Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

2. https://www.youtube.com/watch?v=UwbuW7oK8rk
3. https://www.youtube.com/watch?v=qxXRKVompI8

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

ID,Text
0||Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity revealed. Previous work has shown that CDK10 silencing increases ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The precise mechanisms by which CDK10 modulates ETS2 activity, and more generally the functions of CDK10, remain elusive. Here we demonstrate that CDK10 is a cyclin-dependent kinase by identifying cyclin M as an activating cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR syndrome, a human developmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malformations. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and in conferring tamoxifen resistance to breast cancer cells. CDK10/cyclin M phosphorylates ETS2 in vitro, and in cells it positively controls ETS2 degradation by the proteasome. ETS2 protein levels are increased in cells derived from a STAR patient, and this increase is attributable to decreased cyclin M levels. Altogether, our results reveal an additional regulatory mechanism for ETS2, which plays key roles in cancer and development. They also shed light on the molecular mechanisms underlying STAR syndrome.Cyclin-dependent kinases (CDKs) play a pivotal role in the control of a number of fundamental cellular processes (1). The human genome contains 21 genes encoding proteins

that can be considered as members of the CDK family owing to their sequence similarity with bona fide CDKs, those known to be activated by cyclins (2). Although discovered almost 20 y ago (3, 4), CDK10 remains one of the two CDKs without an identified cyclin partner. This knowledge gap has largely impeded the exploration of its biological functions. CDK10 can act as a positive cell cycle regulator in some cells (5, 6) or as a tumor suppressor in others (7, 8). CDK10 interacts with the ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity through an unknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK pathway, and induces resistance of MCF7 cells to tamoxifen (6). ...

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [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
```

```python
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.feature_selection import SelectKBest, chi2
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
```

```
Using TensorFlow backend.
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtyp
es.py:526: FutureWarning: Passing (type, 1) or '1type' as a synonym of
type is deprecated; in a future version of numpy, it will be understood
as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtyp
es.py:527: FutureWarning: Passing (type, 1) or '1type' as a synonym of
type is deprecated; in a future version of numpy, it will be understood
as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtyp
es.py:528: FutureWarning: Passing (type, 1) or '1type' as a synonym of
type is deprecated; in a future version of numpy, it will be understood
as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtyp
```

```
es.py:529: FutureWarning: Passing (type, 1) or '1type' as a synonym of
type is deprecated; in a future version of numpy, it will be understood
as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtyp
es.py:530: FutureWarning: Passing (type, 1) or '1type' as a synonym of
type is deprecated; in a future version of numpy, it will be understood
as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/usr/local/lib/python3.5/dist-packages/tensorflow/python/framework/dtyp
es.py:535: FutureWarning: Passing (type, 1) or '1type' as a synonym of
type is deprecated; in a future version of numpy, it will be understood
as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:

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

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

Out[2]:

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

	ID	Gene	Variation	Class
3	3	CBL	N454D	3
4	4	CBL	L399V	4

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

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

3.1.2. Reading Text Data

In [3]:

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

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

Out[3]:

	ID	TEXT
0	0	Cyclin-dependent kinases (CDKs) regulate a var...
1	1	Abstract Background Non-small cell lung canc...

	ID	TEXT
2	2	Abstract Background Non-small cell lung canc...
3	3	Recent evidence has demonstrated that acquired...
4	4	Oncogenic mutations in the monomeric Casitas B...

3.1.3. Preprocessing of text

In [4]:

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


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

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

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

In [5]:

```
#text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
```

```
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_tim
e, "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 : 37.336287999999996 seconds
```

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

Out[6]:

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

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

Out[7]:

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

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

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

Out[9]:

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

3.1.4. Test, Train and Cross Validation Split

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

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

# split the data into test and train by maintaining same distribution o
f output varaible 'y_true' [stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(result, y_true, st
ratify=y_true, test_size=0.2)
# split the train data into train and cross validation by maintaining s
ame distribution of output varaible 'y_train' [stratify=y_train]
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, str
atify=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 [11]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0
])
```

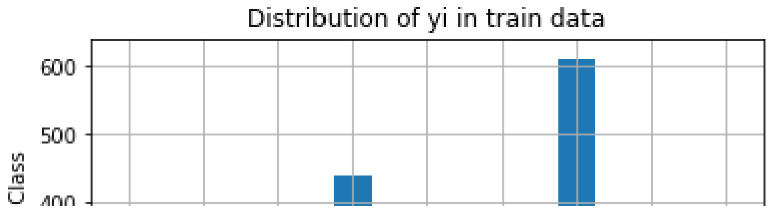
```
Number of data points in train data: 2124
```

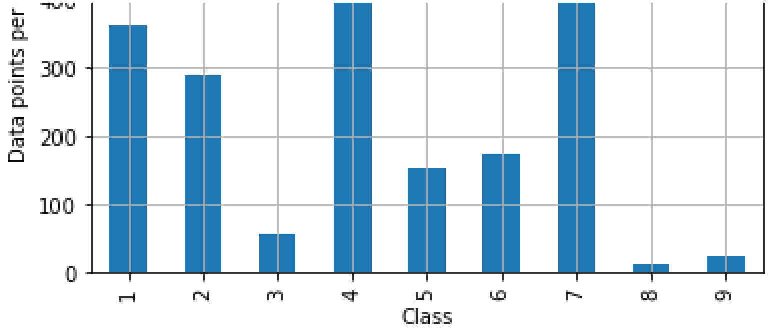
Number of data points in test data: 665
Number of data points in cross validation data: 532

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

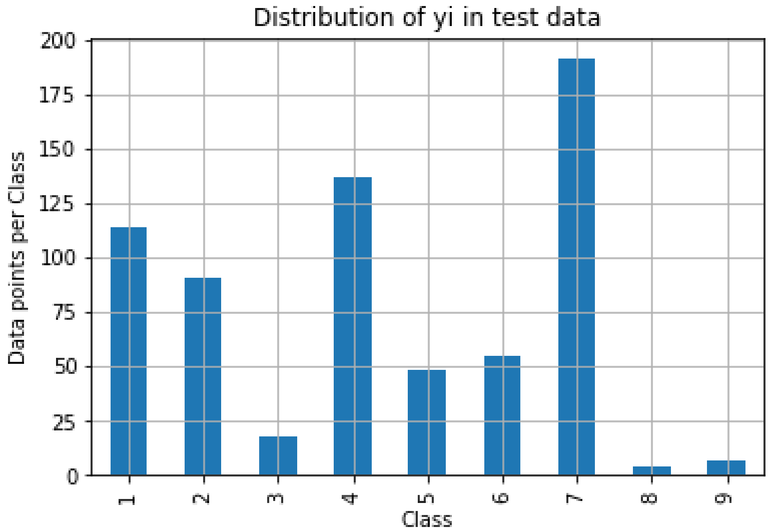
In [12]:
```
# it returns a dict, keys as class labels and values as the number of d
ata points in that class
train_class_distribution = train_df['Class'].value_counts().sort_index
()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/num
py.argsort.html
# -(train_class_distribution.values): the minus sign will give us in de
creasing 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_distri
bution.values[i],
          '(', np.round((train_class_distribution.values[i]/train_df.sh
ape[0]*100), 3), '%)')


print('-'*80)
my_colors = 'rgbkymc'
test_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
```

```
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/num
py.argsort.html
# -(train_class_distribution.values): the minus sign will give us in de
creasing 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_distrib
ution.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')
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/num
py.argsort.html
# -(train_class_distribution.values): the minus sign will give us in de
creasing 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_distribut
ion.values[i], '(',
          np.round((cv_class_distribution.values[i]/cv_df.shape[0]*100
), 3), '%)')
```



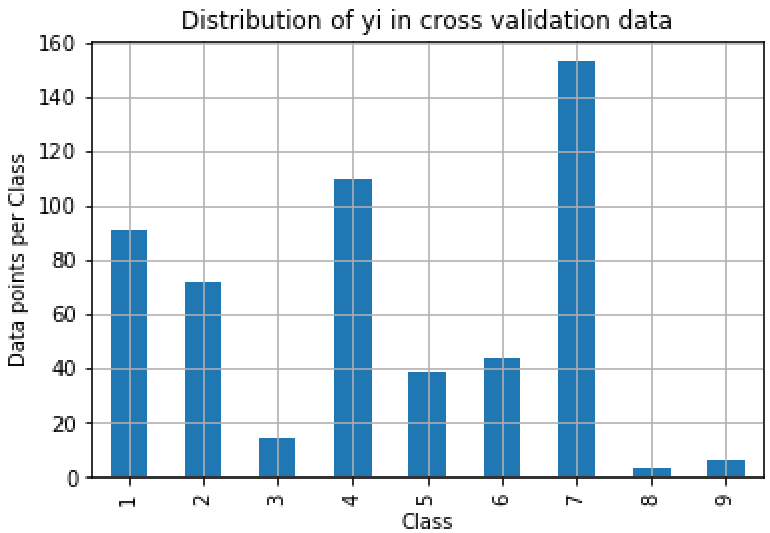



```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298 %)
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 7 : 191 ( 28.722 %)
Number of data points in class 4 : 137 ( 20.602 %)
Number of data points in class 1 : 114 ( 17.143 %)
Number of data points in class 2 : 91 ( 13.684 %)
Number of data points in class 6 : 55 ( 8.271 %)
Number of data points in class 5 : 48 ( 7.218 %)
Number of data points in class 3 : 18 ( 2.707 %)
Number of data points in class 9 : 7 ( 1.053 %)
Number of data points in class 8 : 4 ( 0.602 %)
----------------------------------------------------------------------
---------
```




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

3.2 Prediction using a 'Random' Model

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

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

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

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

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

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

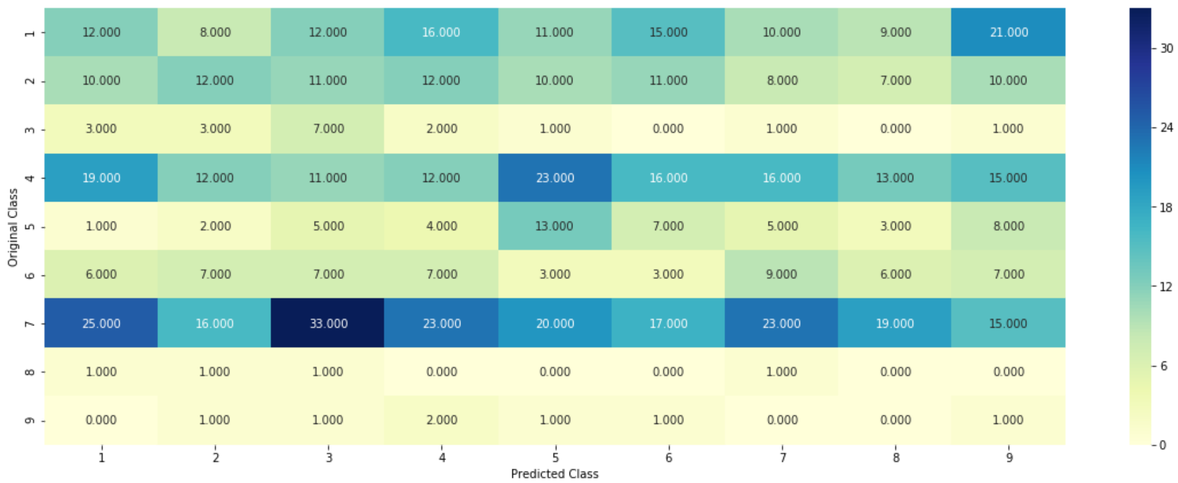
```
    #                      [3/4, 4/6]]

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

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

In [14]:

```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers
 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):
```

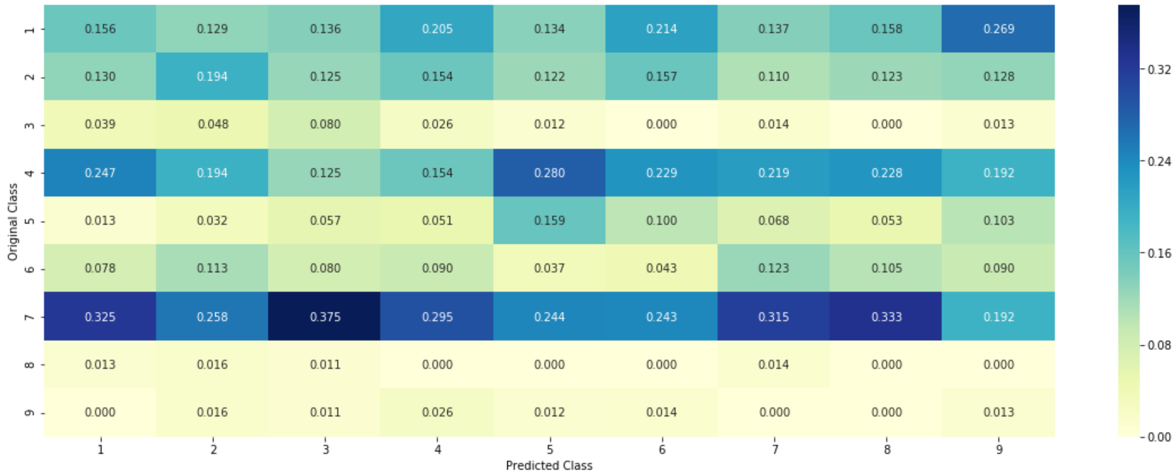
```
    rand_probs = np.random.rand(1,9)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Cross Validation Data using Random Model",log_loss(y
_cv,cv_predicted_y, eps=1e-15))


# 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_p
redicted_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 2.47254882158038
13
Log loss on Test Data using Random Model 2.451950031713821
-------------------- Confusion matrix --------------------
```

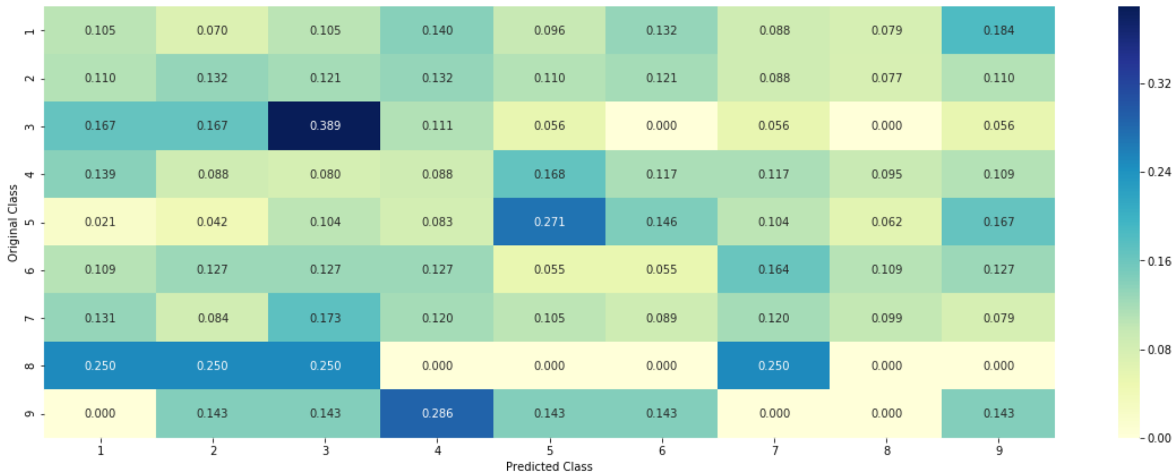


	1	2	3	4	5	6	7	8	9
1	12.000	8.000	12.000	16.000	11.000	15.000	10.000	9.000	21.000
2	10.000	12.000	11.000	12.000	10.000	11.000	8.000	7.000	10.000
3	3.000	3.000	7.000	2.000	1.000	0.000	1.000	0.000	1.000
4	19.000	12.000	11.000	12.000	23.000	16.000	16.000	13.000	15.000
5	1.000	2.000	5.000	4.000	13.000	7.000	5.000	3.000	8.000
6	6.000	7.000	7.000	7.000	3.000	3.000	9.000	6.000	7.000
7	25.000	16.000	33.000	23.000	20.000	17.000	23.000	19.000	15.000
8	1.000	1.000	1.000	0.000	0.000	0.000	1.000	0.000	0.000
9	0.000	1.000	1.000	2.000	1.000	1.000	0.000	0.000	1.000

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




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



3.3 Univariate Analysis

In [15]:
```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
```

```
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feat
ure in train data dataframe
# build a vector (1*9) , the first element = (number of times it occure
d in class1 + 10*alpha / number of time it occurred in total data+90*al
pha)
# gv_dict is like a look up table, for every gene it store a (1*9) repr
esentation 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_f
ea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

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

```
    # Deletion                                     43
    # Amplification                                43
    # Fusions                                      22
    # Overexpression                                3
    # E17K                                          3
    # Q61L                                          3
    # S222D                                         2
    # P130S                                         2
    # ...
    # }
    value_count = train_df[feature].value_counts()

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

    # denominator will contain the number of time that particular featu
re occured in whole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation b
elongs to perticular class
        # vec is 9 diamensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Ge
ne']=='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[f
eature]==i)]
```

```
            # cls_cnt.shape[0](numerator) will contain the number of ti
me that particular feature occured 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 a
s 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, 0.068181
818181818177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.0378787
8787878788, 0.03787878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224
489795918366, 0.27040816326530615, 0.061224489795918366, 0.066326530612
244902, 0.051020408163265307, 0.051020408163265307, 0.05612244897959183
7],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625,
 0.068181818181818177, 0.068181818181818177, 0.0625, 0.3465909090909091
2, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.06060
6060606060608, 0.078787878787878782, 0.1393939393939394, 0.345454545454
54546, 0.060606060606060608, 0.060606060606060608, 0.06060606060606060
8],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.06918
2389937106917, 0.46540880503144655, 0.075471698113207544, 0.06289308176
1006289, 0.069182389937106917, 0.062893081761006289, 0.0628930817610062
89],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.0728476
82119205295, 0.072847682119205295, 0.066225165562913912, 0.066225165562
913912, 0.27152317880794702, 0.066225165562913912, 0.06622516556291391
2],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333
333333333334, 0.073333333333333334, 0.093333333333333338, 0.08000000000
0000002, 0.29999999999999999, 0.066666666666666666, 0.06666666666666666
6],
```

```
    #       ...
    #     }
    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 e
ach 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_fe
a
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#             gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea
```

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

In [16]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
```

```
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 231
BRCA1     173
TP53       99
EGFR       84
PTEN       84
BRCA2      78
KIT        66
BRAF       60
ALK        49
ERBB2      45
PDGFRA     40
Name: Gene, dtype: int64
```

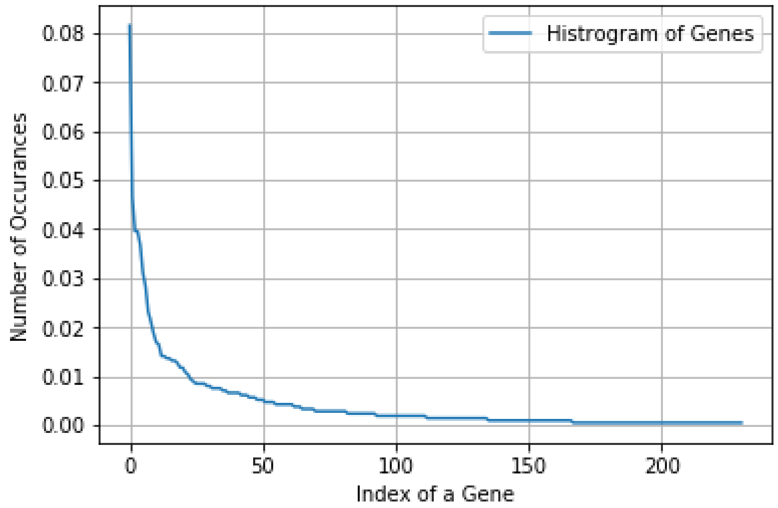
In [17]:

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

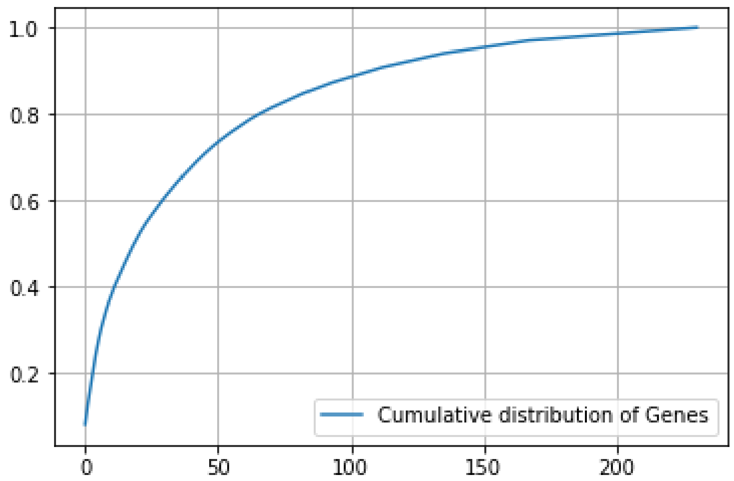
```
Ans: There are 231 different categories of genes in the train data, and they are distibuted as follows
```

In [18]:

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



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



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

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

In [21]:
```
print("train_gene_feature_responseCoding is converted feature using res
pone coding method. The shape of gene feature:", train_gene_feature_res
ponseCoding.shape)
```

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

In [22]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_d
f['Gene'])
```

```
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gen
e'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

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

Out[23]:
```
1329      MLH1
3241      DDR2
2826     BRCA2
1935    CARD11
3286       RET
Name: Gene, dtype: object
```

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

Out[24]:
```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl1',
 'atm',
 'atr',
 'atrx',
 'aurkb',
 'axin1',
 'axl',
 'b2m',
 'bap1',
 'bcl10',
```

```
'bcl2',
'bcl2l11',
'bcor',
'braf',
'brca1',
'brca2',
'brd4',
'brip1',
'btk',
'card11',
'carm1',
'casp8',
'cbl',
'ccnd1',
'ccnd3',
'ccne1',
'cdh1',
'cdk12',
'cdk4',
'cdk6',
'cdk8',
'cdkn1a',
'cdkn1b',
'cdkn2a',
'cdkn2b',
'chek2',
'cic',
'crebbp',
'ctcf',
'ctla4',
'ctnnb1',
'ddr2',
'dicer1',
'dnmt3a',
'dnmt3b',
'dusp4',
'egfr',
'eif1ax',
'elf3',
```

```
'ep300',
'epas1',
'epcam',
'erbb2',
'erbb3',
'erbb4',
'ercc2',
'ercc3',
'ercc4',
'erg',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fam58a',
'fanca',
'fancc',
'fat1',
'fbxw7',
'fgf4',
'fgfr1',
'fgfr2',
'fgfr3',
'fgfr4',
'flt1',
'flt3',
'foxa1',
'foxl2',
'fubp1',
'gata3',
'gli1',
'gnaq',
'gnas',
'h3f3a',
'hist1h1c',
'hla',
'hnf1a',
'hras',
```

```
'idh1',
'idh2',
'igf1r',
'ikbke',
'ikzf1',
'inpp4b',
'jak1',
'jak2',
'kdm5a',
'kdm5c',
'kdr',
'keap1',
'kit',
'klf4',
'kmt2a',
'kmt2c',
'kmt2d',
'knstrn',
'kras',
'lats1',
'lats2',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
'mdm2',
'med12',
'mef2b',
'men1',
'met',
'mga',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'mycn',
'myd88',
```

```
'myod1',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',
'nkx2',
'notch1',
'notch2',
'npm1',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'nup93',
'pak1',
'pax8',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pim1',
'pms1',
'pms2',
'pole',
'ppm1d',
'ppp2r1a',
'ppp6c',
'prdm1',
'ptch1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rab35',
```

```
'rac1',
'rad21',
'rad50',
'rad51d',
'raf1',
'rasa1',
'rb1',
'rbm10',
'ret',
'rheb',
'rhoa',
'rictor',
'rit1',
'rnf43',
'ros1',
'runx1',
'rxra',
'rybp',
'sdhb',
'sdhc',
'setd2',
'sf3b1',
'shoc2',
'smad2',
'smad3',
'smad4',
'smarca4',
'smo',
'sos1',
'sox9',
'spop',
'src',
'stag2',
'stat3',
'stk11',
'tert',
'tet1',
'tet2',
'tgfbr1',
```

```
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'xpo1',
 'xrcc2',
 'yap1']
```

In [25]:
```
print("train_gene_feature_onehotCoding is converted feature using one-h
ot encoding method. The shape of gene feature:", train_gene_feature_one
hotCoding.shape)
```

```
train_gene_feature_onehotCoding is converted feature using one-hot enco
ding method. The shape of gene feature: (2124, 230)
```

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

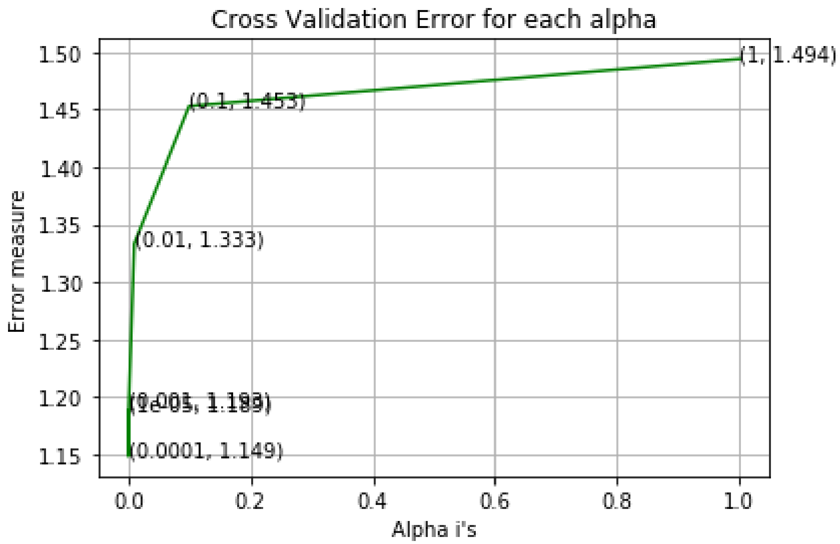
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

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


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state
=42)
    clf.fit(train_gene_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.clas
ses_, 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_arra
y[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


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

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

predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log
 loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation 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 l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.1885745762380122
For values of alpha =  0.0001 The log loss is: 1.1485297681181
For values of alpha =  0.001 The log loss is: 1.1927598010920486
For values of alpha =  0.01 The log loss is: 1.3331017799562281
For values of alpha =  0.1 The log loss is: 1.4527376410912924
For values of alpha =  1 The log loss is: 1.4937824848090988
```



```
For values of best alpha =  0.0001 The train log loss is: 1.02381276072
2981
For values of best alpha =  0.0001 The cross validation log loss is: 1.
1485297681181
For values of best alpha =  0.0001 The test log loss is: 1.203884111709
657
```

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

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

In [27]:

```
print("Q6. How many data points in Test and CV datasets are covered by
 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'])))].shap
e[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  23
1  genes in train dataset?
Ans
1. In test data 635 out of 665 : 95.48872180451127
2. In cross validation data 518 out of  532 : 97.36842105263158
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

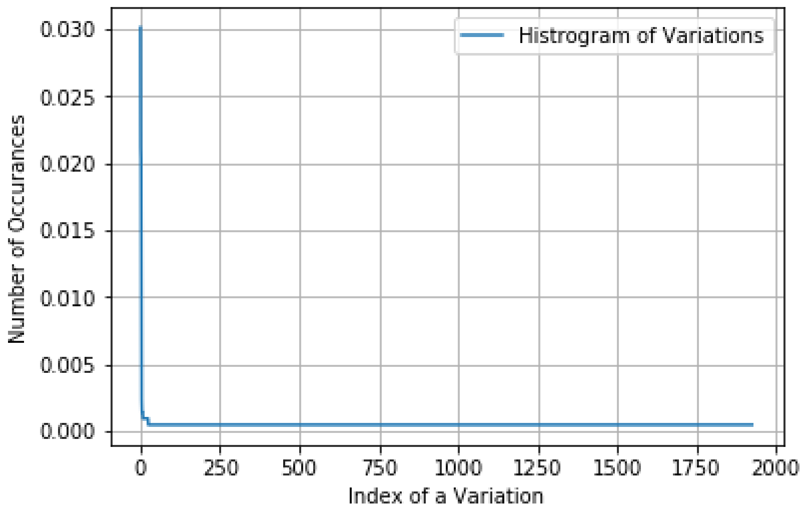
In [28]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1926
Truncating_Mutations    64
Deletion                45
Amplification           43
Fusions                 23
Overexpression           5
G12V                     3
Q61L                     3
E17K                     3
F384L                    2
E542K                    2
Name: Variation, dtype: int64
```

In [29]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categori
es of variations in the train data, and they are distibuted as follows"
,)
```

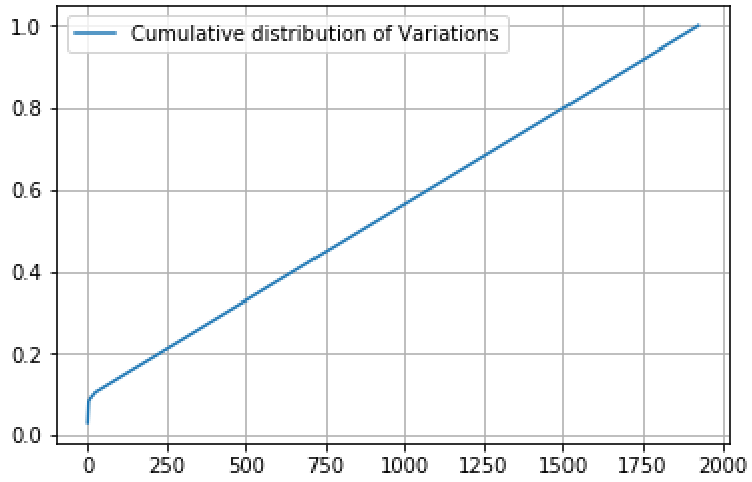
```
Ans: There are 1926 different categories of variations in the train dat
a, and they are distibuted as follows
```

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



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

```
[0.03013183 0.05131827 0.07156309 ... 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 [32]:
```
# 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, "V
ariation", cv_df))
```

In [33]:
```
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 [34]:
```
# 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 [35]:
```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Variation feature:", train_variation_feature_onehotCoding.shape)
```

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Variation feature: (2124, 1959)
```

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

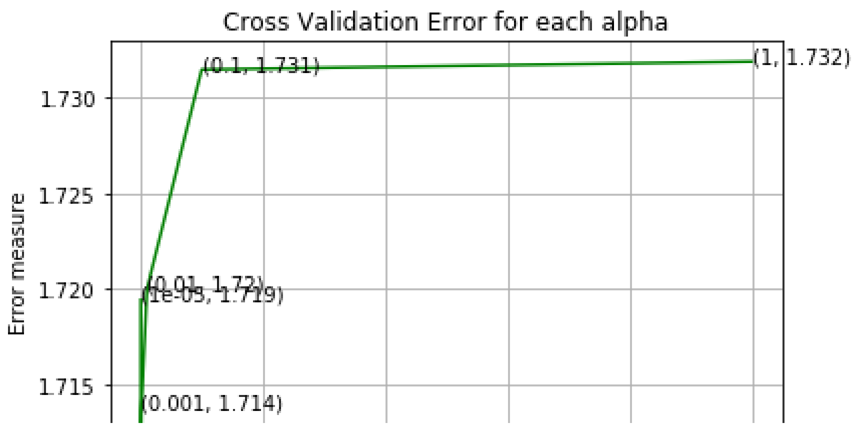
Let's build a model just like the earlier!

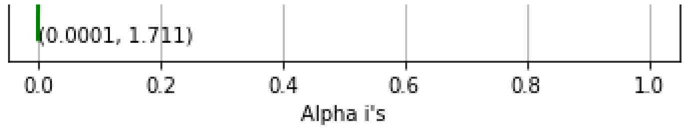
In [36]:
```
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/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, le
```

```
arning_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 S
tochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

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



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

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

    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.clas
ses_, 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_arra
y[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
```

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

predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_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: 1.7194533608607767
For values of alpha =  0.0001 The log loss is: 1.7111639105638812
For values of alpha =  0.001 The log loss is: 1.7137569535769832
For values of alpha =  0.01 The log loss is: 1.7199902607737223
For values of alpha =  0.1 The log loss is: 1.7314355891303508
For values of alpha =  1 The log loss is: 1.7318524504229615
```






```
For values of best alpha =  0.0001 The train log loss is: 0.78537930027
90044
For values of best alpha =  0.0001 The cross validation log loss is: 1.

7111639105638812
For values of best alpha =  0.0001 The test log loss is: 1.695872260299
6184
```

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

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

In [37]:

```
print("Q12. How many data points are covered by total ", unique_variati
ons.shape[0], " genes in test and cross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Vari
ation'])))].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  1926  genes in test and
cross validation data sets?
Ans
1. In test data 75 out of 665 : 11.278195488721805
2. In cross validation data 42 out of  532 : 7.894736842105263
```

3.2.3 Univariate Analysis on Text Feature

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

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

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

In [39]:
```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(t
otal_dict.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_pr
ob/len(row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

In [40]:
```
# building a TFIDF vectorizer
text_vectorizer = TfidfVectorizer(ngram_range=(1,4), min_df=3)
```

```
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_d
f['TEXT'])

# selecting top 2000 features using SelectKBest function
k_best_clf = SelectKBest(chi2, k=2000)
train_text_feature_onehotCoding = k_best_clf.fit_transform(train_text_f
eature_onehotCoding, y_train)
print("The Shape of the One hot encoded TFIDF vectorizer train data :",
 train_text_feature_onehotCoding.shape)

# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

# getting the top 2000 selected feature names
train_text_features = [train_text_features[index] for index, val in enu
merate(k_best_clf.get_support()) if val == True]

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

# zip(list(text_features),text_fea_counts) will zip a word with its num
ber of times it occured
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_count
s))


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

```
The Shape of the One hot encoded TFIDF vectorizer train data : (2124, 2
000)
Total number of unique words in train data : 2000
```

Out[40]:
```
['00',
 '00 deleterious',
 '0094',
 '02',
 '04',
```

```
'06',
'07',

'09',
'09 00',
'10 10 101',
'10 10 101 100',
'10 101',
'10 101 100',
'10 101 100 10',
'10 1158',
'10 1158 2159',
'10 1158 2159 8290',
'100 favor',
'1000 genomes data',
'1004',
'101 100',
'101 100 10',
'1010',
'104',
'108 8639',
'108 8639 japan',
'1158 2159',
'1158 2159 8290',
'1158 2159 8290 cd',
'117',
'12 mds',
'13 0094',
'140 mutations',
'17 p11',
'18 2013',
'18 2013 doi',
'18 2013 doi 10',
'1g',
'20 favor',
'2006 vol 15',
'2011 2011 macmillan',
'2011 2011 macmillan publishers',
'2011 june',
'2011 june 15',
```

```
'2011 macmillan',
'2011 macmillan publishers',

'2011 macmillan publishers limited',
'2013 american',
'2013 american association',
'2013 american association cancer',
'2013 doi',
'2013 doi 10',
'2013 doi 10 1158',
'2017 2013',
'2017 2013 american',
'2017 2013 american association',
'21 44',
'2159',
'2159 8290',
'2159 8290 cd',
'2159 8290 cd 13',
'26 2017',
'26 2017 2013',
'26 2017 2013 american',
'2hg',
'2hg 2hg',
'2hg accumulation',
'2hg levels',
'2hg production',
'2og',
'34 tumors',
'34 tumors showed',
'348 1018',
'39ss',
'3at',
'40 49',
'422 su',
'433',
'433 sequence',
'433 variants',
'478 october',
'478 october 2011',
'478 october 2011 2011',
'50 59',
```

```
'50 59',
'5382insc',

'57',
'57 tumors',
'641',
'7647',
'7721',
'8290',
'8290 cd',
'8290 cd 13',
'8290 cd 13 0094',
'8639',
'8639 japan',
'90 191',
'a2 a3',
'a4',
'a4 a5',
'aacrjournals org may 26',
'abcb7',
'abcc5',
'aberrantly spliced',
'ability pten',
'abl',
'abl kd',
'abl kd mutation',
'abl kd mutations',
'abl transcript',
'abnormalities sf3b1',
'acc',
'according diseases',
'acid receptor alpha',
'act dominantly',
'act1',
'activating',
'activating mtor',
'activating mtor mutations',
'activation',
'activity dimethylated',
'activity pten',
'acts transcriptional repressor',
```

```
'acts transcriptional repressor',
'adult gbm',

'aebp2',
'affected brca',
'ag dinucleotide',
'age 50',
'aggregation',
'ago hook',
'akt',
'akt ha',
'akt immunoprecipitated',
'akt inhibitors',
'akt1',
'akt1 k179m',
'akt2',
'al page cancer',
'al page cancer res',
'ala',
'ala scanning',
'ala substitutions',
'alad',
'alas2',
'alb',
'align',
'align gvgd',
'alk',
'alk mutants',
'alk mutations',
'alk tkd',
'alka1099t',
'alka1099t alkt1151m',
'alka1099t alkt1151m alka1234t',
'alka1099t alkt1151m alka1234t alkr1464stop',
'alka1234t',
'alka1234t alkr1464stop',
'alkd1091n',
'alkd1091n alka1099t',
'alkd1091n alka1099t alkt1151m',
'alkd1091n alka1099t alkt1151m alka1234t',
'alkf1174'
```

```
'alkf1174',
'alkf1174i',

'alkf1174l',
'alkf1174s',
'alkm1166r',
'alkr1464stop',
'alkt1087i',
'alkt1087i alkd1091n',
'alkt1087i alkd1091n alka1099t',
'alkt1087i alkd1091n alka1099t alkt1151m',
'alkt1151m',
'alkt1151m alka1234t',
'alkt1151m alka1234t alkr1464stop',
'allele specific loh',
'alternative acceptor splice',
'alternative acceptor splice sites',
'alternative lengthening',
'alternative lengthening telomeres',
'alternative terminal',
'alternative terminal exons',
'aml1',
'analysis brca1',
'analysis logistic',
'analysis using dexseq',
'and6 appendix',
'and6 appendix tables',
'and6 appendix tables a4',
'ankhd1',
'antibodies indicated proteins',
'apl',
'appendix tables',
'appendix tables a2',
'appendix tables a2 a3',
'appendix tables a4',
'appendix tables a4 a5',
'array rna',
'array rna seq',
'artifi',
'artifi cial',
```

```
'asd',
'asd dd',

'asd dd related',
'asp92',
'asp92 residue',
'assay',
'assays',
'assessment pten',
'assessment pten mutations',
'associated alternative',
'associated alternative lengthening',
'associated mtor',
'associated sf3b1',
'associated sf3b1 mutations',
'association cancer research published',
'asxl1',
'asxl1 cbl',
'atp site',
'atra',
'atra treatment',
'atrx',
'atrx daxx',
'atrx daxx mutations',
'atrx loss',
'aupd',
'author',
'author manuscript',
'author manuscript available',
'author manuscript available pmc',
'author manuscript nih',
'author manuscript nih pa',
'autism',
'auxiliary factor',
'available pathology',
'available pmc',
'available pmc 2011',
'available pmc 2011 june',
'axl',
'azd4547',
```

```
'b6',
'b6 ly5',

'ba',
'ba f3',
'ba f3 cells',
'bach1',
'baf180',
'bap1',
'bard1',
'bard1 e2',
'bard1 heterodimer',
'bard1 interaction',
'basis family',
'basis family history',
'bc oc',
'bclaf1',
'bclaf1 sf3b1',
'bclaf1 sf3b1 complex',
'bcor',
'bcor bcorl1',
'bcor ccnb3',
'bcor mutations',
'bcorl1',
'bcr',
'bcr abl',
'bcr abl kd',
'bcr abl kd mutation',
'bcr abl kd mutations',
'bcr abl transcript',
'bevacizumab',
'bic',
'binding activities',
'bioscope',
'blimp1',
'bmi',
'bone marrow cd34',
'bone marrow cd34 cells',
'bone sarcoma',
'bph',
```

```
'bps',
'braf',

'braf deletions',
'branch',
'branch figure',
'brca',
'brca 50',
'brca ovarian',
'brca ovarian cancer',
'brca sequence',
'brca testing',
'brca variants',
'brca vus',
'brca1',
'brca1 bard1',
'brca1 bard1 heterodimer',
'brca1 bard1 interaction',
'brca1 bclaf1',
'brca1 bclaf1 sf3b1',
'brca1 bclaf1 sf3b1 complex',
'brca1 brca2',
'brca1 brct',
'brca1 brct domain',
'brca1 cdna',
'brca1 defi',
'brca1 defi cient',
'brca1 e2',
'brca1 e2 interaction',
'brca1 mutation',
'brca1 null',
'brca1 protein',
'brca1 sco',
'brca1 sequence',
'brca1 sequence variants',
'brca1 ubiquitin',
'brca1 ubiquitin ligase',
'brca1 ubiquitin ligase activity',
'brca1 variant',
'brca1 variants',
'brca1 vus'
```

```
'brca1 vus',
'brca1 vuss',

'brca1fh',
'brca1fh i26a',
'brca1fh i26a fh',
'brca1fh i26a fh i26a',
'brca1flex2',
'brca1flex2 flex2',
'brca1s1598f',
'brca1s1598f s1598f',
'brca1sco',
'brca2',
'brca2 dna',
'brca2 dna binding',
'brct',
'brct dbd',
'brct domain',
'brct domains',
'brct phospho',
'breast ovarian',
'breast ovarian cancer',
'broad range human cancers',
'bxpc',
'c20r',
'c6',
'caax',
'calculated odds',
'cancer associated mtor',
'cancer res author',
'cancer res author manuscript',
'cancer research published',
'cancer research published onlinefirst',
'cancer risk',
'cancerdiscovery',
'cancerdiscovery aacrjournals',
'cancerdiscovery aacrjournals org',
'cancerdiscovery aacrjournals org may',
'caribbean',
'carm1',
```

```
'cases sf3b1',
'cassette exons',

'catalytic',
'catalytic activity',
'catalytic loops',
'catenin',
'cation',
'cation brca1',
'causality',
'cbl',
'ccnb3',
'ccyr',
'cd 13',
'cd 13 0094',
'cd34 cells',
'cd34 cells mds',
'cd34 cells mds patients',
'cd342ksl',
'cd36 cd71',
'cd71',
'cd74',
'cdh1',
'cdk12',
'cdk19',
'cdk4',
'cdk4 binding',
'cdk4 cdk6',
'cdk6',
'cdk8',
'cell aggregation',
'cell cell aggregation',
'cell cycle rna',
'cell lines sf3b1',
'cell lines sf3b1 knockdown',
'cells',
'cells mds',
'cells mds patients',
'cells mds patients sf3b1',
'cells sf3b1',
'cells sf3b1 knockdown',
```

```
'cells sf3b1 knockdown',
'cetuximab',

'cetuximab panitumumab',
'cetuximab resistant',
'cetuximab resistant cells',
'cfc',
'changes dna methylation',
'chimaerism',
'cial',
'cic',
'cic dux4',
'cient',
'cisplatin',
'cisplatin sensitivity',
'cisplatin sensitivity assay',
'class',
'classifi',
'classifi cation',
'classifi cation brca1',
'classifi ed',
'classification',
'classified',
'classified deleterious',
'classified neutral',
'classified unclassified',
'classify',
'cll',
'cll data',
'cll sf3b1',
'clustering applied',
'co occurrence',
'cohesin',
'combined llr',
'compared wild type u2af35',
'comparing sf3b1',
'comparing sf3b1 mutant',
'comparing sf3b1 mutant control',
'comparing sf3b1 mutant wild',
'comparison sf3b1',
'comparison sf3b1 mutant'
```

```
'comparison sf3b1 mutant',
'comparison sf3b1 mutant wild',

'comparisons sf3b1',
'comparisons sf3b1 mutant',
'comparisons sf3b1 mutant wild',
'complementation',
'complementation assay',
'complementation assays',
'components splicing',
'components splicing machinery',
'confi',
'conservation',
'containing prc2',
'containing prc2 complexes',
'control arm',
'core domain',
'core non core',
'core set probes',
'corresponding probability',
'cosegregation',
'craf',
'crebzf',
'creert2',
'creert2 rmce',
'crenolanib',
'crizotinib',
'crl4dcaf1',
'crnde',
'crnde ankhd1',
'cryptic ag',
'cryptic ss',
'ct tt',
'ctcf',
'ctcf binding',
'ctd',
'cul3',
'cycle rna',
'cyclin cdk8',
'cyclin d1',
```

```
'cyt',
'cyt 387',

'cytogenetic response',
'cytopenia',
'cytopenia multilineage',
'cytopenia multilineage dysplasia',
'cytopenia multilineage dysplasia ring',
'd1',
'd4z4',
'd781g',
'd92n',
'dacomitinib',
'dasatinib',
'dasatinib nilotinib',
'data analysis using',
'data analysis using dexseq',
'data set',
'daxx',
'daxx mutations',
'dcaf1',
'dcr',
'dd',
'dd related',
'defective ssa',
'defi',
'defi cient',
'degree relatives given',
'degree relatives given cancer',
'deleterious',
'deleterious brca1',
'deleterious compared',
'deleterious control',
'deleterious mutation',
'deleterious mutations',
'deleterious variants',
'dependent nadph consumption',
'depletion brca1',
'deptor',
'developed four',
```

```
'dexseq',
'df',

'di trimethylation',
'dicer1',
'different splicing',
'differential exon',
'differential exon usage',
'differential exon usage sf3b1',
'differential exon usage study',
'differential splicing',
'differential splicing events',
'differentially expressed exons',
'differentially expressed sf3b1',
'differentially spliced',
'difi',
'difi cells',
'dimethylated',
'dimethylated peptides',
'dimethylated peptides compared',
'diseases cancers',
'distinct expression profiles',
'dlcl2',
'dna binding',
'dna entry',
'dna methylation',
'dna methylation levels',
'dnmt',
'dnmt1',
'dnmt3b',
'doi 10 1158',
'doi 10 1158 2159',
'dovitinib',
'downloaded cancerdiscovery',
'downloaded cancerdiscovery aacrjournals',
'downloaded cancerdiscovery aacrjournals org',
'dox s34f',
'dox s34f dox',
'dox s34f dox s34f',
'dox wt dox',
```

```
'dr gfp',
'dsbh',

'dsred',
'dusp4',
'dux4',
'dysplasia ring',
'dysplasia ring sideroblasts',
'dysplasia ring sideroblasts rcmd',
'e17k',
'e2',
'e2 disruptive',
'e2 enzyme',
'e2 interaction',
'e255k',
'e3 ligase activity',
'e3 ligase activity brca1',
'ebc1',
'ebc1 cells',
'ec cells',
'ed',
'eed',
'eed suz12',
'effector mutants',
'effects sf3b1',
'effects sf3b1 knockdown',
'egfp marking',
'egfp p53',
'egfr',
'egfr cetuximab',
'egfr mutation',
'egfr mutations',
'egln',
'elf3',
'elongation',
'elongation factors',
'embryonic',
'embryonic stem',
'embryonic stem cells',
'empty rmce',
'empty rmce vector',
```

```
'empty rmce vector',
'emt',

'endogenous brca1',
'entrectinib',
'entry exit',
'enzyme ubch5a',
'epas1',
'erbb2',
'erbb4',
'erbb4 mutations',
'ercc2',
'ercc2 mutations',
'erg',
'erk',
'erlotinib',
'erythroid differentiation',
'estimated deleterious',
'estimated proportion',
'et al page cancer',
'ethnic',
'ethnicity',
'ets2',
'etv1',
'etv4',
'events crnde',
'ewing',
'exel',
'exel 7647',
'exo',
'exon',
'exon 20',
'exon array',
'exon array rna',
'exon array rna seq',
'exon junction',
'exon level',
'exon usage',
'exon usage sf3b1',
'exon usage sf3b1 mutant',
'exon usage study'
```

```
'exon usage study',
'expressed exons',

'expressed sf3b1',
'expression ezh2y641f',
'expression splicing',
'extended data',
'extended data fig',
'extensions',
'ezh1',
'ezh2',
'ezh2 mutant',
'ezh2 mutants',
'ezh2 protein',
'ezh2 wild',
'ezh2 wild type',
'ezh2 wt',
'ezh2 y641',
'ezh2wt',
'ezh2wt ezh2y641f',
'ezh2y641',
'ezh2y641 mutants',
'ezh2y641f',
'ezh2y641f variants',
'ezh2y641n',
'f3',
'f3 cells',
'f39a',
'f39l',
'f39v',
'fa',
'families',
'family',
'family histories',
'family history',
'family personal',
'fanca',
'fancl',
'fat1',
'favor',
'favor causality',
```

```
'favor causality',
'favor deleterious',

'fbw7',
'fedratinib',
'fedratinib binding',
'fedratinib binds',
'ferm',
'ferm sh2',
'fgfr',
'fgfr1',
'fgfr2',
'fgfr3',
'fh',
'fh i26a',
'fibre formation',
'fig lane',
'fig ros',
'figure figure histone',
'figure figure supplement',
'figure histone',
'figure right main',
'figure right main main',
'figure source',
'figure source data',
'figure supplement',
'flex2',
'flex7',
'flt1',
'flt3',
'flt3 itd',
'flt3 s451f',
'flt3 s451f y572c',
'flt3 s451f y572c v592g',
'fniii',
'fniii repeats',
'foretinib',
'forms ezh2',
'four cell lines sf3b1',
'four myeloid',
```

```
'four myeloid cell',
'four myeloid cell lines',

'frequency histogram',
'frequency histogram combined',
'frequency histogram combined llr',
'ftmt',
'full sequence',
'function pten',
'functional',
'functional analysis brca1',
'functional assays',
'functional assessment',
'functional assessment pten',
'functional assessment pten mutations',
'functional classifi',
'functional classifi cation',
'functional classifi cation brca1',
'functional complementation',
'functional effect',
'fusion',
'fusions',
'g101w',
'g1035s',
'g12d p53flex7',
'g12d p53flex7 flex7',
'g161v',
'g1770v',
'g34',
'g34r',
'g34r g34v',
'g34v',
'g831e',
'g9a',
'gab1',
'gas8',
'gbm',
'gefitinib',
'gene fusions',
'gene targets myelodysplasia',
'genes sf3b1'
```

```
'genes sf3b1',
'genes sf3b1 srsf2',

'genes sf3b1 srsf2 u2af1',
'genes showing differential',
'genes showing differential exon',
'genetic laboratories',
'genetics 2006',
'genetics 2006 vol',
'genetics 2006 vol 15',
'genetics counseling',
'genomatix',
'genome exon',
'genome exon junction',
'genome regions',
'genosplice',
'germline phts',
'gfp akt1',
'gist',
'gists',
'given cancer',
'given cancer type',
'global dna',
'global dna methylation',
'global h3k27me3',
'global h3k27me3 levels',
'golga4',
'group ring finger',
'gusbp1',
'gusbp11',
'gusbp11 uqcc',
'gvgd',
'h2286',
'h2405',
'h2ak119',
'h3',
'h3 21',
'h3 21 44',
'h3 mutations',
'h3f3a',
'h3f3a atrx'
```

```
'h3f3a atrx',
'h3f3a idh1',

'h3f3a mutation',
'h3f3a mutations',
'h3f3a tp53',
'h3k27',
'h3k27 methylation',
'h3k27 methylation states',
'h3k27 methylation status',
'h3k27 trimethylation',
'h3k27me0',
'h3k27me1',
'h3k27me2',
'h3k27me3',
'h3k27me3 figure',
'h3k27me3 levels',
'h93r',
'harbor missense',
'harboring sf3b1',
'harboring sf3b1 mutations',
'hbg1',
'hbg1 klf1',
'hbrca1',
'hbs',
'hca',
'hd hd',
'hd hd hd',
'hd hd hd hd',
'hdr',
'hdr assay',
'hec',
'hela ssa',
'hela tf',
'hela tf cells',
'heme',
'heme biosynthesis',
'her2',
'her2 l726f',
'heterogeneity analysis',
```

```
'ng',
'hif',

'hif2',
'histogram combined',
'histogram combined llr',
'histone',
'histone h2b',
'histone h3',
'histone h3k27',
'histopathology',
'histories',
'history',
'history tested',
'hla',
'hmg domain',
'hnf4',
'hnrnpd',
'homologous recombination',
'homologous recombination assay',
'hotspots',
'hr',
'hr activity',
'hspc',
'hspc mds',
'hspc mds patients',
'human alk',
'human alkf1174i',
'human alkf1174l',
'human brca1',
'human brca1 cdna',
'human genome exon',
'human genome exon junction',
'human molecular',
'human molecular genetics',
'human molecular genetics 2006',
'human sense',
'hydroxyglutarate',
'i26a',
'i26a fh',
'i26a fh i26a'
```

```
'i26a fn i26a',
'iarc',

'ic50',
'icf',
'identification bcor',
'idh',
'idh mutant',
'idh mutation',
'idh mutations',
'idh1',
'idh1 idh2',
'idh1 idh2 mutations',
'idh1 mutation',
'idh1 mutations',
'idh1 r132c',
'idh1 r132h',
'idh1 r132h knock',
'idh1 r132h knock mice',
'idh1r132h',
'idh2',
'idh2 mutation',
'idh2 mutations',
'idh2 r140q',
'idh2 r172k',
'idh3',
'ifdi',
'ifdis',
'imatinib',
'imatinib mesylate',
'incorrectly classified',
'increase h3k27me3',
'increase h3k27me3 levels',
'increased h3k27me3',
'increased ring',
'increased ring sideroblasts',
'independent histone',
'independent histone variant',
'independent histone variant h3',
'individual probands',
'individuals',
```

```
'individuals',
'induced activation akt',

'ineffective haematopoiesis',
'inhibit e2',
'inhibitors',
'ins',
'ins p4',
'interaction bard1',
'interaction e2',
'intron retention abcc5',
'iron',
'irx2',
'isocitrate',
'itd',
'ivs11',
'ivs15',
'ivs18',
'ivs19',
'ivs2',
'ivs6',
'jak1',
'jak2',
'jak2 inhibitors',
'jak2 kinase',
'jak2 v617f',
'jak2 v617f y931c',
'japan laboratory',
'japanese men',
'jm',
'jm domain',
'jnj42756493',
'july 18',
'july 18 2013',
'july 18 2013 doi',
'june 15',
'k179m',
'k27',
'k27m',
'k27m g34r',
```

```
'k27m g34r g34v',
'k36',

'k562',
'k562 cells',
'k562 hel',
'k562 tf1',
'k666',
'k666 k700',
'k666t',
'k700e',
'karpas 422',
'karpas 422 su',
'kconfab',
'kd',
'kd mutation',
'kd mutations',
'kdm2b',
'kdm5a',
'kdm5c',
'keap1',
'ketoglutarate',
'ketoglutarate dependent',
'ketoglutarate dependent nadph',
'ketoglutarate dependent nadph consumption',
'kinase',
'kinase domain',
'kit',
'kit exon',
'kit exon 11',
'kit mutations',
'klf1',
'kmt2d',
'known deleterious',
'known deleterious mutation',
'known deleterious mutations',
'kras',
'kraslsl',
'kraslsl g12d',
'kraslsl g12d p53flex7',
'kraslsl g12d p53flex7 flex7'
```

```
 'krasısı gızd p53flex7 flex7',
 'ku tokyo',

 'ku tokyo 108',
 'ku tokyo 108 8639',
 'l1407p',
 'l726f',
 'l858r',
 'l983f',
 'lapatinib',
 'lapatinib resistance',
 'lch',
 'leiomyoma',
 'leiomyoma linked',
 'length change',
 'length changes',
 'lengthening',
 'lengthening telomeres',
 'leu 983',
 'levels h3k27me1',
 'levels h3k27me3',
 ...]
```

In [41]:

```
# building a BoW vectorizer - Unigram
bow_uni_vectorizer = CountVectorizer(min_df=3)
bow_uni_feature_onehotCoding = bow_uni_vectorizer.fit_transform(train_d
f['TEXT'])

# getting all the feature names (words)
bow_uni_train_text_features= bow_uni_vectorizer.get_feature_names()

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

# zip(list(text_features),text_fea_counts) will zip a word with its num
ber of times it occured
bow_uni_text_fea_dict = dict(zip(list(bow_uni_train_text_features),bow_
```

```
uni_train_text_fea_counts))

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

```
Total number of unique words in train data (BOW_UNIGram) : 53647
```

In [42]:

```
# building a BoW vectorizer - BiGram
bow_bi_vectorizer = CountVectorizer(ngram_range=(1,2), min_df=3, max_fe
atures=2000)
bow_bi_feature_onehotCoding = bow_bi_vectorizer.fit_transform(train_df[
'TEXT'])

# getting all the feature names (words)
bow_bi_train_text_features= bow_bi_vectorizer.get_feature_names()

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

# zip(list(text_features),text_fea_counts) will zip a word with its num
ber of times it occured
bow_bi_text_fea_dict = dict(zip(list(bow_bi_train_text_features),bow_bi
_train_text_fea_counts))

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

```
Total number of unique words in train data (BOW_UNIGram) : 2000
```

In [43]:

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

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


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

In [44]:

```
#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 [45]:

```
# 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_t
ext_feature_responseCoding.sum(axis=1)).T
```

In [46]:

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

# we use the same vectorizer that was trained on train data
test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEX
T'])
test_text_feature_onehotCoding = k_best_clf.transform(test_text_feature
_onehotCoding)
# don't forget to normalize every feature
```

```
test_text_feature_onehotCoding = normalize(test_text_feature_onehotCodi
ng, axis=0)

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

In [47]:

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

# we use the same vectorizer that was trained on train data
bow_uni_test_text_feature_onehotCoding = bow_uni_vectorizer.transform(t
est_df['TEXT'])
# don't forget to normalize every feature
bow_uni_test_text_feature_onehotCoding = normalize(bow_uni_test_text_fe
ature_onehotCoding, axis=0)

# we use the same vectorizer that was trained on train data
bow_uni_cv_text_feature_onehotCoding = bow_uni_vectorizer.transform(cv_
df['TEXT'])
# don't forget to normalize every feature
bow_uni_cv_text_feature_onehotCoding = normalize(bow_uni_cv_text_featur
e_onehotCoding, axis=0)
```

In [48]:

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

# we use the same vectorizer that was trained on train data
bow_bi_test_text_feature_onehotCoding = bow_bi_vectorizer.transform(tes
t_df['TEXT'])
# don't forget to normalize every feature
bow_bi_test_text_feature_onehotCoding = normalize(bow_bi_test_text_feat
```

```
ure_onehotCoding, axis=0)

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

In [49]:
```
#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()))
```

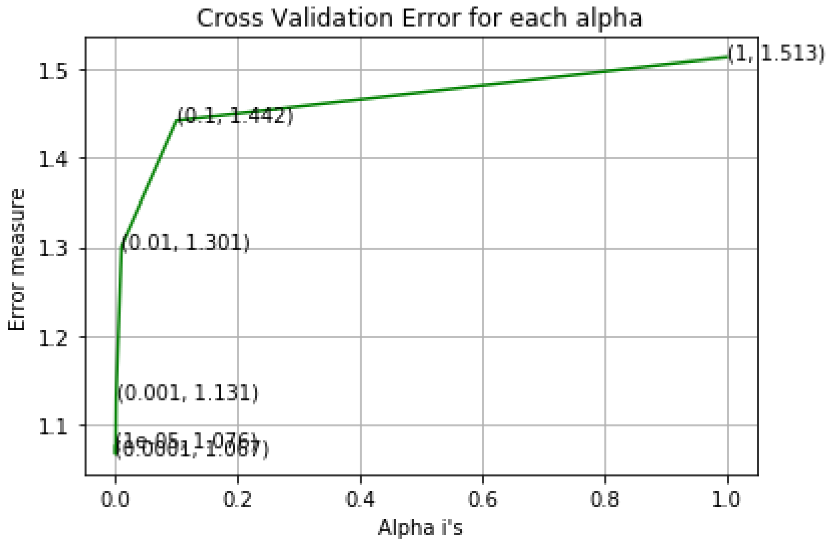
In [50]:
```
# Number of words for a given frequency.
# print(Counter(sorted_text_occur))
```

In [51]:
```
# Train a Logistic regression+Calibration model using text features whi
cha re on-hot encoded
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

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

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


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

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_text_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.clas
ses_, 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_arra
y[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


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

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

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

```
For values of alpha =  1e-05 The log loss is: 1.0755348531503681
For values of alpha =  0.0001 The log loss is: 1.0665037348144368
For values of alpha =  0.001 The log loss is: 1.1310406208415502
For values of alpha =  0.01 The log loss is: 1.3007967019181428
For values of alpha =  0.1 The log loss is: 1.4422595807441734
For values of alpha =  1 The log loss is: 1.513449245986045
```




```
For values of best alpha =  0.0001 The train log loss is: 0.89960205562
75224
For values of best alpha =  0.0001 The cross validation log loss is: 1.
0665037348144368
For values of best alpha =  0.0001 The test log loss is: 1.149083198738
726
```

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

Ans. Yes, it seems like!

In [52]:
```
def get_intersec_text(df):
    df_text_vec = CountVectorizer(min_df=3)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

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

In [53]:
```
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 appe
ared in train data")
```

```
1.968 % of word of test data appeared in train data
2.031 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [54]:
```
# Global Dataframe for storing the report performance of each model tha
t we try
result_report = pd.DataFrame(columns=["Vectorizer", "N-Gram", "Model",
"TRAIN-Score", "CV-Score", "TEST-Score",
                                      "Misclassification-Rate"])
```

In [55]:
```
#Data preparation for ML models.

#Misc. functionns for ML models


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

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

In [56]:
```
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 [57]:
```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text
 or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3)
```

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

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

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

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

Stacking the three types of features

In [58]:
```
# merging gene, variance and text features

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

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding))

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

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

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


# Bow Uni Gram Stack
bow_uni_train_x_onehotCoding = hstack((train_gene_var_onehotCoding, bow_uni_train_text_feature_onehotCoding)).tocsr()
bow_uni_test_x_onehotCoding = hstack((test_gene_var_onehotCoding, bow_uni_test_text_feature_onehotCoding)).tocsr()
bow_uni_cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, bow_uni_cv_text_feature_onehotCoding)).tocsr()

# Bow Bi Gram Stack
```

```
bow_bi_train_x_onehotCoding = hstack((train_gene_var_onehotCoding, bow_
bi_train_text_feature_onehotCoding)).tocsr()
bow_bi_test_x_onehotCoding = hstack((test_gene_var_onehotCoding, bow_bi
_test_text_feature_onehotCoding)).tocsr()
bow_bi_cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, bow_bi_cv_
text_feature_onehotCoding)).tocsr()


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseC
oding,train_variation_feature_responseCoding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCod
ing,test_variation_feature_responseCoding))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,
cv_variation_feature_responseCoding))

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

In [59]:

```
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 = ", t
est_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, 41
89)
(number of data points * number of features) in test data =  (665, 418
9)
(number of data points * number of features) in cross validation data =
(532, 4189)
```

In [60]:

```
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 [61]:

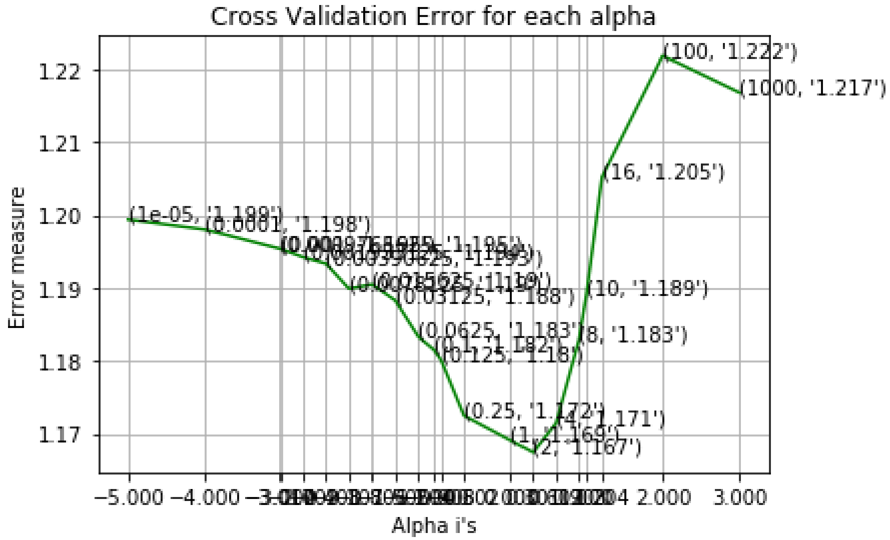
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# 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.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='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 = sorted([0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000] + [2**i f
or i in range(-10, -1)] + [2**i for i in range(1, 5)])
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log
-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))
```

```
fig, ax = plt.subplots()
ax.plot(np.log10(alpha), cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv_log_error_a
rray[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)
train_log_loss = log_loss(y_train, predict_y, labels=clf.classes_, eps=
1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The train log
 loss is:", train_log_loss)
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
cv_log_loss = log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:", cv_log_loss)
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
test_log_loss = log_loss(y_test, predict_y, labels=clf.classes_, eps=1e
-15)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:", test_log_loss)
```

```
for alpha = 1e-05
Log Loss : 1.1994015674711274
for alpha = 0.0001
Log Loss : 1.1980480316313027
for alpha = 0.0009765625
```

```
Log Loss : 1.1953930488735316
for alpha = 0.001
Log Loss : 1.195351580256511
for alpha = 0.001953125
Log Loss : 1.1942422087286404
for alpha = 0.00390625
Log Loss : 1.1933312405319654
for alpha = 0.0078125
Log Loss : 1.1899017559659248
for alpha = 0.015625
Log Loss : 1.1904745395012664
for alpha = 0.03125
Log Loss : 1.1882774093874096
for alpha = 0.0625
Log Loss : 1.1833334565343268
for alpha = 0.1
Log Loss : 1.1815771992392425
for alpha = 0.125
Log Loss : 1.1801394018408375
for alpha = 0.25
Log Loss : 1.1724896115297985
for alpha = 1
Log Loss : 1.1691285110671148
for alpha = 2
Log Loss : 1.1674376025956645
for alpha = 4
Log Loss : 1.171484939977182
for alpha = 8
Log Loss : 1.1829850379189886
for alpha = 10
Log Loss : 1.1891401616781554
for alpha = 16
Log Loss : 1.2051937444582421
for alpha = 100
Log Loss : 1.2218401677131494
for alpha = 1000
Log Loss : 1.216813068676463
```




```
For values of best alpha =  2 The train log loss is: 0.9600649644062621
For values of best alpha =  2 The cross validation log loss is: 1.16743
76025956645
For values of best alpha =  2 The test log loss is: 1.187300697967915
```

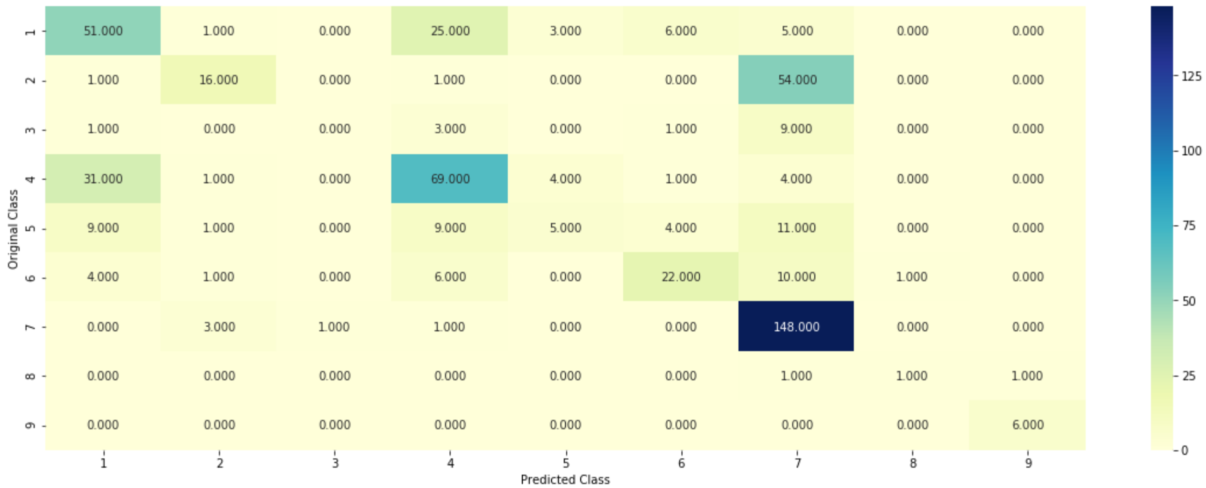
4.1.1.2. Testing the model with best hyper paramters

In [62]:
```
# find more about Multinomial Naive base function here http://scikit-le
arn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_pr
ior=None)
```

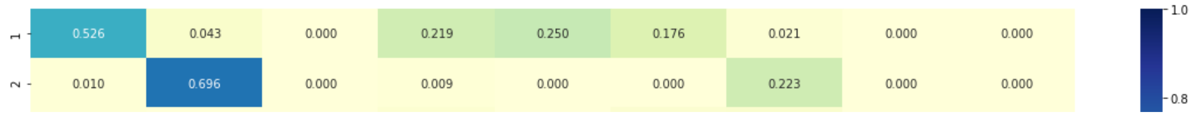
```
# some of methods of MultinomialNB()
# fit(X, y[, sample_weight])    Fit Naive Bayes classifier according to
 X, y
# predict(X)    Perform classification on an array of test vectors X.
# predict_log_proba(X)  Return log-probability estimates for the test v
ector X.
# -----------------------
# video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


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

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

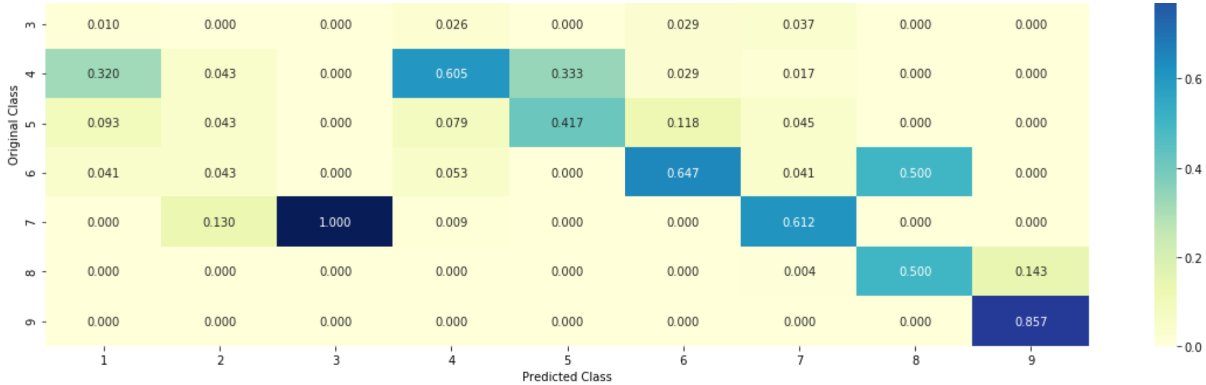
```
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray
()))

result_report = result_report.append({"Vectorizer": "TF-IDF", "N-Gram":
 "(1,4)","Model": "Naive Bayes",
                                      "TRAIN-Score": np.round(train_log
_loss, 4),
                                      "CV-Score": np.round(cv_log_loss,
 4),
                                      "TEST-Score": np.round(test_log_l
oss, 4),
                                      "Misclassification-Rate": '{}%'.f
ormat(np.round(misc_rate * 100, 2))
                                     }, ignore_index=True)
```

```
Log Loss : 1.1674376025956645
Number of missclassified point : 0.40225563909774437
-------------------- Confusion matrix --------------------
```

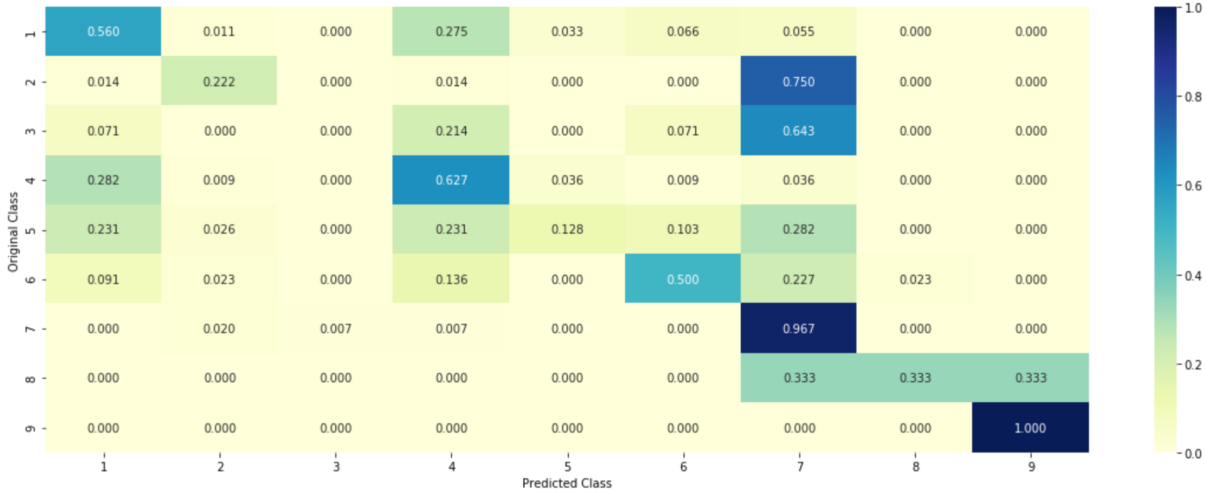



```
-------------------- Precision matrix (Columm Sum=1) -------------------
--
```





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



4.1.1.3. Feature Importance, Correctly classified point

In [63]:

```
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.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 : 1
Predicted Class Probabilities: [[0.7718 0.0417 0.0188 0.0545 0.041  0.0
346 0.0306 0.0034 0.0037]]
Actual Class : 1
--------------------------------------------------
12 Text feature [197] present in test data point [True]
33 Text feature [015] present in test data point [True]
Out of the top  100  features  2 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [64]:

```
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: [[0.0339 0.2699 0.0203 0.0505 0.0443 0.0
363 0.5371 0.0037 0.004 ]]
Actual Class : 7
--------------------------------------------------
64 Text feature [13] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [65]:

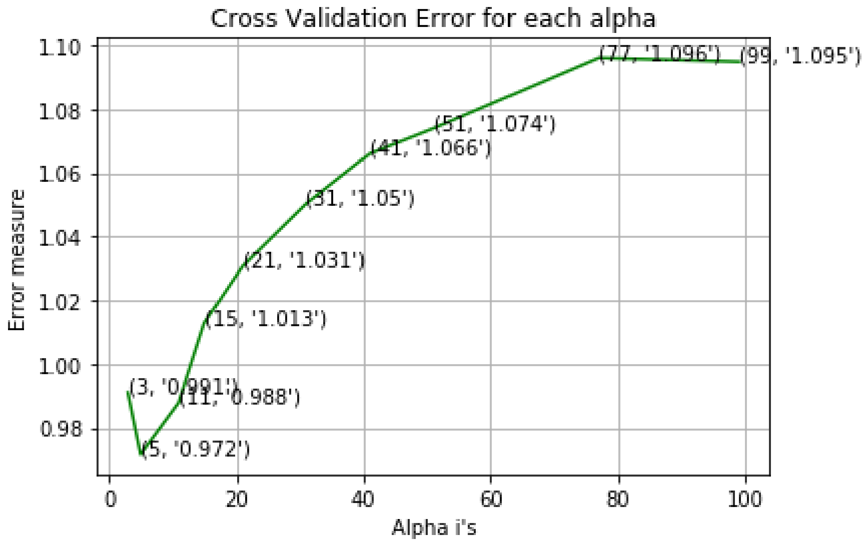
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/s
table/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='aut
o', leaf_size=30, p=2,
# metric='minkowski',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

# methods of
# fit(X, y) : Fit the model using X as training data and y as target va
lues
# 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-examp
le-1/
#--------------------------------------
```

```
# find more about CalibratedClassifierCV here at http://scikit-learn.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='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 = [3, 5, 11, 15, 21, 31, 41, 51, 77, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_responseCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_responseCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log
-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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


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

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

```
for alpha = 3
Log Loss : 0.9910384413696869
for alpha = 5
Log Loss : 0.9715522274465977
for alpha = 11
Log Loss : 0.9878236315160382
for alpha = 15
Log Loss : 1.012790296941515
for alpha = 21
Log Loss : 1.0306150315917233
for alpha = 31
Log Loss : 1.0502470155966943
for alpha = 41
Log Loss : 1.0660151754805027
```

```
for alpha = 51
Log Loss : 1.0739033948311145
for alpha = 77
Log Loss : 1.0960738912696464
for alpha = 99
Log Loss : 1.0948851895210256
```



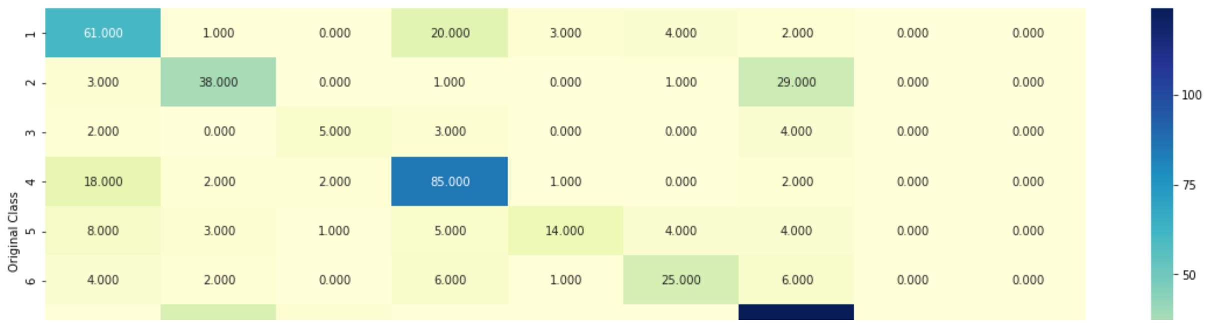

```
For values of best alpha =  5 The train log loss is: 0.5155565698315947
For values of best alpha =  5 The cross validation log loss is: 0.97155
22274465977
For values of best alpha =  5 The test log loss is: 1.0482431438085504
```

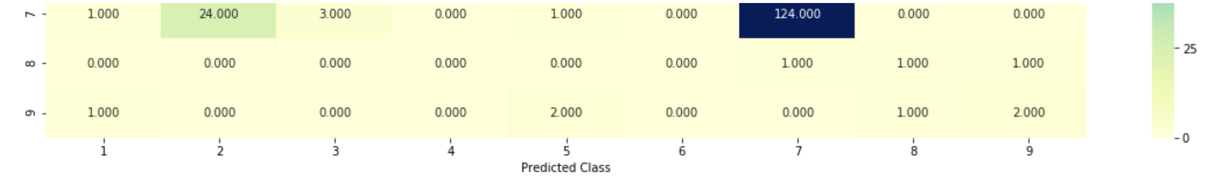
4.2.2. Testing the model with best hyper paramters

In [66]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/s
table/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='aut
o', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)
```

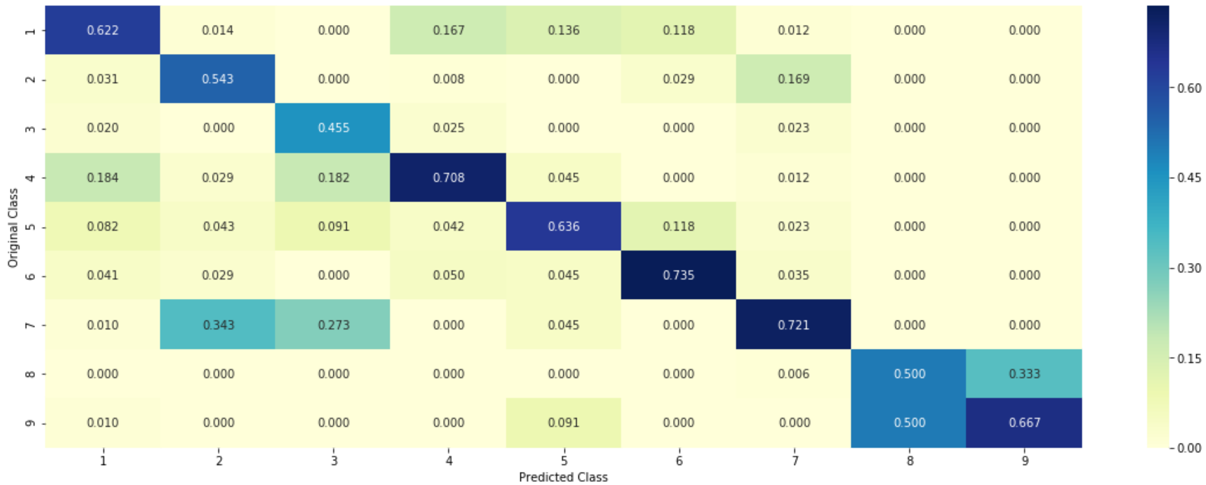
```
# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-neighbo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

result_report = result_report.append({"Vectorizer": "TF-IDF", "N-Gram": "(1,4)","Model": "K-NN",
                                      "TRAIN-Score": np.round(train_log_loss, 4),
                                      "CV-Score": np.round(cv_log_loss, 4),
                                      "TEST-Score": np.round(test_log_loss, 4),
                                      "Misclassification-Rate": '{}%'.format(np.round(misc_rate * 100, 2))
                                     }, ignore_index=True)
```

```
Log loss : 0.9715522274465977
Number of mis-classified points : 0.33270676691729323
-------------------- Confusion matrix --------------------
```





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



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

4.2.3.Sample Query point -1

In [67]:

```
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].resh
ape(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("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 1
Actual Class : 1
The  5  nearest neighbours of the test points belongs to classes [1 1 1
1 1]
Fequency of nearest points : Counter({1: 5})
```

4.2.4. Sample Query Point-2

In [68]:
```
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].resh
ape(1, -1), alpha[best_alpha])
print("the k value for knn is",alpha[best_alpha],"and the nearest neigh
bours of the test points belongs to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 7
the k value for knn is 5 and the nearest neighbours of the test points
belongs to classes [7 7 7 7 2]
Fequency of nearest points : Counter({7: 4, 2: 1})
```

4.3. Logistic Regression

4.3.1 TF-IDF Vectorization

4.3.1.1 With Class balancing

4.3.1.1.1 Hyper paramter tuning

In [69]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
```

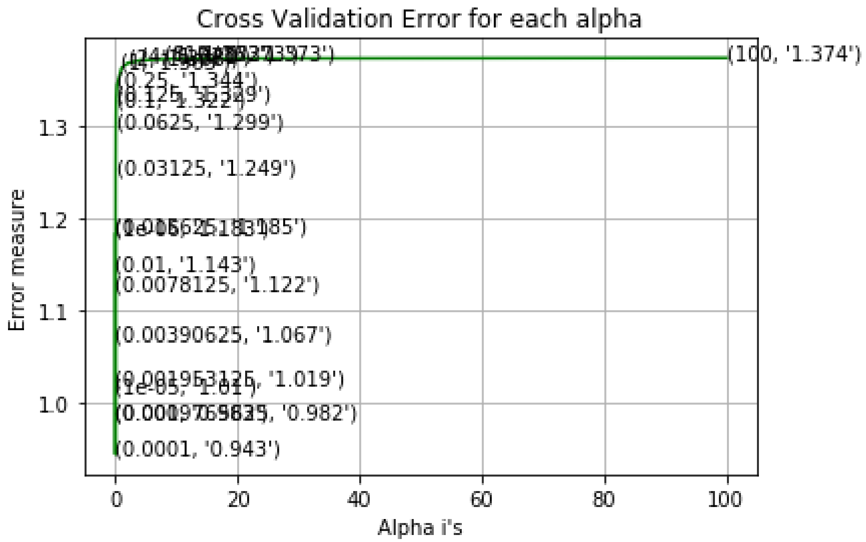
```
# -------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



# find more about CalibratedClassifierCV here at http://scikit-learn.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='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 = sorted([10 ** x for x in range(-6, 3)] + [2**i for i in range(-
10, -1)] + [2**i for i in range(1, 5)])
cv_log_error_array = []
```

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

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='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)
train_log_loss = log_loss(y_train, predict_y, labels=clf.classes_, eps=
1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The train log
 loss is:", train_log_loss)
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
cv_log_loss = log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15)
```

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

```
for alpha = 1e-06
Log Loss : 1.1825440604603867
for alpha = 1e-05
Log Loss : 1.0101024161749477
for alpha = 0.0001
Log Loss : 0.9428619624645861
for alpha = 0.0009765625
Log Loss : 0.9818224990821358
for alpha = 0.001
Log Loss : 0.9830915887856224
for alpha = 0.001953125
Log Loss : 1.019462198839168
for alpha = 0.00390625
Log Loss : 1.0666804720781435
for alpha = 0.0078125
Log Loss : 1.121909898296406
for alpha = 0.01
Log Loss : 1.1434379154991374
for alpha = 0.015625
Log Loss : 1.1853632492084827
for alpha = 0.03125
Log Loss : 1.2493578210742788
for alpha = 0.0625
Log Loss : 1.2994380943800872
for alpha = 0.1
Log Loss : 1.3217795611699972
for alpha = 0.125
Log Loss : 1.3293440363304394
for alpha = 0.25
Log Loss : 1.3444277845627342
for alpha = 1
Log Loss : 1.3631987141922242
```

```
for alpha = 2
Log Loss : 1.3681306654914702
for alpha = 4
Log Loss : 1.371027711424412

for alpha = 8
Log Loss : 1.3721776008041706
for alpha = 10
Log Loss : 1.3726909422497264
for alpha = 16
Log Loss : 1.3729561759443836
for alpha = 100
Log Loss : 1.373601637424621
```

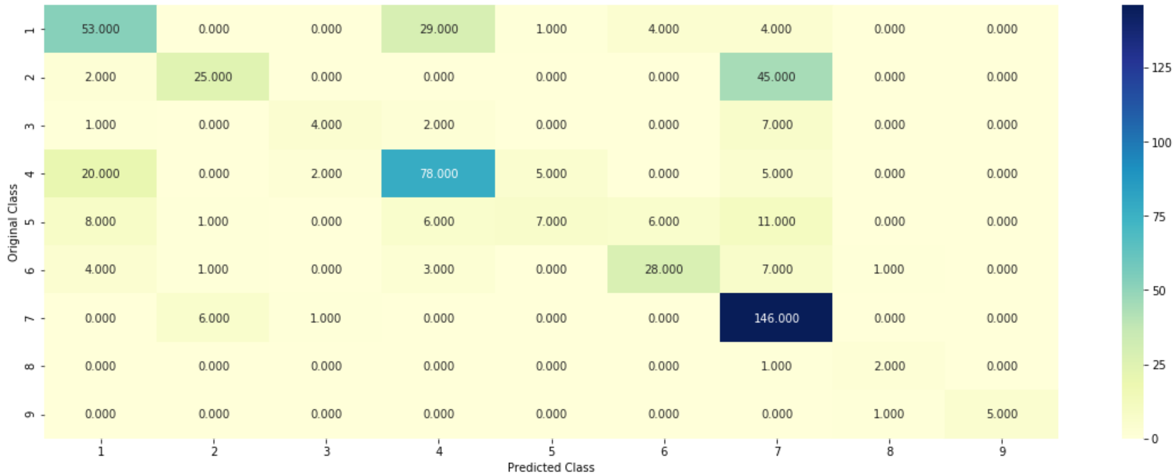



```
For values of best alpha =  0.0001 The train log loss is: 0.48071542081
08645
For values of best alpha =  0.0001 The cross validation log loss is: 0.
9428619624645861
For values of best alpha =  0.0001 The test log loss is: 0.991705315156
9499
```

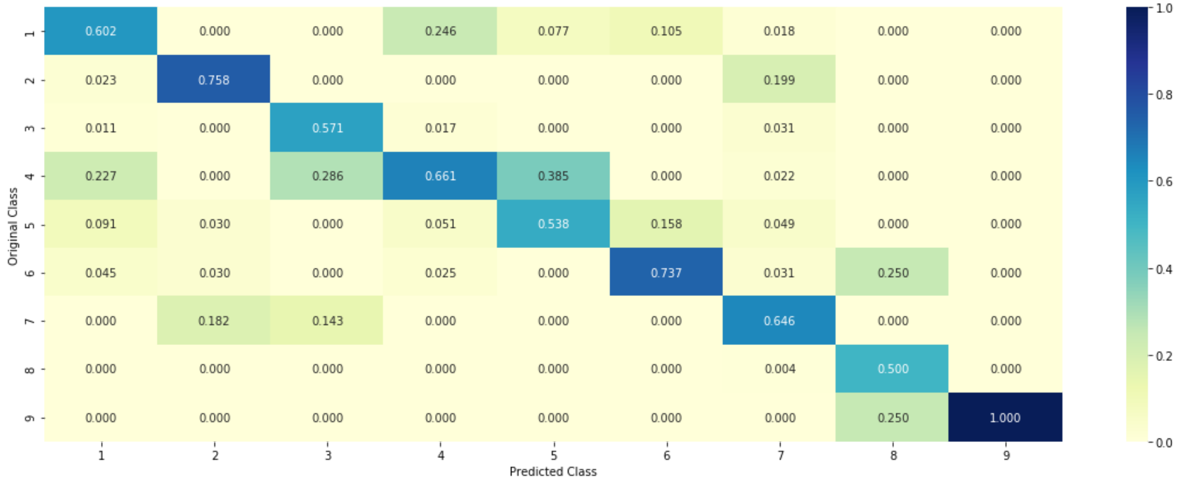
4.3.1.1.2. Testing the model with best hyper paramters

In [70]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

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

result_report = result_report.append({"Vectorizer": "TF-IDF", "N-Gram":
 "(1,4)","Model": "Logistic-Regression (With Class Balanced)",
                                      "TRAIN-Score": np.round(train_log
_loss, 4),
                                      "CV-Score": np.round(cv_log_loss,
 4),
                                      "TEST-Score": np.round(test_log_l
oss, 4),
                                      "Misclassification-Rate": '{}%'.f
ormat(np.round(misc_rate * 100, 2))
                                     }, ignore_index=True)
```

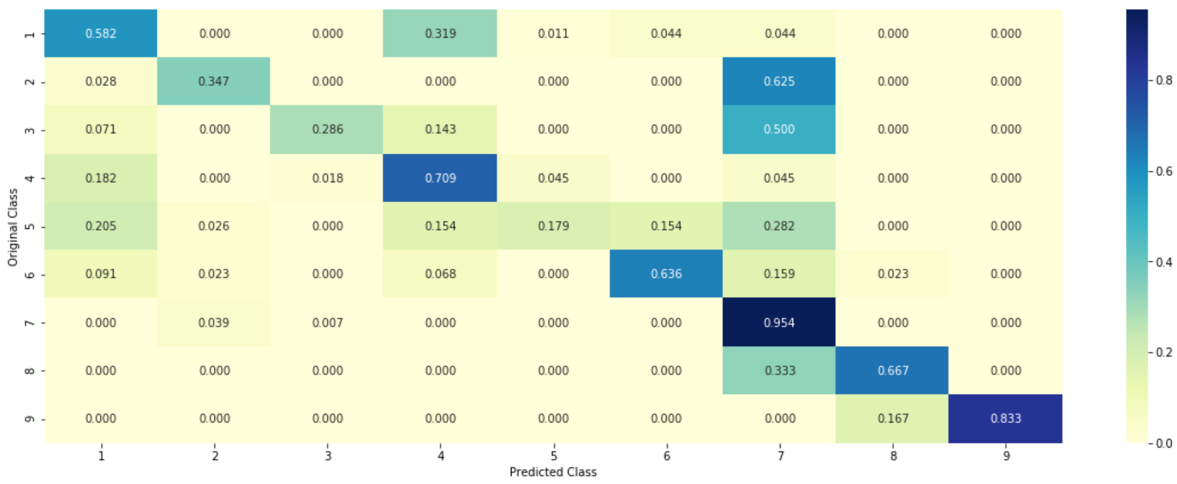
```
Log loss : 0.9428619624645861
Number of mis-classified points : 0.3458646616541353
-------------------- Confusion matrix --------------------
```




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




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



4.3.1.1.3. Feature Importance

In [71]:
```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"
])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features
[i], yes_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our que
ry point")
```

```
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[
0]," class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Pre
sent or Not']))
```

4.3.1.1.3.1. Correctly Classified point

In [72]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='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(-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 : 1
Predicted Class Probabilities: [[0.9029 0.0412 0.0052 0.0117 0.0081 0.0
143 0.0078 0.0035 0.0054]]
Actual Class : 1
--------------------------------------------------
91 Text feature [152] present in test data point [True]
135 Text feature [115] present in test data point [True]
199 Text feature [005] present in test data point [True]
263 Text feature [169] present in test data point [True]
324 Text feature [197] present in test data point [True]
346 Text feature [198] present in test data point [True]
430 Text feature [178] present in test data point [True]
445 Text feature [025] present in test data point [True]
```

```
490 Text feature [101] present in test data point [True]
Out of the top  500  features  9 are present in query point
```

4.3.1.1.3.2. Incorrectly Classified point

In [73]:
```
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(-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: [[0.0199 0.4048 0.0067 0.033  0.0158 0.0
122 0.4964 0.0047 0.0065]]
Actual Class : 7
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.3.1.2. Without Class balancing

4.3.1.2.1. Hyper paramter tuning

In [74]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
```

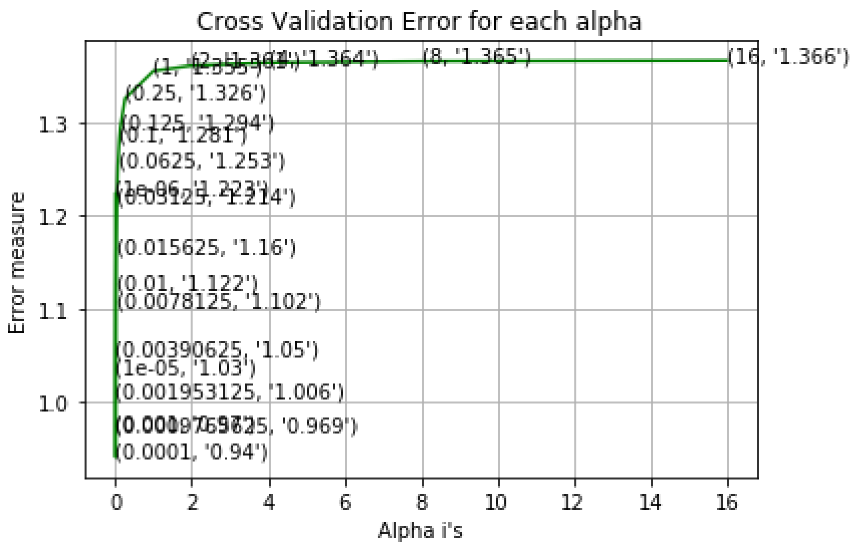
```
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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




# find more about CalibratedClassifierCV here at http://scikit-learn.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='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 = sorted([10 ** x for x in range(-6, 1)] + [2**i for i in range(-
10, -1)] + [2**i for i in range(1, 5)])
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)
train_log_loss = log_loss(y_train, predict_y, labels=clf.classes_, eps=
1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The train log
 loss is:", train_log_loss)
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
cv_log_loss = log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:", cv_log_loss)
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
test_log_loss = log_loss(y_test, predict_y, labels=clf.classes_, eps=1e
```

```
-15)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:", test_log_loss)
```

```
for alpha = 1e-06
Log Loss : 1.2230747187941327
for alpha = 1e-05
Log Loss : 1.0303364902224625
for alpha = 0.0001
Log Loss : 0.9402389827203065
for alpha = 0.0009765625
Log Loss : 0.9691979804482447
for alpha = 0.001
Log Loss : 0.97023830700268
for alpha = 0.001953125
Log Loss : 1.0055329243543287
for alpha = 0.00390625
Log Loss : 1.0498630011520438
for alpha = 0.0078125
Log Loss : 1.1021765024847643
for alpha = 0.01
Log Loss : 1.1219920297431638
for alpha = 0.015625
Log Loss : 1.159602590237377
for alpha = 0.03125
Log Loss : 1.2139813698240507
for alpha = 0.0625
Log Loss : 1.2532643209603098
for alpha = 0.1
Log Loss : 1.280774245468689
for alpha = 0.125
Log Loss : 1.2936551216850933
for alpha = 0.25
Log Loss : 1.3256687535857086
for alpha = 1
Log Loss : 1.3553332510256602
for alpha = 2
Log Loss : 1.3608508222295963
for alpha = 4
Log Loss : 1.3637464233524046
```

```
for alpha = 8
Log Loss : 1.3652453210157056
for alpha = 16
Log Loss : 1.366028748998041
```




```
For values of best alpha =  0.0001 The train log loss is: 0.46051158286
91
For values of best alpha =  0.0001 The cross validation log loss is: 0.
9402389827203065
For values of best alpha =  0.0001 The test log loss is: 0.994301958824
442
```

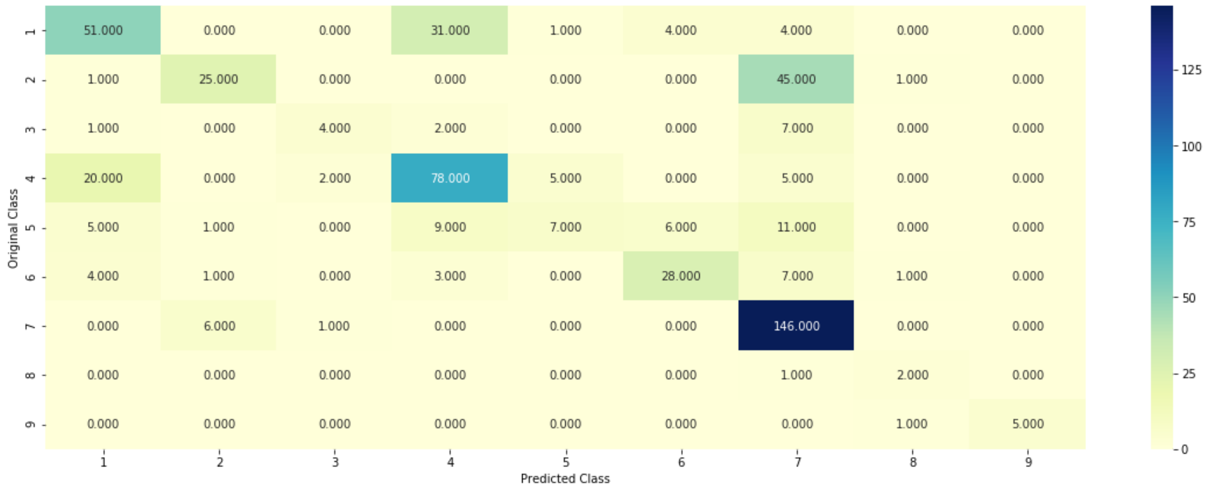
4.3.1.2.2. Testing model with best hyper parameters

In [75]:

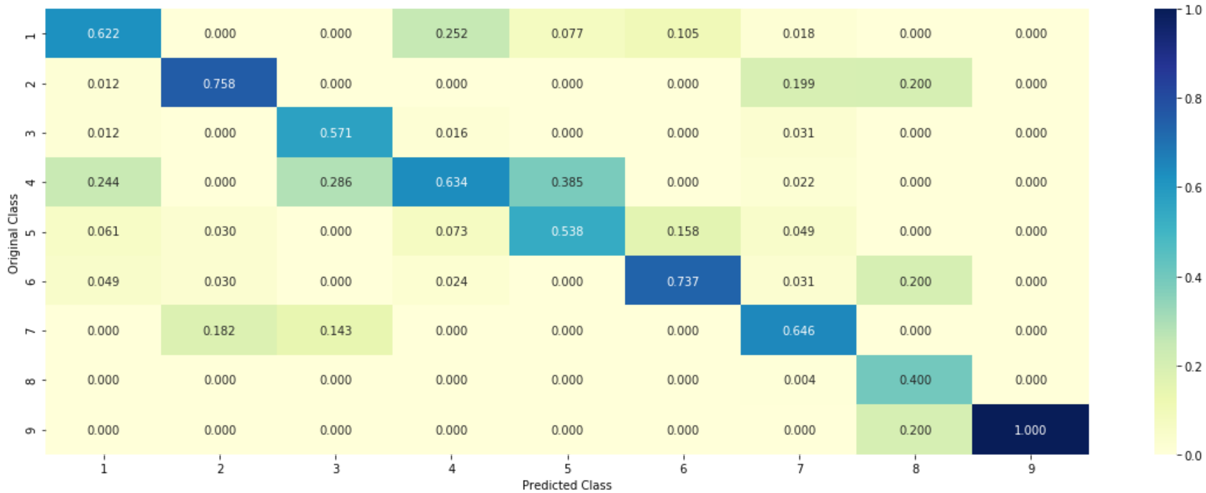
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
```

```
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

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

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
random_state=42)
log_loss_val, misc_rate = predict_and_plot_confusion_matrix(train_x_one
hotCoding, train_y, cv_x_onehotCoding, cv_y, clf)

result_report = result_report.append({"Vectorizer": "TF-IDF", "N-Gram":
 "(1,4)",
                                      "Model": "Logistic-Regression (Wi
thout Class Balanced)",
                                      "TRAIN-Score": np.round(train_log
_loss, 4),
                                      "CV-Score": np.round(cv_log_loss,
 4),
                                      "TEST-Score": np.round(test_log_l
oss, 4),
                                      "Misclassification-Rate": '{}%'.f
ormat(np.round(misc_rate * 100, 2))
                                     }, ignore_index=True)
```

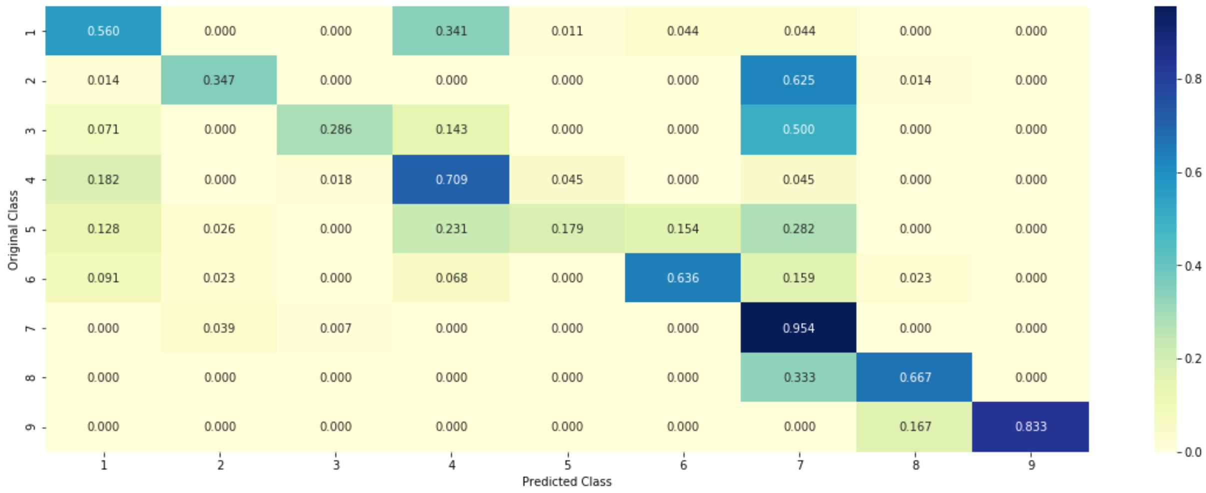
```
Log loss : 0.9402389827203065
Number of mis-classified points : 0.34962406015037595
-------------------- Confusion matrix --------------------
```



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



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



4.3.1.2.3. Feature Importance, Correctly Classified point

In [76]:
```
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(-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 : 1
Predicted Class Probabilities: [[0.9026 0.0451 0.0044 0.0125 0.0083 0.0
14  0.0082 0.0017 0.0032]]
Actual Class : 1
--------------------------------------------------
95 Text feature [152] present in test data point [True]
151 Text feature [115] present in test data point [True]
224 Text feature [005] present in test data point [True]
277 Text feature [169] present in test data point [True]
326 Text feature [198] present in test data point [True]
353 Text feature [197] present in test data point [True]
447 Text feature [178] present in test data point [True]
450 Text feature [025] present in test data point [True]
481 Text feature [101] present in test data point [True]
Out of the top  500  features  9 are present in query point
```

4.3.1.2.4. Feature Importance, Inorrectly Classified point

In [77]:

```
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(-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: [[0.0214 0.4185 0.0055 0.0376 0.0155 0.0
125 0.4818 0.0031 0.0042]]
Actual Class : 7
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.3.2. BoW Vectorizer

4.3.2.1. UniGram

4.3.2.1.1 With Class balancing

4.3.2.1.1.1 Hyper paramter tuning

In [78]:

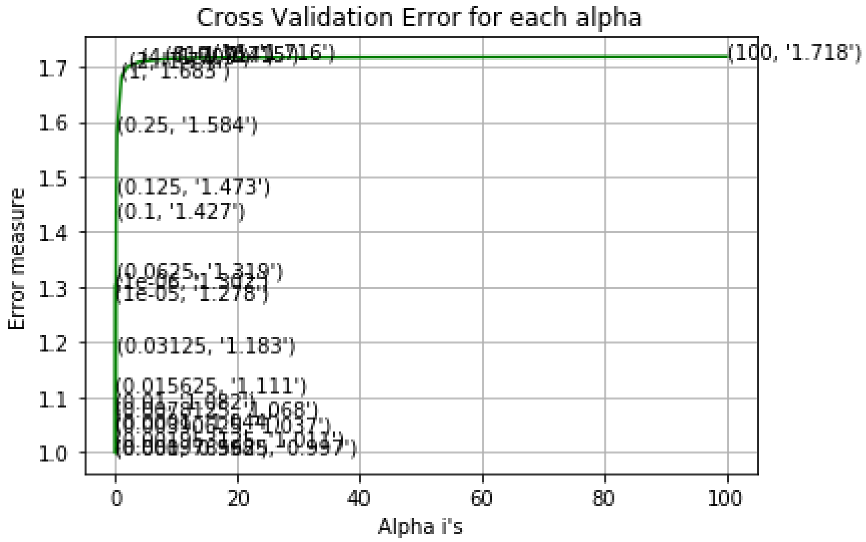
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='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 = sorted([10 ** x for x in range(-6, 3)] + [2**i for i in range(-
10, -1)] + [2**i for i in range(1, 5)])
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(bow_uni_train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(bow_uni_train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(bow_uni_cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log
-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
```

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


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

predict_y = sig_clf.predict_proba(bow_uni_train_x_onehotCoding)
train_log_loss = log_loss(y_train, predict_y, labels=clf.classes_, eps=
1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The train log
 loss is:", train_log_loss)
predict_y = sig_clf.predict_proba(bow_uni_cv_x_onehotCoding)
cv_log_loss = log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:", cv_log_loss)
predict_y = sig_clf.predict_proba(bow_uni_test_x_onehotCoding)
test_log_loss = log_loss(y_test, predict_y, labels=clf.classes_, eps=1e
-15)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:", test_log_loss)
```

```
for alpha = 1e-06
Log Loss : 1.3022266143615377
for alpha = 1e-05
Log Loss : 1.278498073993367
for alpha = 0.0001
Log Loss : 1.0438731444467095
for alpha = 0.0009765625
Log Loss : 0.9973320186639967
for alpha = 0.001
```

```
Log Loss : 0.9979953888731257
for alpha = 0.001953125
Log Loss : 1.0111266550792306
for alpha = 0.00390625
Log Loss : 1.0369976364680977
for alpha = 0.0078125
Log Loss : 1.0684149081586314
for alpha = 0.01
Log Loss : 1.08238027037548
for alpha = 0.015625
Log Loss : 1.110765475097346
for alpha = 0.03125
Log Loss : 1.1833421623385976
for alpha = 0.0625
Log Loss : 1.3194387157865268
for alpha = 0.1
Log Loss : 1.4266873019852564
for alpha = 0.125
Log Loss : 1.4732166565852025
for alpha = 0.25
Log Loss : 1.5836643033492537
for alpha = 1
Log Loss : 1.6829025017701078
for alpha = 2
Log Loss : 1.7003783416324636
for alpha = 4
Log Loss : 1.7091906467812692
for alpha = 8
Log Loss : 1.7136686628619708
for alpha = 10
Log Loss : 1.714582444778357
for alpha = 16
Log Loss : 1.7159598233845796
for alpha = 100
Log Loss : 1.717945188962714
```




```
For values of best alpha =  0.0009765625 The train log loss is: 0.60387
60494277301
For values of best alpha =  0.0009765625 The cross validation log loss
is: 0.9973320186639967
For values of best alpha =  0.0009765625 The test log loss is: 1.076473
0430206366
```

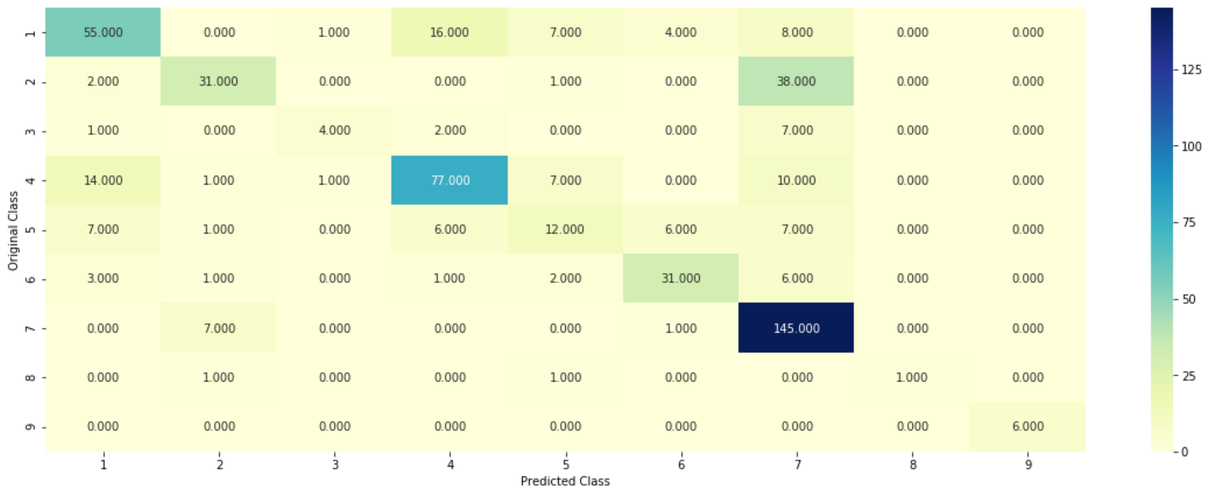
4.3.2.1.1.2. Testing the model with best hyper paramters

In [79]:

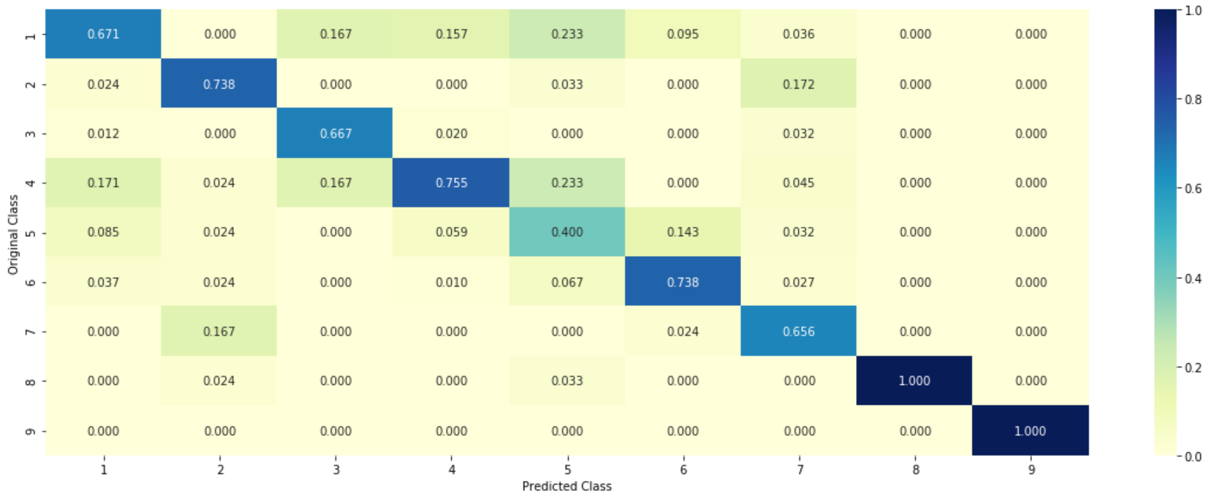
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/geometric-intuition-1/
#------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
log_loss_val, misc_rate = predict_and_plot_confusion_matrix(bow_uni_tra
in_x_onehotCoding, train_y, bow_uni_cv_x_onehotCoding, cv_y, clf)

result_report = result_report.append({"Vectorizer": "BoW", "N-Gram": "
(1,1)","Model": "Logistic-Regression (With Class Balanced)",
                                      "TRAIN-Score": np.round(train_log
_loss, 4),
                                      "CV-Score": np.round(cv_log_loss,
 4),
                                      "TEST-Score": np.round(test_log_l
oss, 4),
                                      "Misclassification-Rate": '{}%'.f
ormat(np.round(misc_rate * 100, 2))
                                     }, ignore_index=True)
```

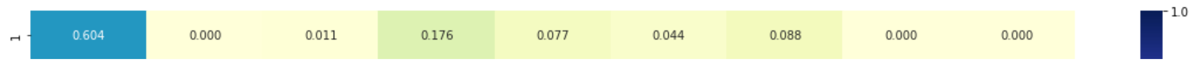
```
Log loss : 0.9973320186639967
Number of mis-classified points : 0.31954887218045114
-------------------- Confusion matrix --------------------
```



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



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



4.3.2.1.2 Without Class balancing

4.3.2.1.2.1 Hyper paramter tuning

In [80]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
```

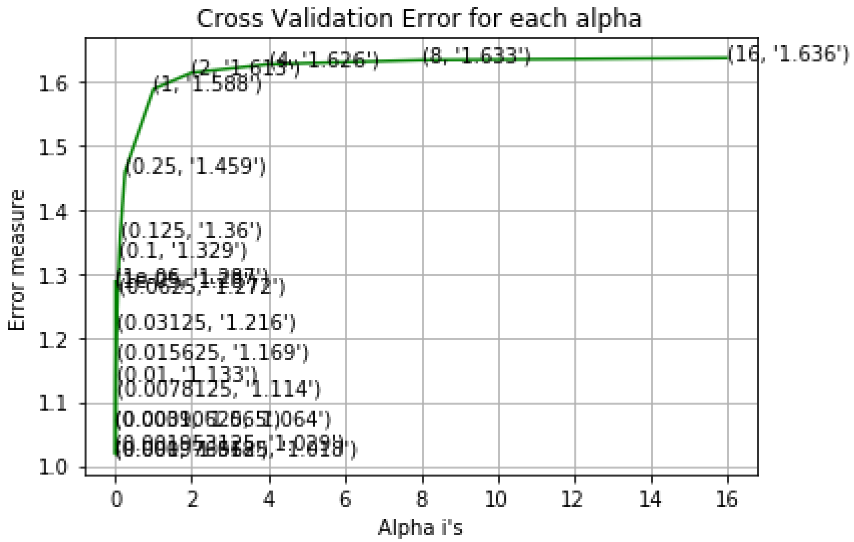
```
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



# find more about CalibratedClassifierCV here at http://scikit-learn.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='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 = sorted([10 ** x for x in range(-6, 1)] + [2**i for i in range(-
10, -1)] + [2**i for i in range(1, 5)])
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(bow_uni_train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(bow_uni_train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(bow_uni_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(bow_uni_train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(bow_uni_train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(bow_uni_train_x_onehotCoding)
train_log_loss = log_loss(y_train, predict_y, labels=clf.classes_, eps=
1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The train log
 loss is:", train_log_loss)
predict_y = sig_clf.predict_proba(bow_uni_cv_x_onehotCoding)
cv_log_loss = log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:", cv_log_loss)
predict_y = sig_clf.predict_proba(bow_uni_test_x_onehotCoding)
```

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

```
for alpha = 1e-06

Log Loss : 1.287370869774238
for alpha = 1e-05
Log Loss : 1.2804995509271229
for alpha = 0.0001
Log Loss : 1.0651089842466672
for alpha = 0.0009765625
Log Loss : 1.017812960223579
for alpha = 0.001
Log Loss : 1.0181969827549977
for alpha = 0.001953125
Log Loss : 1.0293244345446282
for alpha = 0.00390625
Log Loss : 1.0641074536703818
for alpha = 0.0078125
Log Loss : 1.1136110530848202
for alpha = 0.01
Log Loss : 1.1333824646413213
for alpha = 0.015625
Log Loss : 1.1687275665016181
for alpha = 0.03125
Log Loss : 1.2160062017712794
for alpha = 0.0625
Log Loss : 1.2718749490600672
for alpha = 0.1
Log Loss : 1.328971413822722
for alpha = 0.125
Log Loss : 1.3597694200909296
for alpha = 0.25
Log Loss : 1.4585595662432644
for alpha = 1
Log Loss : 1.587547386448657
for alpha = 2
Log Loss : 1.6130206492117263
```

```
for alpha = 4
Log Loss : 1.626131668688131
for alpha = 8
Log Loss : 1.632773359157538
for alpha = 16
Log Loss : 1.6361105323161127
```




```
For values of best alpha =  0.0009765625 The train log loss is: 0.60189
00973229439
For values of best alpha =  0.0009765625 The cross validation log loss
is: 1.017812960223579
For values of best alpha =  0.0009765625 The test log loss is: 1.096453
6915415894
```

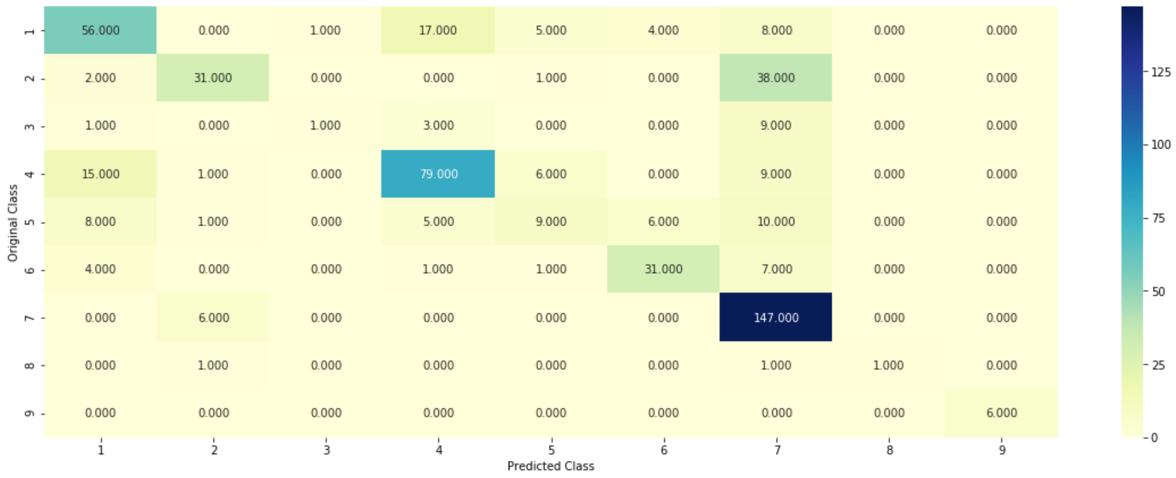
4.3.2.1.2.2. Testing the model with best hyper paramters

In [81]:

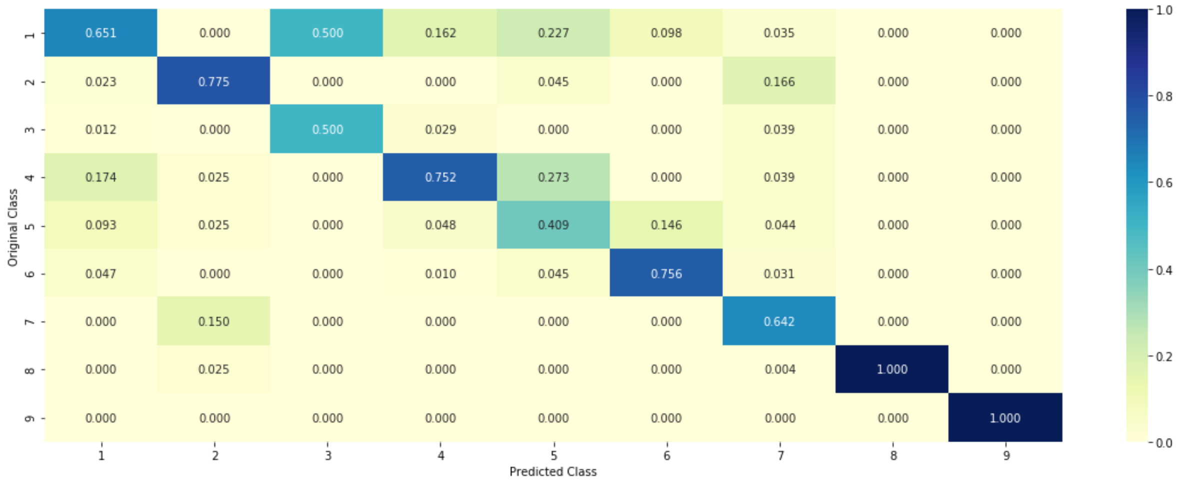
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
```

```
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

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

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
random_state=42)
log_loss_val, misc_rate = predict_and_plot_confusion_matrix(bow_uni_tra
in_x_onehotCoding, train_y, bow_uni_cv_x_onehotCoding, cv_y, clf)

result_report = result_report.append({"Vectorizer": "BoW", "N-Gram": "
(1,1)",
                                      "Model": "Logistic-Regression (Wi
thout Class Balanced)",
                                      "TRAIN-Score": np.round(train_log
_loss, 4),
                                      "CV-Score": np.round(cv_log_loss,
 4),
                                      "TEST-Score": np.round(test_log_l
oss, 4),
                                      "Misclassification-Rate": '{}%'.f
ormat(np.round(misc_rate * 100, 2))
                                     }, ignore_index=True)
```

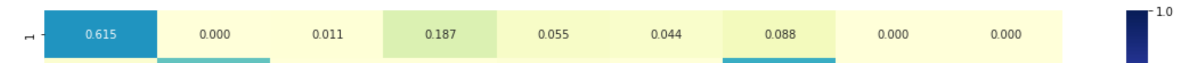
```
Log loss : 1.017812960223579
Number of mis-classified points : 0.32142857142857145
-------------------- Confusion matrix --------------------
```

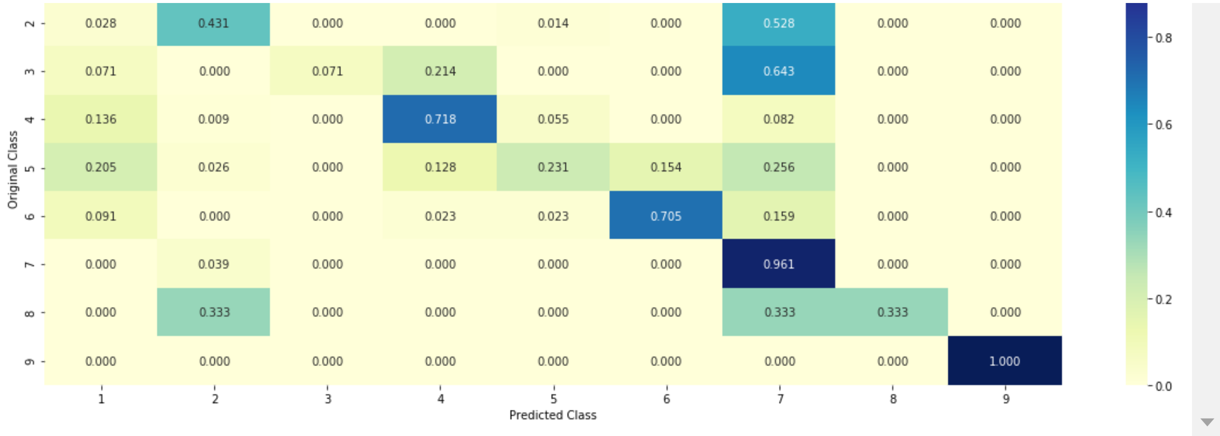


------------------- Precision matrix (Columm Sum=1) ----------------
----



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





4.3.2.2. BiGram

4.3.2.2.1 With Class balancing

4.3.2.2.1.1 Hyper paramter tuning

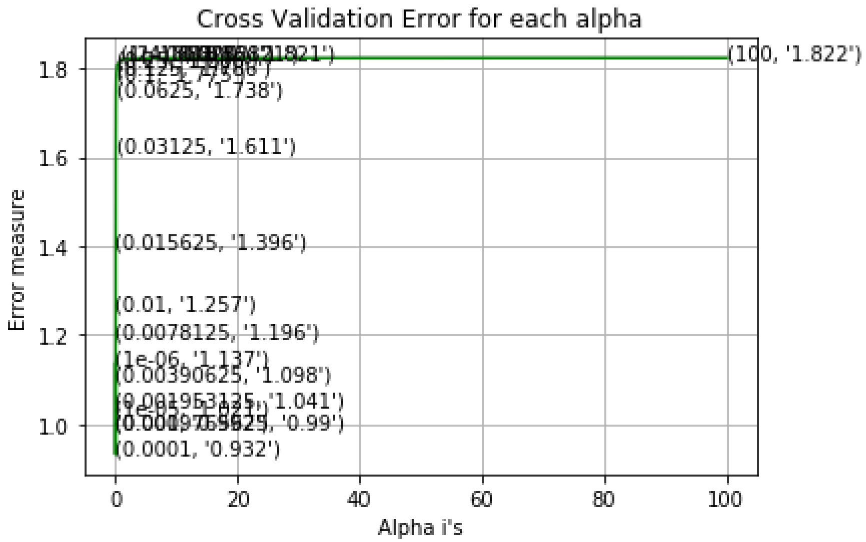
In [82]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=’hinge’, penalty=’l2’, alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.
```

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


# find more about CalibratedClassifierCV here at http://scikit-learn.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='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 = sorted([10 ** x for x in range(-6, 3)] + [2**i for i in range(-
10, -1)] + [2**i for i in range(1, 5)])
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(bow_bi_train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(bow_bi_train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(bow_bi_cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log
-probability estimates
```

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

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


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

predict_y = sig_clf.predict_proba(bow_bi_train_x_onehotCoding)
train_log_loss = log_loss(y_train, predict_y, labels=clf.classes_, eps=
1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The train log
 loss is:", train_log_loss)
predict_y = sig_clf.predict_proba(bow_bi_cv_x_onehotCoding)
cv_log_loss = log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:", cv_log_loss)
predict_y = sig_clf.predict_proba(bow_bi_test_x_onehotCoding)
test_log_loss = log_loss(y_test, predict_y, labels=clf.classes_, eps=1e
-15)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:", test_log_loss)
```

```
for alpha = 1e-06
Log Loss : 1.1365696959490967
for alpha = 1e-05
Log Loss : 1.0213732146310819
for alpha = 0.0001
```

```
Log Loss : 0.9323537293551791
for alpha = 0.0009765625
Log Loss : 0.9903295444097312
for alpha = 0.001
Log Loss : 0.9917903102100303
for alpha = 0.001953125
Log Loss : 1.041495920247892
for alpha = 0.00390625
Log Loss : 1.0979037310777926
for alpha = 0.0078125
Log Loss : 1.1956601217910707
for alpha = 0.01
Log Loss : 1.2568521587572115
for alpha = 0.015625
Log Loss : 1.3964060135705623
for alpha = 0.03125
Log Loss : 1.6110943747498632
for alpha = 0.0625
Log Loss : 1.7376839877143462
for alpha = 0.1
Log Loss : 1.775378563896482
for alpha = 0.125
Log Loss : 1.7862769719402076
for alpha = 0.25
Log Loss : 1.805753027257846
for alpha = 1
Log Loss : 1.8180288848057673
for alpha = 2
Log Loss : 1.8198544043301084
for alpha = 4
Log Loss : 1.8207432901447593
for alpha = 8
Log Loss : 1.8211828712310905
for alpha = 10
Log Loss : 1.8212704904306722
for alpha = 16
Log Loss : 1.821401611647578
for alpha = 100
Log Loss : 1.821583661327231
```




```
For values of best alpha =  0.0001 The train log loss is: 0.45321046157
96889
For values of best alpha =  0.0001 The cross validation log loss is: 0.
9323537293551791
For values of best alpha =  0.0001 The test log loss is: 1.009323477505
1811
```

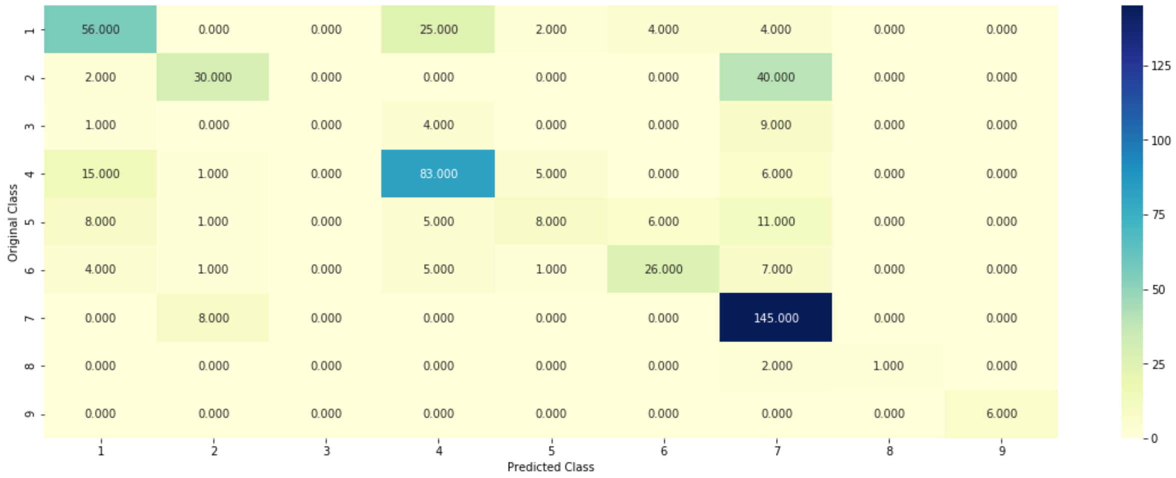
4.3.2.2.1.2. Testing the model with best hyper paramters

In [83]:

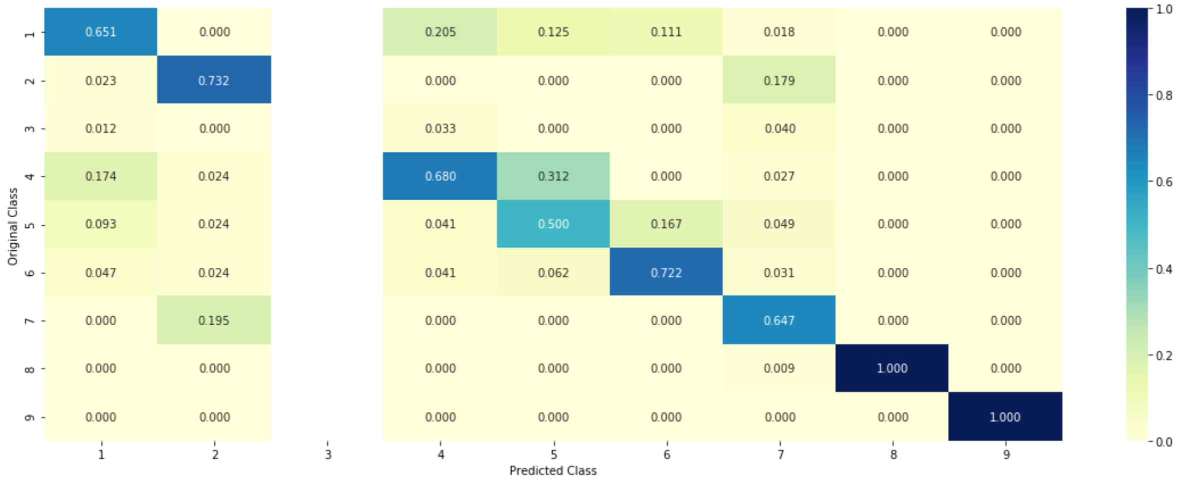
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/geometric-intuition-1/
#------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
log_loss_val, misc_rate = predict_and_plot_confusion_matrix(bow_bi_trai
n_x_onehotCoding, train_y, bow_bi_cv_x_onehotCoding, cv_y, clf)

result_report = result_report.append({"Vectorizer": "BoW", "N-Gram": "
(1,2)","Model": "Logistic-Regression (With Class Balanced)",
                                      "TRAIN-Score": np.round(train_log
_loss, 4),
                                      "CV-Score": np.round(cv_log_loss,
 4),
                                      "TEST-Score": np.round(test_log_l
oss, 4),
                                      "Misclassification-Rate": '{}%'.f
ormat(np.round(misc_rate * 100, 2))
                                     }, ignore_index=True)
```

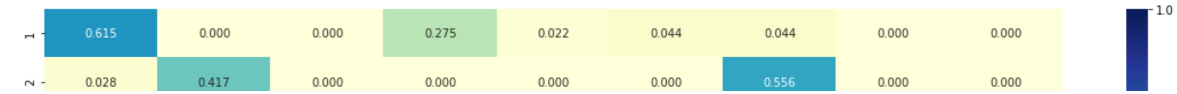
```
Log loss : 0.9323537293551791
Number of mis-classified points : 0.33270676691729323
-------------------- Confusion matrix --------------------
```

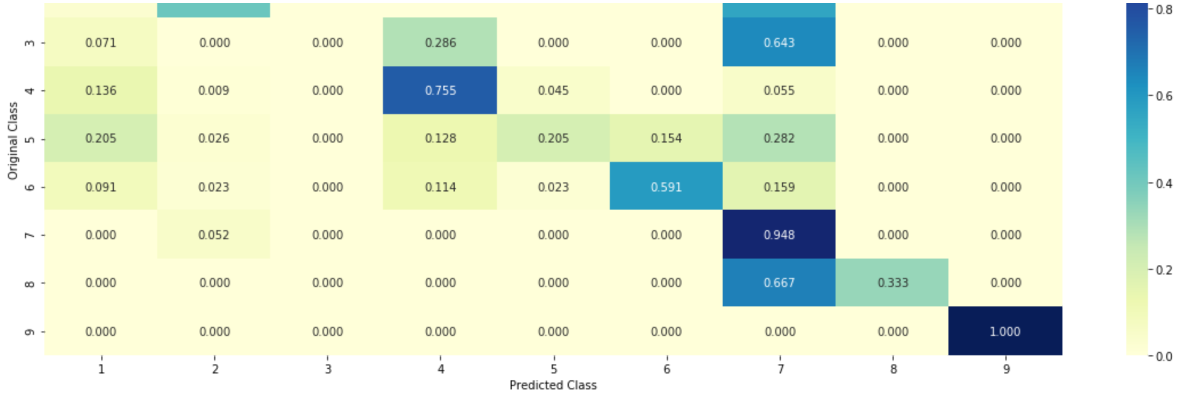


------------------- Precision matrix (Columm Sum=1) ---------------
----



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





4.3.2.2.2 Without Class balancing

4.3.2.2.2.1 Hyper paramter tuning

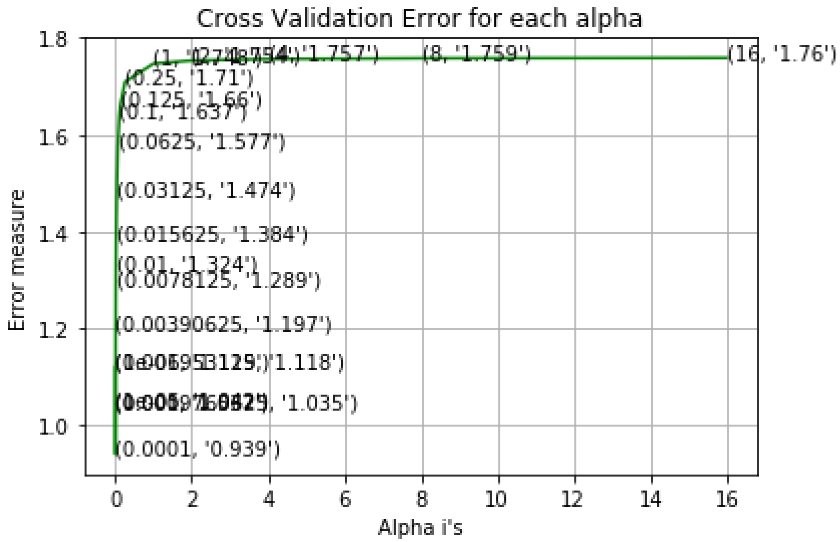
In [84]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.
```

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




# find more about CalibratedClassifierCV here at http://scikit-learn.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='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 = sorted([10 ** x for x in range(-6, 1)] + [2**i for i in range(-
10, -1)] + [2**i for i in range(1, 5)])
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(bow_bi_train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(bow_bi_train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(bow_bi_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))
```

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

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

```
for alpha = 1e-06
Log Loss : 1.1193331722923379
for alpha = 1e-05
Log Loss : 1.0415216342716227
for alpha = 0.0001
Log Loss : 0.9394269693583398
for alpha = 0.0009765625
```

```
Log Loss : 1.0346848030072133
for alpha = 0.001
Log Loss : 1.0371090287790425
for alpha = 0.001953125
Log Loss : 1.1182473674328473
for alpha = 0.00390625

Log Loss : 1.1973310653942206
for alpha = 0.0078125
Log Loss : 1.2890066625692354
for alpha = 0.01
Log Loss : 1.3239775359002488
for alpha = 0.015625
Log Loss : 1.3842877649519114
for alpha = 0.03125
Log Loss : 1.4736935503482935
for alpha = 0.0625
Log Loss : 1.577235177728219
for alpha = 0.1
Log Loss : 1.637326771079047
for alpha = 0.125
Log Loss : 1.6602994277373282
for alpha = 0.25
Log Loss : 1.709654153732341
for alpha = 1
Log Loss : 1.747753861617497
for alpha = 2
Log Loss : 1.754029028514288
for alpha = 4
Log Loss : 1.7571667722630784
for alpha = 8
Log Loss : 1.7587244012975467
for alpha = 16
Log Loss : 1.7595011227616122
```



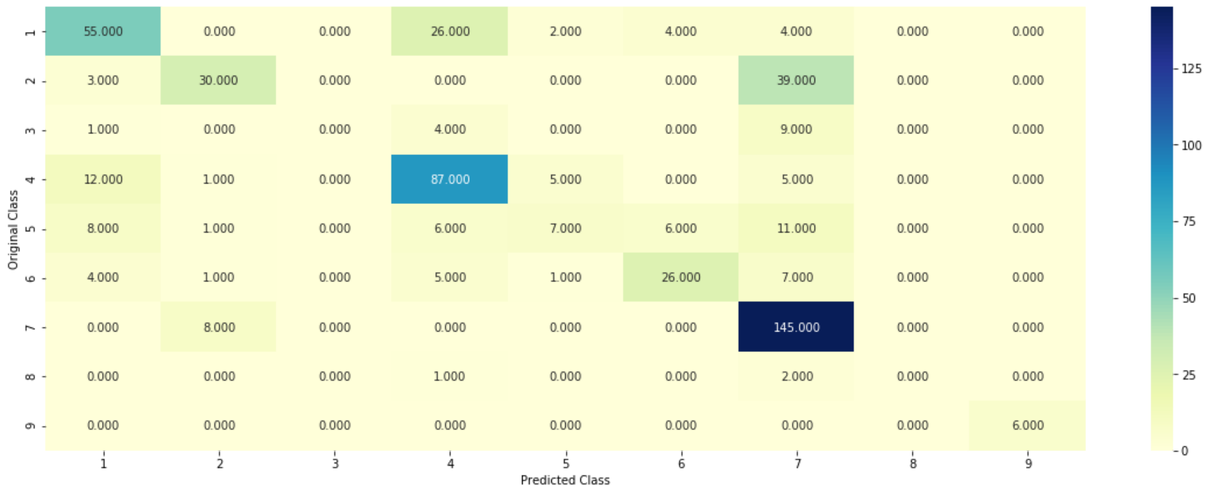

```
For values of best alpha =  0.0001 The train log loss is: 0.44700206170
06088
For values of best alpha =  0.0001 The cross validation log loss is: 0.
9394269693583398
For values of best alpha =  0.0001 The test log loss is: 1.014673003383
2696
```

4.3.2.2.2.2. Testing the model with best hyper paramters

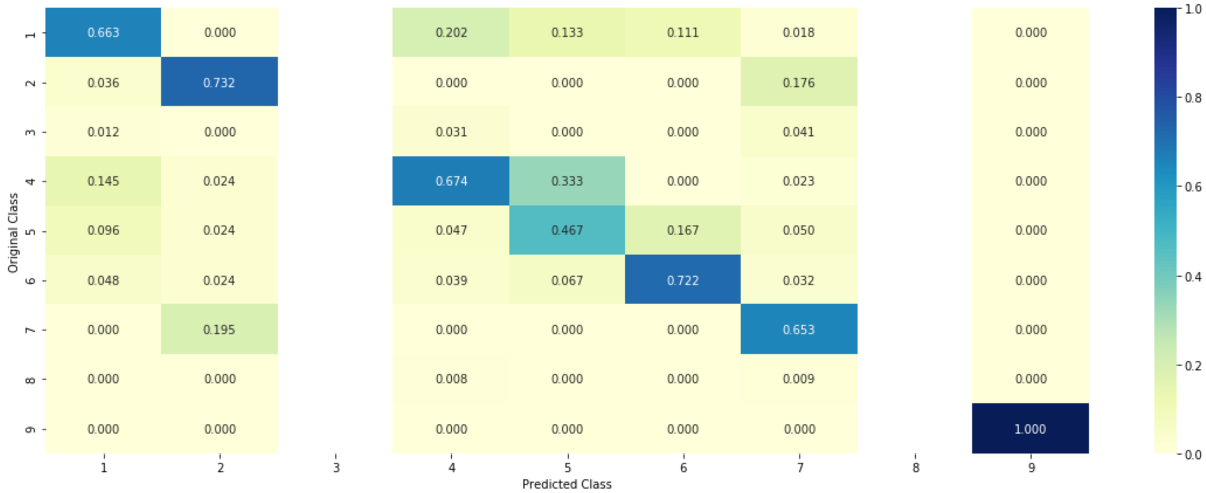
In [85]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
```

```
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

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

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
random_state=42)
log_loss_val, misc_rate = predict_and_plot_confusion_matrix(bow_bi_trai
n_x_onehotCoding, train_y, bow_bi_cv_x_onehotCoding, cv_y, clf)

result_report = result_report.append({"Vectorizer": "BoW", "N-Gram": "
(1,2)",
                                      "Model": "Logistic-Regression (Wi
thout Class Balanced)",
                                      "TRAIN-Score": np.round(train_log
_loss, 4),
                                      "CV-Score": np.round(cv_log_loss,
 4),
                                      "TEST-Score": np.round(test_log_l
oss, 4),
                                      "Misclassification-Rate": '{}%'.f
ormat(np.round(misc_rate * 100, 2))
                                     }, ignore_index=True)
```

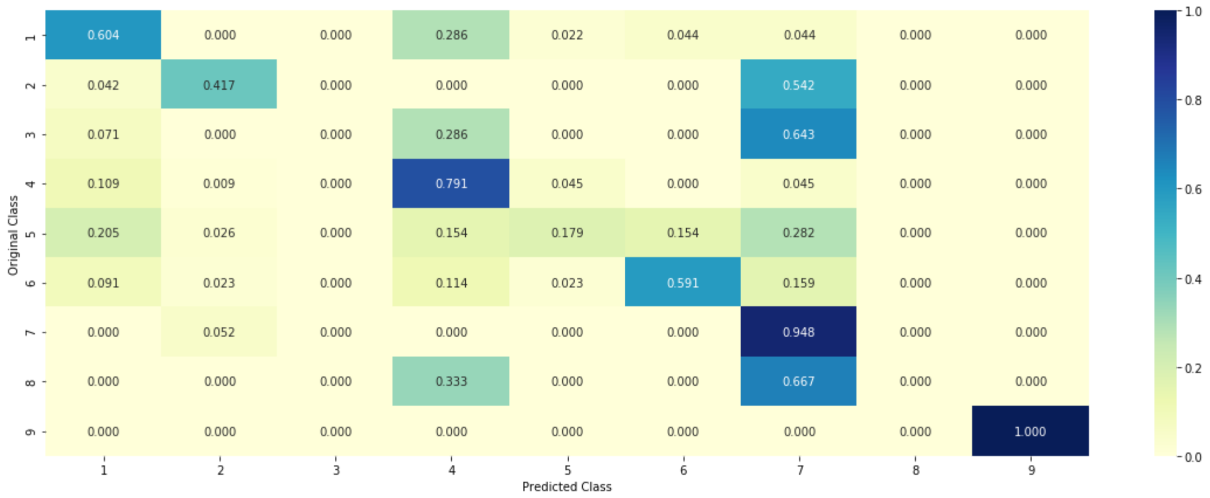
```
Log loss : 0.9394269693583398
Number of mis-classified points : 0.3308270676691729
-------------------- Confusion matrix --------------------
```



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



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



4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [86]:

```
# read more about support vector machines with linear kernals here htt
p://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
```

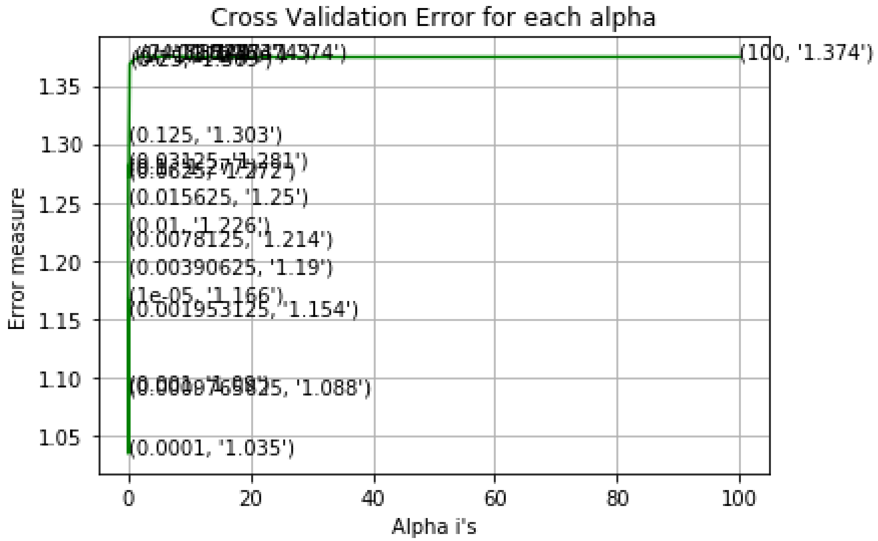
```
=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decisi
on_function_shape='ovr', random_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the give
n 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.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='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 = sorted([10 ** x for x in range(-5, 3)] + [2**i for i in range(-
10, -1)] + [2**i for i in range(1, 5)])
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
#     clf = SVC(C=i,kernel='linear',probability=True, class_weight='bal
anced')
```

```
    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='balance
d')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='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)
train_log_loss = log_loss(y_train, predict_y, labels=clf.classes_, eps=
1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The train log
 loss is:", train_log_loss)
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
cv_log_loss = log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:", cv_log_loss)
```

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

```
for C = 1e-05
Log Loss : 1.1661855590453825
for C = 0.0001
Log Loss : 1.0350319429806698
for C = 0.0009765625
Log Loss : 1.0878206758174886
for C = 0.001
Log Loss : 1.0896968126754782
for C = 0.001953125
Log Loss : 1.153677181132207
for C = 0.00390625
Log Loss : 1.1899275477511284
for C = 0.0078125
Log Loss : 1.2135245119595792
for C = 0.01
Log Loss : 1.2257101778889539
for C = 0.015625
Log Loss : 1.2495694060242486
for C = 0.03125
Log Loss : 1.2814866636622828
for C = 0.0625
Log Loss : 1.2718343636831013
for C = 0.1
Log Loss : 1.2767279023446232
for C = 0.125
Log Loss : 1.3026062972359778
for C = 0.25
Log Loss : 1.3685605871516873
for C = 1
Log Loss : 1.3719698778775216
for C = 2
Log Loss : 1.3744890178775684
for C = 4
Log Loss : 1.3743905944251031
```

```
for C = 8
Log Loss : 1.3744139805073854
for C = 10
Log Loss : 1.3744473926258738
for C = 16
Log Loss : 1.3739592582807751


for C = 100
Log Loss : 1.3741296451914757
```




```
For values of best alpha =  0.0001 The train log loss is: 0.45035289422
985325
For values of best alpha =  0.0001 The cross validation log loss is: 1.
0350319429806698
For values of best alpha =  0.0001 The test log loss is: 1.075295150789
2468
```

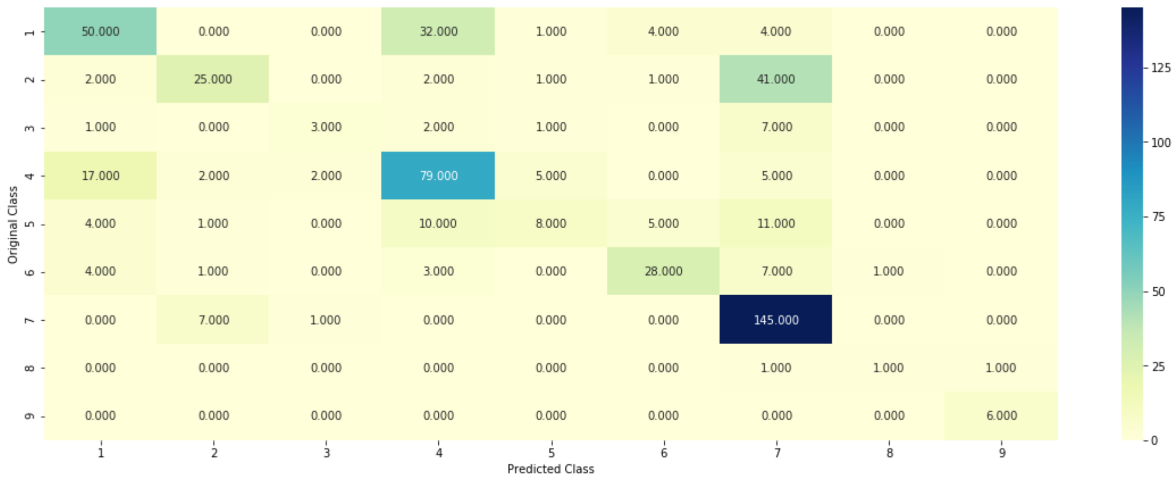
4.4.2. Testing model with best hyper parameters

In [87]:
```
# read more about support vector machines with linear kernals here htt
```

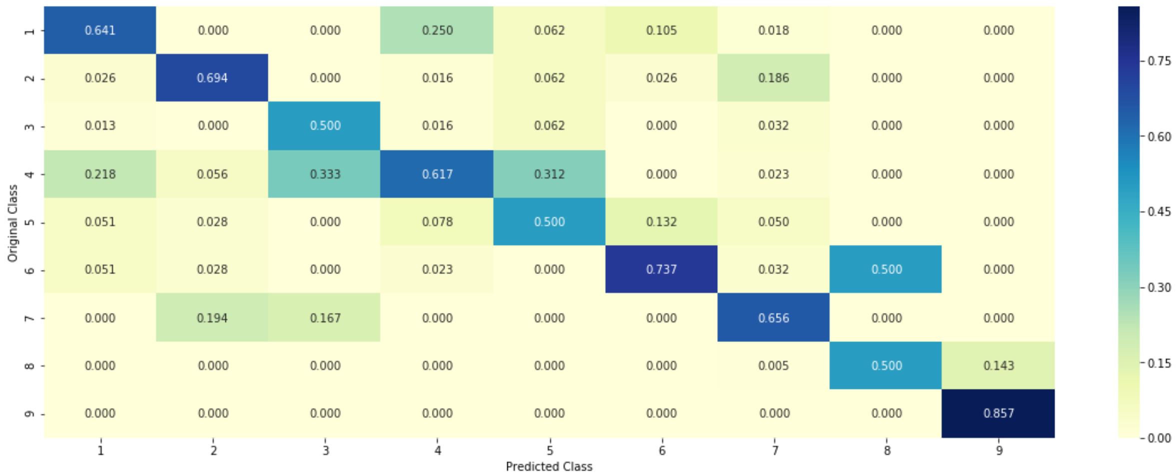
```
p://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

# --------------------------------
# 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, decisi
on_function_shape='ovr', random_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the give
n 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')
log_loss_val, misc_rate = predict_and_plot_confusion_matrix(train_x_one
hotCoding, train_y,cv_x_onehotCoding,cv_y, clf)

result_report = result_report.append({"Vectorizer": "TF-IDF", "N-Gram":
 "(1,4)",
                                      "Model": "Linear SVM",
                                      "TRAIN-Score": np.round(train_log
_loss, 4),
                                      "CV-Score": np.round(cv_log_loss,
 4),
                                      "TEST-Score": np.round(test_log_l
oss, 4),
                                      "Misclassification-Rate": '{}%'.f
ormat(np.round(misc_rate * 100, 2))
                                     }, ignore_index=True)
```

```
Log loss : 1.0350319429806698
```

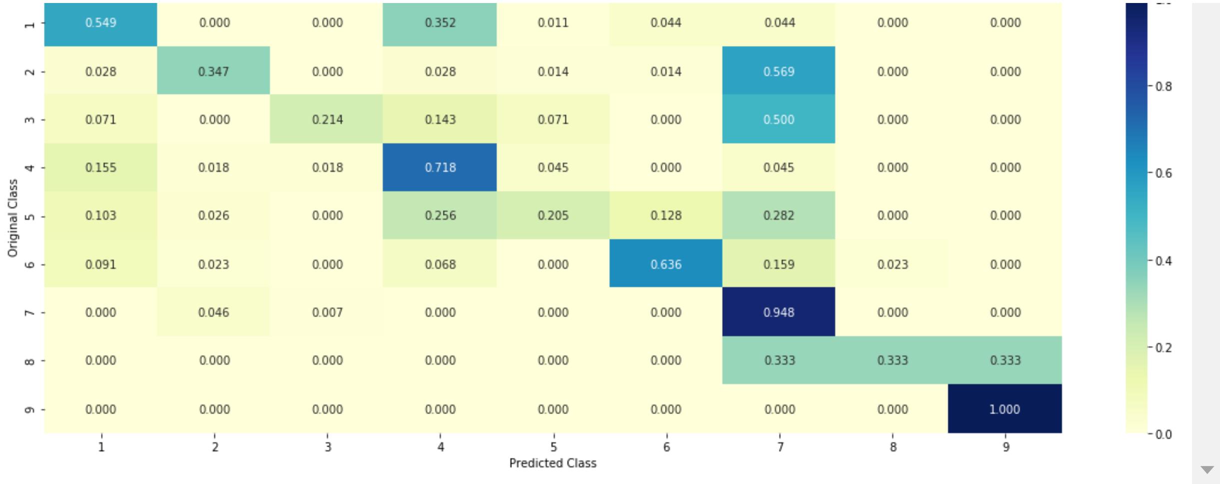
```
Number of mis-classified points : 0.3515037593984962 6
-------------------- Confusion matrix --------------------
```




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




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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [88]:
```
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(-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 : 1

Predicted Class Probabilities: [[0.8727 0.0766 0.0111 0.005  0.0037 0.0
124 0.0118 0.0024 0.0042]]
Actual Class : 1
--------------------------------------------------
91 Text feature [152] present in test data point [True]
317 Text feature [115] present in test data point [True]
321 Text feature [005] present in test data point [True]
322 Text feature [198] present in test data point [True]
343 Text feature [169] present in test data point [True]
403 Text feature [158] present in test data point [True]
408 Text feature [025] present in test data point [True]
414 Text feature [101] present in test data point [True]
425 Text feature [197] present in test data point [True]
446 Text feature [178] present in test data point [True]
489 Text feature [1e] present in test data point [True]
490 Text feature [015] present in test data point [True]
Out of the top  500  features  12 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [89]:

```
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(-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: [[0.0627 0.3546 0.0141 0.1027 0.0421 0.0
26  0.3854 0.0055 0.0068]]
Actual Class : 7
```

```
Actual Class : 7
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [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.h
```

```
tml
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='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, 15, 20, 50, 100, 250]
cv_log_error_array = []
l_log_loss = 999
l_alpha = 0
l_max_depth = 0
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)
        log_loss_val = log_loss(cv_y, sig_clf_probs, labels=clf.classes
_, eps=1e-15)
        cv_log_error_array.append(log_loss_val)
        if l_log_loss > log_loss_val:
            l_log_loss = log_loss_val
            l_alpha = i
            l_max_depth = j
        print("Log Loss :",log_loss_val)
```

```
'''fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ra
vel()
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)), (featur
es[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=l_alpha, criterion='gini', ma
x_depth=l_max_depth, 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)
train_log_loss = log_loss(y_train, predict_y, labels=clf.classes_, eps=
1e-15)
print('For values of best estimator = ', l_alpha, "The train log loss i
s:",train_log_loss)
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
cv_log_loss  = log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15
)
print('For values of best estimator = ', l_alpha, "The cross validation
 log loss is:",cv_log_loss)
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
test_log_loss = log_loss(y_test, predict_y, labels=clf.classes_, eps=1e
-15)
print('For values of best estimator = ', l_alpha, "The test log loss i
s:",test_log_loss)
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2437864559651777
for n_estimators = 100 and max depth =  10
```

```
Log Loss : 1.1494247625006009
for n_estimators = 100 and max depth =  15
Log Loss : 1.0988536677551692
for n_estimators = 100 and max depth =  20
Log Loss : 1.0958361862692791
for n_estimators = 100 and max depth =  50
Log Loss : 1.1061617640432733
for n_estimators = 100 and max depth =  100
Log Loss : 1.107933532045093
for n_estimators = 100 and max depth =  250
Log Loss : 1.107933532045093
for n_estimators = 200 and max depth =  5
Log Loss : 1.232445811223983
for n_estimators = 200 and max depth =  10
Log Loss : 1.1429634478332986
for n_estimators = 200 and max depth =  15
Log Loss : 1.0955932931268155
for n_estimators = 200 and max depth =  20
Log Loss : 1.091872661940601
for n_estimators = 200 and max depth =  50
Log Loss : 1.105389403026831
for n_estimators = 200 and max depth =  100
Log Loss : 1.1061313855067738
for n_estimators = 200 and max depth =  250
Log Loss : 1.1061313855067738
for n_estimators = 500 and max depth =  5
Log Loss : 1.2287353732695592
for n_estimators = 500 and max depth =  10
Log Loss : 1.1430072521059282
for n_estimators = 500 and max depth =  15
Log Loss : 1.0945487118542012
for n_estimators = 500 and max depth =  20
Log Loss : 1.0872671467112454
for n_estimators = 500 and max depth =  50
Log Loss : 1.0982374349589352
for n_estimators = 500 and max depth =  100
Log Loss : 1.0986004951134005
for n_estimators = 500 and max depth =  250
Log Loss : 1.0986004951134005
for n_estimators = 1000 and max depth =  5
```

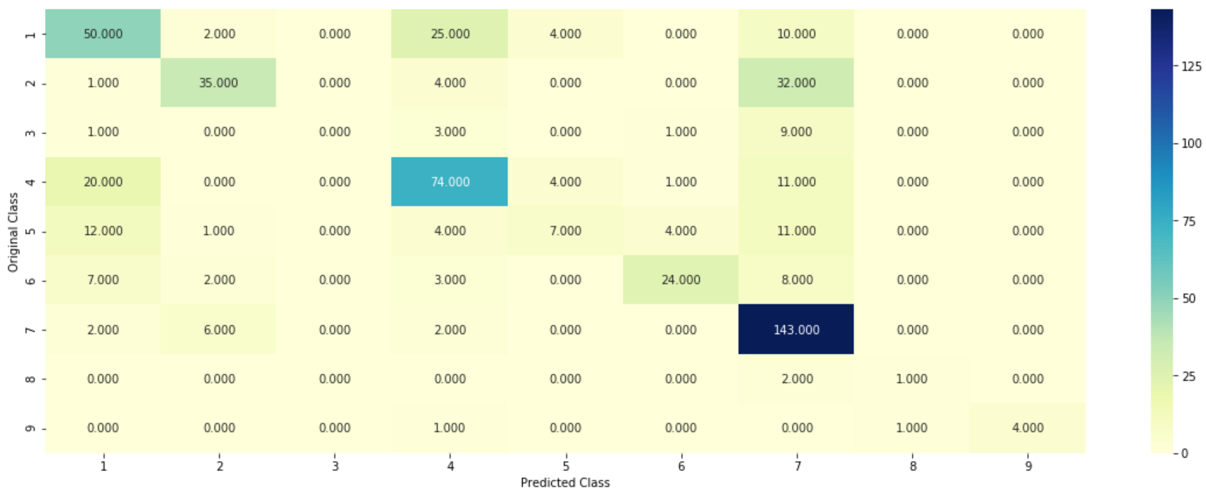
```
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2281256169051962
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1380662455604758
for n_estimators = 1000 and max depth =  15
Log Loss : 1.0937836187135241
for n_estimators = 1000 and max depth =  20
Log Loss : 1.0862083631471784
for n_estimators = 1000 and max depth =  50
Log Loss : 1.0966499904929077

for n_estimators = 1000 and max depth =  100
Log Loss : 1.096937683629313
for n_estimators = 1000 and max depth =  250
Log Loss : 1.096937683629313
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2281332824635873
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1362053434196722
for n_estimators = 2000 and max depth =  15
Log Loss : 1.095682590227396
for n_estimators = 2000 and max depth =  20
Log Loss : 1.0840309283839273
for n_estimators = 2000 and max depth =  50
Log Loss : 1.0947095668734772
for n_estimators = 2000 and max depth =  100
Log Loss : 1.0941934378263891
for n_estimators = 2000 and max depth =  250
Log Loss : 1.0941934378263891
For values of best estimator =  2000 The train log loss is: 0.525556931
0661152
For values of best estimator =  2000 The cross validation log loss is:
1.0840309283839273
For values of best estimator =  2000 The test log loss is: 1.1202578113
517503
```

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

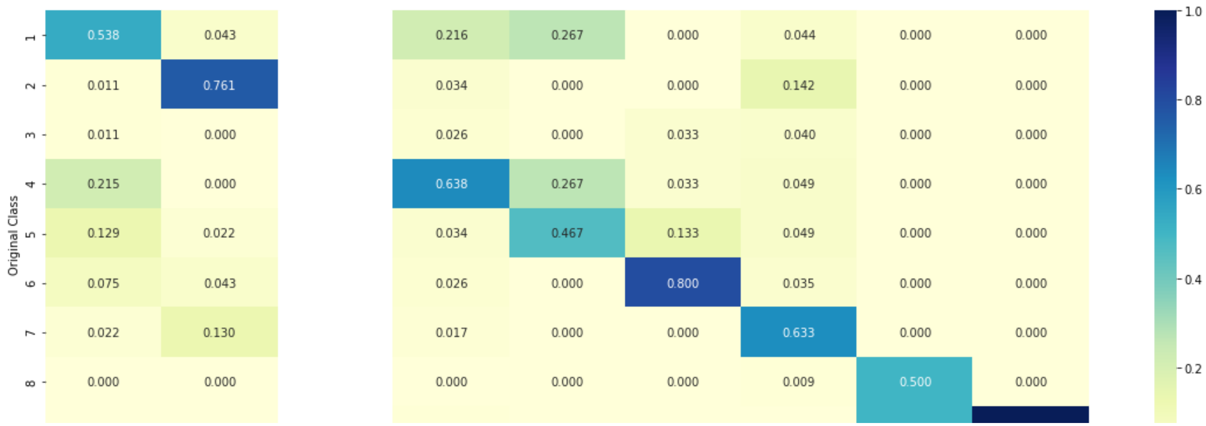
In [91]:
```
# ---------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_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 give
n 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=l_alpha, criterion='gini', ma
x_depth=l_max_depth, random_state=42, n_jobs=-1)
log_loss_val, misc_rate = predict_and_plot_confusion_matrix(train_x_one
hotCoding, train_y,cv_x_onehotCoding,cv_y, clf)

result_report = result_report.append({"Vectorizer": "TF-IDF", "N-Gram":
 "(1,4)",
                                      "Model": "RandomForest (With One-
Hot-Encoding)",
                                      "TRAIN-Score": np.round(train_log
_loss, 4),
                                      "CV-Score": np.round(cv_log_loss,
 4),
                                      "TEST-Score": np.round(test_log_l
```

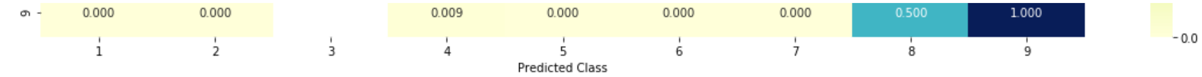
```
oss, 4),
                                    "Misclassification-Rate": '{}%'.f
ormat(np.round(misc_rate * 100, 2))
                                   }, ignore_index=True)
```

```
Log loss : 1.0840309283839273

Number of mis-classified points : 0.36466165413533835
-------------------- Confusion matrix --------------------
```

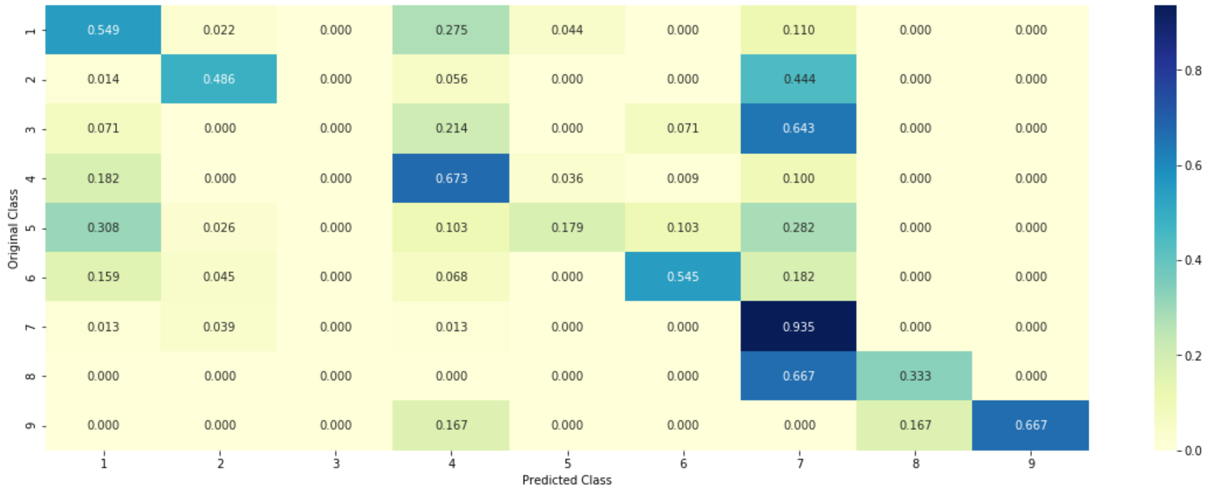



```
-------------------- Precision matrix (Columm Sum=1) ------------------
--
```






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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [92]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=l_alpha, criterion='gini', ma
x_depth=l_max_depth, 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_po
int_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].
iloc[test_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.4961 0.0669 0.0166 0.2447 0.0444 0.0
436 0.0762 0.0052 0.0062]]
Actual Class : 1
--------------------------------------------------
18 Text feature [015] present in test data point [True]
30 Text feature [13] present in test data point [True]
40 Text feature [197] present in test data point [True]
Out of the top  100  features  3 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [93]:

```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
```

```
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_po
int_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].
iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0827 0.186  0.0161 0.0905 0.0426 0.0
417 0.5282 0.0061 0.0062]]
Actuall Class : 7
--------------------------------------------------
30 Text feature [13] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [94]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_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 give
n 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.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# ------------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='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, 2000]
max_depth = [2,3,5,10, 15, 25, 51]
cv_log_error_array = []
l_log_loss = 999
l_alpha = 0
l_max_depth = 0
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)
        log_loss_val = log_loss(cv_y, sig_clf_probs, labels=clf.classes
_, eps=1e-15)
        cv_log_error_array.append(log_loss_val)
        print("Log Loss :",log_loss_val)
        if l_log_loss > log_loss_val:
```

```
            l_log_loss = log_loss_val
            l_alpha = i
            l_max_depth = j
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ra
vel()
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)), (featur
es[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''


best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=l_alpha, criterion='gini', ma
x_depth=l_max_depth, 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)
train_log_loss = log_loss(y_train, predict_y, labels=clf.classes_, eps=
1e-15)
print('For values of best alpha = ', l_alpha, "The train log loss is:",
 train_log_loss)
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
cv_log_loss = log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', l_alpha, "The cross validation log
 loss is:", cv_log_loss)
predict_y = sig_clf.predict_proba(test_x_responseCoding)
test_log_loss = log_loss(y_test, predict_y, labels=clf.classes_, eps=1e
-15)
print('For values of best alpha = ', l_alpha, "The test log loss is:",
test_log_loss)
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.0236553850363266
for n_estimators = 10 and max depth =  3
Log Loss : 1.5807889980583667
for n_estimators = 10 and max depth =  5
Log Loss : 1.3319846120113743
for n_estimators = 10 and max depth =  10
Log Loss : 1.7021254652139617
for n_estimators = 10 and max depth =  15
Log Loss : 1.9725964624615715
for n_estimators = 10 and max depth =  25
Log Loss : 1.9392326546855967
for n_estimators = 10 and max depth =  51
Log Loss : 1.9392326546855967
for n_estimators = 50 and max depth =  2
Log Loss : 1.5629351284196198
for n_estimators = 50 and max depth =  3
Log Loss : 1.3355732634621753
for n_estimators = 50 and max depth =  5
Log Loss : 1.2379195931348093
for n_estimators = 50 and max depth =  10
Log Loss : 1.717936566012422
for n_estimators = 50 and max depth =  15
Log Loss : 1.8458281087393043
for n_estimators = 50 and max depth =  25
Log Loss : 1.8203450354310464
for n_estimators = 50 and max depth =  51
Log Loss : 1.8203450354310464
for n_estimators = 100 and max depth =  2
Log Loss : 1.431879591777632
for n_estimators = 100 and max depth =  3
Log Loss : 1.3385213469262618
for n_estimators = 100 and max depth =  5
Log Loss : 1.202232862553944
for n_estimators = 100 and max depth =  10
Log Loss : 1.6448489436423468
for n_estimators = 100 and max depth =  15
Log Loss : 1.8199276052576956
for n_estimators = 100 and max depth =  25
Log Loss : 1.8129700422835409
```

```
for n_estimators = 100 and max depth =  51
Log Loss : 1.8129700422835409
for n_estimators = 200 and max depth =  2
Log Loss : 1.4908728538345613
for n_estimators = 200 and max depth =  3
Log Loss : 1.3857692718444754
for n_estimators = 200 and max depth =  5
Log Loss : 1.2578248516852375

for n_estimators = 200 and max depth =  10
Log Loss : 1.644467571473362
for n_estimators = 200 and max depth =  15
Log Loss : 1.7859377800187857
for n_estimators = 200 and max depth =  25
Log Loss : 1.770385514193472
for n_estimators = 200 and max depth =  51
Log Loss : 1.770385514193472
for n_estimators = 500 and max depth =  2
Log Loss : 1.5692777165015506
for n_estimators = 500 and max depth =  3
Log Loss : 1.4378886226742553
for n_estimators = 500 and max depth =  5
Log Loss : 1.27915987127656
for n_estimators = 500 and max depth =  10
Log Loss : 1.7088498297896861
for n_estimators = 500 and max depth =  15
Log Loss : 1.8432779228022471
for n_estimators = 500 and max depth =  25
Log Loss : 1.84074544731925
for n_estimators = 500 and max depth =  51
Log Loss : 1.84074544731925
for n_estimators = 1000 and max depth =  2
Log Loss : 1.5423855931227617
for n_estimators = 1000 and max depth =  3
Log Loss : 1.4532187016363762
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2913423594849887
for n_estimators = 1000 and max depth =  10
Log Loss : 1.7137046084235095
for n_estimators = 1000 and max depth =  15
```

```
Log Loss : 1.8514710717846916
for n_estimators = 1000 and max depth =  25
Log Loss : 1.8484539444528376
for n_estimators = 1000 and max depth =  51
Log Loss : 1.8484539444528376
for n_estimators = 2000 and max depth =  2
Log Loss : 1.5820103343500278
for n_estimators = 2000 and max depth =  3

Log Loss : 1.4605379710154736
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2965697338277975
for n_estimators = 2000 and max depth =  10
Log Loss : 1.7207268321949283
for n_estimators = 2000 and max depth =  15
Log Loss : 1.874579108496684
for n_estimators = 2000 and max depth =  25
Log Loss : 1.8771752982317493
for n_estimators = 2000 and max depth =  51
Log Loss : 1.8771752982317493
For values of best alpha =  100 The train log loss is: 0.06077609515841
254
For values of best alpha =  100 The cross validation log loss is: 1.202
232862553944
For values of best alpha =  100 The test log loss is: 1.27912942361283 9
7
```

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

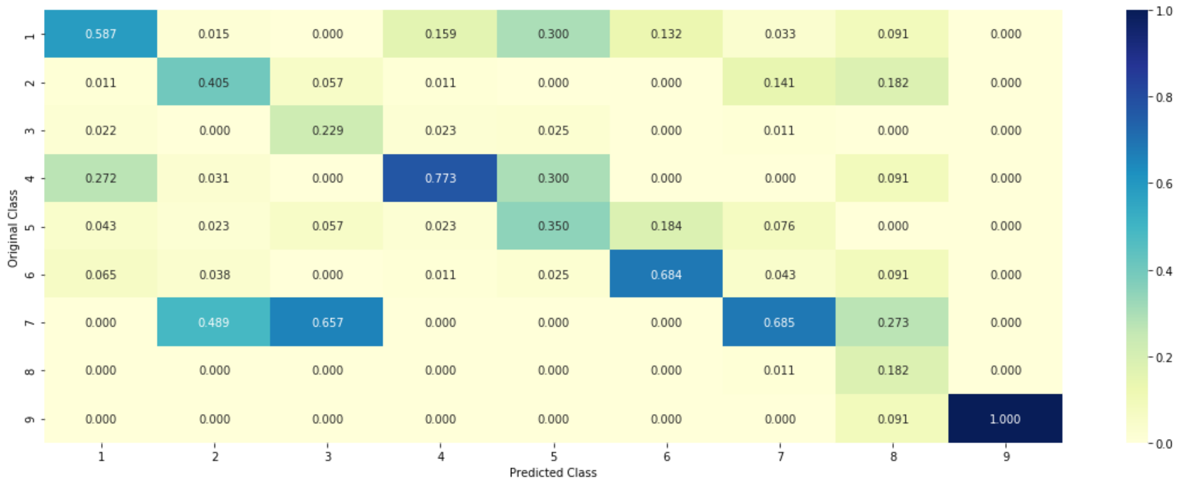
In [95]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
```

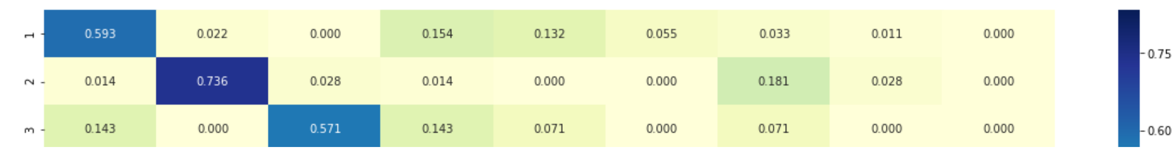
```
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the give
n 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=l_max_depth, n_estimators=l_alph
a, criterion='gini', max_features='auto',random_state=42)
log_loss_val, misc_rate = predict_and_plot_confusion_matrix(train_x_res
ponseCoding, train_y,cv_x_responseCoding,cv_y, clf)

result_report = result_report.append({"Vectorizer": "TF-IDF", "N-Gram":
 "(1,4)",
                                      "Model": "RandomForest (With Resp
onse-Encoding)",
                                      "TRAIN-Score": np.round(train_log
_loss, 4),
                                      "CV-Score": np.round(cv_log_loss,
 4),
                                      "TEST-Score": np.round(test_log_l
oss, 4),
                                      "Misclassification-Rate": '{}%'.f
ormat(np.round(misc_rate * 100, 2))
                                     }, ignore_index=True)
```

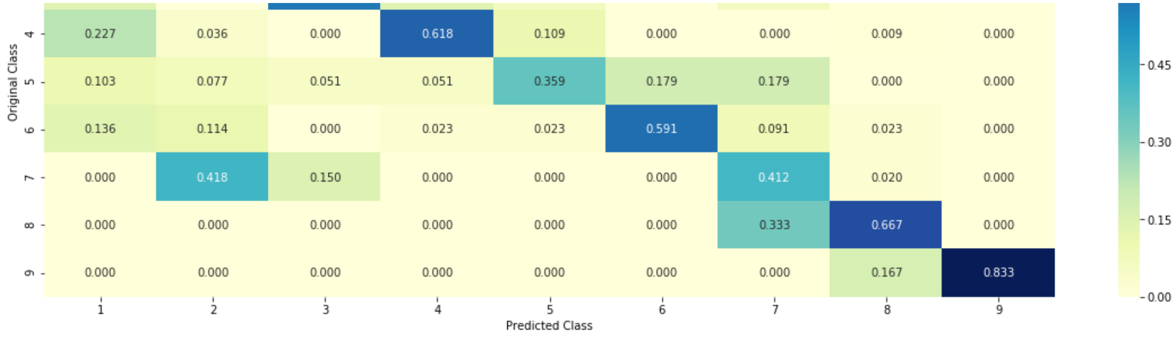
```
Log loss : 1.202232862553944
Number of mis-classified points : 0.4492481203007519
-------------------- Confusion matrix --------------------
```


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




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





4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [96]:
```
clf = RandomForestClassifier(n_estimators=l_alpha, criterion='gini', ma
x_depth=l_max_depth, 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 : 1
Predicted Class Probabilities: [[0.9813 0.0021 0.0014 0.0038 0.0014 0.0
027 0.0019 0.0025 0.0027]]
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text 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
Gene is important feature
Variation 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
```

```
Text is important feature

Variation is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [97]:

```
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: [[0.0161 0.4509 0.0867 0.0194 0.0351 0.0
389 0.3147 0.0237 0.0144]]
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
Text 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
Gene is important feature
Variation 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
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [98]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

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



# read more about support vector machines with linear kernals here htt
p://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, decisi
on_function_shape='ovr', random_state=None)

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



# read more about support vector machines with linear kernals here htt
p://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomFo
restClassifier.html
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
```

```
andom_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 give
n 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_weigh
t='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_cl
f1.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.predic
t_proba(cv_x_onehotCoding))))
print("-"*50)
alpha = sorted([0.0001,0.001,0.01,0.1,1,10]  + [2**i for i in range(-10
, -1)] + [2**i for i in range(1, 5)])
best_alpha = 999
best_alpha_val = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3
], meta_classifier=lr, use_probas=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %
0.3f" % (i, log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha_val > log_error:
        best_alpha_val = log_error
        best_alpha = i
```

```
Logistic Regression :  Log Loss: 0.98
Support vector machines : Log Loss: 1.37
Naive Bayes : Log Loss: 1.20
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.000977 Log Loss: 2.037
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.034
Stacking Classifer : for the value of alpha: 0.001953 Log Loss: 1.923
Stacking Classifer : for the value of alpha: 0.003906 Log Loss: 1.765
Stacking Classifer : for the value of alpha: 0.007812 Log Loss: 1.578
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.510
Stacking Classifer : for the value of alpha: 0.015625 Log Loss: 1.397
Stacking Classifer : for the value of alpha: 0.031250 Log Loss: 1.257
Stacking Classifer : for the value of alpha: 0.062500 Log Loss: 1.173
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.146
Stacking Classifer : for the value of alpha: 0.125000 Log Loss: 1.141
Stacking Classifer : for the value of alpha: 0.250000 Log Loss: 1.153
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.289
Stacking Classifer : for the value of alpha: 2.000000 Log Loss: 1.393
```

```
Stacking Classifer : for the value of alpha: 4.000000 Log Loss: 1.510
Stacking Classifer : for the value of alpha: 8.000000 Log Loss: 1.633
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.673
Stacking Classifer : for the value of alpha: 16.000000 Log Loss: 1.760
```

4.7.2 testing the model with the best hyper parameters

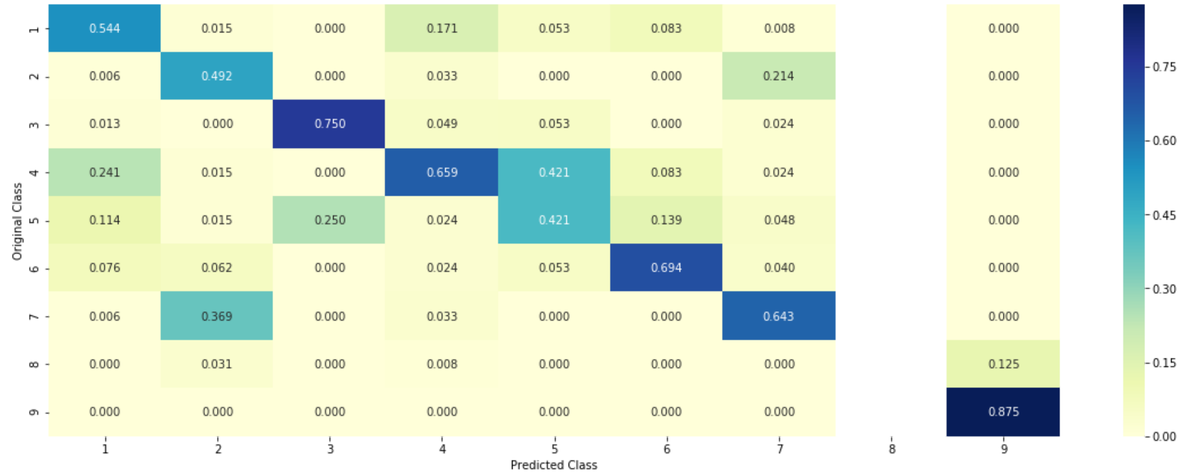
In [99]:

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

log_error_train = log_loss(train_y, sclf.predict_proba(train_x_onehotCo
ding))
print("Log loss (train) on the stacking classifier :",log_error_train)

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

log_error_test = log_loss(test_y, sclf.predict_proba(test_x_onehotCodin
g))
print("Log loss (test) on the stacking classifier :",log_error_test)

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

result_report = result_report.append({"Vectorizer": "TF-IDF", "N-Gram":
 "(1,4)",
                                      "Model": "Stacking [LogisticRegre
ssion, SVM, NaiveBayes ===> LogisticRegression]",
                                      "TRAIN-Score": np.round(log_error
_train, 4),
                                      "CV-Score": np.round(log_error_cv
, 4),
                                      "TEST-Score": np.round(log_error_
```

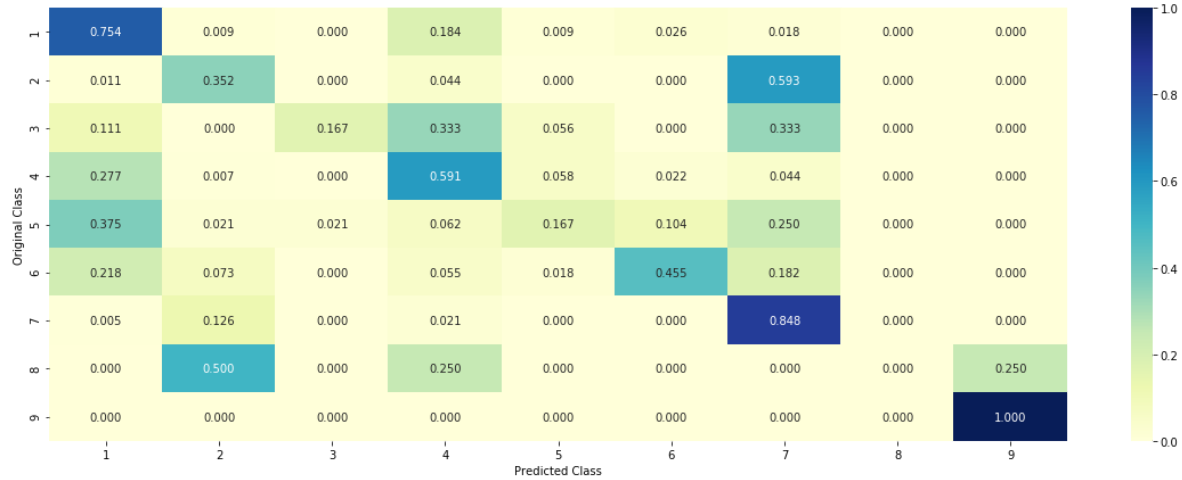
```
test, 4),
                                     "Misclassification-Rate": '{}%'.f
ormat(np.round(misc_rate * 100, 2))
                                    }, ignore_index=True)
```

```
Log loss (train) on the stacking classifier : 0.5724591020092523
Log loss (CV) on the stacking classifier : 1.1405524074096804
Log loss (test) on the stacking classifier : 1.166001181456692
Number of missclassified point : 0.39248120300751880
-------------------- Confusion matrix --------------------
```


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




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



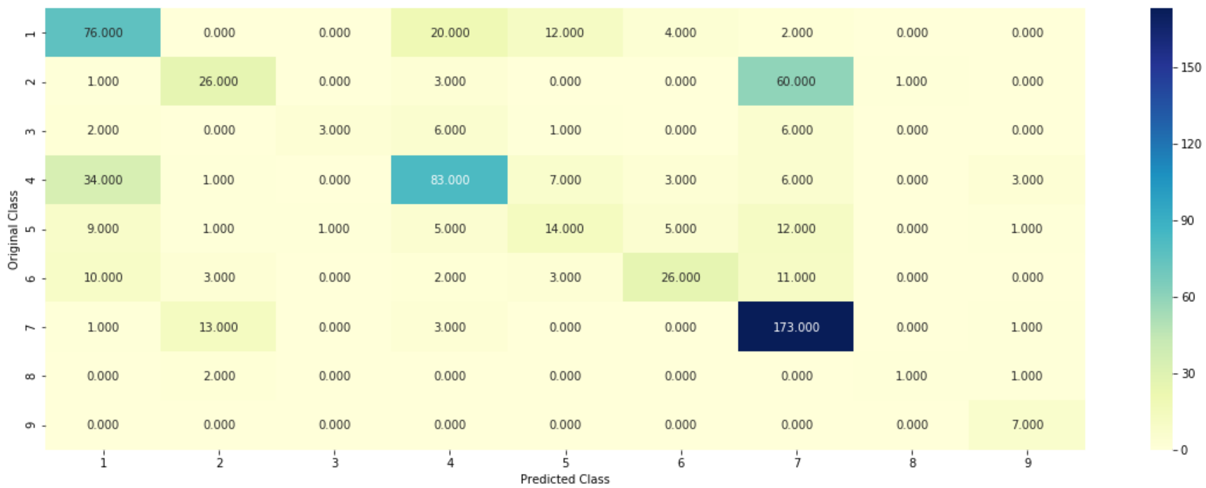
4.7.3 Maximum Voting classifier

In [100]:

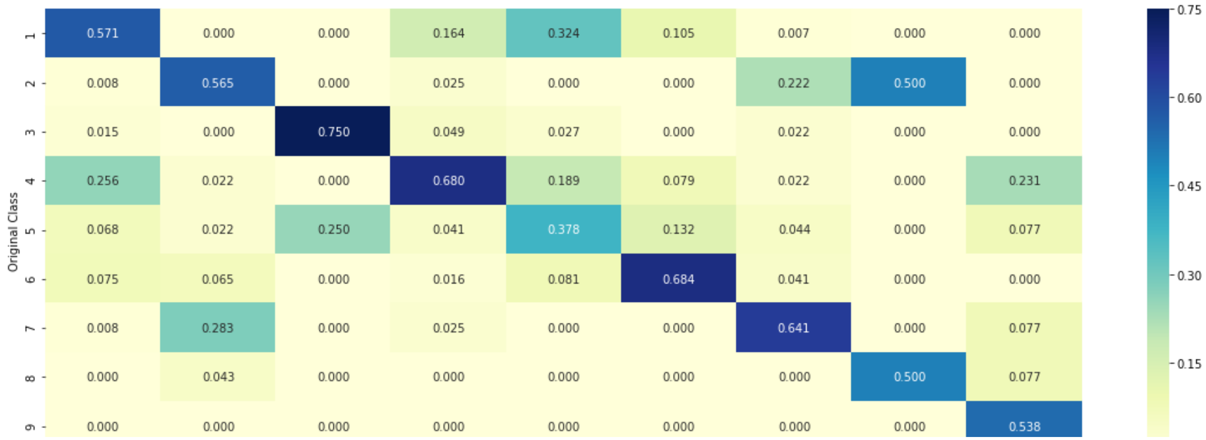
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
```

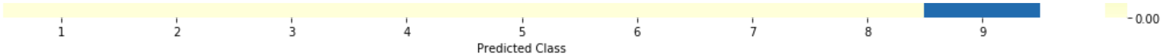
```
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2
), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
log_error_train = log_loss(train_y, vclf.predict_proba(train_x_onehotCo
ding))
print("Log loss (train) on the VotingClassifier :", log_error_train)
log_error_cv = log_loss(cv_y, vclf.predict_proba(cv_x_onehotCoding))
print("Log loss (CV) on the VotingClassifier :", log_error_cv)
log_error_test = log_loss(test_y, vclf.predict_proba(test_x_onehotCodin
g))
print("Log loss (test) on the VotingClassifier :", log_error_test)
misc_rate = np.count_nonzero((vclf.predict(test_x_onehotCoding)- test_y
))/test_y.shape[0]
print("Number of missclassified point :", misc_rate)
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_oneh
otCoding))

result_report = result_report.append({"Vectorizer": "TF-IDF", "N-Gram":
 "(1,4)",
                                       "Model": "Maximum Voting Classifi
er [LogisticRegression, SVM, RandomForest]",
                                       "TRAIN-Score": np.round(log_error
_train, 4),
                                       "CV-Score": np.round(log_error_cv
, 4),
                                       "TEST-Score": np.round(log_error_
test, 4),
                                       "Misclassification-Rate": '{}%'.f
ormat(np.round(misc_rate * 100, 2))
                                      }, ignore_index=True)
```

```
Log loss (train) on the VotingClassifier : 0.8106926083162947
Log loss (CV) on the VotingClassifier : 1.1141393164683024
Log loss (test) on the VotingClassifier : 1.157441526662839
Number of missclassified point : 0.3849624060150376
-------------------- Confusion matrix --------------------
```

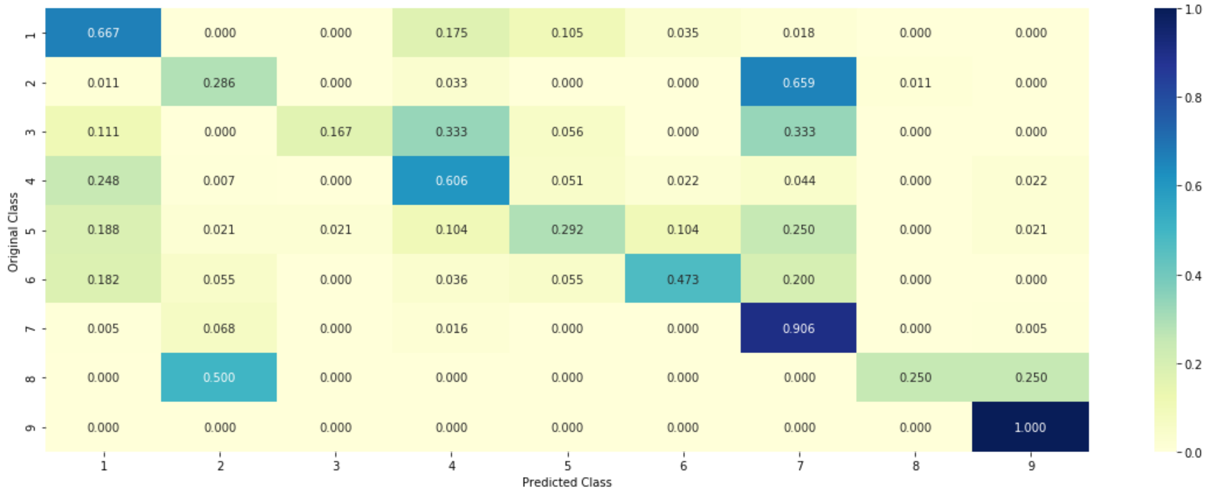


------------------- Precision matrix (Columm Sum=1) -----------------
--





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



5. Conclusions

In [101]:
```
pd.options.display.max_colwidth = 100
result_report
```

Out[101]:

	Vectorizer	N-Gram	Model	TRAIN-Score	CV-Score	TEST-Score	Misclassification-Rate
0	TF-IDF	(1,4)	Naive Bayes	0.9601	1.1674	1.1873	40.23%
1	TF-IDF	(1,4)	K-NN	0.5156	0.9716	1.0482	33.27%

	Vectorizer	N-Gram	Model	TRAIN-Score	CV-Score	TEST-Score	Misclassification-Rate
2	TF-IDF	(1,4)	Logistic-Regression (With Class Balanced)	0.4807	0.9429	0.9917	34.59%
3	TF-IDF	(1,4)	Logistic-Regression (Without Class Balanced)	0.4605	0.9402	0.9943	34.96%
4	BoW	(1,1)	Logistic-Regression (With Class Balanced)	0.6039	0.9973	1.0765	31.95%
5	BoW	(1,1)	Logistic-Regression (Without Class Balanced)	0.6019	1.0178	1.0965	32.14%
6	BoW	(1,2)	Logistic-Regression (With Class Balanced)	0.4532	0.9324	1.0093	33.27%
7	BoW	(1,2)	Logistic-Regression (Without Class Balanced)	0.4470	0.9394	1.0147	33.08%
8	TF-IDF	(1,4)	Linear SVM	0.4504	1.0350	1.0753	35.15%
9	TF-IDF	(1,4)	RandomForest (With One-Hot-Encoding)	0.5256	1.0840	1.1203	36.47%
10	TF-IDF	(1,4)	RandomForest (With Response-Encoding)	0.0608	1.2022	1.2791	44.92%
11	TF-IDF	(1,4)	Stacking [LogisticRegression, SVM, NaiveBayes ===> LogisticRegression]	0.5725	1.1406	1.1660	39.25%
12	TF-IDF	(1,4)	Maximum Voting Classifier [LogisticRegression, SVM, RandomForest]	0.8107	1.1141	1.1574	38.5%

In [103]:

```
print("Printing the minimum test score for model.. ")
result_report[result_report['TEST-Score']== result_report['TEST-Score']
.min()]
```

```
Printing the minimum test score for model..
```

Out[103]:

Vectorizer	N-Gram	Model	TRAIN-Score	CV-Score	TEST-Score	Misclassification-Rate

	Vectorizer	N-Gram	Model	TRAIN-Score	CV-Score	TEST-Score	Misclassification-Rate
2	TF-IDF	(1,4)	Logistic-Regression (With Class Balanced)	0.4807	0.9429	0.9917	34.59%

Summary

This is a multi-class classification problem where the output variable is having values 1-9.
Dataset size ~ 3k

Dataset consist of 2 files - ***variants and text.***

Variant file contains features like ID, Variation, Gene

Text file contains features ID and text

Basically, we had to combine both files on the ID feature. The Variation and Gene feature are the categorical features whereas the Text feature is a text feature which we needed to vectorize. We tried both ***Onehot encoding as well as response coding*** with the categorical features.

So we used ***TF-IDF (Term Frequency and Inverse Document Frequency) vectorizer*** with some minor feature engineering by taking a ***4 gram range and selecting the top 2k features***.

A variety of different models were tried including ***Naive Bayes, Support Vector Machine, K-Nearest Neighbors, Logistic Regression, RandomForest.***

Since Logistic Regression was performing well, We tried with ***BoW(Bag of Words) - both unigram and bi-gram vectorizer*** as well to dig in more.

As we can see the report above, Logistic Regression with Class balancing works well with CV logloss score to be of 0.94 and Test logloss score 0.99. Also the misclassification rate comes out to be only 34%.