Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [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
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 sklearn.linear_model import LogisticRegression
from collections import Counter
from scipy.sparse import hstack
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
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.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [5]:

```
data = pd.read_csv('training_variants')
print('Number of data points : ', data.shape[0])
data.head(3)
```

```
Number of data points :  3321
```

Out[5]:

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

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

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

3.1.1.1 Preprocessing Gene and Variation Data

In [29]:

```
data.Gene=data.Gene.str.replace('\s+', '_')
data.Variation=data.Variation.str.replace('\s+', '_')
```

3.1.2. Reading Text Data

In [21]:

```
# note the seprator in this file
data_text =pd.read_csv("training_text",sep="\|\|",engine="python",names=["ID","TEXT"],skiprows=1)
print('Number of data points : ', data_text.shape[0])
data_text.head(3)
```

```
Number of data points :  3321
```

Out[21]:

	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...

3.1.3 Checking for null values in Text data

In [22]:

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

Out[22]:

	ID	TEXT
1109	1109	NaN
1277	1277	NaN
1407	1407	NaN
1639	1639	NaN
2755	2755	NaN

In [23]:

```
data_text.loc[data_text["TEXT"].isnull(),"TEXT"]=data["Gene"]+"_"+data["Variation"]
```

In [24]:

```
data_text.loc[1277]
```

Out[24]:

```
ID                              1277
```

```
TEXT    ARID5B_Truncating Mutations
Name: 1277, dtype: object
```

3.1.4 Preprocessing Text data

In [25]:

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

def testPreprocessing(textval):
    output=[]
    for text in textval.values:
        string=""
        # replace every special char with space
        total_text = re.sub('[^a-zA-Z0-9\n]', ' ', 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 + " "
        output.append(string)
    return output

data_text["TEXT"]=testPreprocessing(data_text["TEXT"])
```

3.1.5 Merging Text, Gene and Variation

In [30]:

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

Out[30]:

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

3.2 Data split

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

In [34]:

```
y_true = result['Class'].values
# 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
)
```

```
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
```

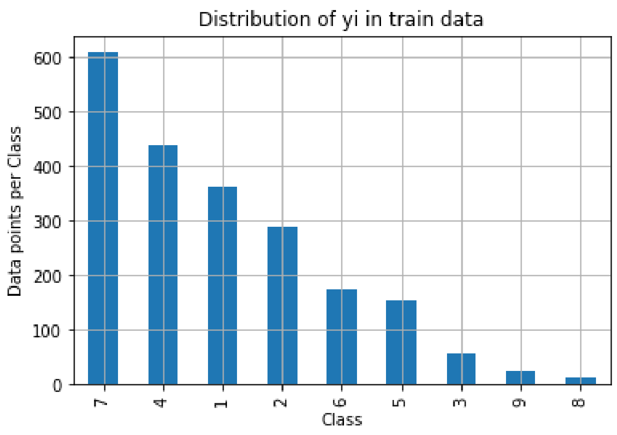
We split the data into train, test and cross validation data sets, preserving the ratio of class distribution in the original data set

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

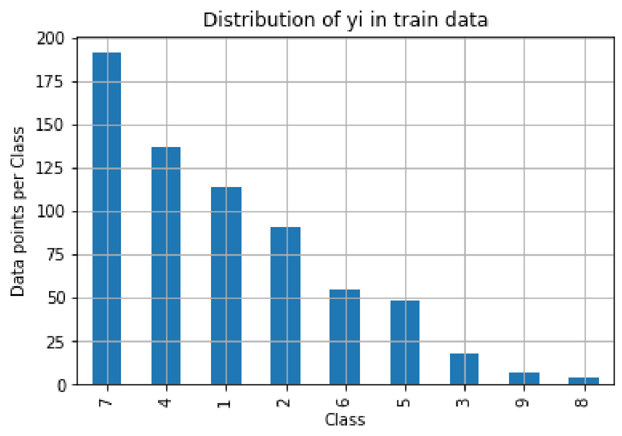
In [36]:

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



# 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()
test_class_distribution = test_df['Class'].value_counts()
cv_class_distribution = cv_df['Class'].value_counts()

print('-'*40,"Train data",'-'*40)
plotDist(train_class_distribution,train_df)

print('-'*40,"Test data",'-'*40)
plotDist(test_class_distribution,test_df)

print('-'*40,"CV data",'-'*40)
plotDist(cv_class_distribution,cv_df)
```

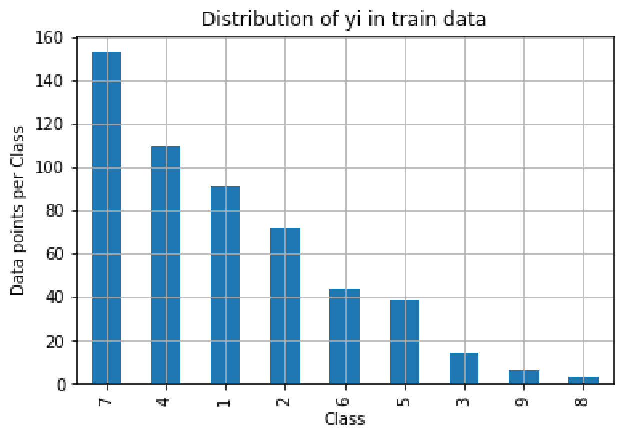
```
---------------------------------------- Train data ----------------------------------------
```



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



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



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

3.3 Univarite Analysis

3.3.1 Univarite Analysis for Gene feature

Q1. How many categories are there and How they are distributed?

In [37]:

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

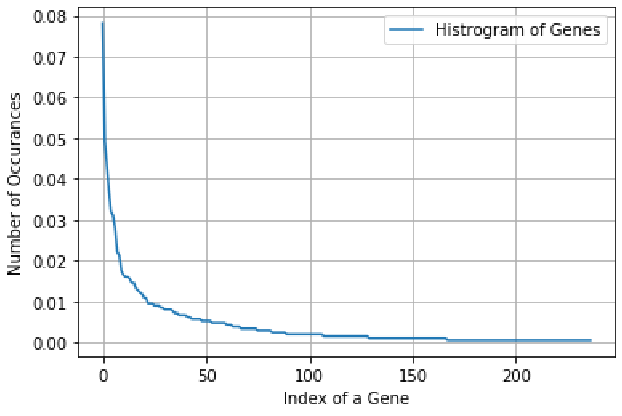
```
# the top 5 genes that occured most
print(unique_genes.head(5))
```

```
Number of Unique Genes : 237
BRCA1     166
TP53      105
EGFR       92
BRCA2      79
PTEN       68
Name: Gene, dtype: int64
```

3.3.1.1 Histogram of Gene

In [38]:

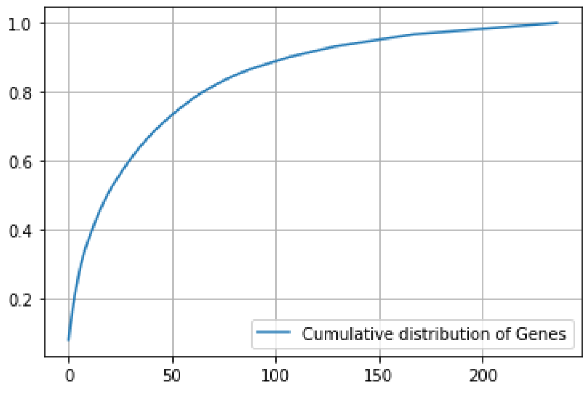
```
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histrogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



3.3.1.2 Cumulative distribution of Genes

In [40]:

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



Q2. 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-

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.

3.3.1.3 Response Encoding defs

In [41]:

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

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

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

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

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

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177,
0.13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788,
0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366,
0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408
163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177,
0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608,
0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608,
0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917,
0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081
761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295,
0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702,
0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334,
0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999,
0.066666666666666666, 0.066666666666666666],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

In [42]:

```
#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))
```

3.3.1.4 Onehot Encoding defs

In [45]:

```
# one-hot encoding of Gene feature.
```

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

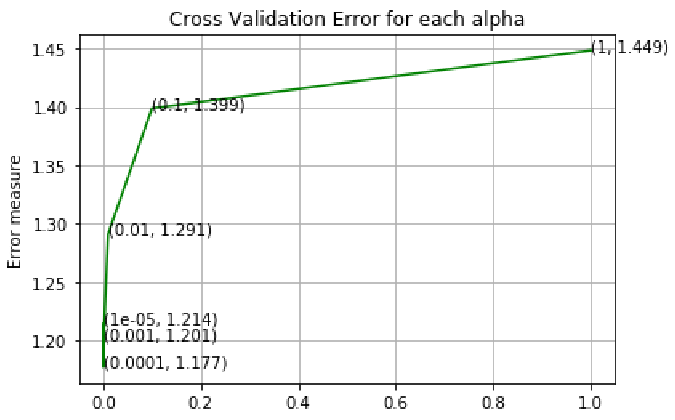
Q3. How good is this gene feature in predicting y_i?

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 [47]:

```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.
cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_gene_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.clas
ses_, eps=1e-15))

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.2136731510437604
For values of alpha =  0.0001 The log loss is: 1.1767674993293058
For values of alpha =  0.001 The log loss is: 1.201093659653355
For values of alpha =  0.01 The log loss is: 1.290896231705644
For values of alpha =  0.1 The log loss is: 1.3990541673849983
For values of alpha =  1 The log loss is: 1.4487170112024914
```



Alpha i's

```
For values of best alpha =  0.0001 The train log loss is: 1.0232964433397422
For values of best alpha =  0.0001 The cross validation log loss is: 1.1767674993293058
For values of best alpha =  0.0001 The test log loss is: 1.1742559195075766
```

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 [48]:

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

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  237  genes in train dataset?
Ans
1. In test data 648 out of 665 : 97.44360902255639
2. In cross validation data 516 out of  532 : 96.99248120300751
```

3.3.2 Univarite Analysis for Variation feature

Q1. How many categories are there?

In [50]:

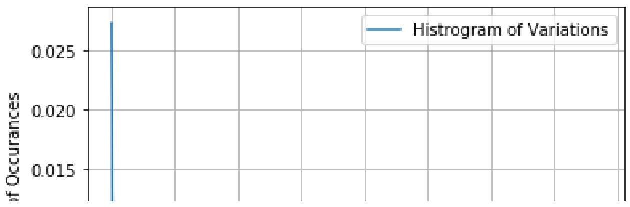
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(5))
```

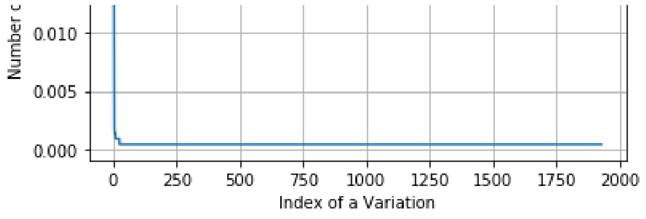
```
Number of Unique Variations : 1929
Truncating_Mutations    58
Deletion                51
Amplification           44
Fusions                 19
Overexpression           4
Name: Variation, dtype: int64
```

3.3.2.1 Histogram of Variation

In [52]:

```
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()
```



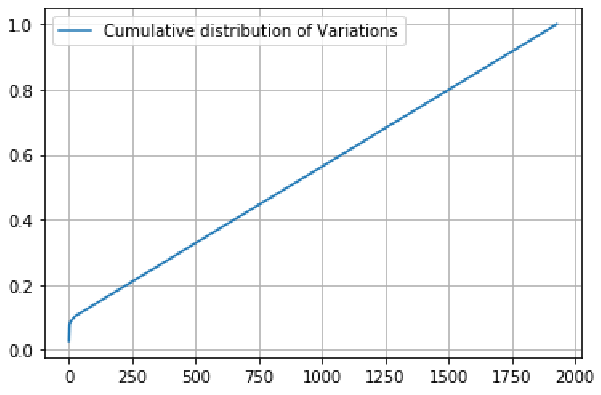


3.3.2.2 Cummulative distribution of Variation

In [53]:

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

```
[0.02730697 0.05131827 0.0720339  ... 0.99905838 0.99952919 1.        ]
```



Q2. 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

3.3.2.3 Response Encoding

In [55]:

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

3.3.2.4 Onehot Encoding

In [56]:

```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer()
```

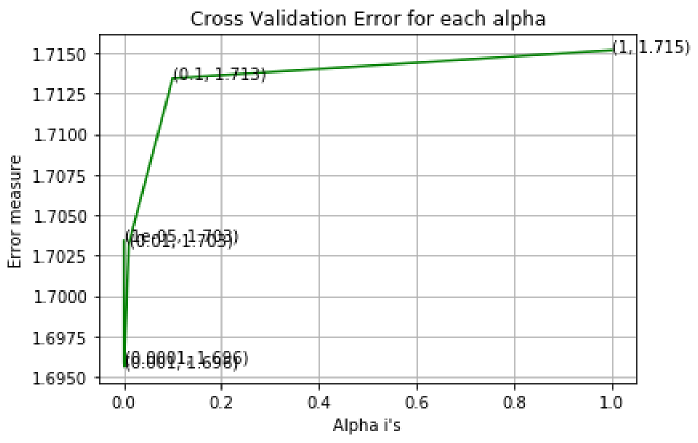
```
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

Q3. How good is this Variation feature in predicting y_i?

In [57]:

```
alpha = [10 ** x for x in range(-5, 1)]
cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_variation_feature_onehotCoding, y_train)

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

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.7034226076639472
For values of alpha =  0.0001 The log loss is: 1.6959080578106314
For values of alpha =  0.001 The log loss is: 1.6956168097905837
For values of alpha =  0.01 The log loss is: 1.7032020846520026
For values of alpha =  0.1 The log loss is: 1.7134385210692828
For values of alpha =  1 The log loss is: 1.7151763114990033
```



```
For values of best alpha =  0.001 The train log loss is: 1.1007298821753175
For values of best alpha =  0.001 The train log loss is: 1.1007298621753175
For values of best alpha =  0.001 The cross validation log loss is: 1.6956168097905837
For values of best alpha =  0.001 The test log loss is: 1.7084528599728996
```

3.2.3 Univariate Analysis on Text Feature

Q2. How to featurize this Text 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 TEXT Feature

3.3.3.3 Response Encoding

In [65]:

```python
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

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)

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

In [66]:

```python
#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)

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

3.3.3.4 Onehot Encoding

3.3.3.4 Onehot Encoding

In [68]:

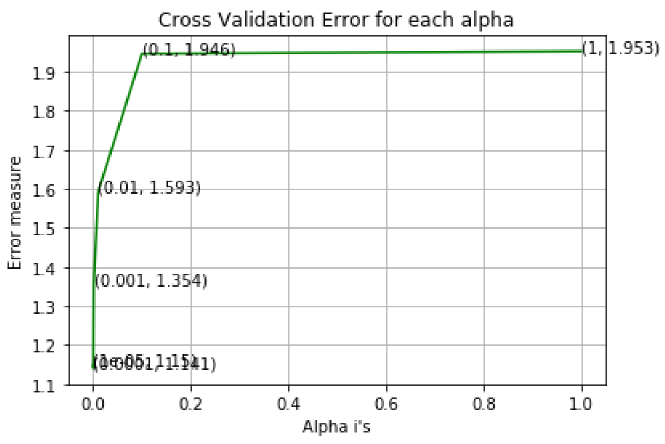
```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = TfidfVectorizer(max_features= 2000, min_df=3)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])

# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEXT'])
test_text_feature_onehotCoding = normalize(test_text_feature_onehotCoding, axis=0)
cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])
cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding, axis=0)



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

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


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

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

In [69]:

```
alpha = [10 ** x for x in range(-5, 1)]
cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_text_feature_onehotCoding, y_train)

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.1497786827230927
For values of alpha =  0.0001 The log loss is: 1.1405366336882368
For values of alpha =  0.001 The log loss is: 1.354069315565397
For values of alpha =  0.01 The log loss is: 1.5925219276317413
For values of alpha =  0.1 The log loss is: 1.9462737656175306
For values of alpha =  1 The log loss is: 1.9525541641964779
```



```
For values of best alpha =  0.0001 The train log loss is: 0.7162359584030088
For values of best alpha =  0.0001 The cross validation log loss is: 1.1405366336882368
For values of best alpha =  0.0001 The test log loss is: 1.0431126162249975
```

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

Ans. Yes, it seems like!

In [70]:

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

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

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

```
6.981 % of word of test data appeared in train data
7.424 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

4.1 Stacking

In [72]:

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

train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_onehotCoding)).tocs
r()
train_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))

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)


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

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 4195)
(number of data points * number of features) in test data =  (665, 4195)
(number of data points * number of features) in cross validation data = (532, 4195)
 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)
```

Model Definations

In [134]:

```
prettyTableoutput=[]

def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = TfidfVectorizer(max_features =2000,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_n
                print(i, "variation feature [{}] present in test data point [{}]".format(word,yes_n
o))
        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")

def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

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

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

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

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

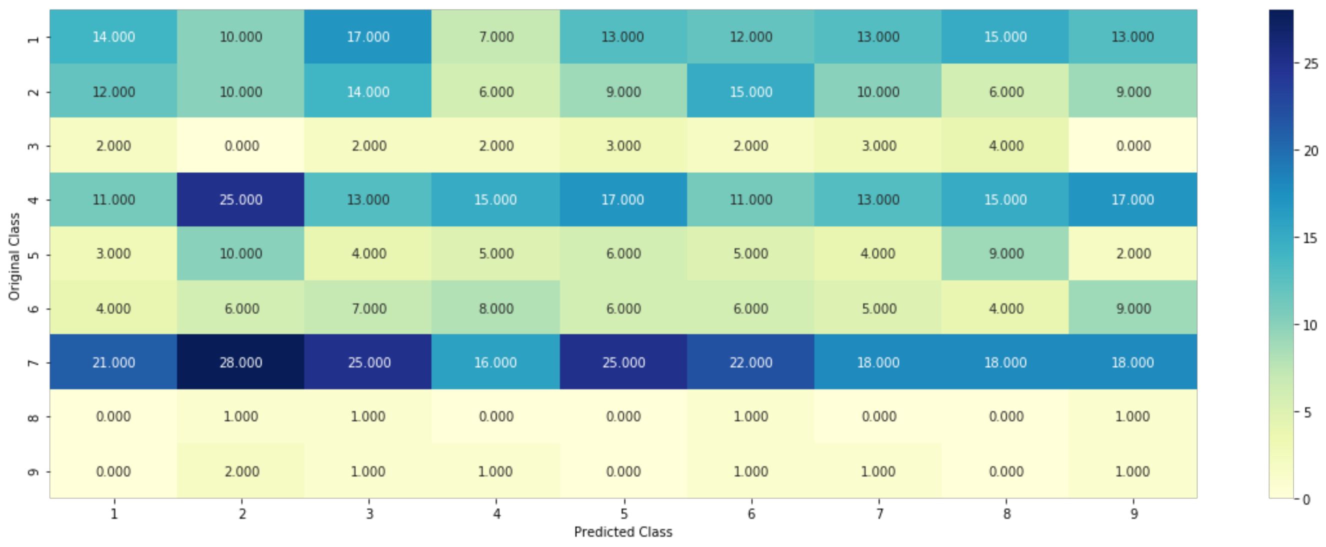
```
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

4.2 Prediction using a 'Random' Model

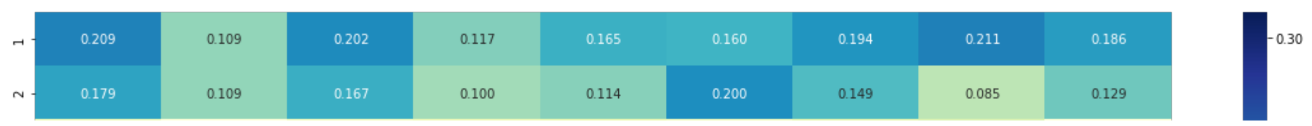
In [135]:

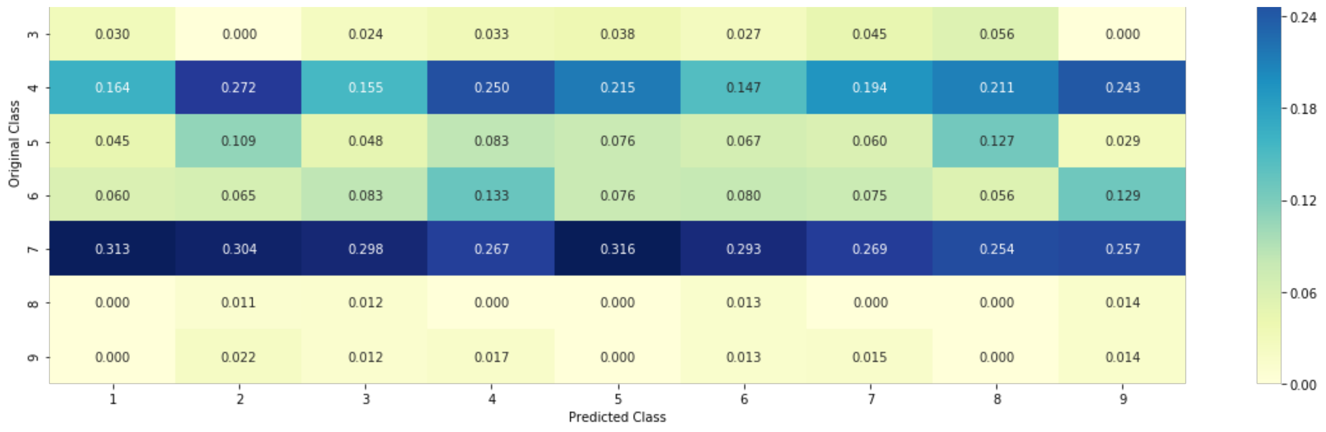
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# 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))

prettyTableoutput.append(["Random Model","NA",log_loss(y_cv,cv_predicted_y, eps=1e-15),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 2.490891796436401
Log loss on Test Data using Random Model 2.5163336395854863
-------------------- Confusion matrix --------------------
```

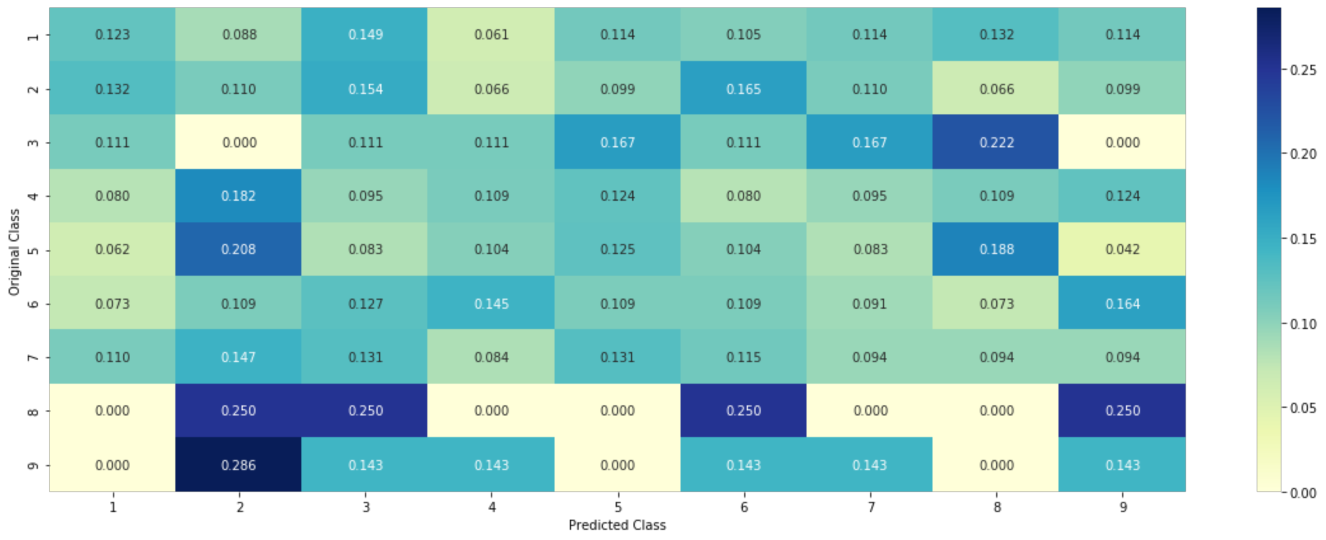


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



4.3. Base Line Model

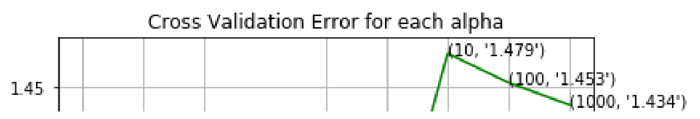
4.3.1. Naive Bayes

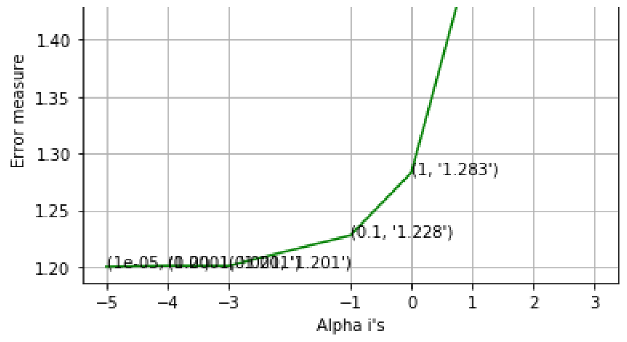
4.3.1.1. Hyper parameter tuning

In [136]:

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

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


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


alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying 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_ytr = 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_ytr, labels=clf.classes_, eps=1e-15))
predict_yte = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_yte, labels=clf.classes_, eps=1e-15))
predict_ycv = 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, p
redict_ycv, labels=clf.classes_, eps=1e-15))
prettyTableoutput.append(["Naive Bayes",log_loss(y_train, predict_ytr, labels=clf.classes_, eps=1e-
15),log_loss(y_cv, predict_yte, labels=clf.classes_, eps=1e-15),log_loss(y_test, predict_ycv, label
s=clf.classes_, eps=1e-15)])
```

```
for alpha = 1e-05
Log Loss : 1.1999985893061975
for alpha = 0.0001
Log Loss : 1.2010348359110108
for alpha = 0.001
Log Loss : 1.200863329689093
for alpha = 0.1
Log Loss : 1.2276165233952847
for alpha = 1
Log Loss : 1.2827268552871975
for alpha = 10
Log Loss : 1.4792137141524406
for alpha = 100
Log Loss : 1.4534364858103337
for alpha = 1000
Log Loss : 1.4340507122126531
```





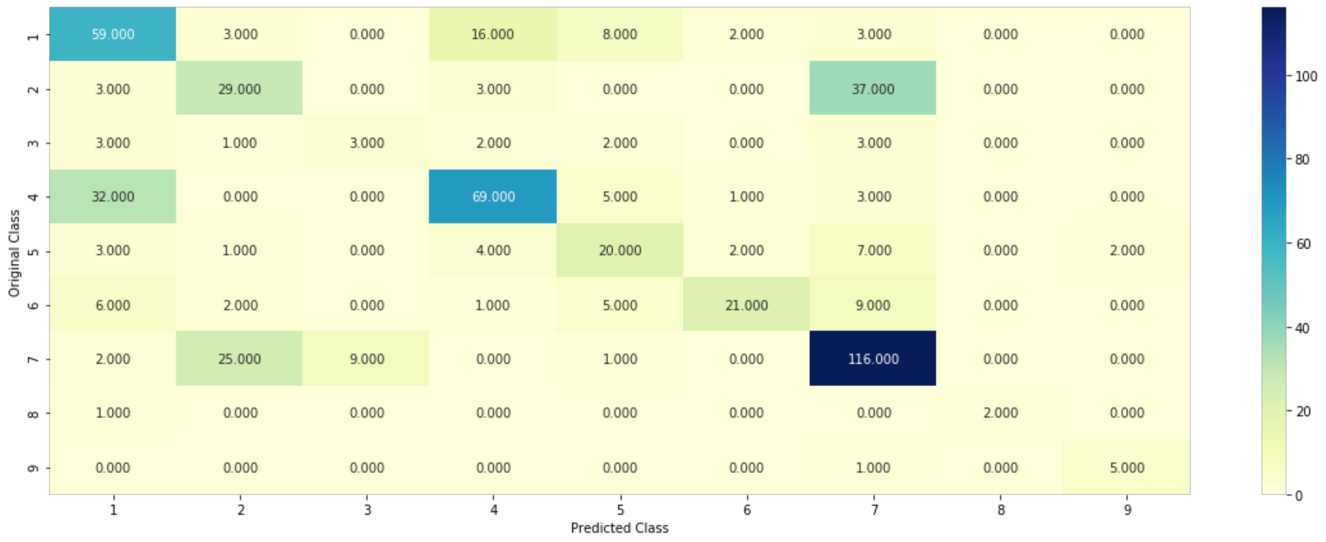
```
For values of best alpha =  1e-05 The train log loss is: 0.5689793768477163
For values of best alpha =  1e-05 The cross validation log loss is: 1.1999985893061975
For values of best alpha =  1e-05 The test log loss is: 1.1701596926393678
```

4.3.1.2. Testing the model with best hyper paramters

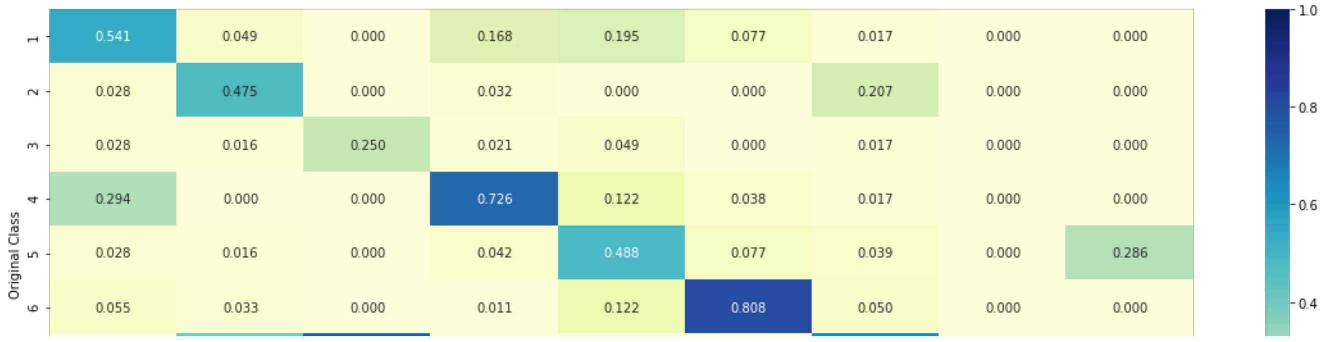
In [83]:

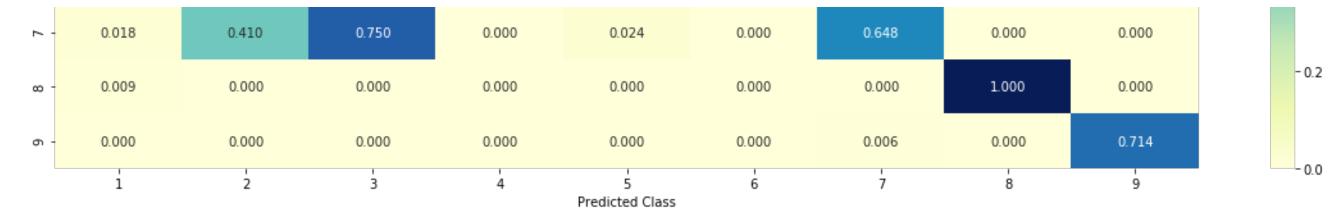
```
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(test_x_onehotCoding)
# to avoid rounding error while multiplying probabilites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

```
Log Loss : 1.1999985893061975
Number of missclassified point : 0.39097744360902253
-------------------- Confusion matrix --------------------
```

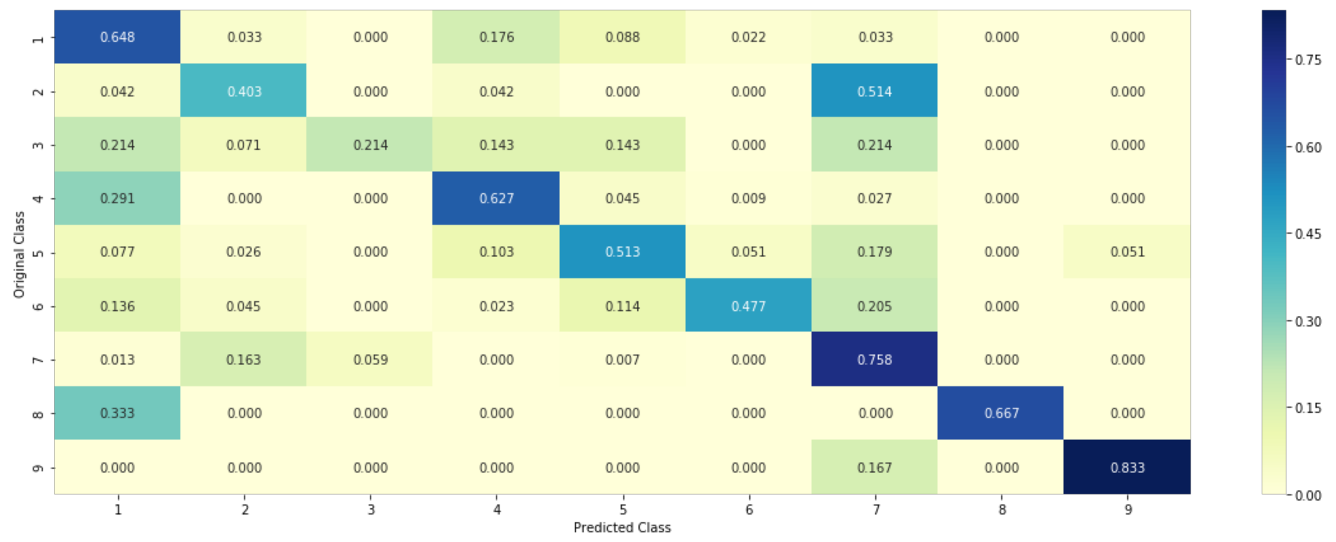


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



4.3.1.3. Feature Importance, Correctly classified point

In [85]:

```
test_point_index = 6
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices=np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0688 0.0569 0.0094 0.0724 0.0361 0.0369 0.7122 0.0048 0.0026]]
Actual Class : 7
--------------------------------------------------
18 Text feature [activation] present in test data point [True]
19 Text feature [activated] present in test data point [True]
20 Text feature [cells] present in test data point [True]
21 Text feature [kinase] present in test data point [True]
24 Text feature [contrast] present in test data point [True]
25 Text feature [expressing] present in test data point [True]
26 Text feature [downstream] present in test data point [True]
27 Text feature [factor] present in test data point [True]
28 Text feature [inhibitor] present in test data point [True]
29 Text feature [cell] present in test data point [True]
30 Text feature [also] present in test data point [True]
31 Text feature [shown] present in test data point [True]
32 Text feature [growth] present in test data point [True]
33 Text feature [activating] present in test data point [True]
34 Text feature [signaling] present in test data point [True]
35 Text feature [however] present in test data point [True]
36 Text feature [independent] present in test data point [True]
37 Text feature [presence] present in test data point [True]
38 Text feature [addition] present in test data point [True]
39 Text feature [10] present in test data point [True]
40 Text feature [compared] present in test data point [True]
```

```
40 Text feature [compared] present in test data point [True]
41 Text feature [phosphorylation] present in test data point [True]
42 Text feature [found] present in test data point [True]
43 Text feature [mutations] present in test data point [True]
44 Text feature [similar] present in test data point [True]
45 Text feature [treated] present in test data point [True]
46 Text feature [constitutive] present in test data point [True]
47 Text feature [previously] present in test data point [True]
48 Text feature [well] present in test data point [True]
49 Text feature [treatment] present in test data point [True]
50 Text feature [showed] present in test data point [True]
51 Text feature [suggest] present in test data point [True]
52 Text feature [sensitive] present in test data point [True]
53 Text feature [may] present in test data point [True]
54 Text feature [mutant] present in test data point [True]
55 Text feature [constitutively] present in test data point [True]
56 Text feature [demonstrated] present in test data point [True]
57 Text feature [higher] present in test data point [True]
58 Text feature [recently] present in test data point [True]
59 Text feature [mechanism] present in test data point [True]
60 Text feature [3b] present in test data point [True]
61 Text feature [different] present in test data point [True]
62 Text feature [increased] present in test data point [True]
63 Text feature [although] present in test data point [True]
64 Text feature [1a] present in test data point [True]
65 Text feature [figure] present in test data point [True]
66 Text feature [inhibition] present in test data point [True]
67 Text feature [mutation] present in test data point [True]
68 Text feature [without] present in test data point [True]
69 Text feature [tyrosine] present in test data point [True]
72 Text feature [described] present in test data point [True]
73 Text feature [increase] present in test data point [True]
74 Text feature [inhibitors] present in test data point [True]
75 Text feature [total] present in test data point [True]
76 Text feature [enhanced] present in test data point [True]
78 Text feature [proliferation] present in test data point [True]
79 Text feature [approximately] present in test data point [True]
80 Text feature [potential] present in test data point [True]
81 Text feature [interestingly] present in test data point [True]
82 Text feature [either] present in test data point [True]
83 Text feature [respectively] present in test data point [True]
84 Text feature [oncogenic] present in test data point [True]
85 Text feature [reported] present in test data point [True]
86 Text feature [examined] present in test data point [True]
87 Text feature [absence] present in test data point [True]
88 Text feature [observed] present in test data point [True]
89 Text feature [12] present in test data point [True]
90 Text feature [serum] present in test data point [True]
91 Text feature [consistent] present in test data point [True]
92 Text feature [3a] present in test data point [True]
93 Text feature [using] present in test data point [True]
94 Text feature [two] present in test data point [True]
95 Text feature [13] present in test data point [True]
96 Text feature [fig] present in test data point [True]
98 Text feature [results] present in test data point [True]
99 Text feature [including] present in test data point [True]
Out of the top  100  features  76 are present in query point
```

4.3.1.4. Feature Importance, Incorrectly classified point

In [86]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1534 0.0699 0.0124 0.2968 0.2879 0.051  0.1188 0.0064 0.0034]]
Actual Class : 1
--------------------------------------------------
12 Text feature [protein] present in test data point [True]
15 Text feature [activity] present in test data point [True]
16 Text feature [proteins] present in test data point [True]
17 Text feature [function] present in test data point [True]
18 Text feature [results] present in test data point [True]
22 Text feature [experiments] present in test data point [True]
23 Text feature [whether] present in test data point [True]
24 Text feature [whereas] present in test data point [True]
25 Text feature [type] present in test data point [True]
26 Text feature [two] present in test data point [True]
27 Text feature [also] present in test data point [True]
28 Text feature [shown] present in test data point [True]
29 Text feature [acid] present in test data point [True]
30 Text feature [described] present in test data point [True]
31 Text feature [determined] present in test data point [True]
32 Text feature [ability] present in test data point [True]
33 Text feature [important] present in test data point [True]
35 Text feature [amino] present in test data point [True]
36 Text feature [wild] present in test data point [True]
37 Text feature [mutations] present in test data point [True]
38 Text feature [retained] present in test data point [True]
40 Text feature [loss] present in test data point [True]
41 Text feature [functional] present in test data point [True]
42 Text feature [bind] present in test data point [True]
43 Text feature [indicate] present in test data point [True]
44 Text feature [may] present in test data point [True]
46 Text feature [effects] present in test data point [True]
47 Text feature [either] present in test data point [True]
48 Text feature [indicated] present in test data point [True]
49 Text feature [related] present in test data point [True]
50 Text feature [previously] present in test data point [True]
51 Text feature [thus] present in test data point [True]
52 Text feature [determine] present in test data point [True]
53 Text feature [although] present in test data point [True]
55 Text feature [reduced] present in test data point [True]
56 Text feature [levels] present in test data point [True]
57 Text feature [30] present in test data point [True]
58 Text feature [three] present in test data point [True]
61 Text feature [therefore] present in test data point [True]
62 Text feature [see] present in test data point [True]
64 Text feature [containing] present in test data point [True]
65 Text feature [terminal] present in test data point [True]
66 Text feature [tagged] present in test data point [True]
67 Text feature [could] present in test data point [True]
68 Text feature [suggest] present in test data point [True]
69 Text feature [lower] present in test data point [True]
70 Text feature [suggesting] present in test data point [True]
71 Text feature [vitro] present in test data point [True]
72 Text feature [similar] present in test data point [True]
73 Text feature [however] present in test data point [True]
74 Text feature [mammalian] present in test data point [True]
75 Text feature [analyzed] present in test data point [True]
76 Text feature [mutants] present in test data point [True]
77 Text feature [discussion] present in test data point [True]
78 Text feature [show] present in test data point [True]
79 Text feature [analysis] present in test data point [True]
80 Text feature [several] present in test data point [True]
81 Text feature [mutant] present in test data point [True]
82 Text feature [expressed] present in test data point [True]
83 Text feature [possible] present in test data point [True]
84 Text feature [result] present in test data point [True]
85 Text feature [critical] present in test data point [True]
86 Text feature [found] present in test data point [True]
87 Text feature [lack] present in test data point [True]
89 Text feature [using] present in test data point [True]
90 Text feature [one] present in test data point [True]
91 Text feature [transfected] present in test data point [True]
92 Text feature [addition] present in test data point [True]
94 Text feature [cells] present in test data point [True]
95 Text feature [purified] present in test data point [True]
96 Text feature [contribute] present in test data point [True]
97 Text feature [cellular] present in test data point [True]
98 Text feature [role] present in test data point [True]
```

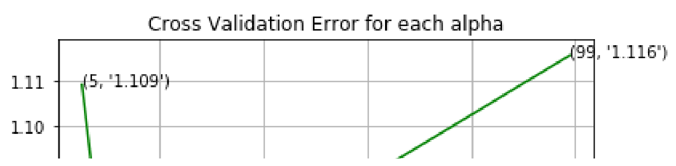
```
98 Text feature [1012] present in test data point [True]
99 Text feature [10] present in test data point [True]
Out of the top  100  features  74 are present in query point
```

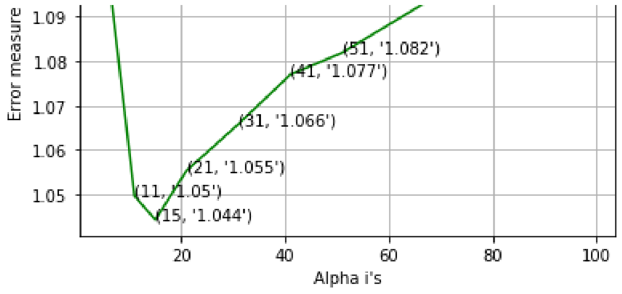
4.4. K Nearest Neighbour Classification

In [137]:

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

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


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

predict_ytr = 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_ytr, labels=clf.classes_, eps=1e-15))
predict_yte = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_yte, labels=clf.classes_, eps=1e-15))
predict_ycv = 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, p
redict_ycv, labels=clf.classes_, eps=1e-15))
prettyTableoutput.append(["K Nearest Neighbour",log_loss(y_train, predict_ytr, labels=clf.classes_,
eps=1e-15),log_loss(y_cv, predict_yte, labels=clf.classes_, eps=1e-15),log_loss(y_test, predict_ycv
, labels=clf.classes_, eps=1e-15)])
```

```
for alpha = 5
Log Loss : 1.1091355021753224
for alpha = 11
Log Loss : 1.0495939928544578
for alpha = 15
Log Loss : 1.0442220202853572
for alpha = 21
Log Loss : 1.0552872854337323
for alpha = 31
Log Loss : 1.0657783579508717
for alpha = 41
Log Loss : 1.0769410590057897
for alpha = 51
Log Loss : 1.0818838948319731
for alpha = 99
Log Loss : 1.115747707177607
```





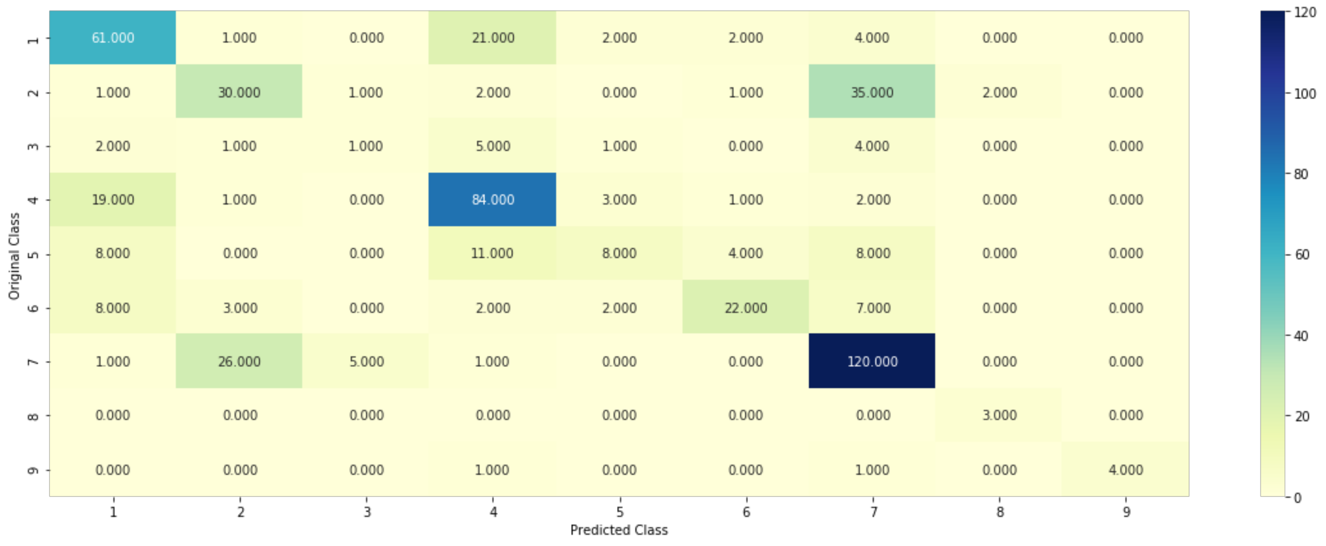
```
For values of best alpha =  15 The train log loss is: 0.7269666051830004
For values of best alpha =  15 The cross validation log loss is: 1.0442220202853572
For values of best alpha =  15 The test log loss is: 1.0230340027479514
```

4.4.2. Testing the model with best hyper paramters

In [91]:

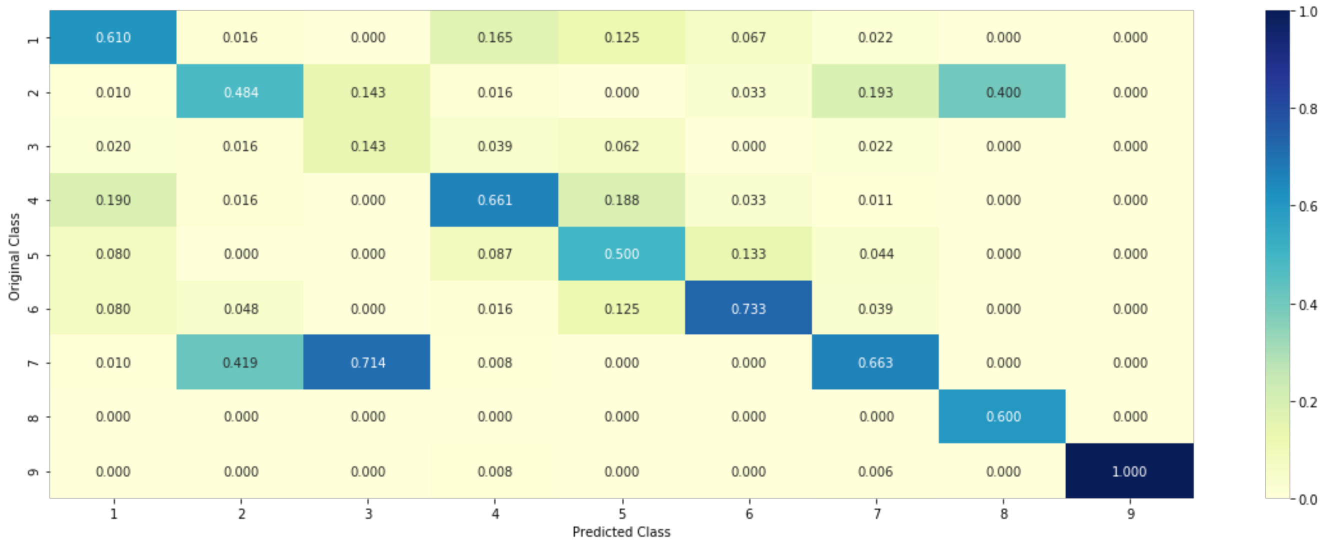
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y, clf)
```

```
Log loss : 1.0442220202853572
Number of mis-classified points : 0.37406015037593987
-------------------- Confusion matrix --------------------
```



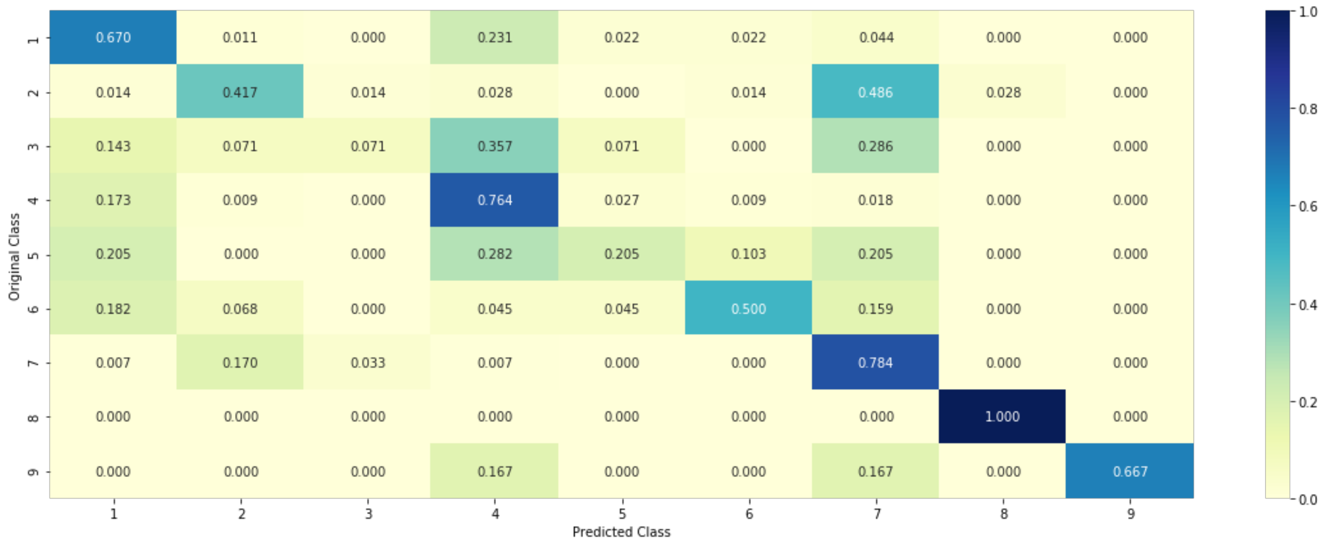
Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	61.000	1.000	0.000	21.000	2.000	2.000	4.000	0.000	0.000
2	1.000	30.000	1.000	2.000	0.000	1.000	35.000	2.000	0.000
3	2.000	1.000	1.000	5.000	1.000	0.000	4.000	0.000	0.000
4	19.000	1.000	0.000	84.000	3.000	1.000	2.000	0.000	0.000
5	8.000	0.000	0.000	11.000	8.000	4.000	8.000	0.000	0.000
6	8.000	3.000	0.000	2.000	2.000	22.000	7.000	0.000	0.000
7	1.000	26.000	5.000	1.000	0.000	0.000	120.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	3.000	0.000
9	0.000	0.000	0.000	1.000	0.000	0.000	1.000	0.000	4.000

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



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.610	0.016	0.000	0.165	0.125	0.067	0.022	0.000	0.000
2	0.010	0.484	0.143	0.016	0.000	0.033	0.193	0.400	0.000
3	0.020	0.016	0.143	0.039	0.062	0.000	0.022	0.000	0.000
4	0.190	0.016	0.000	0.661	0.188	0.033	0.011	0.000	0.000
5	0.080	0.000	0.000	0.087	0.500	0.133	0.044	0.000	0.000
6	0.080	0.048	0.000	0.016	0.125	0.733	0.039	0.000	0.000
7	0.010	0.419	0.714	0.008	0.000	0.000	0.663	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.600	0.000
9	0.000	0.000	0.000	0.008	0.000	0.000	0.006	0.000	1.000

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



4.4.3.Sample Query point -1

In [93]:

```
test_point_index = 4
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), alpha[best_alpha
])
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to classes",train_y
[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 1
Actual Class : 1
The  15  nearest neighbours of the test points belongs to classes [1 1 1 6 5 5 6 6 6 6 6 6 1 6]
Fequency of nearest points : Counter({6: 9, 1: 4, 5: 2})
```

4.4.4. Sample Query Point-2

In [96]:

```
test_point_index = 3

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 be
longs to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 2
Actual Class : 7
the k value for knn is 15 and the nearest neighbours of the test points belongs to classes [7 7 7
7 3 7 7 7 7 7 2 2 2 2 2]
Fequency of nearest points : Counter({7: 9, 2: 5, 3: 1})
```

4.5. Logistic Regression

4.5.1 CountVectorizing with unigram and bigram

In [97]:

```
text_vectorizer = CountVectorizer(ngram_range=(1,2),max_features =2000,min_df=3)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEXT'])
test_text_feature_onehotCoding = normalize(test_text_feature_onehotCoding, axis=0)

cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])
cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding, axis=0)
```

4.5.2 Stacking

In [98]:

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

test_x_ohe = hstack((test_gene_var_onehotCoding, test_text_feature_onehotCoding)).tocsr()

cv_x_ohe = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).tocsr()
```

4.5.3. With Class balancing

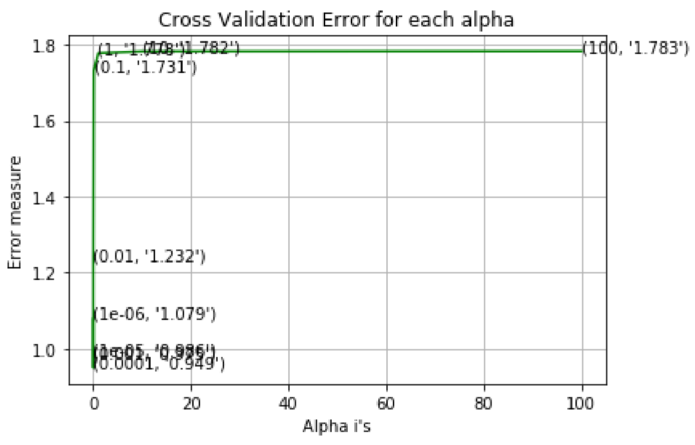
4.5.3.1. Hyper paramter tuning

In [138]:

```
alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42
)
    clf.fit(train_x_ohe, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_ohe, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_ohe)
    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', ran
dom_state=42)
clf.fit(train_x_ohe, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_ohe, train_y)

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

```
predict_yte = sig_clf.predict_proba(cv_x_ohe)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_yte, labels=clf.classes_, eps=1e-15))
predict_ycv = sig_clf.predict_proba(test_x_ohe)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_ycv, labels=clf.classes_, eps=1e-15))
prettyTableoutput.append(["LR with Classbalance",log_loss(y_train, predict_ytr, labels=clf.classes_
, eps=1e-15),log_loss(y_cv, predict_yte, labels=clf.classes_, eps=1e-15),log_loss(y_test, predict_y
cv, labels=clf.classes_, eps=1e-15)])
```

```
for alpha = 1e-06
Log Loss : 1.078918619702524
for alpha = 1e-05
Log Loss : 0.9857884997392994
for alpha = 0.0001
Log Loss : 0.9490146449070358
for alpha = 0.001
Log Loss : 0.975192104313247
for alpha = 0.01
Log Loss : 1.2322507538891345
for alpha = 0.1
Log Loss : 1.7314287289263595
for alpha = 1
Log Loss : 1.7780173952124878
for alpha = 10
Log Loss : 1.7823119972299468
for alpha = 100
Log Loss : 1.7827730745993058
```



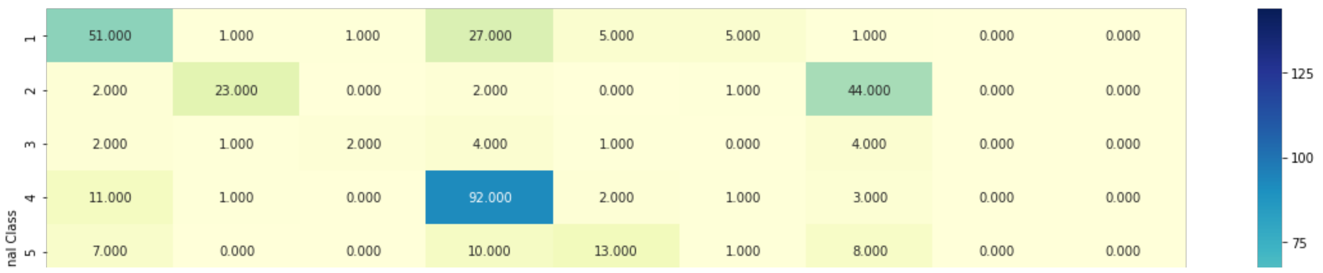
```
For values of best alpha =  0.0001 The train log loss is: 0.4516198785675059
For values of best alpha =  0.0001 The cross validation log loss is: 0.9490146449070358
For values of best alpha =  0.0001 The test log loss is: 0.9384377667266186
```

4.5.3.2. Testing the model with best hyper paramters

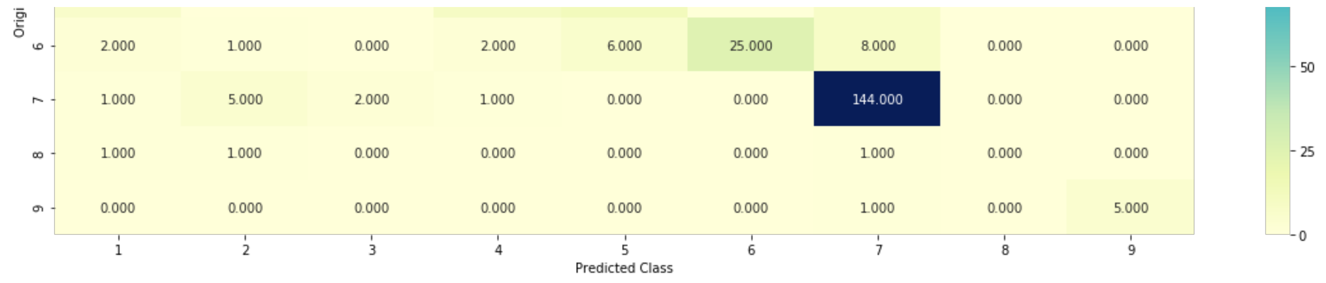
In [100]:

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
predict_and_plot_confusion_matrix(train_x_ohe, train_y, cv_x_ohe, cv_y, clf)
```

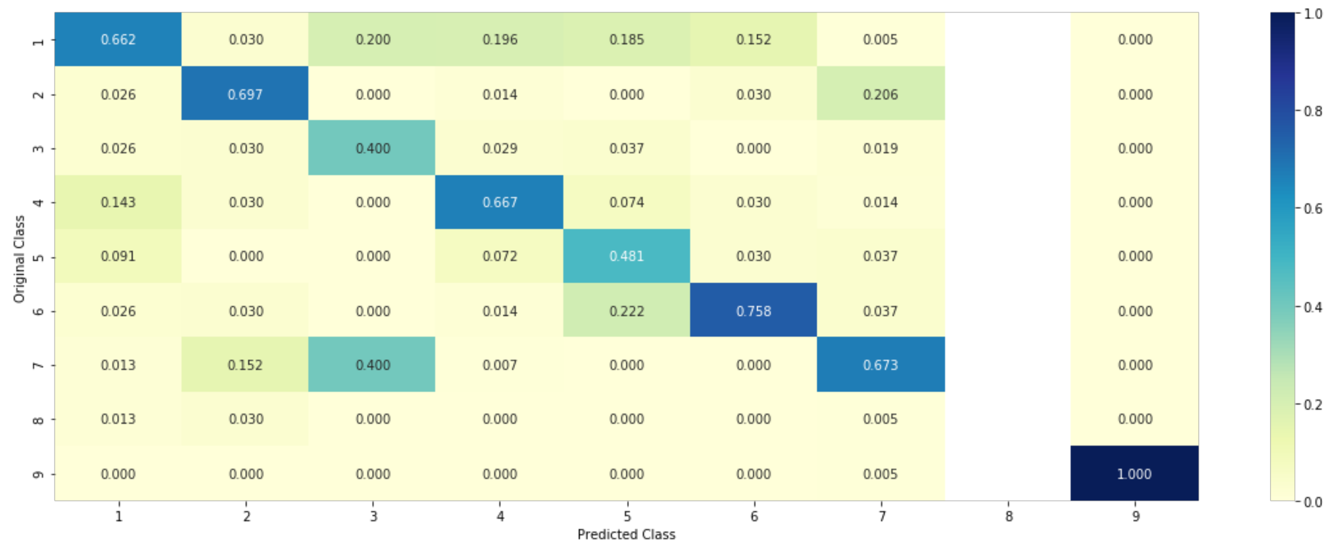
```
Log loss : 0.9490146449070358
Number of mis-classified points : 0.33270676691729323
-------------------- Confusion matrix --------------------
```



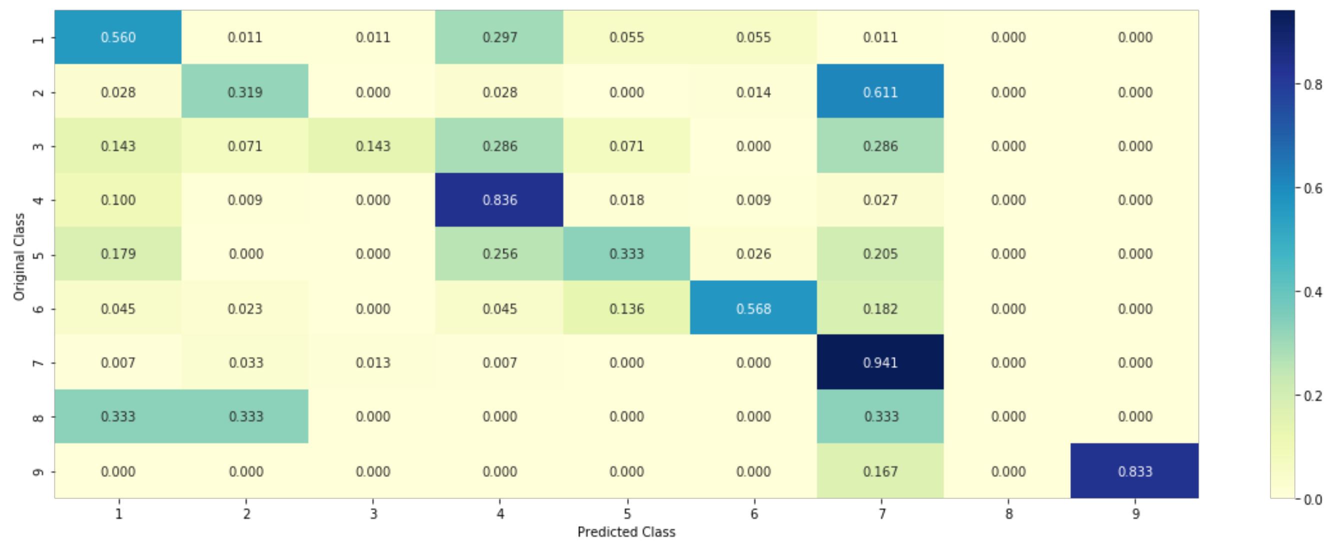
1	51.000	1.000	1.000	27.000	5.000	5.000	1.000	0.000	0.000
2	2.000	23.000	0.000	2.000	0.000	1.000	44.000	0.000	0.000
3	2.000	1.000	2.000	4.000	1.000	0.000	4.000	0.000	0.000
4	11.000	1.000	0.000	92.000	2.000	1.000	3.000	0.000	0.000
5	7.000	0.000	0.000	10.000	13.000	1.000	8.000	0.000	0.000



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



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



4.5.3.3.1. Correctly Classified point

In [101]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
clf.fit(train_x_ohe,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_ohe[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_ohe[test_point_index
]),4))
print("Actual Class :", test_y[test_point_index])
```

```
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 5
Predicted Class Probabilities: [[1.918e-01 1.690e-02 6.000e-04 1.608e-01 5.087e-01 2.300e-03 1.134
e-01
  5.400e-03 2.000e-04]]
Actual Class : 1
--------------------------------------------------
5 Text feature [one] present in test data point [True]
6 Text feature [finally] present in test data point [True]
37 Text feature [63] present in test data point [True]
41 Text feature [phosphate] present in test data point [True]
42 Text feature [49] present in test data point [True]
57 Text feature [incubation] present in test data point [True]
93 Text feature [flow] present in test data point [True]
94 Text feature [ph] present in test data point [True]
116 Text feature [together] present in test data point [True]
128 Text feature [lane] present in test data point [True]
132 Text feature [forms] present in test data point [True]
134 Text feature [group] present in test data point [True]
135 Text feature [separate] present in test data point [True]
152 Text feature [suggests] present in test data point [True]
161 Text feature [frameshift] present in test data point [True]
164 Text feature [32] present in test data point [True]
179 Text feature [bind] present in test data point [True]
188 Text feature [findings] present in test data point [True]
194 Text feature [19] present in test data point [True]
195 Text feature [human] present in test data point [True]
207 Text feature [constructs] present in test data point [True]
212 Text feature [santa] present in test data point [True]
213 Text feature [increased] present in test data point [True]
219 Text feature [basal] present in test data point [True]
228 Text feature [response] present in test data point [True]
252 Text feature [notably] present in test data point [True]
260 Text feature [determine] present in test data point [True]
266 Text feature [essential] present in test data point [True]
276 Text feature [69] present in test data point [True]
280 Text feature [samples] present in test data point [True]
295 Text feature [bars] present in test data point [True]
310 Text feature [years] present in test data point [True]
341 Text feature [min] present in test data point [True]
342 Text feature [10] present in test data point [True]
343 Text feature [along] present in test data point [True]
353 Text feature [components] present in test data point [True]
362 Text feature [45] present in test data point [True]
373 Text feature [found] present in test data point [True]
376 Text feature [transient] present in test data point [True]
381 Text feature [express] present in test data point [True]
392 Text feature [labeled] present in test data point [True]
395 Text feature [light] present in test data point [True]
402 Text feature [page] present in test data point [True]
409 Text feature [similarly] present in test data point [True]
422 Text feature [translation] present in test data point [True]
423 Text feature [yet] present in test data point [True]
429 Text feature [target] present in test data point [True]
435 Text feature [46] present in test data point [True]
439 Text feature [internal] present in test data point [True]
440 Text feature [combination] present in test data point [True]
443 Text feature [earlier] present in test data point [True]
451 Text feature [mainly] present in test data point [True]
454 Text feature [see] present in test data point [True]
463 Text feature [relative] present in test data point [True]
469 Text feature [act] present in test data point [True]
499 Text feature [identical] present in test data point [True]
Out of the top  500  features  56 are present in query point
```

4.5.3.3.2. inCorrectly Classified point

In [102]:

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

```
Predicted Class : 1
Predicted Class Probabilities: [[4.608e-01 1.140e-02 4.400e-03 3.160e-01 1.464e-01 4.130e-02 1.170
e-02
  7.900e-03 1.000e-04]]
Actual Class : 1
--------------------------------------------------
4 Text feature [show] present in test data point [True]
7 Text feature [missense] present in test data point [True]
34 Text feature [represented] present in test data point [True]
37 Text feature [tumor] present in test data point [True]
38 Text feature [confirm] present in test data point [True]
51 Text feature [online] present in test data point [True]
54 Text feature [finally] present in test data point [True]
61 Text feature [non] present in test data point [True]
72 Text feature [42] present in test data point [True]
73 Text feature [construct] present in test data point [True]
87 Text feature [eight] present in test data point [True]
91 Text feature [agreement] present in test data point [True]
92 Text feature [affecting] present in test data point [True]
94 Text feature [right] present in test data point [True]
97 Text feature [modified] present in test data point [True]
100 Text feature [38] present in test data point [True]
101 Text feature [specific] present in test data point [True]
102 Text feature [set] present in test data point [True]
109 Text feature [information] present in test data point [True]
110 Text feature [medium] present in test data point [True]
116 Text feature [role] present in test data point [True]
124 Text feature [1994] present in test data point [True]
125 Text feature [anti] present in test data point [True]
131 Text feature [indeed] present in test data point [True]
132 Text feature [respectively] present in test data point [True]
137 Text feature [ml] present in test data point [True]
142 Text feature [showed] present in test data point [True]
155 Text feature [interestingly] present in test data point [True]
165 Text feature [induce] present in test data point [True]
195 Text feature [2000] present in test data point [True]
206 Text feature [96] present in test data point [True]
207 Text feature [database] present in test data point [True]
210 Text feature [evidence] present in test data point [True]
214 Text feature [homologous] present in test data point [True]
227 Text feature [profile] present in test data point [True]
232 Text feature [present] present in test data point [True]
234 Text feature [unknown] present in test data point [True]
235 Text feature [efficiency] present in test data point [True]
237 Text feature [hence] present in test data point [True]
241 Text feature [provide] present in test data point [True]
245 Text feature [contributes] present in test data point [True]
264 Text feature [foci] present in test data point [True]
266 Text feature [therefore] present in test data point [True]
271 Text feature [interest] present in test data point [True]
284 Text feature [recombination] present in test data point [True]
289 Text feature [degree] present in test data point [True]
290 Text feature [support] present in test data point [True]
292 Text feature [analysis] present in test data point [True]
296 Text feature [figure] present in test data point [True]
306 Text feature [encoded] present in test data point [True]
310 Text feature [determine] present in test data point [True]
329 Text feature [due] present in test data point [True]
348 Text feature [2008] present in test data point [True]
357 Text feature [already] present in test data point [True]
371 Text feature [primarily] present in test data point [True]
372 Text feature [59] present in test data point [True]
375 Text feature [thereby] present in test data point [True]
376 Text feature [implications] present in test data point [True]
```

```
378 Text feature [damage] present in test data point [True]
407 Text feature [express] present in test data point [True]
412 Text feature [carriers] present in test data point [True]
415 Text feature [2011] present in test data point [True]
429 Text feature [phenotype] present in test data point [True]
435 Text feature [allow] present in test data point [True]
442 Text feature [47] present in test data point [True]
445 Text feature [contrast] present in test data point [True]
450 Text feature [loading] present in test data point [True]
461 Text feature [multiple] present in test data point [True]
463 Text feature [outside] present in test data point [True]
469 Text feature [culture] present in test data point [True]
472 Text feature [significant] present in test data point [True]
474 Text feature [process] present in test data point [True]
485 Text feature [final] present in test data point [True]
494 Text feature [truncating] present in test data point [True]
Out of the top  500  features  74 are present in query point
```

4.5.4. Without Class balancing

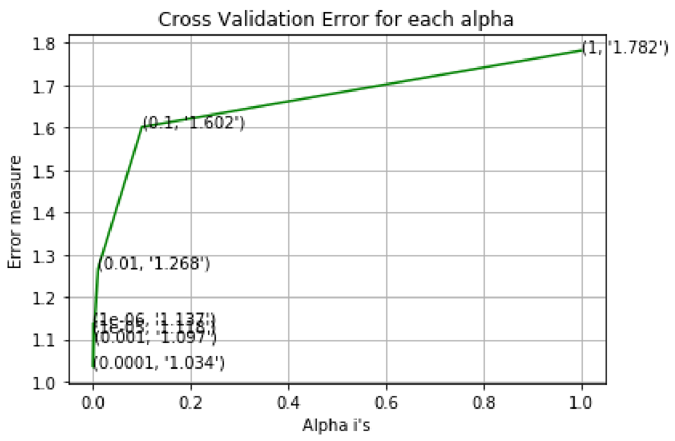
4.5.4.1. Hyper paramter tuning

In [139]:

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

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


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

predict_ytr = 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_ytr, labels=clf.classes_, eps=1e-15))
predict_yte = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_yte, labels=clf.classes_, eps=1e-15))
predict_ycv = 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, p
redict_ycv, labels=clf.classes_, eps=1e-15))

prettyTableoutput.append(["LR without Classbalance",log_loss(y_train, predict_ytr, labels=clf.class
es_, eps=1e-15),log_loss(y_cv, predict_yte, labels=clf.classes_, eps=1e-15),log_loss(y_test, predic
t_ycv, labels=clf.classes_, eps=1e-15)])
```

```
for alpha = 1e-06
Log Loss : 1.13679175843924
for alpha = 1e-05
Log Loss : 1.1179526255871126
for alpha = 0.0001
```

```
Log Loss : 1.0342580261370453
for alpha = 0.001
Log Loss : 1.0970448369312256
for alpha = 0.01
Log Loss : 1.2681365455437765
for alpha = 0.1
Log Loss : 1.6017294136523113
for alpha = 1
Log Loss : 1.7819810124123756
```



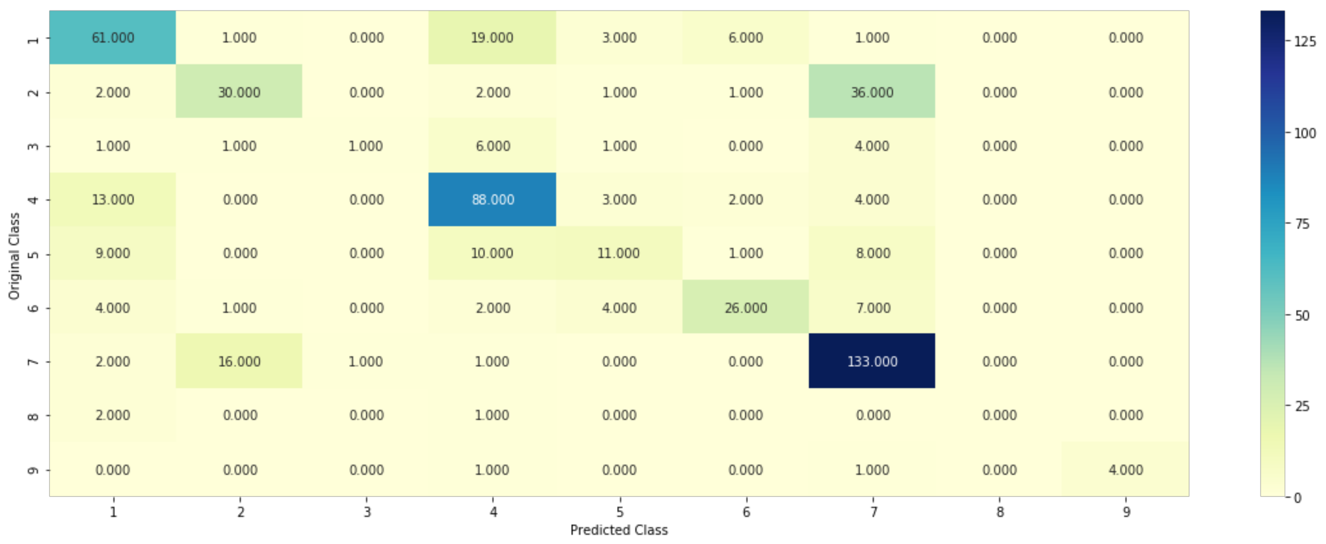
```
For values of best alpha =  0.0001 The train log loss is: 0.43413347783552025
For values of best alpha =  0.0001 The cross validation log loss is: 1.0342580261370453
For values of best alpha =  0.0001 The test log loss is: 0.960636842038146
```

4.5.4.2. Testing model with best hyper parameters

In [104]:

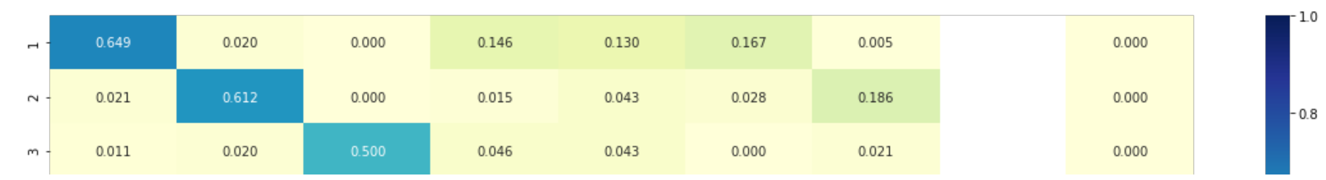
```python
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

```
Log loss : 1.0342580261370453
Number of mis-classified points : 0.33458646616541354
-------------------- Confusion matrix --------------------
```

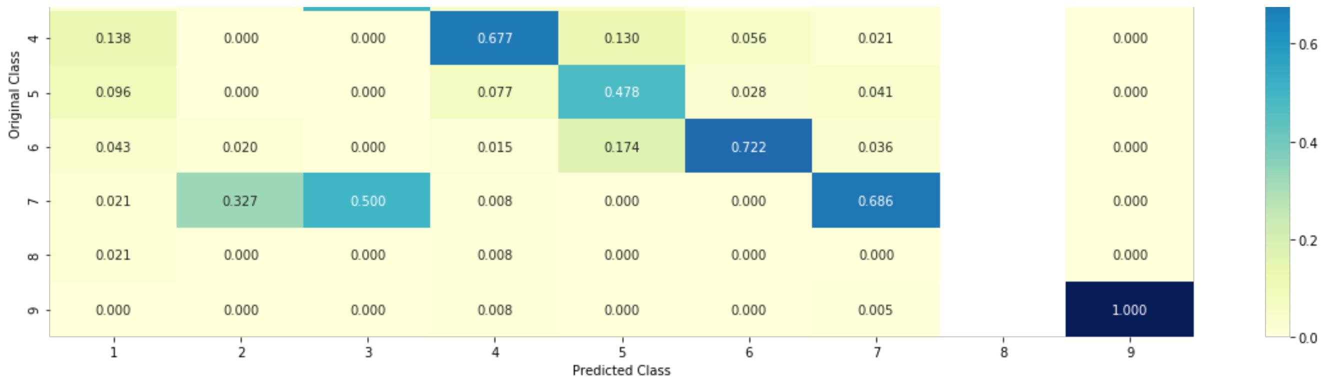


	1	2	3	4	5	6	7	8	9
1	61.000	1.000	0.000	19.000	3.000	6.000	1.000	0.000	0.000
2	2.000	30.000	0.000	2.000	1.000	1.000	36.000	0.000	0.000
3	1.000	1.000	1.000	6.000	1.000	0.000	4.000	0.000	0.000
4	13.000	0.000	0.000	88.000	3.000	2.000	4.000	0.000	0.000
5	9.000	0.000	0.000	10.000	11.000	1.000	8.000	0.000	0.000
6	4.000	1.000	0.000	2.000	4.000	26.000	7.000	0.000	0.000
7	2.000	16.000	1.000	1.000	0.000	0.000	133.000	0.000	0.000
8	2.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000
9	0.000	0.000	0.000	1.000	0.000	0.000	1.000	0.000	4.000

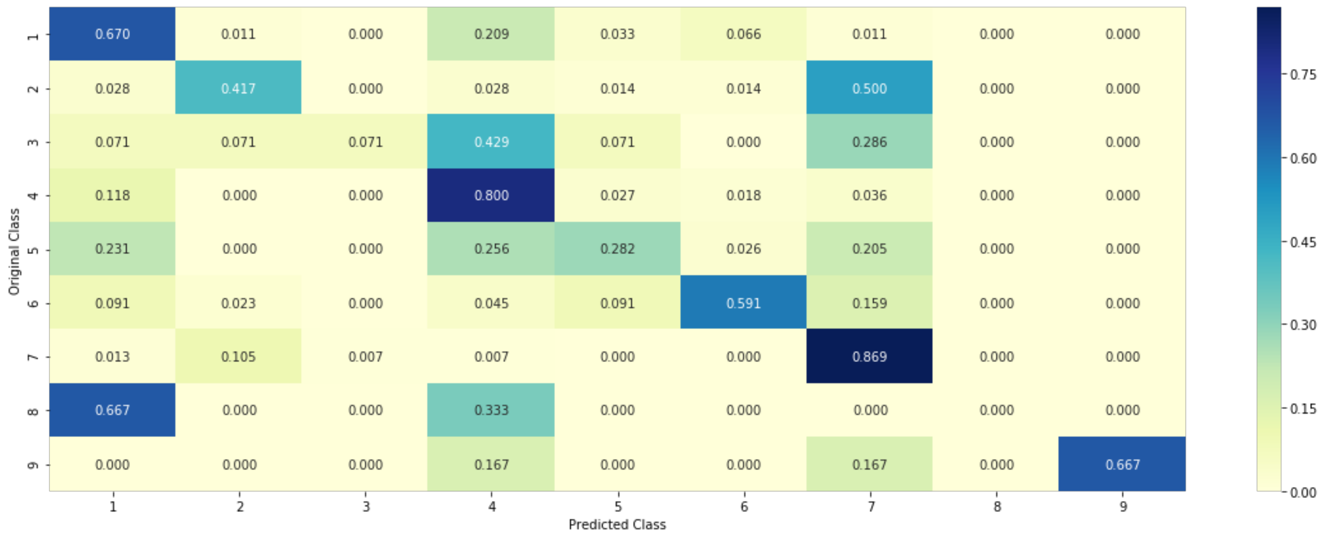
```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



1	0.649	0.020	0.000	0.146	0.130	0.167	0.005		0.000
2	0.021	0.612	0.000	0.015	0.043	0.028	0.186		0.000
3	0.011	0.020	0.500	0.046	0.043	0.000	0.021		0.000



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



4.5.4.3. Feature Importance, Correctly Classified point

In [106]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 2
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[9.008e-01 7.700e-03 5.900e-03 8.000e-03 7.800e-03 6.520e-02 1.300
e-03
  3.100e-03 1.000e-04]]
Actual Class : 1
--------------------------------------------------
0 Text feature [invasive] present in test data point [True]
3 Text feature [system] present in test data point [True]
4 Text feature [receptor] present in test data point [True]
8 Text feature [applied] present in test data point [True]
9 Text feature [51] present in test data point [True]
10 Text feature [important] present in test data point [True]
11 Text feature [et] present in test data point [True]
19 Text feature [complete] present in test data point [True]
21 Text feature [would] present in test data point [True]
29 Text feature [whether] present in test data point [True]
```

```
31 Text feature [cancer] present in test data point [True]
32 Text feature [obtain] present in test data point [True]
42 Text feature [potential] present in test data point [True]
43 Text feature [available] present in test data point [True]
45 Text feature [nature] present in test data point [True]
46 Text feature [1993] present in test data point [True]
49 Text feature [genetics] present in test data point [True]
53 Text feature [development] present in test data point [True]
58 Text feature [media] present in test data point [True]
73 Text feature [negative] present in test data point [True]
80 Text feature [insertions] present in test data point [True]
85 Text feature [supplemented] present in test data point [True]
87 Text feature [institutional] present in test data point [True]
88 Text feature [relative] present in test data point [True]
99 Text feature [unclear] present in test data point [True]
105 Text feature [al] present in test data point [True]
109 Text feature [second] present in test data point [True]
129 Text feature [25] present in test data point [True]
160 Text feature [provide] present in test data point [True]
211 Text feature [provided] present in test data point [True]
229 Text feature [normal] present in test data point [True]
237 Text feature [ref] present in test data point [True]
248 Text feature [initiation] present in test data point [True]
277 Text feature [related] present in test data point [True]
282 Text feature [first] present in test data point [True]
284 Text feature [largely] present in test data point [True]
285 Text feature [breast] present in test data point [True]
286 Text feature [determine] present in test data point [True]
308 Text feature [result] present in test data point [True]
319 Text feature [results] present in test data point [True]
343 Text feature [clearly] present in test data point [True]
353 Text feature [support] present in test data point [True]
360 Text feature [specific] present in test data point [True]
361 Text feature [pcr] present in test data point [True]
362 Text feature [suggested] present in test data point [True]
364 Text feature [significant] present in test data point [True]
382 Text feature [11] present in test data point [True]
400 Text feature [whereas] present in test data point [True]
423 Text feature [30] present in test data point [True]
427 Text feature [small] present in test data point [True]
443 Text feature [assessed] present in test data point [True]
468 Text feature [secondary] present in test data point [True]
479 Text feature [death] present in test data point [True]
482 Text feature [help] present in test data point [True]
Out of the top  500  features  54 are present in query point
```

4.5.4.4. Feature Importance, Inorrectly Classified point

In [107]:

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

```
Predicted Class : 5
Predicted Class Probabilities: [[0.1587 0.0789 0.018  0.1424 0.4394 0.0321 0.1125 0.0091 0.009 ]]
Actual Class : 1
--------------------------------------------------
0 Text feature [conserved] present in test data point [True]
1 Text feature [ref] present in test data point [True]
2 Text feature [detected] present in test data point [True]
13 Text feature [equal] present in test data point [True]
15 Text feature [notably] present in test data point [True]
17 Text feature [4a] present in test data point [True]
22 Text feature [20] present in test data point [True]
```

```
23 Text feature [genomic] present in test data point [True]
38 Text feature [aa] present in test data point [True]
43 Text feature [structural] present in test data point [True]
71 Text feature [dominant] present in test data point [True]
73 Text feature [fragment] present in test data point [True]
88 Text feature [comprehensive] present in test data point [True]
95 Text feature [consensus] present in test data point [True]
96 Text feature [based] present in test data point [True]
105 Text feature [resulted] present in test data point [True]
108 Text feature [possible] present in test data point [True]
128 Text feature [promoter] present in test data point [True]
141 Text feature [phosphate] present in test data point [True]
142 Text feature [prepared] present in test data point [True]
175 Text feature [common] present in test data point [True]
185 Text feature [contain] present in test data point [True]
204 Text feature [highly] present in test data point [True]
208 Text feature [examine] present in test data point [True]
218 Text feature [amount] present in test data point [True]
224 Text feature [expression] present in test data point [True]
243 Text feature [analysis] present in test data point [True]
251 Text feature [present] present in test data point [True]
307 Text feature [56] present in test data point [True]
322 Text feature [enrichment] present in test data point [True]
353 Text feature [translation] present in test data point [True]
354 Text feature [100] present in test data point [True]
389 Text feature [58] present in test data point [True]
392 Text feature [thereby] present in test data point [True]
404 Text feature [download] present in test data point [True]
405 Text feature [defects] present in test data point [True]
410 Text feature [lead] present in test data point [True]
412 Text feature [subset] present in test data point [True]
415 Text feature [clearly] present in test data point [True]
423 Text feature [sigma] present in test data point [True]
438 Text feature [fluorescence] present in test data point [True]
442 Text feature [blotting] present in test data point [True]
445 Text feature [targets] present in test data point [True]
451 Text feature [binding] present in test data point [True]
461 Text feature [dna] present in test data point [True]
478 Text feature [reporter] present in test data point [True]
483 Text feature [41] present in test data point [True]
Out of the top  500  features  47 are present in query point
```

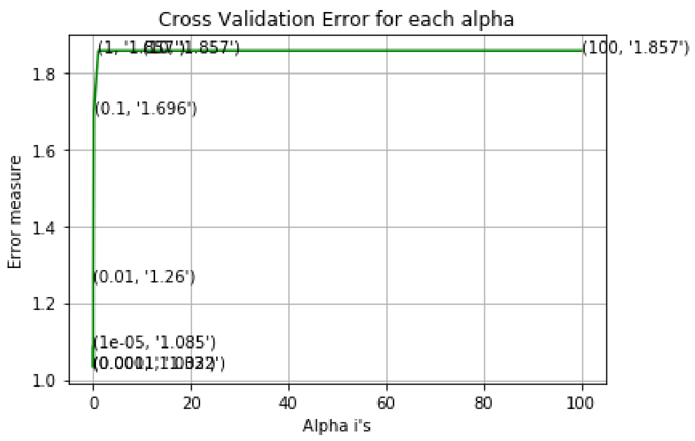
4.6. Linear Support Vector Machines

4.6.1. Hyper paramter tuning

In [140]:

```
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)
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', r
andom_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_ytr = 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_ytr, labels=clf.classes_, eps=1e-15))
predict_yte = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_yte, labels=clf.classes_, eps=1e-15))
predict_ycv = 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, p
redict_ycv, labels=clf.classes_, eps=1e-15))
prettyTableoutput.append(["LSVM",log_loss(y_train, predict_ytr, labels=clf.classes_, eps=1e-15),log
_loss(y_cv, predict_yte, labels=clf.classes_, eps=1e-15),log_loss(y_test, predict_ycv, labels=clf.c
lasses_, eps=1e-15)])
```

```
for C = 1e-05
Log Loss : 1.08523828895255
for C = 0.0001
Log Loss : 1.0324448203268934
for C = 0.001
Log Loss : 1.0319344287433443
for C = 0.01
Log Loss : 1.2602133052312678
for C = 0.1
Log Loss : 1.6964817528224665
for C = 1
Log Loss : 1.8565418591844314
for C = 10
Log Loss : 1.8565421914692832
for C = 100
Log Loss : 1.856542174059652
```



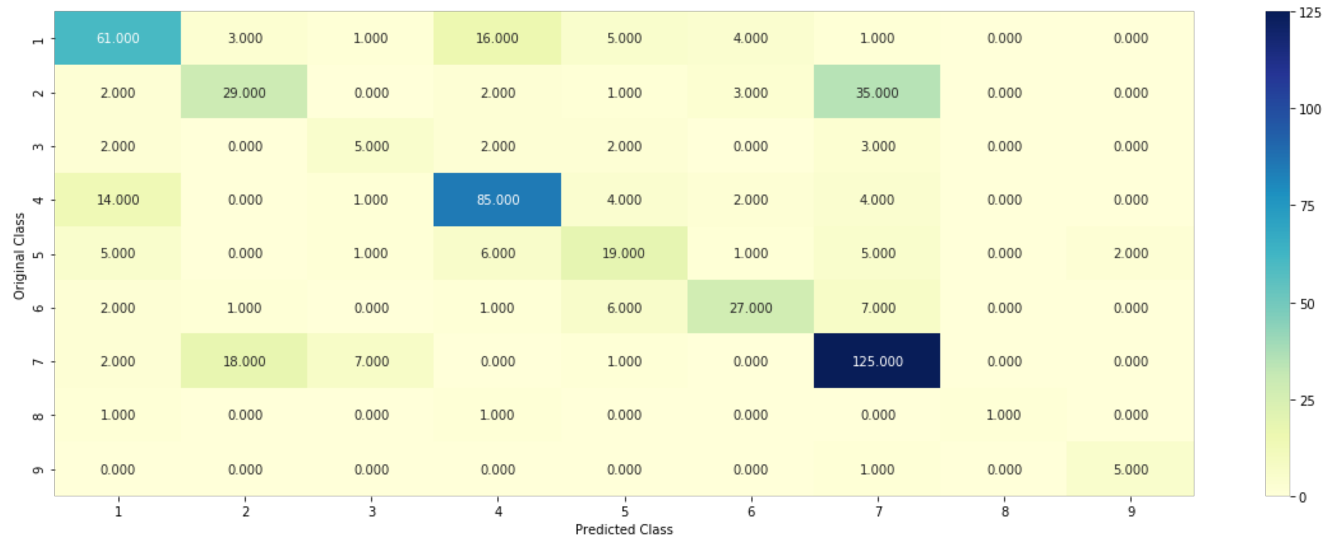
```
For values of best alpha =  0.001 The train log loss is: 0.5102513769349305
For values of best alpha =  0.001 The cross validation log loss is: 1.0319344287433443
For values of best alpha =  0.001 The test log loss is: 0.9912331117571903
```

4.6.2. Testing model with best hyper parameters

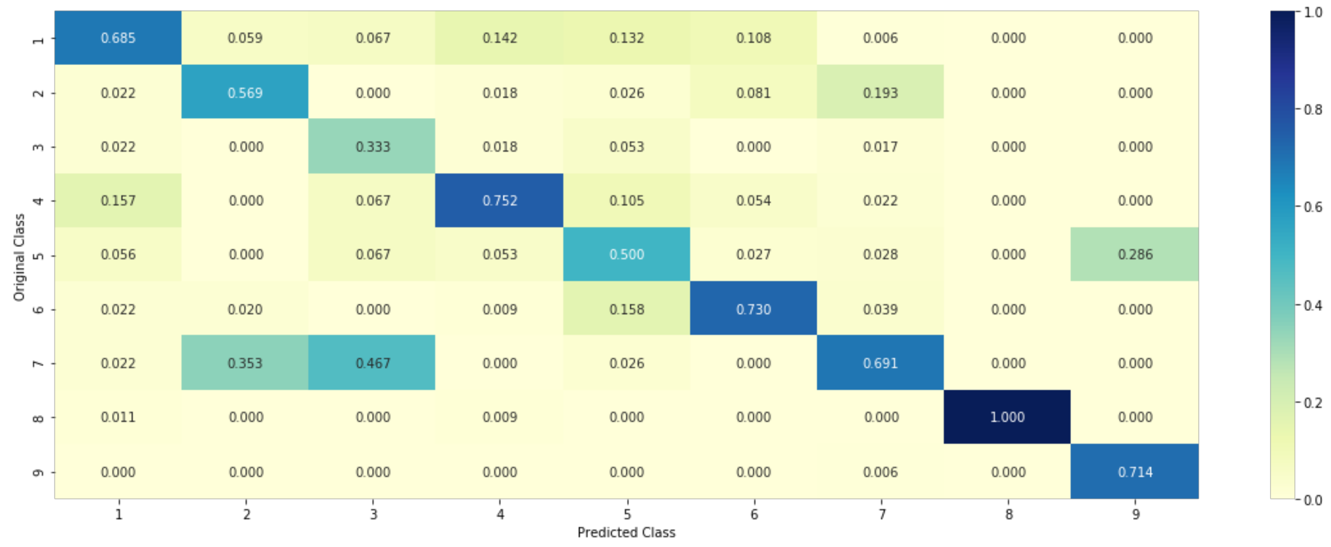
In [109]:

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

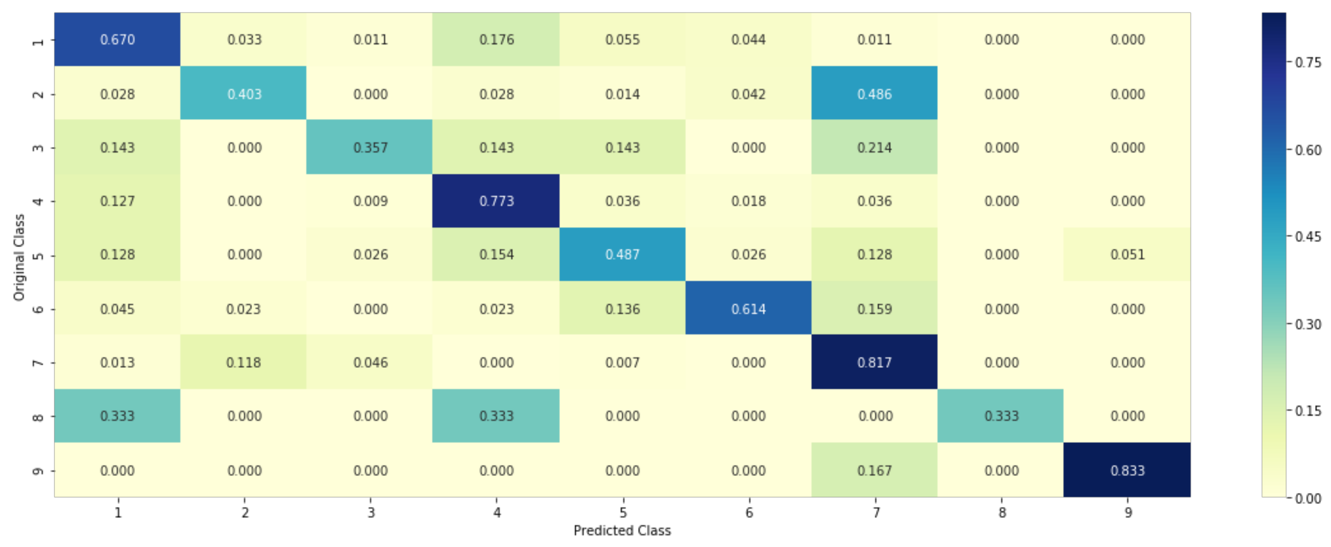
```
Log loss : 1.0319344287433443
Number of mis-classified points : 0.32894736842105265
-------------------- Confusion matrix --------------------
```



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



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



4.6.3.1. For Correctly classified point

In [114]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
```

```
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 3
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(abs(-clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[2.350e-02 5.150e-02 3.800e-03 1.750e-02 1.130e-02 3.000e-03 8.868
e-01
  2.300e-03 3.000e-04]]
Actual Class : 7
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.6.3.2. For Incorrectly classified point

In [111]:

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

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3398 0.0651 0.0099 0.3867 0.0814 0.0141 0.0961 0.0058 0.0009]]
Actual Class : 1
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.7 Random Forest Classifier

4.7.1. Hyper paramter tuning (With One hot Encoding)

In [141]:

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

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

```
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)),
(features[i],cv_log_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_ytr = 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_ytr, labels=clf.classes_, eps=1e-15))
predict_yte = 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_yte, labels=clf.classes_, eps=1e-15))
predict_ycv = 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_ycv, labels=clf.classes_, eps=1e-15))

prettyTableoutput.append(["Random Forest OHE",log_loss(y_train, predict_ytr, labels=clf.classes_, e
ps=1e-15),log_loss(y_cv, predict_yte, labels=clf.classes_, eps=1e-15),log_loss(y_test, predict_ycv,
labels=clf.classes_, eps=1e-15)])
```

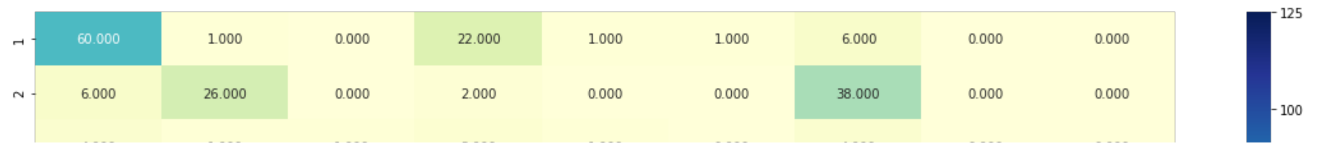
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2147830990471782
for n_estimators = 100 and max depth =  10
Log Loss : 1.2188125185771383
for n_estimators = 200 and max depth =  5
Log Loss : 1.1934903126814675
for n_estimators = 200 and max depth =  10
Log Loss : 1.20392757327882
for n_estimators = 500 and max depth =  5
Log Loss : 1.1723004670941852
for n_estimators = 500 and max depth =  10
Log Loss : 1.1961149471916208
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1725475339293125
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1932918604069347
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1728522682982092
for n_estimators = 2000 and max depth =  10
Log Loss : 1.189996316799993
For values of best estimator =  500 The train log loss is: 0.8724876352330444
For values of best estimator =  500 The cross validation log loss is: 1.1723004670941852
For values of best estimator =  500 The test log loss is: 1.137528902090071
```

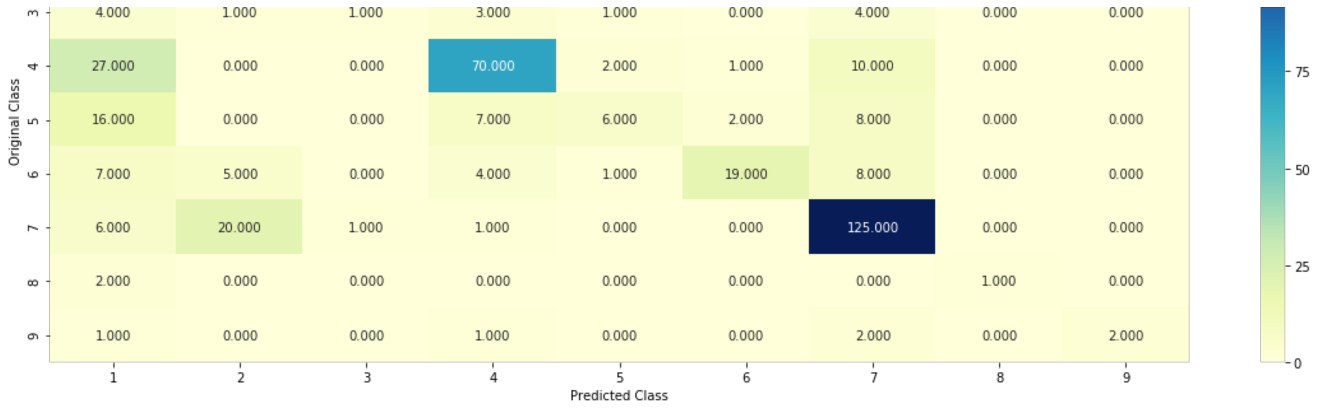
4.7.2. Testing model with best hyper parameters (One Hot Encoding)

In [117]:

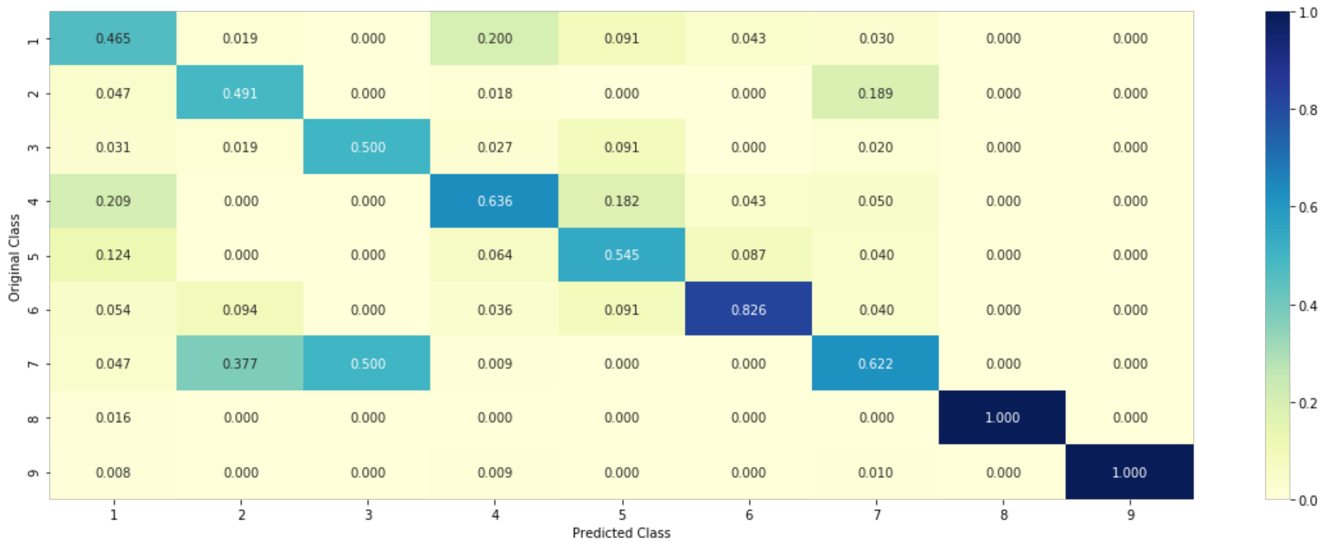
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max
_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

```
Log loss : 1.1723004670941852
Number of mis-classified points : 0.41729323308270677
-------------------- Confusion matrix --------------------
```

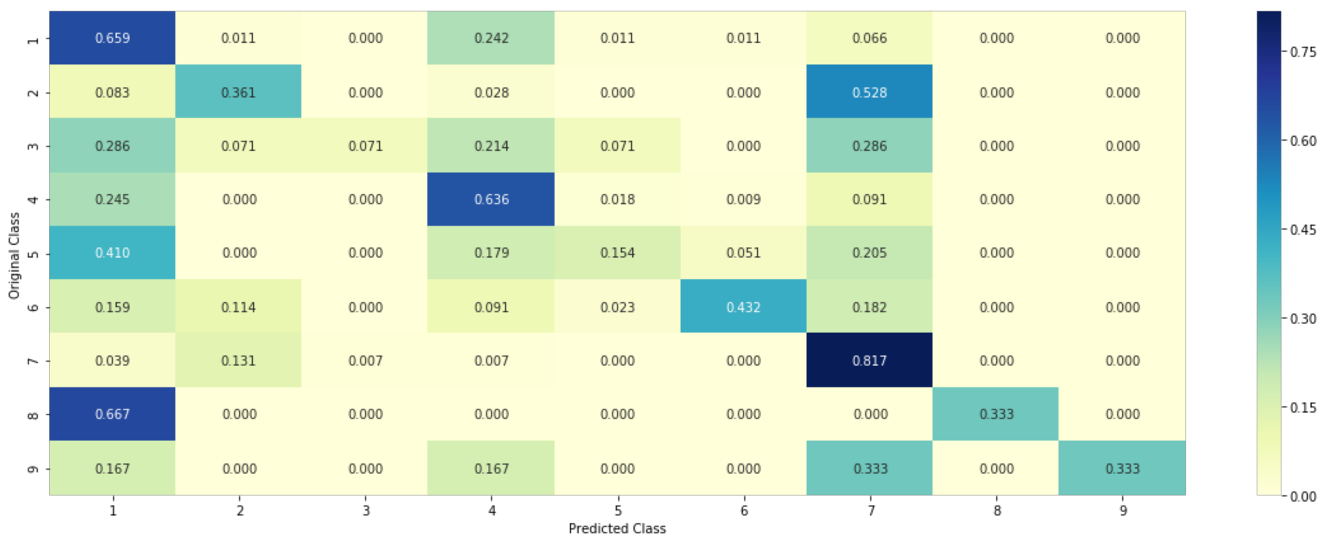




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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



4.7.3.1. Correctly Classified point

In [119]:

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

```
Predicted Class : 1
Predicted Class Probabilities: [[0.4789 0.0575 0.0155 0.226  0.0803 0.102  0.0279 0.0088 0.0032]]
Actual Class : 1
--------------------------------------------------
4 Text feature [suppressor] present in test data point [True]
7 Text feature [function] present in test data point [True]
10 Text feature [inhibitors] present in test data point [True]
11 Text feature [missense] present in test data point [True]
13 Text feature [receptor] present in test data point [True]
14 Text feature [loss] present in test data point [True]
16 Text feature [deleterious] present in test data point [True]
18 Text feature [brca1] present in test data point [True]
20 Text feature [therapy] present in test data point [True]
21 Text feature [months] present in test data point [True]
22 Text feature [nonsense] present in test data point [True]
23 Text feature [inhibitor] present in test data point [True]
27 Text feature [variants] present in test data point [True]
32 Text feature [cells] present in test data point [True]
33 Text feature [patients] present in test data point [True]
34 Text feature [protein] present in test data point [True]
36 Text feature [functional] present in test data point [True]
41 Text feature [therapeutic] present in test data point [True]
43 Text feature [cell] present in test data point [True]
44 Text feature [repair] present in test data point [True]
45 Text feature [likelihood] present in test data point [True]
46 Text feature [brca2] present in test data point [True]
47 Text feature [brca] present in test data point [True]
49 Text feature [drug] present in test data point [True]
50 Text feature [pten] present in test data point [True]
51 Text feature [frameshift] present in test data point [True]
53 Text feature [defective] present in test data point [True]
58 Text feature [variant] present in test data point [True]
61 Text feature [splice] present in test data point [True]
63 Text feature [carriers] present in test data point [True]
67 Text feature [efficacy] present in test data point [True]
69 Text feature [proteins] present in test data point [True]
72 Text feature [ovarian] present in test data point [True]
73 Text feature [carry] present in test data point [True]
77 Text feature [inactivation] present in test data point [True]
78 Text feature [predicted] present in test data point [True]
79 Text feature [treated] present in test data point [True]
87 Text feature [sensitivity] present in test data point [True]
90 Text feature [retained] present in test data point [True]
93 Text feature [lines] present in test data point [True]
94 Text feature [clinical] present in test data point [True]
95 Text feature [odds] present in test data point [True]
97 Text feature [response] present in test data point [True]
Out of the top  100  features  43 are present in query point
```

4.7.3.2. Inorrectly Classified point

In [120]:

```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("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_df['Gene'].
iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.3259 0.0373 0.0169 0.4107 0.0667 0.0549 0.0693 0.0074 0.0107]]
Actuall Class : 1
--------------------------------------------------
1 Text feature [activating] present in test data point [True]
3 Text feature [phosphorylation] present in test data point [True]
5 Text feature [activation] present in test data point [True]
6 Text feature [activated] present in test data point [True]
7 Text feature [function] present in test data point [True]
9 Text feature [signaling] present in test data point [True]
10 Text feature [inhibitors] present in test data point [True]
12 Text feature [erk] present in test data point [True]
13 Text feature [receptor] present in test data point [True]
14 Text feature [loss] present in test data point [True]
15 Text feature [treatment] present in test data point [True]
24 Text feature [stability] present in test data point [True]
30 Text feature [activate] present in test data point [True]
32 Text feature [cells] present in test data point [True]
34 Text feature [protein] present in test data point [True]
36 Text feature [functional] present in test data point [True]
38 Text feature [expression] present in test data point [True]
43 Text feature [cell] present in test data point [True]
51 Text feature [frameshift] present in test data point [True]
56 Text feature [extracellular] present in test data point [True]
60 Text feature [respond] present in test data point [True]
65 Text feature [expressing] present in test data point [True]
69 Text feature [proteins] present in test data point [True]
70 Text feature [functions] present in test data point [True]
74 Text feature [downstream] present in test data point [True]
75 Text feature [phosphatase] present in test data point [True]
78 Text feature [predicted] present in test data point [True]
79 Text feature [treated] present in test data point [True]
83 Text feature [57] present in test data point [True]
85 Text feature [favor] present in test data point [True]
90 Text feature [retained] present in test data point [True]
96 Text feature [active] present in test data point [True]
97 Text feature [response] present in test data point [True]
98 Text feature [disrupt] present in test data point [True]
Out of the top  100  features  34 are present in query point
```

4.7.4. Hyper paramter tuning (With Response Coding)

In [142]:

```
alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42
, n_jobs=-1)
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)),
(features[i],cv_log_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_ytr = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log loss is:",log_loss(y
_train, predict_ytr, labels=clf.classes_, eps=1e-15))
predict_yte = 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_yte, labels=clf.classes_, eps=1e-15))
predict_ycv = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log loss is:",log_loss(y_
test, predict_ycv, labels=clf.classes_, eps=1e-15))
prettyTableoutput.append(["Random Forest Resp Code",log_loss(y_train, predict_ytr,
labels=clf.classes_, eps=1e-15),log_loss(y_cv, predict_yte, labels=clf.classes_, eps=1e-15),log_los
s(y_test, predict_ycv, labels=clf.classes_, eps=1e-15)])
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.1724603995355896
for n_estimators = 10 and max depth =  3
Log Loss : 1.8195146656832983
for n_estimators = 10 and max depth =  5
Log Loss : 1.5634853908805266
for n_estimators = 10 and max depth =  10
Log Loss : 1.8154461553511276
for n_estimators = 50 and max depth =  2
Log Loss : 1.6420357189761514
for n_estimators = 50 and max depth =  3
Log Loss : 1.4033603061830195
for n_estimators = 50 and max depth =  5
Log Loss : 1.4135028168966204
for n_estimators = 50 and max depth =  10
Log Loss : 1.7949639607817192
for n_estimators = 100 and max depth =  2
Log Loss : 1.5263114064459304
for n_estimators = 100 and max depth =  3
Log Loss : 1.465263313911705
for n_estimators = 100 and max depth =  5
Log Loss : 1.380188702223879
for n_estimators = 100 and max depth =  10
Log Loss : 1.7131250089140109
for n_estimators = 200 and max depth =  2
Log Loss : 1.5846328392535531
for n_estimators = 200 and max depth =  3
Log Loss : 1.466317879287116
for n_estimators = 200 and max depth =  5
Log Loss : 1.4406930389580634
for n_estimators = 200 and max depth =  10
Log Loss : 1.7169477180719284
for n_estimators = 500 and max depth =  2
Log Loss : 1.672417220921739
for n_estimators = 500 and max depth =  3
Log Loss : 1.535021688042796
for n_estimators = 500 and max depth =  5
Log Loss : 1.4608488028767863
for n_estimators = 500 and max depth =  10
Log Loss : 1.7895986796530303
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6426579485710626
for n_estimators = 1000 and max depth =  3
Log Loss : 1.539670609459641
for n_estimators = 1000 and max depth =  5
Log Loss : 1.4428409775544606
for n_estimators = 1000 and max depth =  10
Log Loss : 1.7896890483623433
For values of best alpha =  100 The train log loss is: 0.0605710676751971
For values of best alpha =  100 The cross validation log loss is: 1.380188702223879
For values of best alpha =  100 The test log loss is: 1.34479147759558
```

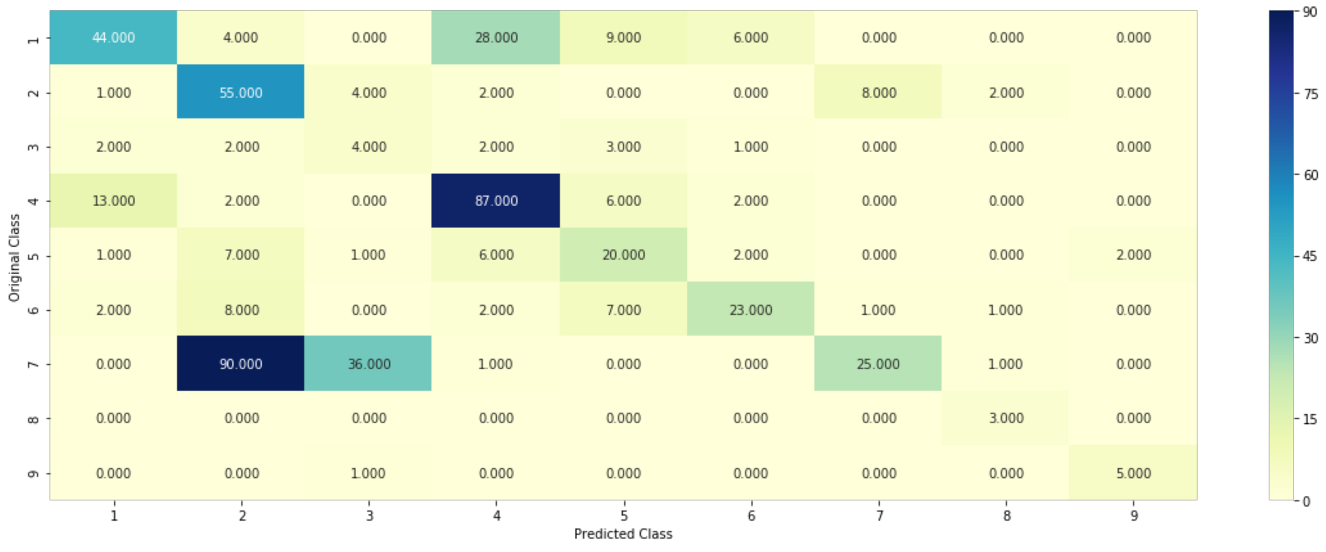
4.7.5. Testing model with best hyper parameters (Response Coding)

Testing model with best hyper parameters (Response Coding)

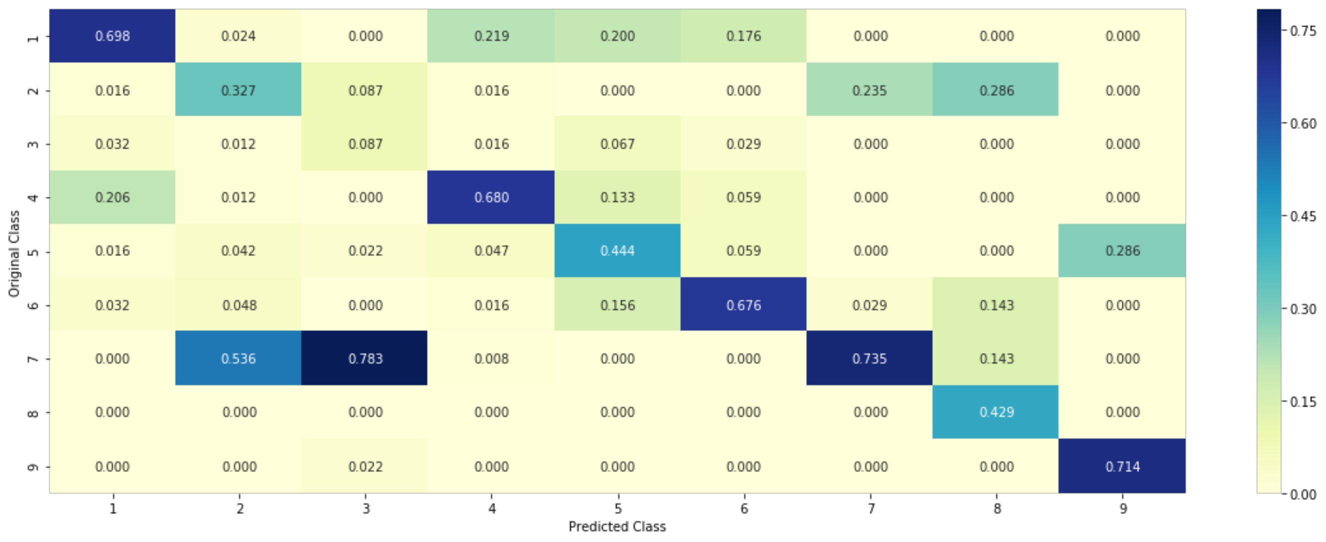
In [122]:

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

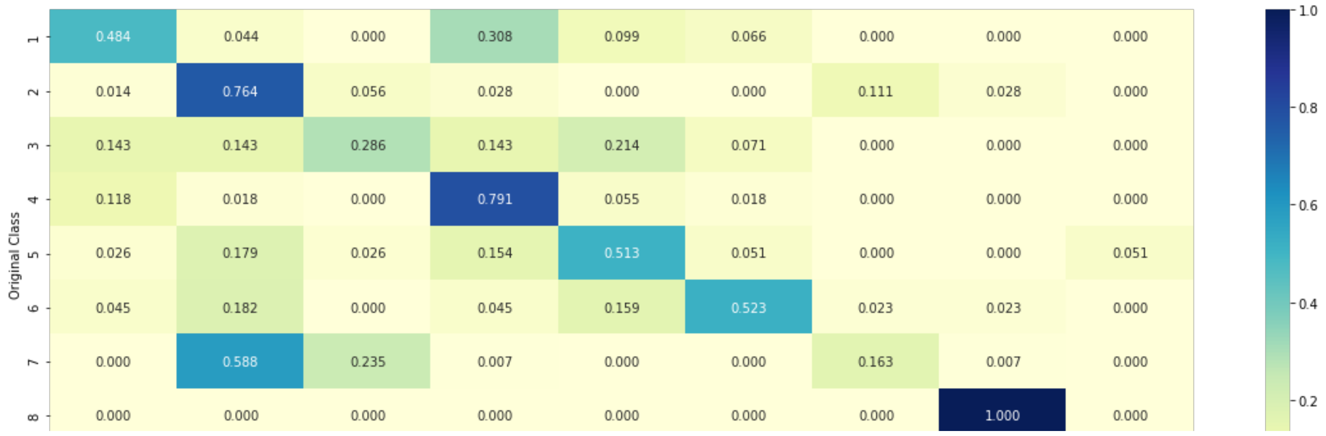
```
Log loss : 1.380188702223879
Number of mis-classified points : 0.5
-------------------- Confusion matrix --------------------
```

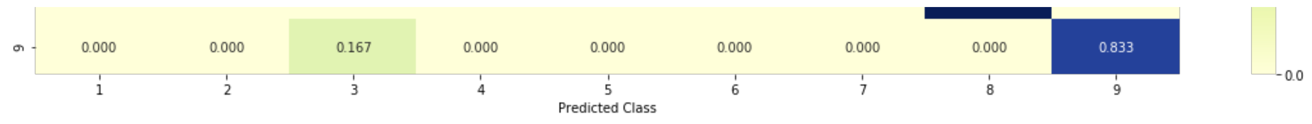


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





4.7.5.1. Correctly Classified point

In [127]:

```
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 = 2
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.9646 0.0025 0.0038 0.0099 0.002  0.0075 0.0019 0.0041 0.0036]]
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text 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
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.7.5.2. Incorrectly Classified point

In [124]:

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

```
Predicted Class : 5
Predicted Class Probabilities: [[0.1209 0.0104 0.0886 0.2064 0.3332 0.2135 0.0057 0.0117 0.0096]]
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text 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
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.8 Stack the models

4.8.1 testing with hyper parameter tuning

In [128]:

```
clf1 = SGDClassifier(alpha=0.0001, 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=0.001, 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=0.00001)
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_onehot
Coding))))
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_p
robas=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, sc
lf.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
```

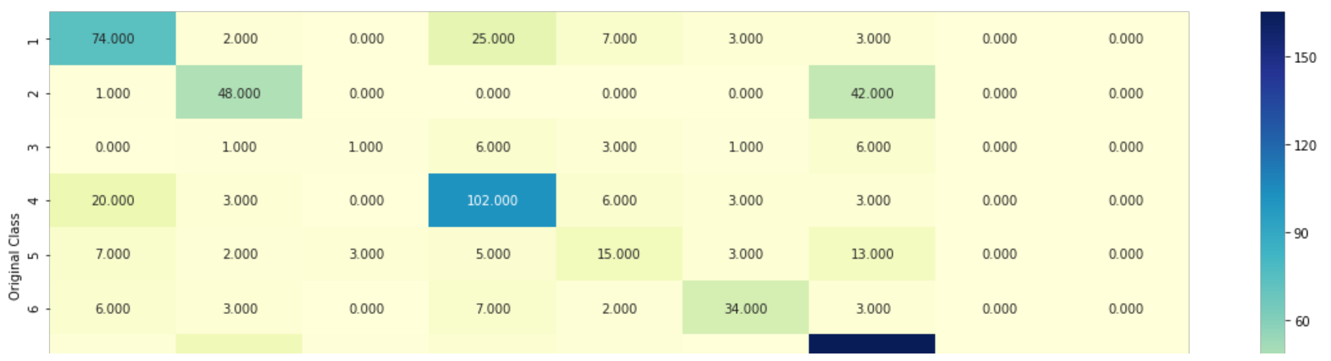
```
Logistic Regression :  Log Loss: 1.01
Support vector machines : Log Loss: 1.04
Naive Bayes : Log Loss: 1.20
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.173
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.987
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.378
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.089
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.338
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.908
```

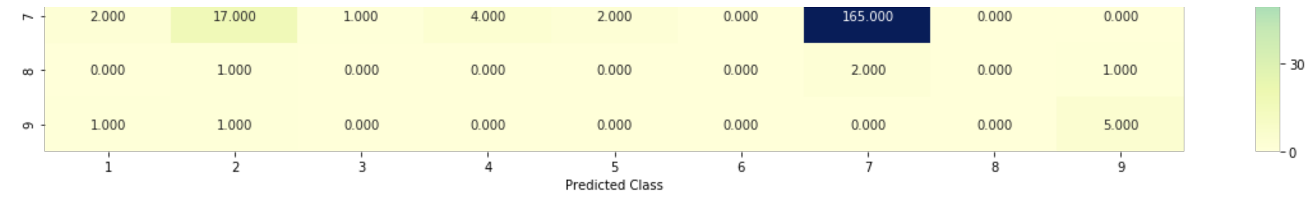
4.8.2 testing the model with the best hyper parameters

In [143]:

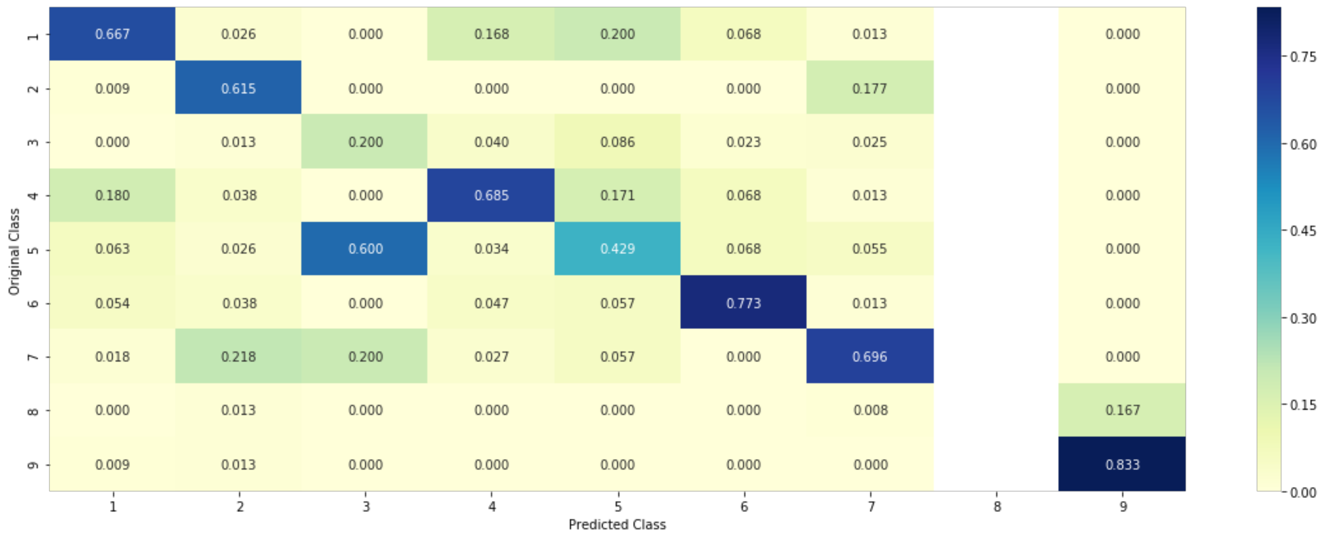
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba
s=True)
sclf.fit(train_x_onehotCoding, train_y)

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)

prettyTableoutput.append(["Stacking  model",log_loss(y_train,
sclf.predict_proba(train_x_onehotCoding), labels=clf.classes_, eps=1e-15),log_loss(y_cv,
sclf.predict_proba(cv_x_onehotCoding), labels=clf.classes_, eps=1e-15),log_loss(y_test, sclf.predic
t_proba(test_x_onehotCoding), labels=clf.classes_, eps=1e-15)])
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 : 0.3557781386447962
Log loss (CV) on the stacking classifier : 1.089114772605015
Log loss (test) on the stacking classifier : 1.021230438993654
Number of missclassified point : 0.3323308270676692
-------------------- Confusion matrix --------------------
```

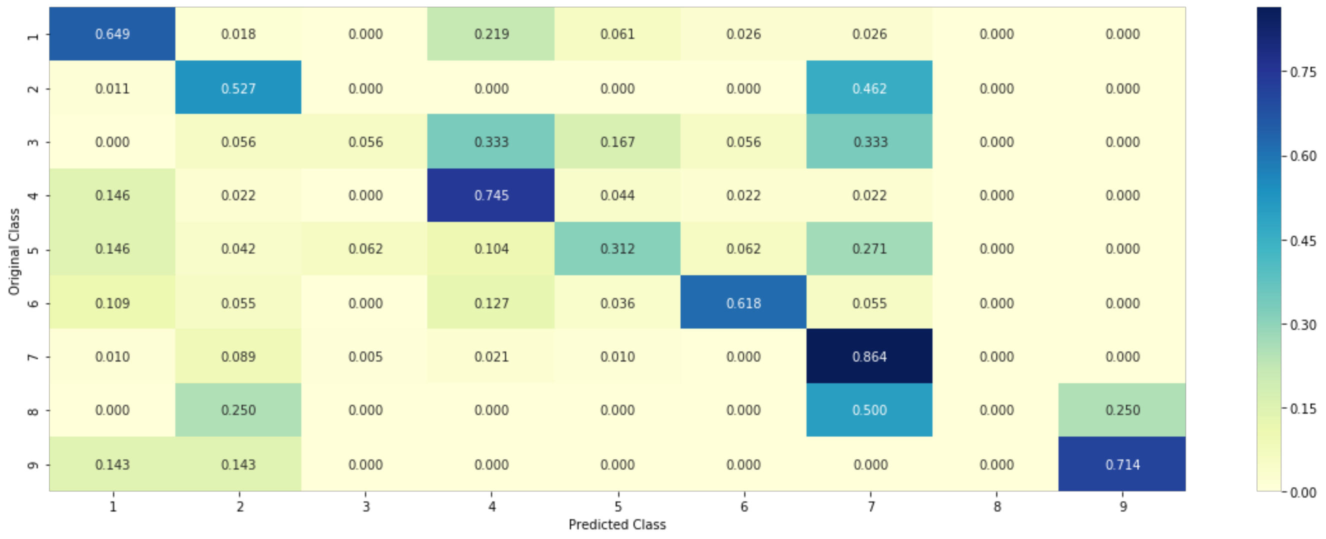




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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

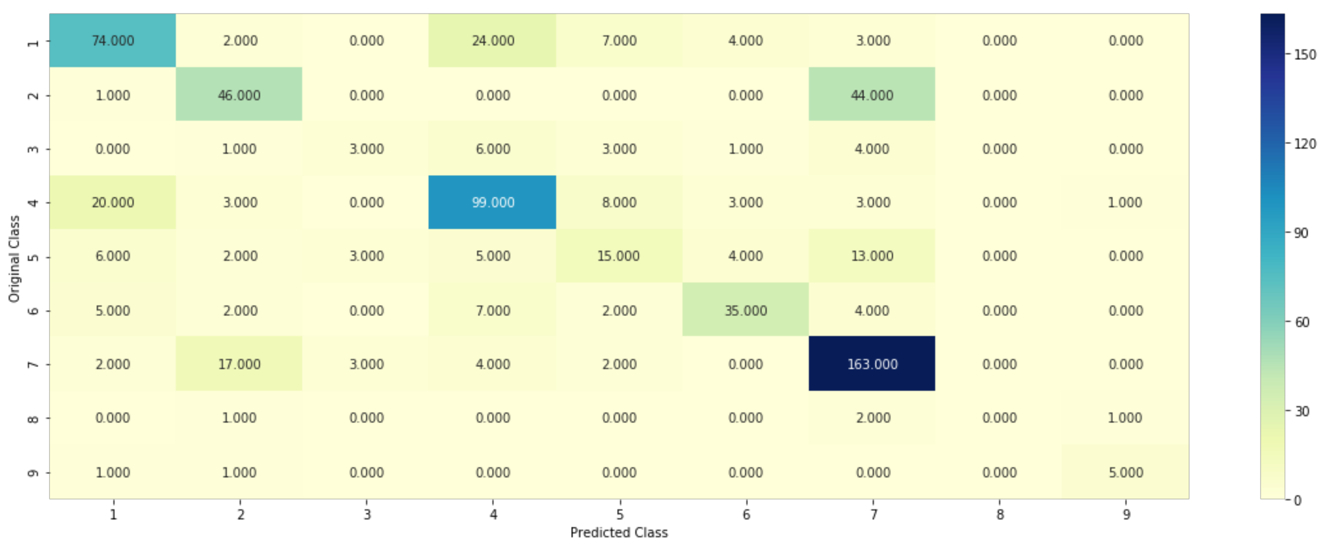


4.8.3 Maximum Voting classifier

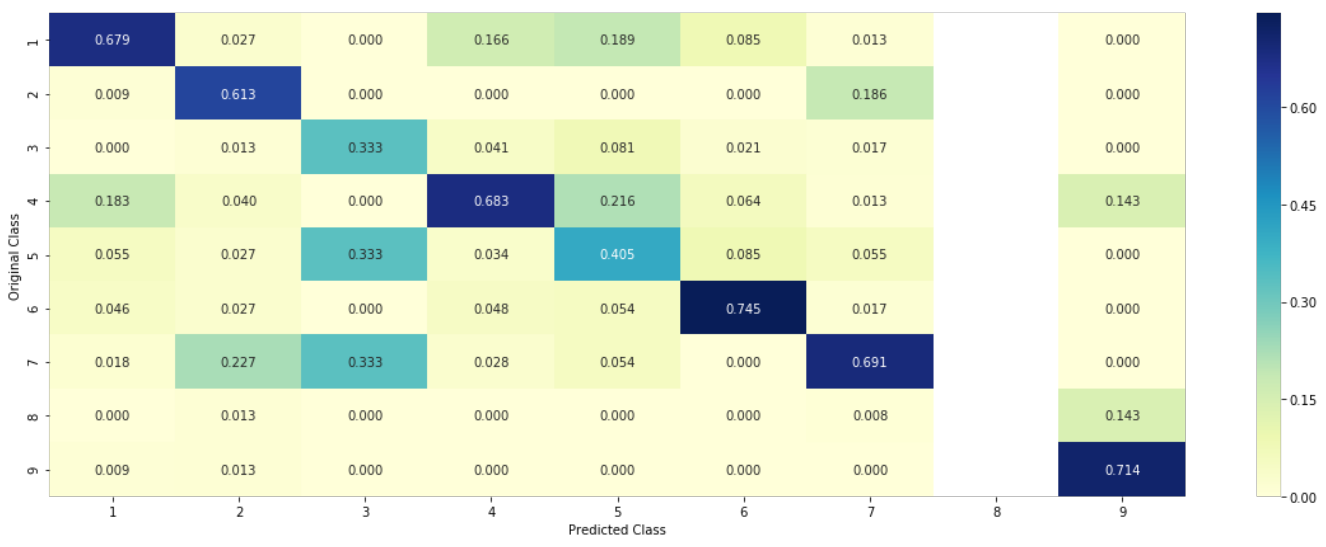
In [144]:

```
#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])
prettyTableoutput.append(["Maximum Voting  model",log_loss(y_train,
vclf.predict_proba(train_x_onehotCoding), labels=clf.classes_, eps=1e-15),log_loss(y_cv,
vclf.predict_proba(train_x_onehotCoding), labels=clf.classes_, eps=1e-15),log_loss(y_cv,
vclf.predict_proba(cv_x_onehotCoding), labels=clf.classes_, eps=1e-15),log_loss(y_test, vclf.predic
t_proba(test_x_onehotCoding), labels=clf.classes_, eps=1e-15)])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

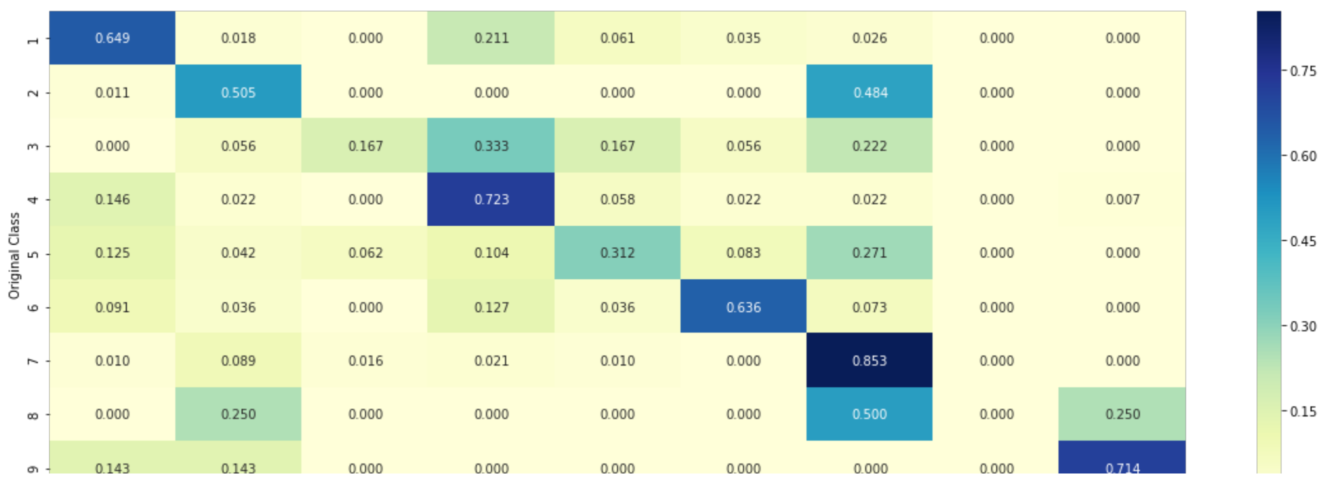
```
Log loss (train) on the VotingClassifier : 0.49827764847386 38
Log loss (CV) on the VotingClassifier : 1.0072106543645414
Log loss (test) on the VotingClassifier : 0.9785075668110692
Number of missclassified point : 0.3383458646616541
-------------------- Confusion matrix --------------------
```

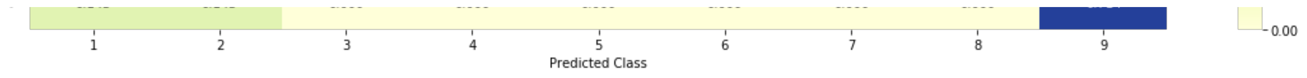


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





Summary

In [145]:

```
from prettytable import PrettyTable

y = PrettyTable()
y.field_names= ["Model","Train loss","CV loss","Test loss"]
for i in prettyTableoutput:
    y.add_row(i)
print(y)
```

```
+--------------------------+---------------------+--------------------+---------------------+
|          Model           |      Train loss     |      CV loss       |      Test loss      |
+--------------------------+---------------------+--------------------+---------------------+
|       Random Model       |          NA         | 2.490891796436401  | 2.5163336395854863 |
|       Naive Bayes        |  0.5689793768477163 | 1.1999985893061975 | 1.1701596926393678 |
|    K Nearest Neighbour   |  0.7269666051830004 | 1.0442220202853572 | 1.0230340027479514 |
|    LR with Classbalance  |  0.4516198785675059 | 0.9490146449070358 | 0.9384377667266186 |
| LR without Classbalance  | 0.43413347783552025 | 1.0342580261370453 | 0.960636842038146  |
|           LSVM           |  0.5102513769349305 | 1.0319344287433443 | 0.9912331117571903 |
|     Random Forest OHE    |  0.8724876352330444 | 1.1723004670941852 | 1.137528902090071  |
| Random Forest Resp Code  |  0.0605710676751971 | 1.380188702223879  |  1.34479147759558  |
|      Stacking  model     |  0.3557781386447962 | 1.089114772605015  | 1.021230438993654  |
|   Maximum Voting  model  |  0.4982776484738638 | 1.0072106543645414 | 0.9785075668110692 |
+--------------------------+---------------------+--------------------+---------------------+
```

In [ ]: