Personalized cancer diagnosis

1. Business Problem

In [2]:

```
from google.colab import drive
drive.mount('/content/drive')
```

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

1.1. Description

In [0]:

```

```

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 [3]:

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

```
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

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

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning: T
he sklearn.metrics.classification module is  deprecated in version 0.22 and will be remov
ed in version 0.24. The corresponding classes / functions should instead be imported from
sklearn.metrics. Anything that cannot be imported from sklearn.metrics is now part of the
private API.
  warnings.warn(message, FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/externals/six.py:31: FutureWarning: The mo
dule is deprecated in version 0.21 and will be removed in version 0.23 since we've droppe
d support for Python 2.7. Please rely on the official version of six (https://pypi.org/pr
oject/six/).
  "(https://pypi.org/project/six/).", FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning: T
he sklearn.neighbors.base module is  deprecated in version 0.22 and will be removed in ve
rsion 0.24. The corresponding classes / functions should instead be imported from sklearn
.neighbors. Anything that cannot be imported from sklearn.neighbors is now part of the pr
ivate API.
  warnings.warn(message, FutureWarning)
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [4]:

```
data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/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[4]:

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

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

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

3.1.2. Reading Text Data

In [5]:

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

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

Out[5]:

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

3.1.3. Preprocessing of text

In [6]:

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

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Out[6]:

```
True
```

In [0]:

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


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

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

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

In [8]:

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

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 28.539808 seconds
```

In [9]:

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

Out[9]:

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

In [10]:

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

Out[10]:

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

In [0]:

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

In [12]:

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

Out[12]:

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

3.1.4. Test, Train and Cross Validation Split

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

In [0]:

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

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

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

In [14]:

```
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])
```

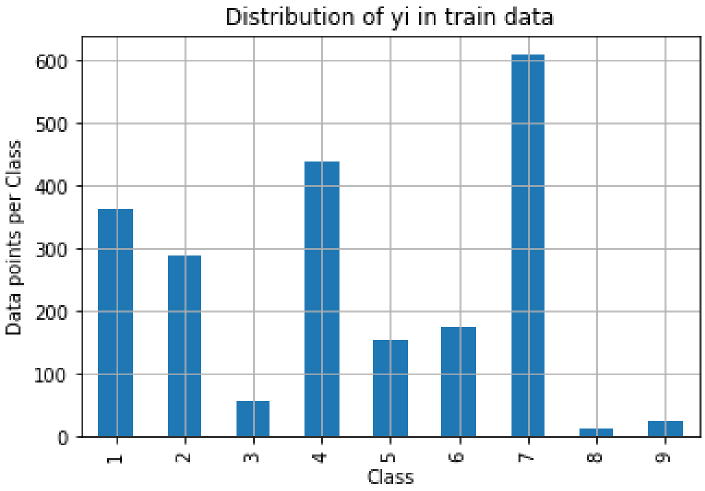
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

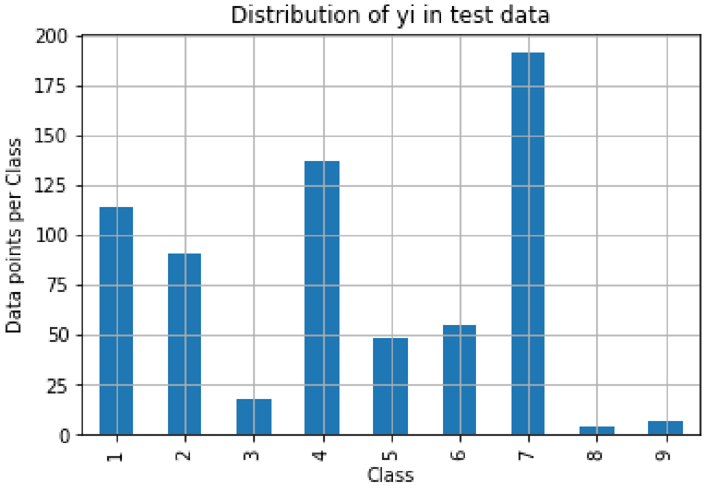
In [15]:

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

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

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

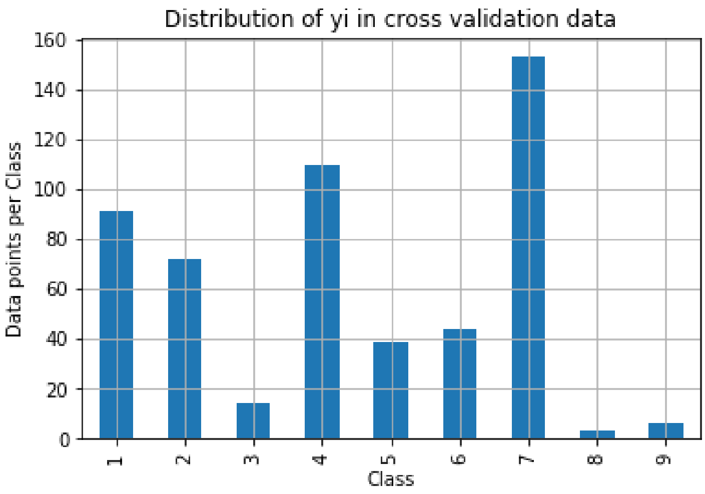
```
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.values[i],
'(', np.round((test_class_distribution.values[i]/test_df.shape[0]*100), 3), '%)')

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

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



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



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



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

3.2 Prediction using a 'Random' Model

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

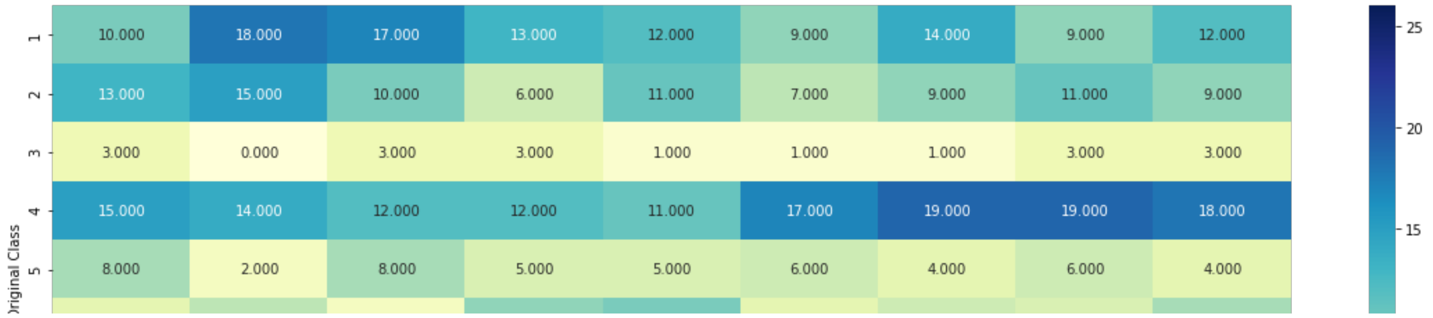
In [0]:

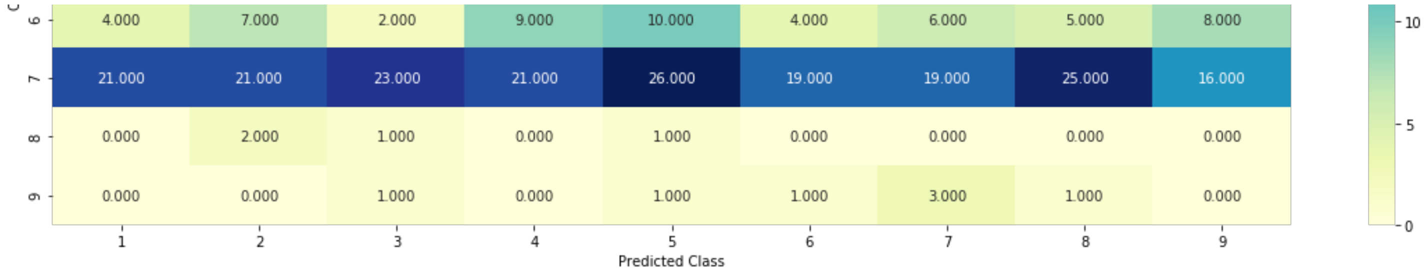
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicte
d class j
    
    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in that column
    
    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in two
diamensional array
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                           [2/3, 4/7]]

    # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
    #                           [3/7, 4/7]]
    # sum of row elements = 1
    
    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in two
diamensional array
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                         [3/4, 4/6]]

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

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

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

In [17]:

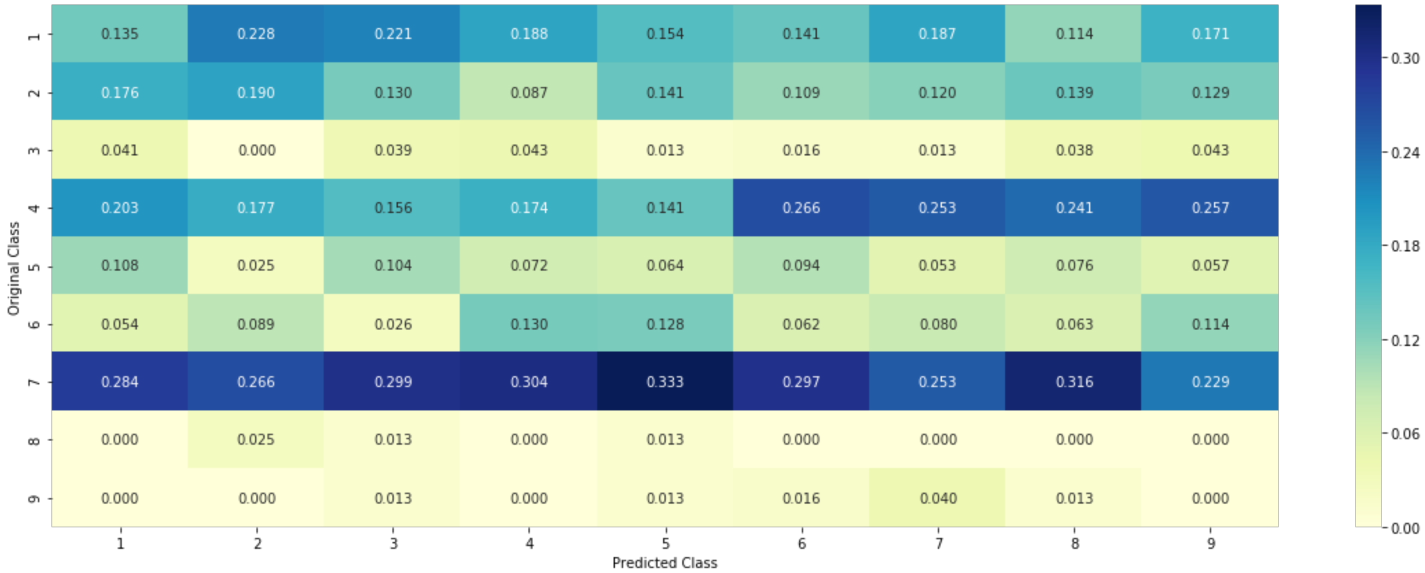
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

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


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

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

```
Log loss on Cross Validation Data using Random Model 2.388578256180409
Log loss on Test Data using Random Model 2.542029111298938
-------------------- Confusion matrix --------------------
```

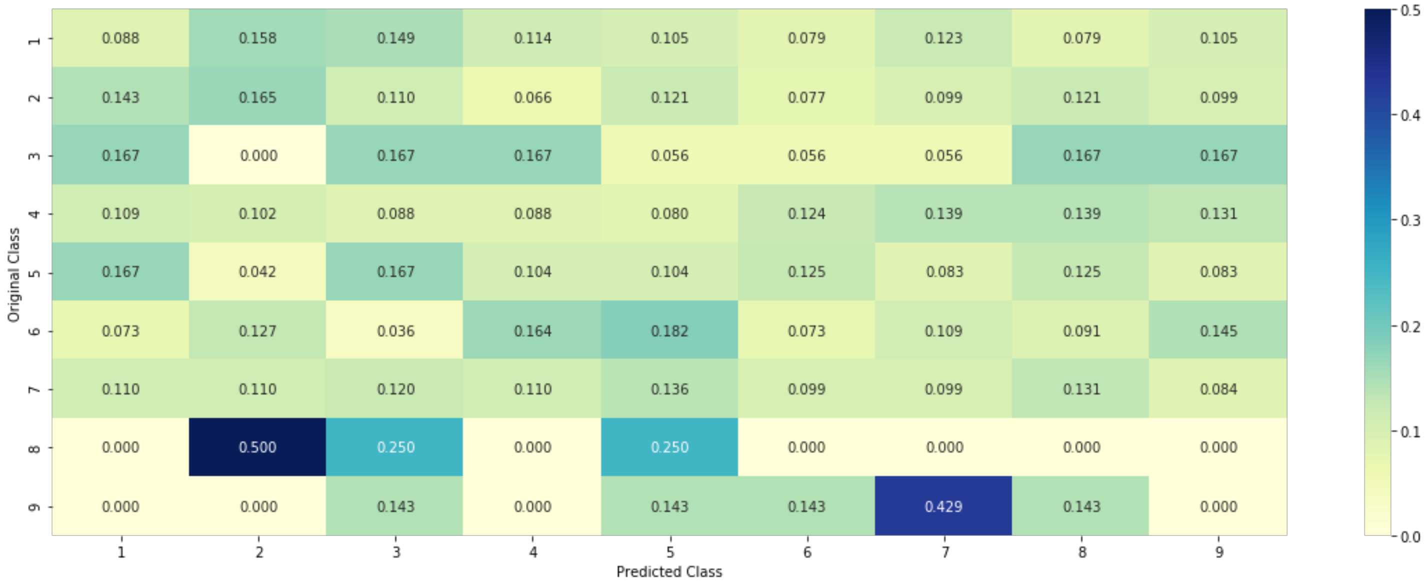




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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



3.3 Univariate Analysis

In [0]:

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

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

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

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

            # cls_cnt.shape[0](numerator) will contain the number of time that particula
r 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 as value
        gv_dict[i]=vec
    return gv_dict

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177, 0.1
3636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788, 0.
03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.2
7040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051
020408163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818181
77, 0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.
078787878787878782, 0.13939393939393939394, 0.34545454545454546, 0.060606060606060608, 0.0606
06060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.4
6540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062
893081761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.07
2847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702, 0.0662
25165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.07
3333333333333334, 0.09333333333333338, 0.08000000000000002, 0.29999999999999999, 0.0666
6666666666666, 0.06666666666666666],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

In [19]:

```
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 : 234
BRCA1     171
TP53      102
EGFR       88
BRCA2      85
PTEN       73
KIT        63
BRAF       57
ERBB2      52
ALK        43
PDGFRA     43
Name: Gene, dtype: int64
```

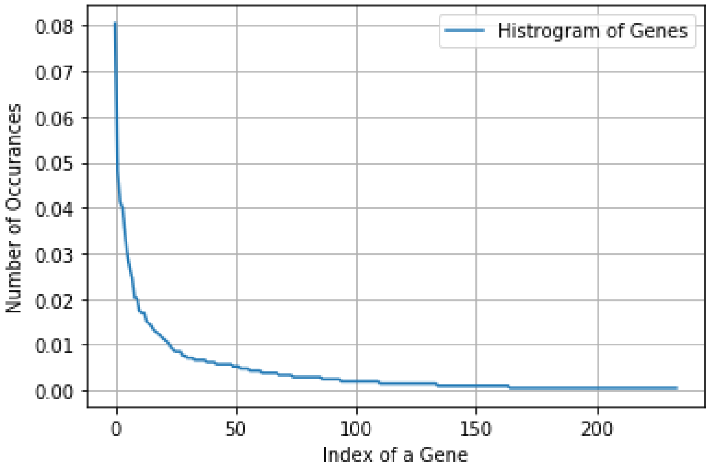
In [20]:

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

```
Ans: There are 234 different categories of genes in the train data, and they are distibut
ed as follows
```

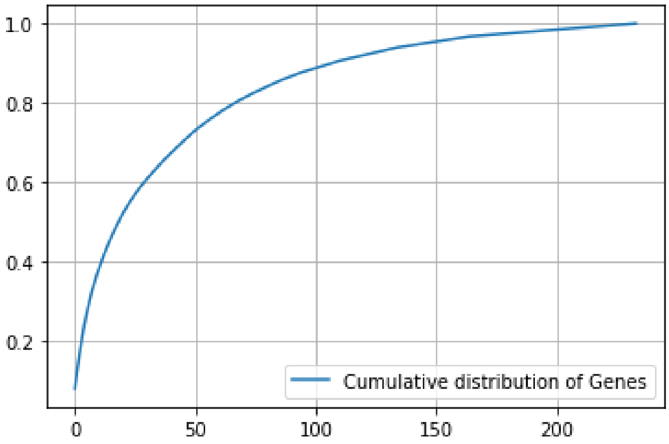
In [21]:

```
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 [22]:

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



Q3. How to featurize this Gene feature ?

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

1. **One hot Encoding**
2. **Response coding**

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

In [0]:

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

In [24]:

In [24]:

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

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

In [0]:

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

In [26]:

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

Out[26]:

```
768     ERBB2
2200     PTEN
3141     KRAS
3031      KIT
1466    FGFR2
Name: Gene, dtype: object
```

In [0]:

```
gene_vectorizer.get_feature_names()
```

In [28]:

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The s
hape of gene feature: (2124, 233)
```

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

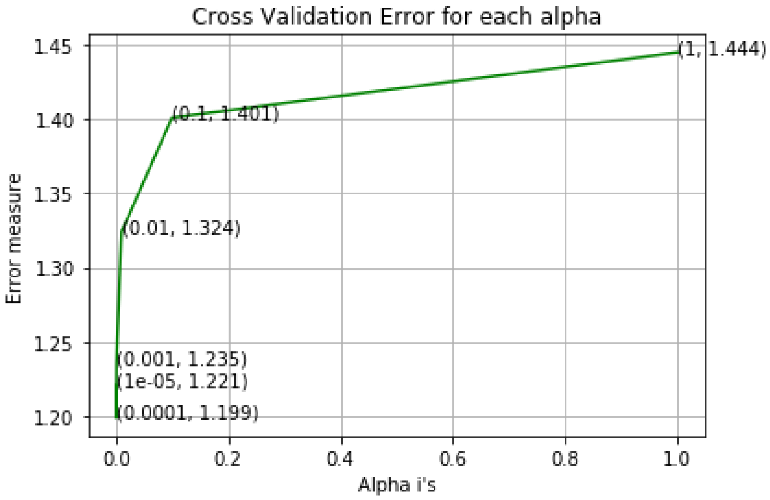
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [29]:

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

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

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

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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.2205860913725093
For values of alpha =  0.0001 The log loss is: 1.1990831943604452
For values of alpha =  0.001 The log loss is: 1.2348573639049238
For values of alpha =  0.01 The log loss is: 1.323856773166869
For values of alpha =  0.1 The log loss is: 1.4007064162077114
For values of alpha =  1 The log loss is: 1.4444181496757889
```



```
For values of best alpha =  0.0001 The train log loss is: 0.9827639669111043
For values of best alpha =  0.0001 The cross validation log loss is: 1.1990831943604452
For values of best alpha =  0.0001 The test log loss is: 1.2015229293140715
```

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 [30]:

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

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  234  genes in train
dataset?
Ans
1. In test data 648 out of 665 : 97.44360902255639
2. In cross validation data 514 out of  532 : 96.61654135338345
```

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 [31]:

```
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 : 1931
Truncating_Mutations    64
Deletion                49
Amplification           41
Fusions                 18
G12V                     4
E17K                     3
F384L                    2
A146V                    2
Q61K                     2
E542K                    2
Name: Variation, dtype: int64
```

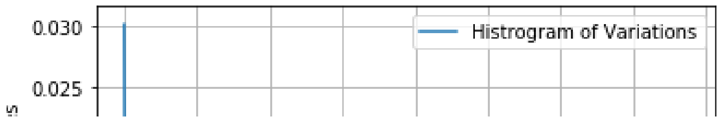
In [32]:

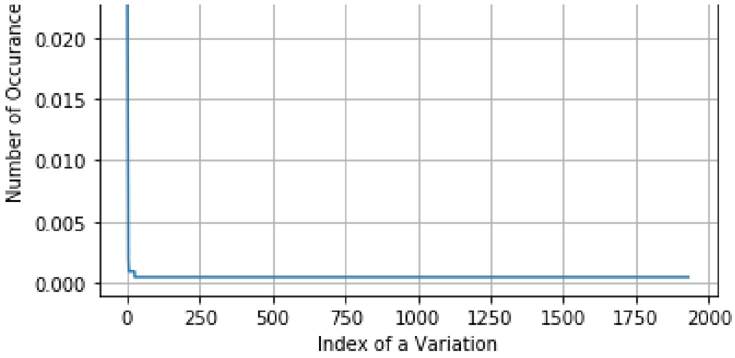
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of variations
in the train data, and they are distibuted as follows",)
```

```
Ans: There are 1931 different categories of variations in the train data, and they are di
stibuted as follows
```

In [33]:

```
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()
```

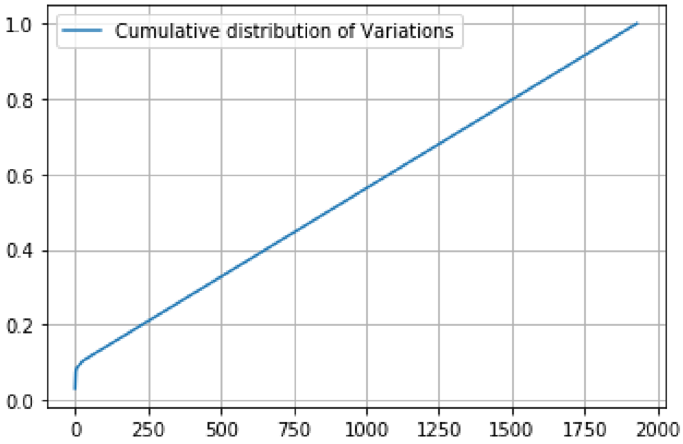




In [34]:

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

```
[0.03013183 0.05320151 0.07250471 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

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

1. **One hot Encoding**
2. **Response coding**

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

In [0]:

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

In [36]:

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

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

In [0]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [0]:

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

In [39]:

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

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding me
thod. The shape of Variation feature: (2124, 1961)
```

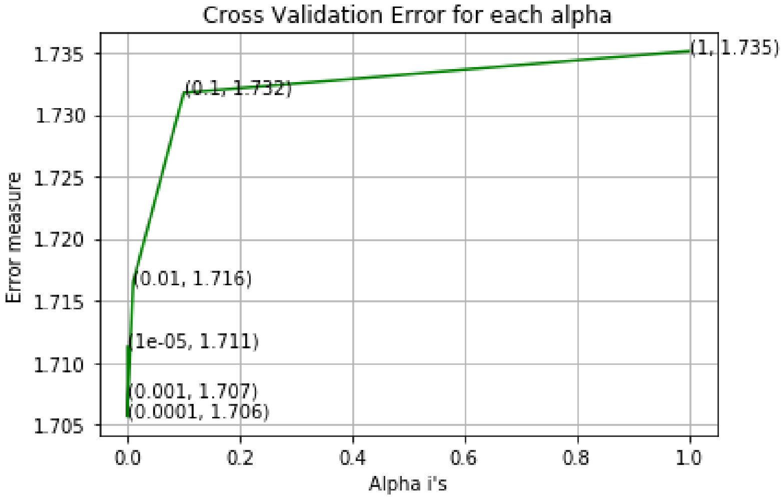
Q10. How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

In [40]:

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

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

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

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


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

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

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

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


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

predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss
(y_train, predict_y, labels=clf.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: 1.7112945679883376
For values of alpha =  0.0001 The log loss is: 1.7056528751549827
For values of alpha =  0.001 The log loss is: 1.7073505483766969
For values of alpha =  0.01 The log loss is: 1.716485097841659
For values of alpha =  0.1 The log loss is: 1.731745828537758
For values of alpha =  1 The log loss is: 1.7351071540708791
```



```
For values of best alpha =  0.0001 The train log loss is: 0.6820249702482761
For values of best alpha =  0.0001 The cross validation log loss is: 1.7056528751549827
For values of best alpha =  0.0001 The test log loss is: 1.7126212682423676
```

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 [41]:

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

```
Q12. How many data points are covered by total  1931  genes in test and cross validation
data sets?
Ans
1. In test data 70 out of 665 : 10.526315789473683
```

```
1. In test data 70 out of 665 : 10.526315789473683
2. In cross validation data 58 out of  532 : 10.902255639097744
```

3.2.3 Univariate Analysis on Text Feature

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

In [0]:

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


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

In [0]:

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

In [44]:

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

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

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


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

```
Total number of unique words in train data : 53788
```

In [0]:

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

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


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

In [0]:

```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
test_text_feature_responseCoding  = get_text_responsecoding(test_df)
cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)
```

In [0]:

```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1


train_text_feature_responseCoding = (train_text_feature_responseCoding.T/train_text_featu
re_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_feature_
responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_respon
seCoding.sum(axis=1)).T
```

In [0]:

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

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

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

In [0]:

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

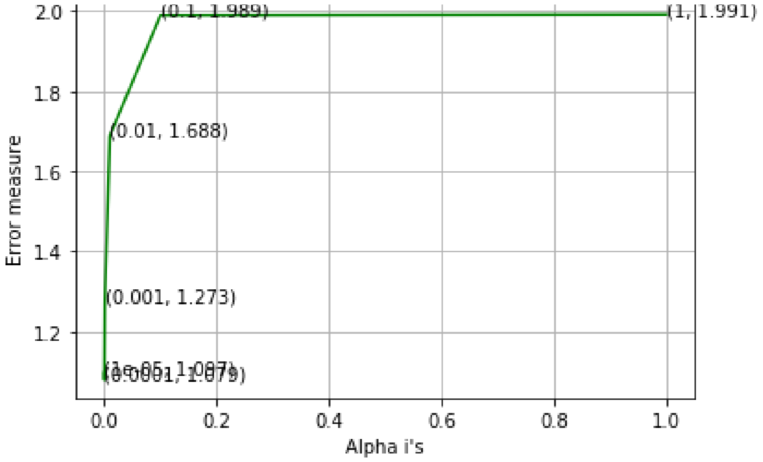
In [0]:

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

In [167]:

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

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

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

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


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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.0965697114292687
For values of alpha =  0.0001 The log loss is: 1.079388021338995
For values of alpha =  0.001 The log loss is: 1.2729612301133522
For values of alpha =  0.01 The log loss is: 1.6881296177059817
For values of alpha =  0.1 The log loss is: 1.9890543934194653
For values of alpha =  1 The log loss is: 1.9912395970364092
```



```
For values of best alpha =  0.0001 The train log loss is: 0.6832108649618762
For values of best alpha =  0.0001 The cross validation log loss is: 1.079388021338995
For values of best alpha =  0.0001 The test log loss is: 1.1354939788133145
```

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

Ans. Yes, it seems like!

In [0]:

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

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

In [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 appeared in train dat
a")
```

```
97.34 % of word of test data appeared in train data
98.709 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [0]:

```
#Data preparation for ML models.

#Misc. functionns for ML models


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

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

```
    plot_confusion_matrix(test_y, pred_y)
```

In [0]:

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

In [0]:

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

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

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

    word_present = 0
    for i,v in enumerate(indices):
        if (v < fea1_len):
            word = gene_vec.get_feature_names()[v]
            yes_no = True if word == gene else False
            if yes_no:
                word_present += 1
                print(i, "Gene feature [{}] present in test data point [{}]".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 point [{}]".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, "are present in query point")
```

Stacking the three types of features

In [0]:

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

train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_onehotCoding)).tocsr()
```

```
train_y = np.array(list(train_df['Class']))

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

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


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

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

In [58]:

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

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 55982)
(number of data points * number of features) in test data =  (665, 55982)
(number of data points * number of features) in cross validation data = (532, 55982)
```

In [59]:

```
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

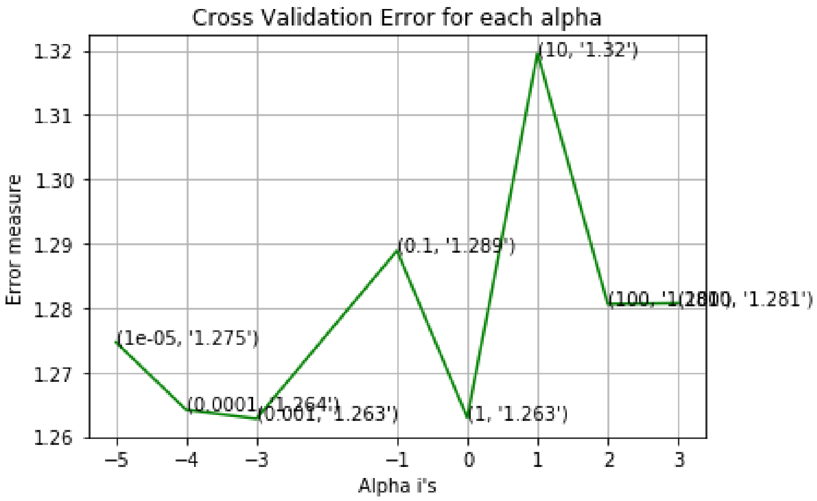
4.1.1.1. Hyper parameter tuning

In [60]:

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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)

# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algorithm-1/
# -----------------------


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

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


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


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

```
for alpha = 1e-05
Log Loss : 1.27462869537422
for alpha = 0.0001
Log Loss : 1.2641269558632222
for alpha = 0.001
```

```
Log Loss : 1.262752362314625
for alpha = 0.1
Log Loss : 1.2888775749625812
for alpha = 1
Log Loss : 1.2629052834081718
for alpha = 10
Log Loss : 1.3195078007163572
for alpha = 100
Log Loss : 1.2805705651054455
for alpha = 1000
Log Loss : 1.2807113585054852
```



```
For values of best alpha =  0.001 The train log loss is: 0.8006127313885127
For values of best alpha =  0.001 The cross validation log loss is: 1.262752362314625
For values of best alpha =  0.001 The test log loss is: 1.2609069121784091
```

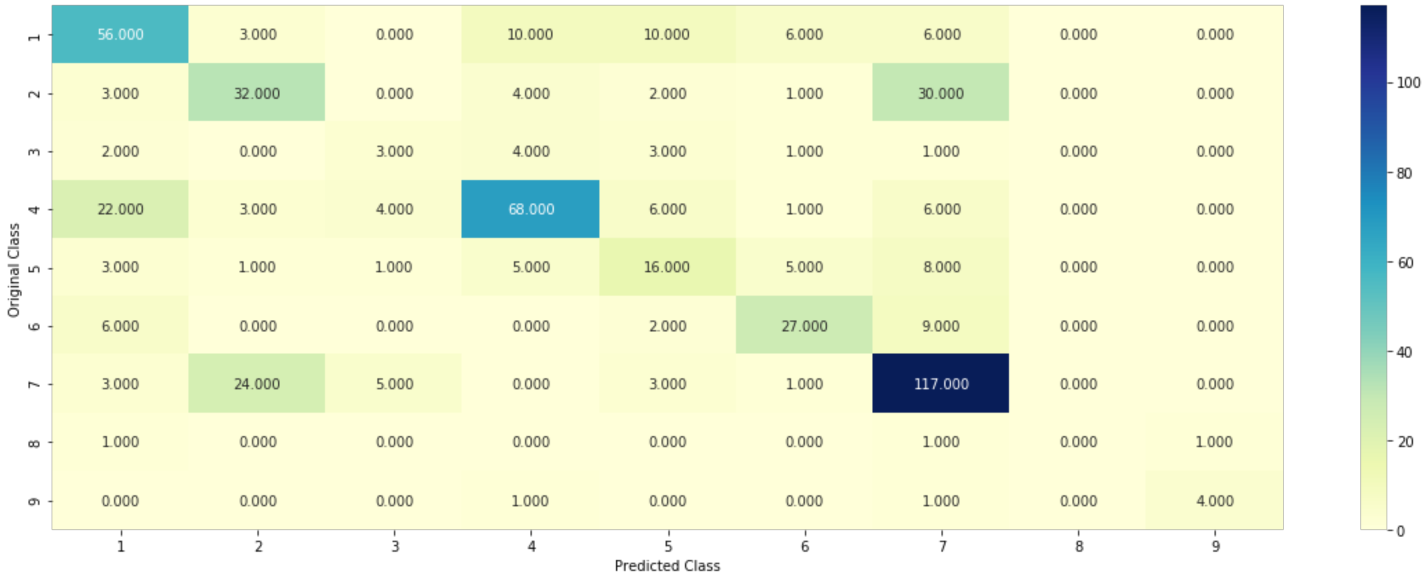
4.1.1.2. Testing the model with best hyper paramters

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

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

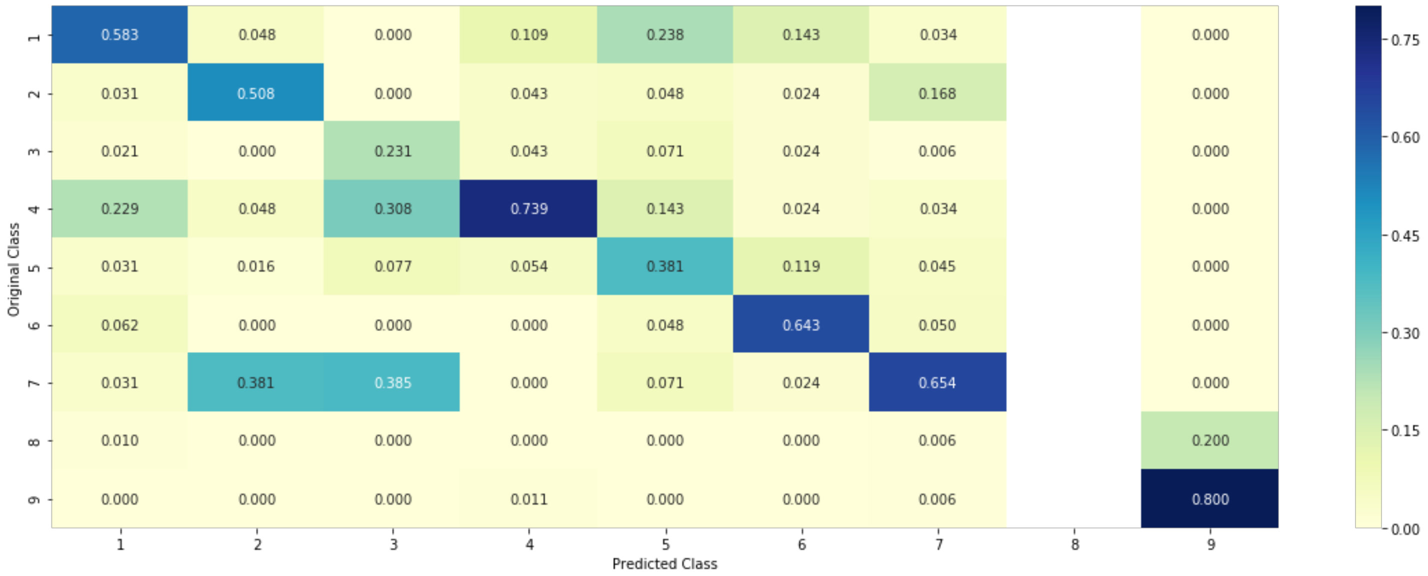
```
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estimates
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotCo
ding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

```
Log Loss : 1.262752362314625
Number of missclassified point : 0.39285714285714285
-------------------- Confusion matrix --------------------
```



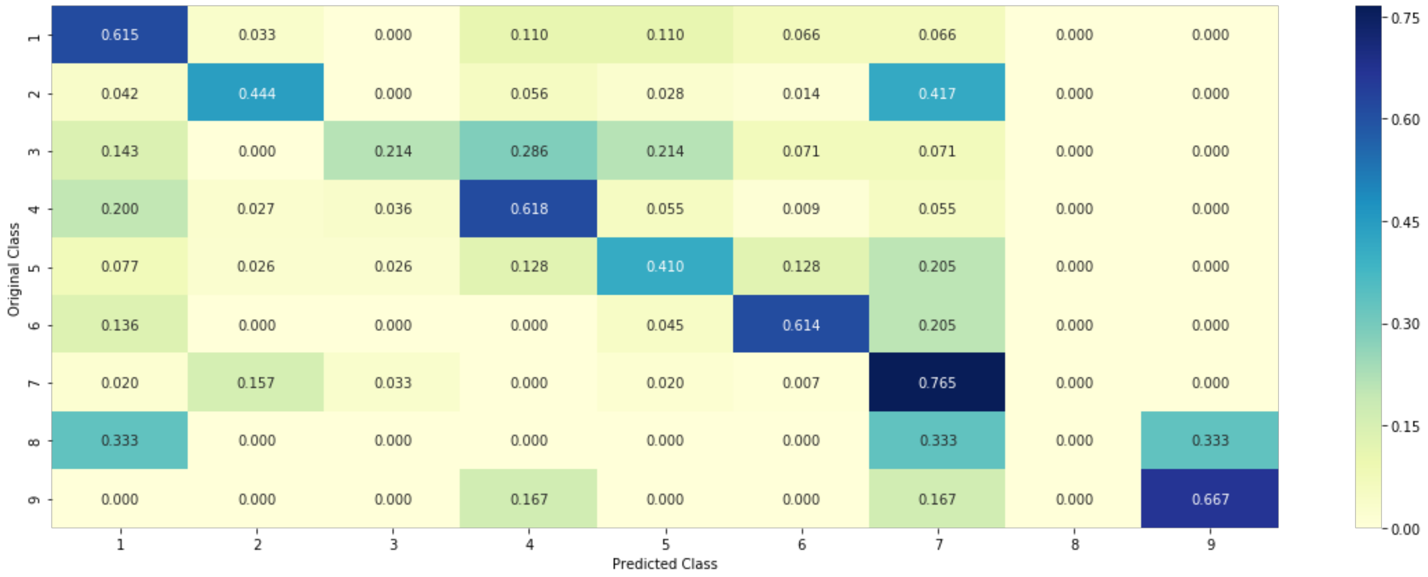
	1	2	3	4	5	6	7	8	9
1	56.000	3.000	0.000	10.000	10.000	6.000	6.000	0.000	0.000
2	3.000	32.000	0.000	4.000	2.000	1.000	30.000	0.000	0.000
3	2.000	0.000	3.000	4.000	3.000	1.000	1.000	0.000	0.000
4	22.000	3.000	4.000	68.000	6.000	1.000	6.000	0.000	0.000
5	3.000	1.000	1.000	5.000	16.000	5.000	8.000	0.000	0.000
6	6.000	0.000	0.000	0.000	2.000	27.000	9.000	0.000	0.000
7	3.000	24.000	5.000	0.000	3.000	1.000	117.000	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	1.000
9	0.000	0.000	0.000	1.000	0.000	0.000	1.000	0.000	4.000

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



	1	2	3	4	5	6	7	8	9
1	0.583	0.048	0.000	0.109	0.238	0.143	0.034		0.000
2	0.031	0.508	0.000	0.043	0.048	0.024	0.168		0.000
3	0.021	0.000	0.231	0.043	0.071	0.024	0.006		0.000
4	0.229	0.048	0.308	0.739	0.143	0.024	0.034		0.000
5	0.031	0.016	0.077	0.054	0.381	0.119	0.045		0.000
6	0.062	0.000	0.000	0.000	0.048	0.643	0.050		0.000
7	0.031	0.381	0.385	0.000	0.071	0.024	0.654		0.000
8	0.010	0.000	0.000	0.000	0.000	0.000	0.006		0.200
9	0.000	0.000	0.000	0.011	0.000	0.000	0.006		0.800

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



	1	2	3	4	5	6	7	8	9
1	0.615	0.033	0.000	0.110	0.110	0.066	0.066	0.000	0.000
2	0.042	0.444	0.000	0.056	0.028	0.014	0.417	0.000	0.000
3	0.143	0.000	0.214	0.286	0.214	0.071	0.071	0.000	0.000
4	0.200	0.027	0.036	0.618	0.055	0.009	0.055	0.000	0.000
5	0.077	0.026	0.026	0.128	0.410	0.128	0.205	0.000	0.000
6	0.136	0.000	0.000	0.000	0.045	0.614	0.205	0.000	0.000
7	0.020	0.157	0.033	0.000	0.020	0.007	0.765	0.000	0.000
8	0.333	0.000	0.000	0.000	0.000	0.000	0.333	0.000	0.333
9	0.000	0.000	0.000	0.167	0.000	0.000	0.167	0.000	0.667

4.1.1.3. Feature Importance, Incorrectly classified point

In [0]:

```
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_onehotCodin
g[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)
```

4.1.1.4. Feature Importance, correctly classified point

In [63]:

```
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_onehotCodin
g[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 : 4
Predicted Class Probabilities: [[0.0712 0.0741 0.0092 0.6756 0.0353 0.0314 0.0946 0.0045
0.004 ]]
Actual Class : 4
--------------------------------------------------
12 Text feature [proteins] present in test data point [True]
13 Text feature [protein] present in test data point [True]
14 Text feature [activity] present in test data point [True]
15 Text feature [experiments] present in test data point [True]
17 Text feature [mammalian] present in test data point [True]
18 Text feature [acid] present in test data point [True]
19 Text feature [loss] present in test data point [True]
20 Text feature [function] present in test data point [True]
21 Text feature [pten] present in test data point [True]
22 Text feature [whereas] present in test data point [True]
23 Text feature [determined] present in test data point [True]
24 Text feature [shown] present in test data point [True]
25 Text feature [results] present in test data point [True]
26 Text feature [ability] present in test data point [True]
27 Text feature [indicated] present in test data point [True]
29 Text feature [amino] present in test data point [True]
30 Text feature [described] present in test data point [True]
31 Text feature [type] present in test data point [True]
32 Text feature [levels] present in test data point [True]
33 Text feature [whether] present in test data point [True]
34 Text feature [related] present in test data point [True]
36 Text feature [two] present in test data point [True]
37 Text feature [abrogate] present in test data point [True]
38 Text feature [functions] present in test data point [True]
39 Text feature [indicate] present in test data point [True]
40 Text feature [expressed] present in test data point [True]
41 Text feature [also] present in test data point [True]
42 Text feature [important] present in test data point [True]
43 Text feature [either] present in test data point [True]
44 Text feature [retained] present in test data point [True]
45 Text feature [wild] present in test data point [True]
```

```
46 Text feature [bind] present in test data point [True]
47 Text feature [catalytic] present in test data point [True]
48 Text feature [see] present in test data point [True]
49 Text feature [containing] present in test data point [True]
50 Text feature [expression] present in test data point [True]
51 Text feature [reduced] present in test data point [True]
55 Text feature [30] present in test data point [True]
56 Text feature [microscopy] present in test data point [True]
57 Text feature [thus] present in test data point [True]
58 Text feature [amount] present in test data point [True]
59 Text feature [tagged] present in test data point [True]
60 Text feature [missense] present in test data point [True]
62 Text feature [mutations] present in test data point [True]
63 Text feature [standard] present in test data point [True]
64 Text feature [vitro] present in test data point [True]
65 Text feature [vivo] present in test data point [True]
66 Text feature [contribute] present in test data point [True]
67 Text feature [using] present in test data point [True]
68 Text feature [tensin] present in test data point [True]
69 Text feature [although] present in test data point [True]
70 Text feature [analyzed] present in test data point [True]
71 Text feature [incubated] present in test data point [True]
73 Text feature [purified] present in test data point [True]
75 Text feature [determine] present in test data point [True]
76 Text feature [percentage] present in test data point [True]
77 Text feature [yielded] present in test data point [True]
78 Text feature [sds] present in test data point [True]
79 Text feature [lower] present in test data point [True]
80 Text feature [indicates] present in test data point [True]
81 Text feature [suggest] present in test data point [True]
82 Text feature [system] present in test data point [True]
83 Text feature [trisphosphate] present in test data point [True]
84 Text feature [tris] present in test data point [True]
85 Text feature [performed] present in test data point [True]
86 Text feature [may] present in test data point [True]
89 Text feature [caenorhabditis] present in test data point [True]
90 Text feature [generated] present in test data point [True]
91 Text feature [cells] present in test data point [True]
92 Text feature [transfected] present in test data point [True]
93 Text feature [associated] present in test data point [True]
94 Text feature [functional] present in test data point [True]
95 Text feature [previously] present in test data point [True]
96 Text feature [marked] present in test data point [True]
97 Text feature [essential] present in test data point [True]
98 Text feature [buffer] present in test data point [True]
99 Text feature [three] present in test data point [True]
Out of the top  100  features  77 are present in query point
```

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

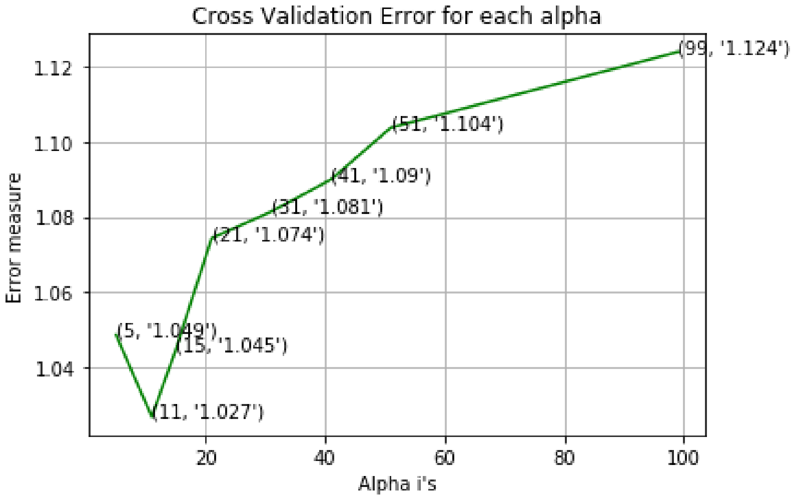
In [64]:

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

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

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


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


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

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


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

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

```
for alpha = 5
Log Loss : 1.0485171536685614
for alpha = 11
Log Loss : 1.0266737917551572
for alpha = 15
Log Loss : 1.04451433797169
for alpha = 21
Log Loss : 1.0742528185503082
for alpha = 31
Log Loss : 1.0813710581610474
for alpha = 41
```

```
Log Loss : 1.0899426586058885
for alpha = 51
Log Loss : 1.1037369636849341
for alpha = 99
Log Loss : 1.1239145158200101
```



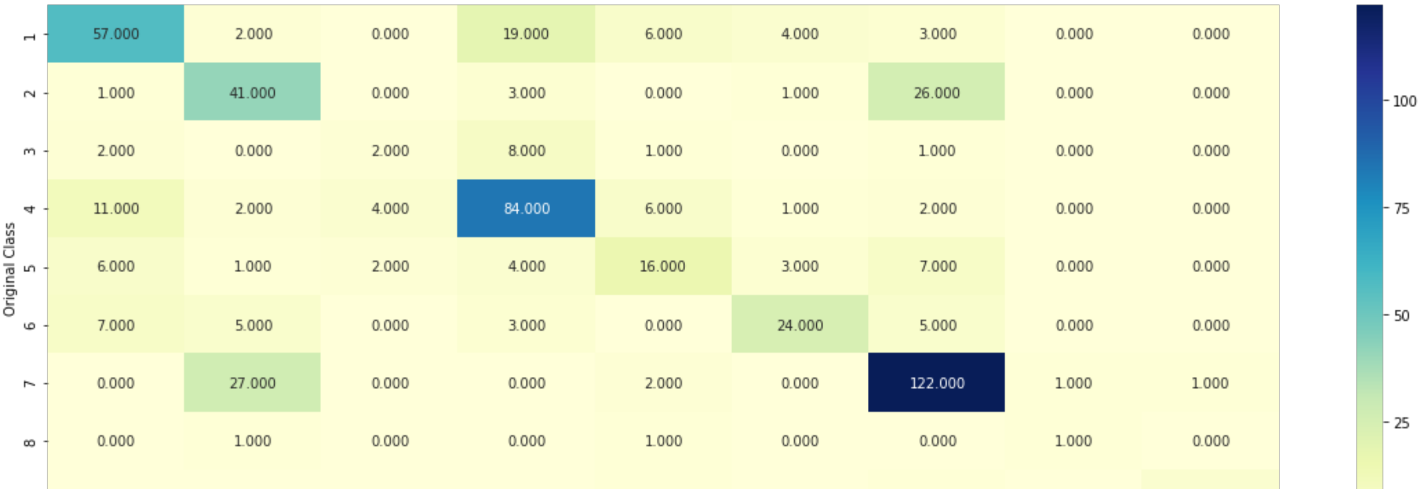
```
For values of best alpha =  11 The train log loss is: 0.5979819360437533
For values of best alpha =  11 The cross validation log loss is: 1.0266737917551572
For values of best alpha =  11 The test log loss is: 1.080940132999745
```

4.2.2. Testing the model with best hyper paramters

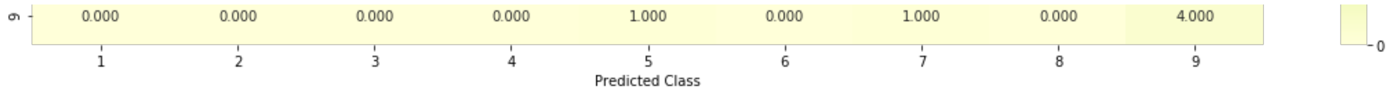
In [65]:

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

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

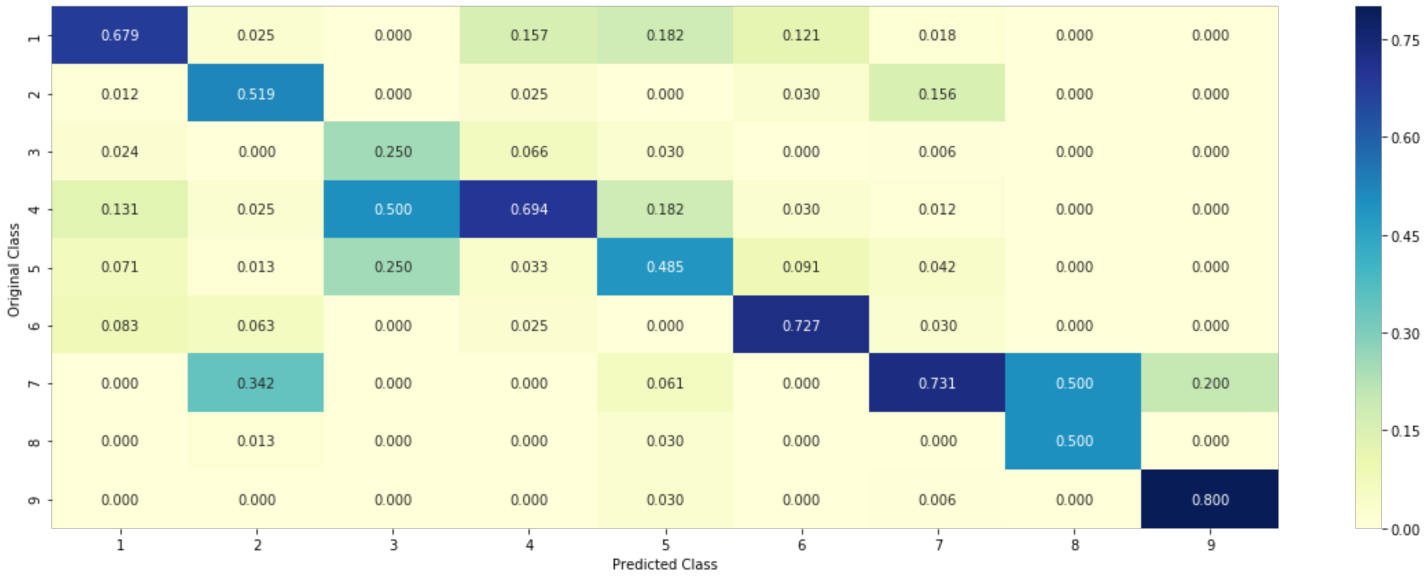
```
Log loss : 1.0266737917551572
Number of mis-classified points : 0.34022556390977443
-------------------- Confusion matrix --------------------
```



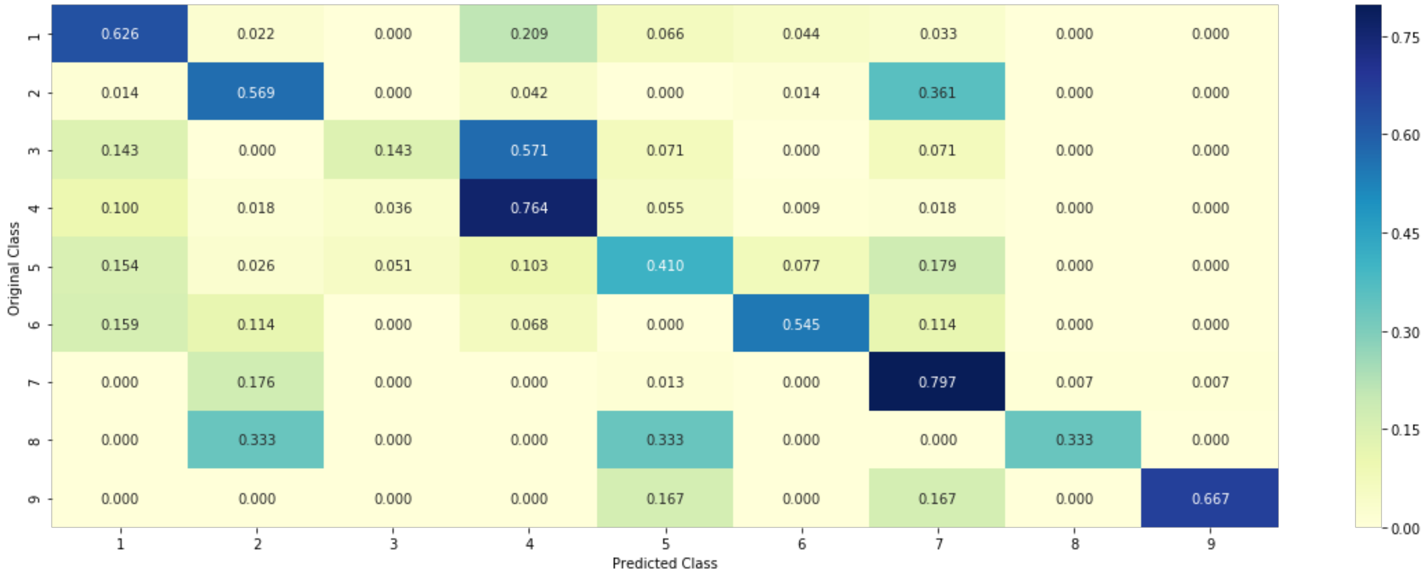
1	57.000	2.000	0.000	19.000	6.000	4.000	3.000	0.000	0.000
2	1.000	41.000	0.000	3.000	0.000	1.000	26.000	0.000	0.000
3	2.000	0.000	2.000	8.000	1.000	0.000	1.000	0.000	0.000
4	11.000	2.000	4.000	84.000	6.000	1.000	2.000	0.000	0.000
5	6.000	1.000	2.000	4.000	16.000	3.000	7.000	0.000	0.000
6	7.000	5.000	0.000	3.000	0.000	24.000	5.000	0.000	0.000
7	0.000	27.000	0.000	0.000	2.000	0.000	122.000	1.000	1.000
8	0.000	1.000	0.000	0.000	1.000	0.000	0.000	1.000	0.000



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



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



4.2.3.Sample Query point -1

In [66]:

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

```
Predicted Class : 5
Actual Class : 7
The  11  nearest neighbours of the test points belongs to classes [2 2 1 2 2 1 1 2 2 4 2]
Fequency of nearest points : Counter({2: 7, 1: 3, 4: 1})
```

4.2.4. Sample Query Point-2

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 = 100

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

```
Predicted Class : 4
Actual Class : 4
the k value for knn is 11 and the nearest neighbours of the test points belongs to classe
s [4 4 4 4 4 4 4 4 4 4 6]
Fequency of nearest points : Counter({4: 10, 6: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

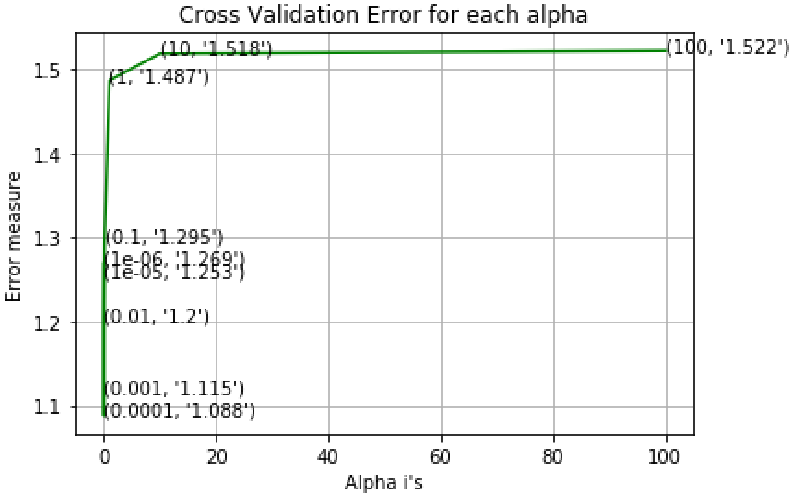
4.3.1.1. Hyper paramter tuning

In [68]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/skl
earn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
ue, max_iter=None, tol=None,

alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', rand
om_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-1
5))
    # to avoid rounding error while multiplying probabilites we use log-probability estim
ates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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

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

```
for alpha = 1e-06
Log Loss : 1.269431817355115
for alpha = 1e-05
Log Loss : 1.2528557665636915
for alpha = 0.0001
Log Loss : 1.0883675258271188
for alpha = 0.001
Log Loss : 1.1149346072436233
for alpha = 0.01
Log Loss : 1.1997768337287629
for alpha = 0.1
Log Loss : 1.2949341011017024
for alpha = 1
Log Loss : 1.4866663147731465
for alpha = 10
Log Loss : 1.5182900373810153
for alpha = 100
Log Loss : 1.5217005724741313
```



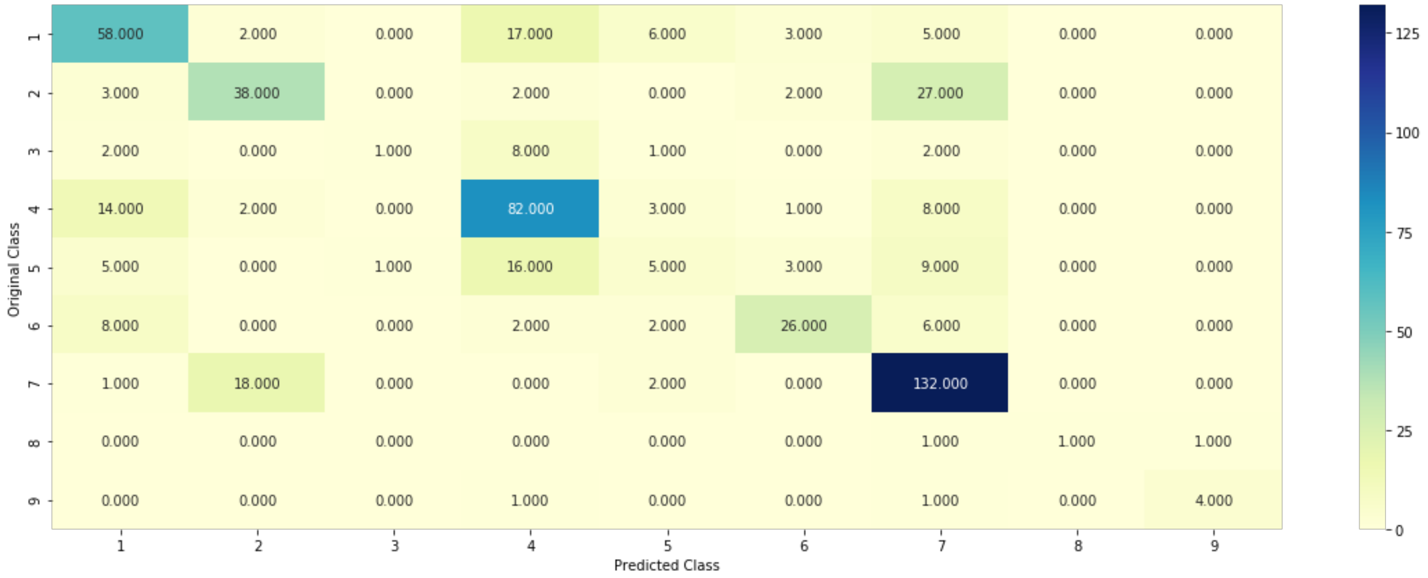
```
For values of best alpha =  0.0001 The train log loss is: 0.4715250588275571
For values of best alpha =  0.0001 The cross validation log loss is: 1.0883675258271188
For values of best alpha =  0.0001 The test log loss is: 1.0024889219807271
```

4.3.1.2. Testing the model with best hyper paramters

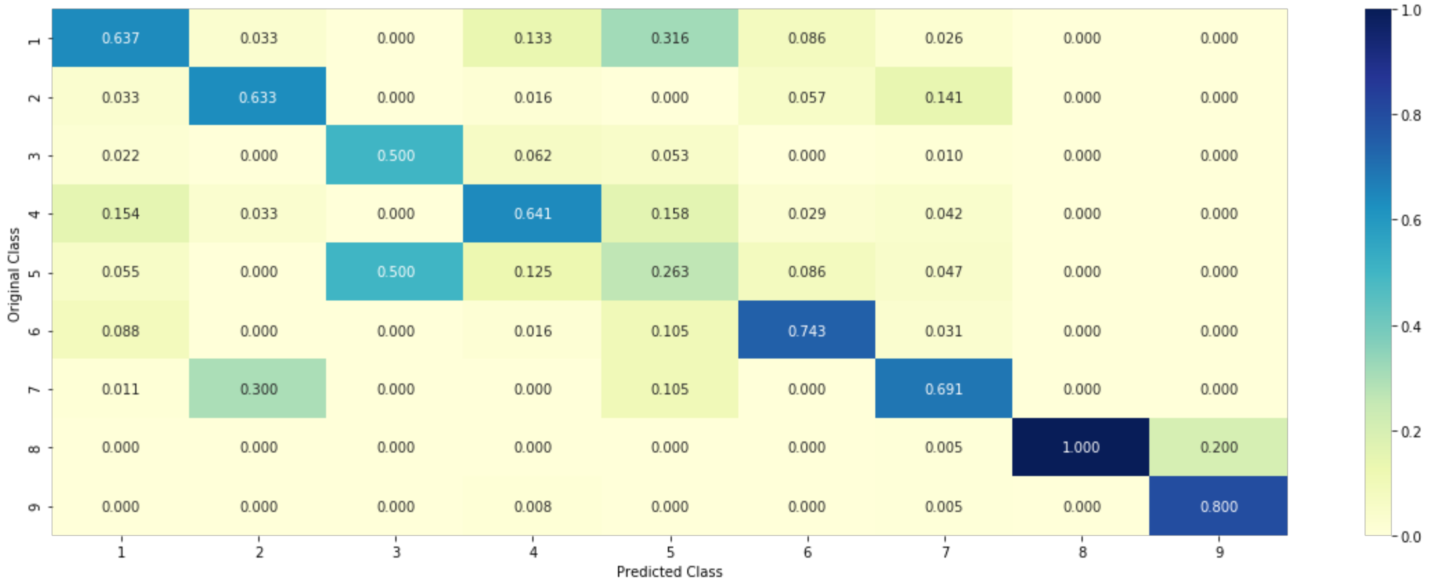
In [69]:

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

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

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

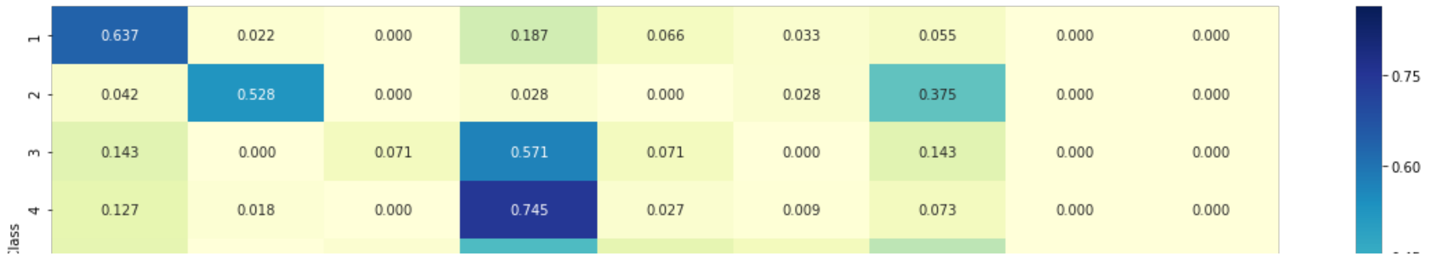
```
Log loss : 1.0883675258271188
Number of mis-classified points : 0.34774436090225563
-------------------- Confusion matrix --------------------
```

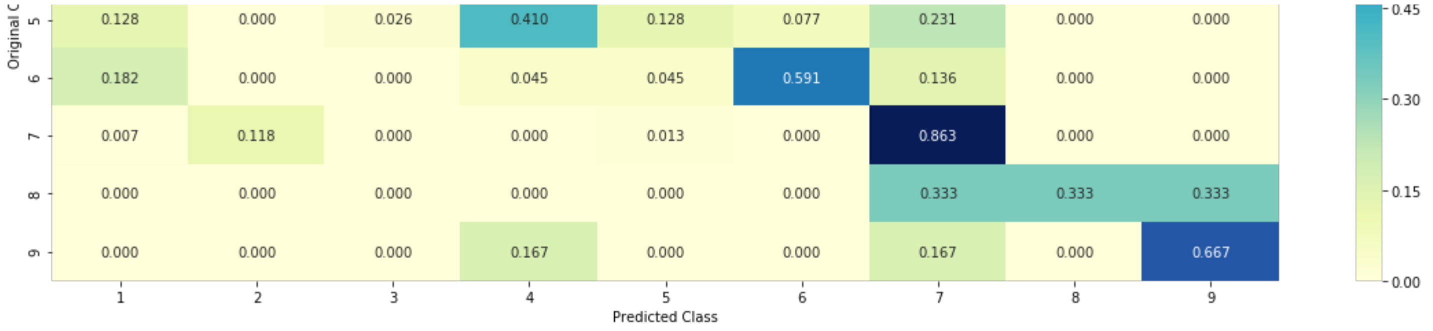


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





4.3.1.3. Feature Importance

In [0]:

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

4.3.1.3.1. Correctly Classified point

In [71]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss
='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCodin
g[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 : 2
Predicted Class Probabilities: [[0.1593 0.3592 0.0461 0.1465 0.0501 0.0546 0.1673 0.0081
0.0088]]
Actual Class : 7
--------------------------------------------------
281 Text feature [surgical] present in test data point [True]
352 Text feature [ffpe] present in test data point [True]
357 Text feature [hmga1] present in test data point [True]
488 Text feature [director] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

4.3.1.3.2. Incorrectly Classified point

In [72]:

```
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_onehotCodin
g[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 : 4
Predicted Class Probabilities: [[6.000e-04 3.000e-03 1.300e-03 9.855e-01 2.500e-03 4.000e
-04 1.600e-03
  3.000e-03 2.100e-03]]
Actual Class : 4
--------------------------------------------------
231 Text feature [mobilization] present in test data point [True]
261 Text feature [come] present in test data point [True]
284 Text feature [young] present in test data point [True]
294 Text feature [cul] present in test data point [True]
310 Text feature [homologues] present in test data point [True]
340 Text feature [hereditary] present in test data point [True]
343 Text feature [senescence] present in test data point [True]
353 Text feature [gm] present in test data point [True]
387 Text feature [recombination] present in test data point [True]
391 Text feature [tagged] present in test data point [True]
403 Text feature [analysed] present in test data point [True]
420 Text feature [microscopy] present in test data point [True]
427 Text feature [instability] present in test data point [True]
469 Text feature [araf] present in test data point [True]
471 Text feature [devoid] present in test data point [True]
Out of the top  500  features  15 are present in query point
```

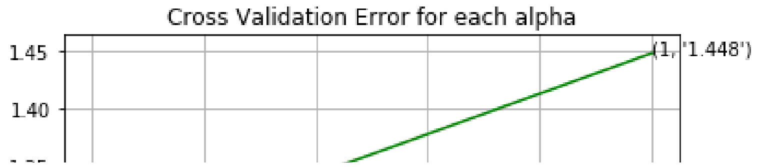
4.3.2. Without Class balancing

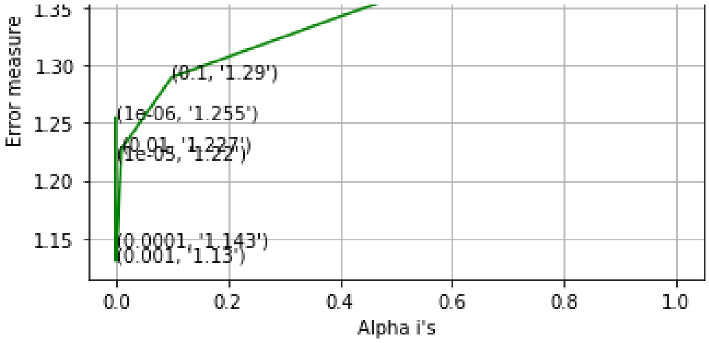
4.3.2.1. Hyper paramter tuning

In [73]:

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

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

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



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

alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-1
5))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

```
for alpha = 1e-06
Log Loss : 1.2545931142902462
for alpha = 1e-05
Log Loss : 1.219931398732453
for alpha = 0.0001
Log Loss : 1.1431835335188274
for alpha = 0.001
Log Loss : 1.1304938048364348
for alpha = 0.01
Log Loss : 1.2272176791879814
for alpha = 0.1
Log Loss : 1.2897503962880008
for alpha = 1
Log Loss : 1.447603097810841
```





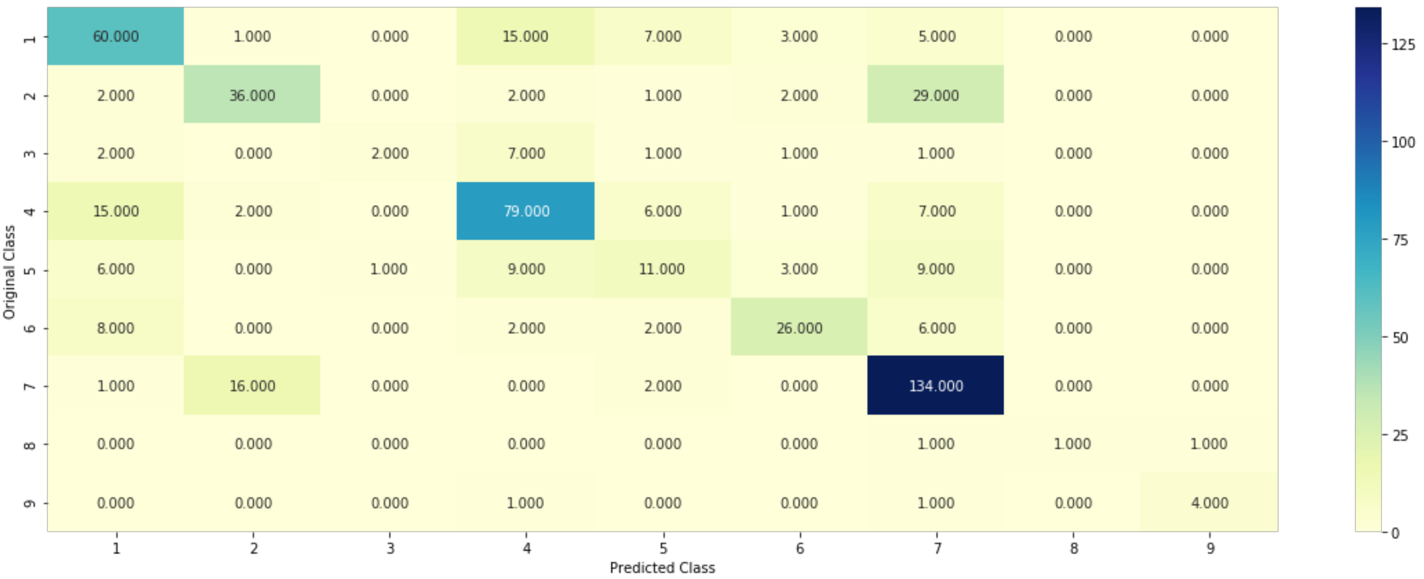
```
For values of best alpha =  0.001 The train log loss is: 0.48934193634190754
For values of best alpha =  0.001 The cross validation log loss is: 1.1304938048364348
For values of best alpha =  0.001 The test log loss is: 1.0070290551165213
```

4.3.2.2. Testing model with best hyper parameters

In [74]:

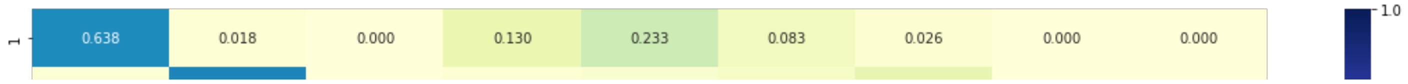
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/skl
earn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=’hinge’, penalty=’l2’, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
ue, max_iter=None, tol=None,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

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

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

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

```
Log loss : 1.1304938048364348
Number of mis-classified points : 0.33646616541353386
-------------------- Confusion matrix --------------------
```

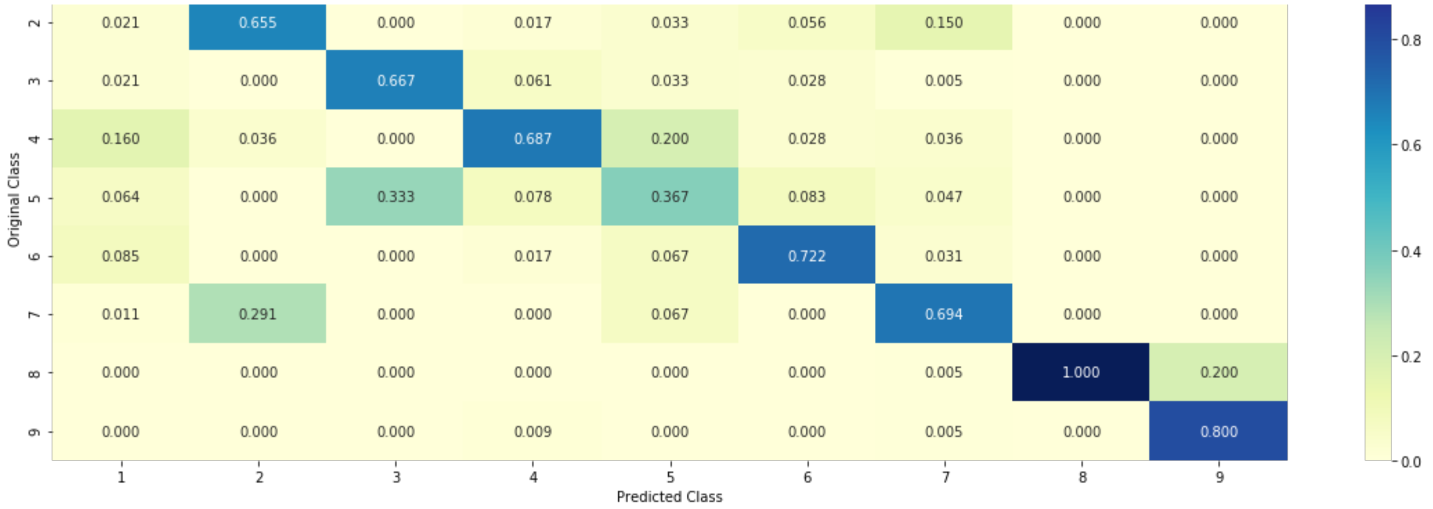


	1	2	3	4	5	6	7	8	9
1	60.000	1.000	0.000	15.000	7.000	3.000	5.000	0.000	0.000
2	2.000	36.000	0.000	2.000	1.000	2.000	29.000	0.000	0.000
3	2.000	0.000	2.000	7.000	1.000	1.000	1.000	0.000	0.000
4	15.000	2.000	0.000	79.000	6.000	1.000	7.000	0.000	0.000
5	6.000	0.000	1.000	9.000	11.000	3.000	9.000	0.000	0.000
6	8.000	0.000	0.000	2.000	2.000	26.000	6.000	0.000	0.000
7	1.000	16.000	0.000	0.000	2.000	0.000	134.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	1.000	1.000	1.000
9	0.000	0.000	0.000	1.000	0.000	0.000	1.000	0.000	4.000

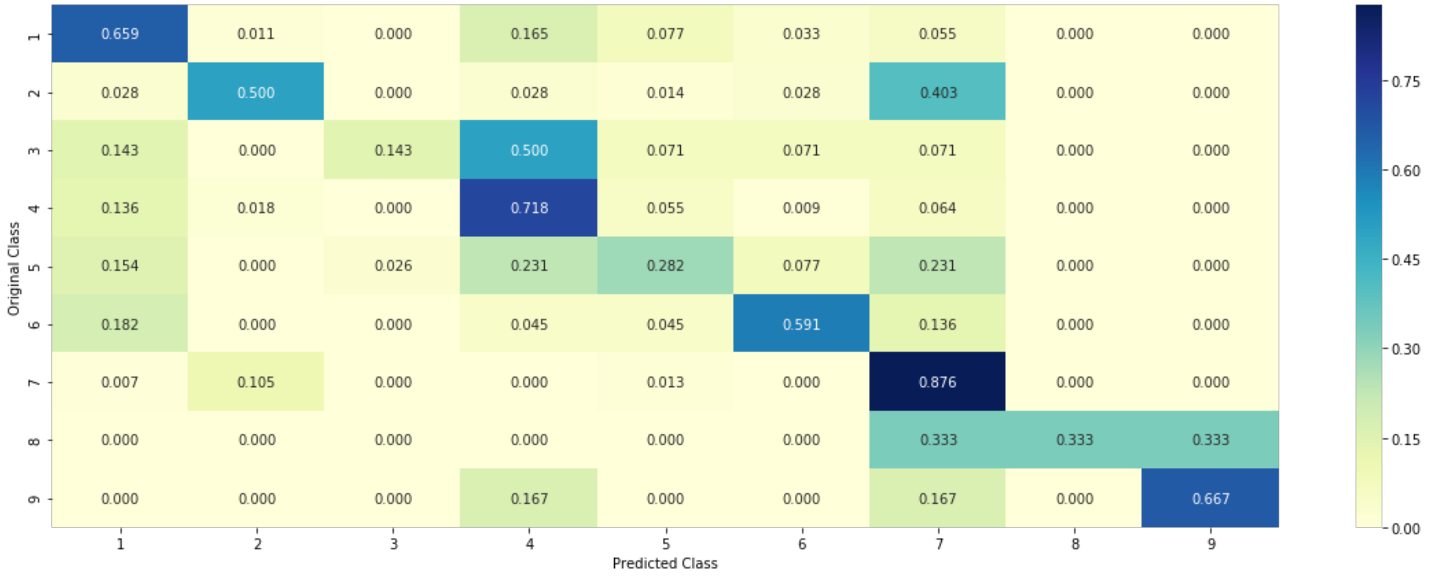
```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



1	0.638	0.018	0.000	0.130	0.233	0.083	0.026	0.000	0.000



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



4.3.2.3. Feature Importance, Correctly Classified point

In [75]:

```
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_onehotCodin
g[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 : 2
Predicted Class Probabilities: [[0.1545 0.4032 0.0223 0.1432 0.0477 0.0455 0.1697 0.0084
0.0054]]
Actual Class : 7
--------------------------------------------------
147 Text feature [surgical] present in test data point [True]
268 Text feature [hmga1] present in test data point [True]
277 Text feature [director] present in test data point [True]
393 Text feature [ffpe] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [76]:

```
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_onehotCodin
g[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 : 4
Predicted Class Probabilities: [[1.200e-03 7.000e-04 0.000e+00 9.942e-01 4.000e-04 1.000e
-04 9.000e-04
  2.600e-03 0.000e+00]]
Actual Class : 4
--------------------------------------------------
216 Text feature [come] present in test data point [True]
241 Text feature [microscopy] present in test data point [True]
284 Text feature [homologues] present in test data point [True]
288 Text feature [instability] present in test data point [True]
338 Text feature [tagged] present in test data point [True]
346 Text feature [analysed] present in test data point [True]
350 Text feature [suppressor] present in test data point [True]
376 Text feature [mammalian] present in test data point [True]
382 Text feature [pten] present in test data point [True]
386 Text feature [young] present in test data point [True]
390 Text feature [phosphatases] present in test data point [True]
400 Text feature [gm] present in test data point [True]
402 Text feature [subfraction] present in test data point [True]
411 Text feature [yeast] present in test data point [True]
427 Text feature [devoid] present in test data point [True]
453 Text feature [mis] present in test data point [True]
457 Text feature [senescence] present in test data point [True]
473 Text feature [germline] present in test data point [True]
481 Text feature [cul] present in test data point [True]
488 Text feature [recombination] present in test data point [True]
Out of the top  500  features  20 are present in query point
```

4.4. Linear Support Vector Machines

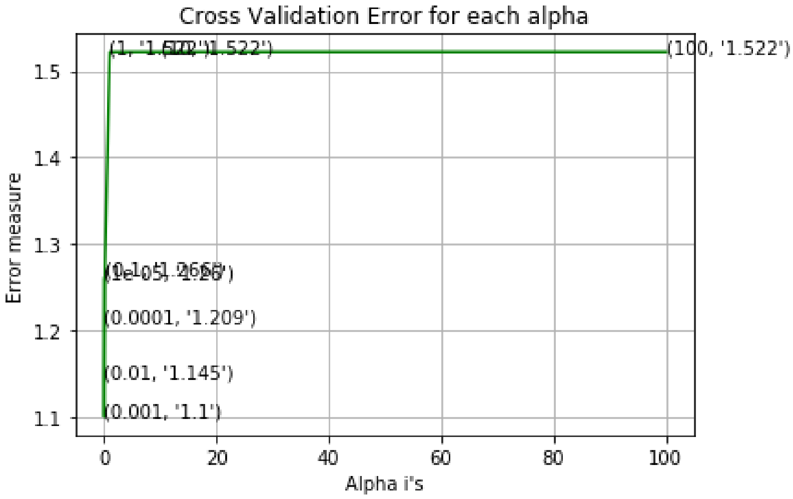
4.4.1. Hyper paramter tuning

In [77]:

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

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

# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/mat
hematical-derivation-copy-8/
# --------------------------------
# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

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

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


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

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

```
for C = 1e-05
Log Loss : 1.259969771427771
for C = 0.0001
Log Loss : 1.2093151787913652
for C = 0.001
Log Loss : 1.1001286434432196
for C = 0.01
Log Loss : 1.1445719905589886
for C = 0.1
Log Loss : 1.2659931540017813
for C = 1
Log Loss : 1.522089522050098
for C = 10
```

```
Log Loss : 1.5221905994346208
for C = 100
Log Loss : 1.5221905436825298
```



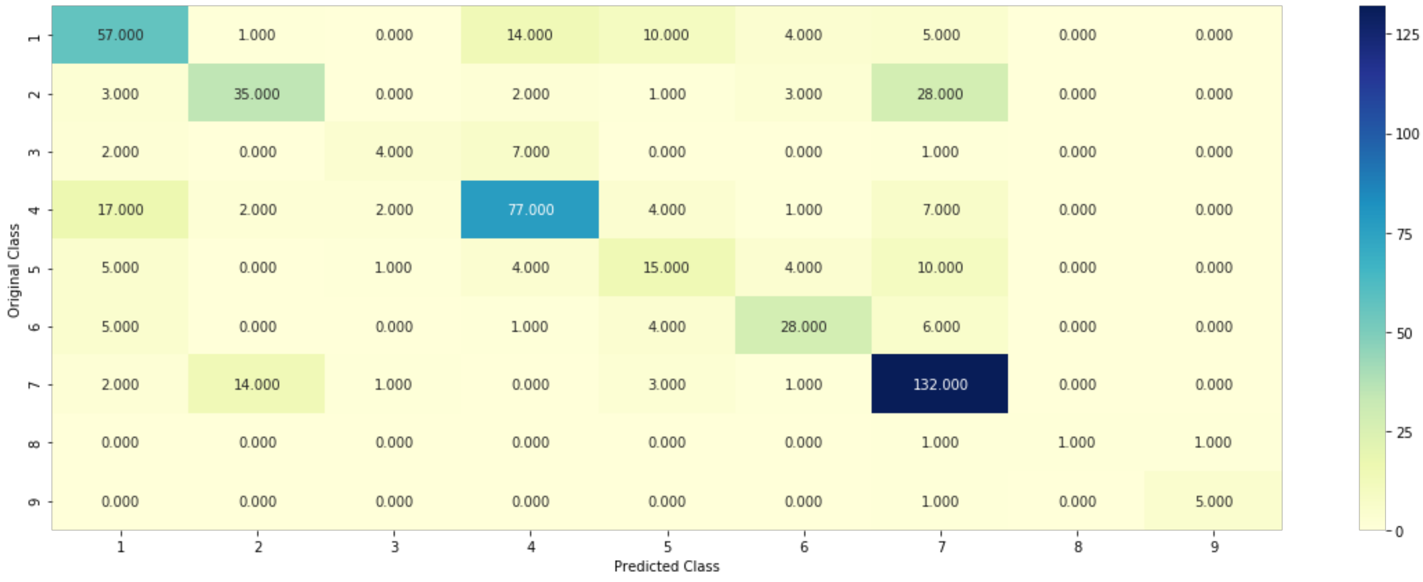
```
For values of best alpha =  0.001 The train log loss is: 0.529344633212878
For values of best alpha =  0.001 The cross validation log loss is: 1.1001286434432196
For values of best alpha =  0.001 The test log loss is: 1.073974608558865
```

4.4.2. Testing model with best hyper parameters

In [78]:

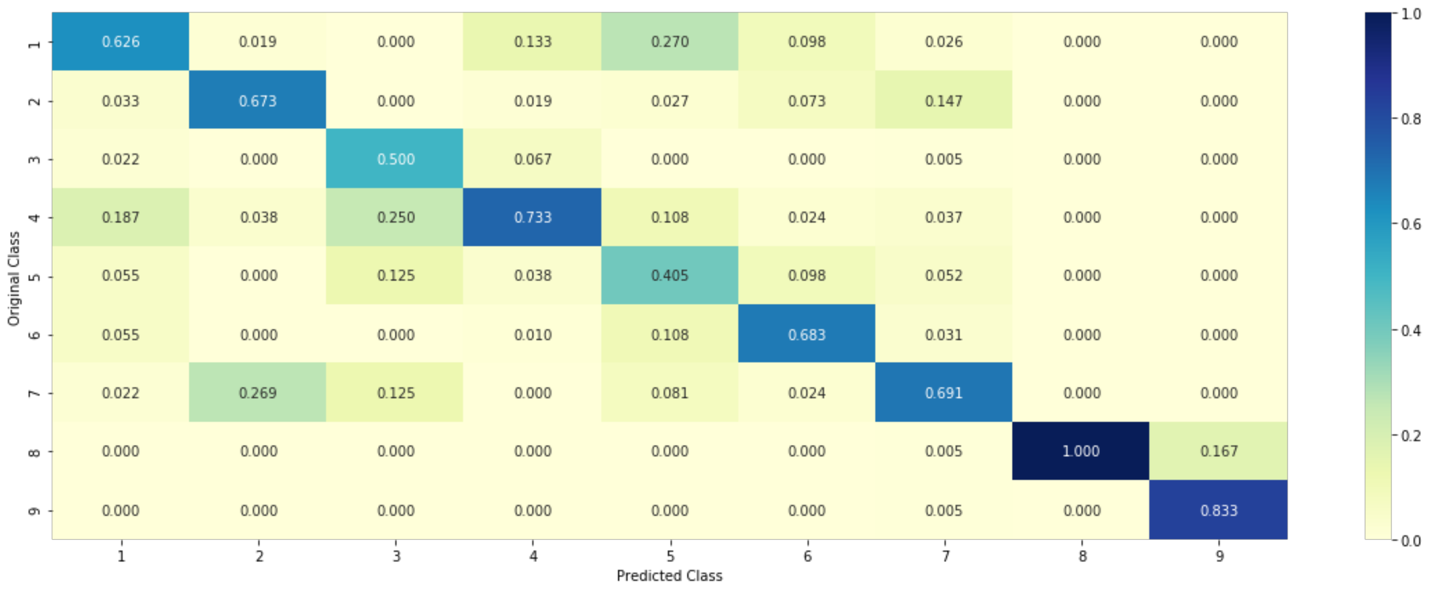
```
# read more about support vector machines with linear kernals here http://scikit-learn.or
g/stable/modules/generated/sklearn.svm.SVC.html

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



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

```
Log loss : 1.1001286434432196
Number of mis-classified points : 0.33458646616541354
-------------------- Confusion matrix --------------------
```

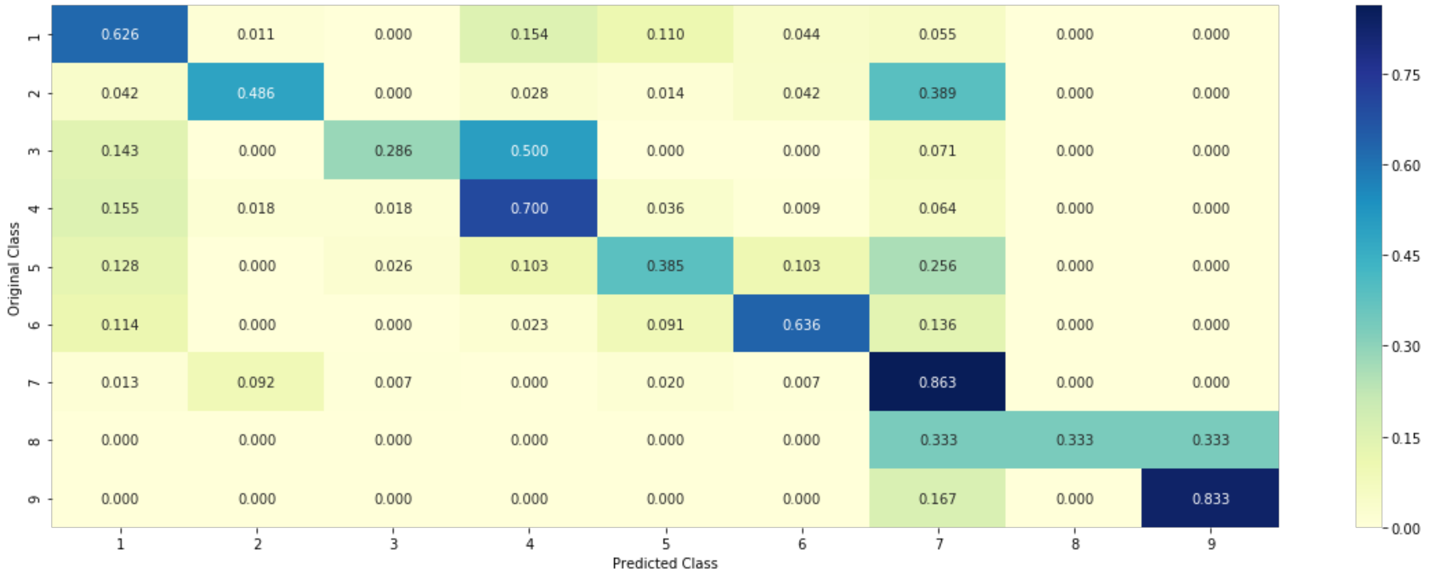


	1	2	3	4	5	6	7	8	9
1	57.000	1.000	0.000	14.000	10.000	4.000	5.000	0.000	0.000
2	3.000	35.000	0.000	2.000	1.000	3.000	28.000	0.000	0.000
3	2.000	0.000	4.000	7.000	0.000	0.000	1.000	0.000	0.000
4	17.000	2.000	2.000	77.000	4.000	1.000	7.000	0.000	0.000
5	5.000	0.000	1.000	4.000	15.000	4.000	10.000	0.000	0.000
6	5.000	0.000	0.000	1.000	4.000	28.000	6.000	0.000	0.000
7	2.000	14.000	1.000	0.000	3.000	1.000	132.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	1.000	1.000	1.000
9	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	5.000

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



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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [79]:

```
clf = SGDClassifier(alpha=0.001, 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_onehotCodin
g[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 : 4
Predicted Class Probabilities: [[0.0118 0.0106 0.0011 0.9407 0.0094 0.0032 0.018  0.0024
0.0028]]
Actual Class : 4
--------------------------------------------------
```

```
--------------------------------------------------
117 Text feature [mobilization] present in test data point [True]
241 Text feature [homologues] present in test data point [True]
242 Text feature [young] present in test data point [True]
245 Text feature [cul] present in test data point [True]
246 Text feature [gm] present in test data point [True]
247 Text feature [senescence] present in test data point [True]
266 Text feature [come] present in test data point [True]
300 Text feature [araf] present in test data point [True]
354 Text feature [csf] present in test data point [True]
360 Text feature [hereditary] present in test data point [True]
429 Text feature [analysed] present in test data point [True]
459 Text feature [tagged] 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 [80]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCodin
g[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 : 4
Predicted Class Probabilities: [[0.0118 0.0106 0.0011 0.9407 0.0094 0.0032 0.018  0.0024
0.0028]]
Actual Class : 4
--------------------------------------------------
117 Text feature [mobilization] present in test data point [True]
241 Text feature [homologues] present in test data point [True]
242 Text feature [young] present in test data point [True]
245 Text feature [cul] present in test data point [True]
246 Text feature [gm] present in test data point [True]
247 Text feature [senescence] present in test data point [True]
266 Text feature [come] present in test data point [True]
300 Text feature [araf] present in test data point [True]
354 Text feature [csf] present in test data point [True]
360 Text feature [hereditary] present in test data point [True]
429 Text feature [analysed] present in test data point [True]
459 Text feature [tagged] present in test data point [True]
Out of the top  500  features  12 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [81]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=No
ne, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=N
one, 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/ran
dom-forest-and-their-construction-2/
# --------------------------------

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

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

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

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2350521139520694
for n_estimators = 100 and max depth =  10
Log Loss : 1.1714918863154022
for n_estimators = 200 and max depth =  5
Log Loss : 1.2229344402680375
for n_estimators = 200 and max depth =  10
Log Loss : 1.1605611885459666
for n_estimators = 500 and max depth =  5
Log Loss : 1.2142532180226544
for n_estimators = 500 and max depth =  10
Log Loss : 1.155950594942328
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2140832045780883
```

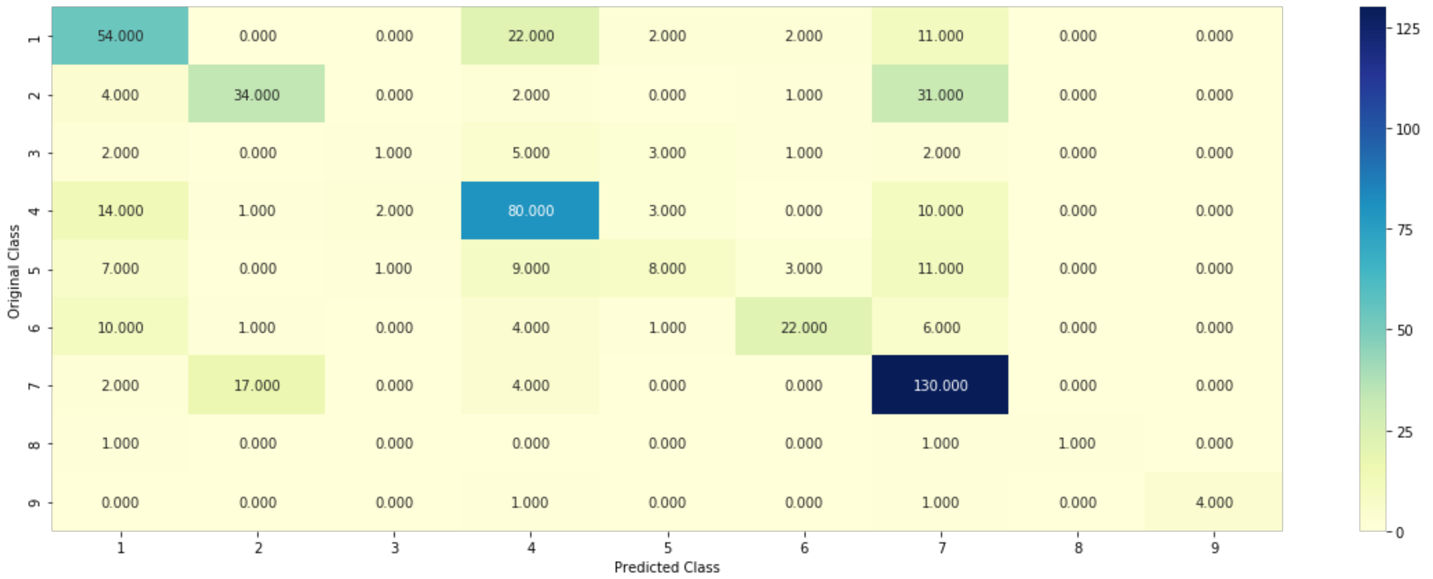
```
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1511635380850236
for n_estimators = 2000 and max depth =  5
Log Loss : 1.210780821347626
for n_estimators = 2000 and max depth =  10
Log Loss : 1.148660576788516
For values of best estimator =  2000 The train log loss is: 0.6196891753908043
For values of best estimator =  2000 The cross validation log loss is: 1.148660576788516
For values of best estimator =  2000 The test log loss is: 1.1302362322759756
```

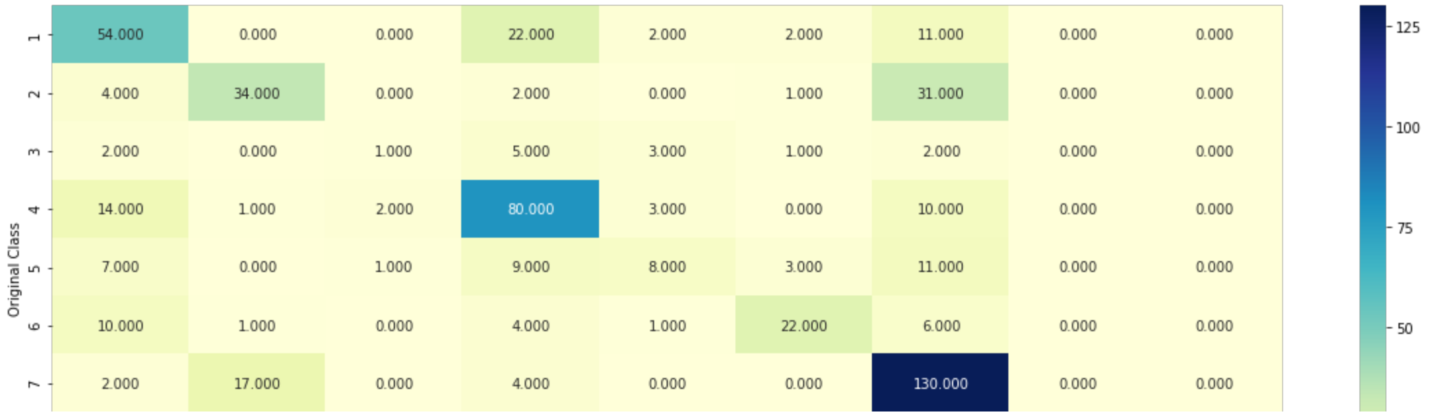
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

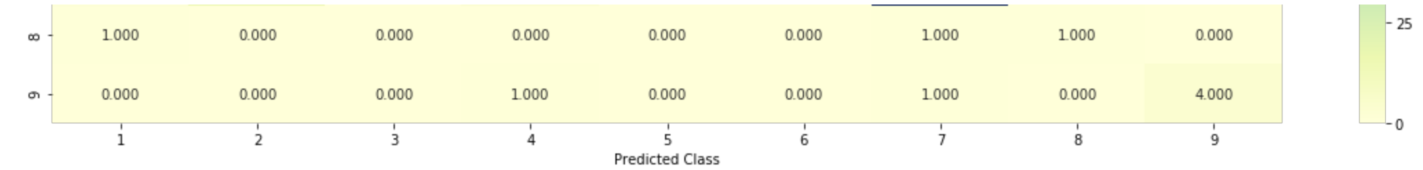
In [82]:

```
# --------------------------------
# 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)



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

```
Log loss : 1.148660576788516
Number of mis-classified points : 0.37218045112781956
-------------------- Confusion matrix --------------------
Log loss : 1.148660576788516
Number of mis-classified points : 0.37218045112781956
-------------------- Confusion matrix --------------------
```

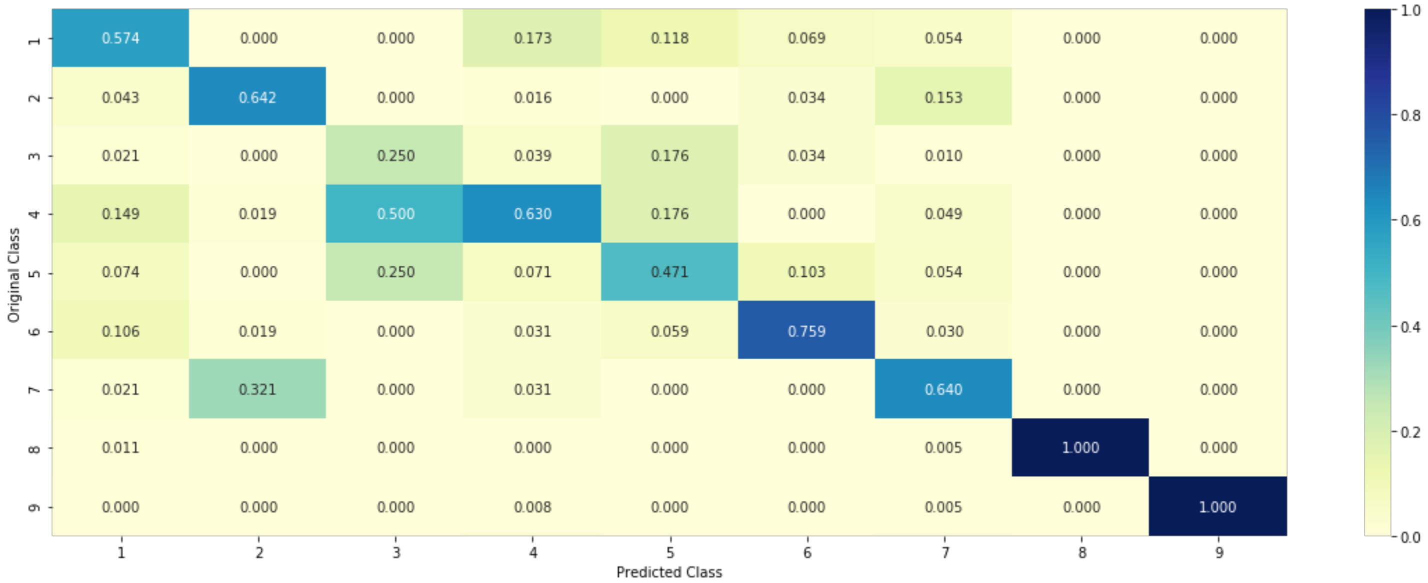


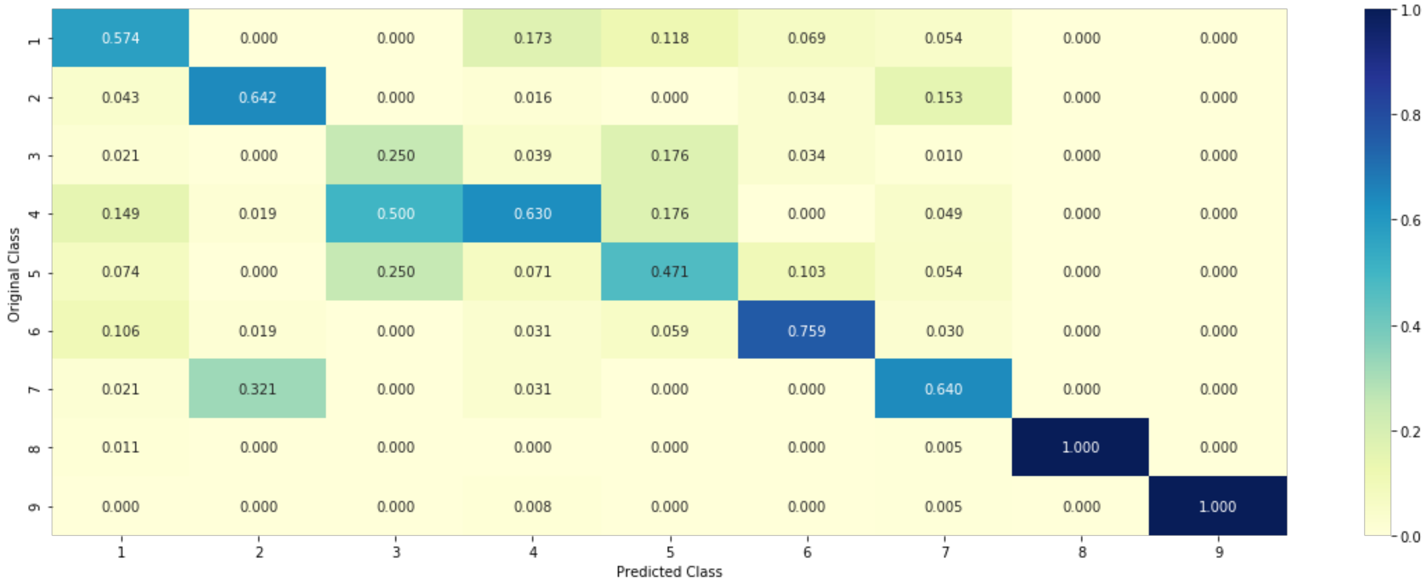




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```

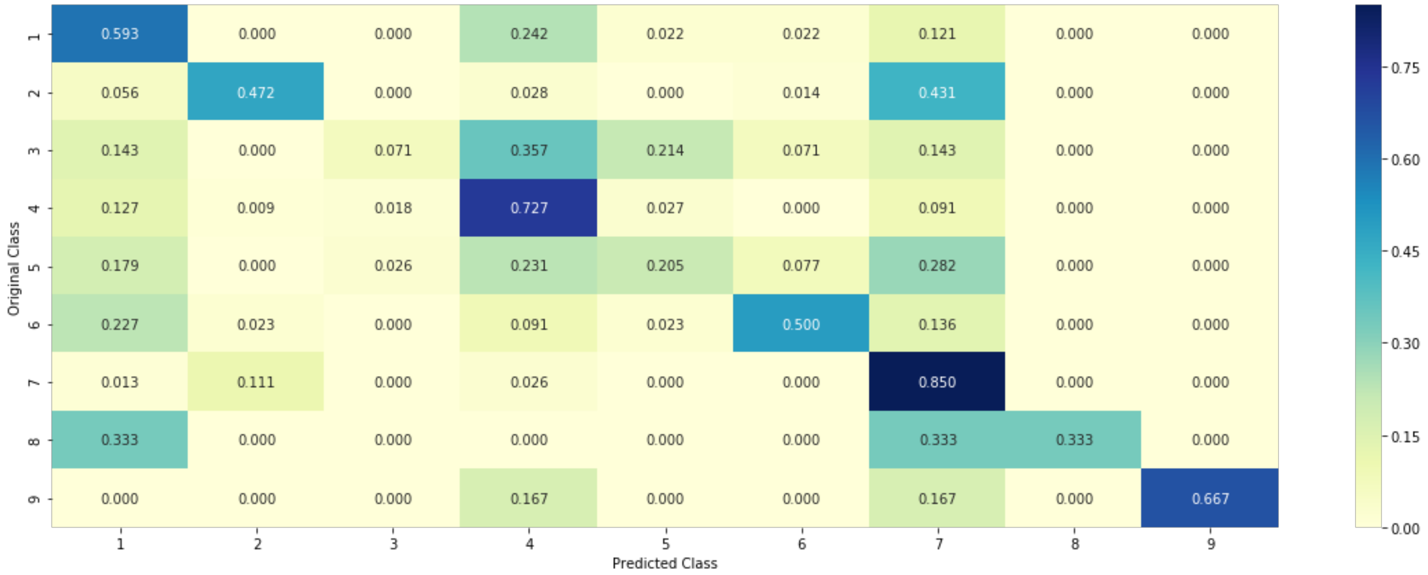
```
-------------------- Precision matrix (Columm Sum=1) --------------------
```

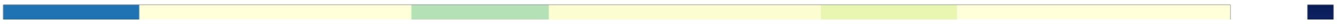


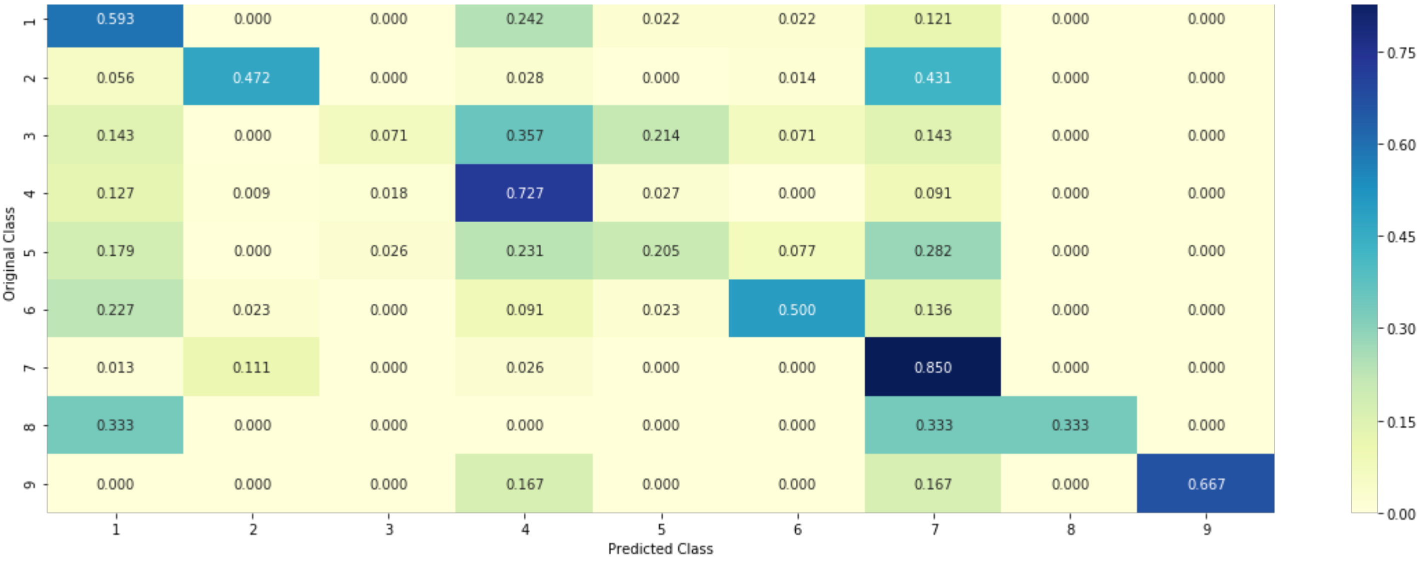


```
-------------------- Recall matrix (Row sum=1) --------------------
```

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







4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [83]:

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

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2741 0.1428 0.0193 0.3082 0.0651 0.0601 0.1075 0.0114
0.0115]]
Actual Class : 7
--------------------------------------------------
1 Text feature [kinase] present in test data point [True]
8 Text feature [treatment] present in test data point [True]
15 Text feature [drug] present in test data point [True]
21 Text feature [cells] present in test data point [True]
23 Text feature [protein] present in test data point [True]
26 Text feature [loss] present in test data point [True]
31 Text feature [missense] present in test data point [True]
43 Text feature [variants] present in test data point [True]
48 Text feature [unstable] present in test data point [True]
50 Text feature [patients] present in test data point [True]
52 Text feature [cell] present in test data point [True]
59 Text feature [expressing] present in test data point [True]
62 Text feature [clinical] present in test data point [True]
64 Text feature [functional] present in test data point [True]
86 Text feature [potential] present in test data point [True]
90 Text feature [atp] present in test data point [True]
91 Text feature [sequence] present in test data point [True]
Out of the top  100  features  17 are present in query point
Predicted Class : 4
Predicted Class Probabilities: [[0.2741 0.1428 0.0193 0.3082 0.0651 0.0601 0.1075 0.0114
Predicted Class Probabilities: [[0.2741 0.1428 0.0193 0.3082 0.0651 0.0601 0.1075 0.0114
0.0115]]
Actual Class : 7
--------------------------------------------------
1 Text feature [kinase] present in test data point [True]
8 Text feature [treatment] present in test data point [True]
15 Text feature [drug] present in test data point [True]
21 Text feature [cells] present in test data point [True]
23 Text feature [protein] present in test data point [True]
26 Text feature [loss] present in test data point [True]
31 Text feature [missense] present in test data point [True]
43 Text feature [variants] present in test data point [True]
48 Text feature [unstable] present in test data point [True]
50 Text feature [patients] present in test data point [True]
52 Text feature [cell] present in test data point [True]
59 Text feature [expressing] present in test data point [True]
62 Text feature [clinical] present in test data point [True]
64 Text feature [functional] present in test data point [True]
86 Text feature [potential] present in test data point [True]
90 Text feature [atp] present in test data point [True]
91 Text feature [sequence] present in test data point [True]
Out of the top  100  features  17 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [0]:

```
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_onehotCodin
g[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_point_index],test_d
f['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature
)
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [0]:

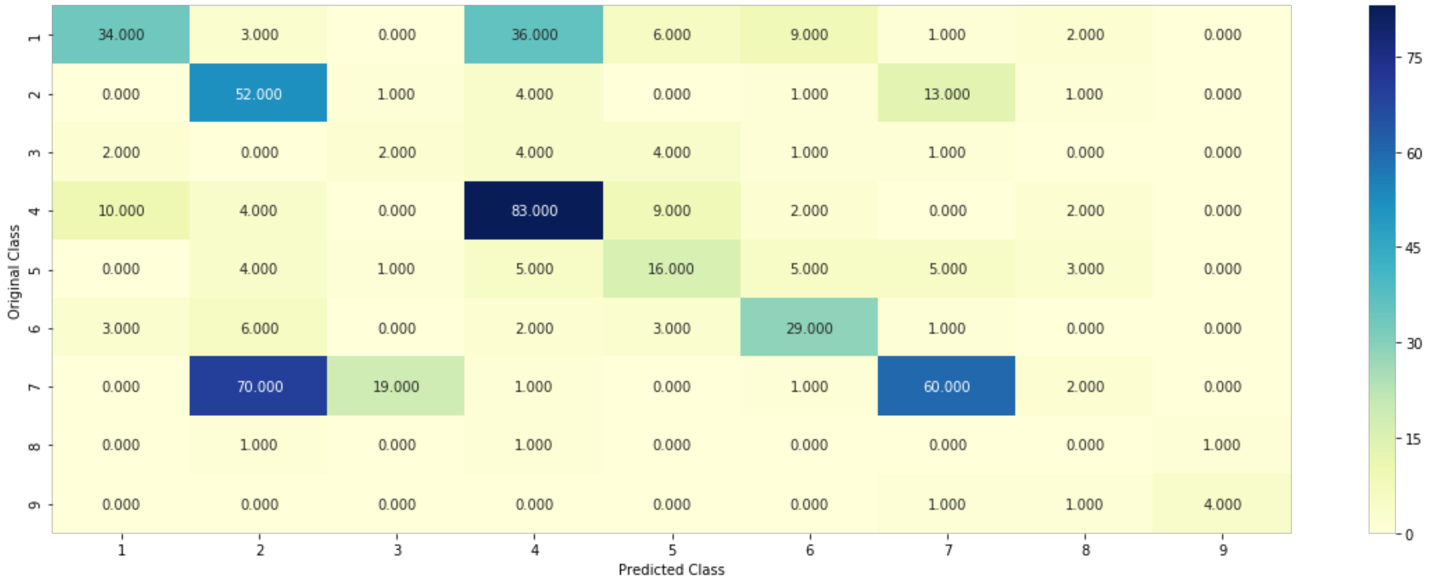
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=No
ne, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=N
one, min_impurity_decrease=0.0,
# min_impurity_split=None, 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/ran
dom-forest-and-their-construction-2/
# --------------------------------
# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/g
enerated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------


alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, rand
om_state=42, n_jobs=-1)
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=
1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (features[i],cv_log_error
_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

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

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

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

In [86]:

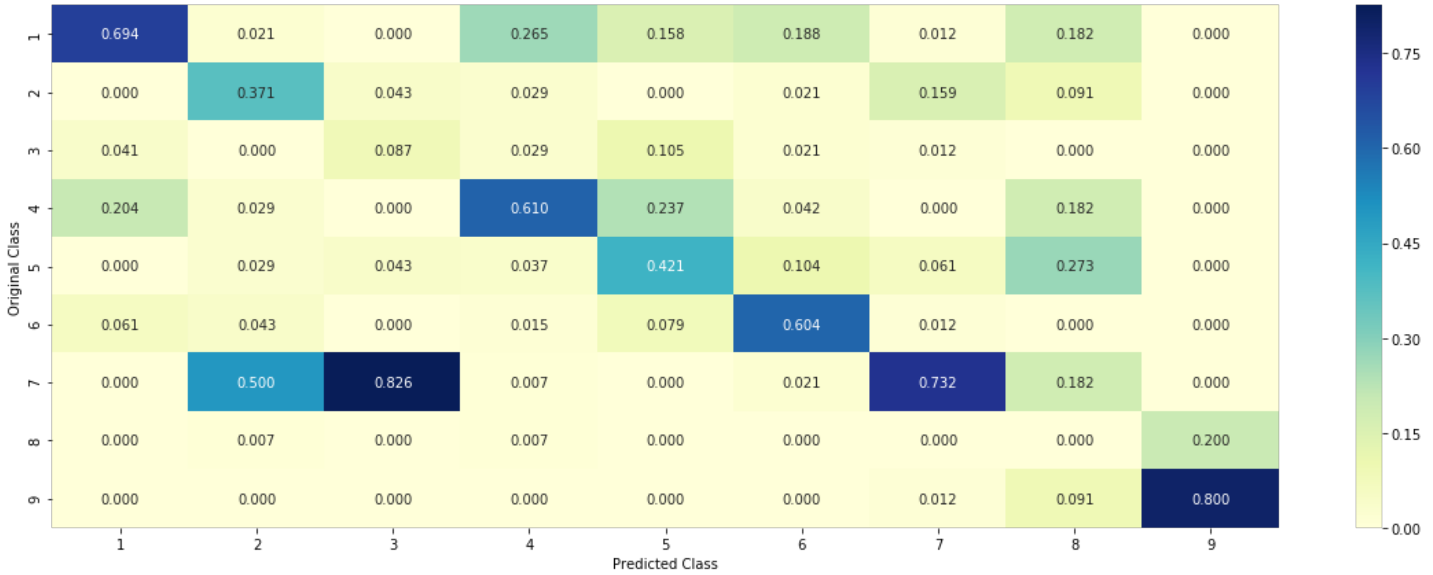
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=No
ne, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=N
one, min_impurity_decrease=0.0,
# min_impurity_split=None, 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/ran
dom-forest-and-their-construction-2/
# --------------------------------

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

```
Log loss : 1.2882770405790362
Number of mis-classified points : 0.47368421052631576
-------------------- Confusion matrix --------------------
```



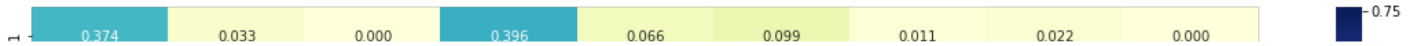
	1	2	3	4	5	6	7	8	9
1	34.000	3.000	0.000	36.000	6.000	9.000	1.000	2.000	0.000
2	0.000	52.000	1.000	4.000	0.000	1.000	13.000	1.000	0.000
3	2.000	0.000	2.000	4.000	4.000	1.000	1.000	0.000	0.000
4	10.000	4.000	0.000	83.000	9.000	2.000	0.000	2.000	0.000
5	0.000	4.000	1.000	5.000	16.000	5.000	5.000	3.000	0.000
6	3.000	6.000	0.000	2.000	3.000	29.000	1.000	0.000	0.000
7	0.000	70.000	19.000	1.000	0.000	1.000	60.000	2.000	0.000
8	0.000	1.000	0.000	1.000	0.000	0.000	0.000	0.000	1.000
9	0.000	0.000	0.000	0.000	0.000	0.000	1.000	1.000	4.000

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

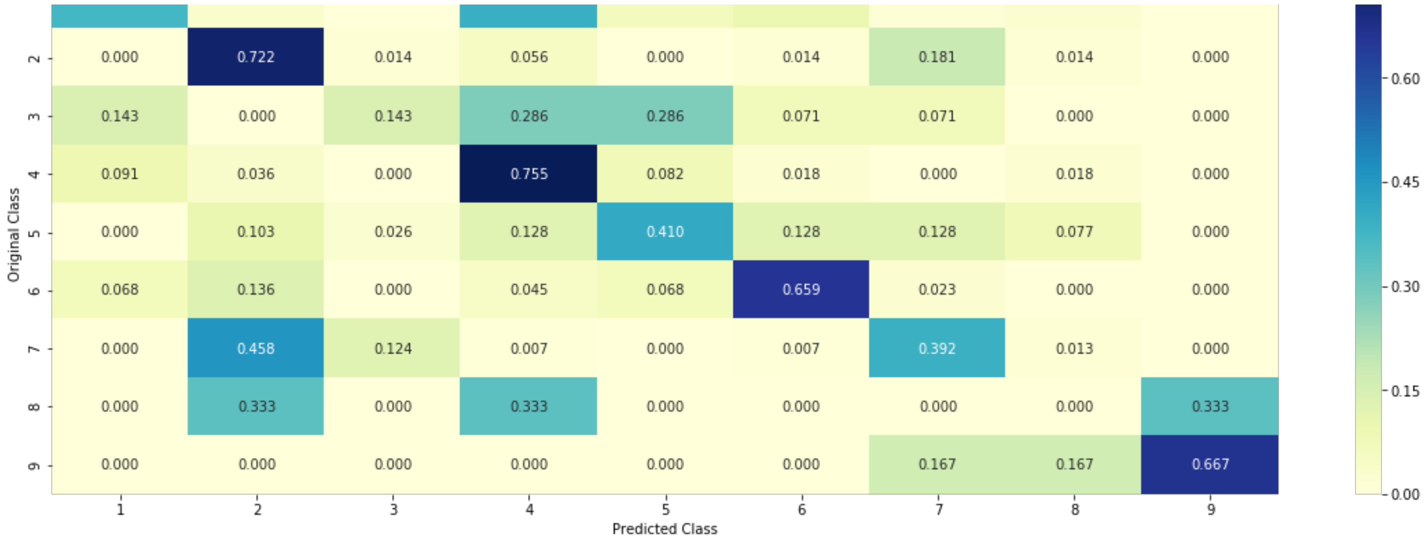


	1	2	3	4	5	6	7	8	9
1	0.694	0.021	0.000	0.265	0.158	0.188	0.012	0.182	0.000
2	0.000	0.371	0.043	0.029	0.000	0.021	0.159	0.091	0.000
3	0.041	0.000	0.087	0.029	0.105	0.021	0.012	0.000	0.000
4	0.204	0.029	0.000	0.610	0.237	0.042	0.000	0.182	0.000
5	0.000	0.029	0.043	0.037	0.421	0.104	0.061	0.273	0.000
6	0.061	0.043	0.000	0.015	0.079	0.604	0.012	0.000	0.000
7	0.000	0.500	0.826	0.007	0.000	0.021	0.732	0.182	0.000
8	0.000	0.007	0.000	0.007	0.000	0.000	0.000	0.000	0.200
9	0.000	0.000	0.000	0.000	0.000	0.000	0.012	0.091	0.800

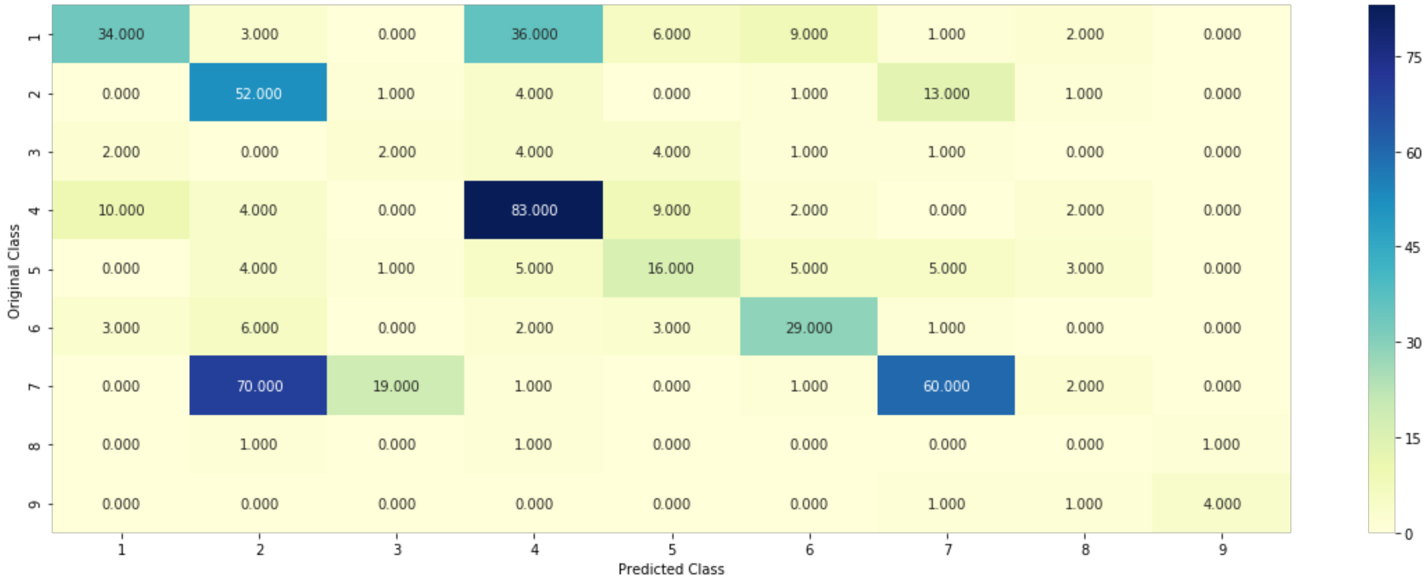
```
-------------------- Recall matrix (Row sum=1) --------------------
```



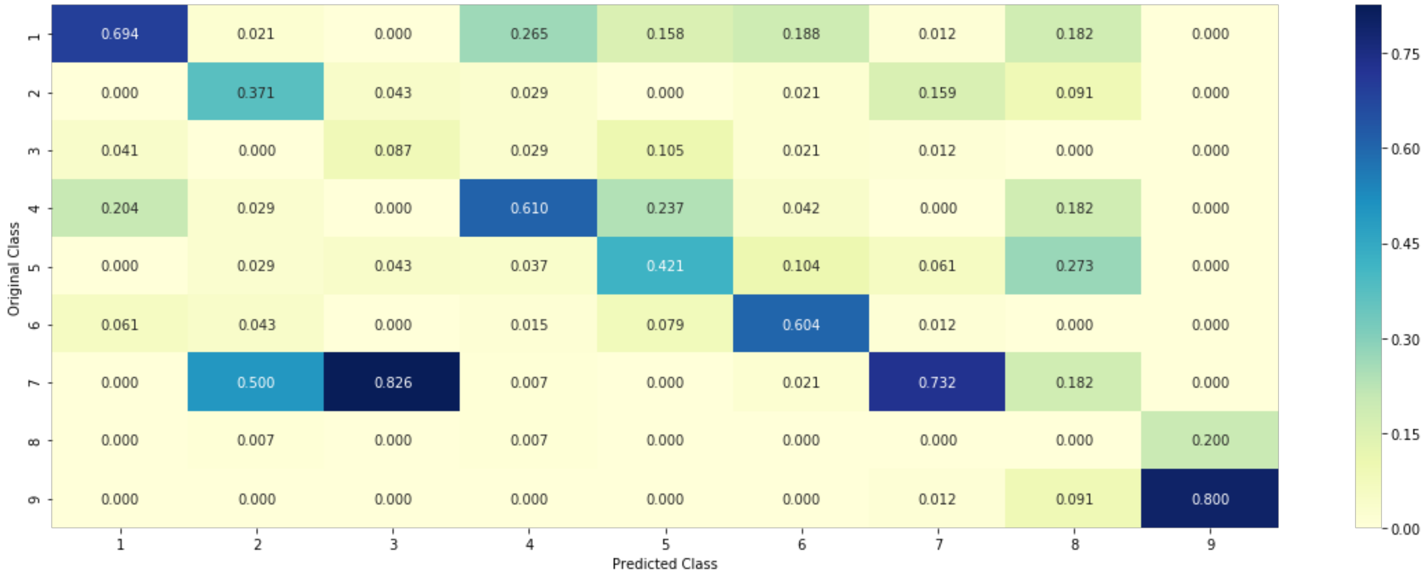
1	0.374	0.033	0.000	0.396	0.066	0.099	0.011	0.022	0.000



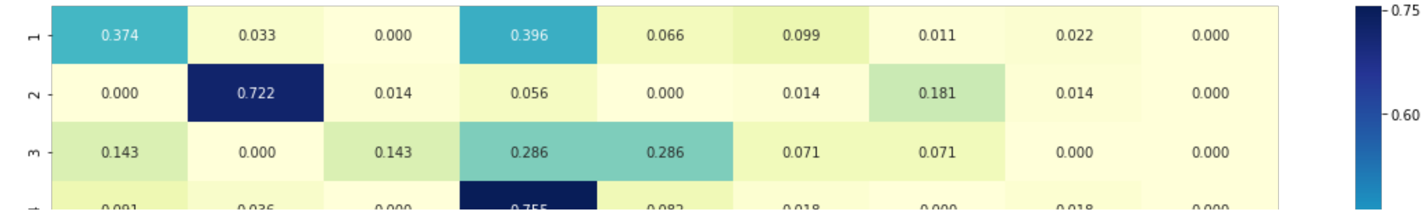
```
Log loss : 1.2882770405790362
Number of mis-classified points : 0.47368421052631576
-------------------- Confusion matrix --------------------
```

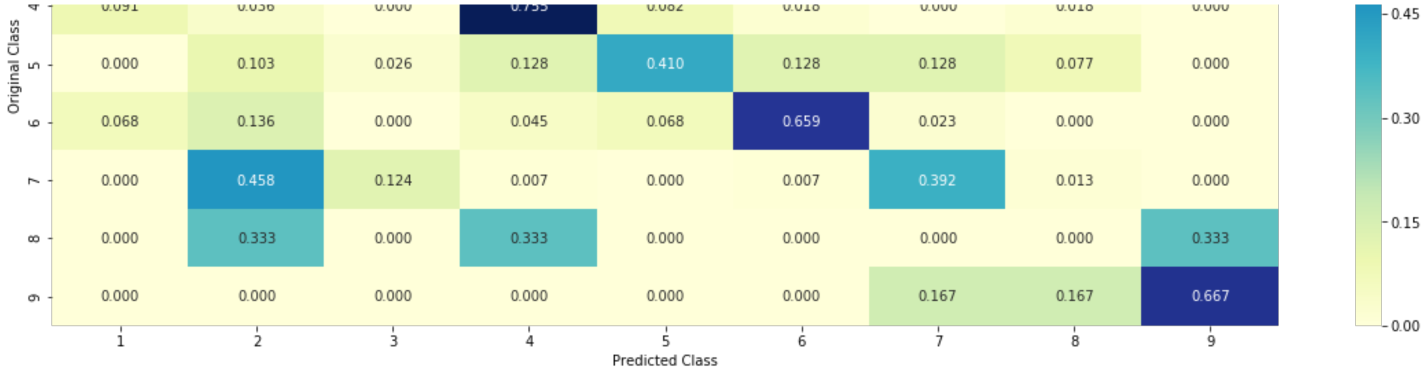


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [87]:

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


test_point_index = 1
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCod
ing[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: [[0.0848 0.366  0.0578 0.0664 0.0293 0.0458 0.0133 0.2832
0.0532]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Predicted Class : 2
Predicted Class Probabilities: [[0.0848 0.366  0.0578 0.0664 0.0293 0.0458 0.0133 0.2832
0.0532]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [88]:

```
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_responseCod
ing[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 : 4
Predicted Class Probabilities: [[0.1201 0.0306 0.0823 0.6399 0.0263 0.0346 0.0152 0.0263
0.0248]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Variation is important feature
```

```
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Predicted Class : 4
Predicted Class Probabilities: [[0.1201 0.0306 0.0823 0.6399 0.0263 0.0346 0.0152 0.0263
0.0248]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [89]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss=’hinge’, penalty=’l2’, alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=’optimal’, eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)
# some of methods
# fit(X, y[, coef_init, intercept_init, …]) Fit linear model with Stochastic Gradient Descent.
# predict(X) Predict class labels for samples in X.

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


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

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


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

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

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

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


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

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


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

sig_clf1.fit(train_x_onehotCoding, train_y)
```

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

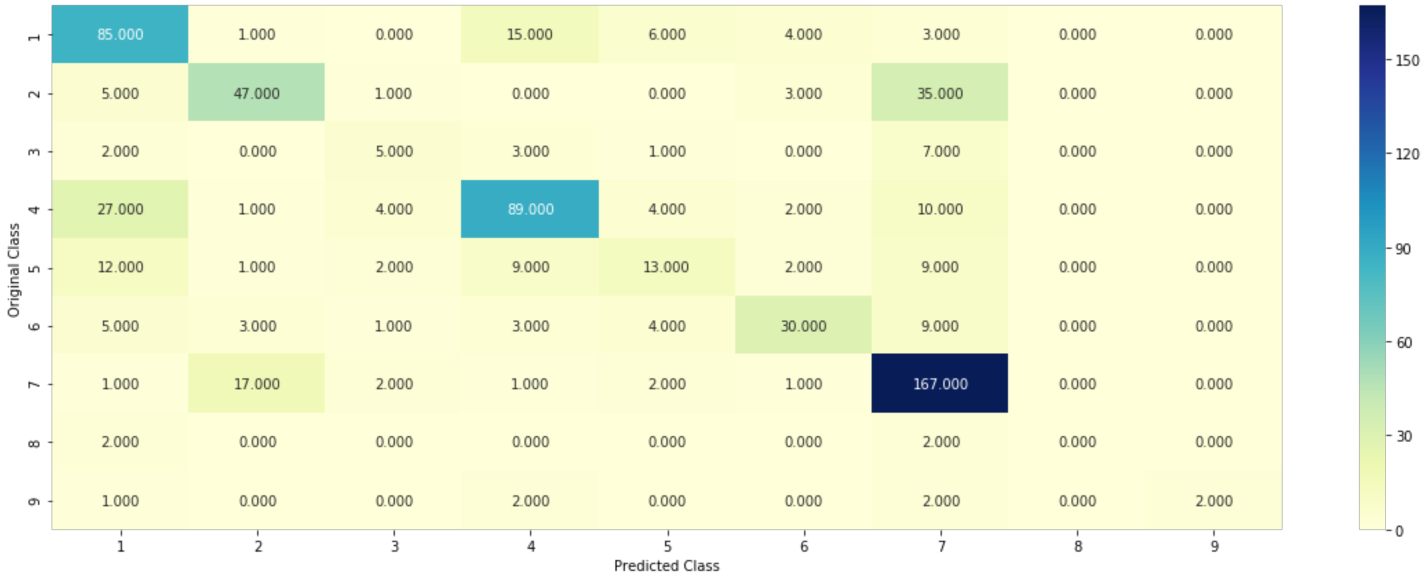
```
Logistic Regression :  Log Loss: 1.11
Support vector machines : Log Loss: 1.52
Naive Bayes : Log Loss: 1.26
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.817
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.708
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.289
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.180
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.469
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.775
Logistic Regression :  Log Loss: 1.11
Support vector machines : Log Loss: 1.52
Naive Bayes : Log Loss: 1.26
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.817
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.708
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.289
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.180
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.469
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.775
```

4.7.2 testing the model with the best hyper parameters

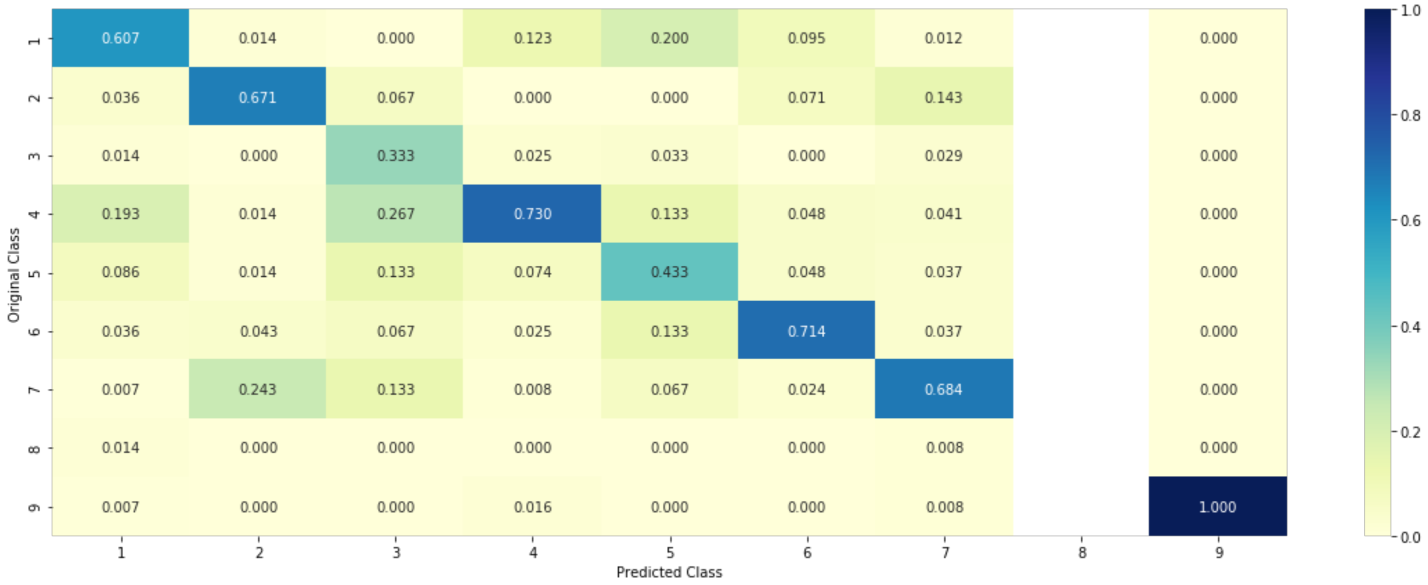
In [90]:

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

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

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

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

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

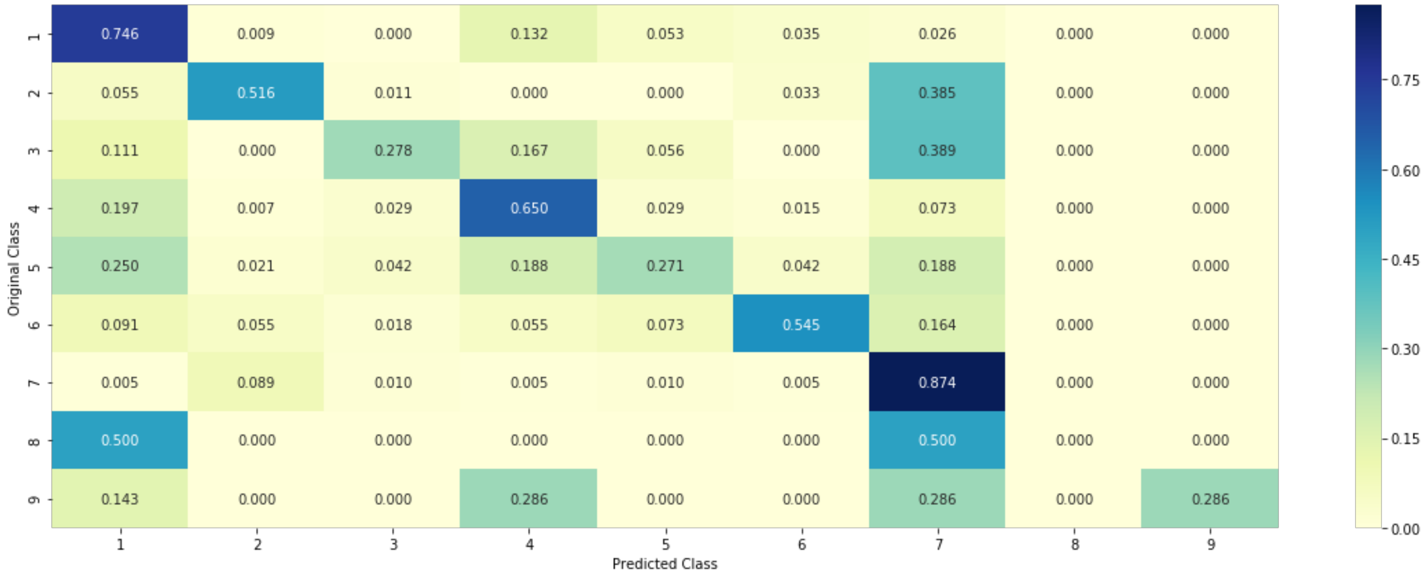
```
Log loss (train) on the stacking classifier : 0.4877960351435632
Log loss (CV) on the stacking classifier : 1.1797505800491734
Log loss (test) on the stacking classifier : 1.1129919879539028
Number of missclassified point : 0.34135338345864663
-------------------- Confusion matrix --------------------
```



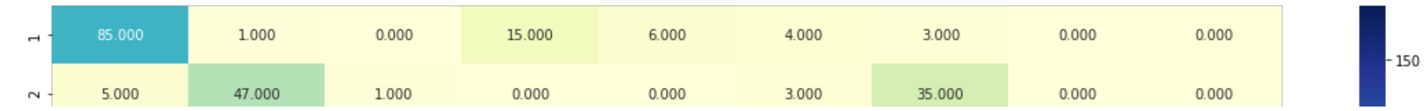
```
-------------------- Precision matrix (Columm Sum=1) --------------------
```

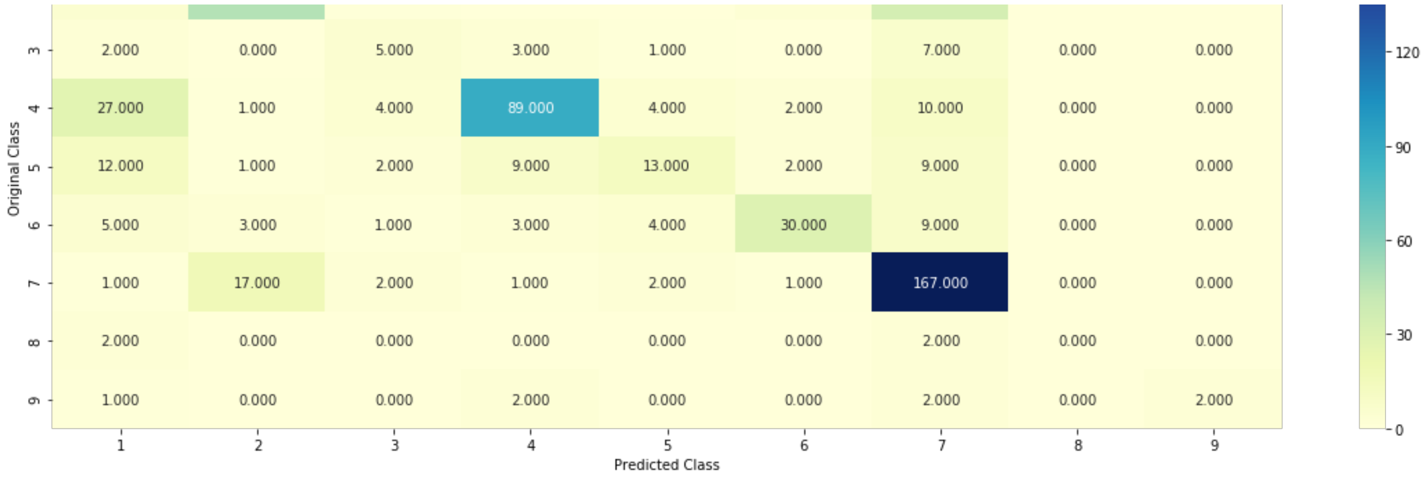


```
-------------------- Recall matrix (Row sum=1) --------------------
```

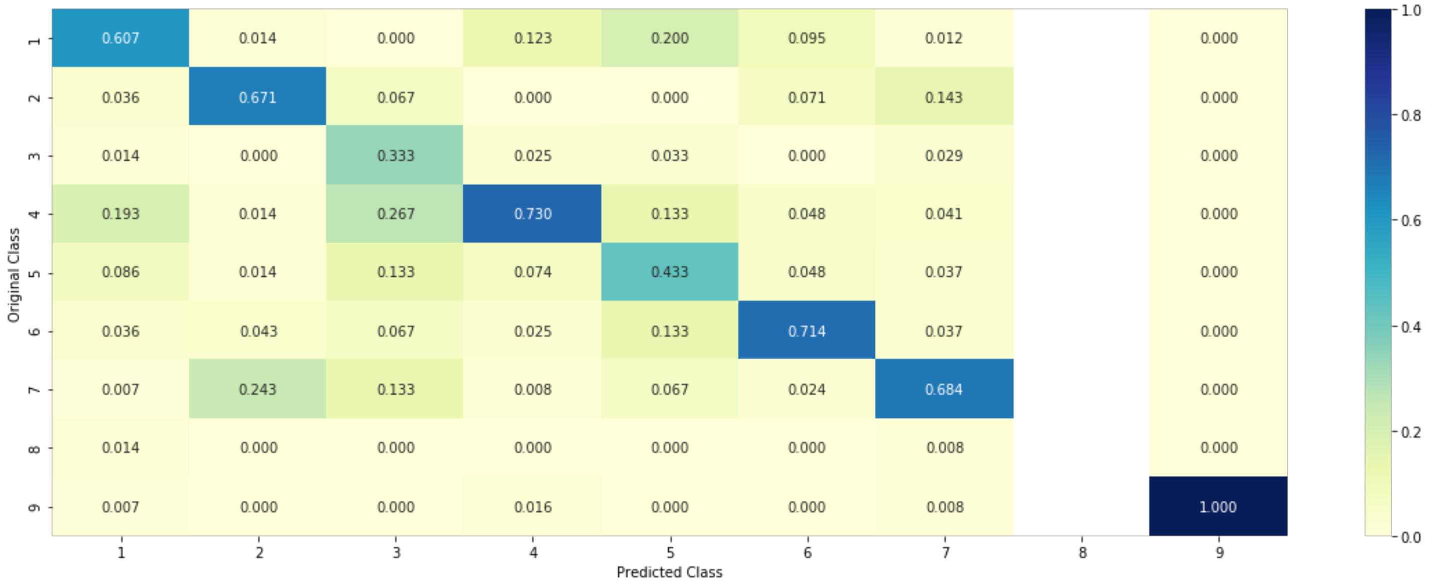


```
Log loss (train) on the stacking classifier : 0.48779603514356332
Log loss (CV) on the stacking classifier : 1.1797505800491734
Log loss (test) on the stacking classifier : 1.1129919879539028
Number of missclassified point : 0.34135338345864663
-------------------- Confusion matrix --------------------
```

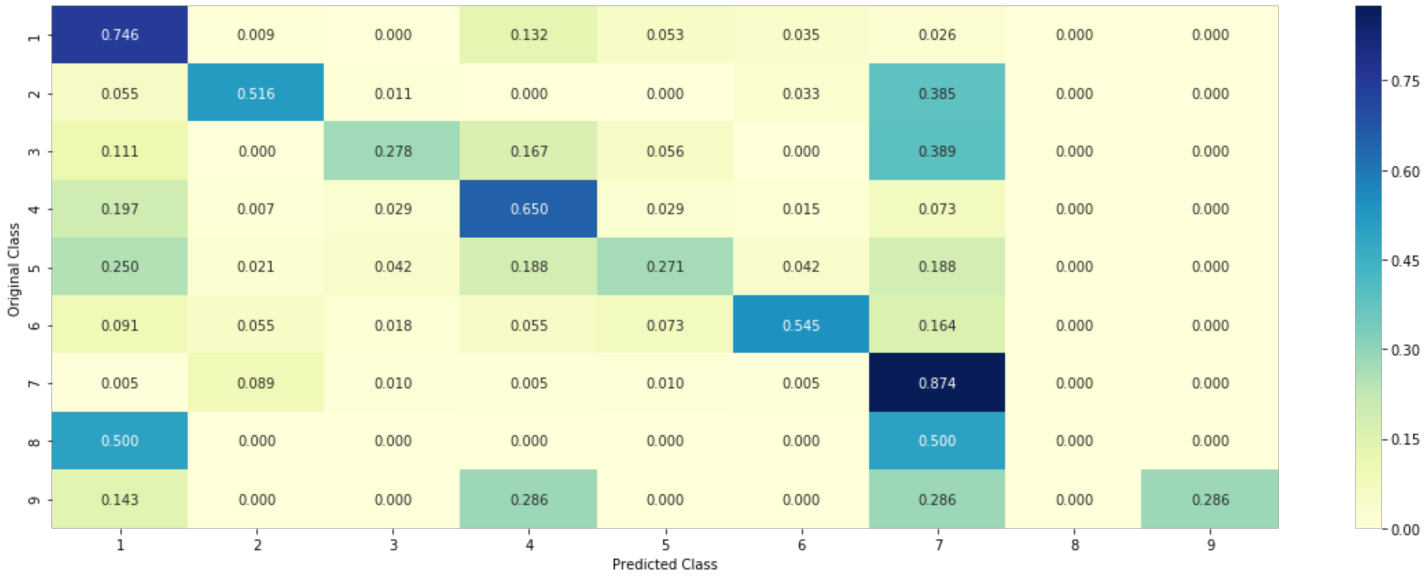




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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



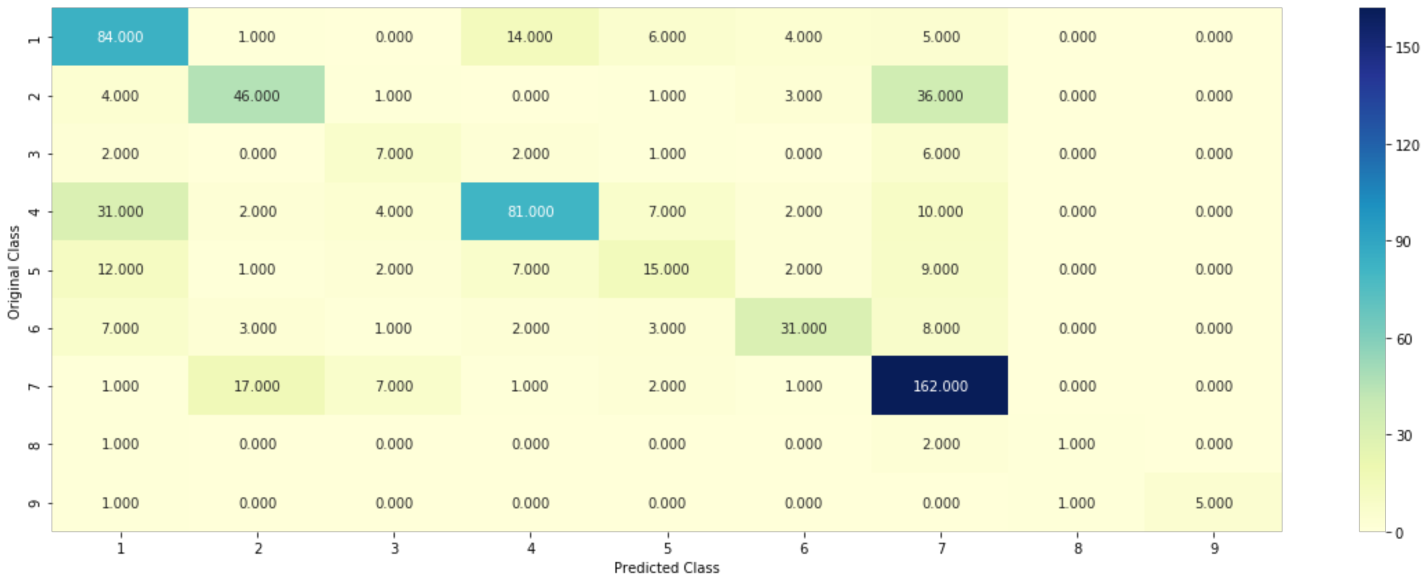
4.7.3 Maximum Voting classifier

In [91]:

```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier
.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3
)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
```

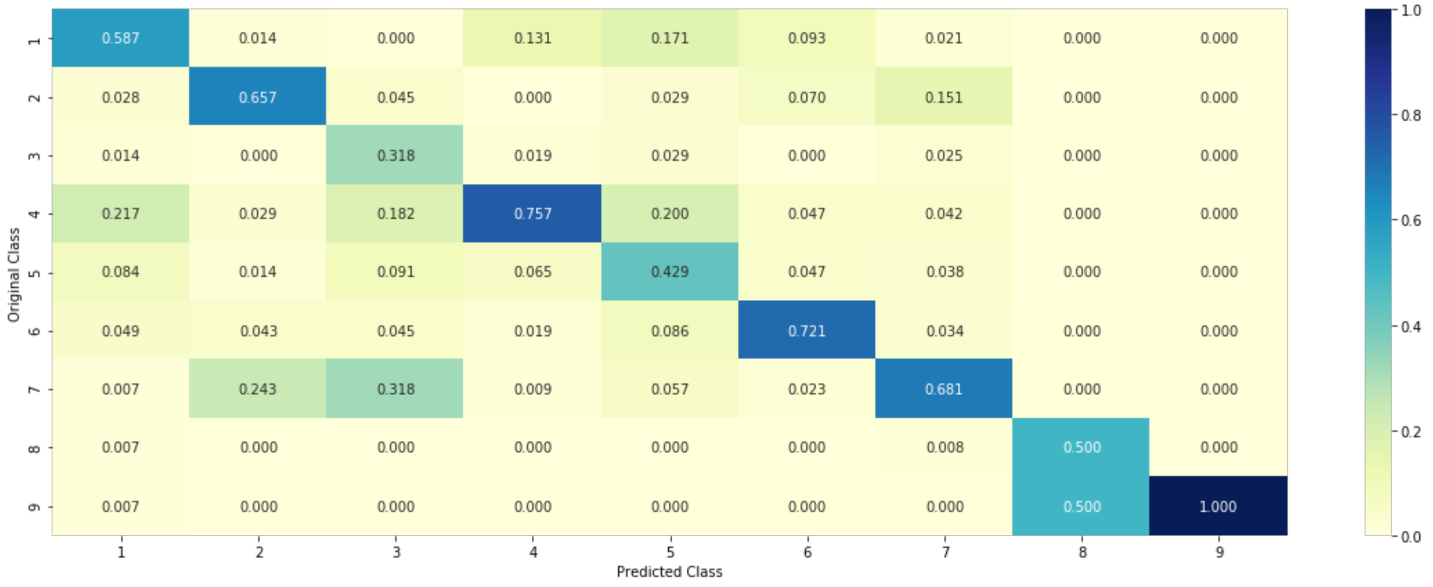
```
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

```
Log loss (train) on the VotingClassifier : 0.8027659329867048
Log loss (CV) on the VotingClassifier : 1.1517892213555991
Log loss (test) on the VotingClassifier : 1.1280348294905913
Number of missclassified point : 0.35037593984962406
-------------------- Confusion matrix --------------------
```



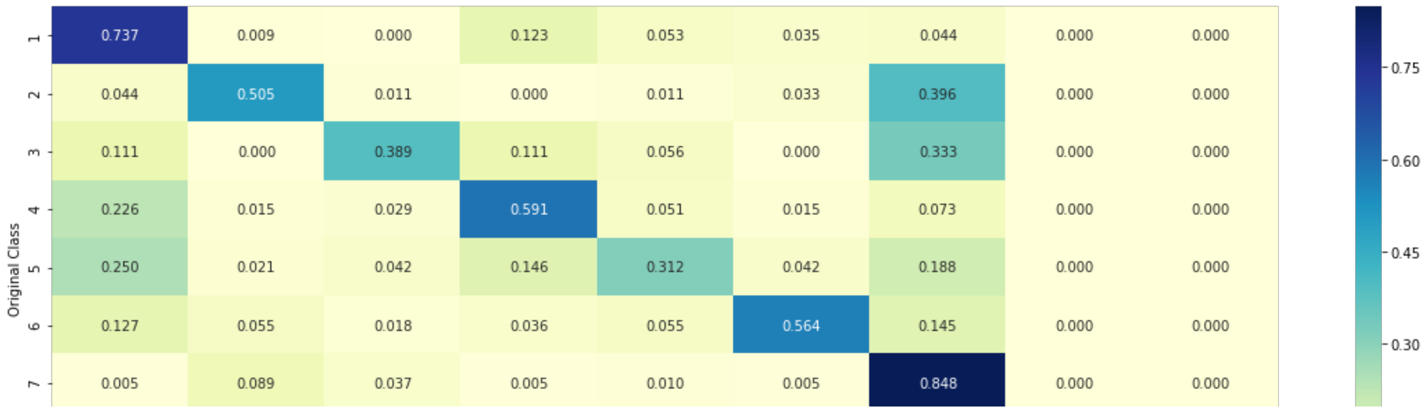
	1	2	3	4	5	6	7	8	9
1	84.000	1.000	0.000	14.000	6.000	4.000	5.000	0.000	0.000
2	4.000	46.000	1.000	0.000	1.000	3.000	36.000	0.000	0.000
3	2.000	0.000	7.000	2.000	1.000	0.000	6.000	0.000	0.000
4	31.000	2.000	4.000	81.000	7.000	2.000	10.000	0.000	0.000
5	12.000	1.000	2.000	7.000	15.000	2.000	9.000	0.000	0.000
6	7.000	3.000	1.000	2.000	3.000	31.000	8.000	0.000	0.000
7	1.000	17.000	7.000	1.000	2.000	1.000	162.000	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	0.000	2.000	1.000	0.000
9	1.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	5.000

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

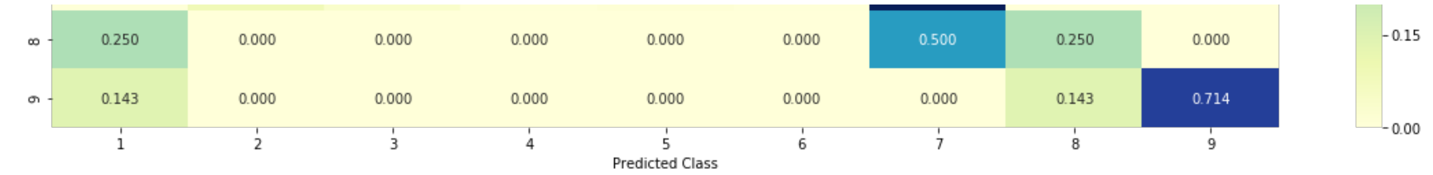


	1	2	3	4	5	6	7	8	9
1	0.587	0.014	0.000	0.131	0.171	0.093	0.021	0.000	0.000
2	0.028	0.657	0.045	0.000	0.029	0.070	0.151	0.000	0.000
3	0.014	0.000	0.318	0.019	0.029	0.000	0.025	0.000	0.000
4	0.217	0.029	0.182	0.757	0.200	0.047	0.042	0.000	0.000
5	0.084	0.014	0.091	0.065	0.429	0.047	0.038	0.000	0.000
6	0.049	0.043	0.045	0.019	0.086	0.721	0.034	0.000	0.000
7	0.007	0.243	0.318	0.009	0.057	0.023	0.681	0.000	0.000
8	0.007	0.000	0.000	0.000	0.000	0.000	0.008	0.500	0.000
9	0.007	0.000	0.000	0.000	0.000	0.000	0.000	0.500	1.000

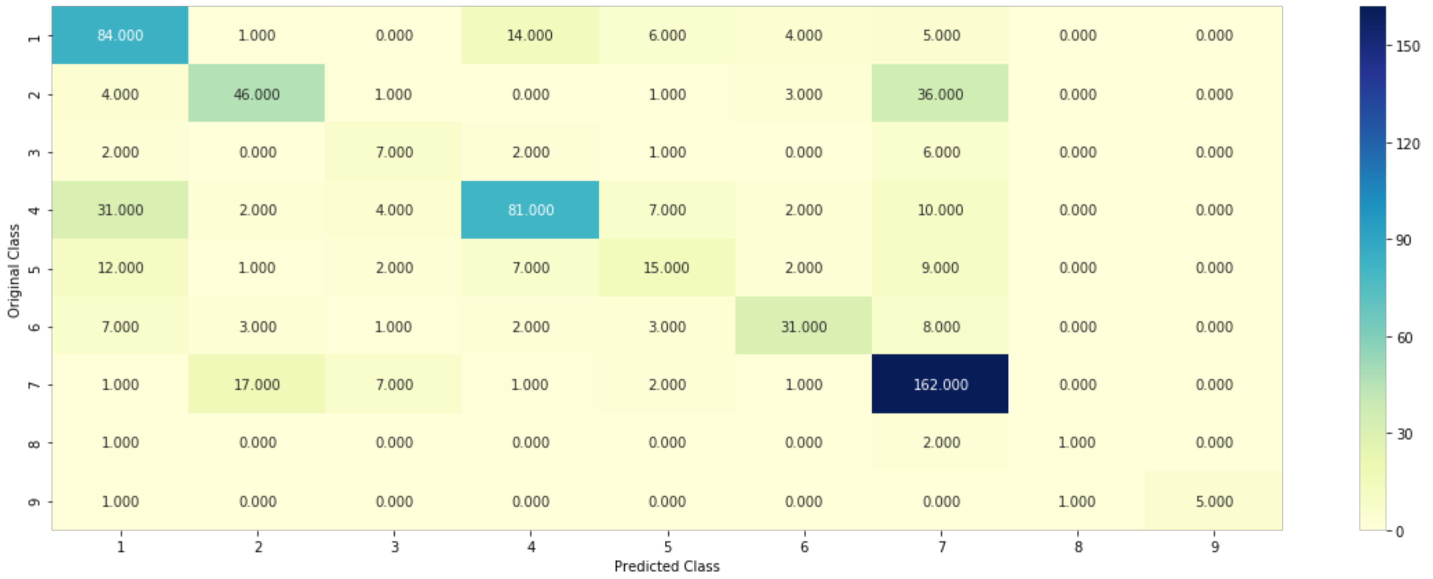
```
-------------------- Recall matrix (Row sum=1) --------------------
```



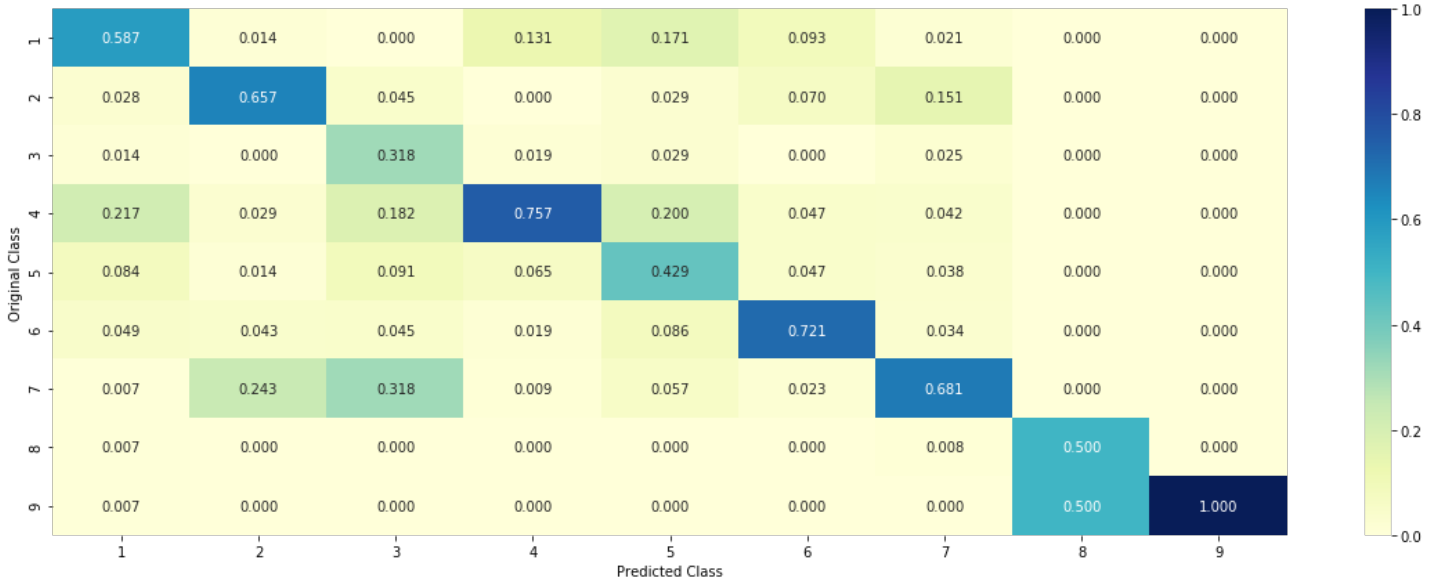
1	0.737	0.009	0.000	0.123	0.053	0.035	0.044	0.000	0.000
2	0.044	0.505	0.011	0.000	0.011	0.033	0.396	0.000	0.000
3	0.111	0.000	0.389	0.111	0.056	0.000	0.333	0.000	0.000
4	0.226	0.015	0.029	0.591	0.051	0.015	0.073	0.000	0.000
5	0.250	0.021	0.042	0.146	0.312	0.042	0.188	0.000	0.000
6	0.127	0.055	0.018	0.036	0.055	0.564	0.145	0.000	0.000
7	0.005	0.089	0.037	0.005	0.010	0.005	0.848	0.000	0.000



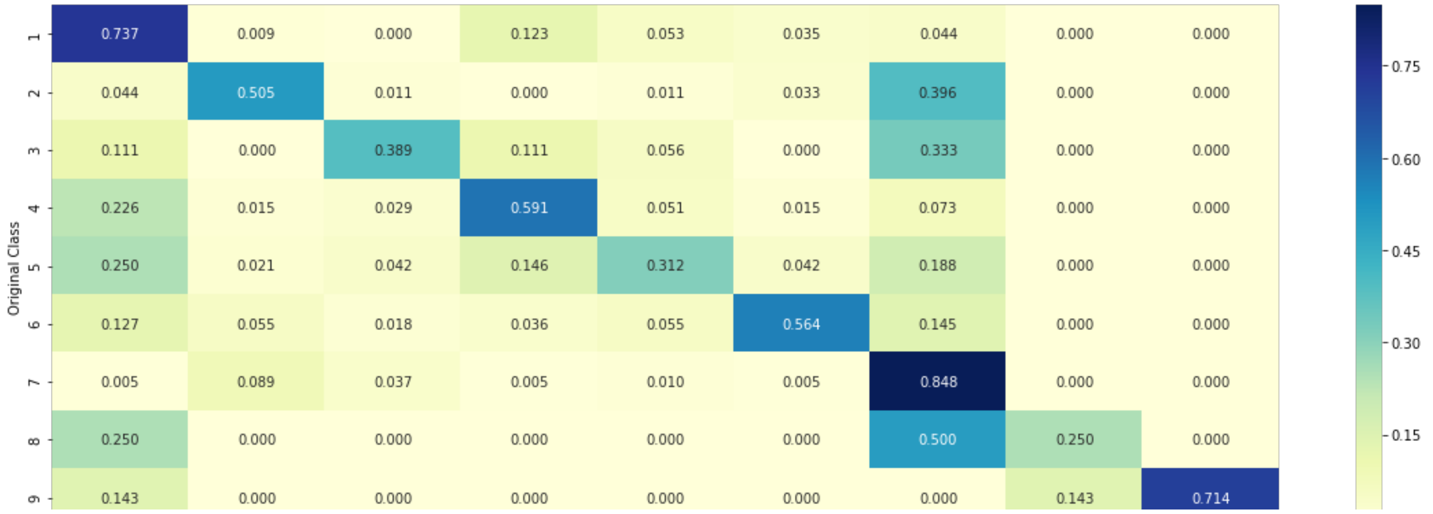
```
Log loss (train) on the VotingClassifier : 0.8027659329867048
Log loss (CV) on the VotingClassifier : 1.1517892213555991
Log loss (test) on the VotingClassifier : 1.1280348294905913
Number of missclassified point : 0.35037593984962406
-------------------- Confusion matrix --------------------
```

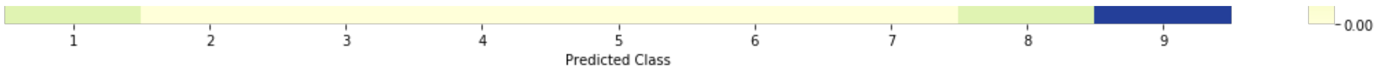


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





5. Assignments

In [0]:

```

```

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

Using top 1000 features and applying them on Random Forest

In [92]:

```
# one-hot encoding of variation feature.
text_vectorizer = TfidfVectorizer(min_df=10,max_df=25)
tfidf=text_vectorizer.fit_transform(result['TEXT'])
print(tfidf.shape)
```

```
(3321, 15484)
(3321, 15484)
```

In [93]:

```
all_features2=text_vectorizer.get_feature_names()
words2=[]
idf2=text_vectorizer.idf_
features=np.argsort(idf2)[::-1]
for i in features[0:1000]:
    words2.append(all_features2[i])
print(words2)
```

```
['1356', 'd275', 'l43', 'l597q', 'toad', 'l792l', 'd368', 'l814l', 'toggle', 'l826m', 're
sorted', 'toledo', 'd275a', '4512', 'tnfr1', 'resists', 'tom', '12r', 'l862v', 'tomizawa'
, 'labvision', '461g20', 'resat', 'top2a', '4691', '2053', 'tnfr2', 'd538', 'cytoband', '
l115p', 'dcog', 'tip3', 'krkljus', 'krongrad', 'tis', 'tissuelyser', 'datto', 'datathe',
'20jc', 'dash', 'l1152p', '44a', 'd572del', 'daehak', 'd877n', 'l1301r', 'd837d', 'tlx1',
'd820g4', 'd816e', 'reuther', 'd67', 'l215p', 'l229f', 'landau', 'reported9', 'corporate'
, 'crizotinibnaive', 'csrd', 'csd34', 'leung', 'cscc', 'transfers', 'crystalline', '4g5j'
, 'reinitiate', 'crosslinker', 'lh', '200m', 'reha', 'leucinyl', '4l', 'creases', 'reh',
'cpt', 'linesalthough', 'lipofectamine2000', 'cov', 'lipogenesis', 'lipomatous', 'lis1',
'refractive', 'leukemia18', 'ctcgag', 'cyclotron', 'tpp', 'cyclohexamide', 'tothe', 'cycc
', 'replications', 'tox', 'lariat', 'laura', 'repertoires', 'cushion', 'leach', 'rep', 'c
uboidal', 'relaxing', 'leeds', 'cttacgcugaguacuucga', 'lengauer', 'lengthened', '4a4a', '
train', 'tramp', 'les', 'lethargy', 'leu1790', 'relevantly', 'rhodes', 'ddx11', 'ribogree
n', 'dhomen', 'theories', 'k659', '4067a', 'k737t', '4105', 'diaminophenylpyrimidine', 'k
756r', '4171a', 'k975e', 'kaganovich', 'rosenzweig', 'dharmafect1', '4058', 'kanai', 'dg'
, 'thermoscript', 'karyotypically', '4173a', 'ronchetti', 'deutz', '4185', 'kdm2a', 'dete
rgents', '4193t', 'themutant', 'dilation', '440g', 'k499e', 'jura', 'tfiid', 'k1409e', 'd
izziness', 'k1436q', 'k22', 'disfavors', 'k299r', 'tgggcagtatctttccagcaac', 'rptor', 'dis
cussions', 'k607t', 'thein', 'discriminatory', 'k618a', 'disciplinary', '3tp', 'dip', 'rp
5', 'k618t', '4007a', 'rowley', '400mgorally', 'routines', 'destroying', 'kdm5d', '2161',
'komolgrov', 'klh', 'kmt2b', 'thymomas', 'delk745', '4306a', 'tiangen', 'knife', 'riva',
'rita', 'tics', 'dei', 'decreaseorlessthan20', 'desperately', 'korona', 'decomposition',
'tiff', 'rimerman', 'dec', 'richelda', 'richards', '4388c', 'krantz', 'dealt', 'timescale
', 'deltyd', 'delv559v560', 'delving', 'delvv', 'desmosomes', 'design32', 'keam', 'descri
bed6', 'keeper', 'thirteenth', 'thomson', 'kelvin', 'thr180', '4225a', 'kimdw', 'deproton
ation', 'depolymerization', 'kindai', 'depletes', 'kirschner', 'deoxynucleotidyltransfera
se', 'kitor', '4235c', 'kix', 'denhardt', 'rmscf', 'demethylate', 'trays', 'lmnb2', 'dobs
on', 'radically', '558del', 'mcelhinny', 'tyr568', 'radiolabeling', 'certinib', 'radiolab
```

```
el', 'cep110', 'centroblasts', 'mcglade', 'mckenna', '1nf1a', 'tyr845', 'tyr416', 'tyr98'
, 'cells3', 'cells27', 'mcpherson', 'mcrc', 'tzahar', '5721', '13department', 'cells13',
'576del', 'celllines', 'radiosurgery', 'maxima', 'rad21', 'chemosensitive', 'txt', 'marqu
e', 'tyd', 'tyedmers', 'chk', 'rak', 'mart', 'types24', 'chiari', 'tyr1221', 'martins', '
tyr15', 'radsource', 'chemilluminescence', 'chemifluorescence', 'tyr182', '1p34', 'checks
', '1p32', 'ch2', 'mathe', 'matsuno', 'cgap', 'matsuoka', 'mdia', '1m14', 'localise', 'mg
h011', '1atp', 'mermel', 'mesa', 'ultrastructure', 'unacceptably', 'metaplastic', '19q12'
, 'methadone', 'methosulfate', 'r659p', 'mfei', 'cav1', 'cci', 'mgh045', 'mgso', 'miame',
'categorize', '19f13', 'microenvironmental', 'microfarads', '0222', 'micrognathia', 'r461
i', '02139', 'ccgg', 'ccl2', 'ceffs', 'cd69', 'me3', 'ceff', 'rabep1', '578del', 'cdk12s'
, 'cdf', 'meaningless', '13q22', 'mechali', 'mecp2', 'r97', 'medlineplus', '586a2', 'mela
nocortin', 'melanoma6', 'cd32', 'ucg', 'uch', 'uci', 'melanomais', 'cd14', 'cd11c', 'melk
i', '1department', '1t', 'chp', 'marburg', 'luciferace', 'recombine', 'computerassisted',
'lrp6', 'ls1034', 'comprehend', 'lsrfortessa', 'ltk', 'comparedwith', 'comparability', 'r
eceptive', 'ltrs', 'commit', 'recombining', 'commented', 'recalibrated', 'trov', 'trovo',
'trp117', 'lujan', 'colorless', 'colonize', 'ly19', 'trp88', 'ly3', 'concentrators', 'low
ell', 'twist1', 'continuity', 'reestablish', 'reenter', 'coqueret', 'locates', 'reedijk',
'locating', 'trended', 'triage', 'reductive', '4v', 'redirects', 'tricine', 'confused', '
lombardo', '500g', 'looney', 'trifluoroacetic', '50jc', 'trihydrate', 'trilineage', 'cons
ervationon', 'connectivity', 'lovec', 'lovly', '06469322', 'lymphangiogenesis', 'reanneal
ing', 'cj236', 'tumorous', '1tup', 'maeda', 'mag', 'mage', 'cleidocranial', '1tsr', 'clea
', 'classifications13', 'clarke', 'ras4a', 'turcot', '1355', 'maleimide', 'malfunctions',
'cir', 'cignal', 'malkin', 'maml2', 'mangoura', '1382', 'manuscripts', 'randomness', 'chr
7', 'macrosomia', '1week', 'clotting', 'tucked', 'genechem', 'colectomy', 'rccs', 'col8a1
', 'col2a1', 'cohesive', '06439015', 'rbm5', 'rbgh', 'tsuzuki', '1367', 'm1149t', 'codd',
'm1b', 'cochran', 'cobalt', 'coall', 'cng', 'm5g', 'ttingen', 'cmin', 'machens', 'cls', '
jung', 'jsnp', 'microlesions', 'ser2', '2867', 'fla', 'striped', 'strn', 'fl5', 'headed',
'fitchburg', 'hegde', 'ser222', 'filamina', 'ser218', 'figure9b', 'stricto', 'hemangiobla
stoma', 'figure4e4e', 'figure4b4b', 'ser111', 'ser10', 'hemangioendothelioma', 'hemangios
arcoma', '28102', 'sty', '2800', 'heptapeptide', '1097', 'stretching', 'herman', 'hamlin'
, 'h3k18ac', '30a', 'fractionations', 'h82', 'h9010', 'haar', 'fortune', 'haeii', 'forked
', 'foregoing', 'hagiwara', 'set2', 'hcs', 'stott', 'set1', 'hani', 'harvester', 'fma3',
'fluoview', 'hatzivassiliou', 'hawkins', '2878', 'hcc1187', 'ser5', 'subcategory', 'hern'
, 'schultz', '1082', 'secured', '2622', 'hmb', 'hmm', 'sec31a', 'holmes', 'scudierio', 'h
omepage', 'scrotum', 'f7', 'f691l', 'hon', 'sedentary', 'sumoylated', 'hopeful', 'hospita
lization', 'f44', 'f367s', 'hotstart', 'housework', 'f2108l', 'hrasv12', '3350', '123a',
'fam131b', 'hkl', 'hernia', 'feso4', 'fibulae', 'fibrodysplasia', 'sensu', 'subfragments'
, 'heterodimerzation', 'hextuplicate', 'fgfr2s252w', 'subq', '2732', 'highgrade', '2696',
'hinged', 'sediment', 'hinging', 'hip', '2678', '3229', 'felix', 'hipk2', 'hips', 'hisa',
'fearon', 'farmer', 'hiv1', 'friable', 'sterne', 'h3k18', 'g909r', 'gist35', 'gist37', 'g
agacuagacaaugagaaa', 'gadgeel', 'slidethree', 'slideschematic', 'gaagaacagtattgacata', 's
pikes', 'slideassessment', 'gaaagaagacaaacagaaa', 'g97d', 'g873q', 'slx4', 'skov3', 'skov
', 'skippy', 'g779g', 'g742d', 'g691s', 'skeletons', 'glut4', 'sitespecific', 'site2', 'g
469v', 'spencer', 'spectrometers', 'sterical', 'snu2535', 'generalizability', 'gdi', 'gen
eva', '114k', 'genosplice', 'gcrma', 'solon', 'solenoid', 'soap', 'soak', 'snyder', '2p16
', 'spectrofluorometer', 'gggacaagtttgtacaaa', 'ggggaccactttgtacaagaaagctgggtcaat', 'snf5
', 'snb19', 'snapfrozen', 'gaussia', 'specialization', 'giacca', 'gastritis', 'smcd', 'sp
ectrochip', 'goksenin', 'sirt1', 'g466e', 'ship1', 'stanchina', 'shrna2', 'fyrn', 'starin
sky', 'gtype', 'guanylate', 'guggguguuggacagugua', 'gure', 'stathmin', 'fusion14', 'fur',
'gyrata', 'g465', 'h1112', 'ste7', '3097', 'shielding', 'h1838', 'shevelev', 'steiger', '
sheng', 'fs2f', 'steponeplus', 'h249', 'gtf', '3031', 'sibley', 'stag2', 'goodman', 'gott
fried', 'singlestrand', 'govindan', 'sprycel', 'spurdle', '1108', 'grdthe', 'g322d', 'gri
gelioniene', 'grm8', 'groin', 'sry', 'grooves', 'signalway', 'grr5', '1101', 'gsf', 'gtag
atacaagtagagaat', 'sickness', 'g13v', 'sibs', '3026', 'schwannomin', 'ht1080', 'jongno',
's218', 'intensified', 'intercalation', 'intercrosses', 'intermediately', 'intraindividua
l', 'e562del', 'tanizaki', 'intravascular', 'intron1', 'e501k', 's222', 'e380q', 'e711t71
2', 'ira1p', 'ira2p', '2256', 'tasigna', 'e318', '3eqc', 's1760a', 'taufkirchen', 'e277a'
, 'e277', 'e275', 'e711', 's29', 'tbhq', 'ind', 'inasmuch', 'incapacitate', 't751insa', '
eff', 'eet', 's459f', 's459', 'eed', 'edwards', '3919a', 'edged', 'taatacgactcactataggg',
'e749del', 'indentified', 'tables11', 'initiators', 'ec50s', 'ink', '3980a', 's323g', 'ea
ger', 'intensification', 's2ab', 's297f', 'isodisomy', 's1463f', 'htseq', 'terlouw', 'tel
1', 'dsrna', 'rxa', 'rx', '2240', 'tens', 'russia', 'jenkins', 'doxycyline', 'terf2', 'do
wnstaging', 'jersey', 'duality', 'jetpei', 'ruiz', 'rui', '102k', 'dorset', 'jfcr', 'jfcr
021', 'jl', 'jmmls', 'jnj', 'jolly', 'rxrxxs', 'jak2wt', 'isolations', 'e011', 'isomer',
'e2419k', 'isopeptidase', 'itching', 'iteration', 'e1356g', 'e1327g', 'e124', 'e1210', 's
1058p', '1035', 's10147', 'dudek', 'ivs7', 'dysmorphology', 'dyskeratosis', 'dysgeusia',
'dyedeoxy', 'jaffe', 'dusp5', 'dupn771', 'dupa767', 'duncan', 'dumaz', 't725t', 't712', '
t710a', 'esteller', 'evi', 'scala', 'hyp564', 'hypermetabolic', 'eurogentec', 'euclidian'
, 'etude', 'hyponatremia', 'etch', 'hypothemycin', 'esterification', 'established11', 'su
zanne', 'esss', 'sweep', 'esrp1', 'swi2', 'sayama', '3555', 'i1307v', 'ernst', 'symposium
', '2517', 'eric', 'sv40t', 'hydroxide', '2368', 'exon6', 'externally', 'expr', 'hudson',
'expecting', 'supraclavicular', 'exor311h', 'supt1', 'exop286r', 'humoral', 'schaefer', '
```

```
schaaper', 'exon21', 'hydrophilicity', '124k', 'exon14', 'exod316n', 'surrogates', 'exod3
16', 'exocyclic', 'exibility', 'scars', 'hydrochloric', 'hydrolysing', 'exemestane', 'syn
drome21', '3610c', 'synergies', 'electrical', 'elsewhere40', 'elonginb', 'elongate', 't25
15i', '1268', 'elisas', '23a', '2374', 'electrochemical', 'electrocardiographic', 'electr
oblotting', '3902t', 'i9', 'elbaz', 't468m', 't485k', 't511a', 'imperial', 'importer', 's
4j', 's46i', 'ehmt1', 'improvision', 'inaccessibility', 't1n0m0', 's846i', '38101x', 'end
ogeneous', '365h22', 'equate', 'equality', 'ichthyosis', '24b', 'ig3', 'epitheloid', '371
9', 'igg2', 'epha1', '3743', 'igg2a', 'igiii', 'envisage', 'sakaguchi', 'sahni', 'ignatiu
s', 't123', 'ihca', 't1324n', 'engenders', 'enforce', 'energetics', 'catalysing', '2q12',
'understanding13', 'prohibits', 'arched', 'oedema', 'offsets', 'ohtsuka', '00299804', '70
01', 'aptag', 'waterman', 'okami', 'olayioye', 'appends', 'promoterless', 'progressiva',
'wc', 'apochromat', 'aplasia', 'profuse', 'oligoastrocytic', 'wean', 'apbs', 'webb', 'apa
ricio', 'arg103', 'proofs', 'proviruses', 'w557del', 'arrayexpress', 'novelc', 'novelfigu
re', 'novelthe', 'arita', 'nrasq61r', 'protocolduetocyp3a4interactions', '180k', 'prothro
mbin', 'nrui', 'nsd2', 'occurrent', 'nse', 'nsp', 'nssnps', '6q22', 'ntc', 'nurd', 'prote
asomally', '1522', 'proportionately', '6weeks', 'ollila', 'weickhardt', 'omics', 'and15',
'1780g', '17806', 'andel', 'andd', 'ossificans', '176876', 'pri', 'osteoclast', '1763', '
ostman', 'otx2', 'omnimax', 'winglow', 'analytics', 'analytically', 'outgrow', 'analysisv
ariant', 'previously25', 'analysismissense', 'previously11', 'analysisan', 'analysis14',
'angulated', 'aniline', 'orchestration', 'orbitrap', 'antivascular', 'procaspase', 'antiu
biquitin', 'oncologia', 'antitumoral', 'weizmann', 'oncor', 'antidigoxigenin', 'welti', '
oncostatin', 'oncotest', 'westphalia', '1786', 'opc', 'anthracyclines', 'wh0051755m2', '1
784', 'ophthalmic', 'opportunistic', 'orbit', 'anisomycin', 'notation', 'nordic', 'pretty
', 'azacytidine', 'netsurfp', 'bacq', 'vectorette']
['1356', 'd275', 'l43', 'l597q', 'toad', 'l792l', 'd368', 'l814l', 'toggle', 'l826m', 're
sorted', 'toledo', 'd275a', '4512', 'tnfr1', 'resists', 'tom', '12r', 'l862v', 'tomizawa'
, 'labvision', '461g20', 'resat', 'top2a', '4691', '2053', 'tnfr2', 'd538', 'cytoband', '
l115p', 'dcog', 'tip3', 'krkljus', 'krongrad', 'tis', 'tissuelyser', 'datto', 'datathe',
'20jc', 'dash', 'l1152p', '44a', 'd572del', 'daehak', 'd877n', 'l1301r', 'd837d', 'tlx1',
'd820g4', 'd816e', 'reuther', 'd67', 'l215p', 'l229f', 'landau', 'reported9', 'corporate'
, 'crizotinibnaive', 'csrd', 'csd34', 'leung', 'cscc', 'transfers', 'crystalline', '4g5j'
, 'reinitiate', 'crosslinker', 'lh', '200m', 'reha', 'leucinyl', '4l', 'creases', 'reh',
'cpt', 'linesalthough', 'lipofectamine2000', 'cov', 'lipogenesis', 'lipomatous', 'lis1',
'refractive', 'leukemia18', 'ctcgag', 'cyclotron', 'tpp', 'cyclohexamide', 'tothe', 'cycc
', 'replications', 'tox', 'lariat', 'laura', 'repertoires', 'cushion', 'leach', 'rep', 'c
uboidal', 'relaxing', 'leeds', 'cttacgcugaguacuucga', 'lengauer', 'lengthened', '4a4a', '
train', 'tramp', 'les', 'lethargy', 'leu1790', 'relevantly', 'rhodes', 'ddx11', 'ribogree
n', 'dhomen', 'theories', 'k659', '4067a', 'k737t', '4105', 'diaminophenylpyrimidine', 'k
756r', '4171a', 'k975e', 'kaganovich', 'rosenzweig', 'dharmafect1', '4058', 'kanai', 'dg'
, 'thermoscript', 'karyotypically', '4173a', 'ronchetti', 'deutz', '4185', 'kdm2a', 'dete
rgents', '4193t', 'themutant', 'dilation', '440g', 'k499e', 'jura', 'tfiid', 'k1409e', 'd
izziness', 'k1436q', 'k22', 'disfavors', 'k299r', 'tgggcagtatctttccagcaac', 'rptor', 'dis
cussions', 'k607t', 'thein', 'discriminatory', 'k618a', 'disciplinary', '3tp', 'dip', 'rp
5', 'k618t', '4007a', 'rowley', '400mgorally', 'routines', 'destroying', 'kdm5d', '2161',
'komolgrov', 'klh', 'kmt2b', 'thymomas', 'delk745', '4306a', 'tiangen', 'knife', 'riva',
'rita', 'tics', 'dei', 'decreaseorlessthan20', 'desperately', 'korona', 'decomposition',
'tiff', 'rimerman', 'dec', 'richelda', 'richards', '4388c', 'krantz', 'dealt', 'timescale
', 'deltyd', 'delv559v560', 'delving', 'delvv', 'desmosomes', 'design32', 'keam', 'descri
bed6', 'keeper', 'thirteenth', 'thomson', 'kelvin', 'thr180', '4225a', 'kimdw', 'deproton
ation', 'depolymerization', 'kindai', 'depletes', 'kirschner', 'deoxynucleotidyltransfera
se', 'kitor', '4235c', 'kix', 'denhardt', 'rmscf', 'demethylate', 'trays', 'lmnb2', 'dobs
on', 'radically', '558del', 'mcelhinny', 'tyr568', 'radiolabeling', 'certinib', 'radiolab
el', 'cep110', 'centroblasts', 'mcglade', 'mckenna', '1nf1a', 'tyr845', 'tyr416', 'tyr98'
, 'cells3', 'cells27', 'mcpherson', 'mcrc', 'tzahar', '5721', '13department', 'cells13',
'576del', 'celllines', 'radiosurgery', 'maxima', 'rad21', 'chemosensitive', 'txt', 'marqu
e', 'tyd', 'tyedmers', 'chk', 'rak', 'mart', 'types24', 'chiari', 'tyr1221', 'martins', '
tyr15', 'radsource', 'chemilluminescence', 'chemifluorescence', 'tyr182', '1p34', 'checks
', '1p32', 'ch2', 'mathe', 'matsuno', 'cgap', 'matsuoka', 'mdia', '1m14', 'localise', 'mg
h011', '1atp', 'mermel', 'mesa', 'ultrastructure', 'unacceptably', 'metaplastic', '19q12'
, 'methadone', 'methosulfate', 'r659p', 'mfei', 'cav1', 'cci', 'mgh045', 'mgso', 'miame',
'categorize', '19f13', 'microenvironmental', 'microfarads', '0222', 'micrognathia', 'r461
i', '02139', 'ccgg', 'ccl2', 'ceffs', 'cd69', 'me3', 'ceff', 'rabep1', '578del', 'cdk12s'
, 'cdf', 'meaningless', '13q22', 'mechali', 'mecp2', 'r97', 'medlineplus', '586a2', 'mela
nocortin', 'melanoma6', 'cd32', 'ucg', 'uch', 'uci', 'melanomais', 'cd14', 'cd11c', 'melk
i', '1department', '1t', 'chp', 'marburg', 'luciferace', 'recombine', 'computerassisted',
'lrp6', 'ls1034', 'comprehend', 'lsrfortessa', 'ltk', 'comparedwith', 'comparability', 'r
eceptive', 'ltrs', 'commit', 'recombining', 'commented', 'recalibrated', 'trov', 'trovo',
'trp117', 'lujan', 'colorless', 'colonize', 'ly19', 'trp88', 'ly3', 'concentrators', 'low
ell', 'twist1', 'continuity', 'reestablish', 'reenter', 'coqueret', 'locates', 'reedijk',
'locating', 'trended', 'triage', 'reductive', '4v', 'redirects', 'tricine', 'confused', '
lombardo', '500g', 'looney', 'trifluoroacetic', '50jc', 'trihydrate', 'trilineage', 'cons
ervationon', 'connectivity', 'lovec', 'lovly', '06469322', 'lymphangiogenesis', 'reanneal
```

```
ing', 'cj236', 'tumorous', '1tup', 'maeda', 'mag', 'mage', 'cleidocranial', '1tsr', 'clea
', 'classifications13', 'clarke', 'ras4a', 'turcot', '1355', 'maleimide', 'malfunctions',
'cir', 'cignal', 'malkin', 'maml2', 'mangoura', '1382', 'manuscripts', 'randomness', 'chr
7', 'macrosomia', '1week', 'clotting', 'tucked', 'genechem', 'colectomy', 'rccs', 'col8a1
', 'col2a1', 'cohesive', '06439015', 'rbm5', 'rbgh', 'tsuzuki', '1367', 'm1149t', 'codd',
'm1b', 'cochran', 'cobalt', 'coall', 'cng', 'm5g', 'ttingen', 'cmin', 'machens', 'cls', '
jung', 'jsnp', 'microlesions', 'ser2', '2867', 'fla', 'striped', 'strn', 'f15', 'headed',
'fitchburg', 'hegde', 'ser222', 'filamina', 'ser218', 'figure9b', 'stricto', 'hemangiobla
stoma', 'figure4e4e', 'figure4b4b', 'ser111', 'ser10', 'hemangioendothelioma', 'hemangios
arcoma', '28102', 'sty', '2800', 'heptapeptide', '1097', 'stretching', 'herman', 'hamlin'
, 'h3k18ac', '30a', 'fractionations', 'h82', 'h9010', 'haar', 'fortune', 'haeii', 'forked
', 'foregoing', 'hagiwara', 'set2', 'hcs', 'stott', 'set1', 'hani', 'harvester', 'fma3',
'fluoview', 'hatzivassiliou', 'hawkins', '2878', 'hcc1187', 'ser5', 'subcategory', 'hern'
, 'schultz', '1082', 'secured', '2622', 'hmb', 'hmm', 'sec31a', 'holmes', 'scudierio', 'h
omepage', 'scrotum', 'f7', 'f691l', 'hon', 'sedentary', 'sumoylated', 'hopeful', 'hospita
lization', 'f44', 'f367s', 'hotstart', 'housework', 'f2108l', 'hrasv12', '3350', '123a',
'fam131b', 'hkl', 'hernia', 'feso4', 'fibulae', 'fibrodysplasia', 'sensu', 'subfragments'
, 'heterodimerzation', 'hextuplicate', 'fgfr2s252w', 'subq', '2732', 'highgrade', '2696',
'hinged', 'sediment', 'hinging', 'hip', '2678', '3229', 'felix', 'hipk2', 'hips', 'hisa',
'fearon', 'farmer', 'hiv1', 'friable', 'sterne', 'h3k18', 'g909r', 'gist35', 'gist37', 'g
agacuagacaaugagaaa', 'gadgeel', 'slidethree', 'slideschematic', 'gaagaacagtattgacata', 's
pikes', 'slideassessment', 'gaaagaagacaaacagaaa', 'g97d', 'g873q', 'slx4', 'skov3', 'skov
', 'skippy', 'g779g', 'g742d', 'g691s', 'skeletons', 'glut4', 'sitespecific', 'site2', 'g
469v', 'spencer', 'spectrometers', 'sterical', 'snu2535', 'generalizability', 'gdi', 'gen
eva', '114k', 'genosplice', 'gcrma', 'solon', 'solenoid', 'soap', 'soak', 'snyder', '2p16
', 'spectrofluorometer', 'gggacaagtttgtacaaa', 'ggggaccactttgtacaagaaagctgggtcaat', 'snf5
', 'snb19', 'snapfrozen', 'gaussia', 'specialization', 'giacca', 'gastritis', 'smcd', 'sp
ectrochip', 'goksenin', 'sirt1', 'g466e', 'ship1', 'stanchina', 'shrna2', 'fyrn', 'starin
sky', 'gtype', 'guanylate', 'gugggguguuggacagugua', 'gure', 'stathmin', 'fusion14', 'fur',
'gyrata', 'g465', 'h1112', 'ste7', '3097', 'shielding', 'h1838', 'shevelev', 'steiger', '
sheng', 'fs2f', 'steponeplus', 'h249', 'gtf', '3031', 'sibley', 'stag2', 'goodman', 'gott
fried', 'singlestrand', 'govindan', 'sprycel', 'spurdle', '1108', 'grdthe', 'g322d', 'gri
gelioniene', 'grm8', 'groin', 'sry', 'grooves', 'signalway', 'grr5', '1101', 'gsf', 'gtag
atacaagtagagaat', 'sickness', 'g13v', 'sibs', '3026', 'schwannomin', 'ht1080', 'jongno',
's218', 'intensified', 'intercalation', 'intercrosses', 'intermediately', 'intraindividua
l', 'e562del', 'tanizaki', 'intravascular', 'intron1', 'e501k', 's222', 'e380q', 'e711t71
2', 'ira1p', 'ira2p', '2256', 'tasigna', 'e318', '3eqc', 's1760a', 'taufkirchen', 'e277a'
, 'e277', 'e275', 'e711', 's29', 'tbhq', 'ind', 'inasmuch', 'incapacitate', 't751insa', '
eff', 'eet', 's459f', 's459', 'eed', 'edwards', '3919a', 'edged', 'taatacgactcactataggg',
'e749del', 'indentified', 'tables11', 'initiators', 'ec50s', 'ink', '3980a', 's323g', 'ea
ger', 'intensification', 's2ab', 's297f', 'isodisomy', 's1463f', 'htseq', 'terlouw', 'tel
1', 'dsrna', 'rxa', 'rx', '2240', 'tens', 'russia', 'jenkins', 'doxycyline', 'terf2', 'do
wnstaging', 'jersey', 'duality', 'jetpei', 'ruiz', 'rui', '102k', 'dorset', 'jfcr', 'jfcr
021', 'jl', 'jmmls', 'jnj', 'jolly', 'rxrxxs', 'jak2wt', 'isolations', 'e011', 'isomer',
'e2419k', 'isopeptidase', 'itching', 'iteration', 'e1356g', 'e1327g', 'e124', 'e1210', 's
1058p', '1035', 's10147', 'dudek', 'ivs7', 'dysmorphology', 'dyskeratosis', 'dysgeusia',
'dyedeoxy', 'jaffe', 'dusp5', 'dupn771', 'dupa767', 'duncan', 'dumaz', 't725t', 't712', '
t710a', 'esteller', 'evi', 'scala', 'hyp564', 'hypermetabolic', 'eurogentec', 'euclidian'
, 'etude', 'hyponatremia', 'etch', 'hypothemycin', 'esterification', 'established11', 'su
zanne', 'esss', 'sweep', 'esrp1', 'swi2', 'sayama', '3555', 'i1307v', 'ernst', 'symposium
', '2517', 'eric', 'sv40t', 'hydroxide', '2368', 'exon6', 'externally', 'expr', 'hudson',
'expecting', 'supraclavicular', 'exor311h', 'supt1', 'exop286r', 'humoral', 'schaefer', '
schaaper', 'exon21', 'hydrophilicity', '124k', 'exon14', 'exod316n', 'surrogates', 'exod3
16', 'exocyclic', 'exibility', 'scars', 'hydrochloric', 'hydrolysing', 'exemestane', 'syn
drome21', '3610c', 'synergies', 'electrical', 'elsewhere40', 'elonginb', 'elongate', 't25
15i', '1268', 'elisas', '23a', '2374', 'electrochemical', 'electrocardiographic', 'electr
oblotting', '3902t', 'i9', 'elbaz', 't468m', 't485k', 't511a', 'imperial', 'importer', 's
4j', 's46i', 'ehmt1', 'improvision', 'inaccessibility', 't1n0m0', 's846i', '38101x', 'end
ogeneous', '365h22', 'equate', 'equality', 'ichthyosis', '24b', 'ig3', 'epitheloid', '371
9', 'igg2', 'epha1', '3743', 'igg2a', 'igiii', 'envisage', 'sakaguchi', 'sahni', 'ignatiu
s', 't123', 'ihca', 't1324n', 'engenders', 'enforce', 'energetics', 'catalysing', '2q12',
'understanding13', 'prohibits', 'arched', 'oedema', 'offsets', 'ohtsuka', '00299804', '70
01', 'aptag', 'waterman', 'okami', 'olayioye', 'appends', 'promoterless', 'progressiva',
'wc', 'apochromat', 'aplasia', 'profuse', 'oligoastrocytic', 'wean', 'apbs', 'webb', 'apa
ricio', 'arg103', 'proofs', 'proviruses', 'w557del', 'arrayexpress', 'novelc', 'novelfigu
re', 'novelthe', 'arita', 'nrasq61r', 'protocolduetocyp3a4interactions', '180k', 'prothro
mbin', 'nrui', 'nsd2', 'occurrent', 'nse', 'nsp', 'nssnps', '6q22', 'ntc', 'nurd', 'prote
asomally', '1522', 'proportionately', '6weeks', 'ollila', 'weickhardt', 'omics', 'and15',
'1780g', '17806', 'andel', 'andd', 'ossificans', '176876', 'pri', 'osteoclast', '1763', '
ostman', 'otx2', 'omnimax', 'winglow', 'analytics', 'analytically', 'outgrow', 'analysisv
ariant', 'previously25', 'analysismissense', 'previously11', 'analysisan', 'analysis14',
'angulated', 'aniline', 'orchestration', 'orbitrap', 'antivascular', 'procaspase', 'antiu
```

```
biquitin', 'oncologia', 'antitumoral', 'weizmann', 'oncor', 'antidigoxigenin', 'welti', '
oncostatin', 'oncotest', 'westphalia', '1786', 'opc', 'anthracyclines', 'wh0051755m2', '1
784', 'ophthalmic', 'opportunistic', 'orbit', 'anisomycin', 'notation', 'nordic', 'pretty
', 'azacytidine', 'netsurfp', 'bacq', 'vectorette']
```

In [0]:

```
df1=pd.DataFrame()
df1['words']=words2
df1['class']=y_true[:1000]
```

In [0]:

```
X=df1['words']
y=df1['class']
```

In [96]:

```
X.shape
```

Out[96]:

```
(1000,)
```

Out[96]:

```
(1000,)
```

In [97]:

```
y.shape
```

Out[97]:

```
(1000,)
```

Out[97]:

```
(1000,)
```

In [98]:

```
X.shape
```

Out[98]:

```
(1000,)
```

Out[98]:

```
(1000,)
```

In [0]:

```
# split the data into test and train by maintaining same distribution of output varaible
'y_true' [stratify=y_true]
X_train, X_test, y_train, y_test = train_test_split(X,y, stratify=y, test_size=0.2)
# split the train data into train and cross validation by maintaining same distribution o
f output varaible 'y_train' [stratify=y_train]
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test
_size=0.2)
```

In [100]:

```
print(X_train.shape)
print(X_test.shape)
print(X_cv.shape)
print(y_train.shape)
print(y_test.shape)
print(y_cv.shape)
```

```
(640,)
(200,)
(160,)
```

```
(640,)
(200,)
(160,)
(640,)
(200,)
(160,)
(640,)
(200,)
(160,)
```

In [0]:

```
# one-hot encoding of variation feature.
text_vectorizer = TfidfVectorizer()
train_text_feature_onehotCoding = text_vectorizer.fit_transform(X_train)
test_text_feature_onehotCoding = text_vectorizer.transform(X_test)
cv_text_feature_onehotCoding = text_vectorizer.transform(X_cv)

# building a TfidfVectorizer with all the words that occured minimum 3 times in train dat
a
```

In [0]:

```
train_text_feature_onehotCoding=normalize(train_text_feature_onehotCoding)
cv_text_feature_onehotCoding=normalize(cv_text_feature_onehotCoding)
test_text_feature_onehotCoding=normalize(test_text_feature_onehotCoding)
```

In [103]:

```
print(train_text_feature_onehotCoding.shape)
print(cv_text_feature_onehotCoding.shape)
print(test_text_feature_onehotCoding.shape)
```

```
(640, 640)
(160, 640)
(200, 640)
(640, 640)
(160, 640)
(200, 640)
```

In [104]:

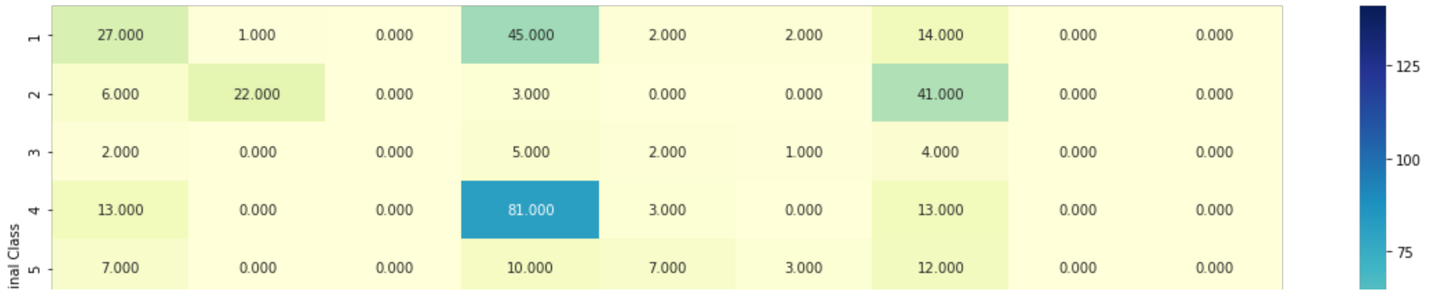
```
alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, rand
om_state=42, n_jobs=-1)
      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


best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max
_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(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 estimator = ', alpha[int(best_alpha/2)], "The train log loss is
:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validation
log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_text_feature_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss is:
",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

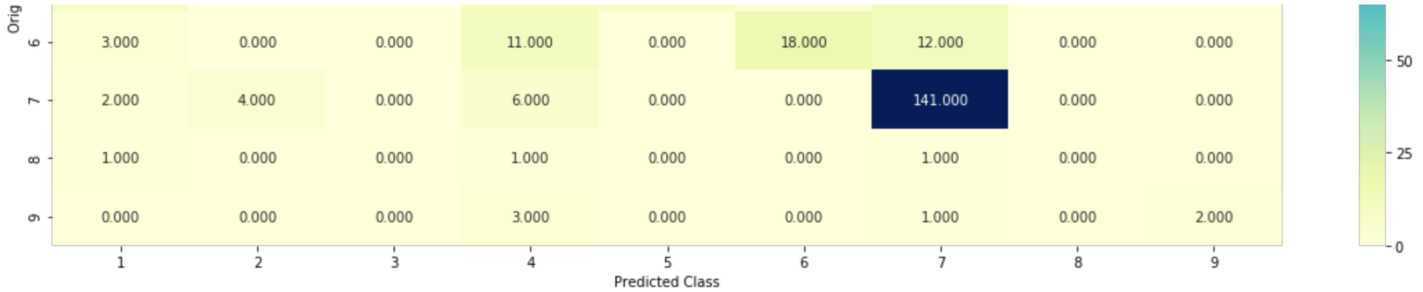
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.7617908686999926
for n_estimators = 100 and max depth =  10
Log Loss : 1.7617908687267039
for n_estimators = 200 and max depth =  5
Log Loss : 1.7617908687370012
for n_estimators = 200 and max depth =  10
Log Loss : 1.7617908687636081
for n_estimators = 500 and max depth =  5
Log Loss : 1.761790868733728
for n_estimators = 500 and max depth =  10
Log Loss : 1.7617908687586794
for n_estimators = 1000 and max depth =  5
Log Loss : 1.7617908687258528
for n_estimators = 1000 and max depth =  10
Log Loss : 1.7617908687491877
for n_estimators = 2000 and max depth =  5
Log Loss : 1.7617908687323727
for n_estimators = 2000 and max depth =  10
Log Loss : 1.761790868758201
For values of best estimator =  100 The train log loss is: 1.7554418868794692
For values of best estimator =  100 The cross validation log loss is: 1.7617908686999926
For values of best estimator =  100 The test log loss is: 1.7487702389291113
for n_estimators = 100 and max depth =  5
Log Loss : 1.7617908686999926
for n_estimators = 100 and max depth =  10
Log Loss : 1.7617908687267039
for n_estimators = 200 and max depth =  5
Log Loss : 1.7617908687370012
for n_estimators = 200 and max depth =  10
Log Loss : 1.7617908687636081
for n_estimators = 500 and max depth =  5
Log Loss : 1.761790868733728
for n_estimators = 500 and max depth =  10
Log Loss : 1.7617908687586794
for n_estimators = 1000 and max depth =  5
Log Loss : 1.7617908687258528
for n_estimators = 1000 and max depth =  10
Log Loss : 1.7617908687491877
for n_estimators = 2000 and max depth =  5
Log Loss : 1.7617908687323727
for n_estimators = 2000 and max depth =  10
Log Loss : 1.761790868758201
For values of best estimator =  100 The train log loss is: 1.7554418868794692
For values of best estimator =  100 The cross validation log loss is: 1.7617908686999926
For values of best estimator =  100 The test log loss is: 1.7487702389291113
```

In [105]:

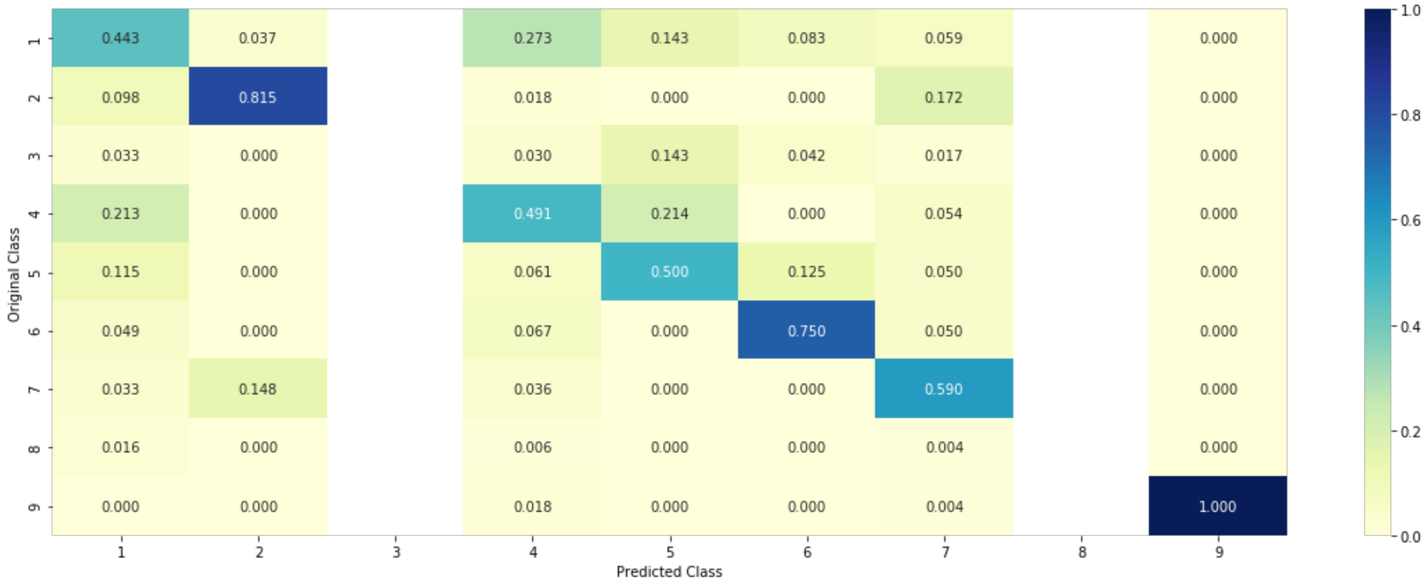
```
clf = RandomForestClassifier(n_estimators=100, criterion='gini', max_depth=5, random_sta
te=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, c
lf)
```

```
Log loss : 1.2350521139520694
Number of mis-classified points : 0.4398496240601504
-------------------- Confusion matrix --------------------
```

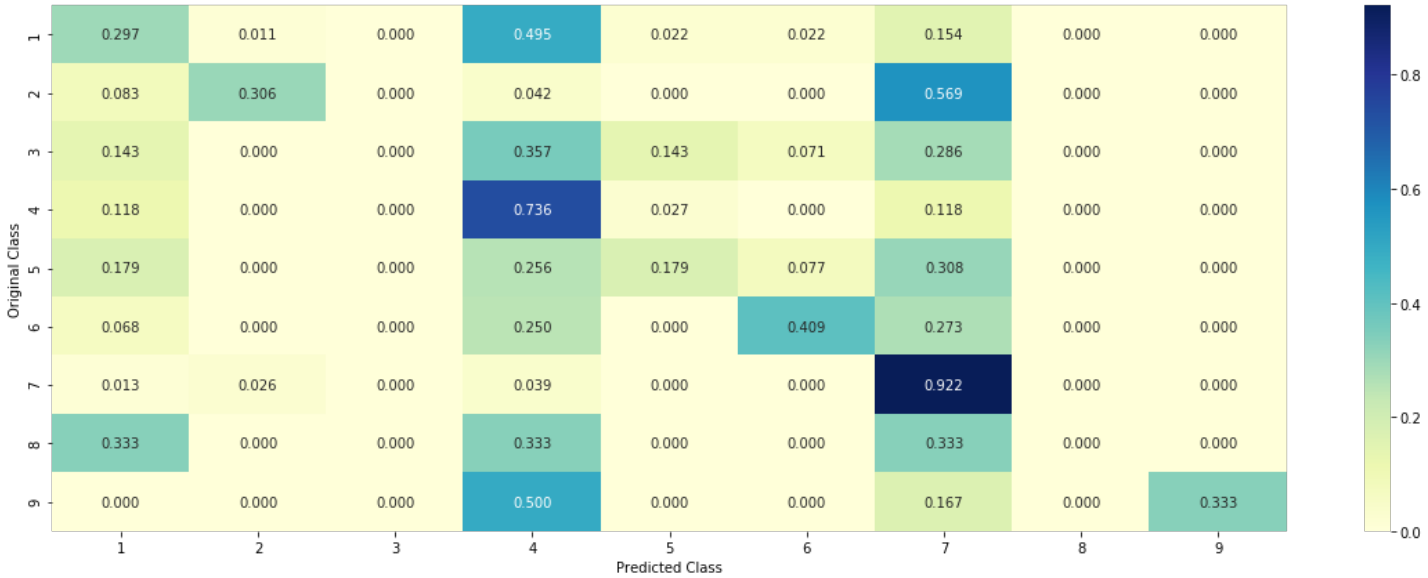




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



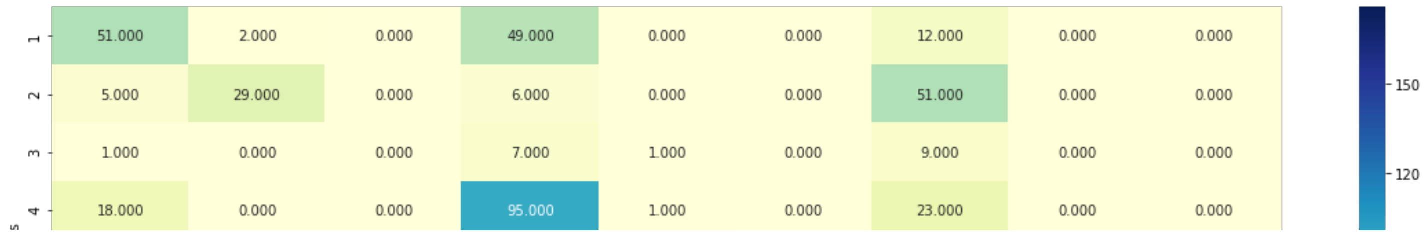
```
-------------------- Recall matrix (Row sum=1) --------------------
```

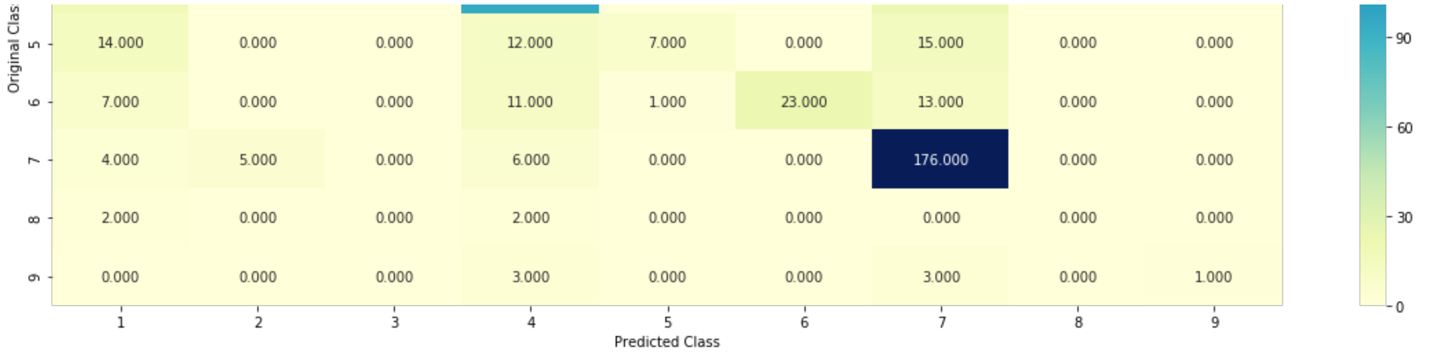


In [106]:

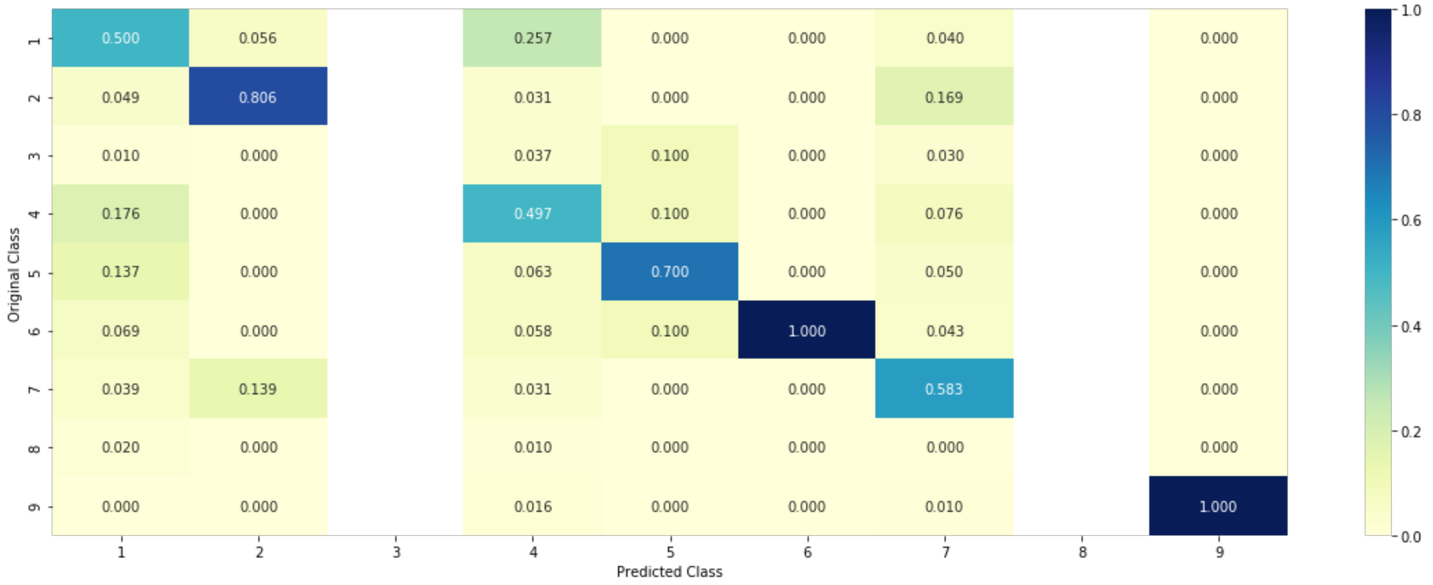
```
clf = RandomForestClassifier(n_estimators=100, criterion='gini', max_depth=5, random_sta
te=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,test_x_onehotCoding,test_
y, clf)
```

```
Log loss : 1.2413278995346566
Number of mis-classified points : 0.4255639097744361
-------------------- Confusion matrix --------------------
```

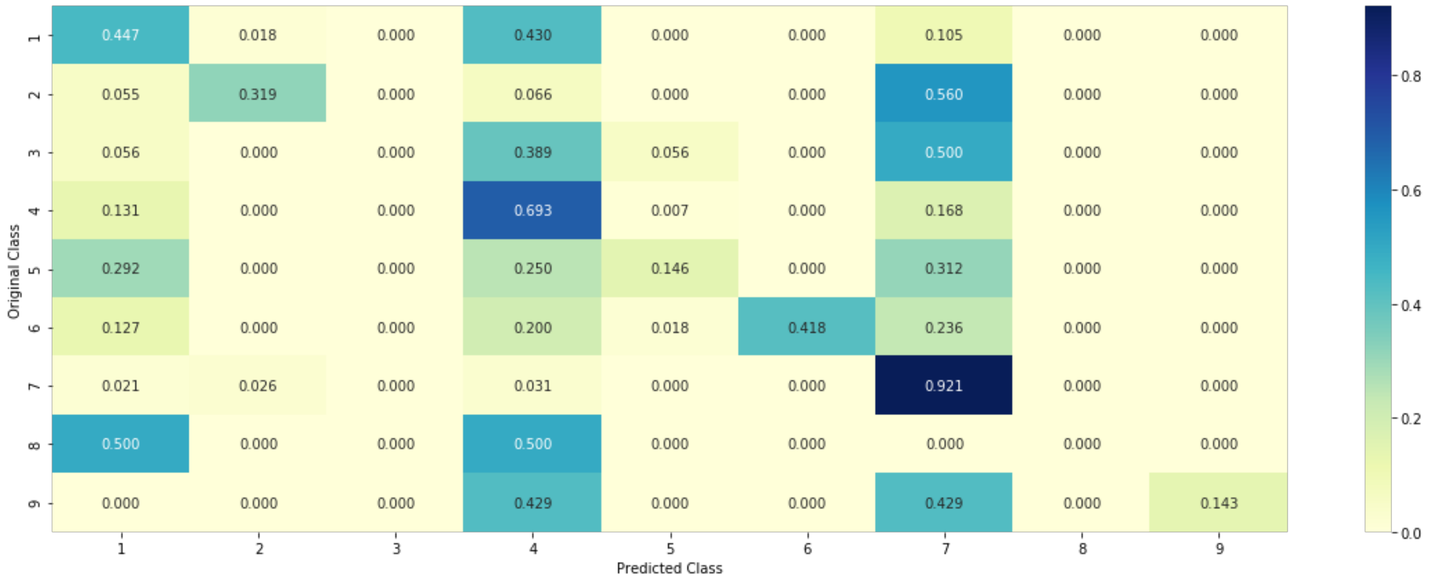




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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



In [0]:

```

```

In [0]:

```

```

In [0]:

```

```

3)Featurizing by Count Vectorizer Using Both Uni-Grams and Bi-Grams

In [0]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer(ngram_range=(1,2))
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [0]:

```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer(ngram_range=(1,2))
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 [0]:

```
# building a TfidfVectorizer with all the words that occured minimum 3 times in train data

text_vectorizer =  CountVectorizer(ngram_range=(1,2),min_df=10)
```

In [0]:

```
# don't forget to normalize every feature
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

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

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

In [0]:

```
train_variation_feature_onehotCoding = normalize(train_variation_feature_onehotCoding)
test_variation_feature_onehotCoding = normalize(test_variation_feature_onehotCoding)
cv_variation_feature_onehotCoding = normalize(cv_variation_feature_onehotCoding)
```

In [0]:

```
train_gene_feature_onehotCoding = normalize(train_gene_feature_onehotCoding)
test_gene_feature_onehotCoding = normalize(test_gene_feature_onehotCoding)
cv_gene_feature_onehotCoding = normalize(cv_gene_feature_onehotCoding)
```

In [0]:

```
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)).tocs
r()
cv_y = np.array(list(cv_df['Class']))
```

In [114]:

```
train_x_onehotCoding.shape
```

Out[114]:

```
(2124, 238219)
```

In [115]:

```
test_x_onehotCoding.shape
```

Out[115]:

```
(665, 238219)
```

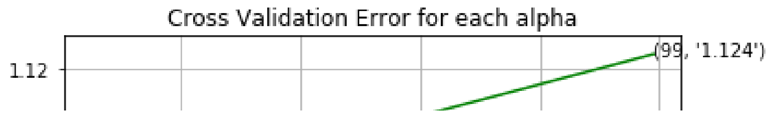
In [0]:

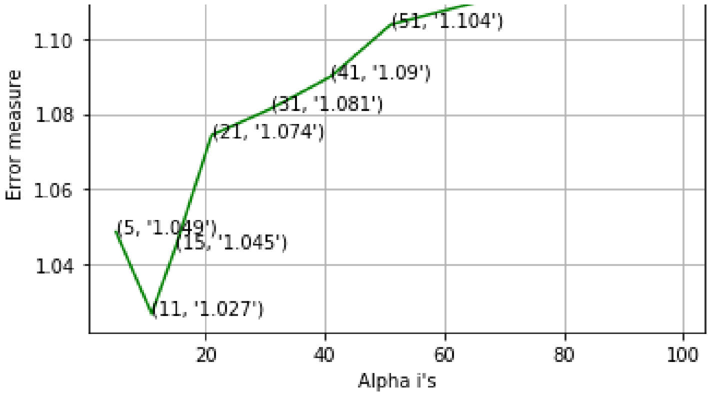
```

```

In [116]:

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

```
for alpha = 5
Log Loss : 1.0485171536685614
for alpha = 11
Log Loss : 1.0266737917551572
for alpha = 15
Log Loss : 1.04451433797169
for alpha = 21
Log Loss : 1.0742528185503082
for alpha = 31
Log Loss : 1.0813710581610474
for alpha = 41
Log Loss : 1.0899426586058885
for alpha = 51
Log Loss : 1.1037369636849341
for alpha = 99
Log Loss : 1.1239145158200101
```





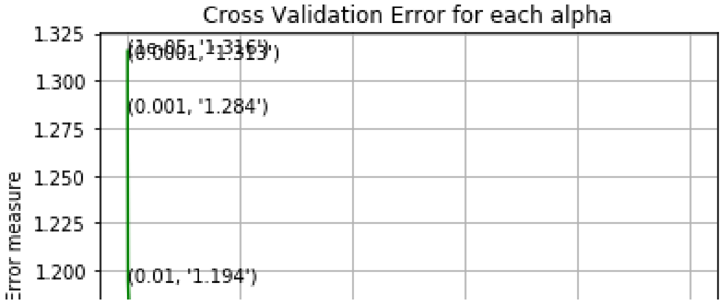
In [0]:

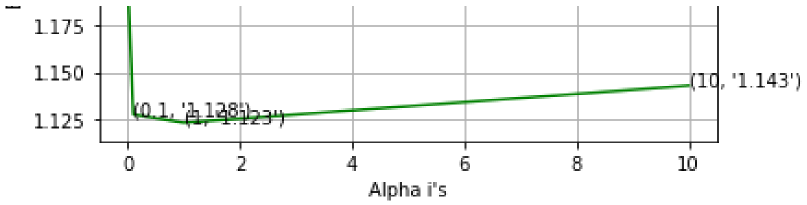
In [0]:

In [117]:

```
ca = [10 ** x for x in range(-5, 2)]
cv_log_error_array = []
for i in ca:
    print("for alpha =", i)
    clf = LogisticRegression(class_weight='balanced', C=i, penalty='l2', random_state=4)
    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, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estim
ates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(ca, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((ca[i],str(txt)), (ca[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()
```

```
for alpha = 1e-05
Log Loss : 1.3156656394104416
for alpha = 0.0001
Log Loss : 1.3127603901396598
for alpha = 0.001
Log Loss : 1.2843889895292018
for alpha = 0.01
Log Loss : 1.1941893462247517
for alpha = 0.1
Log Loss : 1.1277501937793644
for alpha = 1
Log Loss : 1.1233792346832454
for alpha = 10
Log Loss : 1.1428413650044529
```





In [118]:

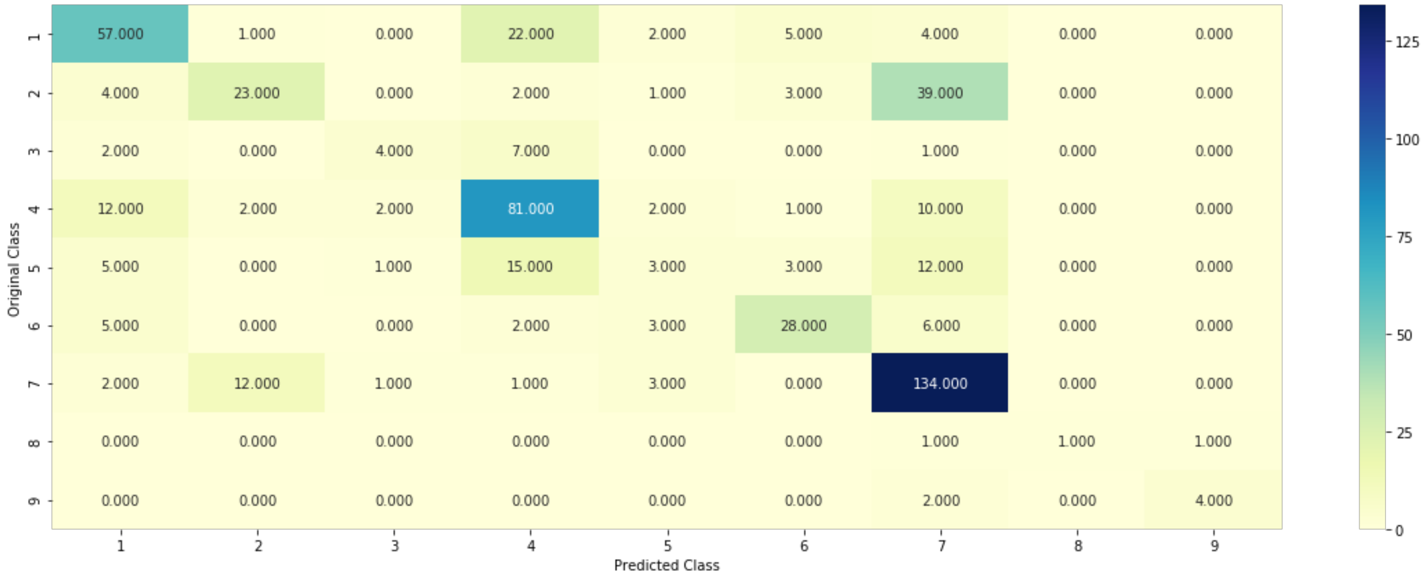
```
best_c = np.argmin(cv_log_error_array)
clf = LogisticRegression(class_weight='balanced', C=best_c, penalty='l2',  random_state=
42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best c = ', best_c, "The train log loss is:",log_loss(train_y, pred
ict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best c = ', best_c, "The cross validation log loss is:",log_loss(cv_
y, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best c = ', best_c, "The test log loss is:",log_loss(test_y, predic
t_y, labels=clf.classes_, eps=1e-15))
```

```
For values of best c =  5 The train log loss is: 0.628830465345872
For values of best c =  5 The cross validation log loss is: 1.1315546759324493
For values of best c =  5 The test log loss is: 1.0723568702619755
```

In [119]:

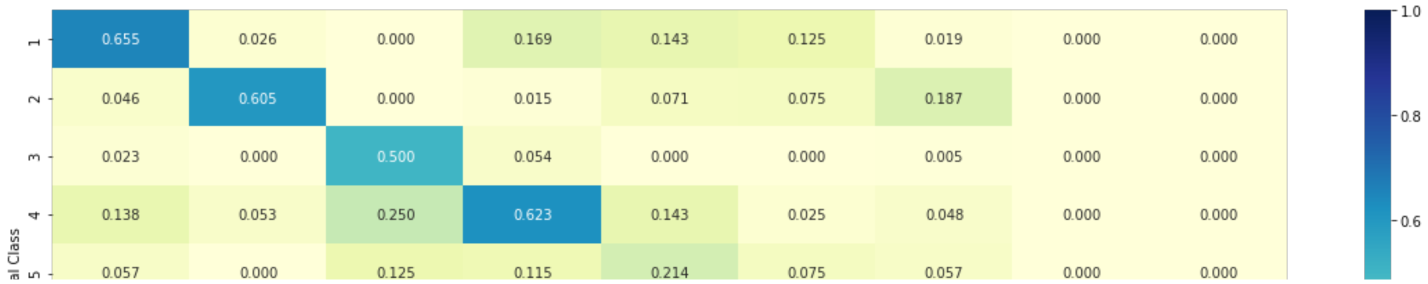
```
clf = LogisticRegression(class_weight='balanced', C=best_c, penalty='l2', random_state=4
2)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y,
clf)
```

```
Log loss : 1.1315546759324493
Number of mis-classified points : 0.37030075187969924
-------------------- Confusion matrix --------------------
```

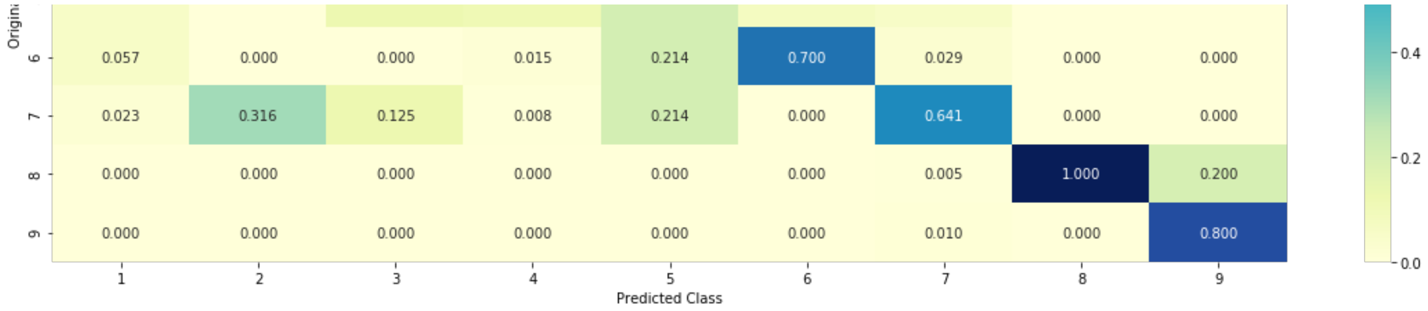


	1	2	3	4	5	6	7	8	9
1	57.000	1.000	0.000	22.000	2.000	5.000	4.000	0.000	0.000
2	4.000	23.000	0.000	2.000	1.000	3.000	39.000	0.000	0.000
3	2.000	0.000	4.000	7.000	0.000	0.000	1.000	0.000	0.000
4	12.000	2.000	2.000	81.000	2.000	1.000	10.000	0.000	0.000
5	5.000	0.000	1.000	15.000	3.000	3.000	12.000	0.000	0.000
6	5.000	0.000	0.000	2.000	3.000	28.000	6.000	0.000	0.000
7	2.000	12.000	1.000	1.000	3.000	0.000	134.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	1.000	1.000	1.000
9	0.000	0.000	0.000	0.000	0.000	0.000	2.000	0.000	4.000

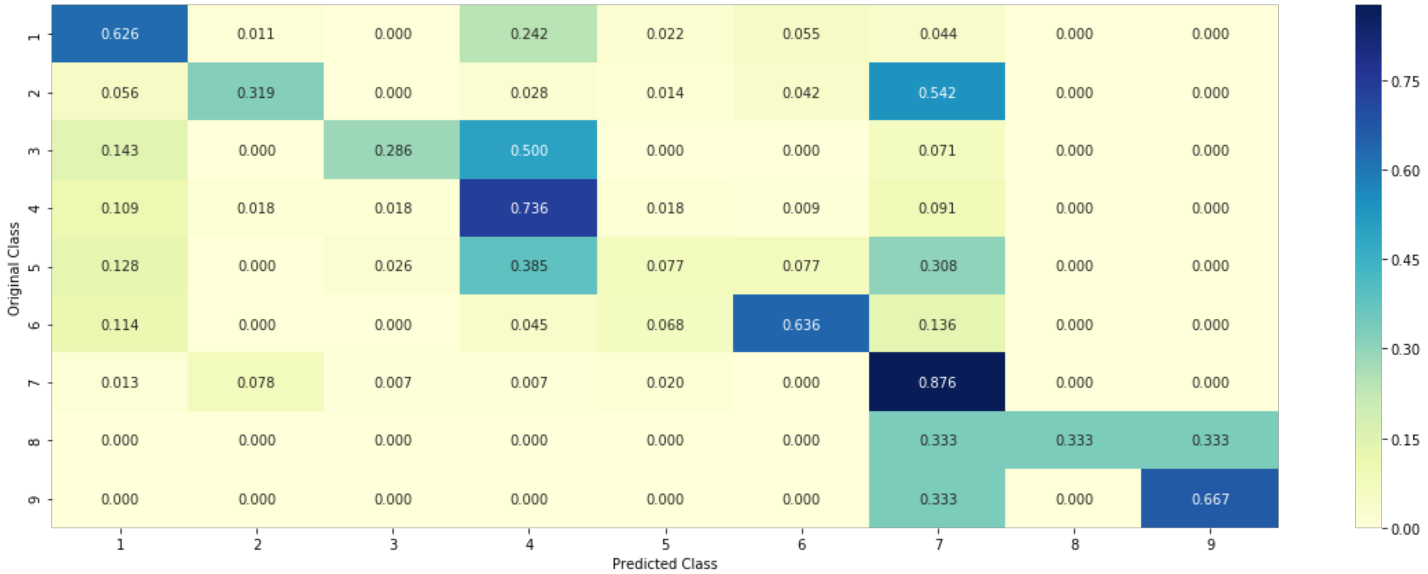
```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



1	0.655	0.026	0.000	0.169	0.143	0.125	0.019	0.000	0.000
2	0.046	0.605	0.000	0.015	0.071	0.075	0.187	0.000	0.000
3	0.023	0.000	0.500	0.054	0.000	0.000	0.005	0.000	0.000
4	0.138	0.053	0.250	0.623	0.143	0.025	0.048	0.000	0.000
5	0.057	0.000	0.125	0.115	0.214	0.075	0.057	0.000	0.000



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



In [0]:

In [0]:

In [0]:

In [0]:

4) Feature Engineering

In [0]:

```
result = pd.merge(data, data_text,on='ID', how='left')
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

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

In [0]:

```
# get_gv_fea_dict: Get Gene varaition Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    value_count = train_df[feature].value_counts()
    gv_dict = dict()
    for i, denominator in value_count.items():
        vec=[]
        for k in range(1,10):

            cls_cnt = train_df.loc[(train_df['Class']==k) & (train_df[feature]==i)]
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    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 = []
    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])
    return gv_fea
```

In [0]:

```
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [0]:

```
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Varia
tion'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation']
)
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [0]:

```
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
import math

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

In [0]:

```
text_vectorizer = TfidfVectorizer()
```

```
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it occured
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))
print("Total number of unique words in train data :", len(train_text_features))


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)
```

```
Total number of unique words in train data : 123380
```

In [0]:

```
# don't forget to normalize every feature
train_text_feature_onehotCoding = text_vectorizer.transform(train_df['TEXT'])
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

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

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

In [0]:

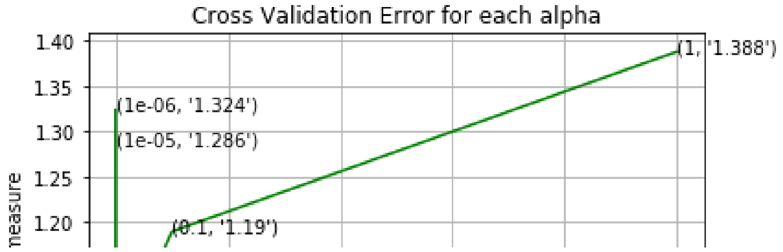
```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding))

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']))
```

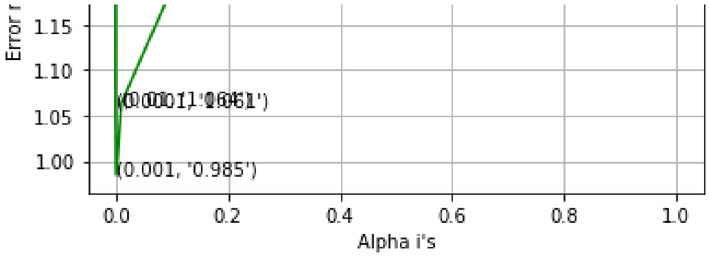
```
cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).tocs
r()
cv_y = np.array(list(cv_df['Class']))
```

In [0]:

```
alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', rand
om_state=0)
    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-1
5))
    # to avoid rounding error while multiplying probabilites we use log-probability estim
ates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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

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

```
for alpha = 1e-06
Log Loss : 1.323657450535767
for alpha = 1e-05
Log Loss : 1.2857083290097742
for alpha = 0.0001
Log Loss : 1.0612583372062478
for alpha = 0.001
Log Loss : 0.9850343663687395
for alpha = 0.01
Log Loss : 1.0636460233716885
for alpha = 0.1
Log Loss : 1.1897524491929794
for alpha = 1
Log Loss : 1.38769382568496
```





```
For values of best alpha =  0.001 The train log loss is: 0.48947175785231106
For values of best alpha =  0.001 The cross validation log loss is: 0.9862292130558538
For values of best alpha =  0.001 The test log loss is: 1.0992014131729917
```

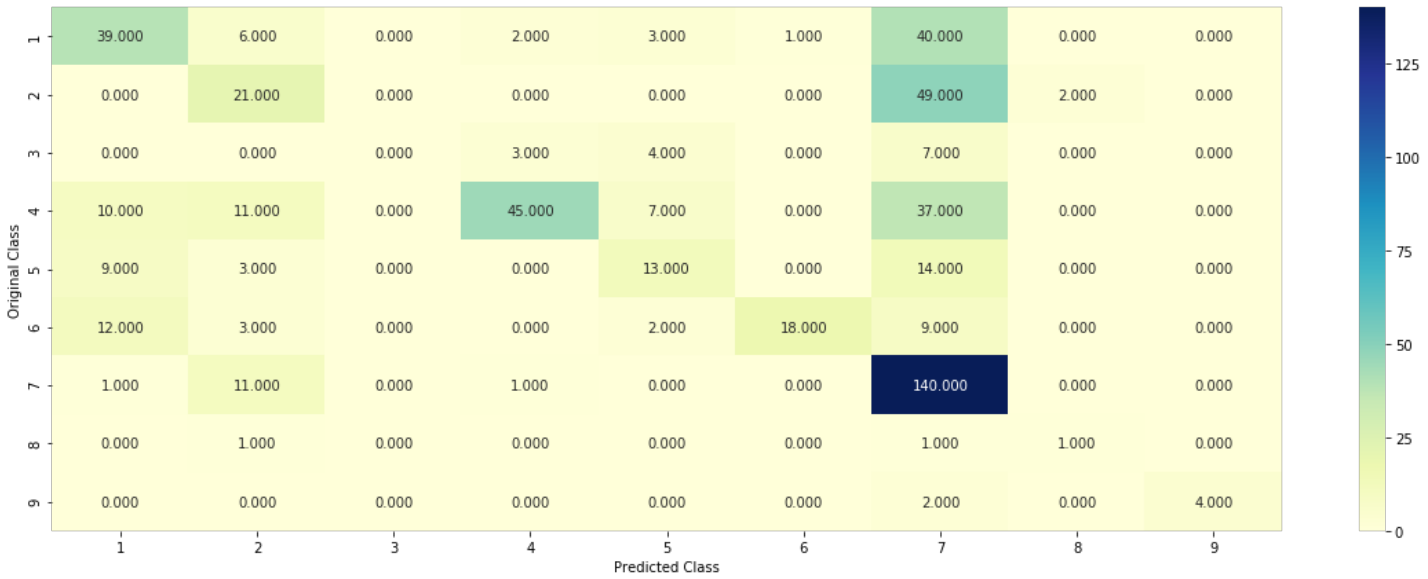
In [0]:

```

```

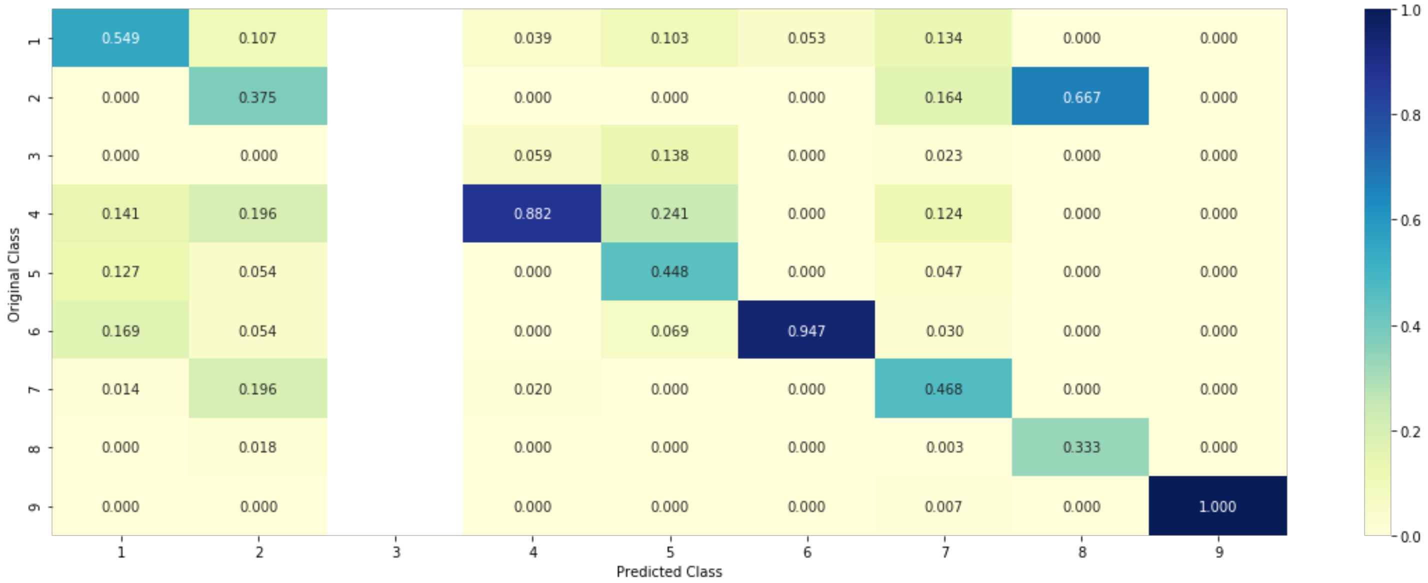
In [0]:

```
clf = SGDClassifier(alpha=best_alpha, penalty='l2', loss='hinge', random_state=42,class_
weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, c
lf)
```

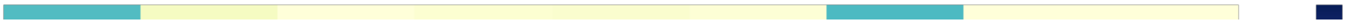
```
Log loss : 1.4338545648358878
Number of mis-classified points : 0.4718045112781955
-------------------- Confusion matrix --------------------
```

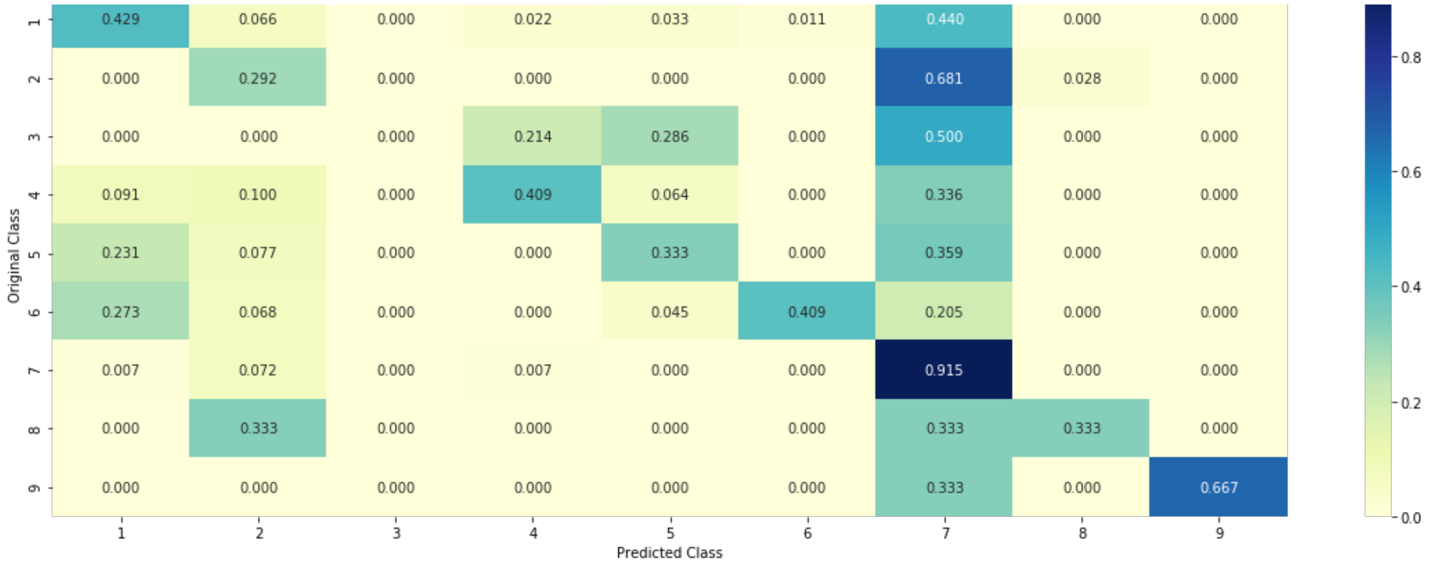


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





	1	2	3	4	5	6	7	8	9
1	0.429	0.066	0.000	0.022	0.033	0.011	0.440	0.000	0.000
2	0.000	0.292	0.000	0.000	0.000	0.000	0.681	0.028	0.000
3	0.000	0.000	0.000	0.214	0.286	0.000	0.500	0.000	0.000
4	0.091	0.100	0.000	0.409	0.064	0.000	0.336	0.000	0.000
5	0.231	0.077	0.000	0.000	0.333	0.000	0.359	0.000	0.000
6	0.273	0.068	0.000	0.000	0.045	0.409	0.205	0.000	0.000
7	0.007	0.072	0.000	0.007	0.000	0.000	0.915	0.000	0.000
8	0.000	0.333	0.000	0.000	0.000	0.000	0.333	0.333	0.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.333	0.000	0.667

In [0]:

```

```

In [0]:

```

```

Feature Engineering by adding ngram_range=(1,4)

In [0]:

```
result = pd.merge(data, data_text,on='ID', how='left')
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

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

In [0]:

```
# get_gv_fea_dict: Get Gene varaition Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    value_count = train_df[feature].value_counts()
    gv_dict = dict()
    for i, denominator in value_count.items():
        vec=[]
        for k in range(1,10):

            cls_cnt = train_df.loc[(train_df['Class']==k) & (train_df[feature]==i)]
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    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 = []
    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])
    return gv_fea
```

In [0]:

```
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])

variation_vectorizer = TfidfVectorizer()
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 [0]:

```
text_vectorizer = TfidfVectorizer(ngram_range=(1, 4), min_df = 20, max_features = 2300)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it occured
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))
print("Total number of unique words in train data :", len(train_text_features))


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

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


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

In [0]:

```
# don't forget to normalize every feature
train_text_feature_onehotCoding = text_vectorizer.transform(train_df['TEXT'])
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

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

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

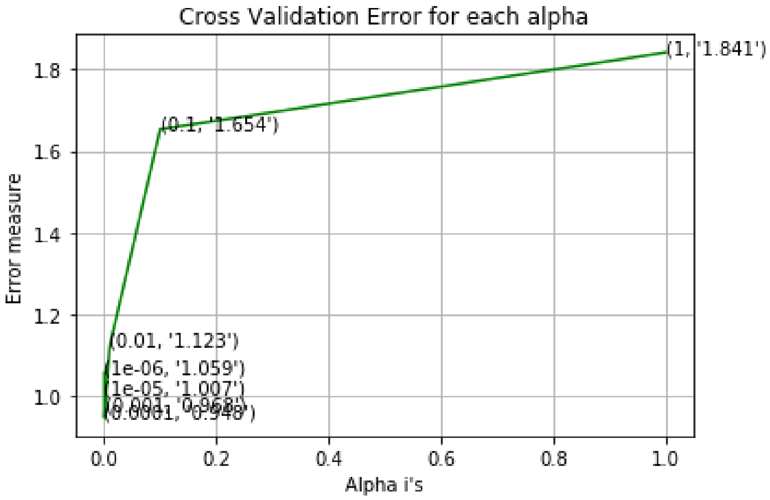
In [0]:

```
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']))
```

In [164]:

```
alpha = [10 ** x for x in range(-6, 1)]
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=0)
    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], penalty='l2', loss='log')
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
for alpha = 1e-06
Log Loss : 1.0587504803141206
for alpha = 1e-05
Log Loss : 1.00684923989845
for alpha = 0.0001
Log Loss : 0.9480621219817159
for alpha = 0.001
Log Loss : 0.9675200998068222
for alpha = 0.01
Log Loss : 1.6536057771681516
for alpha = 1
Log Loss : 1.8411876360875463
```

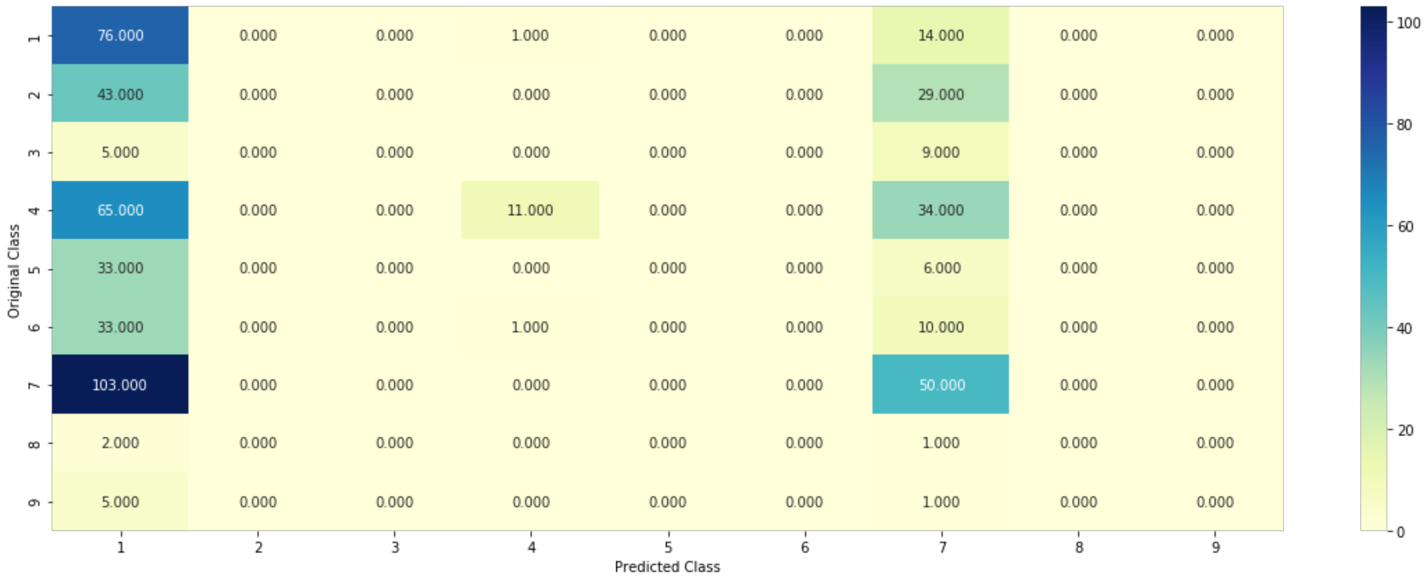


```
For values of best alpha =  0.0001 The train log loss is: 0.39278374472987404
For values of best alpha =  0.0001 The cross validation log loss is: 0.9456280035863609
For values of best alpha =  0.0001 The test log loss is: 0.9946653296889651
```

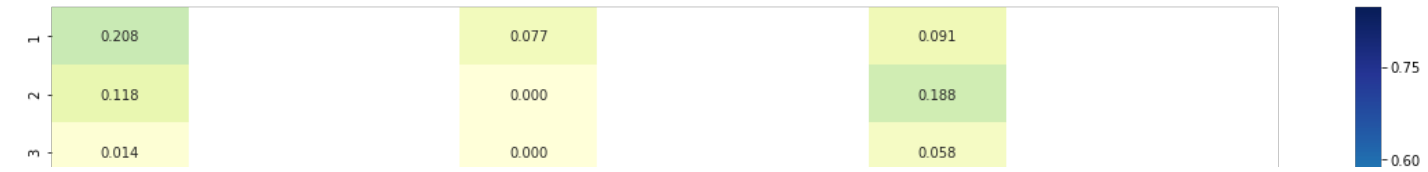
In [165]:

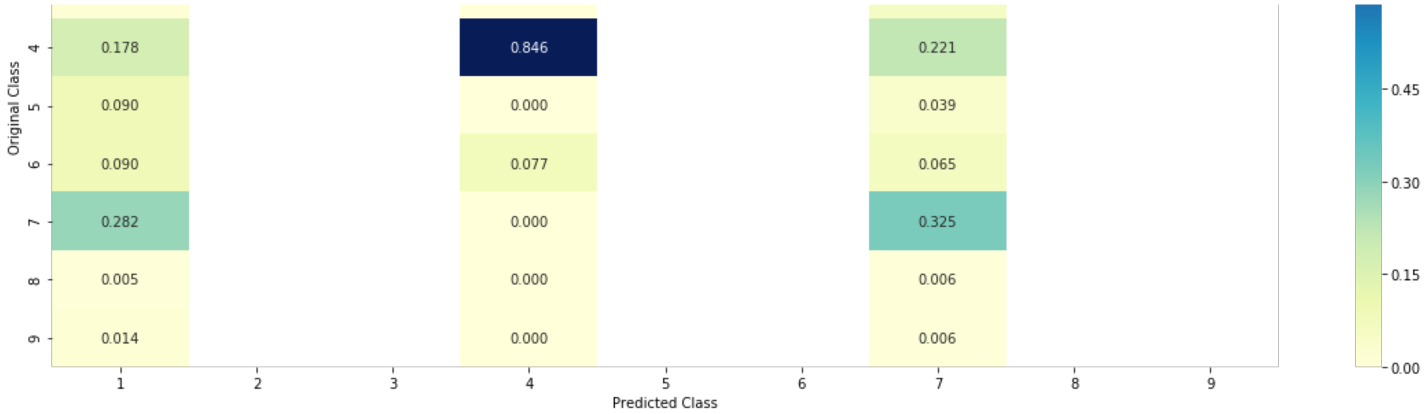
```
clf = SGDClassifier(alpha=best_alpha, penalty='l2', loss='log', random_state=41,class_we
ight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, c
lf)
```

```
Log loss : 1.852038924784792
Number of mis-classified points : 0.7424812030075187
-------------------- Confusion matrix --------------------
```

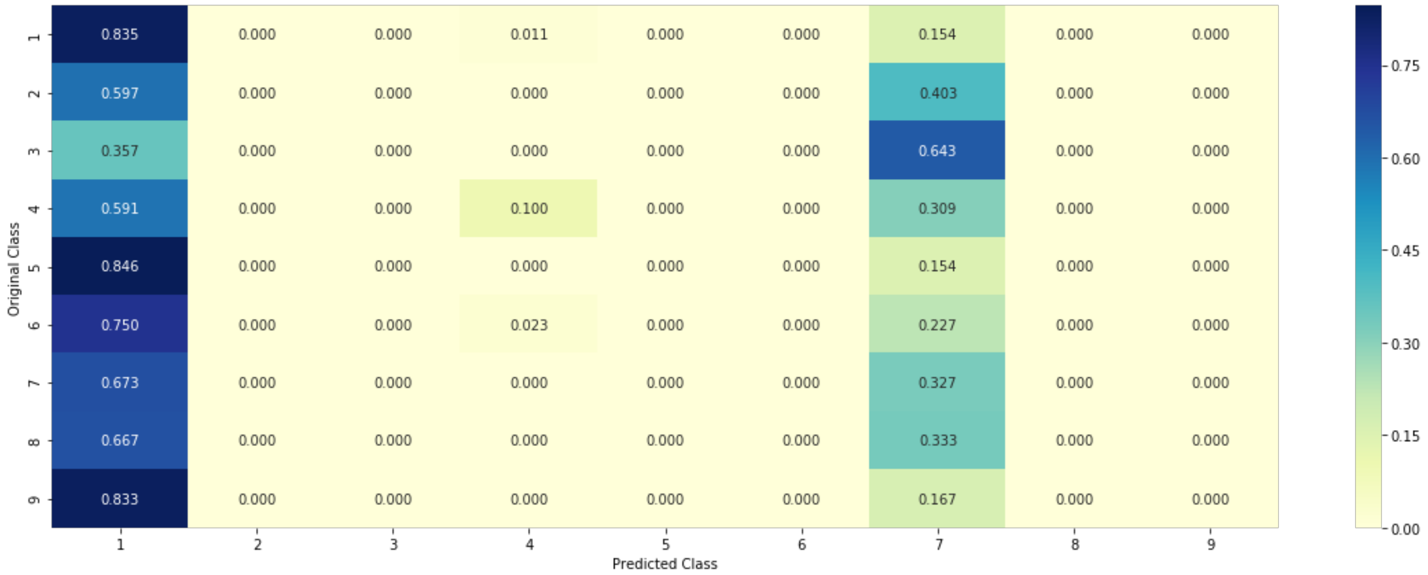


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



In [0]:

In [0]:

In [0]:

In [0]:

Conclusions

Conclusion 1

In [148]:

```
from prettytable import PrettyTable

x=PrettyTable()

model=['Naive Bayes  ','KNN','Logistic Regression With Class balancing  ','Logistic Regression Without Class balancing','Linear SVM  ','Random Forest Classifier With One hot Encoding','Random Forest Classifier With Response Coding','Stack Models:LR+NB+SVM','Maximum Voting classifier']
train =[0.91,0.71,0.47,0.47,0.51,0.62,0.05,0.47,0.79]
cv=[1.19,1.05,1.08,1.11,1.13, 1.152,1.24,1.19,1.12]
```

```
test = [1.22,1.10,1.09,1.09,1.11,1.155,1.34,1.16,0.36]
misclassified_points=[0.40,0.36,0.35,0.35,0.36,0.38,0.42,0.37,0.36]

x.add_column("model",model)
x.add_column("train",train)
x.add_column("cv",cv)
x.add_column("test",test)
x.add_column("% Misclassified Points",misclassified_points)
print(x)
```

```
+------------------------------------------------+-------+-------+-------+-------------
-----------+
|                     model                      | train |   cv  |  test | % Misclassifi
ed Points |
+------------------------------------------------+-------+-------+-------+-------------
-----------+
|                 Naive Bayes                    |  0.91 |  1.19 |  1.22 |          0.4
|
|                      KNN                       |  0.71 |  1.05 |  1.1  |          0.3
6          |
|   Logistic Regression With Class balancing     |  0.47 |  1.08 |  1.09 |          0.35
|
|  Logistic Regression Without Class balancing   |  0.47 |  1.11 |  1.09 |          0.35
|
|                  Linear SVM                    |  0.51 |  1.13 |  1.11 |          0.36
|
| Random Forest Classifier With One hot Encoding |  0.62 | 1.152 | 1.155 |          0.38
|
| Random Forest Classifier With Response Coding  |  0.05 |  1.24 |  1.34 |          0.42
|
|             Stack Models:LR+NB+SVM             |  0.47 |  1.19 |  1.16 |          0.37
|
|           Maximum Voting classifier            |  0.79 |  1.12 |  0.36 |          0.36
|
+------------------------------------------------+-------+-------+-------+-------------
-----------+
```

Conclusion 2

Using 1000 words,Count Vectorizer and Feature Engineering

In [166]:

```
y=PrettyTable()

model=['Top 1000 features on Random Forest', 'Count Vec on logistic Regression', 'Feature Engineering','Feature Engineering by ngrams=(1,4)']
train =[1.75,0.60,0.48,0.39]
cv=[1.76,1.08,0.98,0.94]
test = [1.75,1.13,1.09,0.99]

y.add_column("model",model)
y.add_column("train",train)
y.add_column("cv",cv)
y.add_column("test",test)
print(y)
```

```
+-------------------------------------+-------+------+------+
|                model                | train |  cv  | test |
+-------------------------------------+-------+------+------+
|  Top 1000 features on Random Forest |  1.75 | 1.76 | 1.75 |
|   Count Vec on logistic Regression  |  0.6  | 1.08 | 1.13 |
|         Feature Engineering         |  0.48 | 0.98 | 1.09 |
| Feature Engineering by ngrams=(1,4) |  0.39 | 0.94 | 0.99 |
+-------------------------------------+-------+------+------+
```

1) By the above values of Conclusion 1, we can tell that Logistic Regression and Random forest gave good values of less log loss compared to others. It is better to use TfIdf vectorizer with those models to get good

results.
2) By the Conclusion 2, we can tell that TfIdf gave a better score compared to Count Vectorizer values. After Feature Engineering for the first time, from the Conclusion 1, we got more accurate results compared to the above values.After doing feature engineering with other method, we got a much better results compared to the previous values.

In [0]:

```
```

In [0]:

```
```