

Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

1. <https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25> (<https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25>)
2. <https://www.youtube.com/watch?v=UwbuW7oK8rk> (<https://www.youtube.com/watch?v=UwbuW7oK8rk>)
3. <https://www.youtube.com/watch?v=qxXRKVompl8> (<https://www.youtube.com/watch?v=qxXRKVompl8>)

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learning Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.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
```

C:\Users\Akhil\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:144: FutureWarning: The sklearn.metrics.classification module is deprecated in version 0.22 and will be removed in version 0.24. The corresponding classes / functions should instead be imported from sklearn.metrics. Anything that cannot be imported from sklearn.metrics is now part of the private API.

```
warnings.warn(message, FutureWarning)
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [3]:

```
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(10)
```

Number of data points : 3321

Number of features : 4

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

Out[3]:

	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
5	5	CBL	V391I	4
6	6	CBL	V430M	5
7	7	CBL	Deletion	1
8	8	CBL	Y371H	4
9	9	CBL	C384R	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 [4]:

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

Number of data points : 3321

Number of features : 2

Features : ['ID' 'TEXT']

Out[4]:

	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 [5]:

```
# Loading stop words from nltk library
stop_words = set(stopwords.words('english'))

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

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

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

In [6]:

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

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 193.8427166 seconds
```

In [9]:

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

Out[9]:

	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 [10]:

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

Out[10]:

	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 [11]:

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


In [13]:

```
print(result[result['ID']==1109])
result[result['ID']==1277]
```

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

Out[13]:

	ID	Gene	Variation	Class	TEXT
1277	1277	ARID5B	Truncating Mutations	1	ARID5B Truncating Mutations

In [16]:

```
result.head()
```

Out[16]:

	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...

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 [17]:

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

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

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

In [18]:

```
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

In [19]:

```
train_df.head()
```

Out[19]:

	ID	Gene	Variation	Class	TEXT
2094	2094	CDK12	Y901C	4	introduction brca1 brca2 mutations found 20 hi...
647	647	CDKN2A	D84G	4	background point mutations tumor suppressor ge...
86	86	CCNE1	Amplification	7	variety results point transcription factor e2f...
1698	1698	PMS2	I18V	3	identification high risk disease causing const...
1614	1614	VHL	H191D	4	chuvash polycythemia cp rare congenital form p...

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

In [20]:

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

#print(train_df['Class'].value_counts())
#print(train_class_distribution)

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

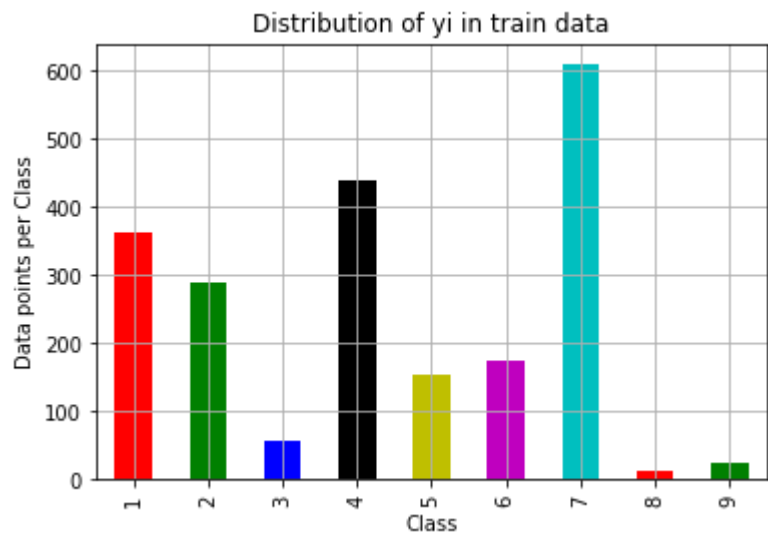
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':', train_class_distribution.values[i],
          '(', np.round((train_class_distribution.values[i]/train_df.shape[0]*100), 3), '%)')

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

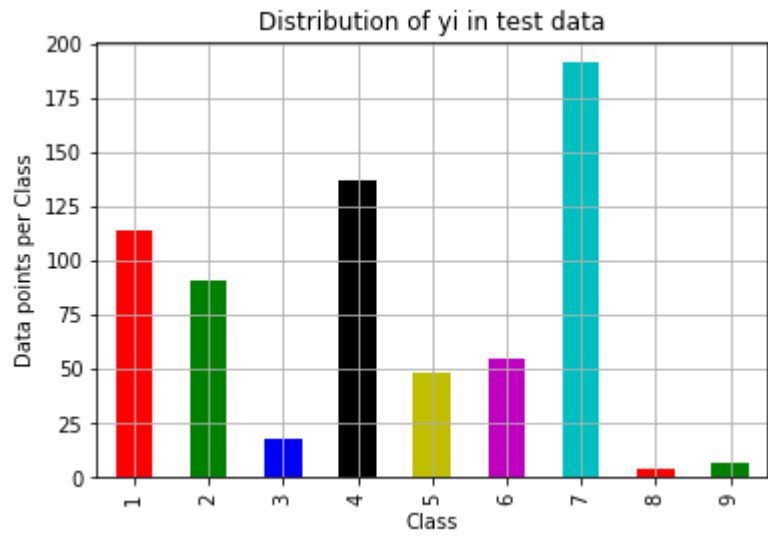
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':', test_class_distribution.values[i],
          '(', np.round((test_class_distribution.values[i]/test_df.shape[0]*100), 3), '%)')

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

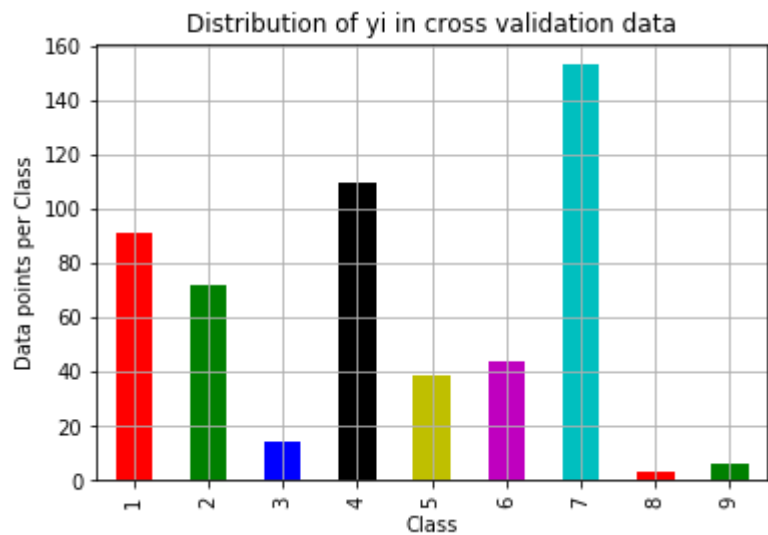
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':', cv_class_distribution.values[i],
          '(', np.round((cv_class_distribution.values[i]/cv_df.shape[0]*100), 3), '%)')
```



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



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



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

3.2 Prediction using a 'Random' Model

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

In [21]:

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

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

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

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

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

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

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

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


In [22]:

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

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

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

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


Log loss on Cross Validation Data using Random Model 2.545942788564111

Log loss on Test Data using Random Model 2.468273503272135

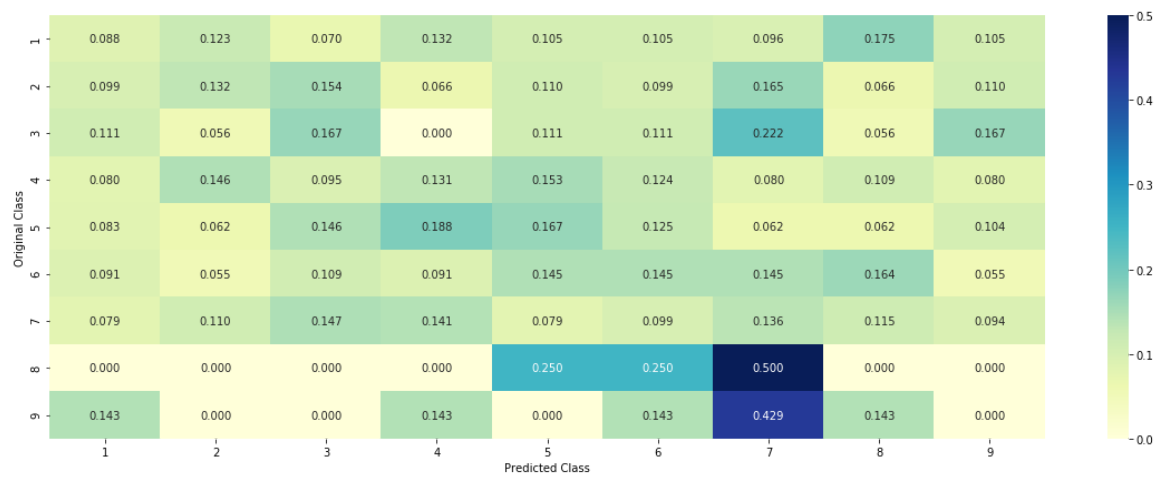
----- Confusion matrix -----



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



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



3.3 Univariate Analysis

In [23]:

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

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

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

    # denominator will contain the number of times that particular feature occurred in whole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to particular class
```

```

# vec is 9 dimensional vector
vec = []
for k in range(1,10):
    # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=='BRCA1')])
    #
    # ID      Gene      Variation  Class
    # 2470    2470    BRCA1      S1715C    1
    # 2486    2486    BRCA1      S1841R    1
    # 2614    2614    BRCA1      M1R      1
    # 2432    2432    BRCA1      L1657P    1
    # 2567    2567    BRCA1      T1685A    1
    # 2583    2583    BRCA1      E1660G    1
    # 2634    2634    BRCA1      W1718L    1
    # cls_cnt.shape[0] will return the number of rows

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

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #
    # {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.06818181818181817, 0.13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.06818181818181817, 0.06818181818181817, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608, 0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702, 0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999, 0.066666666666666666, 0.066666666666666666],
    #      ...
    #      }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

```

```

        gv_fea.append(gv_dict[row[feature]])
    else:
        gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#         gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea

```

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

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

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 [24]:

```

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 : 235

BRCA1	172
EGFR	99
TP53	96
BRCA2	78
PTEN	73
KIT	66
BRAF	65
ERBB2	49
ALK	42
MTOR	37

Name: Gene, dtype: int64

In [25]:

```

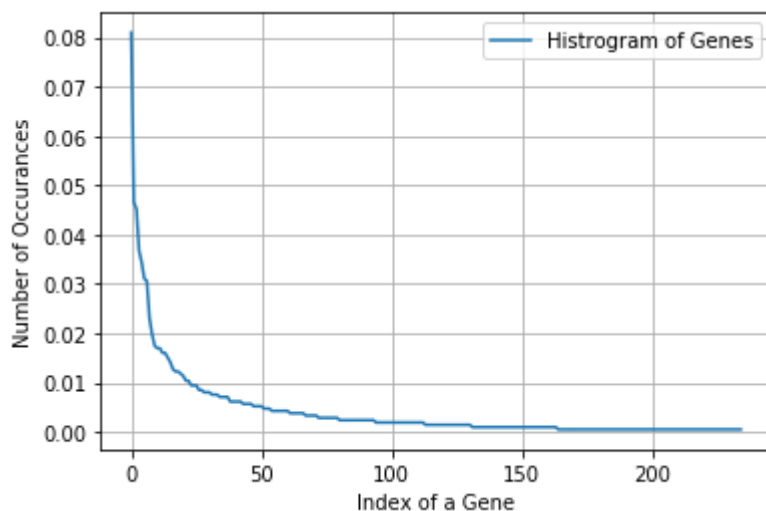
print("Ans: There are", unique_genes.shape[0] , "different categories of genes in the train data, and they are distributed as follows",)

```

Ans: There are 235 different categories of genes in the train data, and they are distributed as follows

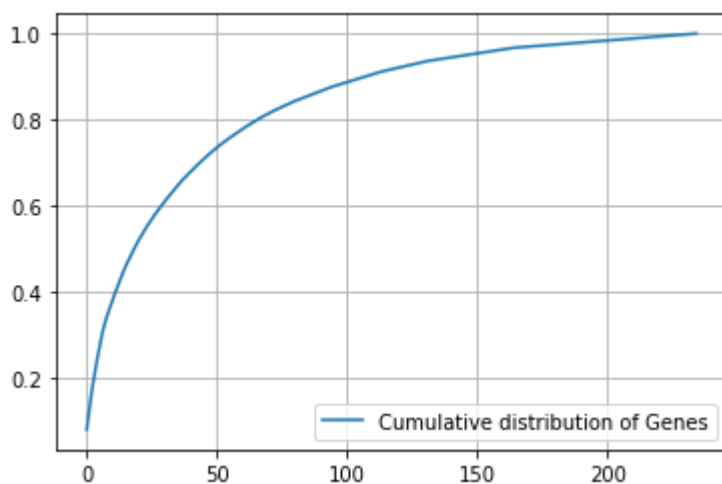
In [26]:

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



In [27]:

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



Q3. How to featurize this Gene feature ?

Ans.there are two ways we can featurize this variable check out this video:

<https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/>

1. One hot Encoding
2. Response coding

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

In [28]:

```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
```

In [29]:

```
print("train_gene_feature_responseCoding is converted feature using response coding meth
od. The shape of gene feature:", train_gene_feature_responseCoding.shape)
```

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

In [30]:

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

In [31]:

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

Out[31]:

```
2094    CDK12
647     CDKN2A
86      CCNE1
1698    PMS2
1614    VHL
Name: Gene, dtype: object
```

In [33]:

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature: (2124, 235)
```

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 [34]:

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

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

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

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

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

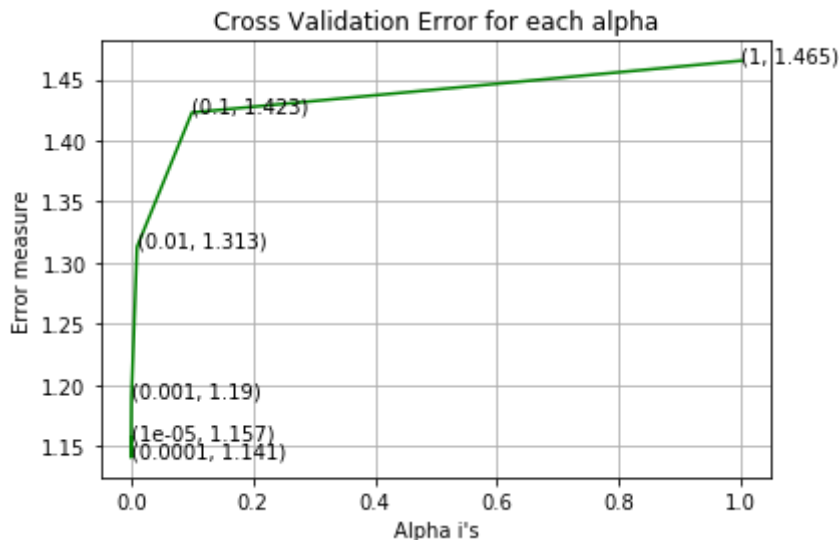
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], np.round(txt, 3)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error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

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

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

```
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

For values of alpha = 1e-05 The log loss is: 1.156637687040569
 For values of alpha = 0.0001 The log loss is: 1.1406114135720467
 For values of alpha = 0.001 The log loss is: 1.190384235375181
 For values of alpha = 0.01 The log loss is: 1.312946968850221
 For values of alpha = 0.1 The log loss is: 1.4229369503065068
 For values of alpha = 1 The log loss is: 1.465339976377039



For values of best alpha = 0.0001 The train log loss is: 1.0134912348327223
 For values of best alpha = 0.0001 The cross validation log loss is: 1.1406114135720467
 For values of best alpha = 0.0001 The test log loss is: 1.1745838641936008

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 [37]:

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

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

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

Q6. How many data points in Test and CV datasets are covered by the 235 genes in train dataset?

Ans

1. In test data 648 out of 665 : 97.44360902255639
2. In cross validation data 515 out of 532 : 96.80451127819549

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

In [38]:

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

```
Number of Unique Variations : 1928
Truncating_Mutations      67
Amplification              47
Deletion                  46
Fusions                   20
Overexpression             3
P34R                      2
G12V                     2
G13C                     2
T286A                    2
P130S                    2
Name: Variation, dtype: int64
```

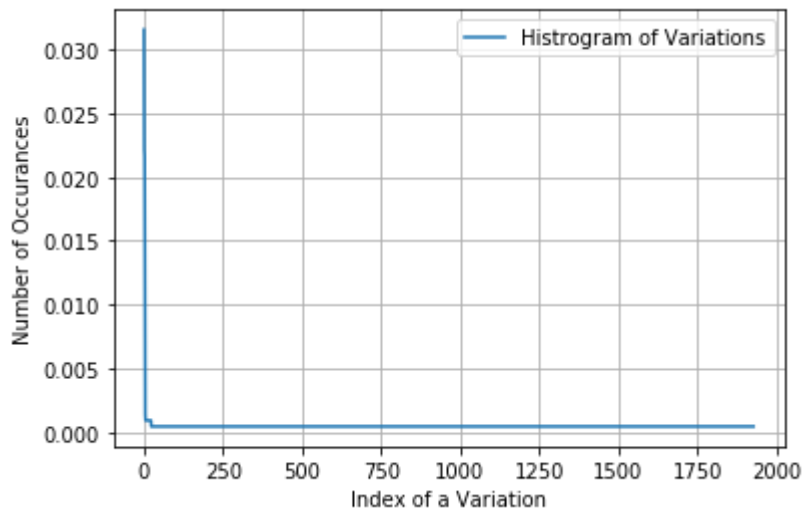
In [39]:

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

Ans: There are 1928 different categories of variations in the train data,
and they are distributed as follows

In [40]:

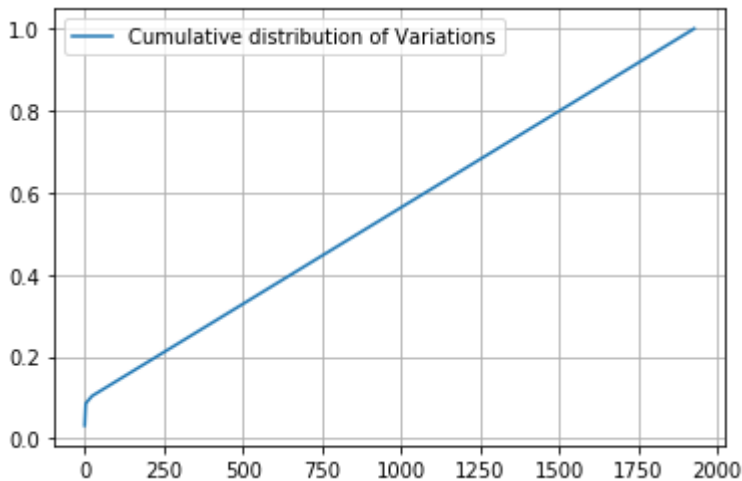
```
s = sum(unique_variations.values);  
h = unique_variations.values/s;  
plt.plot(h, label="Histogram of Variations")  
plt.xlabel('Index of a Variation')  
plt.ylabel('Number of Occurances')  
plt.legend()  
plt.grid()  
plt.show()
```



In [41]:

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

[0.03154426 0.05367232 0.07532957 ... 0.99905838 0.99952919 1.]



Q9. How to featurize this Variation feature ?

Ans. There are two ways we can featurize this variable check out this video:

<https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/>

1. One hot Encoding
2. Response coding

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

In [42]:

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

In [43]:

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

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

In [44]:

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

In [45]:

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

train_variation_feature_onehotEncoded is converted feature using the one-hot encoding method. The shape of Variation feature: (2124, 1956)

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

Let's build a model just like the earlier!

In [46]:

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

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

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

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

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

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

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

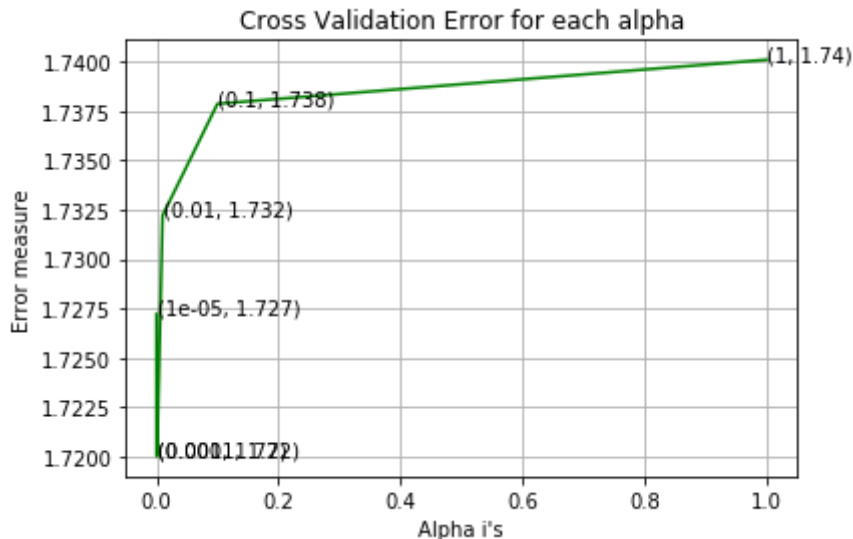
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], np.round(txt, 3)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error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

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

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

```
predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

For values of alpha = 1e-05 The log loss is: 1.7272139066485623
 For values of alpha = 0.0001 The log loss is: 1.7200647652038614
 For values of alpha = 0.001 The log loss is: 1.72001087071577
 For values of alpha = 0.01 The log loss is: 1.7322033118097195
 For values of alpha = 0.1 The log loss is: 1.7378572218755242
 For values of alpha = 1 The log loss is: 1.7400862545975924



For values of best alpha = 0.001 The train log loss is: 1.0853782727080554
 For values of best alpha = 0.001 The cross validation log loss is: 1.72001087071577
 For values of best alpha = 0.001 The test log loss is: 1.7004253610750355

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 [47]:

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

Q12. How many data points are covered by total 1928 genes in test and cross validation data sets?

Ans

1. In test data 68 out of 665 : 10.225563909774436
2. In cross validation data 55 out of 532 : 10.338345864661653

3.2.3 Univariate Analysis on Text Feature

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

In [48]:

```
# 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 [49]:

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

In [50]:

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

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

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

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

Total number of unique words in train data : 53993

In [51]:

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

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

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

In [53]:

```
#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 [54]:

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

In [55]:

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

In [56]:

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

In [13]:

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

In [58]:

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

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

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

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

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

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

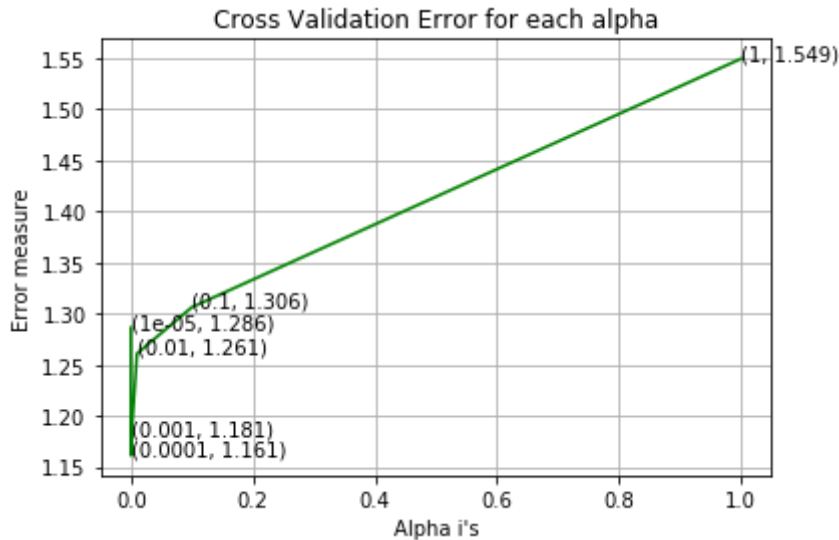
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], np.round(txt, 3)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error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

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

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

```
is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_text_feature_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

For values of alpha = 1e-05 The log loss is: 1.286200204927833
 For values of alpha = 0.0001 The log loss is: 1.161254388559632
 For values of alpha = 0.001 The log loss is: 1.1812751162429704
 For values of alpha = 0.01 The log loss is: 1.2608458087037864
 For values of alpha = 0.1 The log loss is: 1.306160960040914
 For values of alpha = 1 The log loss is: 1.548991762676919



For values of best alpha = 0.0001 The train log loss is: 0.5986140518040964
 For values of best alpha = 0.0001 The cross validation log loss is: 1.161254388559632
 For values of best alpha = 0.0001 The test log loss is: 1.0980293032404917

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

Ans. Yes, it seems like!

In [59]:

```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=3)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

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

In [60]:

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

98.416 % of word of test data appeared in train data

97.901 % of word of Cross Validation appeared in train data

4. Machine Learning Models

In [61]:

```
#Data preparation for ML models.

#Misc. functions for ML models

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

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

In [62]:

```
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 [63]:

```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3)

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

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

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

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

Stacking the three types of features

In [64]:

```
# merging gene, variance and text features

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

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding, train_variation_f
eature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding, test_variation_feat
ure_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding, cv_variation_feature_on
ehotCoding))

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

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

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

train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding, train_vari
ation_feature_responseCoding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding, test_variati
on_feature_responseCoding))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding, cv_variation_fea
ture_responseCoding))

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature_r
esponseCoding))
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_resp
onseCoding))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_responseCo
ding))
```


In [65]:

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

```
One hot encoding features :
(number of data points * number of features) in train data = (2124, 56184)
(number of data points * number of features) in test data = (665, 56184)
(number of data points * number of features) in cross validation data = (532, 56184)
```

In [66]:

```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_responseCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_responseCoding.shape)
```

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

4.1. Base Line Model

4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [67]:

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

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

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

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

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

```

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

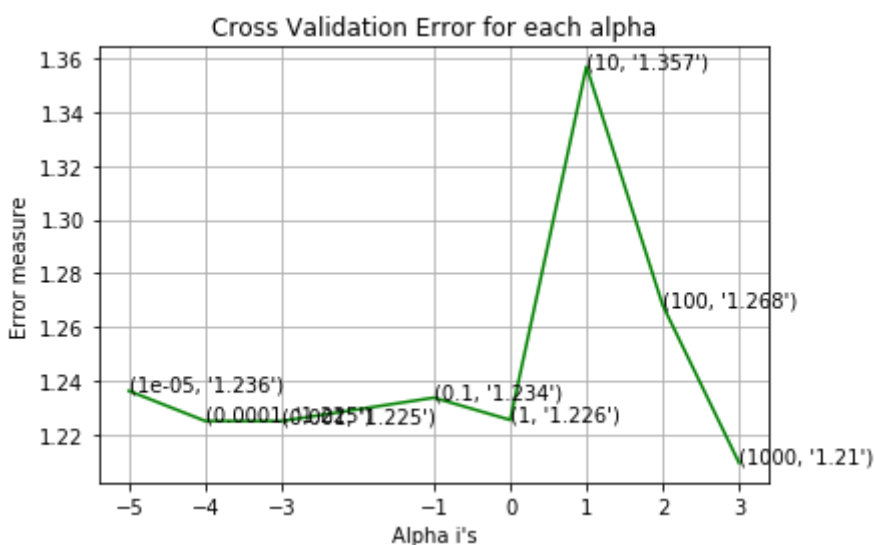
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha],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

```

```

for alpha = 1e-05
Log Loss : 1.236344554472682
for alpha = 0.0001
Log Loss : 1.225157993446364
for alpha = 0.001
Log Loss : 1.2251081676159707
for alpha = 0.1
Log Loss : 1.233810870030191
for alpha = 1
Log Loss : 1.2255385207952596
for alpha = 10
Log Loss : 1.356905775151062
for alpha = 100
Log Loss : 1.2681763681714764
for alpha = 1000
Log Loss : 1.209664478025655

```



```

For values of best alpha = 1000 The train log loss is: 0.9193884167535696
For values of best alpha = 1000 The cross validation log loss is: 1.20966
4478025655
For values of best alpha = 1000 The test log loss is: 1.1893816938761783

```

4.1.1.2. Testing the model with best hyper paramters

In [68]:

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

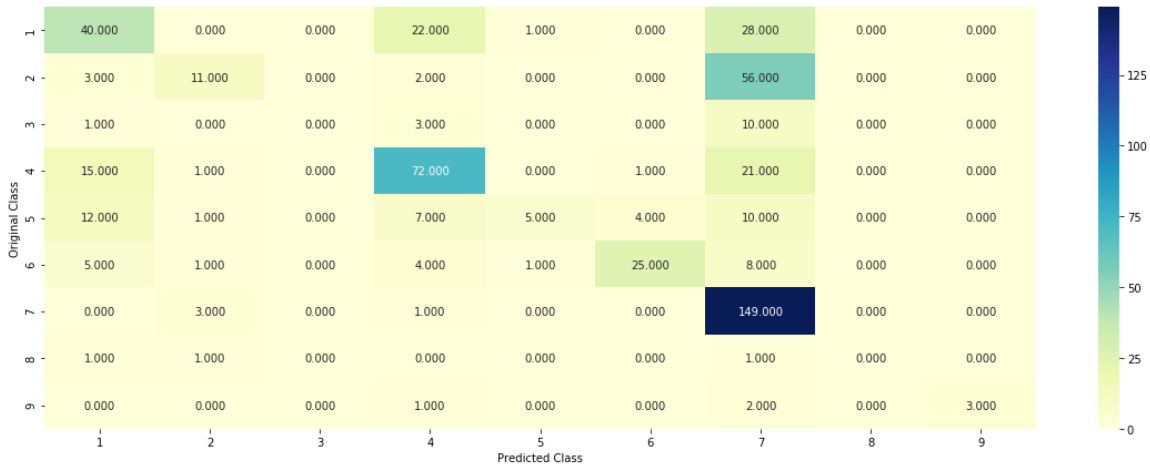
# some of methods of MultinomialNB()
# fit(X, y[, sample_weight])    Fit Naive Bayes classifier according to X, y
# predict(X)    Perform classification on an array of test vectors X.
# predict_log_proba(X)    Return log-probability estimates for the test vector X.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naiv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

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

clf = MultinomialNB(alpha=alpha[best_alpha])
clf.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 probabilitites we use log-probability estimates
print("Log Loss :", log_loss(cv_y, sig_clf_probs))
print("Number of misclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotCoding) - cv_y)/cv_y.shape[0]))
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

Log Loss : 1.209664478025655
Number of missclassified point : 0.4266917293233083

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



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



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



4.1.1.3. Feature Importance, Incorrectly classified point

In []:

In [69]:

```
test_point_index = 2
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices=np.argsort(-clf.coef_)[predicted_cls-1][:, :no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index].lower(),test_df['Variation'].iloc[test_point_index], no_feature
)
```


Predicted Class : 4

Predicted Class Probabilities: [[3.500e-02 8.630e-02 2.600e-03 6.708e-01
6.670e-02 2.420e-02 1.119e-01
2.200e-03 4.000e-04]]

Actual Class : 6

7 Gene feature [cbl] present in test data point [True]
11 Text feature [proteins] present in test data point [True]
12 Text feature [protein] present in test data point [True]
13 Text feature [activity] present in test data point [True]
16 Text feature [experiments] present in test data point [True]
17 Text feature [acid] present in test data point [True]
18 Text feature [function] present in test data point [True]
19 Text feature [shown] present in test data point [True]
20 Text feature [indicated] present in test data point [True]
22 Text feature [results] present in test data point [True]
23 Text feature [whether] present in test data point [True]
24 Text feature [loss] present in test data point [True]
25 Text feature [two] present in test data point [True]
26 Text feature [determined] present in test data point [True]
27 Text feature [amino] present in test data point [True]
28 Text feature [whereas] present in test data point [True]
29 Text feature [mammalian] present in test data point [True]
30 Text feature [also] present in test data point [True]
31 Text feature [described] present in test data point [True]
32 Text feature [bind] present in test data point [True]
33 Text feature [important] present in test data point [True]
35 Text feature [partially] present in test data point [True]
36 Text feature [containing] present in test data point [True]
37 Text feature [ability] present in test data point [True]
38 Text feature [type] present in test data point [True]
39 Text feature [expressed] present in test data point [True]
40 Text feature [tagged] present in test data point [True]
41 Text feature [indicate] present in test data point [True]
42 Text feature [either] present in test data point [True]
43 Text feature [transfected] present in test data point [True]
44 Text feature [missense] present in test data point [True]
45 Text feature [levels] present in test data point [True]
46 Text feature [functions] present in test data point [True]
48 Text feature [reduced] present in test data point [True]
49 Text feature [wild] present in test data point [True]
50 Text feature [retained] present in test data point [True]
52 Text feature [thus] present in test data point [True]
53 Text feature [mutations] present in test data point [True]
55 Text feature [related] present in test data point [True]
58 Text feature [expression] present in test data point [True]
59 Text feature [standard] present in test data point [True]
60 Text feature [analyzed] present in test data point [True]
62 Text feature [catalytic] present in test data point [True]
63 Text feature [determine] present in test data point [True]
64 Text feature [terminal] present in test data point [True]
65 Text feature [incubated] present in test data point [True]
66 Text feature [30] present in test data point [True]
67 Text feature [although] present in test data point [True]
69 Text feature [vivo] present in test data point [True]
70 Text feature [three] present in test data point [True]
71 Text feature [see] present in test data point [True]
72 Text feature [using] present in test data point [True]
74 Text feature [effects] present in test data point [True]
76 Text feature [sds] present in test data point [True]
84 Text feature [residues] present in test data point [True]

85 Text feature [discussion] present in test data point [True]
86 Text feature [vector] present in test data point [True]
87 Text feature [previously] present in test data point [True]
88 Text feature [contribute] present in test data point [True]
90 Text feature [show] present in test data point [True]
92 Text feature [cells] present in test data point [True]
95 Text feature [may] present in test data point [True]
96 Text feature [suppressor] present in test data point [True]
97 Text feature [possible] present in test data point [True]
98 Text feature [associated] present in test data point [True]
99 Text feature [involved] present in test data point [True]
Out of the top 100 features 66 are present in query point

4.1.1.4. Feature Importance, Correctly classified point

In [70]:

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

Predicted Class : 7

Predicted Class Probabilities: [[8.00e-04 2.91e-01 0.00e+00 4.00e-04 3.00e-03 1.40e-03 7.03e-01 3.00e-04 0.00e+00]]

Actual Class : 7

13 Text feature [cells] present in test data point [True]
14 Text feature [activated] present in test data point [True]
15 Text feature [kinase] present in test data point [True]
16 Text feature [activation] present in test data point [True]
21 Text feature [downstream] present in test data point [True]
22 Text feature [factor] present in test data point [True]
23 Text feature [cell] present in test data point [True]
24 Text feature [phosphorylation] present in test data point [True]
25 Text feature [contrast] present in test data point [True]
26 Text feature [presence] present in test data point [True]
27 Text feature [expressing] present in test data point [True]
28 Text feature [shown] present in test data point [True]
29 Text feature [also] present in test data point [True]
30 Text feature [inhibitor] present in test data point [True]
31 Text feature [recently] present in test data point [True]
34 Text feature [signaling] present in test data point [True]
35 Text feature [growth] present in test data point [True]
36 Text feature [however] present in test data point [True]
37 Text feature [suggest] present in test data point [True]
38 Text feature [10] present in test data point [True]
39 Text feature [mechanism] present in test data point [True]
40 Text feature [independent] present in test data point [True]
41 Text feature [increased] present in test data point [True]
42 Text feature [addition] present in test data point [True]
43 Text feature [treated] present in test data point [True]
44 Text feature [compared] present in test data point [True]
45 Text feature [previously] present in test data point [True]
46 Text feature [well] present in test data point [True]
47 Text feature [figure] present in test data point [True]
48 Text feature [similar] present in test data point [True]
49 Text feature [found] present in test data point [True]
50 Text feature [1a] present in test data point [True]
51 Text feature [described] present in test data point [True]
52 Text feature [3b] present in test data point [True]
53 Text feature [enhanced] present in test data point [True]
54 Text feature [consistent] present in test data point [True]
55 Text feature [tyrosine] present in test data point [True]
56 Text feature [showed] present in test data point [True]
57 Text feature [mutant] present in test data point [True]
58 Text feature [constitutively] present in test data point [True]
59 Text feature [mutations] present in test data point [True]
62 Text feature [interestingly] present in test data point [True]
63 Text feature [potential] present in test data point [True]
64 Text feature [demonstrated] present in test data point [True]
65 Text feature [higher] present in test data point [True]
66 Text feature [constitutive] present in test data point [True]
67 Text feature [may] present in test data point [True]
68 Text feature [obtained] present in test data point [True]
69 Text feature [using] present in test data point [True]
70 Text feature [antibodies] present in test data point [True]
71 Text feature [followed] present in test data point [True]
72 Text feature [sensitive] present in test data point [True]
73 Text feature [treatment] present in test data point [True]
74 Text feature [activating] present in test data point [True]
75 Text feature [including] present in test data point [True]

76 Text feature [inhibition] present in test data point [True]
77 Text feature [increase] present in test data point [True]
78 Text feature [approximately] present in test data point [True]
79 Text feature [observed] present in test data point [True]
80 Text feature [without] present in test data point [True]
81 Text feature [antibody] present in test data point [True]
82 Text feature [respectively] present in test data point [True]
83 Text feature [3a] present in test data point [True]
84 Text feature [furthermore] present in test data point [True]
85 Text feature [inhibitors] present in test data point [True]
86 Text feature [activate] present in test data point [True]
87 Text feature [fig] present in test data point [True]
88 Text feature [mutation] present in test data point [True]
89 Text feature [either] present in test data point [True]
90 Text feature [various] present in test data point [True]
91 Text feature [proliferation] present in test data point [True]
92 Text feature [serum] present in test data point [True]
93 Text feature [reported] present in test data point [True]
94 Text feature [suggesting] present in test data point [True]
95 Text feature [total] present in test data point [True]
96 Text feature [performed] present in test data point [True]
97 Text feature [1b] present in test data point [True]
98 Text feature [confirmed] present in test data point [True]
99 Text feature [domain] present in test data point [True]
Out of the top 100 features 79 are present in query point

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [71]:

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

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

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

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

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

```

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

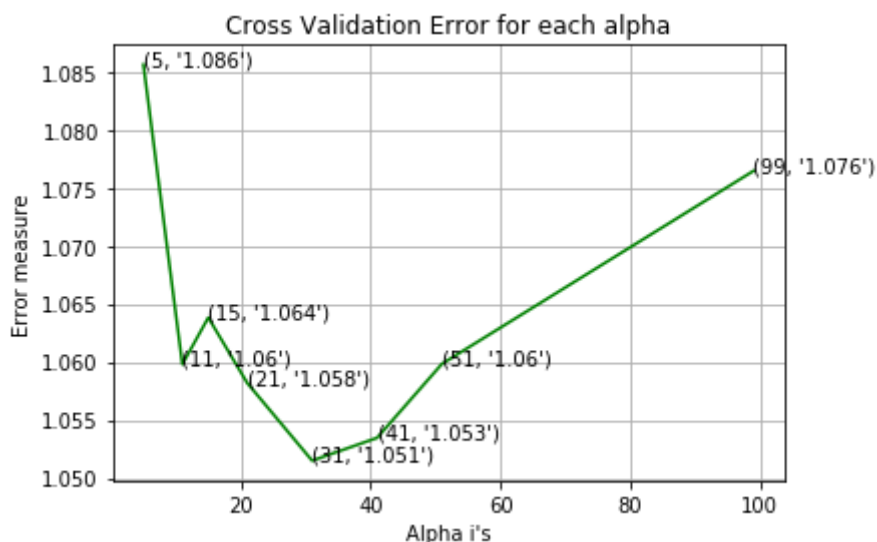
predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha],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

```

```

for alpha = 5
Log Loss : 1.085612225419843
for alpha = 11
Log Loss : 1.0597941549527259
for alpha = 15
Log Loss : 1.0638183904270289
for alpha = 21
Log Loss : 1.0581788530979566
for alpha = 31
Log Loss : 1.0514906056459514
for alpha = 41
Log Loss : 1.0534523966783724
for alpha = 51
Log Loss : 1.059827471915034
for alpha = 99
Log Loss : 1.076453553905757

```



```

For values of best alpha = 31 The train log loss is: 0.801026639748292
For values of best alpha = 31 The cross validation log loss is: 1.0514906
056459514
For values of best alpha = 31 The test log loss is: 1.0758886066927404

```

4.2.2. Testing the model with best hyper paramters

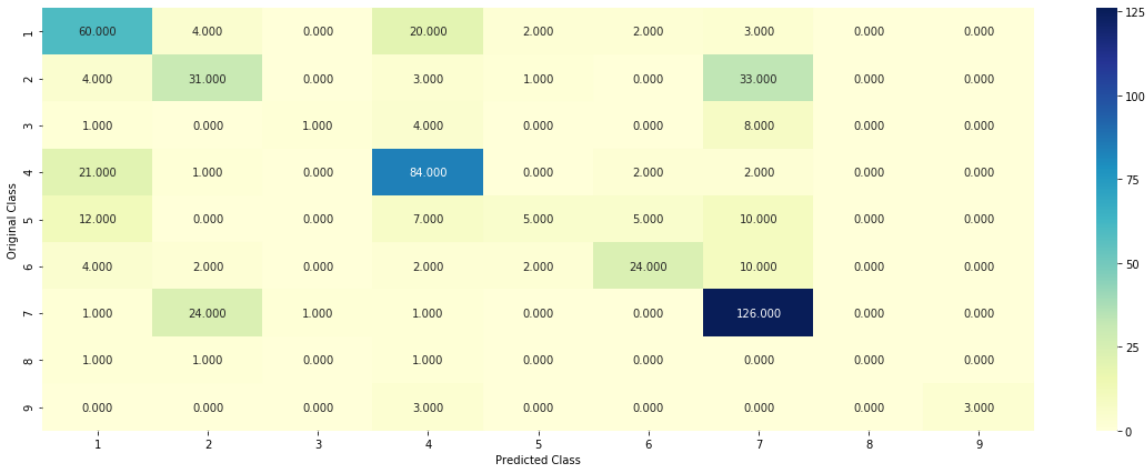
In [72]:

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

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


Log loss : 1.0514906056459514
Number of mis-classified points : 0.37218045112781956

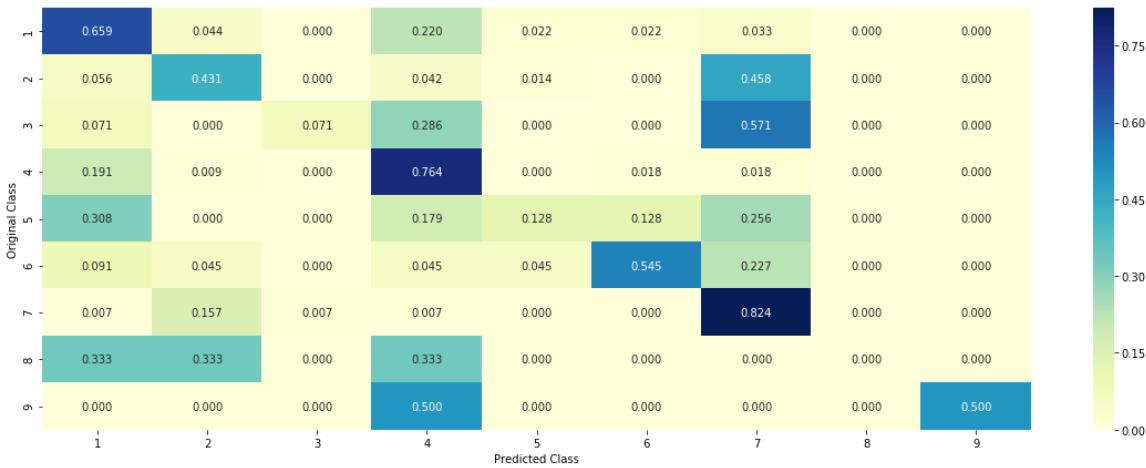
----- Confusion matrix -----



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



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



4.2.3.Sample Query point -1

In [73]:

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

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

Predicted Class : 1

Actual Class : 6

The 31 nearest neighbours of the test points belongs to classes [6 5 1 4

4 4 4 1 5 1 1 6 1 1 1 6 4 1 1 6 1 1 1 5 5 4 5 4 1 4 4]

Frequency of nearest points : Counter({1: 13, 4: 9, 5: 5, 6: 4})

4.2.4. Sample Query Point-2

In [74]:

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

test_point_index = 100

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

Predicted Class : 7

Actual Class : 7

the k value for knn is 31 and the nearest neighbours of the test points belongs to classes [7 7 7 7 7 7 7 7 2 2 7 2 7 7 2 2 2 2 1 2 7 7 7 2 7 2 7

2 7 7]

Frequency of nearest points : Counter({7: 18, 2: 12, 1: 1})

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [75]:

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

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

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

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

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

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

```
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

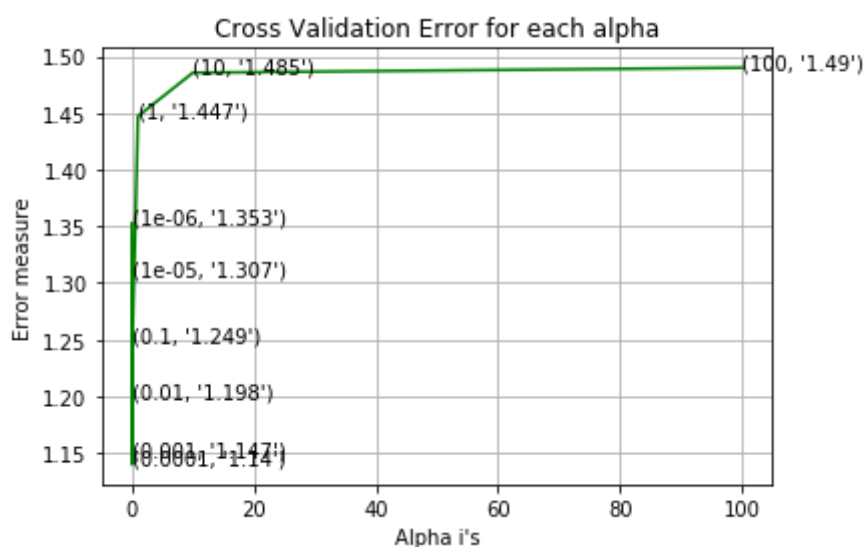
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```

for alpha = 1e-06
Log Loss : 1.3525662705252717
for alpha = 1e-05
Log Loss : 1.3068798893160976
for alpha = 0.0001
Log Loss : 1.139834987934025
for alpha = 0.001
Log Loss : 1.1467974871058073
for alpha = 0.01
Log Loss : 1.198342730814999
for alpha = 0.1
Log Loss : 1.2490496162357898
for alpha = 1
Log Loss : 1.4470774014161265
for alpha = 10
Log Loss : 1.485432860807836
for alpha = 100
Log Loss : 1.4898568355934372

```



For values of best alpha = 0.0001 The train log loss is: 0.4638859489249474

For values of best alpha = 0.0001 The cross validation log loss is: 1.139834987934025

For values of best alpha = 0.0001 The test log loss is: 1.0419421617494864

4.3.1.2. Testing the model with best hyper paramters

In []:

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

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

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

4.3.1.3. Feature Importance

In [77]:

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

4.3.1.3.1. Correctly Classified point

In [78]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]), 4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:, :no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index], test_df['Gene'].iloc[test_point_index].lower(), test_df['Variation'].iloc[test_point_index], no_feature)
```

Predicted Class : 6

Predicted Class Probabilities: [[0.1382 0.0261 0.0039 0.1212 0.071 0.6205
0.0094 0.0066 0.0031]]

Actual Class : 6

```
-----
56 Gene feature [brca1] present in test data point [True]
198 Text feature [transfected] present in test data point [True]
258 Text feature [cells] present in test data point [True]
319 Text feature [suggest] present in test data point [True]
371 Text feature [consistent] present in test data point [True]
382 Text feature [domain] present in test data point [True]
388 Text feature [transfection] present in test data point [True]
395 Text feature [figure] present in test data point [True]
416 Text feature [previously] present in test data point [True]
456 Text feature [expression] present in test data point [True]
469 Text feature [901] present in test data point [True]
471 Text feature [results] present in test data point [True]
473 Text feature [western] present in test data point [True]
475 Text feature [either] present in test data point [True]
Out of the top 500 features 14 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [79]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

Predicted Class : 7

Predicted Class Probabilities: [[1.000e-03 7.770e-02 1.800e-03 1.200e-03
1.400e-03 6.000e-04 9.039e-01
1.030e-02 2.000e-03]]

Actual Class : 7

```
-----
292 Text feature [activated] present in test data point [True]
336 Text feature [constitutive] present in test data point [True]
341 Text feature [3t3] present in test data point [True]
365 Text feature [technology] present in test data point [True]
393 Text feature [mitogen] present in test data point [True]
399 Text feature [activation] present in test data point [True]
415 Text feature [oncogene] present in test data point [True]
430 Text feature [cmml1] present in test data point [True]
452 Text feature [compound] present in test data point [True]
472 Text feature [tk] present in test data point [True]
Out of the top 500 features 10 are present in query point
```

4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

In [80]:

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

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

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

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

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

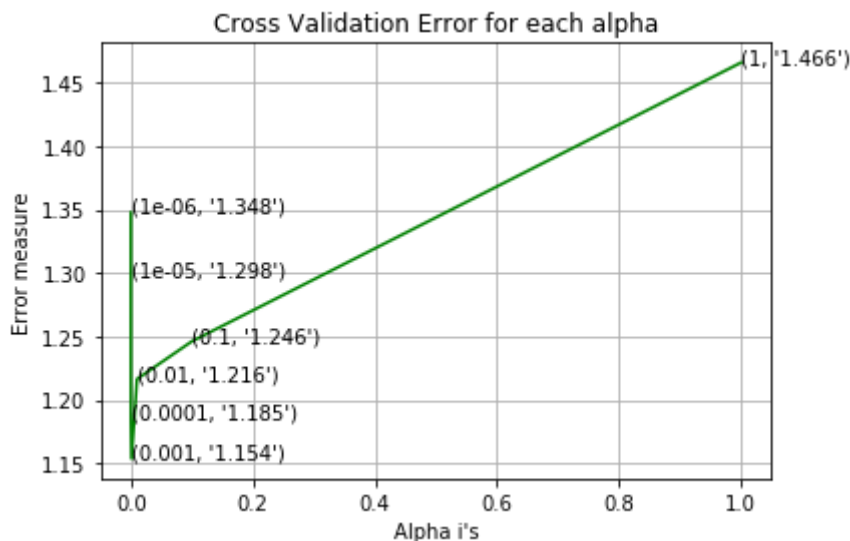
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")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

```
plt.show()

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

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

```
for alpha = 1e-06
Log Loss : 1.347570327562142
for alpha = 1e-05
Log Loss : 1.298297981962169
for alpha = 0.0001
Log Loss : 1.1852909912110532
for alpha = 0.001
Log Loss : 1.1536937507887735
for alpha = 0.01
Log Loss : 1.2163478367981022
for alpha = 0.1
Log Loss : 1.2460913564483314
for alpha = 1
Log Loss : 1.4656323948020837
```



```
For values of best alpha = 0.001 The train log loss is: 0.471796555851063
56
For values of best alpha = 0.001 The cross validation log loss is: 1.1536
937507887735
For values of best alpha = 0.001 The test log loss is: 1.0421884848829193
```

4.3.2.2. Testing model with best hyper parameters

In [81]:

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

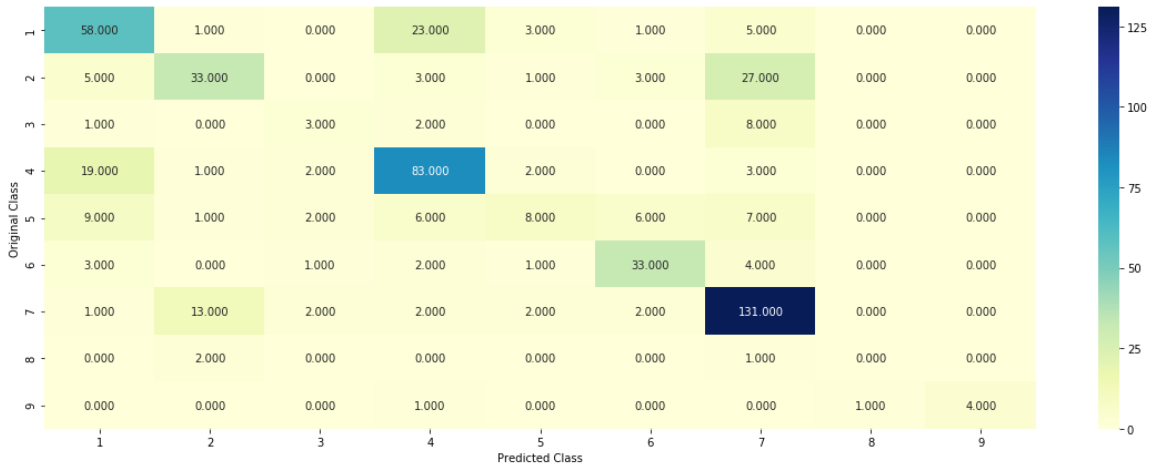
# some of methods
# fit(X, y[, coef_init, intercept_init, ...])    Fit linear model with Stochastic Gradi
ent Descent.
# predict(X)    Predict class labels for samples in X.

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

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

Log loss : 1.1536937507887735
Number of mis-classified points : 0.33646616541353386

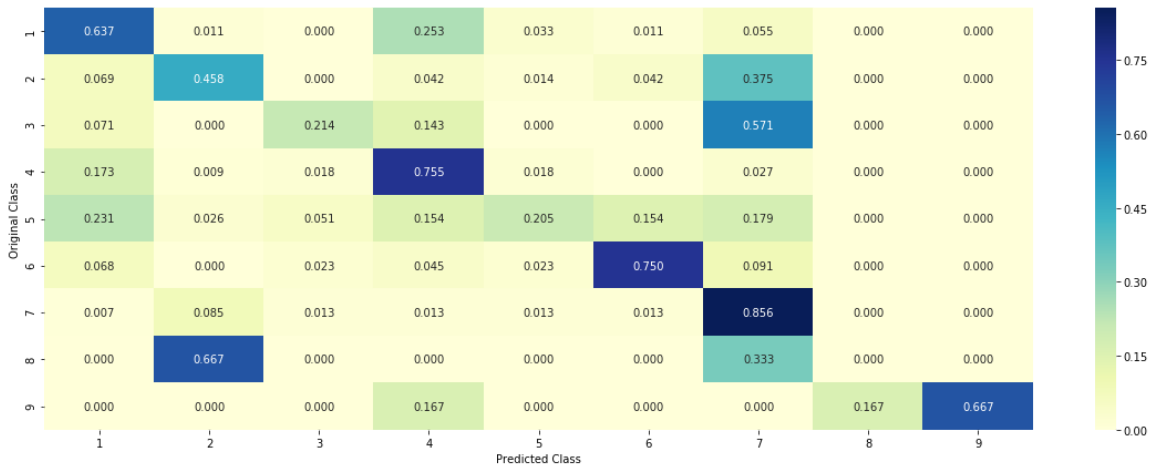
----- Confusion matrix -----



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



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



4.3.2.3. Feature Importance, Correctly Classified point

In [82]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

Predicted Class : 6

Predicted Class Probabilities: [[1.307e-01 5.900e-03 1.400e-03 1.154e-01
5.070e-02 6.885e-01 5.000e-03
2.300e-03 1.000e-04]]

Actual Class : 6

71 Text feature [transfected] present in test data point [True]
81 Text feature [cells] present in test data point [True]
113 Text feature [consistent] present in test data point [True]
115 Text feature [figure] present in test data point [True]
116 Text feature [suggest] present in test data point [True]
129 Text feature [transfection] present in test data point [True]
130 Text feature [previously] present in test data point [True]
135 Text feature [described] present in test data point [True]
140 Text feature [expression] present in test data point [True]
152 Text feature [western] present in test data point [True]
166 Text feature [results] present in test data point [True]
176 Text feature [cell] present in test data point [True]
182 Text feature [domain] present in test data point [True]
185 Text feature [presence] present in test data point [True]
186 Text feature [containing] present in test data point [True]
189 Text feature [indicate] present in test data point [True]
191 Text feature [either] present in test data point [True]
192 Text feature [expressed] present in test data point [True]
209 Text feature [expressing] present in test data point [True]
219 Text feature [detected] present in test data point [True]
237 Text feature [performed] present in test data point [True]
239 Text feature [whereas] present in test data point [True]
242 Text feature [level] present in test data point [True]
245 Text feature [similar] present in test data point [True]
254 Text feature [represent] present in test data point [True]
262 Text feature [complex] present in test data point [True]
265 Text feature [shown] present in test data point [True]
266 Text feature [indicating] present in test data point [True]
276 Text feature [contribute] present in test data point [True]
277 Text feature [proteins] present in test data point [True]
281 Text feature [confirmed] present in test data point [True]
286 Text feature [mutations] present in test data point [True]
299 Text feature [found] present in test data point [True]
311 Text feature [indicated] present in test data point [True]
331 Text feature [despite] present in test data point [True]
336 Text feature [whether] present in test data point [True]
338 Text feature [control] present in test data point [True]
360 Text feature [possible] present in test data point [True]
361 Text feature [effects] present in test data point [True]
362 Text feature [901] present in test data point [True]
365 Text feature [experiments] present in test data point [True]
367 Text feature [ambiguous] present in test data point [True]
370 Text feature [furthermore] present in test data point [True]
377 Text feature [present] present in test data point [True]
378 Text feature [contrast] present in test data point [True]
390 Text feature [dependent] present in test data point [True]
393 Text feature [increased] present in test data point [True]
394 Text feature [directly] present in test data point [True]
400 Text feature [another] present in test data point [True]
402 Text feature [compared] present in test data point [True]
405 Text feature [increase] present in test data point [True]
406 Text feature [could] present in test data point [True]
455 Text feature [deletion] present in test data point [True]
458 Text feature [point] present in test data point [True]
460 Text feature [growth] present in test data point [True]

462 Text feature [role] present in test data point [True]
467 Text feature [negative] present in test data point [True]
468 Text feature [directed] present in test data point [True]
470 Text feature [capable] present in test data point [True]
472 Text feature [absence] present in test data point [True]
493 Text feature [common] present in test data point [True]
Out of the top 500 features 61 are present in query point

4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [83]:

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

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

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

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

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

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

```
plt.show()

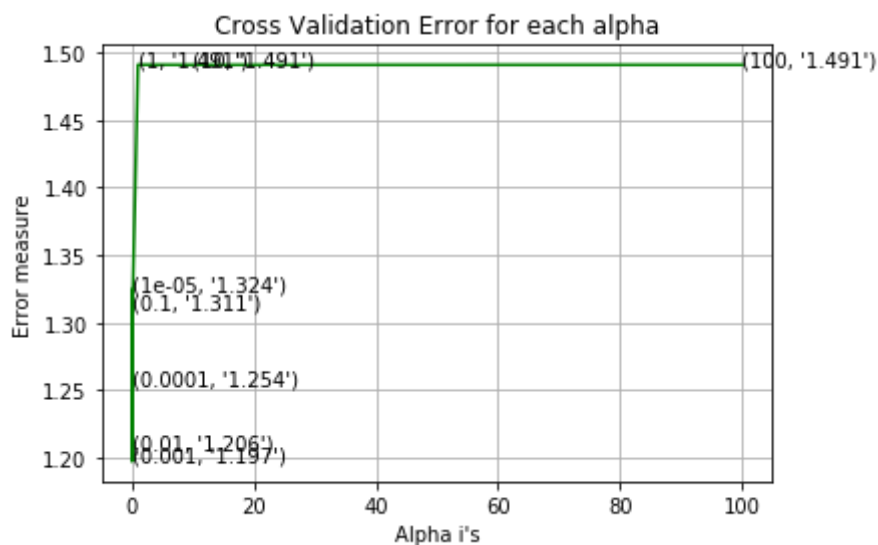
best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i, kernel='linear', probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```

for C = 1e-05
Log Loss : 1.3242261882328288
for C = 0.0001
Log Loss : 1.2540990706696649
for C = 0.001
Log Loss : 1.1972718642013265
for C = 0.01
Log Loss : 1.206432980264267
for C = 0.1
Log Loss : 1.3111788442400782
for C = 1
Log Loss : 1.4907413773338065
for C = 10
Log Loss : 1.490641140173027
for C = 100
Log Loss : 1.4906634432391028

```



For values of best alpha = 0.001 The train log loss is: 0.517443762821345
 1
 For values of best alpha = 0.001 The cross validation log loss is: 1.1972718642013265
 For values of best alpha = 0.001 The test log loss is: 1.0858159042749824

4.4.2. Testing model with best hyper parameters

In [84]:

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

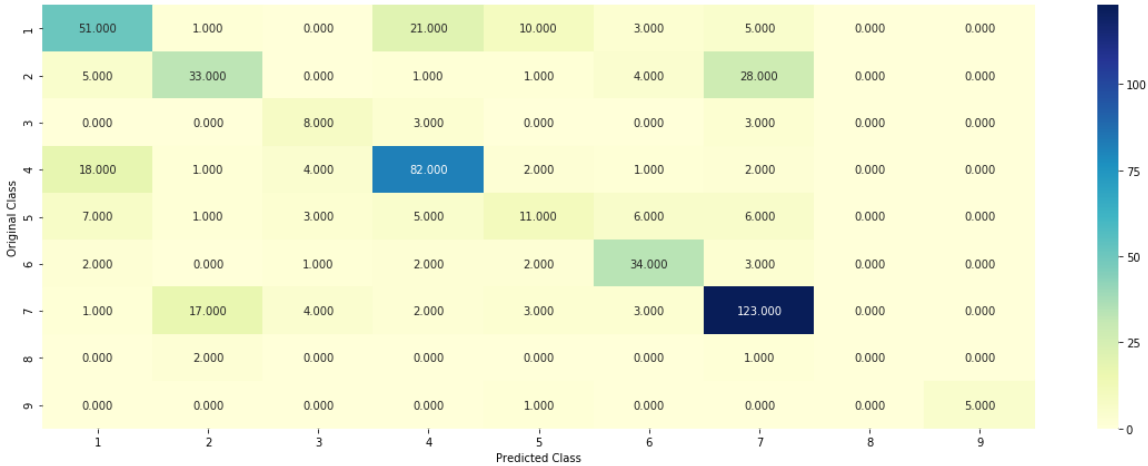
# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

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

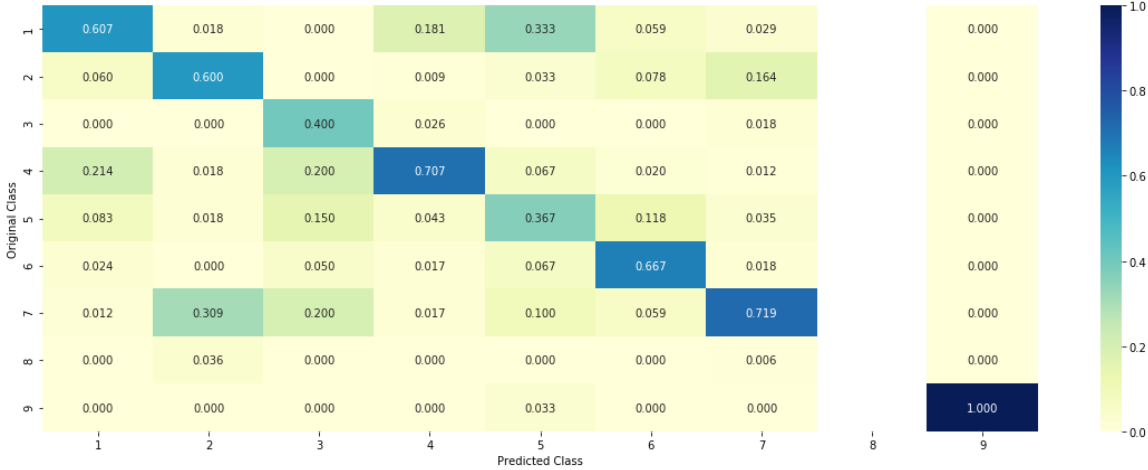
# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42, class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y,
clf)
```

Log loss : 1.1972718642013265
Number of mis-classified points : 0.34774436090225563

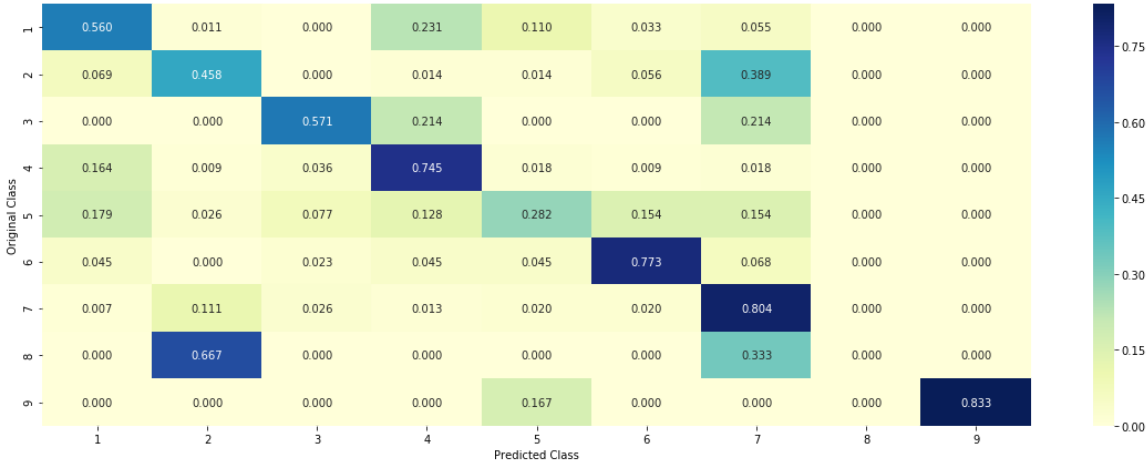
----- Confusion matrix -----



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



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



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', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]), 4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:, :no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index], test_df['Gene'].iloc[test_point_index], test_df['Variation'].iloc[test_point_index], no_feature)
```

Predicted Class : 6

Predicted Class Probabilities: [[0.0557 0.06 0.003 0.0665 0.055 0.6915
0.0569 0.007 0.0044]]

Actual Class: 6

205 Text feature [unresolved] present in test data point [True]
263 Text feature [meaningful] present in test data point [True]
458 Text feature [aspartate] present in test data point [True]
497 Text feature [transfected] present in test data point [True]
Out of the top 500 features 4 are present in query point

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [86]:

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

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

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

# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-construction-2/
# -----

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

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

'''fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[: ,None], np.array(max_depth)[None]).ravel()
```

```

ax.plot(features, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)), (features[i],cv_log_err
or_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', m
ax_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log loss
is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validatio
n log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss i
s:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

```

```

for n_estimators = 100 and max depth = 5
Log Loss : 1.2505460700488158
for n_estimators = 100 and max depth = 10
Log Loss : 1.181386343996467
for n_estimators = 200 and max depth = 5
Log Loss : 1.2356399743811328
for n_estimators = 200 and max depth = 10
Log Loss : 1.171320117702292
for n_estimators = 500 and max depth = 5
Log Loss : 1.230503750654064
for n_estimators = 500 and max depth = 10
Log Loss : 1.1699290510168248
for n_estimators = 1000 and max depth = 5
Log Loss : 1.2315844928535562
for n_estimators = 1000 and max depth = 10
Log Loss : 1.1685383727303011
for n_estimators = 2000 and max depth = 5
Log Loss : 1.232149869261221
for n_estimators = 2000 and max depth = 10
Log Loss : 1.1666788323782282
For values of best estimator = 2000 The train log loss is: 0.635501129027
8859
For values of best estimator = 2000 The cross validation log loss is: 1.1
666788323782282
For values of best estimator = 2000 The test log loss is: 1.1526855236761
597

```

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

In [87]:

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

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

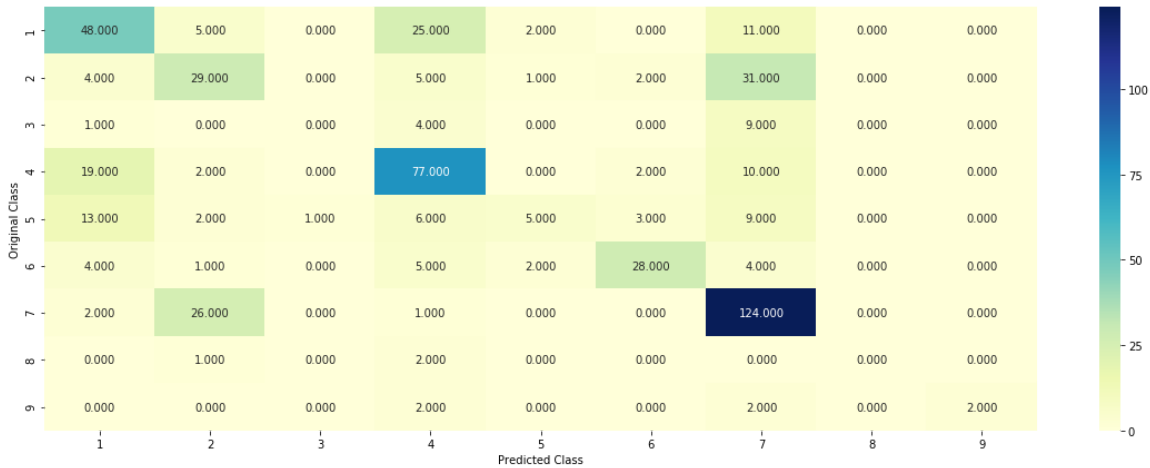
# some of attributes of RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-construction-2/
# -----

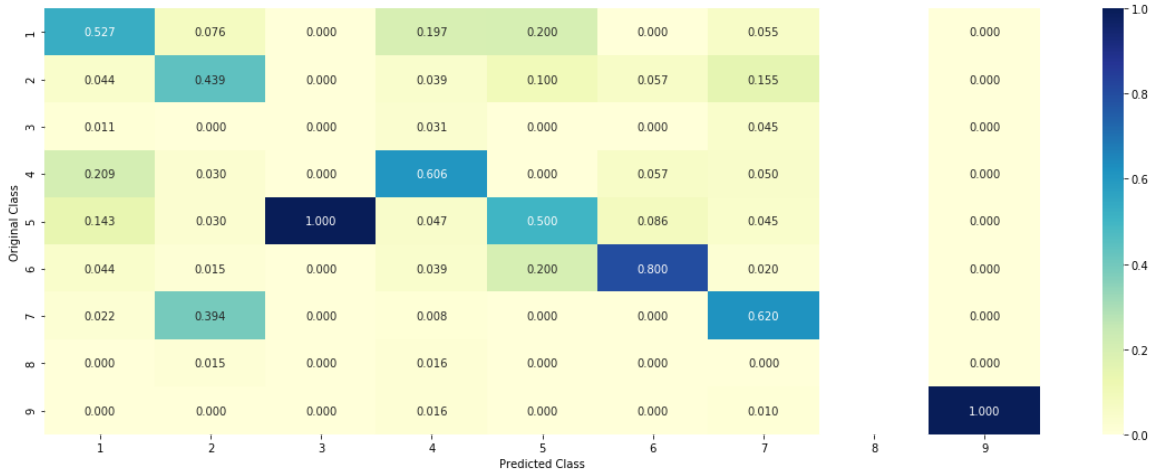
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y,
clf)
```

Log loss : 1.1666788323782282
Number of mis-classified points : 0.4116541353383459

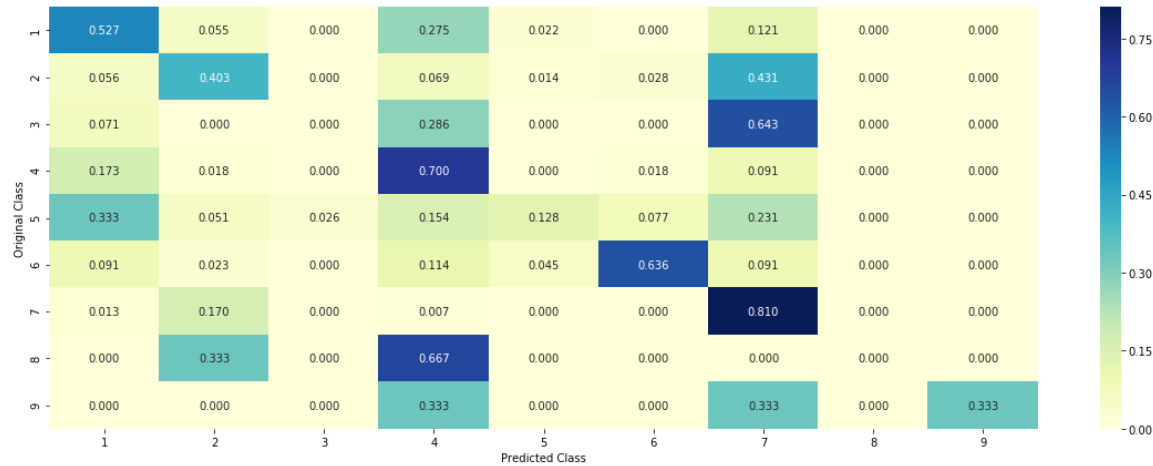
----- Confusion matrix -----



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



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [88]:

```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

Predicted Class : 1

Predicted Class Probabilities: [[0.2893 0.0434 0.0206 0.22 0.0942 0.279 0.0404 0.0062 0.0068]]

Actual Class : 6

```
-----
4 Text feature [activation] present in test data point [True]
7 Text feature [treatment] present in test data point [True]
11 Text feature [suppressor] present in test data point [True]
15 Text feature [function] present in test data point [True]
17 Text feature [missense] present in test data point [True]
20 Text feature [inhibition] present in test data point [True]
22 Text feature [therapy] present in test data point [True]
27 Text feature [yeast] present in test data point [True]
33 Text feature [nonsense] present in test data point [True]
43 Text feature [growth] present in test data point [True]
44 Text feature [stability] present in test data point [True]
47 Text feature [variants] present in test data point [True]
48 Text feature [cell] present in test data point [True]
49 Text feature [brca1] present in test data point [True]
50 Text feature [cells] present in test data point [True]
52 Text feature [clinical] present in test data point [True]
53 Text feature [deleterious] present in test data point [True]
56 Text feature [protein] present in test data point [True]
57 Text feature [expressing] present in test data point [True]
61 Text feature [predicted] present in test data point [True]
67 Text feature [proteins] present in test data point [True]
68 Text feature [respond] present in test data point [True]
74 Text feature [frameshift] present in test data point [True]
75 Text feature [repair] present in test data point [True]
77 Text feature [potential] present in test data point [True]
78 Text feature [functional] present in test data point [True]
84 Text feature [variant] present in test data point [True]
86 Text feature [sensitivity] present in test data point [True]
97 Text feature [splice] present in test data point [True]
Out of the top 100 features 29 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [89]:

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

Predicted Class : 7

Predicted Class Probabilities: [[0.0463 0.2833 0.0125 0.0265 0.0338 0.0325
0.5566 0.0042 0.0043]]

Actual Class : 7

0 Text feature [kinase] present in test data point [True]
1 Text feature [inhibitors] present in test data point [True]
2 Text feature [activated] present in test data point [True]
3 Text feature [activating] present in test data point [True]
4 Text feature [activation] present in test data point [True]
5 Text feature [phosphorylation] present in test data point [True]
6 Text feature [tyrosine] present in test data point [True]
7 Text feature [treatment] present in test data point [True]
9 Text feature [oncogenic] present in test data point [True]
10 Text feature [inhibitor] present in test data point [True]
11 Text feature [suppressor] present in test data point [True]
12 Text feature [trials] present in test data point [True]
13 Text feature [constitutive] present in test data point [True]
14 Text feature [drug] present in test data point [True]
15 Text feature [function] present in test data point [True]
19 Text feature [downstream] present in test data point [True]
20 Text feature [inhibition] present in test data point [True]
21 Text feature [inhibited] present in test data point [True]
22 Text feature [therapy] present in test data point [True]
23 Text feature [loss] present in test data point [True]
24 Text feature [signaling] present in test data point [True]
25 Text feature [months] present in test data point [True]
26 Text feature [receptor] present in test data point [True]
28 Text feature [activate] present in test data point [True]
30 Text feature [constitutively] present in test data point [True]
31 Text feature [patients] present in test data point [True]
32 Text feature [mitogen] present in test data point [True]
33 Text feature [nonsense] present in test data point [True]
35 Text feature [extracellular] present in test data point [True]
36 Text feature [resistance] present in test data point [True]
37 Text feature [therapeutic] present in test data point [True]
38 Text feature [kinases] present in test data point [True]
39 Text feature [defective] present in test data point [True]
40 Text feature [akt] present in test data point [True]
41 Text feature [treated] present in test data point [True]
42 Text feature [proliferation] present in test data point [True]
43 Text feature [growth] present in test data point [True]
44 Text feature [stability] present in test data point [True]
45 Text feature [phospho] present in test data point [True]
47 Text feature [variants] present in test data point [True]
48 Text feature [cell] present in test data point [True]
50 Text feature [cells] present in test data point [True]
51 Text feature [efficacy] present in test data point [True]
52 Text feature [clinical] present in test data point [True]
56 Text feature [protein] present in test data point [True]
57 Text feature [expressing] present in test data point [True]
58 Text feature [response] present in test data point [True]
59 Text feature [amplification] present in test data point [True]
60 Text feature [phosphorylated] present in test data point [True]
61 Text feature [predicted] present in test data point [True]
62 Text feature [survival] present in test data point [True]
64 Text feature [dose] present in test data point [True]
65 Text feature [factor] present in test data point [True]
66 Text feature [mapk] present in test data point [True]
67 Text feature [proteins] present in test data point [True]
68 Text feature [respond] present in test data point [True]

69 Text feature [ras] present in test data point [True]
71 Text feature [nsc1c] present in test data point [True]
72 Text feature [serum] present in test data point [True]
73 Text feature [oncogene] present in test data point [True]
77 Text feature [potential] present in test data point [True]
78 Text feature [functional] present in test data point [True]
80 Text feature [patient] present in test data point [True]
81 Text feature [functions] present in test data point [True]
83 Text feature [responses] present in test data point [True]
84 Text feature [variant] present in test data point [True]
85 Text feature [3t3] present in test data point [True]
86 Text feature [sensitivity] present in test data point [True]
88 Text feature [sensitive] present in test data point [True]
89 Text feature [phosphatase] present in test data point [True]
90 Text feature [erk1] present in test data point [True]
92 Text feature [ligand] present in test data point [True]
93 Text feature [kit] present in test data point [True]
94 Text feature [days] present in test data point [True]
96 Text feature [stimulation] present in test data point [True]
99 Text feature [egfr] present in test data point [True]
Out of the top 100 features 76 are present in query point

4.5.3. Hyper paramter tuning (With Response Coding)

In [90]:

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

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

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

# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-construction-2/
# -----

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

alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
    ...
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[: ,None], np.array(max_depth)[None]).ravel()
```

```

ax.plot(features, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (features[i],cv_log_err
or_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', m
ax_depth=max_depth[int(best_alpha%4)], random_state=42, n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log loss is:"
,log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross validation lo
g loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log loss is:",
log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

```

```
for n_estimators = 10 and max depth = 2
Log Loss : 2.058722605877042
for n_estimators = 10 and max depth = 3
Log Loss : 1.5959309303460119
for n_estimators = 10 and max depth = 5
Log Loss : 1.3975525241336613
for n_estimators = 10 and max depth = 10
Log Loss : 1.7421135886587296
for n_estimators = 50 and max depth = 2
Log Loss : 1.706639389195456
for n_estimators = 50 and max depth = 3
Log Loss : 1.39424435778715
for n_estimators = 50 and max depth = 5
Log Loss : 1.2436956415797655
for n_estimators = 50 and max depth = 10
Log Loss : 1.6552636642731
for n_estimators = 100 and max depth = 2
Log Loss : 1.4953056633550477
for n_estimators = 100 and max depth = 3
Log Loss : 1.4034735983968847
for n_estimators = 100 and max depth = 5
Log Loss : 1.2071518802004757
for n_estimators = 100 and max depth = 10
Log Loss : 1.6557939192227453
for n_estimators = 200 and max depth = 2
Log Loss : 1.535368047281117
for n_estimators = 200 and max depth = 3
Log Loss : 1.3724126970946164
for n_estimators = 200 and max depth = 5
Log Loss : 1.2867732620714953
for n_estimators = 200 and max depth = 10
Log Loss : 1.6664901392491769
for n_estimators = 500 and max depth = 2
Log Loss : 1.5689796411897152
for n_estimators = 500 and max depth = 3
Log Loss : 1.4164512607024449
for n_estimators = 500 and max depth = 5
Log Loss : 1.2854079899711581
for n_estimators = 500 and max depth = 10
Log Loss : 1.6540580873926083
for n_estimators = 1000 and max depth = 2
Log Loss : 1.5663791417533448
for n_estimators = 1000 and max depth = 3
Log Loss : 1.4252234294732848
for n_estimators = 1000 and max depth = 5
Log Loss : 1.2884118652034482
for n_estimators = 1000 and max depth = 10
Log Loss : 1.6602122080648876
For values of best alpha = 100 The train log loss is: 0.06357084522984031
For values of best alpha = 100 The cross validation log loss is: 1.207151
8802004757
For values of best alpha = 100 The test log loss is: 1.2294664006791725
```

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

In [91]:

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

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

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

# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-construction-2/
# -----

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimators=alpha
[int(best_alpha/4)], criterion='gini', max_features='auto', random_state=42)
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCoding,c
v_y, clf)
```

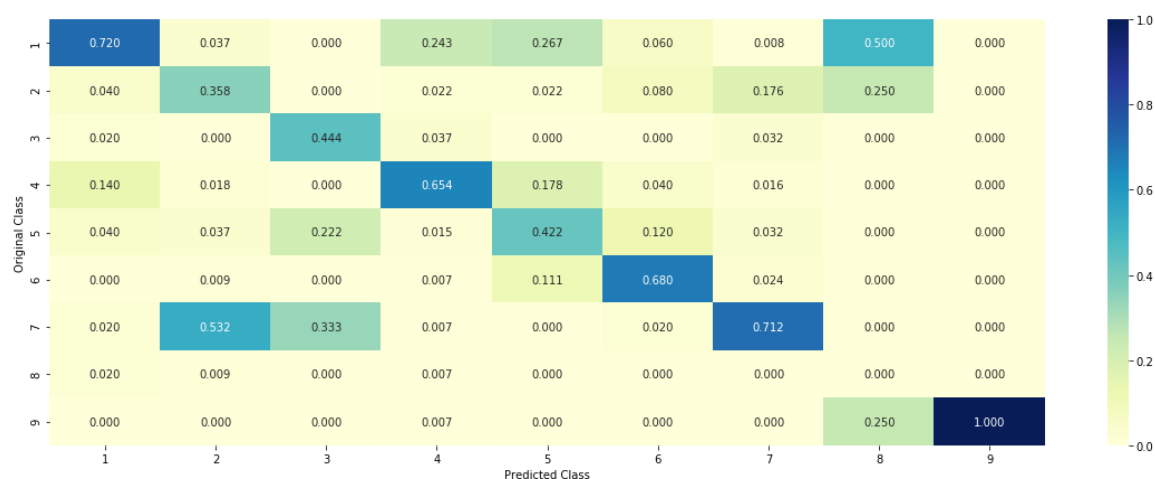
Log loss : 1.2071518802004757

Number of mis-classified points : 0.40977443609022557

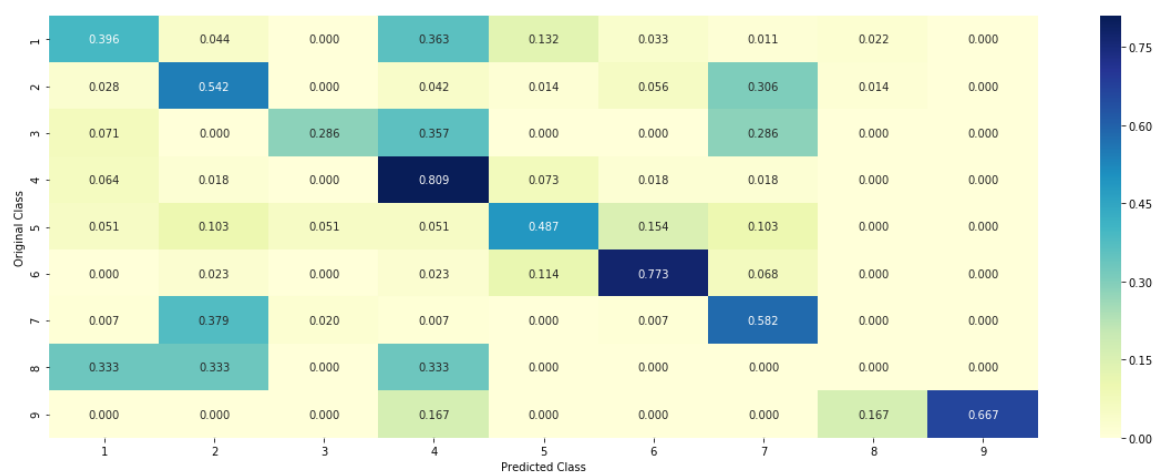
----- Confusion matrix -----



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



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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [92]:

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=max_depth[int(best_alpha%4)], random_state=42, n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 1
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

Predicted Class : 5

Predicted Class Probabilities: [[0.0582 0.0075 0.1273 0.0809 0.4608 0.2408
0.0071 0.0063 0.0111]]

Actual Class : 6

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

4.5.5.2. Incorrectly Classified point

In [93]:

```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

Predicted Class : 2

Predicted Class Probabilities: [[0.0155 0.4288 0.0539 0.0255 0.0315 0.0423
0.367 0.0208 0.0147]]

Actual Class : 7

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

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [94]:

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

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

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

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

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

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

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

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

```

# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-construction-2/
# -----

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

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

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

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

```

```

Logistic Regression : Log Loss: 1.15
Support vector machines : Log Loss: 1.49
Naive Bayes : Log Loss: 1.23

```

```

-----
Stacking Classifier : for the value of alpha: 0.000100 Log Loss: 1.816
Stacking Classifier : for the value of alpha: 0.001000 Log Loss: 1.707
Stacking Classifier : for the value of alpha: 0.010000 Log Loss: 1.278
Stacking Classifier : for the value of alpha: 0.100000 Log Loss: 1.182
Stacking Classifier : for the value of alpha: 1.000000 Log Loss: 1.508
Stacking Classifier : for the value of alpha: 10.000000 Log Loss: 1.925

```

4.7.2 testing the model with the best hyper parameters

In [95]:

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

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

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

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

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

Log loss (train) on the stacking classifier : 0.4672740312988268

Log loss (CV) on the stacking classifier : 1.1818344581885143

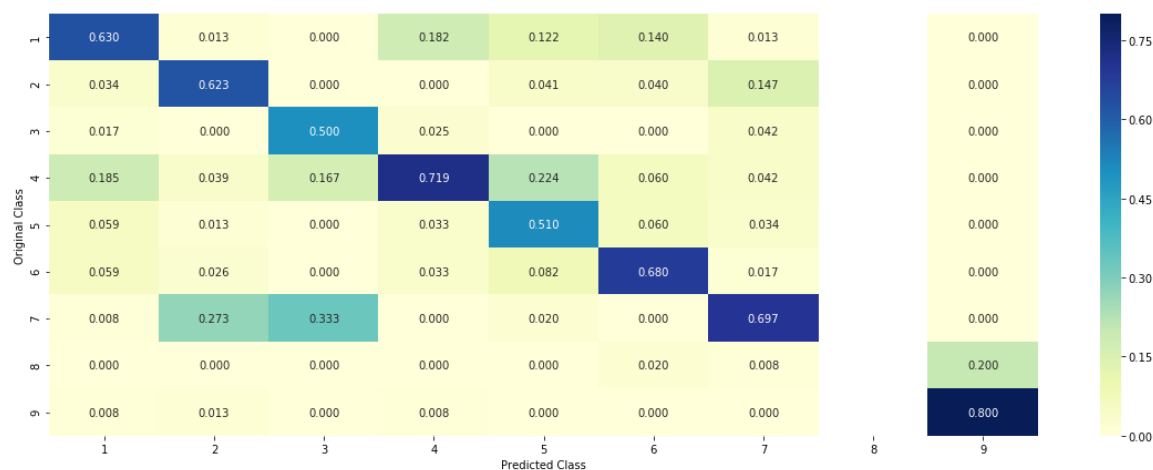
Log loss (test) on the stacking classifier : 1.1196387770049572

Number of missclassified point : 0.33533834586466166

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



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



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



4.7.3 Maximum Voting classifier

In [96]:

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

Log loss (train) on the VotingClassifier : 0.7944079044882274

Log loss (CV) on the VotingClassifier : 1.1214476721452349

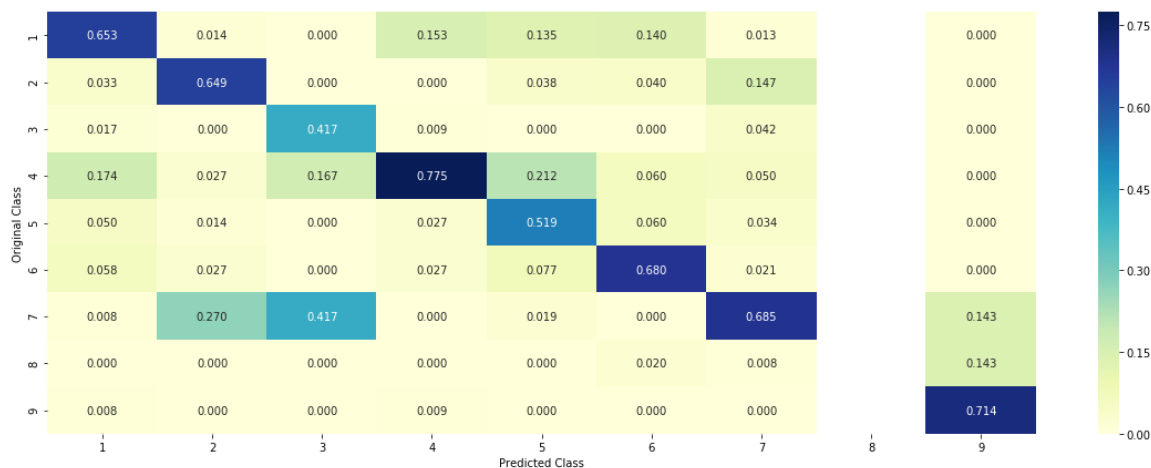
Log loss (test) on the VotingClassifier : 1.1167971687358895

Number of missclassified point : 0.32781954887218046

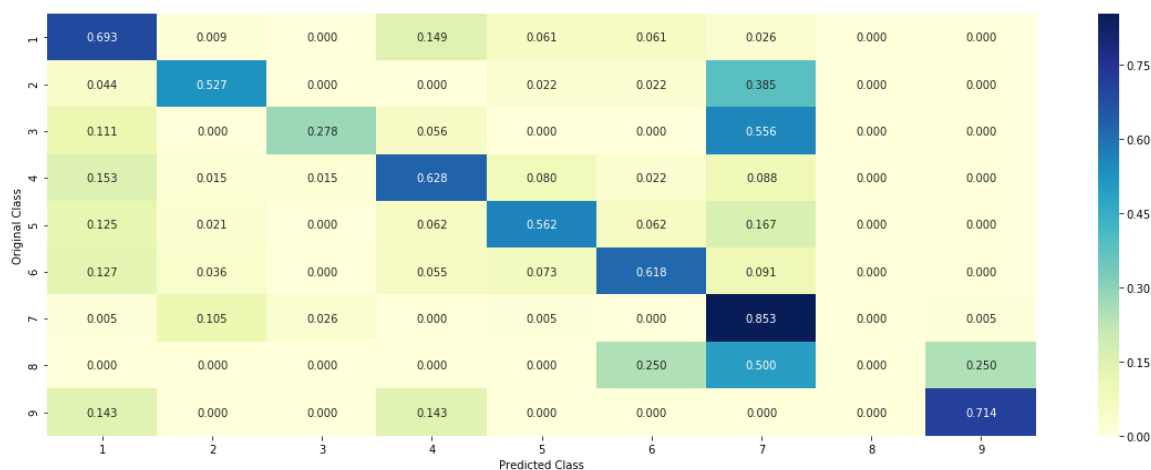
----- Confusion matrix -----



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



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



5. Assignments

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

TASK 2

In [97]:

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

In [100]:

```
train_top_1000 = SelectKBest(chi2,k=1000).fit_transform(train_text_feature_tfidf, train_y)
cv_top_1000 = SelectKBest(chi2,k=1000).fit_transform(cv_text_feature_tfidf, cv_y)
test_top_1000 = SelectKBest(chi2,k=1000).fit_transform(test_text_feature_tfidf, y_test)
```

In [101]:

```
print(train_top_1000.shape)
print(cv_top_1000.shape)
print(test_top_1000.shape)
```

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

In [107]:

```
type(train_top_1000)
```

Out[107]:

```
scipy.sparse.csr.csr_matrix
```

In [111]:

```
train1000 = hstack((train_gene_var_onehotCoding,train_top_1000)).tocsr()
cv1000 = hstack((cv_gene_var_onehotCoding,cv_top_1000)).tocsr()
test1000 = hstack((test_gene_var_onehotCoding,test_top_1000)).tocsr()
```

Random Forest with top 1000 tfidf

In [112]:

```
alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42, n_jobs=-1)
        clf.fit(train1000, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train1000, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv1000)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train1000, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train1000, train_y)

predict_y = sig_clf.predict_proba(train1000)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv1000)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test1000)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```



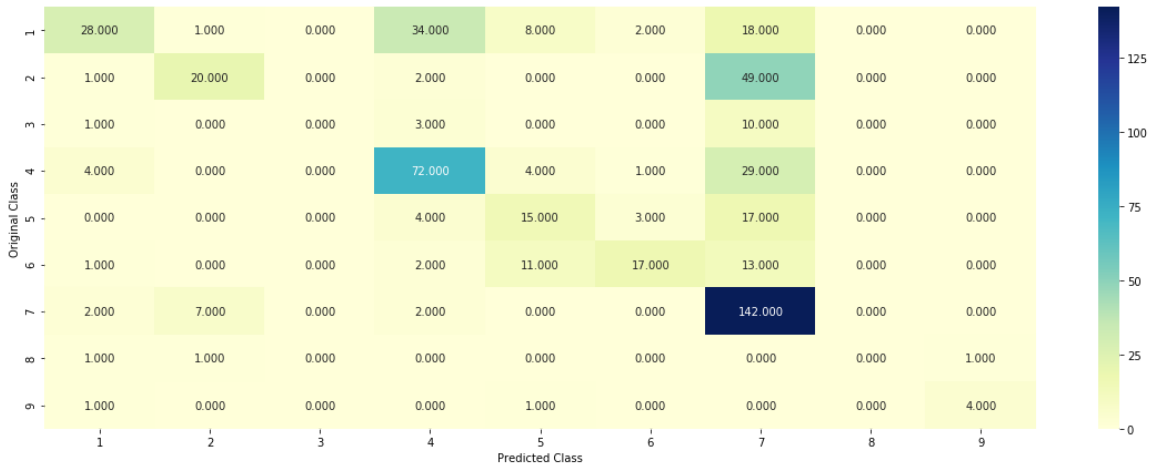
```
for n_estimators = 100 and max depth = 5
Log Loss : 1.3441167636996445
for n_estimators = 100 and max depth = 10
Log Loss : 1.3179878785768506
for n_estimators = 200 and max depth = 5
Log Loss : 1.3347754781039711
for n_estimators = 200 and max depth = 10
Log Loss : 1.3059824557110216
for n_estimators = 500 and max depth = 5
Log Loss : 1.3209257147466733
for n_estimators = 500 and max depth = 10
Log Loss : 1.2990499653854541
for n_estimators = 1000 and max depth = 5
Log Loss : 1.3143147692311712
for n_estimators = 1000 and max depth = 10
Log Loss : 1.295645557896047
for n_estimators = 2000 and max depth = 5
Log Loss : 1.3087687029995945
for n_estimators = 2000 and max depth = 10
Log Loss : 1.292728549501082
For values of best estimator = 2000 The train log loss is: 1.018665707188
1717
For values of best estimator = 2000 The cross validation log loss is: 1.2
92728549501082
For values of best estimator = 2000 The test log loss is: 1.3010182942066
217
```

In [113]:

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', m
ax_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train1000, train_y,cv1000,cv_y, clf)
```

Log loss : 1.2927285495010823
Number of mis-classified points : 0.4398496240601504

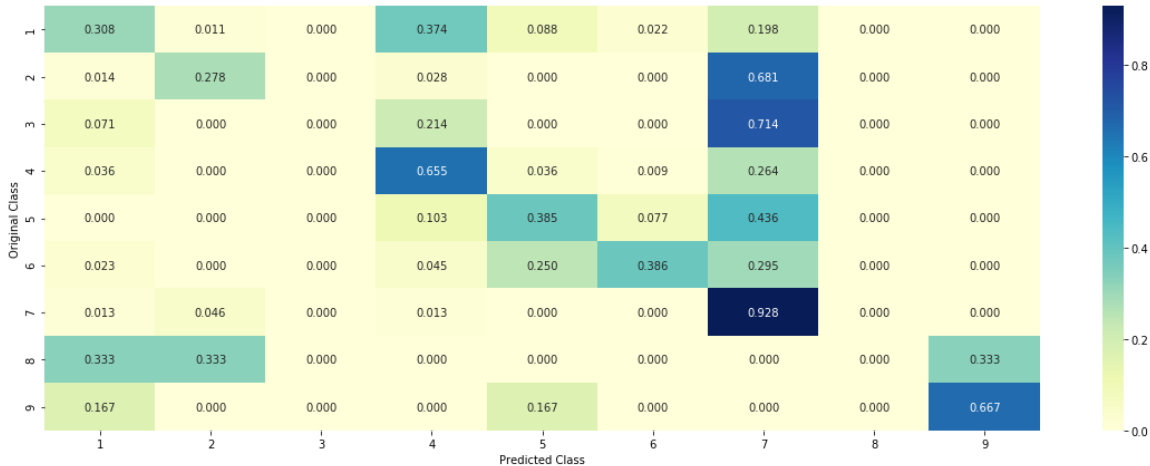
----- Confusion matrix -----



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



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



Logistic regression with top 1000 tfidf

In [114]:

```
# Logistic regression for top 1000 tfidf values

alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train1000, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train1000, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv1000)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilities we use log-probability estimates
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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

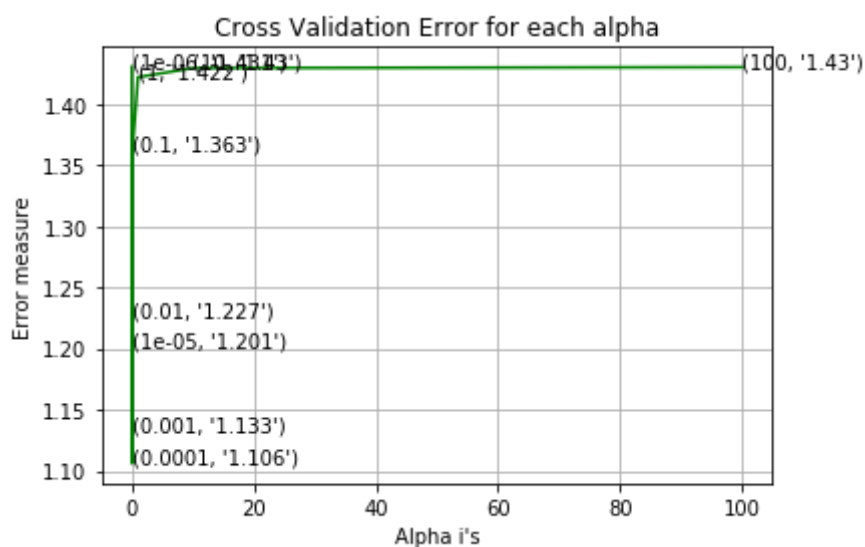
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train1000, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train1000, train_y)

predict_y = sig_clf.predict_proba(train1000)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv1000)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test1000)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```

for alpha = 1e-06
Log Loss : 1.430676614932548
for alpha = 1e-05
Log Loss : 1.20123517349416
for alpha = 0.0001
Log Loss : 1.1059192218590044
for alpha = 0.001
Log Loss : 1.1325492792376088
for alpha = 0.01
Log Loss : 1.2265621515946228
for alpha = 0.1
Log Loss : 1.362988722947367
for alpha = 1
Log Loss : 1.4221093499217863
for alpha = 10
Log Loss : 1.4295541951031125
for alpha = 100
Log Loss : 1.4303341219368102

```



For values of best alpha = 0.0001 The train log loss is: 0.4903307095055881

For values of best alpha = 0.0001 The cross validation log loss is: 1.1059192218590044

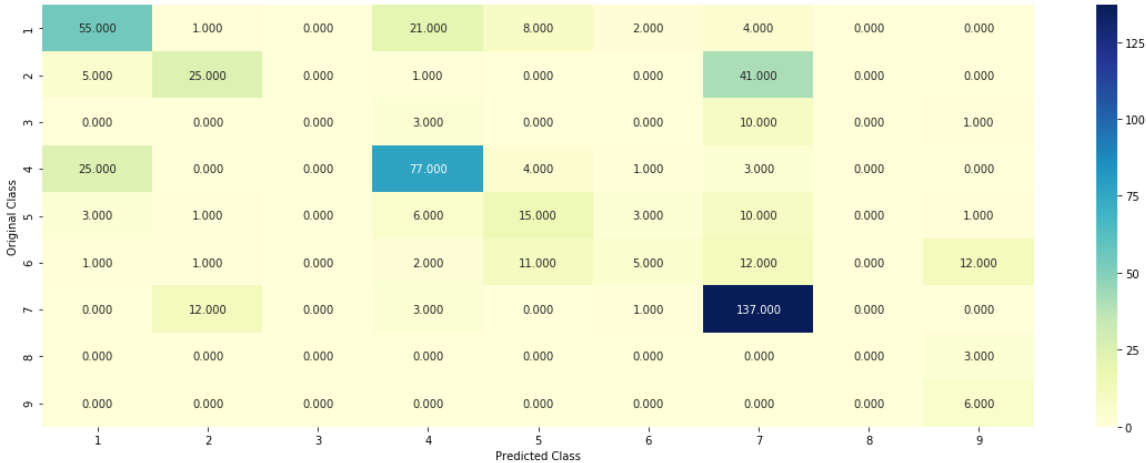
For values of best alpha = 0.0001 The test log loss is: 1.092226613685411

In [115]:

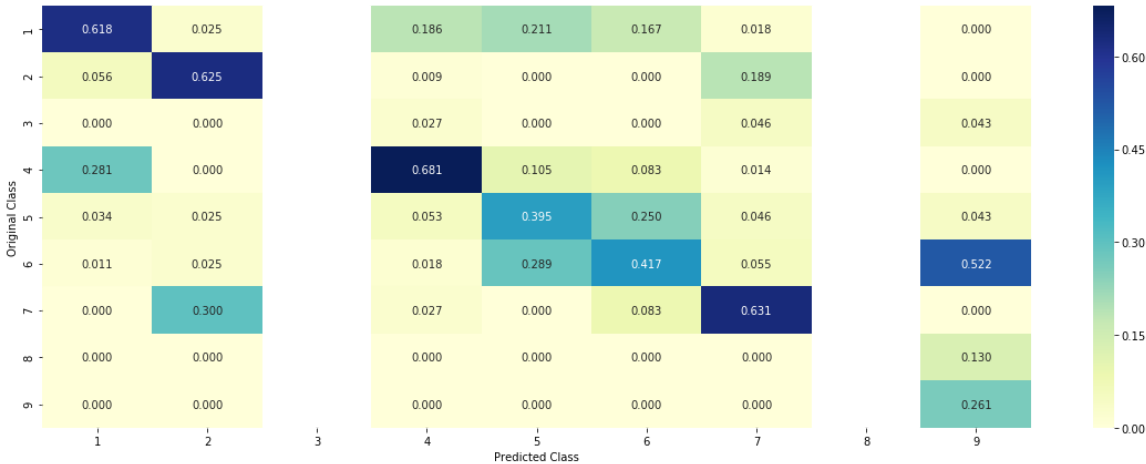
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train1000, train_y, cv1000, cv_y, clf)
```

Log loss : 1.1059192218590044
Number of mis-classified points : 0.39849624060150374

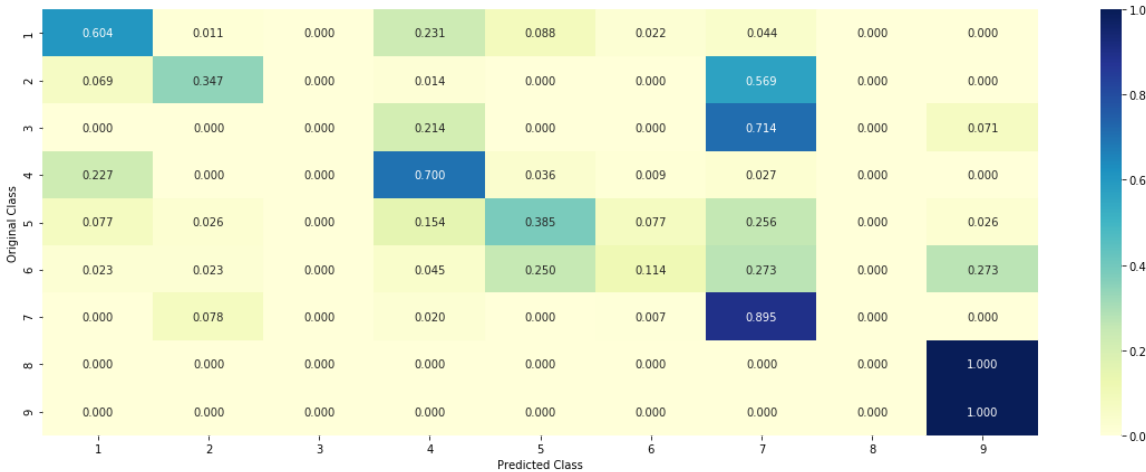
----- Confusion matrix -----



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



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



TASK 3

Count Vectorizer on Logistic regression with unigram and bigram

In [120]:

```
bow_vec = CountVectorizer(ngram_range=(1,2),min_df=3,max_features=100000)
train_text_feature_bow = bow_vec.fit_transform(train_df['TEXT'])
cv_text_feature_bow = bow_vec.transform(cv_df['TEXT'])
test_text_feature_bow = bow_vec.transform(test_df['TEXT'])
```

In [121]:

```
train_text_feature_bow = normalize(train_text_feature_bow, axis=0)
cv_text_feature_bow = normalize(cv_text_feature_bow, axis=0)
test_text_feature_bow = normalize(test_text_feature_bow, axis=0)
```

In [123]:

```
train_bow = hstack((train_gene_var_onehotCoding,train_text_feature_bow)).tocsr()
cv_bow = hstack((cv_gene_var_onehotCoding,cv_text_feature_bow)).tocsr()
test_bow = hstack((test_gene_var_onehotCoding,test_text_feature_bow)).tocsr()
```

In []:

In [124]:

```
train_bow.shape
```

Out[124]:

```
(2124, 102191)
```

In []:

In [125]:

```
# Logistic regression for top 1000 tfidf values

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

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

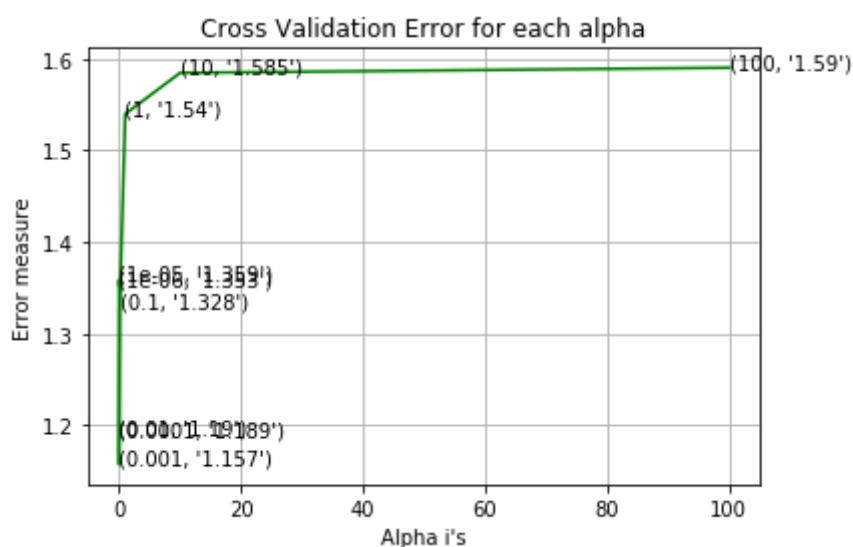
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_bow, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_bow, train_y)

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

```

for alpha = 1e-06
Log Loss : 1.353159414864387
for alpha = 1e-05
Log Loss : 1.358733393185087
for alpha = 0.0001
Log Loss : 1.189237105089507
for alpha = 0.001
Log Loss : 1.1570352002164062
for alpha = 0.01
Log Loss : 1.189695575738901
for alpha = 0.1
Log Loss : 1.3282524580826007
for alpha = 1
Log Loss : 1.5397984068712922
for alpha = 10
Log Loss : 1.584735653896891
for alpha = 100
Log Loss : 1.590332918008494

```



For values of best alpha = 0.001 The train log loss is: 0.5321697366509359

For values of best alpha = 0.001 The cross validation log loss is: 1.1570352002164062

For values of best alpha = 0.001 The test log loss is: 1.0681593073345577

In [126]:

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_bow, train_y, cv_bow, cv_y, clf)
```

Log loss : 1.1570352002164062

Number of mis-classified points : 0.32894736842105265

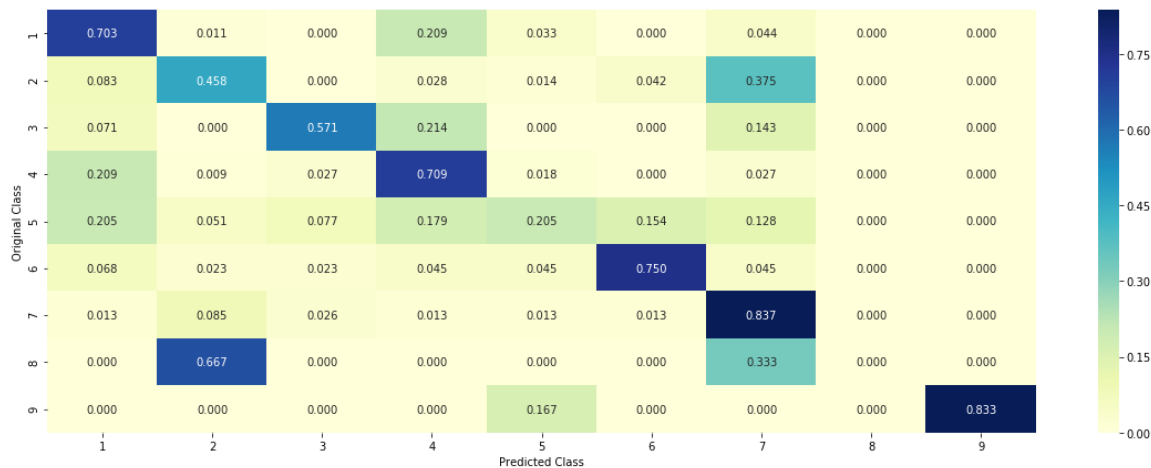
----- Confusion matrix -----



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



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



Conclusion

In [5]:

```
from prettytable import PrettyTable
x = PrettyTable()
x.title = " Comparision of Models for TFIDF vectorized words and Response Coding"

x.field_names = ["Vectorizer", "Model Name", "Train Log Loss", "Test Log Loss", "CV Log Loss", "Misclassification %"]
x.add_row(["OHE + TF-IDF(Text)", "Naive Bayes", 0.919, 1.189, 1.209, 42.6])
x.add_row(["Response Coding", "KNN", 0.801, 1.075, 1.051, 37.2])
x.add_row(["OHE + TF-IDF(Text)", "Logistic Reg (Class Balancing)", 0.463, 1.041, 1.139, 33.4])
x.add_row(["OHE + TF-IDF(Text)", "Logistic Reg (No Class Balancing)", 0.471, 1.153, 1.042, 33.6])
x.add_row(["OHE + TF-IDF(Text)", "Linear SVM", 0.517, 1.197, 1.085, 34.7])
x.add_row(["OHE + TF-IDF(Text)", "Random Forest", 0.635, 1.166, 1.152, 41.1])
x.add_row(["Response Coding", "Random Forest", 0.063, 1.207, 1.22, 40.0])
x.add_row(["OHE + TF-IDF(Text)", "Stacking Classifier", 0.467, 1.181, 1.119, 33.5])

print(x)
```

Vectorizer	Model Name	Train Log Loss
Test Log Loss CV Log Loss Misclassification %		
OHE + TF-IDF(Text)	Naive Bayes	0.919
1.189 1.209 42.6		
Response Coding	KNN	0.801
1.075 1.051 37.2		
OHE + TF-IDF(Text)	Logistic Reg (Class Balancing)	0.463
1.041 1.139 33.4		
OHE + TF-IDF(Text)	Logistic Reg (No Class Balancing)	0.471
1.153 1.042 33.6		
OHE + TF-IDF(Text)	Linear SVM	0.517
1.197 1.085 34.7		
OHE + TF-IDF(Text)	Random Forest	0.635
1.166 1.152 41.1		
Response Coding	Random Forest	0.063
1.207 1.22 40.0		
OHE + TF-IDF(Text)	Stacking Classifier	0.467
1.181 1.119 33.5		

In [8]:

```

from prettytable import PrettyTable
x = PrettyTable()
x.title = " Comparision of Models for 1000 TFIDF vectorized words and Response Coding"

x.field_names = ["Vectorizer", "Model Name", "Train Log Loss", "Test Log Loss", "CV Log Loss", "Misclassification %"]
x.add_row(["OHE + TF-IDF(Text)", "Logistic Reg (Class Balancing)", 0.490, 1.105, 1.092, 39.8])
x.add_row(["Response Coding", "Random Forest", 1.018, 1.292, 1.301, 43.0])

print(x)

```

Vectorizer	Model Name	Train Log Loss	Test Log Loss	CV Log Loss	Misclassification %
OHE + TF-IDF(Text)	Logistic Reg (Class Balancing)	0.49	1.105	1.092	39.8
Response Coding	Random Forest	1.018	1.292	1.301	43.0

In [12]:

```

from prettytable import PrettyTable
x = PrettyTable()
x.title = " Logistic Regression on Count vectorized words using unigrams and bi grams"

x.field_names = [ "Train Log Loss", "Test Log Loss", "CV Log Loss", "Misclassification %"]
x.add_row([0.532, 1.157, 1.068, 32.8])
print(x)

```

Train Log Loss	Test Log Loss	CV Log Loss	Misclassification %
0.532	1.157	1.068	32.8

In []:

In []:

In []:

In []: