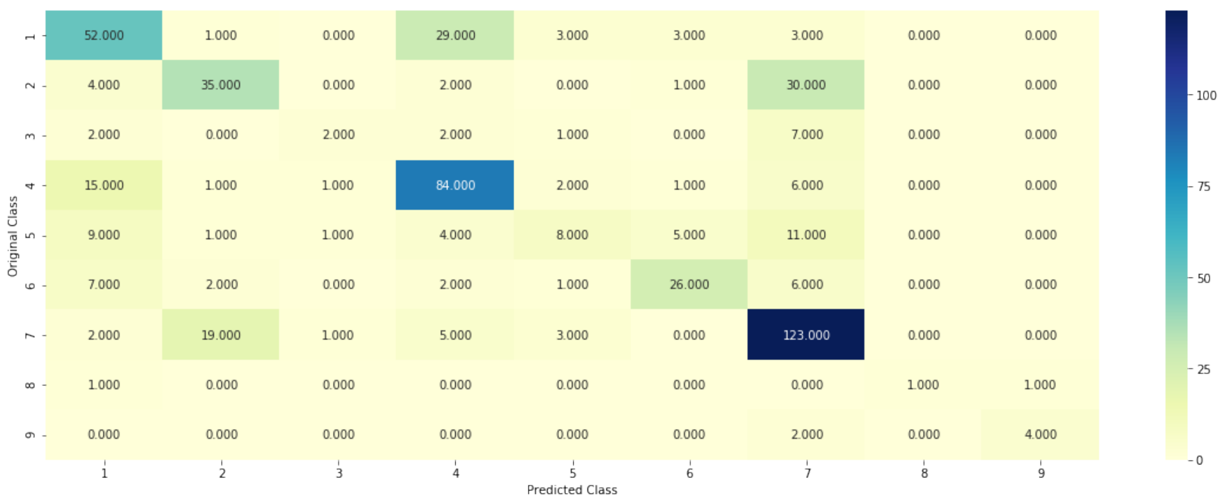


```
For values of best alpha =  0.0001 The train log loss is: 0.46056748
405811104
For values of best alpha =  0.0001 The cross validation log loss is:
1.097110069926837
For values of best alpha =  0.0001 The test log loss is: 1.119198278
9975787
```

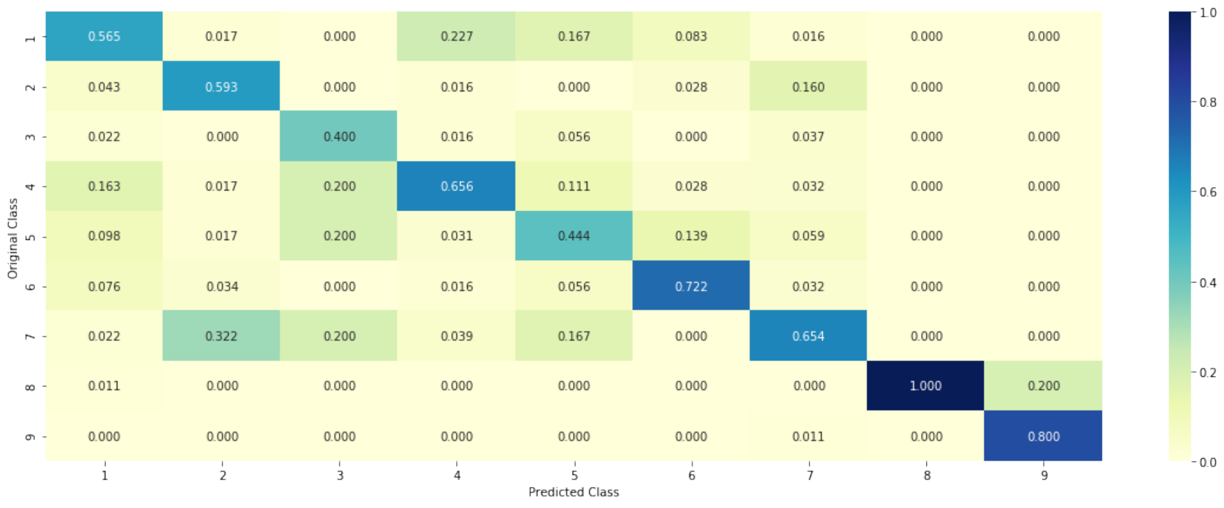
4.4.2. Testing model with best hyper parameters

In [83]:
```
# read more about support vector machines with linear kernals here htt
# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinkin
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decis

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


# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, clas
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_o
```

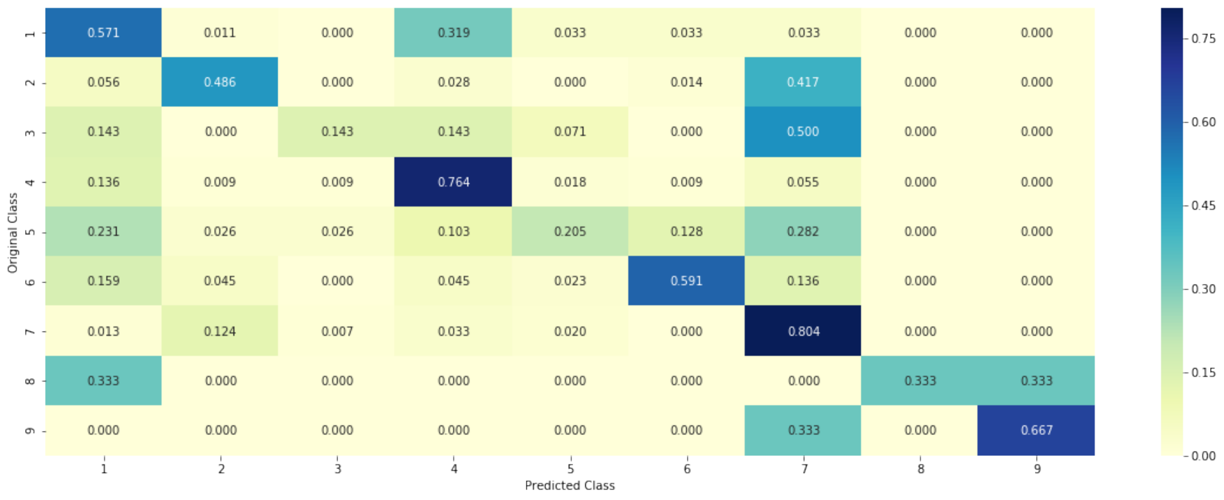
```
Log loss : 1.097110069926837
Number of mis-classified points : 0.37030075187969924
-------------------- Confusion matrix --------------------
```



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



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



	1	2	3	4	5	6	7	8	9
1	0.571	0.011	0.000	0.319	0.033	0.033	0.033	0.000	0.000
2	0.056	0.486	0.000	0.028	0.000	0.014	0.417	0.000	0.000
3	0.143	0.000	0.143	0.143	0.071	0.000	0.500	0.000	0.000
4	0.136	0.009	0.009	0.764	0.018	0.009	0.055	0.000	0.000
5	0.231	0.026	0.026	0.103	0.205	0.128	0.282	0.000	0.000
6	0.159	0.045	0.000	0.045	0.023	0.591	0.136	0.000	0.000
7	0.013	0.124	0.007	0.033	0.020	0.000	0.804	0.000	0.000
8	0.333	0.000	0.000	0.000	0.000	0.000	0.000	0.333	0.333
9	0.000	0.000	0.000	0.000	0.000	0.000	0.333	0.000	0.667

4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [85]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 6
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.065  0.0637 0.0203 0.3946 0.0963
0.0105 0.3388 0.0046 0.0062]]
Actual Class : 4
--------------------------------------------------
494 Text feature [10q11] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [87]:
```
test_point_index = 30
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1418 0.1011 0.0142 0.6434 0.0152
0.0266 0.0485 0.0036 0.0055]]
Actual Class : 1
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [88]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, 
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the giv
# 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
# --------------------------------


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

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

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

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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The
```

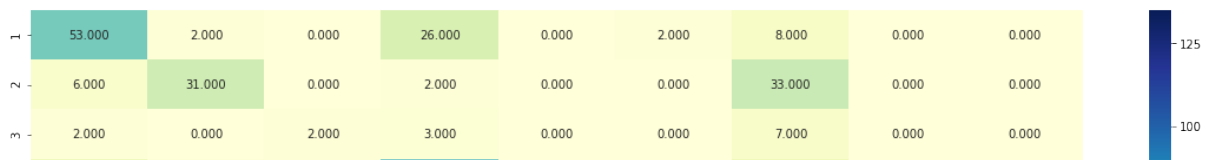
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.1983810581429026
for n_estimators = 100 and max depth =  10
Log Loss : 1.2006496153322366
for n_estimators = 200 and max depth =  5
Log Loss : 1.1888218549522238
for n_estimators = 200 and max depth =  10
Log Loss : 1.1858253680305344
for n_estimators = 500 and max depth =  5
```

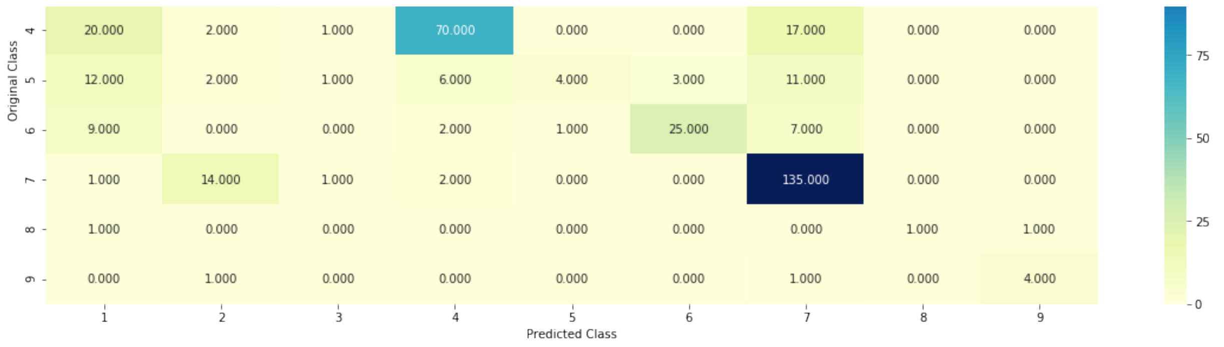
```
Log Loss : 1.1775509666635244
for n_estimators = 500 and max depth =  10
Log Loss : 1.1793733316569492
for n_estimators = 1000 and max depth =  5
Log Loss : 1.175951787711729
for n_estimators = 1000 and max depth =  10
Log Loss : 1.176893049742884
for n_estimators = 2000 and max depth =  5
Log Loss : 1.174028625562613
for n_estimators = 2000 and max depth =  10
Log Loss : 1.17506571365467
For values of best estimator =  2000 The train log loss is: 0.829953
0536193377
For values of best estimator =  2000 The cross validation log loss i
s: 1.174028625562613
For values of best estimator =  2000 The test log loss is: 1.2086183
010415124
```

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

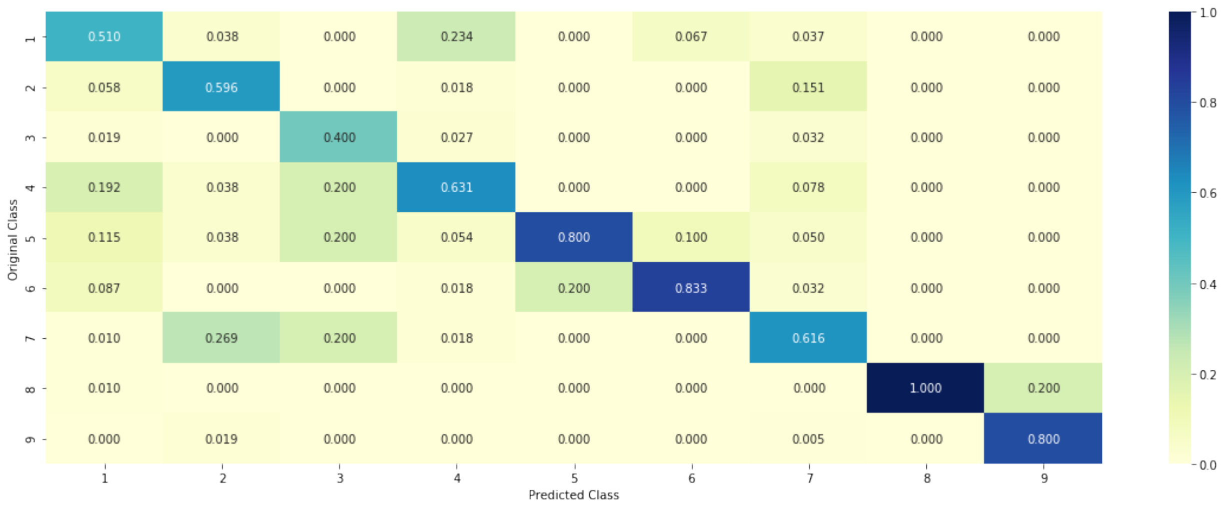
In [89]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, 
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the giv
# 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
# --------------------------------

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cr
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_o
```

```
Log loss : 1.174028625562613
Number of mis-classified points : 0.3890977443609023
-------------------- Confusion matrix --------------------
```

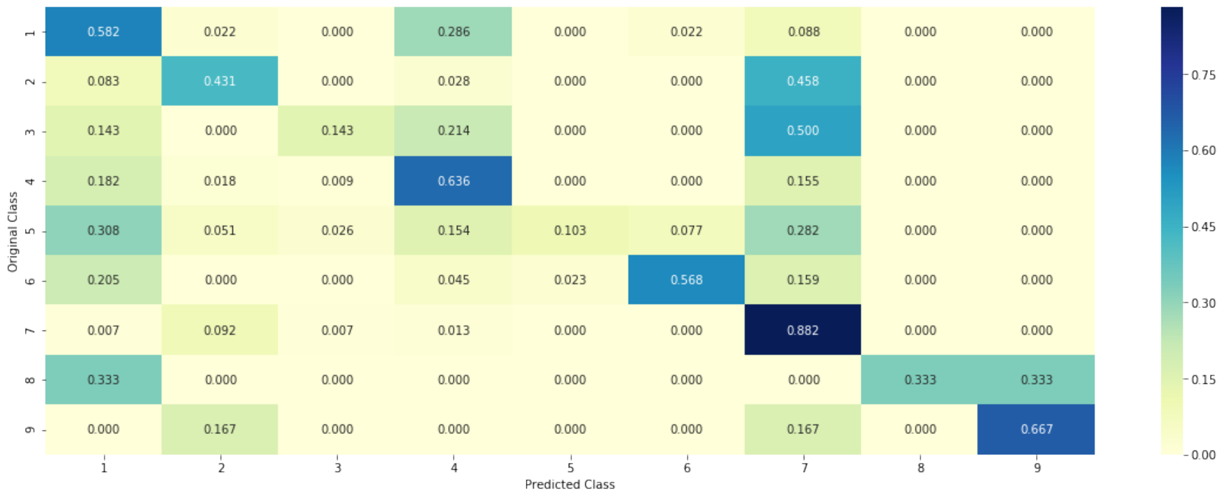




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [96]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cr
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2226 0.0315 0.0134 0.5818 0.0484
0.0442 0.0383 0.0049 0.015 ]]
Actual Class : 4
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [95]:
```
test_point_index = 100
no_feature = 25
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_p
```

```
Predicted Class : 1
Predicted Class Probabilities: [[5.763e-01 7.100e-03 1.340e-02 4.300
e-02 2.834e-01 6.180e-02 1.300e-02
  1.700e-03 4.000e-04]]
Actuall Class : 5
--------------------------------------------------
Out of the top  25  features  0 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [97]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the give
# 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
# --------------------------------


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

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

```
'''

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

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The tra
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cro
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The tes
```

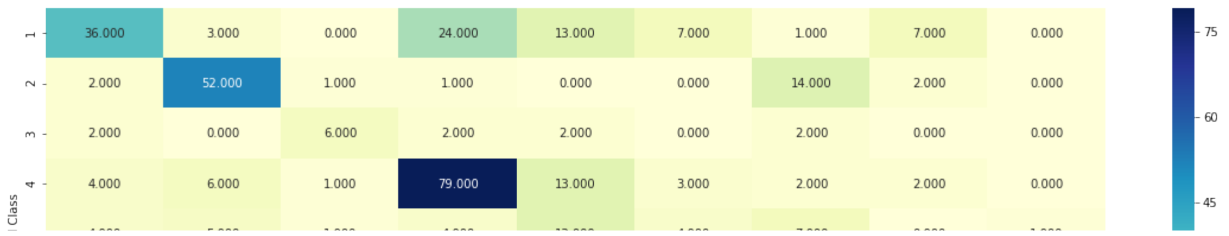
```
for n_estimators = 10 and max depth =  2
Log Loss : 2.2383987599657407
for n_estimators = 10 and max depth =  3
Log Loss : 1.7513231366704312
for n_estimators = 10 and max depth =  5

Log Loss : 1.6883882091764337
for n_estimators = 10 and max depth =  10
Log Loss : 1.947425712352598
for n_estimators = 50 and max depth =  2
Log Loss : 1.791881321320487
for n_estimators = 50 and max depth =  3
Log Loss : 1.6037535227931847
for n_estimators = 50 and max depth =  5
Log Loss : 1.4330271256520364
for n_estimators = 50 and max depth =  10
Log Loss : 1.7147416902717125
for n_estimators = 100 and max depth =  2
Log Loss : 1.6546520305617434
for n_estimators = 100 and max depth =  3
Log Loss : 1.6412814830912448
for n_estimators = 100 and max depth =  5
Log Loss : 1.3948859003319565
for n_estimators = 100 and max depth =  10
Log Loss : 1.765021769526246
for n_estimators = 200 and max depth =  2
Log Loss : 1.6952050911266836
for n_estimators = 200 and max depth =  3
Log Loss : 1.687819156787606
for n_estimators = 200 and max depth =  5
Log Loss : 1.44830262535371
for n_estimators = 200 and max depth =  10
Log Loss : 1.8027410942773414
for n_estimators = 500 and max depth =  2
Log Loss : 1.7818212493281296
for n_estimators = 500 and max depth =  3
Log Loss : 1.712233932362757
for n_estimators = 500 and max depth =  5
Log Loss : 1.5005013380938976
for n_estimators = 500 and max depth =  10
```

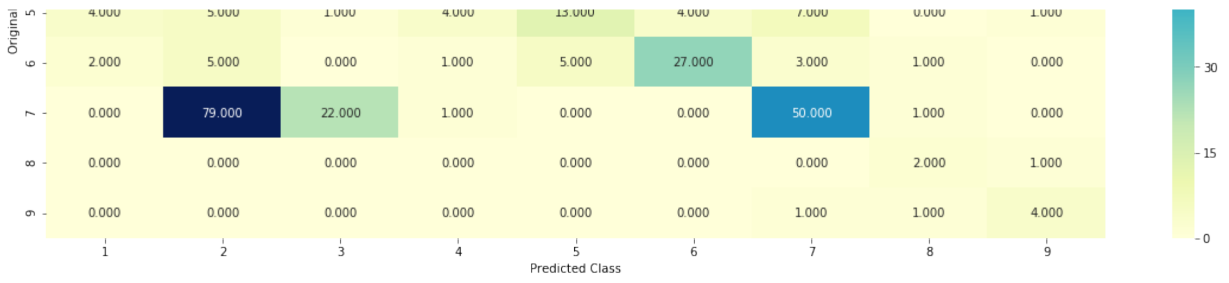
```
Log Loss : 1.8133753194585522
for n_estimators = 1000 and max depth =  2
Log Loss : 1.7681104175995912
for n_estimators = 1000 and max depth =  3
Log Loss : 1.7206393723289777
for n_estimators = 1000 and max depth =  5
Log Loss : 1.487107551090544
for n_estimators = 1000 and max depth =  10
Log Loss : 1.7514803475638165
For values of best alpha =  100 The train log loss is: 0.05294057805
651875
For values of best alpha =  100 The cross validation log loss is: 1.
3948859003319558
For values of best alpha =  100 The test log loss is: 1.350929897590
002
```

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

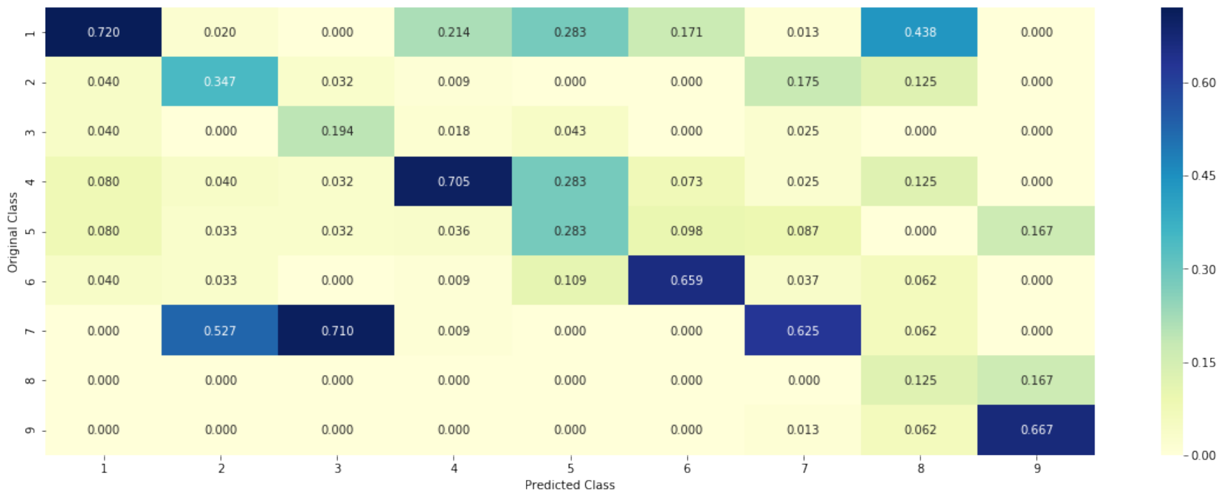
In [98]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto'
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the give
# 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-
# --------------------------------

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_
```

```
Log loss : 1.3948859003319563
Number of mis-classified points : 0.4943609022556391
-------------------- Confusion matrix --------------------
```

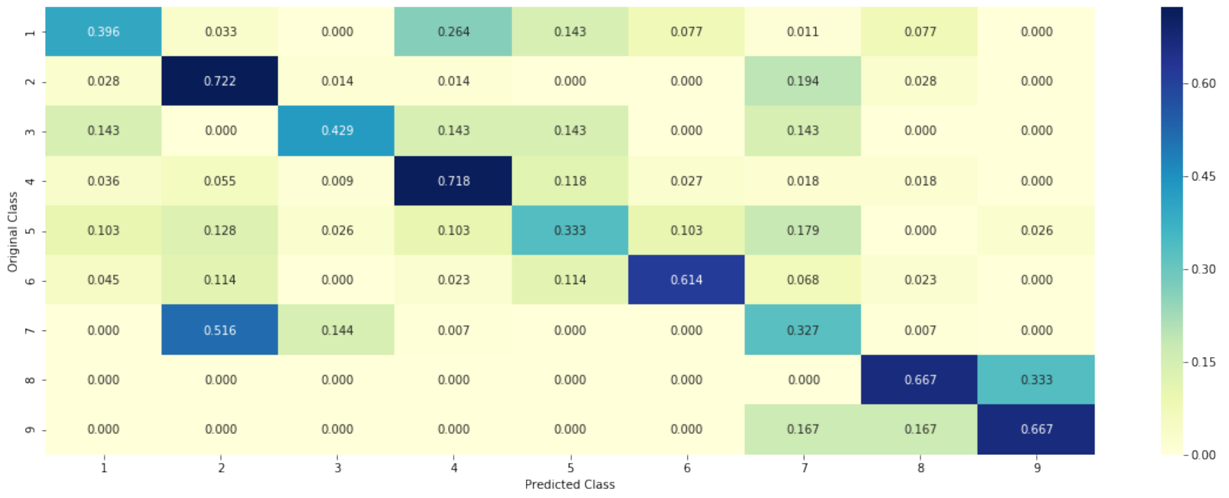




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [99]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], cr
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)
```

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0583 0.0489 0.0543 0.6874 0.0164
0.0293 0.0112 0.0615 0.0326]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [100]:
```
test_point_index = 42
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0064 0.7952 0.0246 0.0091 0.0095
0.0205 0.1211 0.007  0.0067]]
Actual Class : 2
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [101]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
# -------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
# 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 Stoc
# predict(X)     Predict class labels for samples in X.

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



# read more about support vector machines with linear kernals here htt
# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinkin
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decis

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



# read more about support vector machines with linear kernals here htt
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, 
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])     Fit the SVM model according to the giv
# 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
# --------------------------------
```

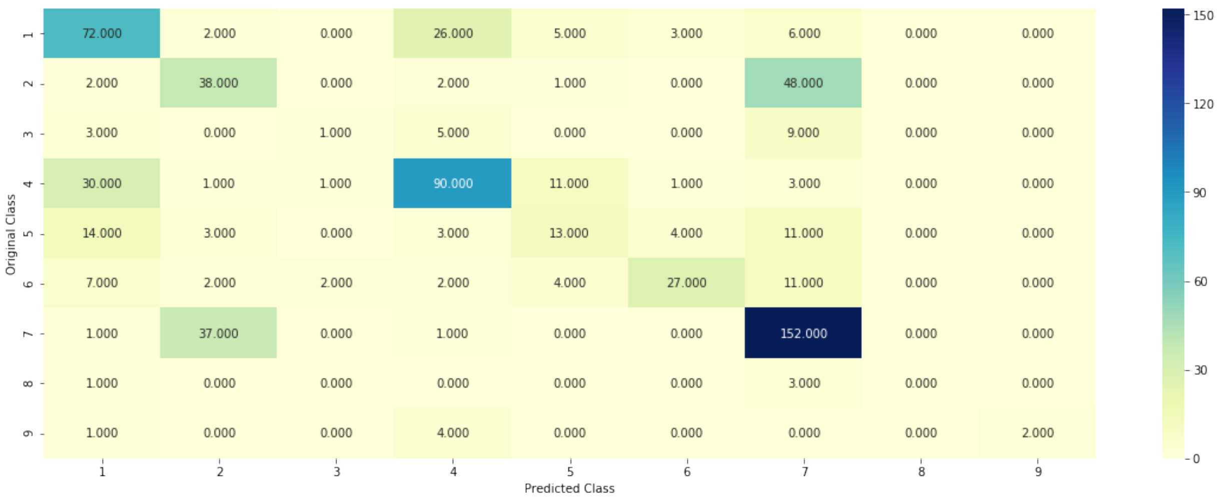
```
clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weig
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

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


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

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_c
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, si
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predi
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_clf
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

```
Logistic Regression :  Log Loss: 1.15
Support vector machines : Log Loss: 1.78
Naive Bayes : Log Loss: 1.23
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.17
7
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.02
8
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.49
4
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.21
1
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.46
2
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.9
05
```

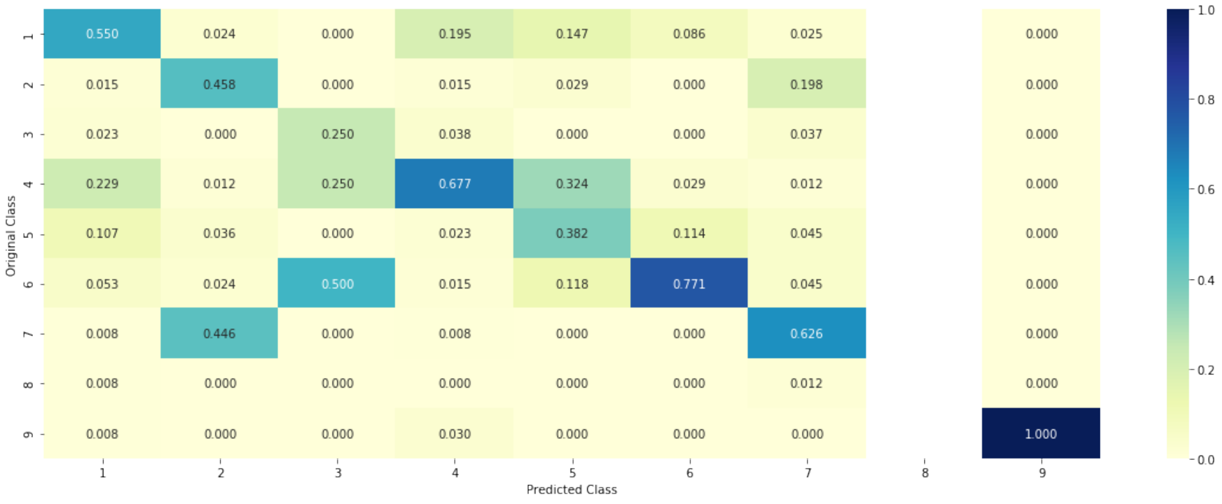
4.7.2 testing the model with the best hyper parameters

```
In [102]: lr = LogisticRegression(C=0.1)
          sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], m
          sclf.fit(train_x_onehotCoding, train_y)

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

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

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

          print("Number of missclassified point :", np.count_nonzero((sclf.predic
          plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_oneh
```

```
Log loss (train) on the stacking classifier : 0.5581683561833035
Log loss (CV) on the stacking classifier : 1.2110328764892686
Log loss (test) on the stacking classifier : 1.2344391630274536
Number of missclassified point : 0.40601503759398494
-------------------- Confusion matrix --------------------
```

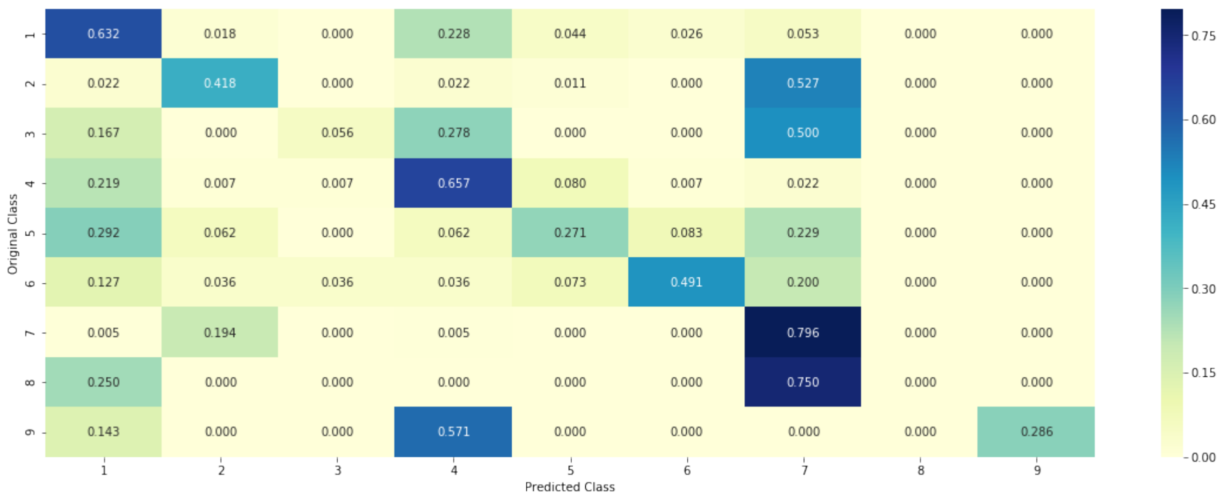


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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

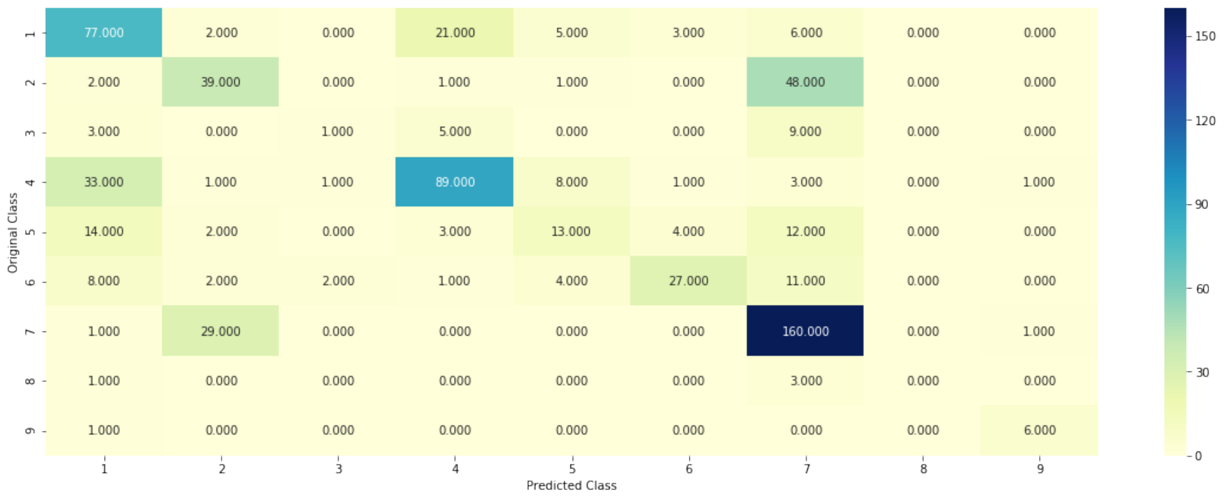
```
-------------------- Recall matrix (Row sum=1) --------------------
```



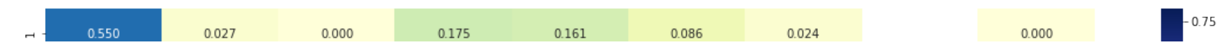
4.7.3 Maximum Voting classifier

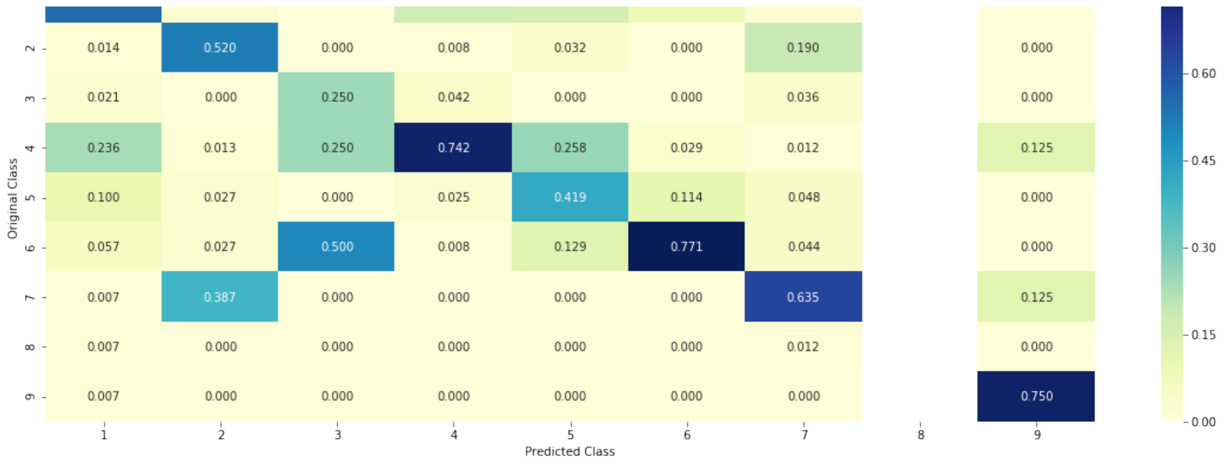
In [103]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemb
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, 
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.p
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vc
print("Number of missclassified point :", np.count_nonzero((vclf.predi
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_one
```

```
Log loss (train) on the VotingClassifier : 0.8209129867460512
Log loss (CV) on the VotingClassifier : 1.2029054454616537
Log loss (test) on the VotingClassifier : 1.2200751341978684
Number of missclassified point : 0.38045112781954887
-------------------- Confusion matrix --------------------
```

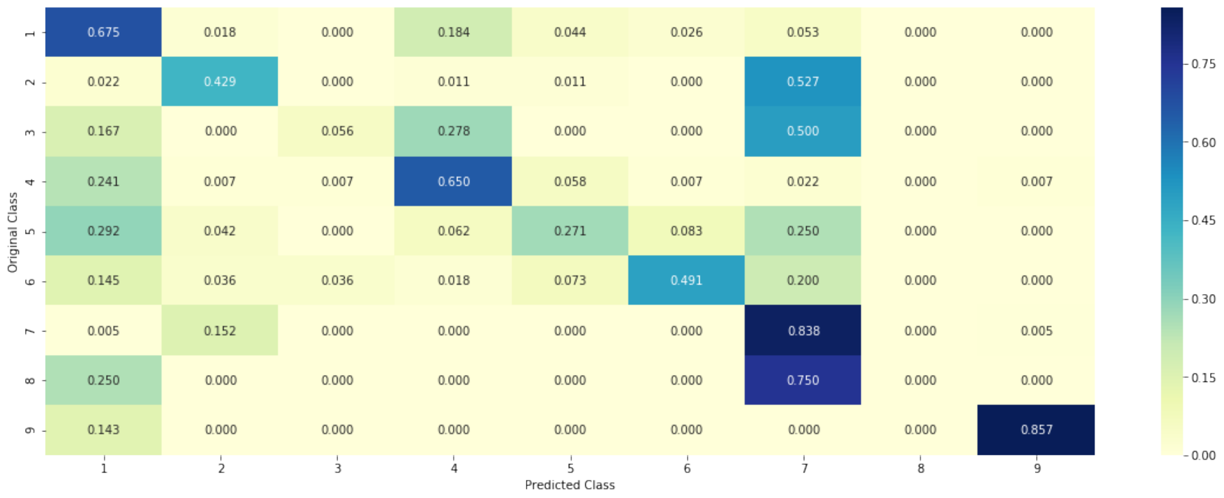


```
-------------------- Precision matrix (Columm Sum=1) ---------------
-----
```





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



5. OBSERVATION

In [1]:
```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Algorithm used","Train Score","Test Score","CV score

x.add_row(["Naive Bayes", 0.54616,1.22585,1.26962,39.09])
x.add_row(["KNN", 0.63548,1.12654,1.10334,40.60])
x.add_row(["LR (Balanced data)", 0.40614, 1.11342,  1.09340, 36.27])
x.add_row(["LR (Without Balanced data)",0.39568, 1.15175, 1.12636, 36.
x.add_row(["Linear SVM",0.46056,  1.09711, 1.11919, 37.03])
x.add_row(["Random Forest (one hot encoding)",0.82995, 1.17402, 1.2086
x.add_row(["Random Forest (Response Coding)", 0.05294, 1.39488, 1.3509
x.add_row(["Stacking model (LR, SVM, NB)",0.55816, 1.21103, 1.23443, 4
x.add_row(["Maximum Voting Classifier", 0.82091,  1.20290, 1.22007, 38
print(x)
```

```
+----------------------------------+-------------+------------+-----
-----+------------------+
|          Algorithm used          | Train Score | Test Score | CV s
core | % misclassified |
+----------------------------------+-------------+------------+-----
-----+------------------+
|           Naive Bayes            |   0.54616   |  1.22585   | 1.26
962  |      39.09       |
|               KNN                |   0.63548   |  1.12654   | 1.10
334  |       40.6       |
|        LR (Balanced data)        |   0.40614   |  1.11342   |  1.0
934  |      36.27       |
|    LR (Without Balanced data)    |   0.39568   |  1.15175   | 1.12
636  |      36.09       |
|            Linear SVM            |   0.46056   |  1.09711   | 1.11
919  |      37.03       |
| Random Forest (one hot encoding) |   0.82995   |  1.17402   | 1.20
861  |       38.9       |
| Random Forest (Response Coding)  |   0.05294   |  1.39488   | 1.35
092  |      49.43       |
|   Stacking model (LR, SVM, NB)   |   0.55816   |  1.21103   | 1.23
443  |       40.6       |
|    Maximum Voting Classifier     |   0.82091   |   1.2029   | 1.22
007  |      38.04       |
+----------------------------------+-------------+------------+-----
-----+------------------+
```