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 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. 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 [102]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.cross_validation import StratifiedKFold
from collections import Counter, defaultdict
```

```python
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

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

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [103]:

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

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

Out[103]:

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

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

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

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

3.1.2. Reading Text Data

In [104]:

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

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

Out[104]:

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

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

3.1.3. Preprocessing of text

In [105]:
```
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))


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

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

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

In [106]:
```
#text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
```

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

Time took for preprocessing the text : 645.784023893335 seconds

In [107]:
```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[107]:

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

In [108]:
```
result.loc[result['TEXT'] =='null ','TEXT']
```

Out[108]:
```
1109    null
1277    null
1407    null
1639    null
2755    null
Name: TEXT, dtype: object
```

In [109]:
```
result.loc[result['TEXT'] =='null ','TEXT'] = result['Gene'] +' '+resul
t['Variation']
```

In [110]:
```
result[result['ID']==1109]
```

Out[110]:

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

3.1.4. Test, Train and Cross Validation Split

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

In [111]:
```
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

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

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

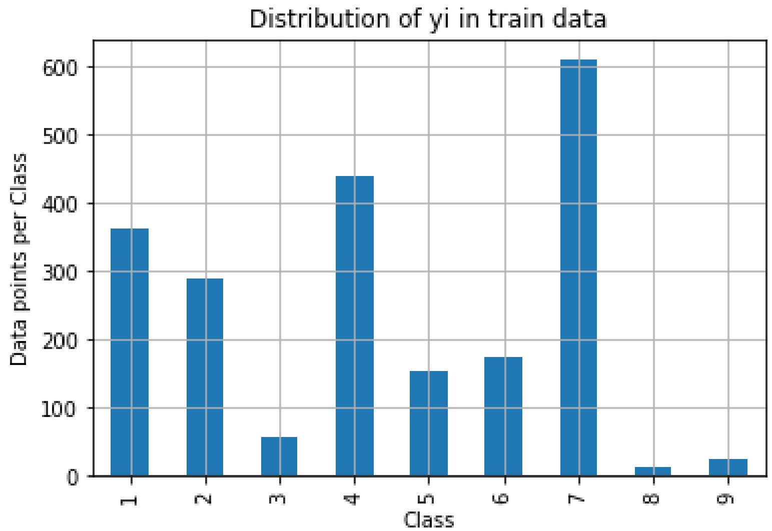
In [112]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0
])
```

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

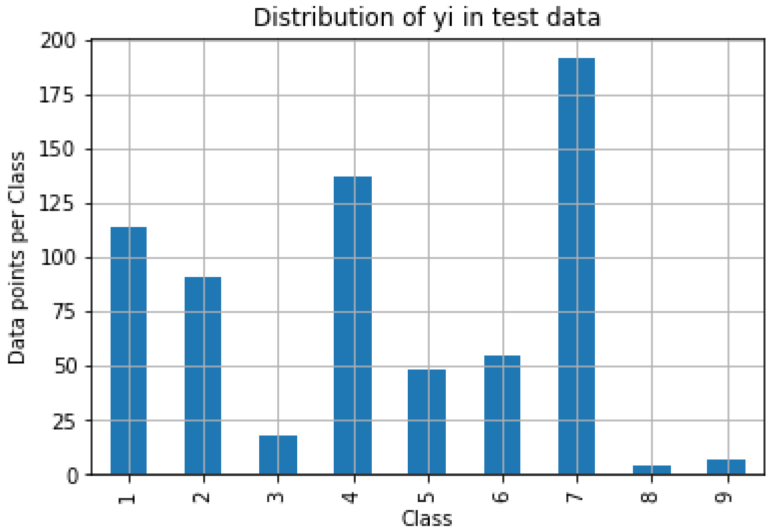
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [113]:
```
train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

my_colors = 'rgbkymc'
train_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distri
bution.values[i], '(', np.round((train_class_distribution.values[i]/tra
in_df.shape[0]*100), 3), '%)')


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

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

print('-'*80)
my_colors = 'rgbkymc'
cv_class_distribution.plot(kind='bar')
```

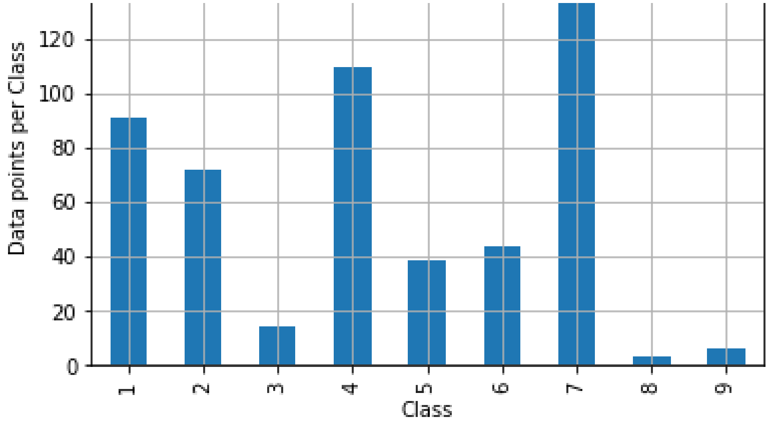
```
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribut
ion.values[i], '(', np.round((cv_class_distribution.values[i]/cv_df.sha
pe[0]*100), 3), '%)')
```




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




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




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

3.2 Prediction using a 'Random' Model

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

In [114]:
```
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    A =(((C.T)/(C.sum(axis=1))).T)
    B =(C/C.sum(axis=0))
    labels = [1,2,3,4,5,6,7,8,9]
    print("-"*20, "Confusion matrix", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=la
```

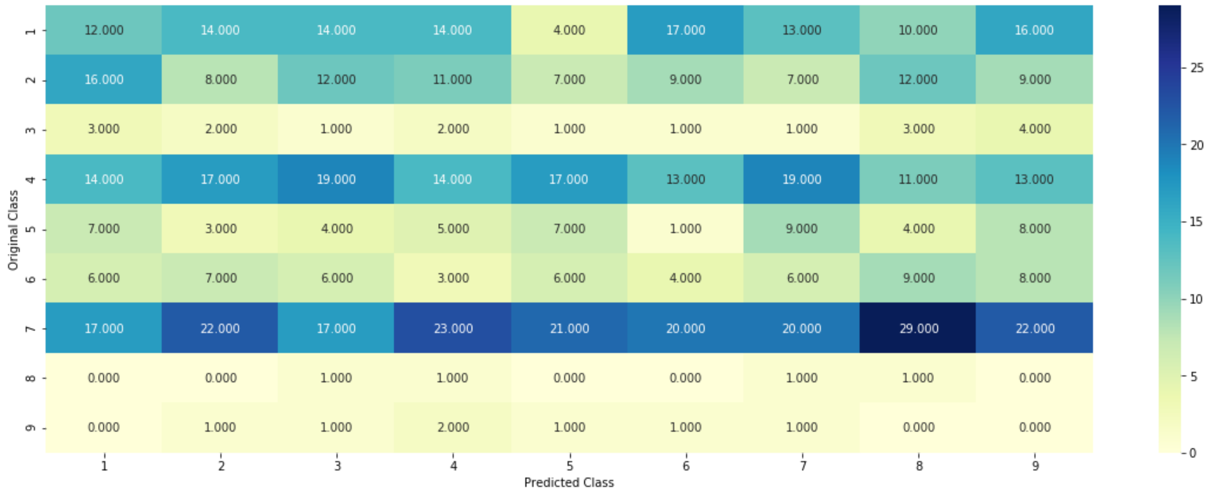
```
bels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

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

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

In [115]:

```
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_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    #print(rand_probs)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
    #print(test_predicted_y[i])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_p
```

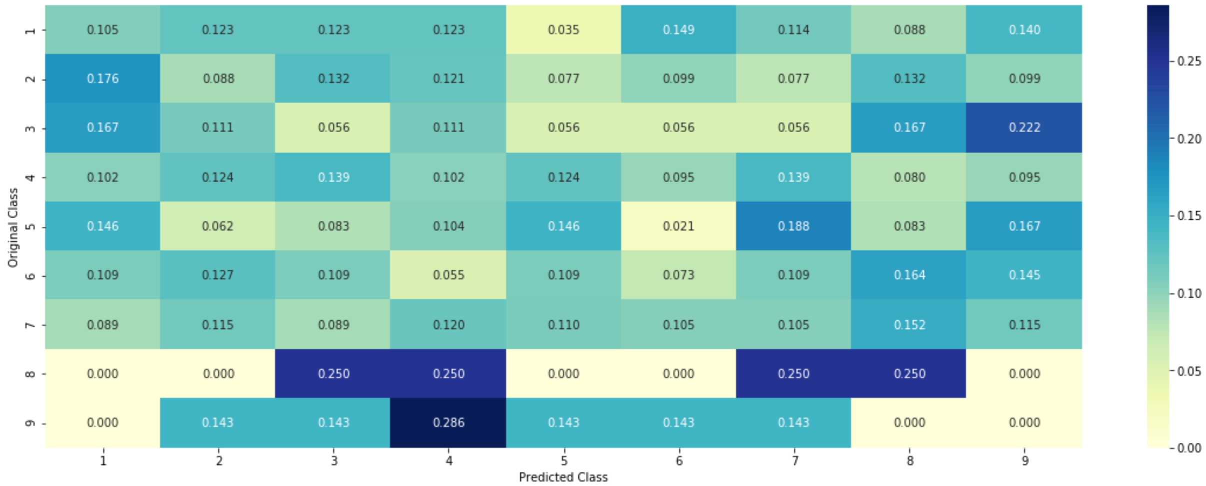
```
redicted_y, eps=1e-15))
predicted_y =np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y+1)
```

```
Log loss on Cross Validation Data using Random Model 2.45879619573
Log loss on Test Data using Random Model 2.45745405995
-------------------- Confusion matrix --------------------
```




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


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



3.3 Univariate Analysis

In [116]:

```
def get_gv_fea_dict(alpha, feature, df):
    value_count = train_df[feature].value_counts()
    gv_dict = dict()
```

```
    for i, denominator in value_count.items():
        vec = []
        for k in range(1,10):

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

    # gv_fea: Gene_variation feature, it will contain the feature for e
ach feature value in the data
    gv_fea = []
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
    return gv_fea
```

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

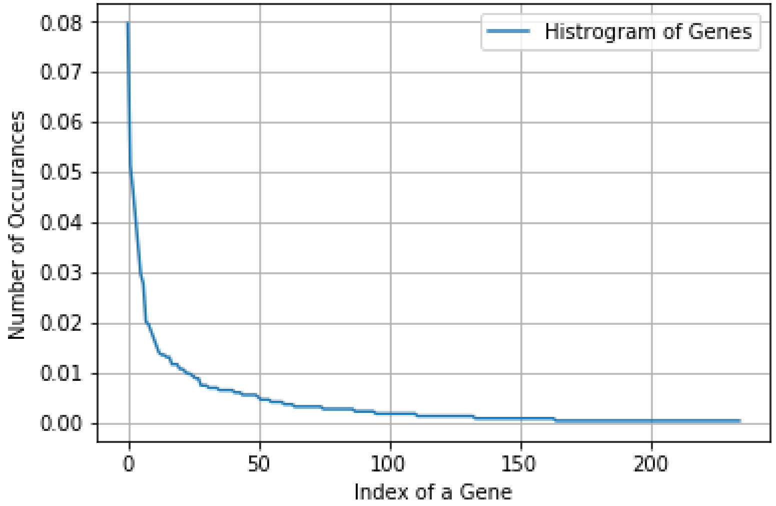
In [117]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 235
BRCA1     169
TP53      109
EGFR       99
BRCA2      87
PTEN       75
BRAF       63
KIT        59
ERBB2      43
ALK        42
PDGFRA     39
Name: Gene, dtype: int64
```

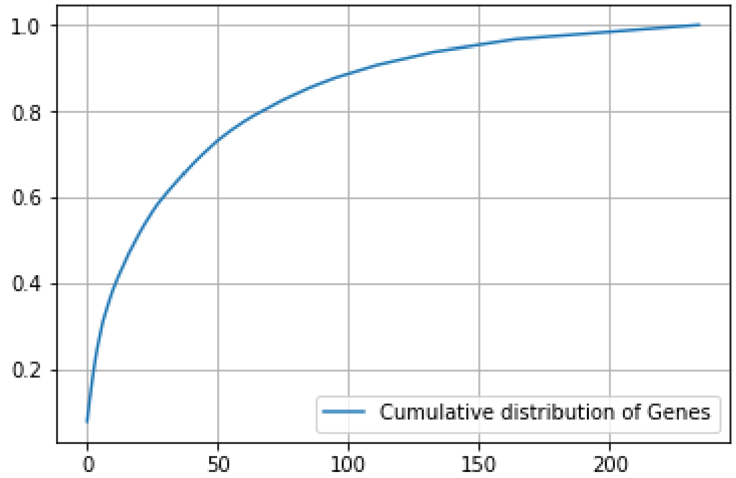
In [118]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of
 genes in the train data, and they are distibuted as follows",)
```

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

In [119]:
```
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histrogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [120]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

In [121]:
```
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gen
e", train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gen
e", test_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene",
 cv_df))
```

In [122]:
```
print("train_gene_feature_responseCoding is converted feature using res
pone coding method. The shape of gene feature:", train_gene_feature_res
ponseCoding.shape)
```

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

In [123]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
#gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_d
f['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gen
```

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

In [124]:
```
train_df['Gene'].head()
```

Out[124]:
```
549      SMAD2
879     PDGFRA
642     CDKN2A
349       CDH1
2693      BRAF
Name: Gene, dtype: object
```

In [125]:
```
gene_vectorizer.get_feature_names()
```

Out[125]:
```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl1',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'aurkb',
 'axin1',
 'axl',
 'b2m',
 'bap1',
 'bard1',
```

```
'bcl10',
'bcl2',
'bcl2l11',
'bcor',
'braf',
'brca1',
'brca2',
'brd4',
'brip1',
'btk',
'card11',
'carm1',
'casp8',
'cbl',
'ccnd1',
'ccnd2',
'ccnd3',
'cdh1',
'cdk12',
'cdk4',
'cdk6',
'cdk8',
'cdkn1a',
'cdkn1b',
'cdkn2a',
'cdkn2b',
'cdkn2c',
'chek2',
'cic',
'crebbp',
'ctcf',
'ctnnb1',
'ddr2',
'dicer1',
'dnmt3a',
'dnmt3b',
'egfr',
'eif1ax',
'elf3',
```

```
'ep300',
'epas1',
'erbb2',
'erbb3',
'erbb4',
'ercc2',
'ercc3',
'ercc4',
'erg',
'errfi1',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fam58a',
'fanca',
'fat1',
'fbxw7',
'fgf19',
'fgf3',
'fgf4',
'fgfr1',
'fgfr2',
'fgfr3',
'fgfr4',
'flt1',
'flt3',
'foxa1',
'foxl2',
'foxo1',
'foxp1',
'fubp1',
'gata3',
'gna11',
'gnaq',
'gnas',
'h3f3a',
'hla',
```

```
'hnf1a',
'hras',
'idh1',
'idh2',
'igf1r',
'ikzf1',
'il7r',
'inpp4b',
'jak1',
'jak2',
'kdm5a',
'kdm5c',
'kdm6a',
'kdr',
'keap1',
'kit',
'kmt2a',
'kmt2b',
'kmt2c',
'kmt2d',
'knstrn',
'kras',
'lats1',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
'mdm2',
'med12',
'mef2b',
'met',
'mga',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'mycn',
```

```
'ncor1',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',
'nkx2',
'notch1',
'notch2',
'npm1',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'nup93',
'pax8',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3r1',
'pik3r2',
'pim1',
'pms1',
'pms2',
'pole',
'ppm1d',
'ppp2r1a',
'ppp6c',
'prdm1',
'ptch1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rab35',
'rac1',
'rad21',
```

```
'rad50',
'rad51b',
'rad51d',
'rad54l',
'raf1',
'rasa1',
'rb1',
'rbm10',
'ret',
'rheb',
'rhoa',
'rictor',
'rit1',
'ros1',
'runx1',
'rxra',
'rybp',
'sdhb',
'setd2',
'sf3b1',
'shoc2',
'smad2',
'smad3',
'smad4',
'smarca4',
'smarcb1',
'smo',
'sos1',
'sox9',
'spop',
'src',
'srsf2',
'stag2',
'stat3',
'stk11',
'tcf7l2',
'tert',
'tet1',
'tet2',
```

```
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vhl',
 'whsc1',
 'whsc1l1',
 'xpo1',
 'xrcc2',
 'yap1']
```

In [126]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature:", train_gene_feature_onehotCoding.shape)
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature: (2124, 234)
```

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

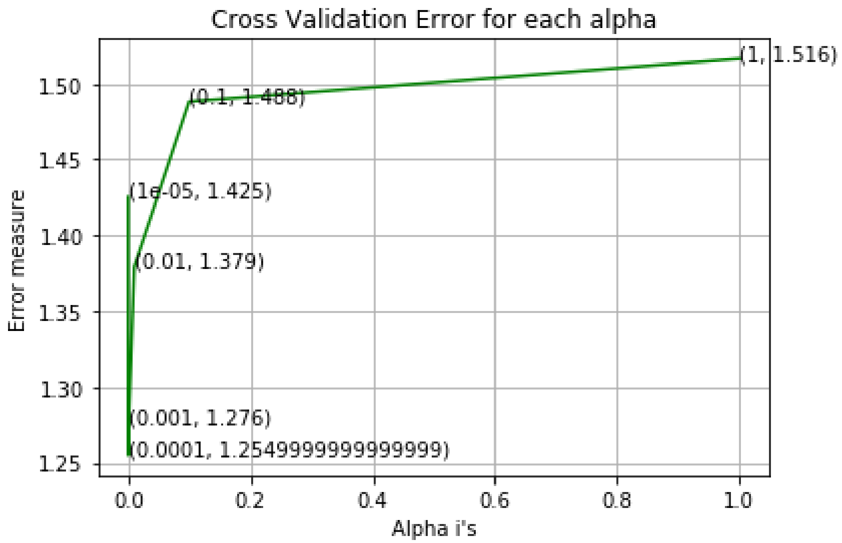
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [127]:
```
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.clas
ses_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv
, predict_y, labels=clf.classes_, eps=1e-15))

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.42535303258
For values of alpha =  0.0001 The log loss is: 1.25494852594
```

```
For values of alpha =  0.001 The log loss is: 1.27601595152
For values of alpha =  0.01 The log loss is: 1.37923009286
For values of alpha =  0.1 The log loss is: 1.48825775601
For values of alpha =  1 The log loss is: 1.51644458295
```




```
For values of best alpha =  0.0001 The train log loss is: 1.04305905998
For values of best alpha =  0.0001 The cross validation log loss is: 1.
25494852594
For values of best alpha =  0.0001 The test log loss is: 1.19603539299
```

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 [128]:
```
print("Q6. How many data points in Test and CV datasets are covered by
 the ", unique_genes.shape[0], " genes in train dataset?")

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

```
cv_coverage=cv_df[cv_df['Gene'].isin(list(set(train_df['Gene'])))].shap
e[0]

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

```
Q6. How many data points in Test and CV datasets are covered by the  23
5  genes in train dataset?
Ans
1. In test data 647 out of 665 : 97.29323308270676
2. In cross validation data 514 out of  532 : 96.61654135338345
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

In [129]:

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

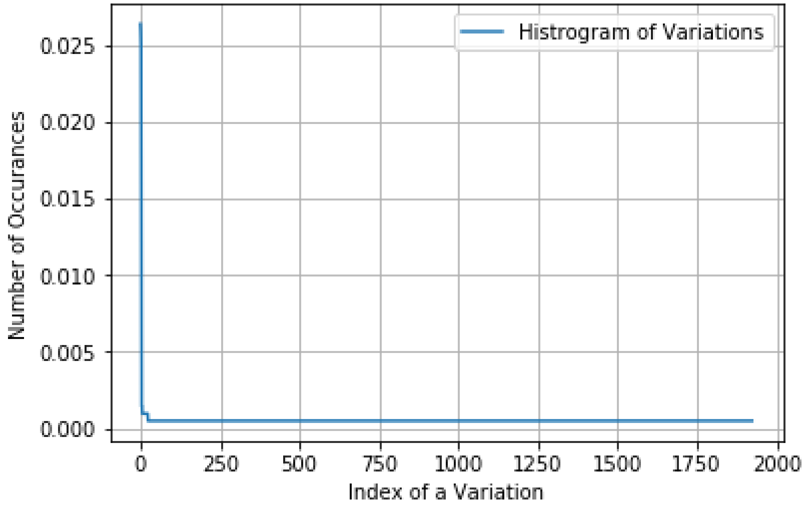
```
Number of Unique Variations : 1923
Truncating_Mutations    56
Amplification           55
Deletion                53
Fusions                 19
Q61H                     3
Overexpression           3
K117N                    2
G67R                     2
G12D                     2
```

```
S308A                         2
Name: Variation, dtype: int64
```

In [130]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categori
es of variations in the train data, and they are distibuted as follows"
,)
```

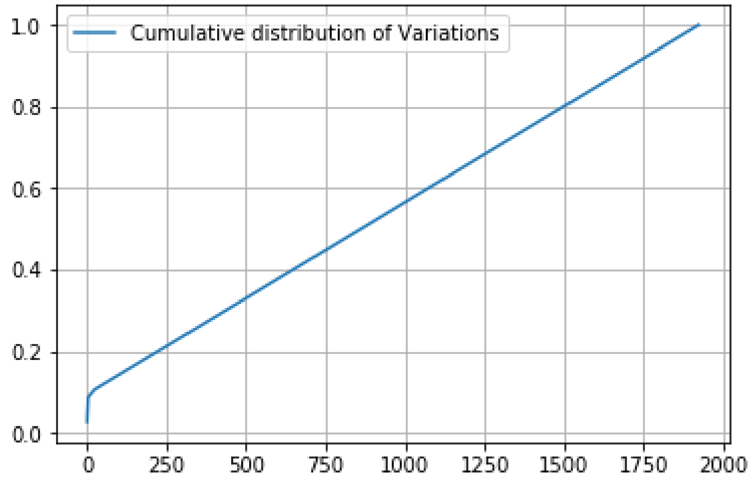
```
Ans: There are 1923 different categories of variations in the train dat
a, and they are distibuted as follows
```

In [131]:
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [132]:
```
c = np.cumsum(h)
#print(c)
plt.plot(c,label='Cumulative distribution of Variations')
```

```
plt.grid()
plt.legend()
plt.show()
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [133]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
 "Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
```

```
"Variation", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "V
ariation", cv_df))
```

In [134]:
```
print("train_variation_feature_responseCoding is a converted feature us
ing the response coding method. The shape of Variation feature:", train
_variation_feature_responseCoding.shape)
```

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

In [135]:
```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer()
# variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transfo
rm(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(te
st_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_d
f['Variation'])
```

In [136]:
```
print("train_variation_feature_onehotEncoded is converted feature using
 the onne-hot encoding method. The shape of Variation feature:", train_
variation_feature_onehotCoding.shape)
```

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

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

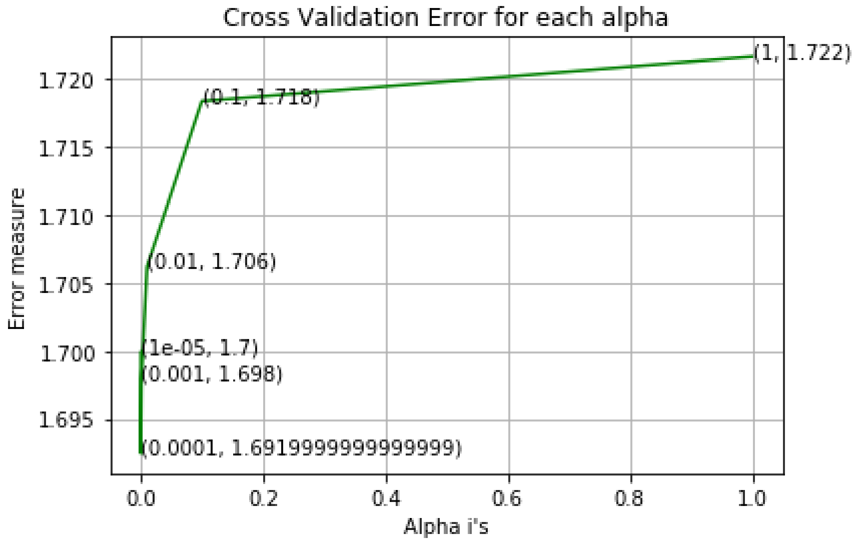
Let's build a model just like the earlier!

In [137]:
```
alpha = [10 ** x for x in range(-5, 1)]

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

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

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

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



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

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

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

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

```
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


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

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

```
For values of alpha =  1e-05 The log loss is: 1.69990475574
For values of alpha =  0.0001 The log loss is: 1.69244124476
For values of alpha =  0.001 The log loss is: 1.69794751396
For values of alpha =  0.01 The log loss is: 1.70617925488
For values of alpha =  0.1 The log loss is: 1.71837521039
For values of alpha =  1 The log loss is: 1.72163633164
```




```
For values of best alpha =  0.0001 The train log loss is: 0.74996071628
4
For values of best alpha =  0.0001 The cross validation log loss is: 1.
69244124476
For values of best alpha =  0.0001 The test log loss is: 1.71711277535
```

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

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

In [138]:
```
print("Q12. How many data points are covered by total ", unique_variati
ons.shape[0], " genes in test and cross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Vari
ation'])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'
])))].shape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0],
":",(test_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[
0],":" ,(cv_coverage/cv_df.shape[0])*100)
```

```
Q12. How many data points are covered by total  1923  genes in test and
cross validation data sets?
Ans
1. In test data 60 out of 665 : 9.022556390977442
2. In cross validation data 59 out of  532 : 11.090225563909774
```

3.2.3 Univariate Analysis on Text Feature

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

In [139]:
```
# cls_text is a data frame
# for every row in data fram consider the 'TEXT'
# split the words by space
# make a dict with those words
# increment its count whenever we see that word

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

In [140]:
```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
```

```
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(t
otal_dict.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_pr
ob/len(row['TEXT'].split()))
            row_index += 1
    return text_feature_responseCoding
```

Applying Tf-idf Vectorizer on text and selecting top 1000 words

In [141]:

```
text_vectorizer = TfidfVectorizer(max_features = 2000,min_df = 3,ngram_
range = (1,4))
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_d
f['TEXT'])
train_text_features= text_vectorizer.get_feature_names()
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_count
s))

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

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

In [142]:

```
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


total_dict = extract_dictionary_paddle(train_df)


confuse_array = []
```

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

In [143]:
```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)
```

In [144]:
```python
train_text_feature_responseCoding = (train_text_feature_responseCoding.
T/train_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/
test_text_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_t
ext_feature_responseCoding.sum(axis=1)).T
```

In [145]:
```python
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCo
ding, axis=0)

# we use the same vectorizer that was trained on train data
test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEX
T'])

# don't forget to normalize every feature
test_text_feature_onehotCoding = normalize(test_text_feature_onehotCodi
ng, axis=0)

# we use the same vectorizer that was trained on train data
cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])

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

In [146]:
```
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x:
 x[1] , reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

In [147]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({7.6272037311447018: 6, 11.440805596717052: 5, 6.33312512929955
58: 3, 6.0820774104945237: 3, 10.277326558129131: 2, 10.04515823125764
5: 2, 9.393871660449804: 2, 9.3849947064589596: 2, 8.0900659259204133:
2, 8.0612114257454408: 2, 7.822452809488734: 2, 7.4080513612140564: 2,
7.0403022221138514: 2, 6.996448344107729: 2, 6.3891675115886342: 2, 6.3
531199355077748: 2, 6.3486118396317845: 2, 5.7466209644373549: 2, 210.0
8857729536334: 1, 145.77421340117382: 1, 120.49461153687841: 1, 107.265
65251784898: 1, 104.63905951790812: 1, 97.169586733699248: 1, 97.012338
570003109: 1, 96.869242966832275: 1, 96.655541713764464: 1, 91.96961332
3286325: 1, 89.578150606369576: 1, 89.477950889985706: 1, 78.5586197493
52175: 1, 77.226495328104306: 1, 74.810442176475291: 1, 71.70329916676
329: 1, 66.325818790260428: 1, 65.810917044826368: 1, 65.45268483133818
9: 1, 64.379670181853157: 1, 63.835738973480737: 1, 63.763751652365507:
1, 61.193443806567025: 1, 57.415721795780684: 1, 56.69280198359332: 1,
56.396954453096406: 1, 56.05757341568102: 1, 55.127758497378053: 1, 53.
984236938275757: 1, 53.817390364804957: 1, 53.683174040231187: 1, 52.80
9226074141399: 1, 52.598141683020806: 1, 51.759637958771833: 1, 50.6157
09593364322: 1, 50.073736710433977: 1, 49.548610781316881: 1, 47.666429
460699014: 1, 47.023296799442917: 1, 46.678965136692838: 1, 46.48735859
7141338: 1, 45.635567258186143: 1, 44.068451672110335: 1, 43.4384316716
2867: 1, 43.27146920149108: 1, 42.117873005555602: 1, 41.71975469253103
3: 1, 40.883334709067604: 1, 38.893616991796733: 1, 38.514500702744478:
1, 38.09239200772493 9: 1, 37.555907718762676: 1, 37.446120311891789: 1,
37.216267172060256: 1, 37.193939515124768: 1, 36.199847024218087: 1, 3
6.195098861950804: 1, 35.788669469548879: 1, 35.750522840776831: 1, 35.
60464249318283: 1, 35.364590411771601: 1, 35.119048831478679: 1, 35.051
095759787302: 1, 34.844020442706935: 1, 34.795705299920463: 1, 34.68378
3619172033: 1, 34.666419943286655: 1, 34.50172093512235: 1, 34.20468920
0656297: 1, 34.062757247039826: 1, 33.772406305454211: 1, 33.6140435234
20641: 1, 33.512124828221936: 1, 33.460593007148951: 1, 32.437732674194
507: 1, 31.962931955785969: 1, 31.53035523482502: 1, 31.34348567838447
```

9: 1, 30.967124301093076: 1, 30.793538450920451: 1, 30.772971451873694:
1, 30.260194190532712: 1, 29.92945217533628: 1, 29.538856291220409: 1,
29.295706066501683: 1, 29.192093609843482: 1, 29.165988168330969: 1, 2
9.157777036852146: 1, 29.045891665010135: 1, 29.034682478881123: 1, 28.
969965207783311: 1, 28.841781194881179: 1, 28.737701578179145: 1, 28.51
0667539681833: 1, 28.462094053020689: 1, 27.796149577258895: 1, 27.3840
12417805959: 1, 27.289548985634259: 1, 27.068046556235942: 1, 27.016641
180232934: 1, 27.004684199587846: 1, 26.923436378017389: 1, 26.80782491
3827535: 1, 26.755896549670375: 1, 26.738695802147106: 1, 26.7087619825
01894: 1, 26.648473735911683: 1, 26.538248225241478: 1, 26.422027874457
545: 1, 26.410512068835416: 1, 26.408365907152589: 1, 26.25676051447643
3: 1, 26.162747223960402: 1, 25.768567148940747: 1, 25.700335840184536:
1, 25.693271729241829: 1, 25.398600150818933: 1, 25.328830890148954: 1,
25.247588319842158: 1, 25.109113366227607: 1, 24.84447651581273: 1, 24.
720485174821736: 1, 24.714025414795287: 1, 24.701761856812897: 1, 24.65
2858023162597: 1, 24.583564743604111: 1, 24.581936046811059: 1, 24.4594
43863494119: 1, 24.322272961342033: 1, 24.305223242070351: 1, 24.121009
35007366: 1, 24.089795642618657: 1, 23.787281786617438: 1, 23.709709758
024001: 1, 23.682198133905494: 1, 23.65728172249899: 1, 23.594934372757
94: 1, 23.552122354143922: 1, 23.491638109058105: 1, 23.46690048262209
8: 1, 23.347349505208534: 1, 23.272544800629131: 1, 23.223149783447457:
1, 23.168445469700345: 1, 23.00772114001666: 1, 22.892636450650645: 1,
22.859217511888268: 1, 22.852355162028893: 1, 22.530868975588685: 1, 2
2.492312104769024: 1, 22.309228768482729: 1, 22.268462255689279: 1, 22.
189474158361154: 1, 22.03698243265163: 1, 21.844460883930331: 1, 21.561
753818032162: 1, 21.536511934874486: 1, 21.318954131968184: 1, 21.27418
8899380068: 1, 21.241429886935244: 1, 21.147121628843351: 1, 21.1359795
48725111: 1, 21.069080773506062: 1, 20.985044758893881: 1, 20.978609299
895815: 1, 20.975101133520859: 1, 20.912420830504988: 1, 20.81386513186
5274: 1, 20.791153844175369: 1, 20.769995031075332: 1, 20.7347150095677
08: 1, 20.692851036217164: 1, 20.453088157075385: 1, 20.40258632053115:
1, 20.34264582800499: 1, 20.214096246716061: 1, 19.882258422837026: 1,
19.850511216533054: 1, 19.754938653757478: 1, 19.686899482267616: 1, 1
9.680042450158741: 1, 19.679688315310557: 1, 19.634393419236378: 1, 19.
587590269916891: 1, 19.580976722966923: 1, 19.546107929926574: 1, 19.45
8226614215111: 1, 19.405993402338009: 1, 19.375697955234816: 1, 19.2883
91068652906: 1, 19.243814522258045: 1, 19.21588650727794: 1, 19.1919117
04847644: 1, 19.171459646042155: 1, 19.074997273630071: 1, 19.037497010
139813: 1, 19.028535725075972: 1, 18.936947224210524: 1, 18.92167980832

1368: 1, 18.895693536375095: 1, 18.8536479575097: 1, 18.81163349783191:
1, 18.798994066980775: 1, 18.718894970792569: 1, 18.68383637113502З: 1,
18.64365909735119: 1, 18.643117298133859: 1, 18.63115300082648: 1, 18.6
22281424802097: 1, 18.598045603992098: 1, 18.544526041577598: 1, 18.491
694382137492: 1, 18.476446880903556: 1, 18.464614960215659: 1, 18.40579
0342121442: 1, 18.381763232455491: 1, 18.380085392200154: 1, 18.2589284
01440762: 1, 18.211933561222459: 1, 18.207296766685637: 1, 18.174136378
162206: 1, 18.169540418494108: 1, 18.142720856499345: 1, 18.11334466919
2305: 1, 18.109287099400486: 1, 18.077432110481823: 1, 17.9991041683406
4: 1, 17.9602005018974: 1, 17.940613622293899: 1, 17.876812880824136:
1, 17.847900760138284: 1, 17.810953806444964: 1, 17.807082280387004: 1,
17.77853645729472: 1, 17.760024146593832: 1, 17.748213728265426: 1, 17.
633295384967639: 1, 17.598576849521162: 1, 17.585529557245049: 1, 17.55
5108511202388: 1, 17.540493464869666: 1, 17.524700330918659: 1, 17.5101
36809827703: 1, 17.370802838461962: 1, 17.363499671214537: 1, 17.358833
552932175: 1, 17.346843416734803: 1, 17.328726463376938: 1, 17.25129330
5416866: 1, 17.215045412969644: 1, 17.208043394158974: 1, 17.1764832358
8412: 1, 17.165000418640915: 1, 17.11146011922175: 1, 17.08887398516576
2: 1, 16.994187031970995: 1, 16.903411287502212: 1, 16.880314970798533:
1, 16.873028504565454: 1, 16.808945136978689: 1, 16.795105214773464: 1,
16.774952508109699: 1, 16.75152473912247: 1, 16.746413525917585: 1, 16.
715655212968198: 1, 16.715546832692485: 1, 16.694198234286162: 1, 16.68
6393960094602: 1, 16.649257296333701: 1, 16.629551362162459: 1, 16.6122
49284588639: 1, 16.597529750654385: 1, 16.565155919931815: 1, 16.531131
254511692: 1, 16.452777829187589: 1, 16.444499714079381: 1, 16.44314112
3399993: 1, 16.436556638751803: 1, 16.393437485512724: 1, 16.3880735068
34701: 1, 16.38481510332387: 1, 16.294277440928465: 1, 16.2724439670673
36: 1, 16.219391485693205: 1, 16.212187480419558: 1, 16.17955868737651
3: 1, 16.16449481669801: 1, 16.147839811691217: 1, 16.142225443377178:
1, 16.138577479474456: 1, 16.114162781671663: 1, 16.096728908944829: 1,
16.066113737140526: 1, 16.0097682106131: 1, 15.947593695325276: 1, 15.9
37841598531275: 1, 15.869484920896046: 1, 15.834557244076931: 1, 15.804
486727641342: 1, 15.802457500996125: 1, 15.732477757681417: 1, 15.67715
5244137062: 1, 15.607489800720693: 1, 15.576228516035407: 1, 15.5187024
56382886: 1, 15.475689655292552: 1, 15.463002425039972: 1, 15.375039137
682332: 1, 15.371574566620518: 1, 15.366674380640676: 1, 15.32707098086
3677: 1, 15.316111396445736: 1, 15.307586809948026: 1, 15.2467127287205
58: 1, 15.22409956575677: 1, 15.1783463419395: 1, 15.163532280299407:
1, 15.15523734901876: 1, 15.145330413684082: 1, 15.123804432273145: 1,

15.062079528830596: 1, 14.951911038839061: 1, 14.949750748419396: 1, 1
4.928180950803807: 1, 14.920558884720077: 1, 14.871589324189209: 1, 14.
865377032555472: 1, 14.824584446408045: 1, 14.814359247939395: 1, 14.78
8878246944186: 1, 14.783752406865389: 1, 14.772846717880073: 1, 14.7408
7630892423: 1, 14.737461245853307: 1, 14.718093463565666: 1, 14.7048659
72213879: 1, 14.692253145006637: 1, 14.690271461710966: 1, 14.679656253
457651: 1, 14.663991518459152: 1, 14.659324622467899: 1, 14.63717378175
758: 1, 14.579041664800954: 1, 14.571024206932812: 1, 14.48746943493510
7: 1, 14.445450022651515: 1, 14.444247057652762: 1, 14.443325704272349:
1, 14.441784281400089: 1, 14.417627047316394: 1, 14.36902592619642: 1,
14.337945461610341: 1, 14.273089677127402: 1, 14.261108256844588: 1, 1
4.254278514443635: 1, 14.25098277922671: 1, 14.241453378078564: 1, 14.2
26852614760023: 1, 14.196704459754157: 1, 14.187950395134946: 1, 14.177
027805605951: 1, 14.158628685706386: 1, 14.156187623924284: 1, 14.04453
0857889386: 1, 14.032714235397377: 1, 14.008631115477709: 1, 13.9654474
07576212: 1, 13.956613620776757: 1, 13.956545390763965: 1, 13.919307449
103451: 1, 13.752334215268478: 1, 13.74053870542153: 1, 13.734509608658
63: 1, 13.727866422133609: 1, 13.721962760094186: 1, 13.69799205375373
3: 1, 13.671391072929822: 1, 13.662837743905907: 1, 13.636891124648203:
1, 13.63594379512474: 1, 13.625843527164655: 1, 13.618744190306691: 1,
13.617029925253217: 1, 13.560673736243459: 1, 13.540022868968807: 1, 1
3.534062072241159: 1, 13.513846033843906: 1, 13.476753845416116: 1, 13.
474359148817319: 1, 13.425202302256102: 1, 13.425050123390442: 1, 13.40
9369634909133: 1, 13.389108274997739: 1, 13.363584086066059: 1, 13.3576
1798500179: 1, 13.346160953500465: 1, 13.345291777363917: 1, 13.3430345
4814118: 1, 13.327316775934611: 1, 13.305433829476653: 1, 13.2611677416
91956: 1, 13.235223362536873: 1, 13.209182226377365: 1, 13.205294504890
116: 1, 13.199891524658399: 1, 13.173315637395865: 1, 13.16900293128273
3: 1, 13.151769762115887: 1, 13.140232532352485: 1, 13.129841346592798:
1, 13.126057411767418: 1, 13.106459948275184: 1, 13.074937857472763: 1,
13.071958371072052: 1, 13.026895165199578: 1, 13.02503415924947: 1, 12.
99166717548686: 1, 12.938410513203472: 1, 12.92438298967148: 1, 12.9051
44147610008: 1, 12.896664472077664: 1, 12.895191351022319: 1, 12.884418
631874517: 1, 12.84110830859662: 1, 12.837164974959801: 1, 12.827596901
642677: 1, 12.796853159334121: 1, 12.78484141360573 8: 1, 12.77570228952
2933: 1, 12.767908063403279: 1, 12.720763113121039: 1, 12.6946157347128
78: 1, 12.686690559304354: 1, 12.682393165952204: 1, 12.67313984680347
4: 1, 12.659419940672411: 1, 12.653430254355303: 1, 12.652573112862591:
1, 12.630607914881351: 1, 12.620694639220957: 1, 12.608567787890957: 1,

12.592446861576104: 1, 12.577038045912754: 1, 12.550169255495135: 1, 1
2.550157608743131: 1, 12.535223474393263: 1, 12.534116552052923: 1, 12.
494137165684887: 1, 12.457396903779541: 1, 12.449745486781756: 1, 12.44
3099698636455: 1, 12.435940797052174: 1, 12.418163602235413: 1, 12.4088
18287810906: 1, 12.398970750150928: 1, 12.397307886376453: 1, 12.386704
448811448: 1, 12.3803228125842: 1, 12.355277774333935: 1, 12.3180620368
78716: 1, 12.305443780772491: 1, 12.278693065916295: 1, 12.276937969449
92: 1, 12.276070457487545: 1, 12.262155922431605: 1, 12.23846268096399
5: 1, 12.204526118761592: 1, 12.192346802795983: 1, 12.161835368879892:
1, 12.131411801230984: 1, 12.091378648403953: 1, 12.086286033748777: 1,
12.084258342329845: 1, 12.078961317473871: 1, 12.071903478659928: 1, 1
2.057418113367998: 1, 12.05533908618345: 1, 12.046883788880706: 1, 12.0
23416794979086: 1, 12.019118412125241: 1, 11.999999412618001: 1, 11.996
35916516212: 1, 11.971864476707809: 1, 11.908231455271588: 1, 11.884159
359133214: 1, 11.869083876515901: 1, 11.86479626570738: 1, 11.864070494
936959: 1, 11.862671675821646: 1, 11.828011848274318: 1, 11.81259514904
7219: 1, 11.81259302220365: 1, 11.793693497424387: 1, 11.7872416925100
1: 1, 11.784992362843306: 1, 11.768770866535426: 1, 11.762876020368308:
1, 11.751314105643241: 1, 11.751157531389333: 1, 11.743780772835718: 1,
11.723394129087442: 1, 11.722320972634934: 1, 11.706598711484519: 1, 1
1.703859381027751: 1, 11.697894974834083: 1, 11.674362654335152: 1, 11.
673835454179333: 1, 11.644038224292638: 1, 11.63013748374601: 1, 11.623
616234540705: 1, 11.591233788266457: 1, 11.590515820782237: 1, 11.57883
3994916691: 1, 11.572528127078469: 1, 11.495105032433193: 1, 11.4944217
17164133: 1, 11.457942379218398: 1, 11.453104793873228: 1, 11.440125616
47474: 1, 11.433595773118903: 1, 11.418863791954964: 1, 11.393937753900
499: 1, 11.390992180536099: 1, 11.380883935384931: 1, 11.3676244569125
32: 1, 11.364845021487149: 1, 11.361669224846764: 1, 11.354826977319171:
1, 11.354566168773623: 1, 11.306523336550942: 1, 11.292384310748282: 1,
11.267259776631279: 1, 11.26048160243637: 1, 11.245185716939929: 1, 11.
220010115113066: 1, 11.203005006027281: 1, 11.173723672483408: 1, 11.16
2578629321553: 1, 11.146730777382: 1, 11.135385505141491: 1, 11.1176364
22914771: 1, 11.106898542127617: 1, 11.078960460108465: 1, 11.069350345
170699: 1, 11.05651868894884: 1, 11.053904832198347: 1, 11.025367438803
697: 1, 11.001627659288527: 1, 10.987992109452593: 1, 10.9631325458225
8: 1, 10.962616531531106: 1, 10.959334544161837: 1, 10.920607447639009:
1, 10.919880034575824: 1, 10.89959400038431: 1, 10.895180524685843: 1,
10.876722641463713: 1, 10.865107305499025: 1, 10.855218955011619: 1, 1
0.846478972568109: 1, 10.803323769073479: 1, 10.802220108319752: 1, 10.

79561623229816: 1, 10.782438783131781: 1, 10.777730971244695: 1, 10.773
171044315861: 1, 10.769431854191929: 1, 10.7598392987086: 1, 10.7442337
5446332: 1, 10.723058692185552: 1, 10.722269476367043: 1, 10.7175048370
85498: 1, 10.691087822612385: 1, 10.687551249122681: 1, 10.660713380429
655: 1, 10.655765323868611: 1, 10.647592502480435: 1, 10.6427368511976
2: 1, 10.639527828216861: 1, 10.639152703619851: 1, 10.630483186952958:
1, 10.62239503706375: 1, 10.616293556891756: 1, 10.593082923239024: 1,
10.579248930156609: 1, 10.56031156042129: 1, 10.556261895880569: 1, 10.
533910801508707: 1, 10.493818994577815: 1, 10.485760844226446: 1, 10.44
2716888342106: 1, 10.420567161735104: 1, 10.410204161305918: 1, 10.3955
74690523681: 1, 10.381391264331468: 1, 10.371071833069953: 1, 10.361530
138668426: 1, 10.351817429131428: 1, 10.339937369565712: 1, 10.32256003
6801102: 1, 10.317816957550832: 1, 10.312633474691728: 1, 10.3059220050
24552: 1, 10.294382424990673: 1, 10.290480359072435: 1, 10.278942787257
732: 1, 10.258374400709227: 1, 10.250276533321353: 1, 10.23647830890531
9: 1, 10.214860109655175: 1, 10.197181779383072: 1, 10.184878383375601:
1, 10.178578594957045: 1, 10.143292466547674: 1, 10.133764780437671: 1,
10.125990034599182: 1, 10.10767531676119: 1, 10.099404886318617: 1, 10.
097404875723383: 1, 10.062371760911661: 1, 10.05977400988932: 1, 10.045
912601682337: 1, 10.042045328895698: 1, 10.041900806849101: 1, 10.04049
4695679616: 1, 10.014441256326331: 1, 10.00294558816416: 1, 9.997098094
584814: 1, 9.9886970279839815: 1, 9.9870185181781945: 1, 9.959123831097
7328: 1, 9.9303501504544798: 1, 9.921787793906919: 1, 9.888487904536951
6: 1, 9.8762306912138538: 1, 9.843978227178118: 1, 9.8098790869447843:
1, 9.8032548106417732: 1, 9.7860134825453606: 1, 9.7666285930097256: 1,
9.740387114614709: 1, 9.7071540721902529: 1, 9.7068559850501455: 1, 9.6
810823879104966: 1, 9.6727041145913244: 1, 9.6685574659060194: 1, 9.655
1893396133686: 1, 9.650739040655683: 1, 9.6407393030004638: 1, 9.630217
5740910556: 1, 9.6284519360067815: 1, 9.6241410677179022: 1, 9.61193993
68742897: 1, 9.6020301108910697: 1, 9.6009867083632869: 1, 9.5855896930
859483: 1, 9.5682069296362151: 1, 9.5671240477896973: 1, 9.560533345479
9801: 1, 9.541874829909716: 1, 9.5343208216023925: 1, 9.525927260625074
7: 1, 9.5221753357260308: 1, 9.516969295589119: 1, 9.5016781036559088:
1, 9.5009525375346158: 1, 9.498951213559149: 1, 9.4693585475721544: 1,
9.453827715687293: 1, 9.4363534918185703: 1, 9.4237468724253297: 1, 9.4
13622357827105: 1, 9.4058558260678353: 1, 9.403357610705223: 1, 9.38863
85359163356: 1, 9.3841479694673726: 1, 9.3819207575362498: 1, 9.3765146
93157187: 1, 9.3754094000096515: 1, 9.3741409663237416: 1, 9.3632442606
646595: 1, 9.3479978460629312: 1, 9.3412328281639194: 1, 9.341225479785

7948: 1, 9.336510139668901: 1, 9.3344906978136688: 1, 9.318326820323383
3: 1, 9.3136744934459568: 1, 9.2873971050975985: 1, 9.2787895023807678:
1, 9.2752572751103841: 1, 9.2707066672011589: 1, 9.2668166937622765: 1,
9.2459735573931336: 1, 9.2435404598764741: 1, 9.2391013707578651: 1, 9.
2352226523741159: 1, 9.2297265718732735: 1, 9.2288643589922827: 1, 9.22
19787999239546: 1, 9.2212099992881349: 1, 9.2206708969509314: 1, 9.2052
959218033852: 1, 9.1974446165379824: 1, 9.1755800078983256: 1, 9.157956
8189133873: 1, 9.1532778488428566: 1, 9.153090612995971: 1, 9.111696155
4180751: 1, 9.1073768701293609: 1, 9.1070935007624598: 1, 9.10304726165
09961: 1, 9.1018327149611817: 1, 9.0847170938342678: 1, 9.0641918804307
622: 1, 9.053044199556: 1, 9.0510867024433974: 1, 9.0491311177974154:
1, 9.0432067799887381: 1, 9.0410727492225611: 1, 9.0319797953380494: 1,
9.0256114856000664: 1, 9.0182954282855619: 1, 9.0165723305949346: 1, 9.
009103916915203: 1, 9.0044231518040814: 1, 9.000385677802532: 1, 8.9993
172651231212: 1, 8.9970119440058234: 1, 8.9924431618684242: 1, 8.968019
2649862871: 1, 8.963972102896177: 1, 8.9526272920898453: 1, 8.929435379
4812373: 1, 8.9263917335428733: 1, 8.9179003283929639: 1, 8.91434147208
0439: 1, 8.9142550168021994: 1, 8.9124855148741027: 1, 8.90977128486204
61: 1, 8.9048794454873761: 1, 8.9040788338234016: 1, 8.901430399172683
7: 1, 8.9005101187363937: 1, 8.8987196938636917: 1, 8.8951112770285725:
1, 8.8754968703338157: 1, 8.8663319440215851: 1, 8.855847383692991: 1,
8.8396238415740118: 1, 8.8284798745995428: 1, 8.8276927863686119: 1, 8.
8242134392708014: 1, 8.8239358734318873: 1, 8.818354427181454: 1, 8.814
8156529840129: 1, 8.8066694236668042: 1, 8.7980989386115009: 1, 8.78031
79881987283: 1, 8.7787629294948992: 1, 8.7721826654150803: 1, 8.7516103
355416917: 1, 8.749534139649036: 1, 8.746370546456955: 1, 8.73703269693
72218: 1, 8.7285398500150961: 1, 8.7269447521317325: 1, 8.7193111228813
152: 1, 8.7187901161560148: 1, 8.7056376513005702: 1, 8.698481584351407
4: 1, 8.676707910014871: 1, 8.6709952333723823: 1, 8.6698423632947428:
1, 8.6436991940320596: 1, 8.639299167935512: 1, 8.6303296762851858: 1,
8.6286029881630046: 1, 8.6264979482415178: 1, 8.618847144208063: 1, 8.6
145582963113476: 1, 8.6134418960951944: 1, 8.6075910282163282: 1, 8.602
2313106913035: 1, 8.5964900888843214: 1, 8.5834293516715352: 1, 8.57715
22364012807: 1, 8.5567707623011149: 1, 8.5505754407829091: 1, 8.5360348
022936776: 1, 8.5044298865749006: 1, 8.4988303431812682: 1, 8.497120982
819391: 1, 8.4955280969744713: 1, 8.4902243878140506: 1, 8.476455617914
4197: 1, 8.4691819134414281: 1, 8.46550190192084: 1, 8.464614575170674
7: 1, 8.4606422318995804: 1, 8.4590959002881547: 1, 8.4492752323497875:
1, 8.4382537136362501: 1, 8.4364692143299909: 1, 8.4286851516942551: 1,

8.4211331412863846: 1, 8.4204946906756089: 1, 8.4119672898467925: 1, 8.
3926195525386316: 1, 8.3857385034374889: 1, 8.3823699798620126: 1, 8.38
19715366959322: 1, 8.3791626371426027: 1, 8.3715352930439675: 1, 8.3690
373969318053: 1, 8.3643281944094294: 1, 8.357433574126059: 1, 8.3561376
116112722: 1, 8.3480397501874961: 1, 8.3471155743349978: 1, 8.345629108
7118917: 1, 8.3439215444149397: 1, 8.3430912219912976: 1, 8.33932094776
9826: 1, 8.331499269106736: 1, 8.3190210760491734: 1, 8.308822123748171
9: 1, 8.3040598775985011: 1, 8.3007174113566915: 1, 8.29933293931572:
1, 8.2992758204313031: 1, 8.2919369266503189: 1, 8.2882693944271129: 1,
8.2880974565093819: 1, 8.2852482472251552: 1, 8.2827553943124332: 1, 8.
2793048761664636: 1, 8.2714068412564146: 1, 8.27041817157032: 1, 8.2701
365401229019: 1, 8.2620389337328213: 1, 8.255160178173897: 1, 8.2518416
981906064: 1, 8.2512486200888961: 1, 8.2341843029738921: 1, 8.225927036
2609254: 1, 8.222029905186643: 1, 8.2158291172163374: 1, 8.195896413172
6972: 1, 8.1916477817702784: 1, 8.1888884813061775: 1, 8.18571780603846
65: 1, 8.1806210616938184: 1, 8.1705177655626162: 1, 8.167265858730742
1: 1, 8.1641095817894218: 1, 8.1631002874997396: 1, 8.1589759608741126:
1, 8.1583607197874457: 1, 8.1535023039065386: 1, 8.153040439090872 8: 1,
8.1421409832026157: 1, 8.12853587303651: 1, 8.1248766679944744: 1, 8.11
79810321163188: 1, 8.1096752155924072: 1, 8.0941824191553096: 1, 8.0908
576780631982: 1, 8.0866919954939913: 1, 8.0806612830142157: 1, 8.073646
8489649251: 1, 8.0691754651893426: 1, 8.0666333803252925: 1, 8.05922921
48961935: 1, 8.0559678869361484: 1, 8.0506784423102449: 1, 8.0452304361
73707: 1, 8.0321026706548935: 1, 8.0292043646035207: 1, 8.0249519658131
145: 1, 8.0206685493020995: 1, 8.0125719723434159: 1, 8.003032882744019
4: 1, 8.0003202967382343: 1, 7.9986132851499461: 1, 7.9870784832885402:
1, 7.9810283149453634: 1, 7.97368804419171: 1, 7.9690636646433868: 1,
7.9632648588361059: 1, 7.9581171188283122: 1, 7.9550247406311971: 1, 7.
9473213057965859: 1, 7.9398750556758468: 1, 7.9348573849388666: 1, 7.93
05801604396571: 1, 7.9172048630588234: 1, 7.9084794062679817: 1, 7.9080
114869066049: 1, 7.8966164499396472: 1, 7.8914768038152463: 1, 7.885833
0894816824: 1, 7.884770782601052: 1, 7.8843008466824944: 1, 7.884052220
8631132: 1, 7.8800058694126687: 1, 7.8697610461619156: 1, 7.86894081103
03622: 1, 7.8612996003099926: 1, 7.8473105646922816: 1, 7.8364656968003
69: 1, 7.8283551249063583: 1, 7.8120222500443006: 1, 7.809484313649504:
1, 7.8041758872505476: 1, 7.8035195784699454: 1, 7.7986701958665803: 1,
7.7906242677123521: 1, 7.7812665593425585: 1, 7.7784339023490698: 1, 7.
773048658398741: 1, 7.7679043365154721: 1, 7.7667857435702672: 1, 7.762
6155163590571: 1, 7.7580219679964681: 1, 7.7552350593359431: 1, 7.74779

08390124401: 1, 7.7467261883184007: 1, 7.7434568552392387: 1, 7.7360060
821621692: 1, 7.7281549176310138: 1, 7.7269707510763848: 1, 7.721642403
4522957: 1, 7.7191852583332992: 1, 7.7132694766789625: 1, 7.70858303991
94899: 1, 7.6899401725496466: 1, 7.6890162517632801: 1, 7.6890114164263
572: 1, 7.6802910976850782: 1, 7.6797096250572494: 1, 7.677654310559083
8: 1, 7.6773405560317407: 1, 7.6692447577910681: 1, 7.6650786272089597:
1, 7.6556703153929631: 1, 7.6534849120290831: 1, 7.6509085828493983: 1,
7.6379097167562771: 1, 7.6304643730491941: 1, 7.6283759909819615: 1, 7.
6243614496389789: 1, 7.6220721040878718: 1, 7.6026678208176115: 1, 7.60
25562622441107: 1, 7.6005766205608047: 1, 7.5995499011639103: 1, 7.5958
328379934406: 1, 7.5938416185069837: 1, 7.5922876439948883: 1, 7.590257
1329621749: 1, 7.5761821373824771: 1, 7.5747640011954713: 1, 7.57228870
03121997: 1, 7.5679708396526468: 1, 7.5637011086114363: 1, 7.5605886081
848412: 1, 7.5598907602743282: 1, 7.5572392121277945: 1, 7.554330370948
2075: 1, 7.5427421377914579: 1, 7.5411394156227241: 1, 7.5410472070604939: 1, 7.5397211339451857: 1, 7.5333107619028912: 1, 7.527401843451618
1: 1, 7.5237876912366994: 1, 7.5162984100908661: 1, 7.5141459321490691:
1, 7.5088391113808139: 1, 7.5086166572338806: 1, 7.5083036598256001: 1,
7.5080544459389342: 1, 7.507485693371807: 1, 7.5012858517249361: 1, 7.4
872478559658457: 1, 7.483801105321537: 1, 7.4818416577387339: 1, 7.4807
022999483523: 1, 7.479790304507671: 1, 7.4794880159904551: 1, 7.4786703
350948018: 1, 7.4617728189843069: 1, 7.457550589141559: 1, 7.4509068086
188508: 1, 7.437332302515081: 1, 7.4366932148098863: 1, 7.4352679532250
381: 1, 7.4339361591783648: 1, 7.4282881278646: 1, 7.4269848044967386:
1, 7.4240169120028288: 1, 7.4120150983674327: 1, 7.4059129495808547: 1,
7.4046534103252606: 1, 7.4030662618068437: 1, 7.4002272236123599: 1, 7.
3984554445226225: 1, 7.3840445745160821: 1, 7.3819978732845417: 1, 7.38
18476893315177: 1, 7.3721725346832603: 1, 7.3716347039130898: 1, 7.3705
994540494357: 1, 7.3666651199451731: 1, 7.3649818340946975: 1, 7.360085
6356405382: 1, 7.3571132735490288: 1, 7.3514182091080382: 1, 7.34386804
97654723: 1, 7.3414543976600015: 1, 7.3392133472106691: 1, 7.3367195231
999096: 1, 7.3341364650208911: 1, 7.3268472032017229: 1, 7.325878656124
3889: 1, 7.3202860809549675: 1, 7.3185623763753531: 1, 7.31109537462630
23: 1, 7.3028314064449189: 1, 7.3003683780694351: 1, 7.297538623038494
9: 1, 7.2956658763488527: 1, 7.2920233009257514: 1, 7.28743983561069:
1, 7.2845941517011878: 1, 7.2806724439208379: 1, 7.267461468809655: 1,
7.2582017276540265: 1, 7.2482134502445481: 1, 7.2439460700295424: 1, 7.
2422845375070048: 1, 7.2390068231883662: 1, 7.2372475253853805: 1, 7.23
04085371050446: 1, 7.2114687740086465: 1, 7.2098085737806716: 1, 7.2082

581567389852: 1, 7.2038926702365078: 1, 7.2016830561154599: 1, 7.197605
1748297003: 1, 7.190550245965472: 1, 7.1864783308470868: 1, 7.184546947
466198: 1, 7.18013295579801: 1, 7.179231480736763: 1, 7.176516004882750
5: 1, 7.1732661193038405: 1, 7.1701919653104307: 1, 7.1595112619534058:
1, 7.1514861171823485: 1, 7.1489274786281021: 1, 7.145973542546141: 1,
7.1416856425301383: 1, 7.1310334301617351: 1, 7.1156407907031562: 1, 7.
1089682939406469: 1, 7.1059591614848117: 1, 7.1057037220094443: 1, 7.10
55250214860335: 1, 7.0816831820223491: 1, 7.0807114177133572: 1, 7.0794
617134806144: 1, 7.0625117074503132: 1, 7.0620765694067398: 1, 7.061213
7572670131: 1, 7.0576273519555368: 1, 7.055160992967263: 1, 7.052869218
347551: 1, 7.048047483053411: 1, 7.0441706577907972: 1, 7.0439889430136
74: 1, 7.0384842201268301: 1, 7.0383007774697672: 1, 7.027589156278970
1: 1, 7.0259526083059303: 1, 7.0063291508510712: 1, 6.9982359418016804:
1, 6.9977668298926519: 1, 6.9951846678377851: 1, 6.991275970556325: 1,
6.9863240977006615: 1, 6.9839455327544808: 1, 6.9811962953710101: 1, 6.
9773479224727799: 1, 6.9758755716690102: 1, 6.9651445254410094: 1, 6.96
18386612266212: 1, 6.9574910683964957: 1, 6.9529796078086505: 1, 6.9444
672845268167: 1, 6.9422807746702286: 1, 6.9401145380146625: 1, 6.935042
0017912029: 1, 6.9254328986671529: 1, 6.92432958267473: 1, 6.9234425229
403485: 1, 6.9230772403077427: 1, 6.9130740430223643: 1, 6.907145679471
5651: 1, 6.8977270811930964: 1, 6.8937738299620435: 1, 6.89131783607239
43: 1, 6.8823878485048438: 1, 6.8758609225962122: 1, 6.868713183475335
9: 1, 6.8660965810092067: 1, 6.8514125655342317: 1, 6.8457193272824926:
1, 6.8400209525558795: 1, 6.8382548979822717: 1, 6.8373800372713029: 1,
6.8371819099418785: 1, 6.8321529297732235: 1, 6.8255423067860796: 1, 6.
8197356443037638: 1, 6.8169816042915965: 1, 6.806048525593936: 1, 6.799
259683925003: 1, 6.7961780439725681: 1, 6.79040394879795: 1, 6.76902604
81669897: 1, 6.7631660368447593: 1, 6.7569784930592443: 1, 6.7501712090
776902: 1, 6.7411273297749936: 1, 6.7370241576260925: 1, 6.724600098286
4071: 1, 6.7239134610357265: 1, 6.720512855149499: 1, 6.71737455724309
3: 1, 6.7111302976605476: 1, 6.7084969365299232: 1, 6.707047145009704:
1, 6.7066282502933321: 1, 6.7019728627670334: 1, 6.6914051088598852: 1,
6.6857851177611067: 1, 6.6818374984302302: 1, 6.6799733637013805: 1, 6.
6773679128655319: 1, 6.6753538139849375: 1, 6.6742973495539282: 1, 6.67
05154617765752: 1, 6.6700314206941842: 1, 6.6685344134856219: 1, 6.6528
560507312982: 1, 6.6483443856946618: 1, 6.6367761129026253: 1, 6.635240
1736467703: 1, 6.6224597789536679: 1, 6.6182590896768145: 1, 6.61413624
77529313: 1, 6.6052477217666281: 1, 6.601176579046256: 1, 6.60030581988
30598: 1, 6.5987262772166364: 1, 6.5952093720313467: 1, 6.5946327885740

459: 1, 6.5935306624425802: 1, 6.5920394064693992: 1, 6.592008014747847
3: 1, 6.5848212136808915: 1, 6.5718877565869329: 1, 6.5635841492068003:
1, 6.5533114458920885: 1, 6.5504379609696466: 1, 6.5406276606437741: 1,
6.5181614965753187: 1, 6.5178584990950457: 1, 6.5150091666189773: 1, 6.
5109323319337946: 1, 6.506007512364806: 1, 6.5043289223857332: 1, 6.501
0542983712751: 1, 6.4885806953565561: 1, 6.4885527173671509: 1, 6.48736
33345518087: 1, 6.4862948058234347: 1, 6.4744108182870148: 1, 6.4729352
128105058: 1, 6.4678823646691184: 1, 6.4678650425612956: 1, 6.466174364
7162222: 1, 6.4569551340783748: 1, 6.4565421855286127: 1, 6.45582121095
46229: 1, 6.4554463253113319: 1, 6.451352562298481 5: 1, 6.4497566495647
778: 1, 6.4496347809738648: 1, 6.4491933098251648: 1, 6.447591152856730
9: 1, 6.4429556500460547: 1, 6.4401752189756296: 1, 6.4377920458941214:
1, 6.4365762492505958: 1, 6.4338272215870624: 1, 6.4245030154646061: 1,
6.4068291285475611: 1, 6.3980375220124053: 1, 6.3977688890945048: 1, 6.
3956470612656622: 1, 6.3943767827391937: 1, 6.3931956394183551: 1, 6.38
65582101885563: 1, 6.3827300940707774: 1, 6.382520829899093: 1, 6.37708
59794440442: 1, 6.3737525310144116: 1, 6.3723478595831731: 1, 6.3647780
983978857: 1, 6.3631463622502213: 1, 6.361810137480246: 1, 6.3603425243
938538: 1, 6.3549679470518718: 1, 6.3508026439186605: 1, 6.343624693588
4002: 1, 6.3434401720575293: 1, 6.3377563391639349: 1, 6.33717310390946
23: 1, 6.3325736445328884: 1, 6.3310277967039266: 1, 6.328586388368849
6: 1, 6.3282966822716658: 1, 6.3265301022473359: 1, 6.3237352536138465:
1, 6.3170449871910526: 1, 6.3023089781405544: 1, 6.3020587498687091: 1,
6.2986631561511883: 1, 6.2870658880051362: 1, 6.283941831654742: 1, 6.2
812962405434796: 1, 6.2750290215780797: 1, 6.274108731065299: 1, 6.2716
322334901085: 1, 6.2508026007074964: 1, 6.2415323530863933: 1, 6.240958
9595098742: 1, 6.2342042424355677: 1, 6.2300811796487654: 1, 6.22978538
53711: 1, 6.2288362377454503: 1, 6.227400312380964: 1, 6.21768960834744
18: 1, 6.2169747543412024: 1, 6.2069101358508751: 1, 6.197967853096626
5: 1, 6.1974629200327751: 1, 6.1947130500050616: 1, 6.1881051707306165:
1, 6.1841782460056134: 1, 6.1754750893032435: 1, 6.175449273209451: 1,
6.1742075304581752: 1, 6.1738079989416175: 1, 6.1713652543986806: 1, 6.
171287852230904: 1, 6.1630249684197658: 1, 6.1628363378189857: 1, 6.160
2620833815047: 1, 6.1587743229260052: 1, 6.1585579128888961: 1, 6.15122
3660730305: 1, 6.1481083541949619: 1, 6.1460059673640028: 1, 6.14587728
13072287: 1, 6.1451464933826863: 1, 6.1450281687976247: 1, 6.1404144084
998462: 1, 6.1310433161945683: 1, 6.1285064056230221: 1, 6.119740198152
682: 1, 6.1154222281662127: 1, 6.110727728241736: 1, 6.101374255037932
8: 1, 6.0915817113812354: 1, 6.0902888039195942: 1, 6.0715742336902494:

1, 6.069789315073205: 1, 6.0663447987085171: 1, 6.063300425511513: 1,
6.0615479459722863: 1, 6.0601165792322487: 1, 6.0564603230515512: 1, 6.
0559237001677308: 1, 6.0557723193350386: 1, 6.0556497693477098: 1, 6.04
77616781066539: 1, 6.0477380596221586: 1, 6.0362334177639578: 1, 6.0357
174179571906: 1, 6.0340269107308009: 1, 6.020948639665316: 1, 6.0193742
88400722: 1, 6.0192442485011037: 1, 6.0178272184459445: 1, 6.0174979418
519339: 1, 6.0132820343269238: 1, 6.0095504960420181: 1, 6.007608118567
858: 1, 6.0067679216022869: 1, 6.004882331368484: 1, 5.999827847859841
7: 1, 5.9981390084367501: 1, 5.9947465540774258: 1, 5.9862069426412434:
1, 5.9848825173727924: 1, 5.9725671636398019: 1, 5.9695335208896578: 1,
5.9681948872706592: 1, 5.967849186092125: 1, 5.9657586125617614: 1, 5.9
638951459482925: 1, 5.9595090246841584: 1, 5.9548941832082765: 1, 5.952
8006665333786: 1, 5.9521338623659918: 1, 5.9520441381961025: 1, 5.94630
75034169766: 1, 5.9309427872558045: 1, 5.9237764351522157: 1, 5.9144102
852619254: 1, 5.9111096841525557: 1, 5.9110586059574191: 1, 5.908256308
3226134: 1, 5.9055341122548954: 1, 5.893480080063938: 1, 5.891996029265
3757: 1, 5.8908229678396378: 1, 5.8900872332248548: 1, 5.88110978008051
52: 1, 5.8804673957500375: 1, 5.8796741176538587: 1, 5.873697167772148
1: 1, 5.8645046448153222: 1, 5.8604469334115059: 1, 5.8543268423754737:
1, 5.8518578352893345: 1, 5.8517836595701009: 1, 5.8516528085284474: 1,
5.8462629240854902: 1, 5.8462006442102261: 1, 5.8442787197444162: 1, 5.
8431432508680334: 1, 5.838830697508187: 1, 5.8327204054772421: 1, 5.830
9923387253839: 1, 5.8253714656739417: 1, 5.8210142925818626: 1, 5.81898
36918040854: 1, 5.8111399779688329: 1, 5.8101360149632431: 1, 5.7976086
431370026: 1, 5.795856752683604: 1, 5.7927424979502016: 1, 5.7925751802
17284: 1, 5.7890462460676808: 1, 5.7830839316512277: 1, 5.7775410776405
778: 1, 5.7769700304183607: 1, 5.7707898217824178: 1, 5.770156877611054
1: 1, 5.768592919120052: 1, 5.7624910060902819: 1, 5.755238260359322:
1, 5.7546272313037443: 1, 5.7418686658264306: 1, 5.7341016595590695: 1,
5.7334931854010733: 1, 5.731721520449911: 1, 5.7298857941312802: 1, 5.7
171798678889161: 1, 5.708986578373449: 1, 5.7060778775526888: 1, 5.7024
678812025913: 1, 5.7015690291814725: 1, 5.7002493182418261: 1, 5.699972
9023664676: 1, 5.6950287652351959: 1, 5.6933066033406456: 1, 5.68866310
2756828: 1, 5.6808728344959487: 1, 5.6804747898648777: 1, 5.67626345180
67356: 1, 5.6731845734431623: 1, 5.6668934329863623: 1, 5.6586101207765
962: 1, 5.6586041723713683: 1, 5.6576993695385189: 1, 5.65389453231017
3: 1, 5.6534812458180133: 1, 5.6528928456606451: 1, 5.6499117414446074:
1, 5.647189098951567: 1, 5.6450244063684183: 1, 5.6414699475339782: 1,
5.6394015976961063: 1, 5.6389459050274144: 1, 5.6354736603969897: 1, 5.

6326835394240993: 1, 5.6298476055896813: 1, 5.6297893738909277: 1, 5.62
61358312331895: 1, 5.6166866426053144: 1, 5.6144390360272185: 1, 5.6123
673275284824: 1, 5.6064960517981053: 1, 5.6062605550355356: 1, 5.600258
9295098648: 1, 5.5940642998589833: 1, 5.5828768327799043: 1, 5.58202148
7427788: 1, 5.5788561406020181: 1, 5.5763041970640694: 1, 5.57550803718
12857: 1, 5.5735702872383577: 1, 5.5734112865855687: 1, 5.5715817997200
592: 1, 5.5710670167384926: 1, 5.5687723164976157: 1, 5.56560598986666
3: 1, 5.5604888143927589: 1, 5.5582676668081943: 1, 5.5573368522945952:
1, 5.5502233825133107: 1, 5.5459583921728397: 1, 5.5437428411373091: 1,
5.5426957736552032: 1, 5.5418837667478824: 1, 5.5406222320478413: 1, 5.
5377583227971519: 1, 5.5316713647302524: 1, 5.5269042606558552: 1, 5.52
57994432625059: 1, 5.5200825036490615: 1, 5.5171422695102885: 1, 5.5008
64738616758: 1, 5.4960698425431751: 1, 5.4886332657227994: 1, 5.4878696
151213422: 1, 5.4811312689224518: 1, 5.4741950908704498: 1, 5.474101970
9370065: 1, 5.4579425843737095: 1, 5.455275393186513: 1, 5.443318486481
0746: 1, 5.4410523418227674: 1, 5.440432546938025: 1, 5.42848127145806
9: 1, 5.427701763101382: 1, 5.4253503236589617: 1, 5.4217622997284316:
1, 5.4213581051592996: 1, 5.4163553841720491: 1, 5.4094501715158865: 1,
5.4080117598533448: 1, 5.4056799065634031: 1, 5.4035278107599458: 1, 5.
4003803450015644: 1, 5.3982631026843739: 1, 5.3971858170712466: 1, 5.39
6785484516232: 1, 5.3905963474362757: 1, 5.3892243569630773: 1, 5.38715
85690311443: 1, 5.3835985751996098: 1, 5.3823934646783735: 1, 5.3775125
831934689: 1, 5.374251685044829: 1, 5.3699682884145599: 1, 5.3676128379
14181: 1, 5.3664493781746181: 1, 5.36478088397355: 1, 5.360357103073245
5: 1, 5.3547838826400103: 1, 5.3530059171303774: 1, 5.3527705810400938:
1, 5.351060992791286: 1, 5.3486948240697254: 1, 5.3486201750143652: 1,
5.3444053075231848: 1, 5.3432643833983917: 1, 5.3417648936795628: 1, 5.
3326327028311269: 1, 5.3319134806876152: 1, 5.3245627551653021: 1, 5.32
17210734172014: 1, 5.3203263001905299: 1, 5.3171027095833017: 1, 5.3120
748902026245: 1, 5.3106217101631286: 1, 5.3106043812053496: 1, 5.303313
0688455516: 1, 5.3033024548833465: 1, 5.303163939144083: 1, 5.300376759
649307: 1, 5.2956250141463768: 1, 5.2935198195346214: 1, 5.292570298762
6906: 1, 5.2898479052424516: 1, 5.2888080690405239: 1, 5.28849598167541
49: 1, 5.2879376069177075: 1, 5.2799759622340678: 1, 5.279295884430760
9: 1, 5.2786787199463401: 1, 5.2769960474130855: 1, 5.275860051832753:
1, 5.2737444080117442: 1, 5.2630278844637619: 1, 5.2585216337626344: 1,
5.2582050443164841: 1, 5.2565521945023725: 1, 5.2558100978419127: 1, 5.
2556316155290546: 1, 5.2551858870498513: 1, 5.2495068333094537: 1, 5.24
81662231936106: 1, 5.2475944944395039: 1, 5.244501271898983: 1, 5.24421

70324704298: 1, 5.2326278661262204: 1, 5.2297982369208142: 1, 5.2288300
358708764: 1, 5.2257352779412516: 1, 5.2206269357539634: 1, 5.219303325
3767522: 1, 5.2189926072843784: 1, 5.2187622869197652: 1, 5.21828443115
16137: 1, 5.2132130973530959: 1, 5.2099484044014845: 1, 5.1828747990250
088: 1, 5.1812014607538543: 1, 5.1770898429666401: 1, 5.176856407634542
9: 1, 5.1765675245304958: 1, 5.1744936394430754: 1, 5.1738371450059999:
1, 5.171882633720279: 1, 5.1658272051109622: 1, 5.1582287023912983: 1,
5.1552511610458351: 1, 5.1525818486034005: 1, 5.1479081176844694: 1, 5.
1478792828252491: 1, 5.147299554877713: 1, 5.1471773881799798: 1, 5.146
0189433482348: 1, 5.1450740566042032: 1, 5.1421293008460944: 1, 5.13885
535435357: 1, 5.1387563051249909: 1, 5.1384484131828421: 1, 5.138002035
9614527: 1, 5.1375027819365728: 1, 5.1367803284301417: 1, 5.13584889125
44233: 1, 5.1342493808100214: 1, 5.1205052320922109: 1, 5.1198859845805
709: 1, 5.1183153242915687: 1, 5.115123576054148: 1, 5.113374416996127
7: 1, 5.1110530386146129: 1, 5.1046708959667972: 1, 5.1038789087676042:
1, 5.1005791116213182: 1, 5.0991402721118675: 1, 5.0967168068663558: 1,
5.096164431234345: 1, 5.0886326027389117: 1, 5.0885820084967071: 1, 5.0
863727569140975: 1, 5.0862666284379303: 1, 5.0858951763189317: 1, 5.084
3146828265384: 1, 5.0832831608411899: 1, 5.0800789567318336: 1, 5.07938
86312427139: 1, 5.078971754603983: 1, 5.075337936406803: 1, 5.071740830
0603184: 1, 5.0696588509589784: 1, 5.0696329340139874: 1, 5.06634645410
99215: 1, 5.0660537385092903: 1, 5.0659725549462582: 1, 5.0658899390336
822: 1, 5.0640509292572915: 1, 5.0630439431998449: 1, 5.059401541430811
6: 1, 5.0556960544793501: 1, 5.0543120093853799: 1, 5.0516910404172508:
1, 5.0470762497471826: 1, 5.0445817908555881: 1, 5.0419997475687426: 1,
5.0408113217299828: 1, 5.0396472367725353: 1, 5.0358345878723112: 1, 5.
0356318881104087: 1, 5.0283032714831721: 1, 5.0278365295741843: 1, 5.02
76645163838909: 1, 5.0238076740689968: 1, 5.0211960772045785: 1, 5.0209
115055909468: 1, 5.0101813172354381: 1, 5.0086892240637155: 1, 5.007592
8666182792: 1, 5.0060153492128885: 1, 5.0041244904506295: 1, 5.00306144
38087371: 1, 5.0013881682469936: 1, 4.999837556938532: 1, 4.99896282309
06034: 1, 4.9988790774476088: 1, 4.992058362036512: 1, 4.98380408492607
16: 1, 4.9826679092792192: 1, 4.9801747032218344: 1, 4.973802793745844
2: 1, 4.9725628548660925: 1, 4.9724785375614804: 1, 4.9681233473354141:
1, 4.9650746516039126: 1, 4.964249934986138: 1, 4.9641694911177225: 1,
4.9624917251623941: 1, 4.9623943046651915: 1, 4.9621106224051728: 1, 4.
9580116069964255: 1, 4.9575282199070738: 1, 4.9564285216562523: 1, 4.95
20175871197916: 1, 4.9441495661435795: 1, 4.9400302777355947: 1, 4.9398
08333179478: 1, 4.9391368596591869: 1, 4.9378813351881554: 1, 4.9300856

165738631: 1, 4.929924178916651: 1, 4.9271752068634171: 1, 4.9244902309
721699: 1, 4.9234996914518359: 1, 4.9097270750961135: 1, 4.905926159118
7804: 1, 4.9047731606363918: 1, 4.8955676592500765: 1, 4.89115199483722
48: 1, 4.8900292210227869: 1, 4.8888098054040752: 1, 4.882171650173885:
1, 4.8821310402536886: 1, 4.8803838045792967: 1, 4.8742472253534581: 1,
4.8715492568980601: 1, 4.8662006096208223: 1, 4.8638859260103224: 1, 4.
8620802380008881: 1, 4.8619128429029557: 1, 4.8514694806700289: 1, 4.85
0116047352687: 1, 4.8499974141295601: 1, 4.8452455628973068: 1, 4.83855
05966963978: 1, 4.824932828515677: 1, 4.8225797033855455: 1, 4.82229236
52410206: 1, 4.8173196229773181: 1, 4.812793182668333: 1, 4.81243463847
81802: 1, 4.8058281515896812: 1, 4.805565264517929: 1, 4.80488635079036
1: 1, 4.8019676351674141: 1, 4.8009858340356564: 1, 4.7978910873834115:
1, 4.7915273588865306: 1, 4.7909811578390844: 1, 4.7906718547414755: 1,
4.7876465724497255: 1, 4.7829764278975553: 1, 4.7810534104881617: 1, 4.
7789519153356022: 1, 4.770037770309826: 1, 4.7697747610372172: 1, 4.767
618404549693: 1, 4.7650171952976432: 1, 4.7586015024144093: 1, 4.756257
4492384462: 1, 4.7552931062213428: 1, 4.7505677955728576: 1, 4.75047085
11993643: 1, 4.7446862312049136: 1, 4.7432688999442254: 1, 4.7421679223
481696: 1, 4.7360647077075706: 1, 4.7340828681759568: 1, 4.732845673489
5639: 1, 4.732699868716197: 1, 4.7226076794586884: 1, 4.721302467046617
9: 1, 4.7190925216196646: 1, 4.7158331384038537: 1, 4.7146182636774503:
1, 4.7145802123074398: 1, 4.7141553651797761: 1, 4.7131095575009097: 1,
4.7116209942664602: 1, 4.7109863096894422: 1, 4.7080552096676156: 1, 4.
7070087012934687: 1, 4.7031702219222513: 1, 4.6989133420007363: 1, 4.69
80502024004451: 1, 4.6972616372434564: 1, 4.6953086264768586: 1, 4.6945
717723972358: 1, 4.688859111101042: 1, 4.6886148482787116: 1, 4.6834684
109102431: 1, 4.6823715356325: 1, 4.6738124079724619: 1, 4.673147643968
0932: 1, 4.6721875776781268: 1, 4.6715733976702349: 1, 4.66801606242400
61: 1, 4.6646484021405685: 1, 4.6567758389372216: 1, 4.651440792577468
6: 1, 4.6512209918725134: 1, 4.6496301892359915: 1, 4.6438285313707937:
1, 4.6431461396363609: 1, 4.6413596326968118: 1, 4.6378601023679691: 1,
4.6305421199861323: 1, 4.6300615312141211: 1, 4.6293401297110925: 1, 4.
6270484254817958: 1, 4.6223021012097023: 1, 4.6212198291620128: 1, 4.61
90287592546317: 1, 4.6187741916238494: 1, 4.6149506076158948: 1, 4.6100
558635879292: 1, 4.6083145069157743: 1, 4.6063457413961633: 1, 4.605007
9273391766: 1, 4.6008587048828238: 1, 4.5946590016258311: 1, 4.58184613
72134722: 1, 4.5598299810039959: 1, 4.5585429829072233: 1, 4.5567836824
619805: 1, 4.5515668857160705: 1, 4.5503093819027693: 1, 4.549043584232
1411: 1, 4.5436083763196269: 1, 4.5402354536724925: 1, 4.53846278617614

64: 1, 4.5355772547457729: 1, 4.5315869268238576: 1, 4.523946540856060
8: 1, 4.5238393566153654: 1, 4.5225960152179692: 1, 4.5199620391268702:
1, 4.5199390700533106: 1, 4.510698753700817: 1, 4.5097547001997436: 1,
4.5089070839044103: 1, 4.5086638917309587: 1, 4.5069684276572151: 1, 4.
5023063296164665: 1, 4.5007344560997895: 1, 4.4980294954166542: 1, 4.49
46413542991301: 1, 4.4866924621391879: 1, 4.4845374210869959: 1, 4.4782
775842286346: 1, 4.4782648515560677: 1, 4.4771042834891128: 1, 4.472770
9380478622: 1, 4.4666473655805898: 1, 4.4664261325663466: 1, 4.46205044
14268716: 1, 4.4606885020610276: 1, 4.4599302891874872: 1, 4.4534494871
106629: 1, 4.4534477987511965: 1, 4.4446965527298481: 1, 4.431967479654
7828: 1, 4.4313606586117125: 1, 4.4205372924358066: 1, 4.41891308044744
5: 1, 4.41231408444047: 1, 4.4122467460401449: 1, 4.4090273866921565:
1, 4.4067312449433427: 1, 4.3991841764471342: 1, 4.3912895675825352: 1,
4.3887979180505488: 1, 4.3851912955118344: 1, 4.3814160218219236: 1, 4.
381040869922014: 1, 4.3786506821589031: 1, 4.3767183009405652: 1, 4.366
9048504934587: 1, 4.3667743881541847: 1, 4.3657763054367873: 1, 4.35889
59195307044: 1, 4.3556503665375006: 1, 4.354765224454308: 1, 4.34944019
59578601: 1, 4.3449753641665136: 1, 4.3383239991201314: 1, 4.3321006941
952946: 1, 4.3277654347823225: 1, 4.3258470776310993: 1, 4.324182815680
8637: 1, 4.3195821090400734: 1, 4.3152820496517723: 1, 4.31284194305028
82: 1, 4.307087965637928: 1, 4.3066434823696662: 1, 4.3064476873939164:
1, 4.2979714882440625: 1, 4.2934257623544667: 1, 4.2897475334024193: 1,
4.2886878029269715: 1, 4.2881904077054234: 1, 4.281987968612655: 1, 4.2
698126985144675: 1, 4.2641520338504391: 1, 4.2636727774050911: 1, 4.263
1852920275914: 1, 4.2592597098092195: 1, 4.2542236364225667: 1, 4.25225
62787511449: 1, 4.2514703016482898: 1, 4.2509418826133363: 1, 4.2476679
753109261: 1, 4.2433452952627837: 1, 4.2430836649661892: 1, 4.241603474
3509794: 1, 4.2409353917891703: 1, 4.2352082969510061: 1, 4.23439544678
06659: 1, 4.2304150869596651: 1, 4.2300740073638412: 1, 4.2260564295945
198: 1, 4.2246052241371723: 1, 4.2052067948907697: 1, 4.202574325086536
6: 1, 4.1997448622058178: 1, 4.1996072317319033: 1, 4.1981635340511643:
1, 4.1890758363296499: 1, 4.1890220724207978: 1, 4.1843180841586269: 1,
4.1835587646571248: 1, 4.1727029450400934: 1, 4.1609198718157341: 1, 4.
1597672486019519: 1, 4.1590949491614113: 1, 4.1508275186228936: 1, 4.14
82611067525275: 1, 4.1478284862922061: 1, 4.1385029628066903: 1, 4.1358
847738075415: 1, 4.1341724775704689: 1, 4.1328715355026588: 1, 4.129450
2388066432: 1, 4.1292695355946787: 1, 4.1252530092775919: 1, 4.11304915
41155418: 1, 4.1087451336179708: 1, 4.0973303181858833: 1, 4.0770303407
291744: 1, 4.0704112134817239: 1, 4.0639298765302216: 1, 4.059321433141

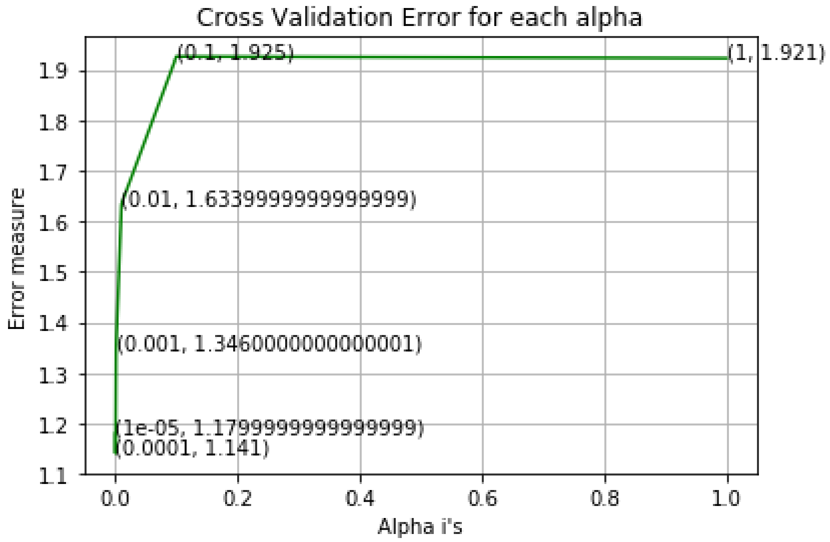
```
1573: 1, 4.045040332373115: 1, 4.0406078281620603: 1, 4.03759490636274
7: 1, 4.0335115717257697: 1, 4.0319981014049606: 1, 4.0286602495546484:
1, 4.0179246453490771: 1, 4.0111587810127309: 1, 4.0036123574122975: 1,
3.9917143931387193: 1, 3.9898735877897491: 1, 3.9884515554306903: 1, 3.
985841177578489: 1, 3.9835521296281127: 1, 3.9793005961060692: 1, 3.961
2030350416574: 1, 3.9600999186842989: 1, 3.9525833753477801: 1, 3.95083
60916111718: 1, 3.947233023942418: 1, 3.9336762716768785: 1, 3.93079304
2122398: 1, 3.91919465500037: 1, 3.910280212333634: 1, 3.90934116539081
39: 1, 3.9054791462210967: 1, 3.9011224509723674: 1, 3.897889680987024
7: 1, 3.8757255692742878: 1, 3.8756936050983146: 1, 3.8730592142910978:
1, 3.8674232241213278: 1, 3.8651099967099856: 1, 3.8600641195522374: 1,
3.8341389550453724: 1, 3.8241552557441687: 1, 3.8239688986016711: 1, 3.
8108680413213101: 1, 3.806464974936195: 1, 3.8045798657234697: 1, 3.799
4311376362786: 1, 3.7950379419393134: 1, 3.7941066131193: 1, 3.78133830
65595652: 1, 3.7787239725771387: 1, 3.7726052483491568: 1, 3.7679443810
374829: 1, 3.7405487125863259: 1, 3.7334272415535796: 1, 3.728534788989
0708: 1, 3.681841213007182: 1, 3.6800013578049557: 1, 3.667984857520131
5: 1, 3.599823505824594: 1, 3.599019807913403: 1, 3.5721674414056732:
1, 3.5037717913155033: 1, 3.4472123250395814: 1, 3.0486671972586645:
1})
```

In [148]:

```
# Train a Logistic regression+Calibration model using text features whi
cha re on-hot encoded
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.clas
ses_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv
, predict_y, labels=clf.classes_, eps=1e-15))
```

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.18026262047
For values of alpha =  0.0001 The log loss is: 1.14091090865
For values of alpha =  0.001 The log loss is: 1.34639600593
For values of alpha =  0.01 The log loss is: 1.63376856285
For values of alpha =  0.1 The log loss is: 1.92486427781
For values of alpha =  1 The log loss is: 1.92142957417
```




```
For values of best alpha =  0.0001 The train log loss is: 0.74441796837
1
For values of best alpha =  0.0001 The cross validation log loss is: 1.
14091090865
For values of best alpha =  0.0001 The test log loss is: 1.10009576811
```

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

Ans. Yes, it seems like!

In [149]:

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

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

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

In [150]:
```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in
 train data")
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appe
ared in train data")
```

```
5.916 % of word of test data appeared in train data
6.772 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [151]:
```
def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y,
clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    pred_y = sig_clf.predict(test_x)

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

In [152]:
```
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

In [153]:
```
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
#     text_count_vec = CountVectorizer(min_df=3)
#     gene_count_vec = TfidfVectorizer()
#     var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3,max_features = 3000)

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

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

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

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

Stacking the three types of features

In [154]:
```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding))

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

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

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


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

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

```
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_fe
ature_responseCoding))
```

In [155]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ",
train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", t
est_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation
 data =", cv_x_onehotCoding.shape)
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 41
90)
(number of data points * number of features) in test data =  (665, 419
0)
(number of data points * number of features) in cross validation data =
(532, 4190)
```

In [156]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ",
train_x_responseCoding.shape)
print("(number of data points * number of features) in test data = ", t
est_x_responseCoding.shape)
print("(number of data points * number of features) in cross validation
 data =", cv_x_responseCoding.shape)
```

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 2
7)
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data =
(532, 27)
```

4.1. Base Line Model

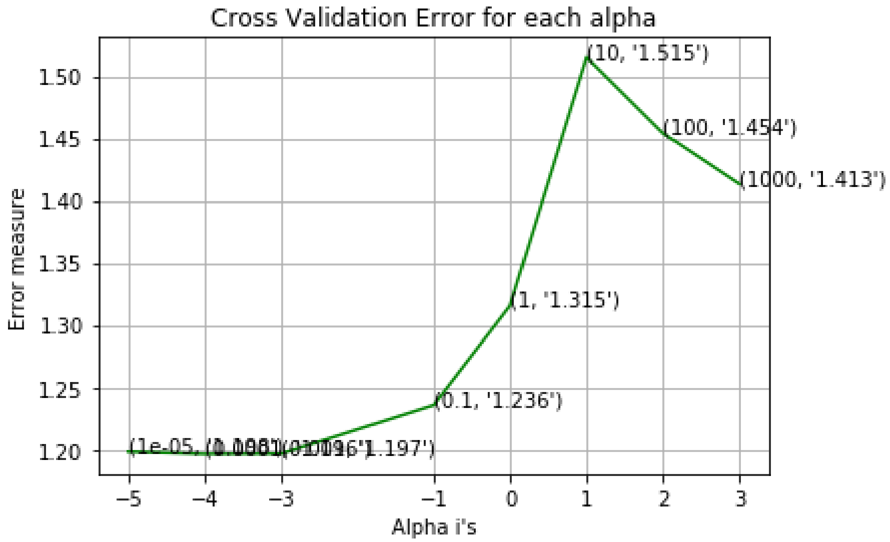
4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [157]:
```
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_a
rray[i]))
plt.grid()
plt.xticks(np.log10(alpha))
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


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

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

```
for alpha = 1e-05
Log Loss : 1.19825829735
for alpha = 0.0001
Log Loss : 1.1963818424
for alpha = 0.001
Log Loss : 1.19660810764
for alpha = 0.1
Log Loss : 1.23559332485
for alpha = 1
Log Loss : 1.31542236211
for alpha = 10
Log Loss : 1.51509520151
for alpha = 100
Log Loss : 1.45415825156
for alpha = 1000
Log Loss : 1.41330933624
```



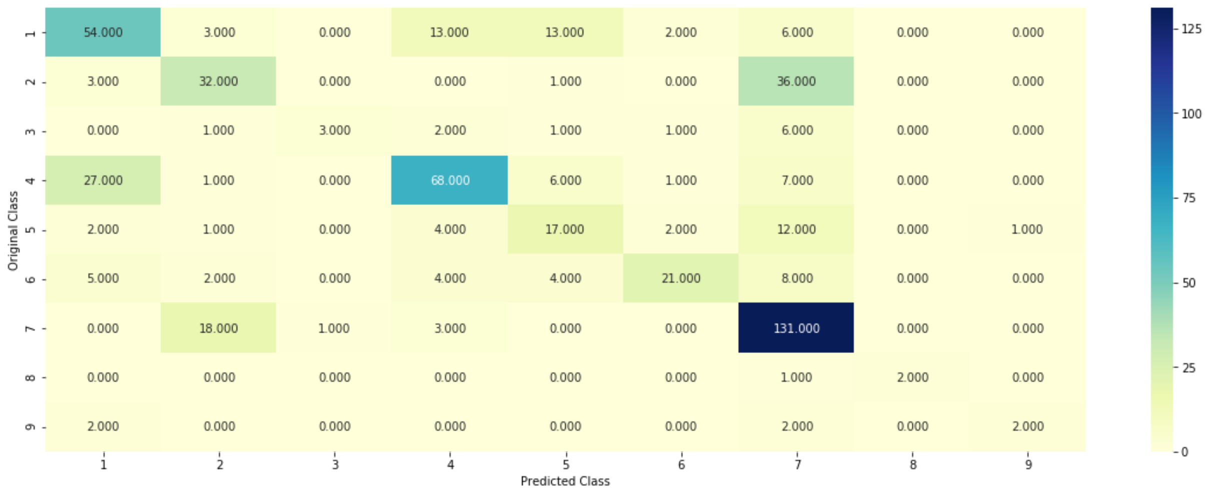

```
For values of best alpha =  0.0001 The train log loss is: 0.56020156213
1
For values of best alpha =  0.0001 The cross validation log loss is: 1.
1963818424
For values of best alpha =  0.0001 The test log loss is: 1.20901211339
```

4.1.1.2. Testing the model with best hyper paramters

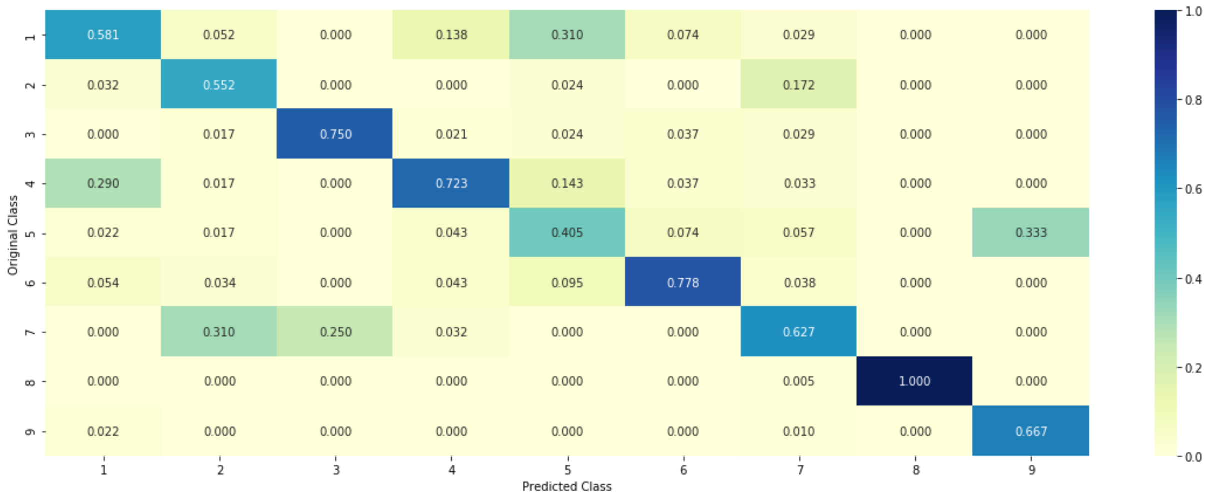
In [158]:
```
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-pro
bability estimates
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.pre
dict(cv_x_onehotCoding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray
()))
```

```
Log Loss : 1.1963818424
```

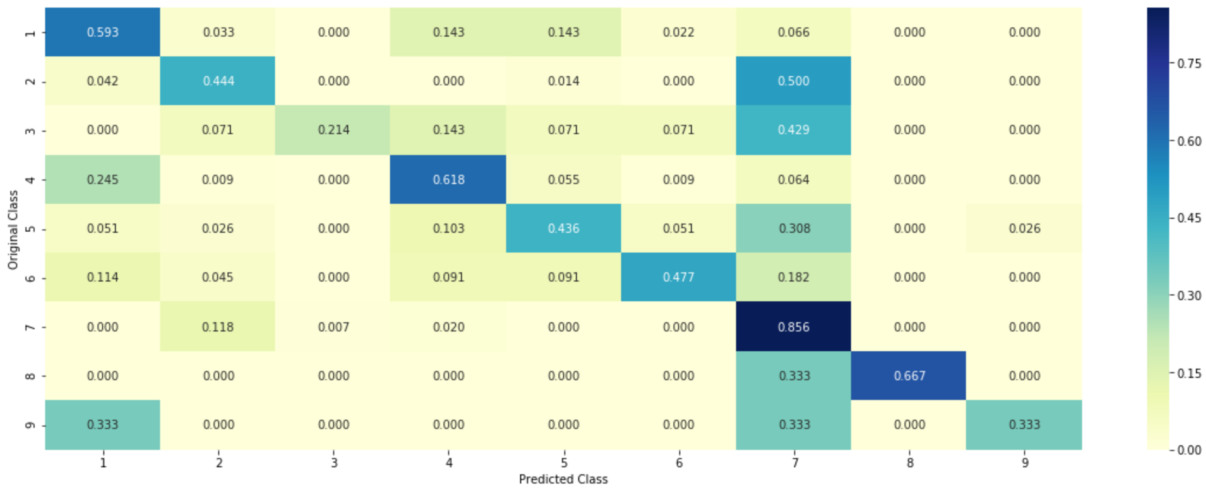
```
Log Loss : 1.1963616424
Number of missclassified point : 0.37969924812030076
------------------- Confusion matrix -------------------
```




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




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



4.1.1.3. Feature Importance, Correctly classified point

In [159]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
#print(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
#print(indices)
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: [[ 0.0641  0.0597  0.0116  0.0731  0.034
9  0.039   0.7092  0.0048  0.0036]]
Actual Class : 6
--------------------------------------------------
```

```
17 Text feature [78] present in test data point [True]
20 Text feature [currently] present in test data point [True]
23 Text feature [inhibiting] present in test data point [True]
25 Text feature [combinations] present in test data point [True]
28 Text feature [available] present in test data point [True]
32 Text feature [endogenous] present in test data point [True]
33 Text feature [drosophila] present in test data point [True]
36 Text feature [disrupt] present in test data point [True]
42 Text feature [initial] present in test data point [True]
52 Text feature [followed] present in test data point [True]
53 Text feature [makes] present in test data point [True]
55 Text feature [nt] present in test data point [True]
62 Text feature [equivalent] present in test data point [True]
65 Text feature [middle] present in test data point [True]
67 Text feature [given] present in test data point [True]
68 Text feature [doses] present in test data point [True]
74 Text feature [encode] present in test data point [True]
78 Text feature [data] present in test data point [True]
80 Text feature [overlapping] present in test data point [True]
81 Text feature [activation] present in test data point [True]
83 Text feature [14] present in test data point [True]
84 Text feature [outcome] present in test data point [True]
85 Text feature [22] present in test data point [True]
86 Text feature [nearly] present in test data point [True]
87 Text feature [23] present in test data point [True]
91 Text feature [heterogeneous] present in test data point [True]
92 Text feature [10] present in test data point [True]
93 Text feature [large] present in test data point [True]
94 Text feature [lethal] present in test data point [True]
95 Text feature [1c] present in test data point [True]
97 Text feature [classified] present in test data point [True]
98 Text feature [interact] present in test data point [True]
99 Text feature [modification] present in test data point [True]
Out of the top  100  features  33 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [160]:
```
test_point_index = 105
```

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

```
Predicted Class : 4
Predicted Class Probabilities: [[ 0.1456  0.0475  0.0119  0.6271  0.035
1  0.0382  0.086   0.0049  0.0038]]
Actual Class : 4
--------------------------------------------------
14 Text feature [detectable] present in test data point [True]
20 Text feature [6c] present in test data point [True]
21 Text feature [active] present in test data point [True]
23 Text feature [detection] present in test data point [True]
30 Text feature [actin] present in test data point [True]
33 Text feature [overexpressed] present in test data point [True]
35 Text feature [outcome] present in test data point [True]
36 Text feature [np] present in test data point [True]
39 Text feature [outcomes] present in test data point [True]
40 Text feature [male] present in test data point [True]
42 Text feature [ligand] present in test data point [True]
44 Text feature [idea] present in test data point [True]
45 Text feature [moderate] present in test data point [True]
52 Text feature [mutations] present in test data point [True]
53 Text feature [findings] present in test data point [True]
54 Text feature [investigation] present in test data point [True]
56 Text feature [fetal] present in test data point [True]
59 Text feature [followed] present in test data point [True]
73 Text feature [activation] present in test data point [True]
75 Text feature [implicated] present in test data point [True]
82 Text feature [markedly] present in test data point [True]
86 Text feature [associate] present in test data point [True]
87 Text feature [motif] present in test data point [True]
```

```
89 Text feature [gift] present in test data point [True]
92 Text feature [current] present in test data point [True]
96 Text feature [finger] present in test data point [True]
Out of the top  100  features  26 are present in query point
```

4.2. K Nearest Neighbour Classification

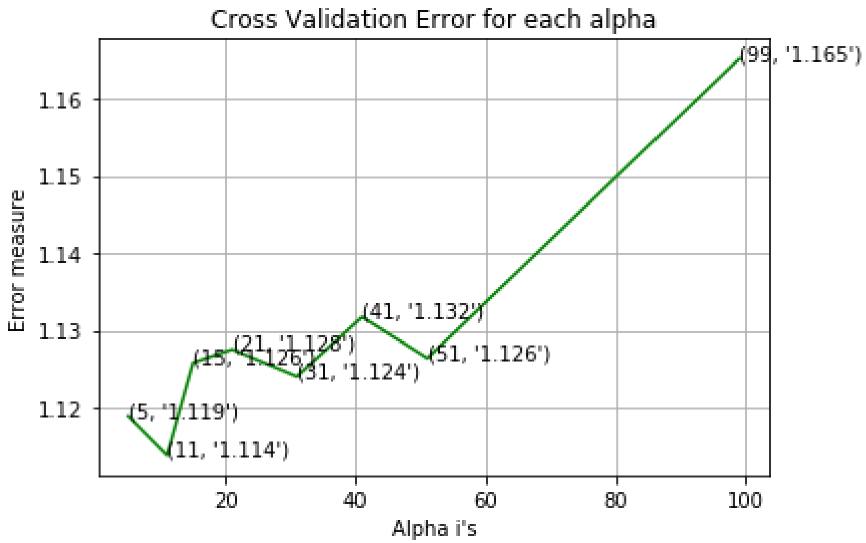
4.2.1. Hyper parameter tuning

In [161]:

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

```
for alpha = 5
Log Loss : 1.11888358474
for alpha = 11
Log Loss : 1.11379549785
for alpha = 15
Log Loss : 1.12579473104
for alpha = 21
Log Loss : 1.12750878712
for alpha = 31
Log Loss : 1.12402391974
for alpha = 41
Log Loss : 1.13169897212
for alpha = 51
Log Loss : 1.12636658394
for alpha = 99
Log Loss : 1.16512905421
```

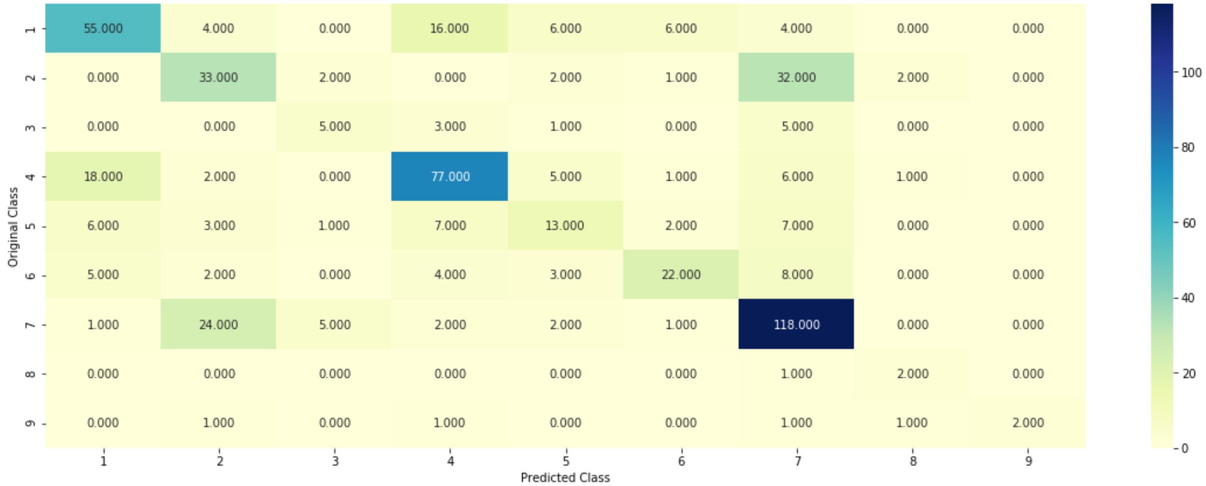



```
For values of best alpha =  11 The train log loss is: 0.634325041646
For values of best alpha =  11 The cross validation log loss is: 1.1137
9549785
For values of best alpha =  11 The test log loss is: 1.05439028978
```

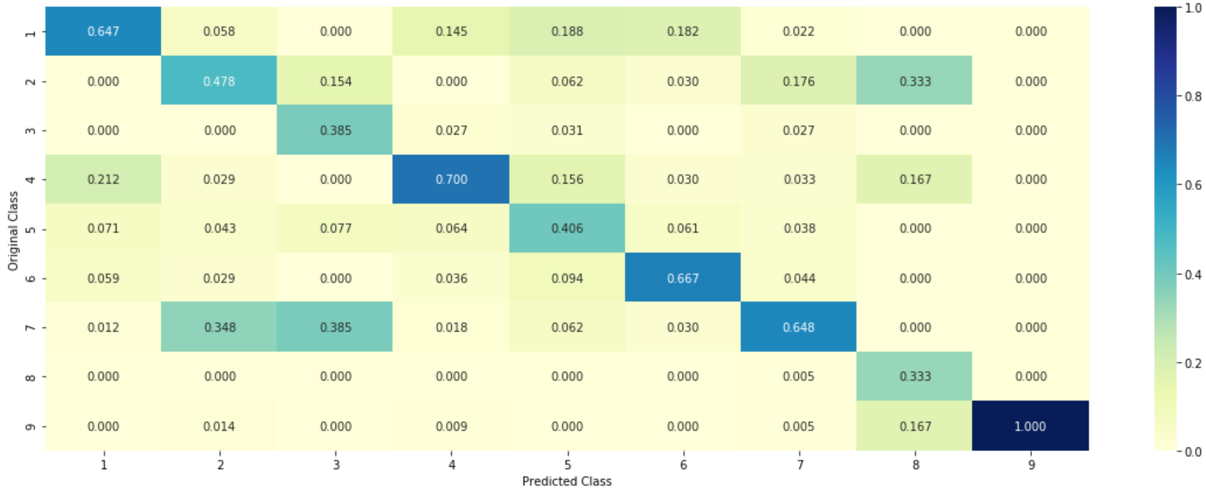
4.2.2. Testing the model with best hyper paramters

In [162]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x
_responseCoding, cv_y, clf)
```

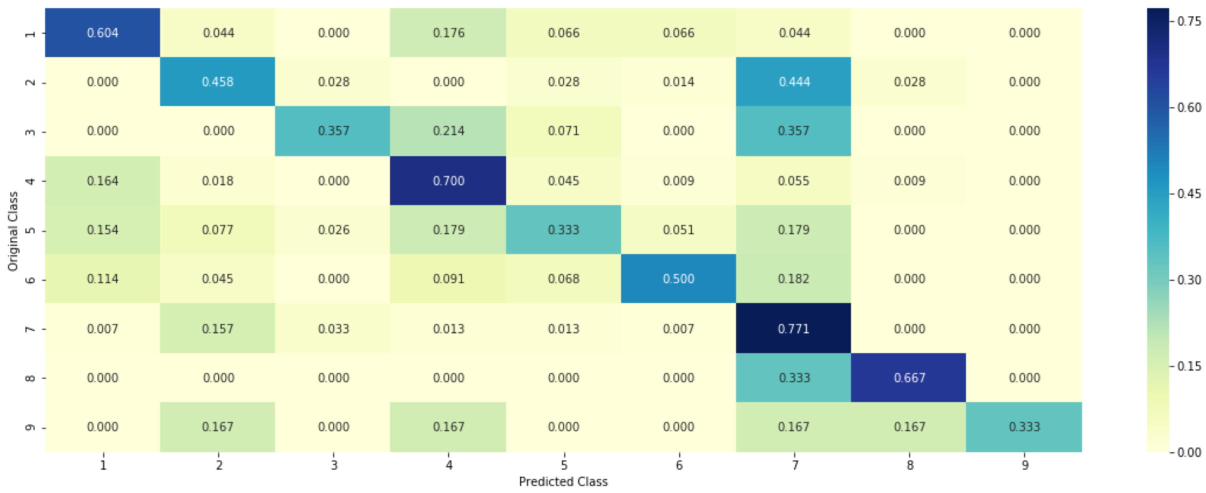
```
Log loss : 1.11379549785
Number of mis-classified points : 0.38533834586466165
-------------------- Confusion matrix --------------------
```



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



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



4.2.3.Sample Query point -1

In [163]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

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

```
Predicted Class : 2
Actual Class : 6
The  11  nearest neighbours of the test points belongs to classes [7 6
```

```
6 2 7 6 2 7 7 2 6]
Fequency of nearest points : Counter({7: 4, 6: 4, 2: 3})
```

4.2.4. Sample Query Point-2

In [164]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 100

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

```
Predicted Class : 2
Actual Class : 2
the k value for knn is 11 and the nearest neighbours of the test points
belongs to classes [1 2 8 2 1 2 2 4 1 4 1]
Fequency of nearest points : Counter({1: 4, 2: 4, 4: 2, 8: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

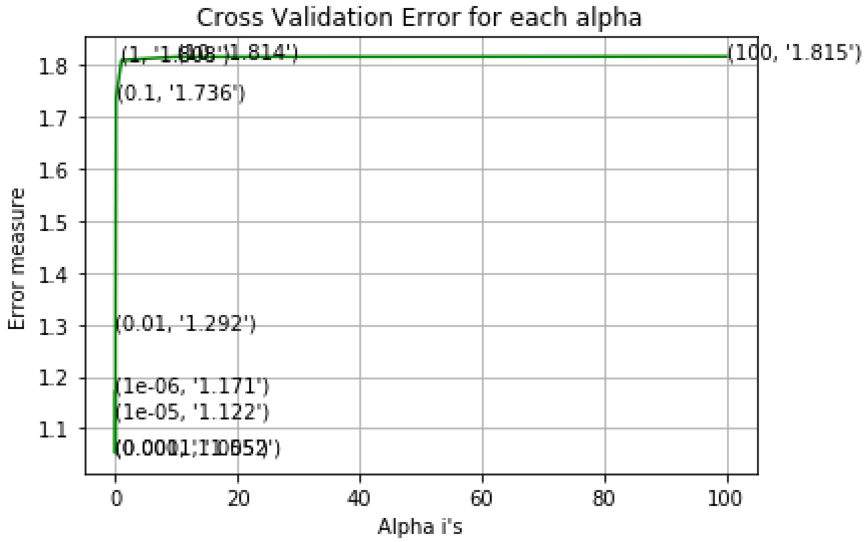
4.3.1.1. Hyper paramter tuning

In [165]:

```
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_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log
-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

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

```
for alpha = 1e-06
Log Loss : 1.1713789118
for alpha = 1e-05
Log Loss : 1.1218123291
for alpha = 0.0001
Log Loss : 1.05198598734
for alpha = 0.001
Log Loss : 1.05465507316
for alpha = 0.01
Log Loss : 1.29231225506
for alpha = 0.1
Log Loss : 1.73589937607
for alpha = 1
Log Loss : 1.80827681236
for alpha = 10
Log Loss : 1.81446676123
for alpha = 100
Log Loss : 1.81512845576
```



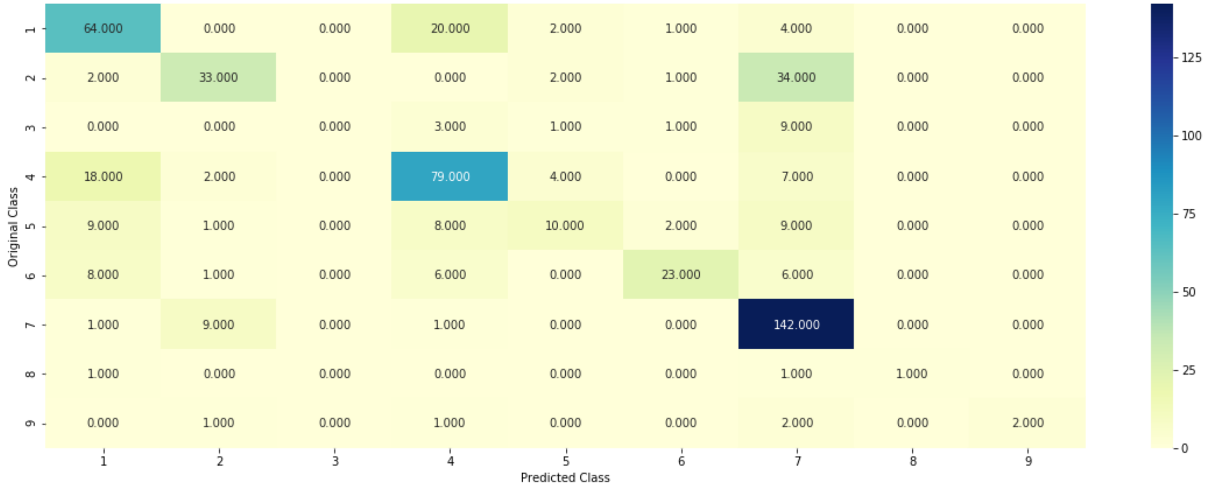

```
For values of best alpha =  0.0001 The train log loss is: 0.43556840715
2
For values of best alpha =  0.0001 The cross validation log loss is: 1.
05198598734
For values of best alpha =  0.0001 The test log loss is: 0.963724985893
```

4.3.1.2. Testing the model with best hyper paramters

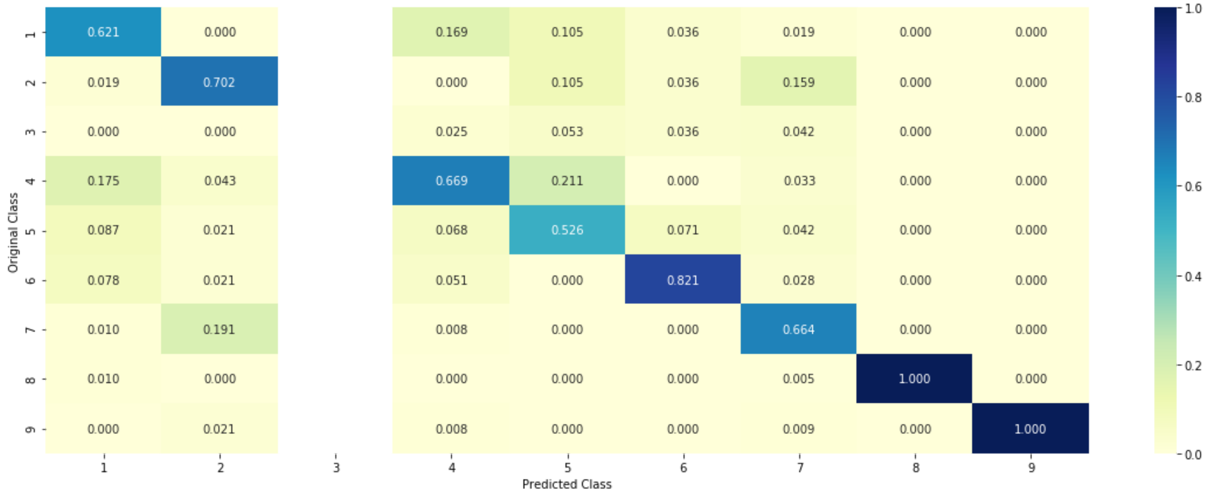
In [166]:

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_o
nehotCoding, cv_y, clf)
```

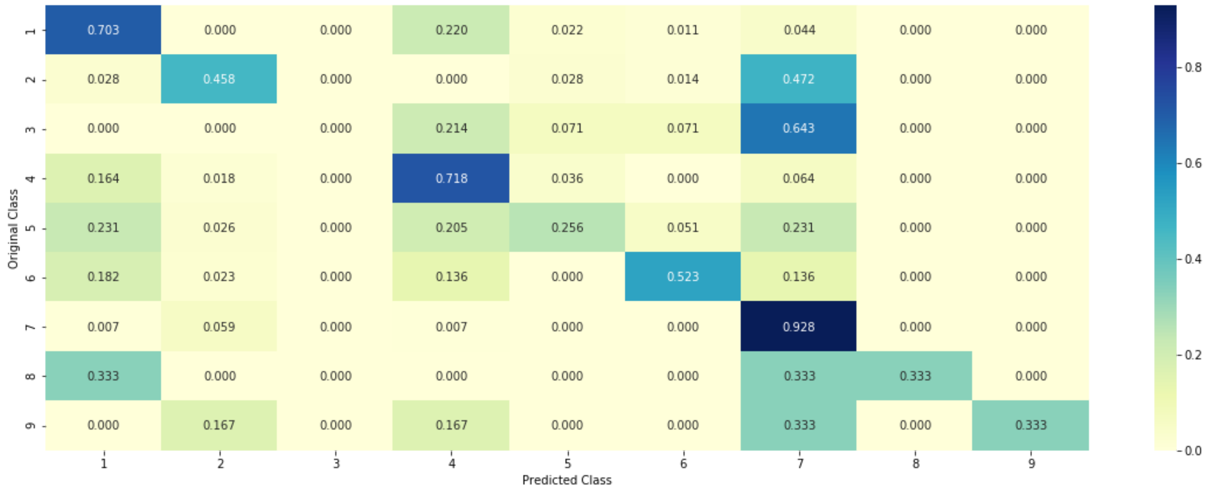
```
Log loss : 1.05198598734
Number of mis-classified points : 0.33458646616541354
-------------------- Confusion matrix --------------------
```



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



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



4.3.1.3. Feature Importance

```
In [167]: def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"
])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features
[i], yes_no])
        incresingorder_ind += 1
```

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

4.3.1.3.1. Correctly Classified point

In [168]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-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: [[ 0.0165  0.0934  0.0058  0.0157  0.0068  0.3324  0.5254  0.0028  0.0012]]
Actual Class : 6
--------------------------------------------------
28 Text feature [78] present in test data point [True]
55 Text feature [ability] present in test data point [True]
61 Text feature [highly] present in test data point [True]
70 Text feature [helix] present in test data point [True]
86 Text feature [combinations] present in test data point [True]
99 Text feature [discovery] present in test data point [True]
```

```
130 Text feature [central] present in test data point [True]
139 Text feature [map] present in test data point [True]
154 Text feature [basal] present in test data point [True]
175 Text feature [currently] present in test data point [True]
185 Text feature [catalytic] present in test data point [True]
187 Text feature [next] present in test data point [True]
194 Text feature [conformational] present in test data point [True]
229 Text feature [lower] present in test data point [True]
273 Text feature [likely] present in test data point [True]
302 Text feature [modification] present in test data point [True]
304 Text feature [features] present in test data point [True]
305 Text feature [mek2] present in test data point [True]
334 Text feature [dose] present in test data point [True]
339 Text feature [families] present in test data point [True]
340 Text feature [loss] present in test data point [True]
342 Text feature [associated] present in test data point [True]
343 Text feature [asp] present in test data point [True]
346 Text feature [endothelial] present in test data point [True]
356 Text feature [causing] present in test data point [True]
366 Text feature [developed] present in test data point [True]
376 Text feature [expressing] present in test data point [True]
378 Text feature [40] present in test data point [True]
391 Text feature [erk2] present in test data point [True]
405 Text feature [dimer] present in test data point [True]
413 Text feature [included] present in test data point [True]
414 Text feature [forms] present in test data point [True]
428 Text feature [antibody] present in test data point [True]
451 Text feature [local] present in test data point [True]
453 Text feature [important] present in test data point [True]
458 Text feature [introduction] present in test data point [True]
466 Text feature [indicates] present in test data point [True]
470 Text feature [fgfr1] present in test data point [True]
474 Text feature [black] present in test data point [True]
477 Text feature [center] present in test data point [True]
481 Text feature [25] present in test data point [True]
488 Text feature [chemical] present in test data point [True]
489 Text feature [intrinsic] present in test data point [True]
494 Text feature [interaction] present in test data point [True]
```

```
496 Text feature [conservation] present in test data point [True]
Out of the top  500  features  45 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [169]:

```
test_point_index = 19
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.0032  0.0453  0.0016  0.0067  0.006
1  0.0018  0.9275  0.0061  0.0018]]
Actual Class : 7
--------------------------------------------------
18 Text feature [germline] present in test data point [True]
61 Text feature [highly] present in test data point [True]
63 Text feature [conjugated] present in test data point [True]
87 Text feature [carry] present in test data point [True]
99 Text feature [discovery] present in test data point [True]
128 Text feature [biosystems] present in test data point [True]
175 Text feature [currently] present in test data point [True]
180 Text feature [80] present in test data point [True]
187 Text feature [next] present in test data point [True]
228 Text feature [description] present in test data point [True]
229 Text feature [lower] present in test data point [True]
297 Text feature [69] present in test data point [True]
304 Text feature [features] present in test data point [True]
316 Text feature [cruz] present in test data point [True]
334 Text feature [dose] present in test data point [True]
```

```
335 Text feature [arrow] present in test data point [True]
342 Text feature [associated] present in test data point [True]
347 Text feature [bcr] present in test data point [True]
376 Text feature [expressing] present in test data point [True]
378 Text feature [40] present in test data point [True]
414 Text feature [forms] present in test data point [True]
421 Text feature [directly] present in test data point [True]
428 Text feature [antibody] present in test data point [True]
453 Text feature [important] present in test data point [True]
458 Text feature [introduction] present in test data point [True]
465 Text feature [enzymatic] present in test data point [True]
473 Text feature [days] present in test data point [True]
477 Text feature [center] present in test data point [True]
478 Text feature [initiation] present in test data point [True]
481 Text feature [25] present in test data point [True]
488 Text feature [chemical] present in test data point [True]
489 Text feature [intrinsic] present in test data point [True]
Out of the top  500  features  32 are present in query point
```

4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

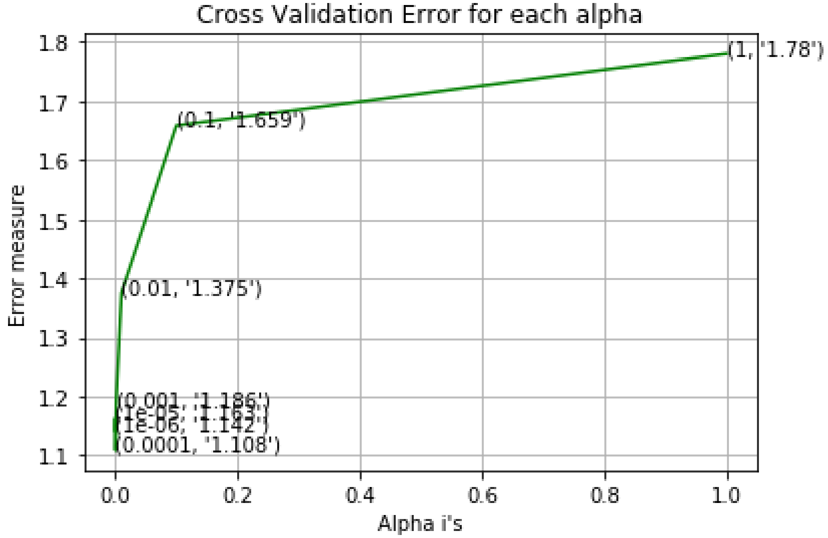
In [170]:

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.14205816426
for alpha = 1e-05
Log Loss : 1.16332931608
for alpha = 0.0001
Log Loss : 1.10818659655
for alpha = 0.001
Log Loss : 1.1855258653
for alpha = 0.01
```

```
Log Loss : 1.37524820549
for alpha = 0.1
Log Loss : 1.65861316348
for alpha = 1
Log Loss : 1.78040180967
```



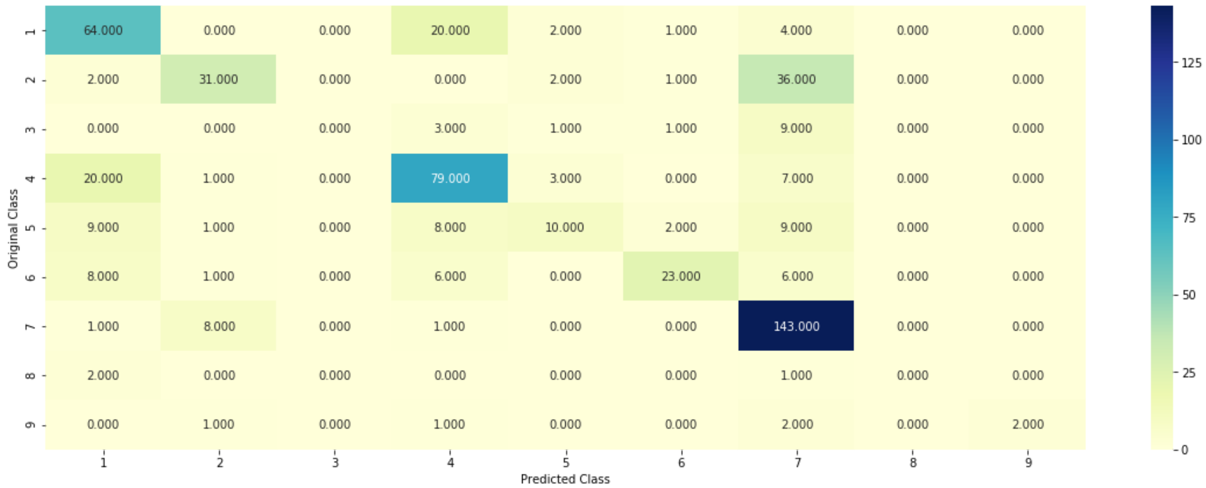

```
For values of best alpha =  0.0001 The train log loss is: 0.43668544722
5
For values of best alpha =  0.0001 The cross validation log loss is: 1.
10818659655
For values of best alpha =  0.0001 The test log loss is: 0.997120843967
```

4.3.2.2. Testing model with best hyper parameters

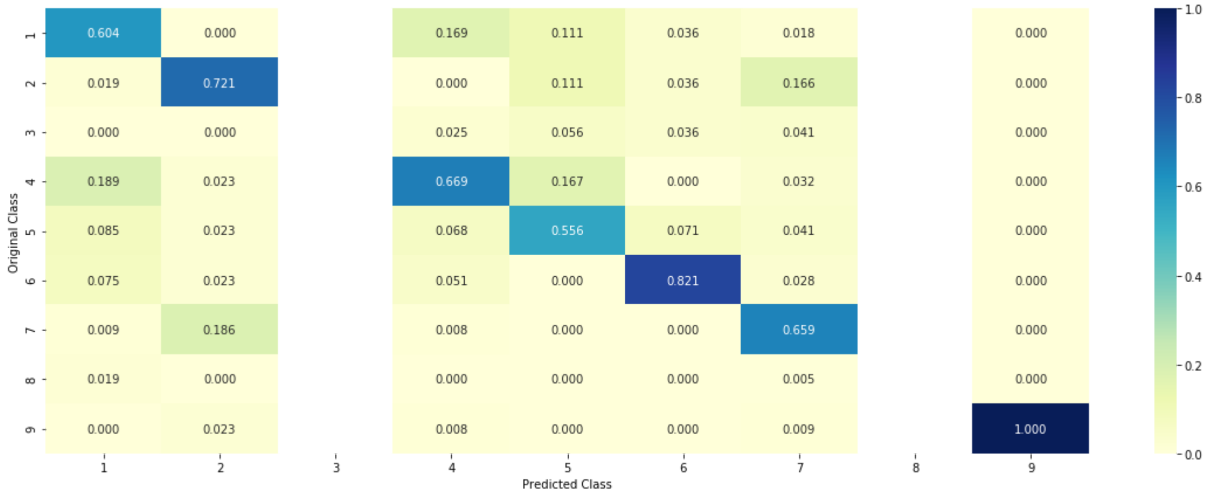
In [171]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_o
nehotCoding, cv_y, clf)
```

```
Log loss : 1.10818659655
Number of mis-classified points : 0.3383458646616541
```

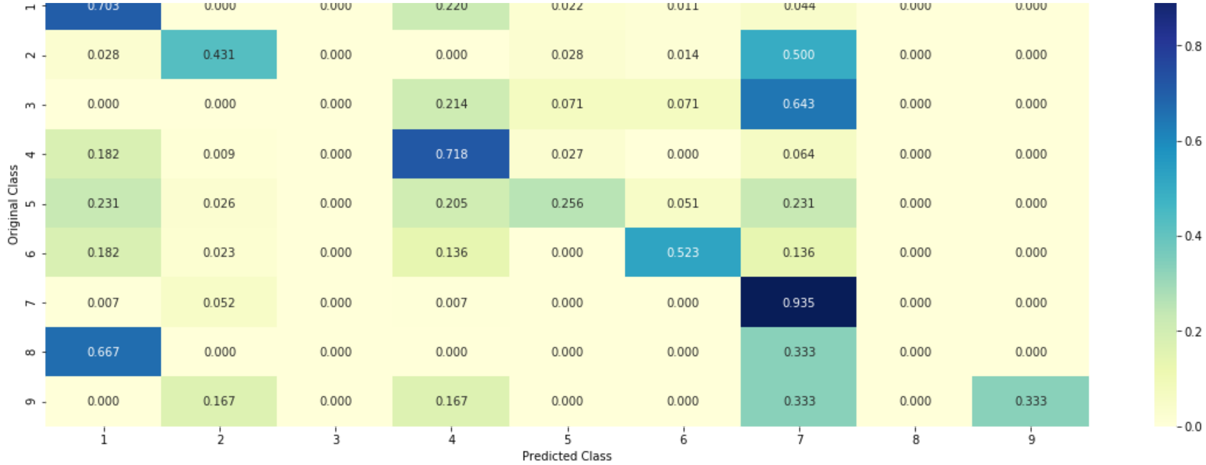
------------------- Confusion matrix -------------------



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



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



4.3.2.3. Feature Importance, Correctly Classified point

In [172]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[  1.86000000e-02   9.35000000e-02   2.
70000000e-03   1.41000000e-02
    7.40000000e-03   3.15400000e-01   5.38500000e-01   9.40000000e-03
    3.00000000e-04]]
Actual Class : 6
```

```
---------------------------------------------------
63 Text feature [78] present in test data point [True]
75 Text feature [helix] present in test data point [True]
82 Text feature [ability] present in test data point [True]
120 Text feature [combinations] present in test data point [True]
129 Text feature [discovery] present in test data point [True]
165 Text feature [highly] present in test data point [True]
178 Text feature [catalytic] present in test data point [True]
209 Text feature [basal] present in test data point [True]
217 Text feature [conformational] present in test data point [True]
220 Text feature [lower] present in test data point [True]
221 Text feature [map] present in test data point [True]
238 Text feature [currently] present in test data point [True]
257 Text feature [central] present in test data point [True]
284 Text feature [causing] present in test data point [True]
286 Text feature [families] present in test data point [True]
287 Text feature [erk2] present in test data point [True]
304 Text feature [mek2] present in test data point [True]
335 Text feature [dose] present in test data point [True]
350 Text feature [loss] present in test data point [True]
361 Text feature [endothelial] present in test data point [True]
363 Text feature [likely] present in test data point [True]
380 Text feature [features] present in test data point [True]
396 Text feature [asp] present in test data point [True]
401 Text feature [introduction] present in test data point [True]
403 Text feature [modification] present in test data point [True]
409 Text feature [included] present in test data point [True]
417 Text feature [forms] present in test data point [True]
419 Text feature [indicates] present in test data point [True]
435 Text feature [black] present in test data point [True]
436 Text feature [next] present in test data point [True]
441 Text feature [40] present in test data point [True]
442 Text feature [fgfr1] present in test data point [True]
472 Text feature [dimer] present in test data point [True]
476 Text feature [intrinsic] present in test data point [True]
477 Text feature [expressing] present in test data point [True]
481 Text feature [developed] present in test data point [True]
489 Text feature [associated] present in test data point [True]

495 Text feature [critical] present in test data point [True]
```

```
Out of the top  500  features  38 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [173]:
```
test_point_index = 19
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[  4.00000000e-03   4.16000000e-02   2.
00000000e-03   7.20000000e-03
    4.70000000e-03   1.50000000e-03   9.31900000e-01   7.00000000e-03
    1.00000000e-04]]
Actual Class : 7
--------------------------------------------------
39 Text feature [germline] present in test data point [True]
81 Text feature [conjugated] present in test data point [True]
129 Text feature [discovery] present in test data point [True]
139 Text feature [biosystems] present in test data point [True]
157 Text feature [carry] present in test data point [True]
165 Text feature [highly] present in test data point [True]
220 Text feature [lower] present in test data point [True]
226 Text feature [80] present in test data point [True]
238 Text feature [currently] present in test data point [True]
313 Text feature [description] present in test data point [True]
327 Text feature [arrow] present in test data point [True]
335 Text feature [dose] present in test data point [True]
372 Text feature [69] present in test data point [True]
380 Text feature [features] present in test data point [True]
```

```
401 Text feature [introduction] present in test data point [True]
412 Text feature [initiation] present in test data point [True]
417 Text feature [forms] present in test data point [True]
418 Text feature [cruz] present in test data point [True]
426 Text feature [directly] present in test data point [True]
436 Text feature [next] present in test data point [True]
441 Text feature [40] present in test data point [True]
476 Text feature [intrinsic] present in test data point [True]
477 Text feature [expressing] present in test data point [True]
480 Text feature [initiated] present in test data point [True]
489 Text feature [associated] present in test data point [True]
494 Text feature [bcr] present in test data point [True]
499 Text feature [days] present in test data point [True]
Out of the top  500  features  27 are present in query point
```

4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

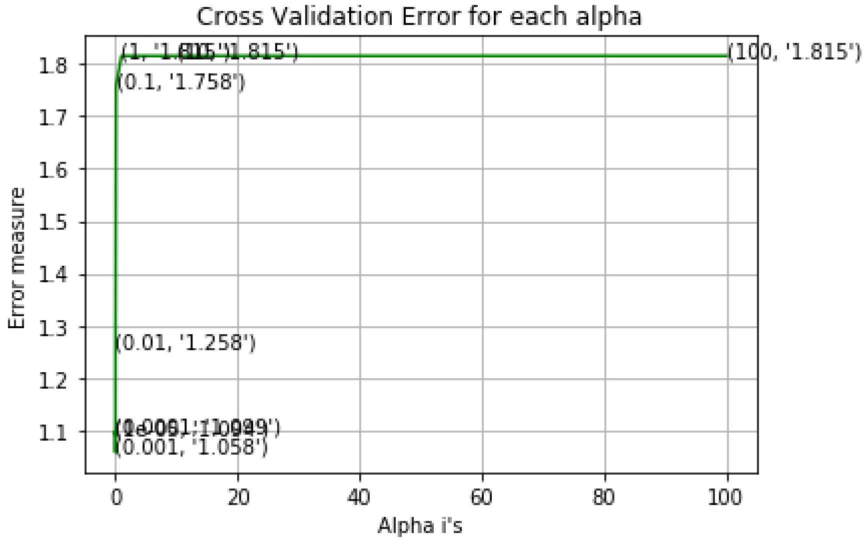
In [174]:

```
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='bal
anced')
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2'
, loss='hinge', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
```

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


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

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

```
for C = 1e-05
Log Loss : 1.09353503316
for C = 0.0001
Log Loss : 1.09891525827
for C = 0.001
Log Loss : 1.05791308119
for C = 0.01
Log Loss : 1.25790508079
for C = 0.1
```

```
Log Loss : 1.75802163206
for C = 1
Log Loss : 1.81526158622
for C = 10
Log Loss : 1.81526167146
for C = 100
Log Loss : 1.81526159594
```



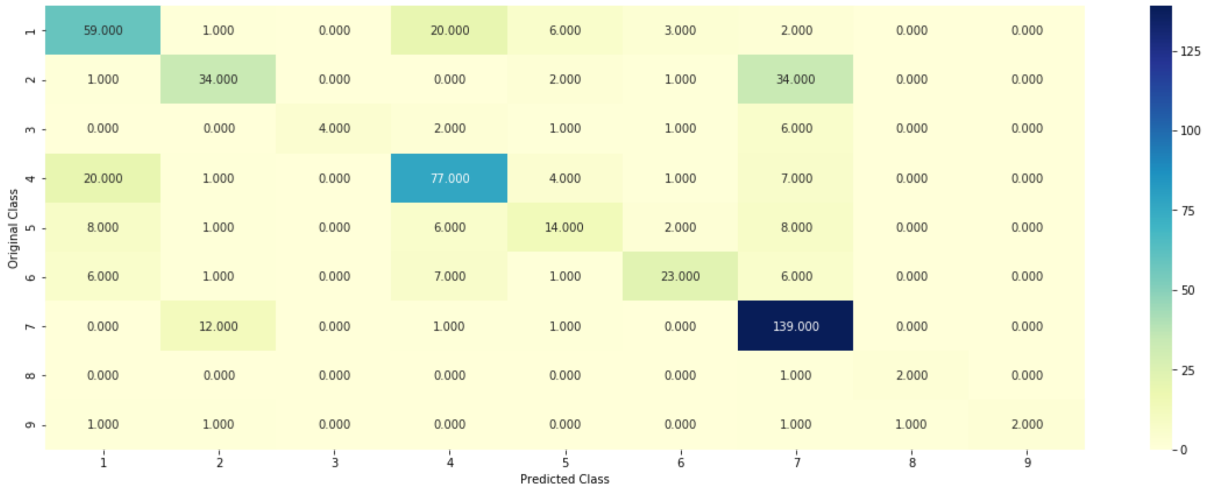

```
For values of best alpha =  0.001 The train log loss is: 0.549583303236
For values of best alpha =  0.001 The cross validation log loss is: 1.0
5791308119
For values of best alpha =  0.001 The test log loss is: 1.0192120915
```

4.4.2. Testing model with best hyper parameters

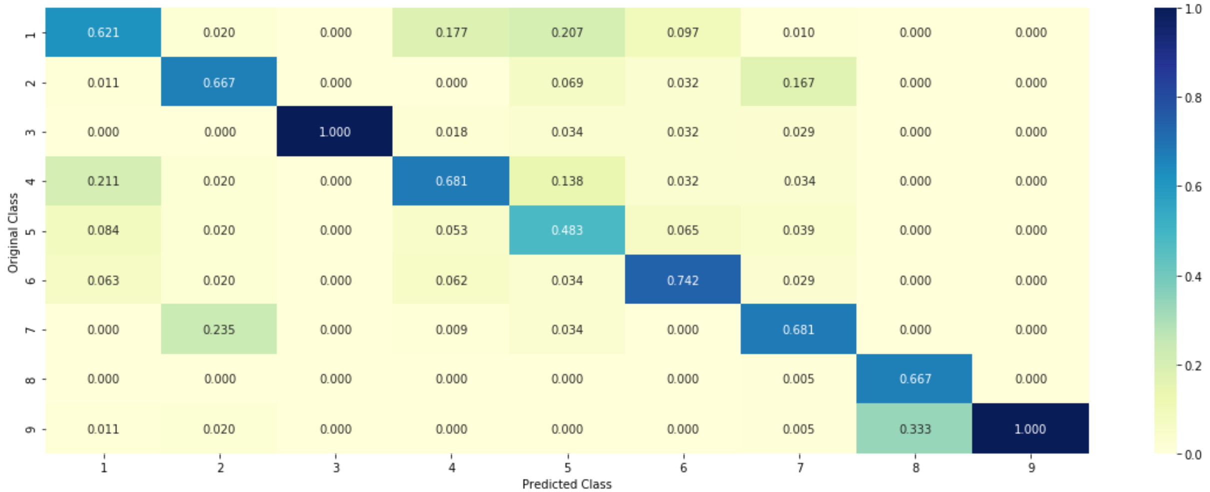
In [175]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge'
, random_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_on
ehotCoding,cv_y, clf)
```

```
Log loss : 1.05791308119
```

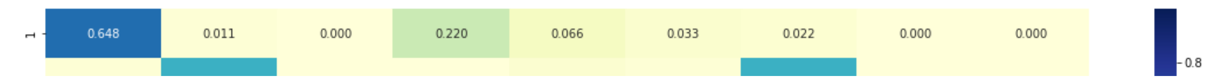
```
Number of mis-classified points : 0.33458646616541354
------------------- Confusion matrix -------------------
```

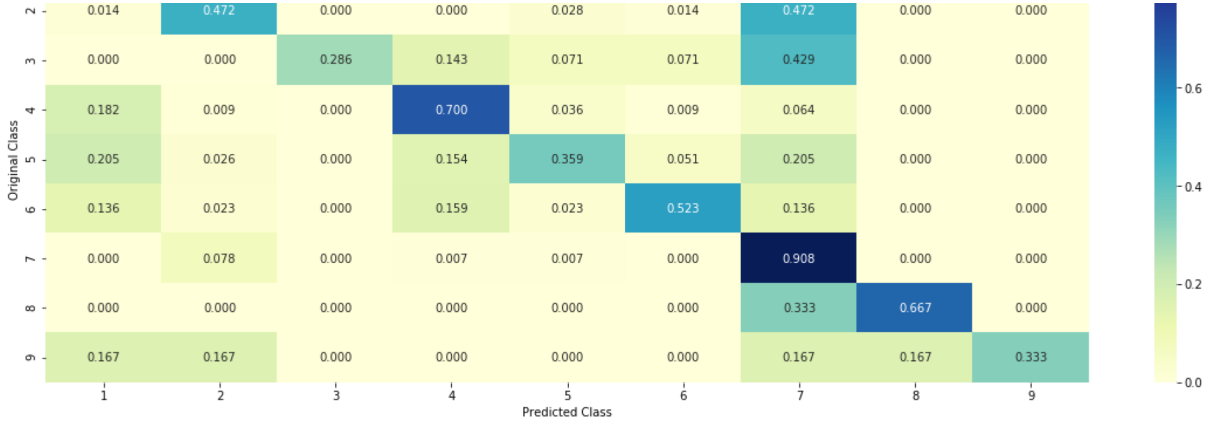



```
------------------- Precision matrix (Columm Sum=1) -----------------
--
```




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





4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [176]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge'
, random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.0329  0.064   0.0105  0.039   0.013
```

```
1  0.4041  0.4321  0.0024  0.002 ]]
Actual Class : 6
--------------------------------------------------
31 Text feature [ability] present in test data point [True]
34 Text feature [78] present in test data point [True]
39 Text feature [helix] present in test data point [True]
51 Text feature [map] present in test data point [True]
56 Text feature [currently] present in test data point [True]
63 Text feature [lower] present in test data point [True]
64 Text feature [combinations] present in test data point [True]
216 Text feature [discovery] present in test data point [True]
221 Text feature [likely] present in test data point [True]
223 Text feature [highly] present in test data point [True]
227 Text feature [erk2] present in test data point [True]
228 Text feature [next] present in test data point [True]
229 Text feature [central] present in test data point [True]
233 Text feature [basal] present in test data point [True]
240 Text feature [catalytic] present in test data point [True]
241 Text feature [developed] present in test data point [True]
242 Text feature [mek2] present in test data point [True]
247 Text feature [conformational] present in test data point [True]
248 Text feature [endothelial] present in test data point [True]
253 Text feature [25] present in test data point [True]
254 Text feature [causing] present in test data point [True]
257 Text feature [introduction] present in test data point [True]
413 Text feature [melanomas] present in test data point [True]
416 Text feature [dose] present in test data point [True]
419 Text feature [loss] present in test data point [True]
421 Text feature [interaction] present in test data point [True]
422 Text feature [asp] present in test data point [True]
423 Text feature [important] present in test data point [True]
426 Text feature [included] present in test data point [True]
428 Text feature [23] present in test data point [True]
432 Text feature [consequences] present in test data point [True]
434 Text feature [indicates] present in test data point [True]
435 Text feature [help] present in test data point [True]
436 Text feature [modification] present in test data point [True]
437 Text feature [associated] present in test data point [True]
446 Text feature [distributed] present in test data point [True]
```

```
450 Text feature [harbor] present in test data point [True]
456 Text feature [chemical] present in test data point [True]
458 Text feature [features] present in test data point [True]
461 Text feature [88] present in test data point [True]
464 Text feature [allowed] present in test data point [True]
470 Text feature [critical] present in test data point [True]
474 Text feature [fgfr1] present in test data point [True]
479 Text feature [allowing] present in test data point [True]
480 Text feature [families] present in test data point [True]
482 Text feature [expressing] present in test data point [True]
483 Text feature [forms] present in test data point [True]
484 Text feature [center] present in test data point [True]
489 Text feature [87] present in test data point [True]
493 Text feature [dimer] present in test data point [True]
494 Text feature [endogenous] present in test data point [True]
Out of the top  500  features  51 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [177]:

```
test_point_index = 19
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.0194  0.0675  0.0067  0.022   0.021
2  0.0043  0.8483  0.008   0.0025]]
Actual Class : 7
--------------------------------------------------
36 Text feature [germline] present in test data point [True]
```

```
38 Text feature [carry] present in test data point [True]
56 Text feature [currently] present in test data point [True]
63 Text feature [lower] present in test data point [True]
68 Text feature [biosystems] present in test data point [True]
216 Text feature [discovery] present in test data point [True]
222 Text feature [80] present in test data point [True]
223 Text feature [highly] present in test data point [True]
228 Text feature [next] present in test data point [True]
230 Text feature [conjugated] present in test data point [True]
235 Text feature [directly] present in test data point [True]
246 Text feature [2003] present in test data point [True]
253 Text feature [25] present in test data point [True]
256 Text feature [cruz] present in test data point [True]
257 Text feature [introduction] present in test data point [True]
405 Text feature [bcr] present in test data point [True]
408 Text feature [arrow] present in test data point [True]
414 Text feature [initiation] present in test data point [True]
416 Text feature [dose] present in test data point [True]
423 Text feature [important] present in test data point [True]
424 Text feature [days] present in test data point [True]
432 Text feature [consequences] present in test data point [True]
437 Text feature [associated] present in test data point [True]
448 Text feature [blood] present in test data point [True]
456 Text feature [chemical] present in test data point [True]
458 Text feature [features] present in test data point [True]
466 Text feature [nucleus] present in test data point [True]
467 Text feature [cultured] present in test data point [True]
468 Text feature [abl] present in test data point [True]
469 Text feature [enzymatic] present in test data point [True]
473 Text feature [69] present in test data point [True]
481 Text feature [biotechnology] present in test data point [True]
482 Text feature [expressing] present in test data point [True]
483 Text feature [forms] present in test data point [True]
484 Text feature [center] present in test data point [True]
495 Text feature [initiated] present in test data point [True]
Out of the top  500  features  36 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [178]:
```
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,class_weight = 'bal
anced', 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]).ra
vel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)), (featur
es[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''


best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)],clas
s_weight = 'balanced', criterion='gini', max_depth=max_depth[int(best_a
lpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
```

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

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.25804014021
for n_estimators = 100 and max depth =  10
Log Loss : 1.25302008211
for n_estimators = 200 and max depth =  5
Log Loss : 1.22180217587
for n_estimators = 200 and max depth =  10
Log Loss : 1.23962734195
for n_estimators = 500 and max depth =  5
Log Loss : 1.20009615088
for n_estimators = 500 and max depth =  10
Log Loss : 1.22111340647
for n_estimators = 1000 and max depth =  5
Log Loss : 1.19554123936
for n_estimators = 1000 and max depth =  10
Log Loss : 1.21969550316
for n_estimators = 2000 and max depth =  5
Log Loss : 1.19114628533
for n_estimators = 2000 and max depth =  10
Log Loss : 1.21592443538
For values of best estimator =  2000 The train log loss is: 0.867514709
882
For values of best estimator =  2000 The cross validation log loss is:
```

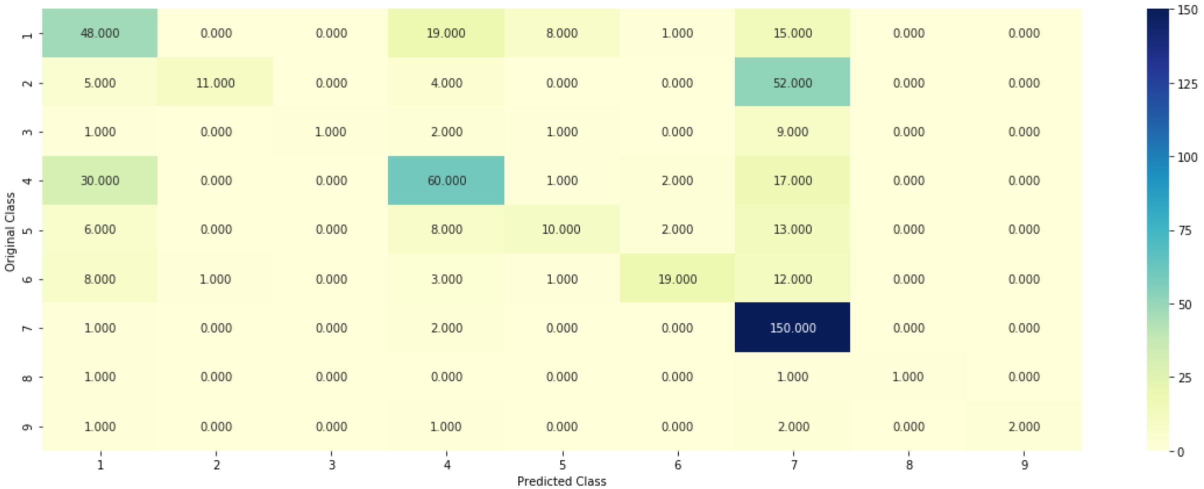
```
1.19114628533
For values of best estimator =  2000 The test log loss is: 1.1946788869
```

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

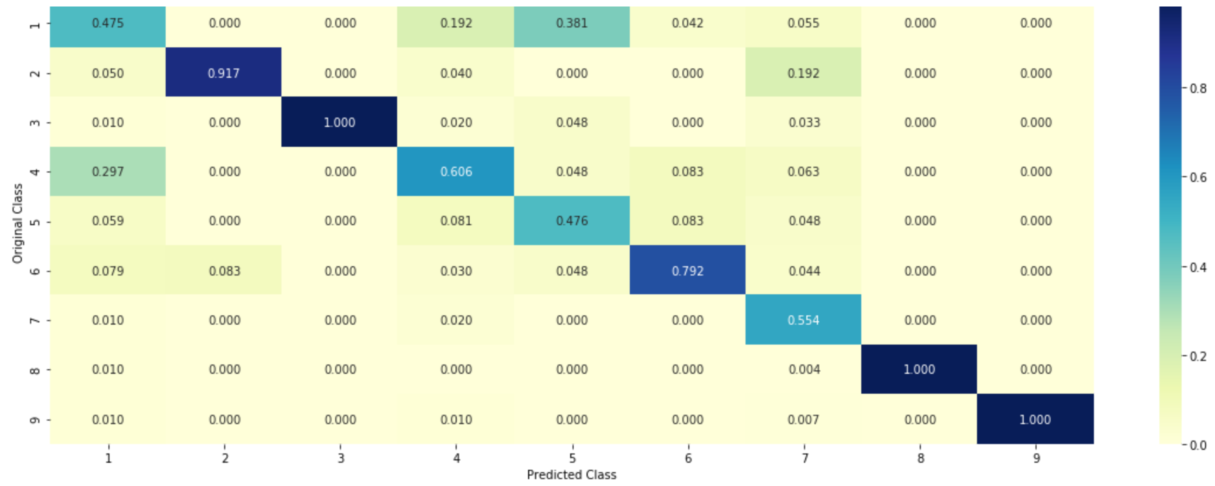
In [179]:

```python
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)],class_weight = 'balanced', 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 : 1.19114628533
Number of mis-classified points : 0.4323308270676692
-------------------- Confusion matrix --------------------
```




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




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

4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [180]:

```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)],class_weight = 'balanced', 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 = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 8
Predicted Class Probabilities: [[ 0.144   0.2451  0.0118  0.0373  0.037
   0.0312  0.2096  0.2762  0.0078]]
Actual Class : 2
--------------------------------------------------
0 Text feature [maintained] present in test data point [True]
6 Text feature [exclude] present in test data point [True]
7 Text feature [conventional] present in test data point [True]
9 Text feature [downloaded] present in test data point [True]
12 Text feature [characterized] present in test data point [True]
13 Text feature [effector] present in test data point [True]
14 Text feature [accompanied] present in test data point [True]
15 Text feature [80] present in test data point [True]
16 Text feature [must] present in test data point [True]
17 Text feature [exogenous] present in test data point [True]
18 Text feature [activating] present in test data point [True]
19 Text feature [crystal] present in test data point [True]
20 Text feature [appear] present in test data point [True]
21 Text feature [medicine] present in test data point [True]
```

```
23 Text feature [1992] present in test data point [True]
28 Text feature [cruz] present in test data point [True]
30 Text feature [grow] present in test data point [True]
32 Text feature [26] present in test data point [True]
33 Text feature [apparent] present in test data point [True]
35 Text feature [detection] present in test data point [True]
38 Text feature [39] present in test data point [True]
40 Text feature [81] present in test data point [True]
42 Text feature [occur] present in test data point [True]
46 Text feature [akt1] present in test data point [True]
49 Text feature [assembly] present in test data point [True]
52 Text feature [evaluated] present in test data point [True]
54 Text feature [acute] present in test data point [True]
56 Text feature [gift] present in test data point [True]
57 Text feature [available] present in test data point [True]
58 Text feature [keywords] present in test data point [True]
59 Text feature [identified] present in test data point [True]
61 Text feature [high] present in test data point [True]
62 Text feature [800] present in test data point [True]
64 Text feature [obtain] present in test data point [True]
69 Text feature [observations] present in test data point [True]
70 Text feature [71] present in test data point [True]
72 Text feature [applied] present in test data point [True]
73 Text feature [45] present in test data point [True]
74 Text feature [exclusive] present in test data point [True]
76 Text feature [black] present in test data point [True]
79 Text feature [functionally] present in test data point [True]
84 Text feature [appears] present in test data point [True]
85 Text feature [liquid] present in test data point [True]
87 Text feature [dependence] present in test data point [True]
89 Text feature [either] present in test data point [True]
93 Text feature [antibodies] present in test data point [True]
94 Text feature [4c] present in test data point [True]
96 Text feature [1995] present in test data point [True]
97 Text feature [equivalent] present in test data point [True]
Out of the top  100  features  49 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [181]:

```python
test_point_index = 19
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_po
int_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].
iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.0408  0.1974  0.0191  0.0402  0.041
   0.054   0.5878  0.0131  0.0067]]
Actuall Class : 7
--------------------------------------------------
14 Text feature [accompanied] present in test data point [True]
15 Text feature [80] present in test data point [True]
18 Text feature [activating] present in test data point [True]
20 Text feature [appear] present in test data point [True]
26 Text feature [culture] present in test data point [True]
28 Text feature [cruz] present in test data point [True]
32 Text feature [26] present in test data point [True]
36 Text feature [needed] present in test data point [True]
51 Text feature [bottom] present in test data point [True]
52 Text feature [evaluated] present in test data point [True]
54 Text feature [acute] present in test data point [True]
57 Text feature [available] present in test data point [True]
59 Text feature [identified] present in test data point [True]
61 Text feature [high] present in test data point [True]
69 Text feature [observations] present in test data point [True]
75 Text feature [institute] present in test data point [True]
89 Text feature [either] present in test data point [True]
90 Text feature [consent] present in test data point [True]
92 Text feature [bcr] present in test data point [True]
Out of the top  100  features  19 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [182]:

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


best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], cri
terion='gini',class_weight = 'balanced', max_depth=max_depth[int(best_a
lpha%4)], random_state=42, n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)
```

```
predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The tra
in log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=
1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cro
ss validation log loss is:",log_loss(y_cv, predict_y, labels=clf.classe
s_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The tes
t log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e
-15))
```

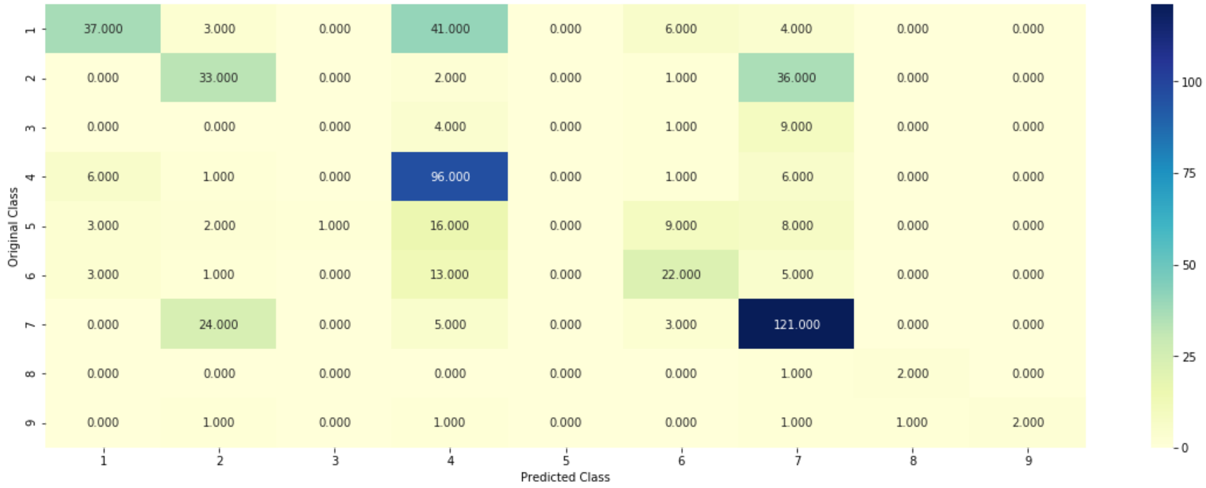
```
for n_estimators = 10 and max depth =  2
Log Loss : 1.8823607902
for n_estimators = 10 and max depth =  3
Log Loss : 1.59993620916
for n_estimators = 10 and max depth =  5
Log Loss : 1.44482399754
for n_estimators = 10 and max depth =  10
Log Loss : 1.54272757494
for n_estimators = 50 and max depth =  2
Log Loss : 1.5051325653
for n_estimators = 50 and max depth =  3
Log Loss : 1.58843458309
for n_estimators = 50 and max depth =  5
Log Loss : 1.3785562222
for n_estimators = 50 and max depth =  10
Log Loss : 1.35389545996
for n_estimators = 100 and max depth =  2
Log Loss : 1.62196866728
for n_estimators = 100 and max depth =  3
Log Loss : 1.52805993565
for n_estimators = 100 and max depth =  5
Log Loss : 1.32573866602
for n_estimators = 100 and max depth =  10
Log Loss : 1.35491894232

for n_estimators = 200 and max depth =  2
Log Loss : 1.61796783749
```

```
for n_estimators = 200 and max depth =  3
Log Loss : 1.53957709528
for n_estimators = 200 and max depth =  5
Log Loss : 1.34716934883
for n_estimators = 200 and max depth =  10
Log Loss : 1.34944006073
for n_estimators = 500 and max depth =  2
Log Loss : 1.64233010136
for n_estimators = 500 and max depth =  3
Log Loss : 1.5241505167
for n_estimators = 500 and max depth =  5
Log Loss : 1.33358392313
for n_estimators = 500 and max depth =  10
Log Loss : 1.329711402
for n_estimators = 1000 and max depth =  2
Log Loss : 1.63593960213
for n_estimators = 1000 and max depth =  3
Log Loss : 1.53556776799
for n_estimators = 1000 and max depth =  5
Log Loss : 1.33004880172
for n_estimators = 1000 and max depth =  10
Log Loss : 1.31178255581
For values of best alpha =  1000 The train log loss is: 0.0307524867024
For values of best alpha =  1000 The cross validation log loss is: 1.31
178255581
For values of best alpha =  1000 The test log loss is: 1.30325381137
```

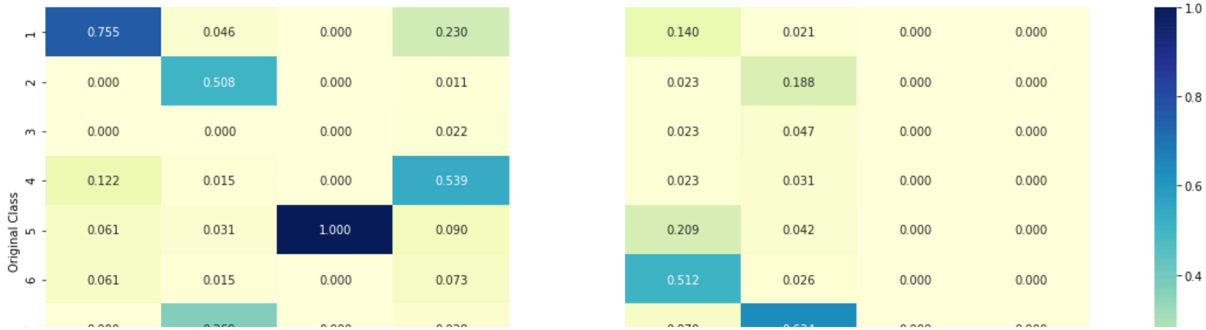
4.5.4. Testing model with best hyper parameters (Response Coding)

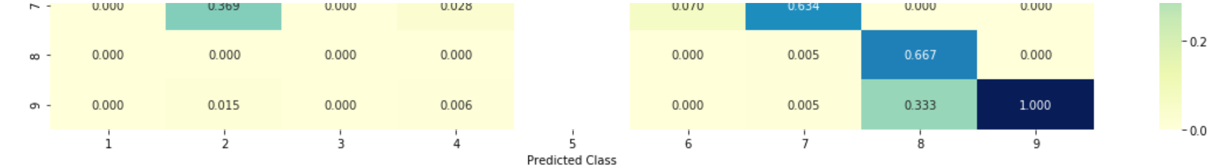
In [183]:
```
clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_
estimators=alpha[int(best_alpha/4)],class_weight = 'balanced', criterio
n='gini', max_features='auto',random_state=42)
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_
responseCoding,cv_y, clf)
```

```
Log loss : 1.31178255581
Number of mis-classified points : 0.4116541353383459
-------------------- Confusion matrix --------------------
```

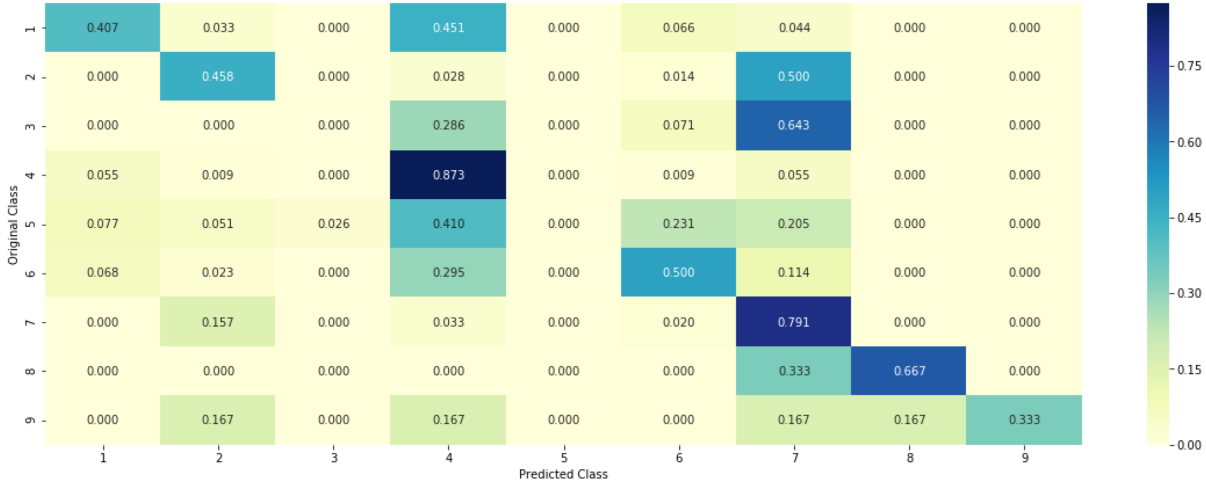


------------------- Precision matrix (Columm Sum=1) -------------------






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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [184]:

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


test_point_index = 1
no_feature = 27
```

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

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.0563  0.246   0.0206  0.1182  0.049
3  0.2157  0.2603  0.0185  0.0151]]
Actual Class : 6
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
```

```
Gene is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [185]:

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

```
Predicted Class : 2
Predicted Class Probabilities: [[ 0.1962  0.3294  0.0149  0.1726  0.027
8  0.0468  0.0731  0.1246  0.0146]]
Actual Class : 2
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
```

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

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [186]:
```
clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weigh
t='balanced', random_state=0)
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

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

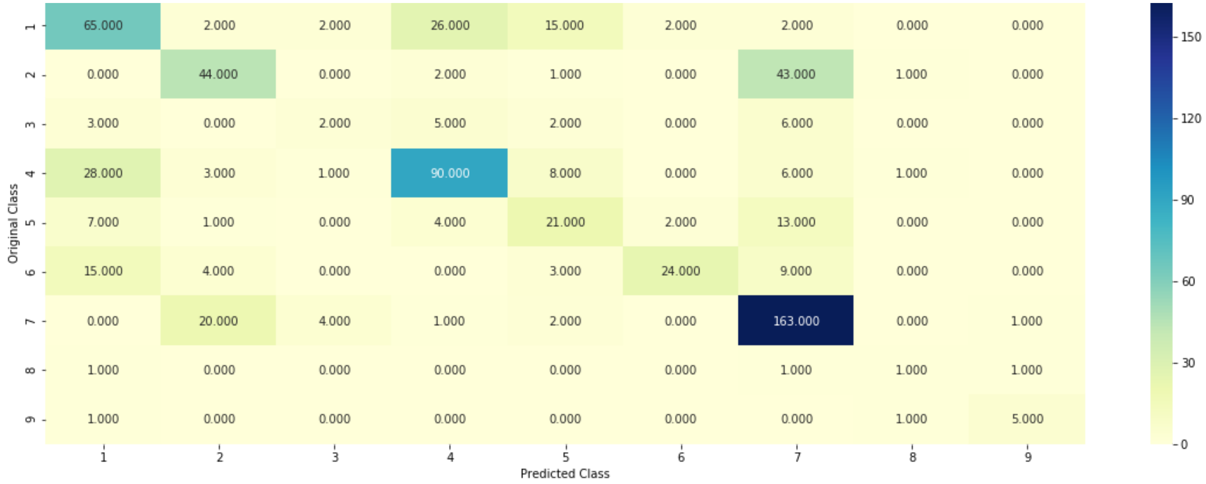
```python
clf3 = MultinomialNB(alpha=0.001)
clf3.fit(train_x_onehotCoding, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_cl
f1.predict_proba(cv_x_onehotCoding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig
_clf2.predict_proba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predic
t_proba(cv_x_onehotCoding))))
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3
], meta_classifier=lr, use_probas=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %
0.3f" % (i, log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

```
Logistic Regression :  Log Loss: 1.05
Support vector machines : Log Loss: 1.82
Naive Bayes : Log Loss: 1.20
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.177
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.029
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.488
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.154
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.339
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.732
```

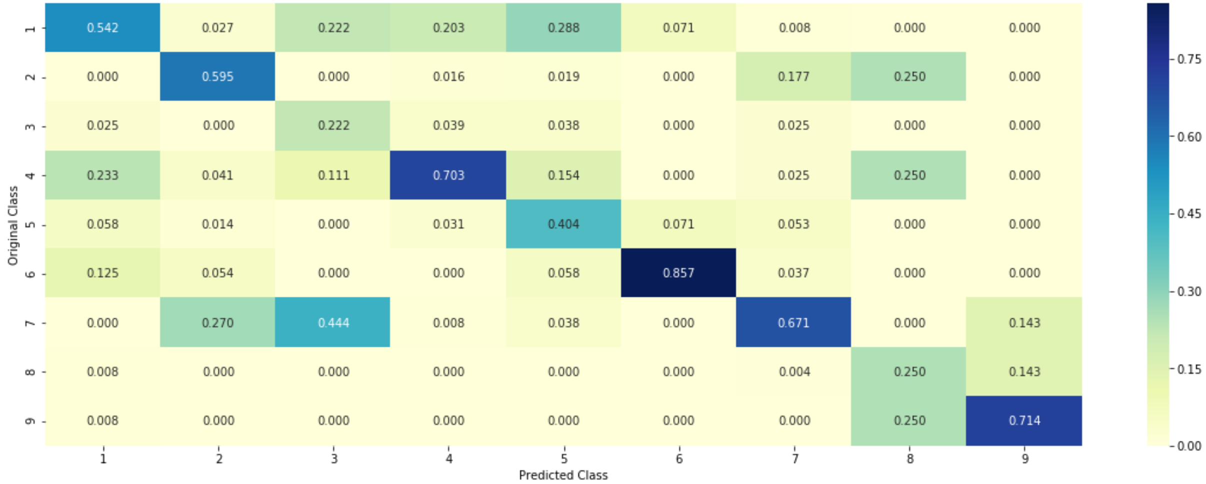
4.7.2 testing the model with the best hyper parameters

In [187]:
```
lr = LogisticRegression(C=0.1, class_weight='balanced')
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], m
eta_classifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

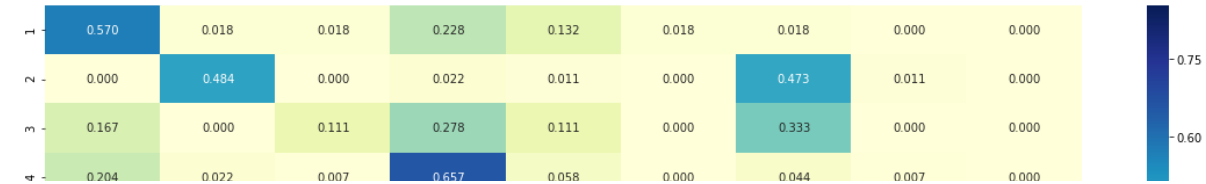
```
Log loss (train) on the stacking classifier : 0.643868285977
Log loss (CV) on the stacking classifier : 1.20059035957
Log loss (test) on the stacking classifier : 1.20125909032
Number of missclassified point : 0.37593984962406013
-------------------- Confusion matrix --------------------
```

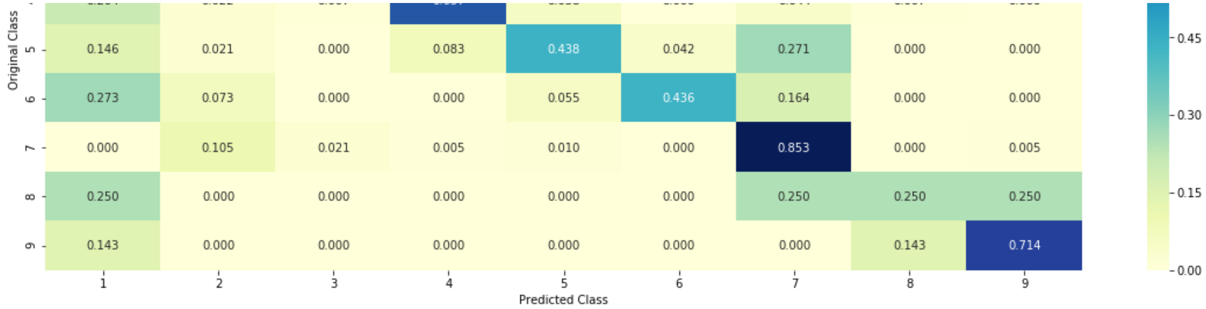


------------------- Precision matrix (Columm Sum=1) ------------------
--



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



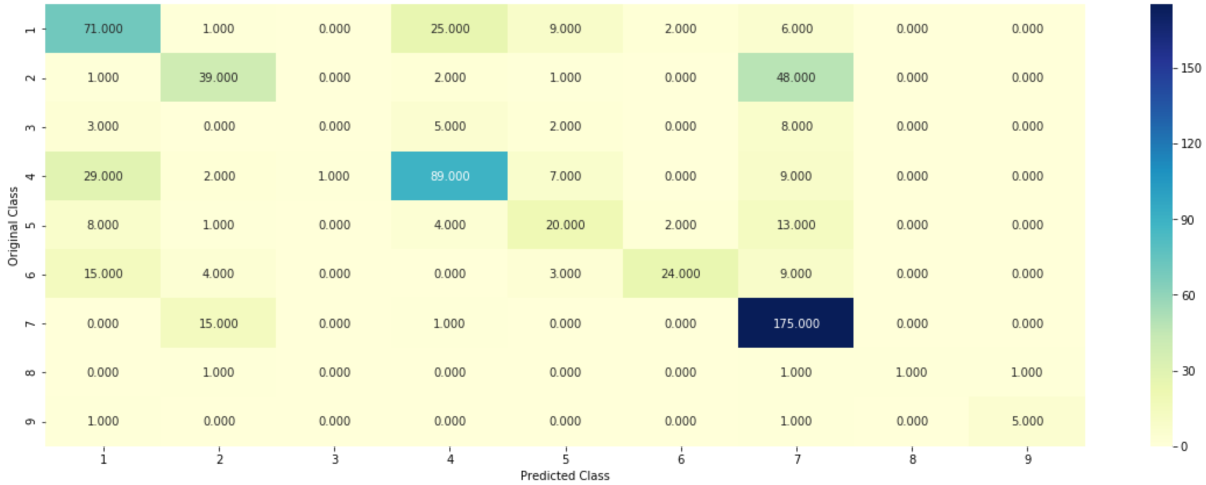


4.7.3 Maximum Voting classifier

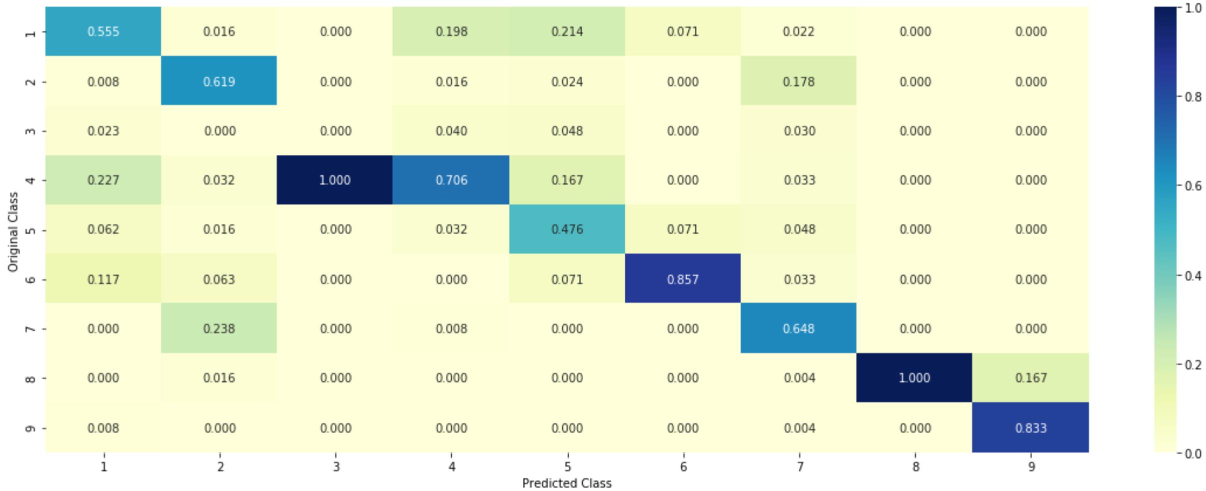
In [188]:

```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensembl
e.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, v
clf.predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.pr
edict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vcl
f.predict_proba(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predic
t(test_x_onehotCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_oneh
otCoding))
```

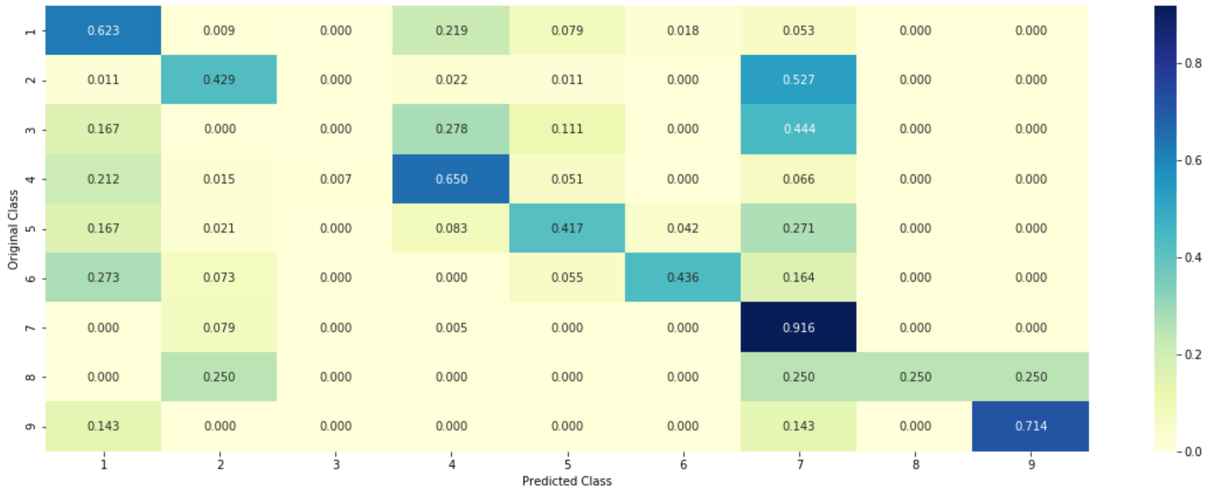
```
Log loss (train) on the VotingClassifier : 0.851746323643
Log loss (CV) on the VotingClassifier : 1.18016503889
Log loss (test) on the VotingClassifier : 1.18096199517
Number of missclassified point : 0.362406015037594
-------------------- Confusion matrix --------------------
```




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




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



Logistic Regression with CountVectorizer(bigrams and unigrams)

with class balancing

hypertunning parameter

In [189]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_d
f['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gen
e'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [190]:
```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer()
```

```
train_variation_feature_onehotCoding = variation_vectorizer.fit_transfo
rm(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(te
st_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_d
f['Variation'])
```

In [191]:

```
text_vectorizer = CountVectorizer(min_df=3,ngram_range=(1,2))
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_d
f['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

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

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

In [192]:

```
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCo
ding, axis=0)

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

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

```
cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding, axis=0)
```

In [193]:

```
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3,ngram_range = (1,2))

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

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

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

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

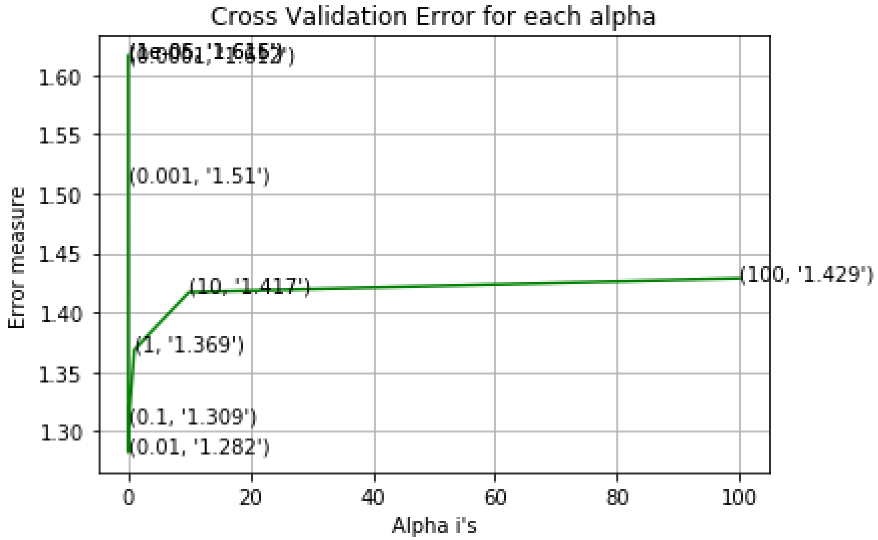
In [194]:
```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,t
rain_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,tes
t_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_vari
ation_feature_onehotCoding))

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

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

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

In [195]:
```
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_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log
-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))
```

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


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

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

```
for alpha = 1e-06
Log Loss : 1.61635025652
for alpha = 1e-05
Log Loss : 1.61505586625
for alpha = 0.0001
Log Loss : 1.61184529544
for alpha = 0.001
Log Loss : 1.510250644
for alpha = 0.01
Log Loss : 1.28218987167
```

```
for alpha = 0.1
Log Loss : 1.30872034162
for alpha = 1
Log Loss : 1.36880628341
for alpha = 10
Log Loss : 1.41718778462
for alpha = 100
Log Loss : 1.42879292956
```

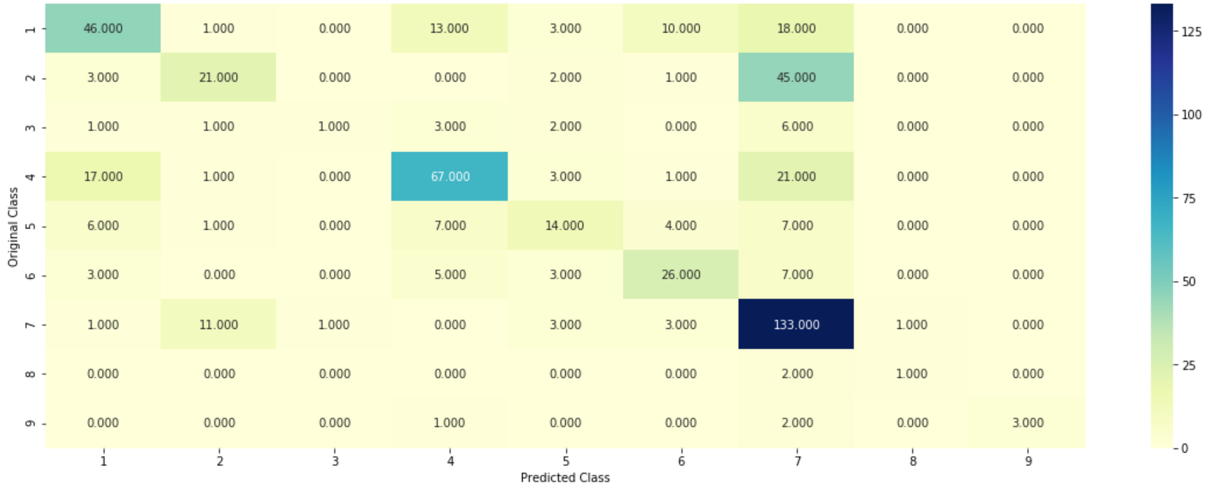



```
For values of best alpha =  0.01 The train log loss is: 0.85003282197
For values of best alpha =  0.01 The cross validation log loss is: 1.28
218987167
For values of best alpha =  0.01 The test log loss is: 1.19878581139
```

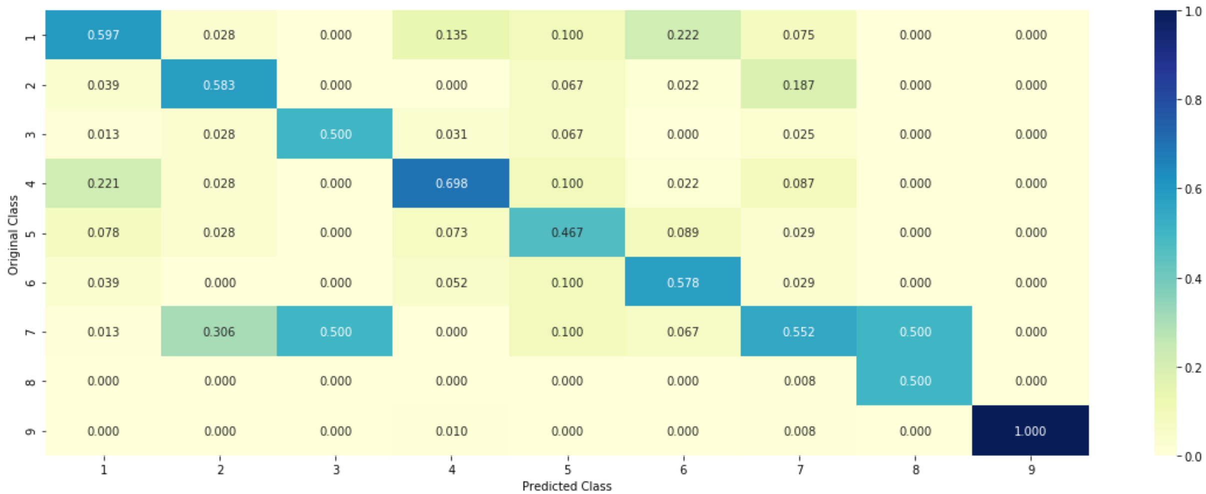
In [196]:

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_o
nehotCoding, cv_y, clf)
```

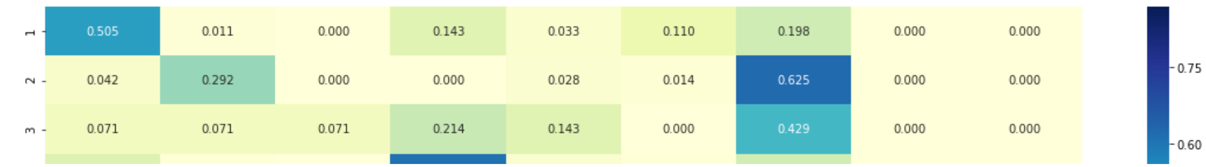
```
Log loss : 1.28218987167
Number of mis-classified points : 0.41353383458646614
-------------------- Confusion matrix --------------------
```

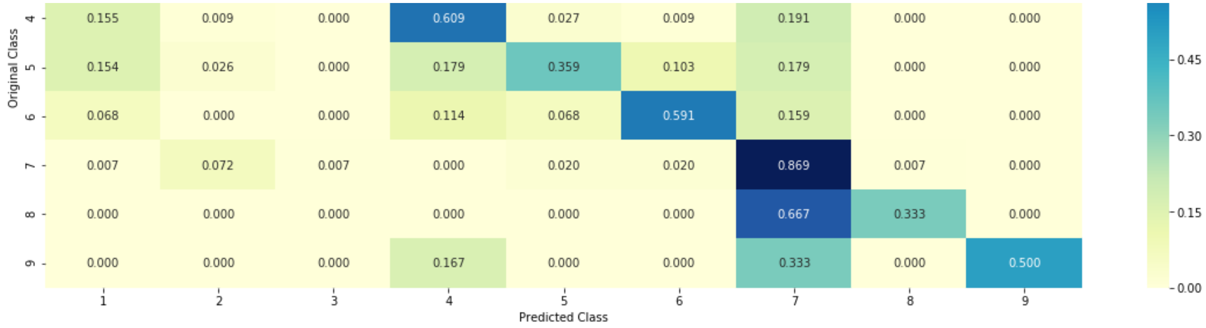


-------------------- Precision matrix (Columm Sum=1) --------------------



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





In [197]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 10
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.1859  0.1547  0.0265  0.1451  0.064
2  0.0626  0.3475  0.0062  0.0072]]
Actual Class : 7
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

In [198]:

```
test_point_index = 19
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
```

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

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.1626  0.1588  0.0247  0.1581  0.066
1  0.069   0.3485  0.0055  0.0067]]
Actual Class : 7
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

without class balancing

hypertunning parameter

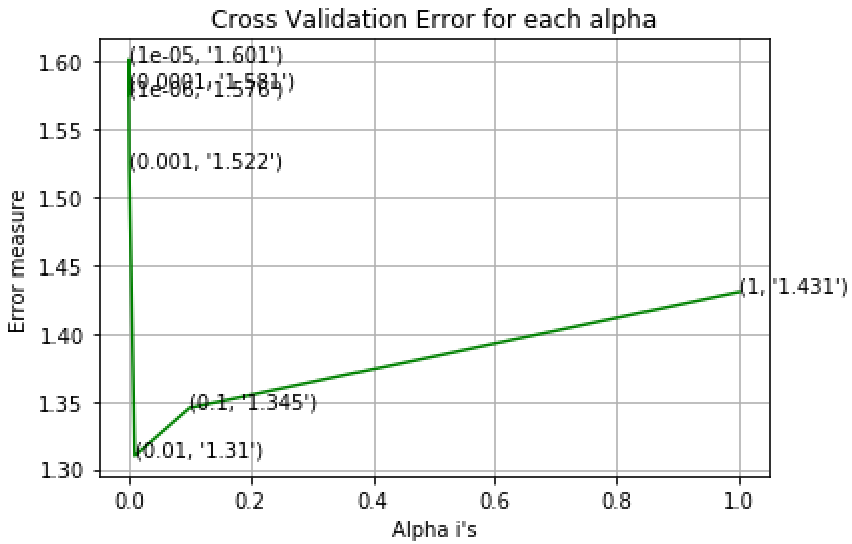
In [199]:

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

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.57583076038
for alpha = 1e-05
Log Loss : 1.60123997096
for alpha = 0.0001
Log Loss : 1.58075105516
for alpha = 0.001
Log Loss : 1.52228702657
for alpha = 0.01
Log Loss : 1.30987283581
for alpha = 0.1
Log Loss : 1.34521259861

for alpha = 1
```

```
For alpha = 1
Log Loss : 1.43059237646
```

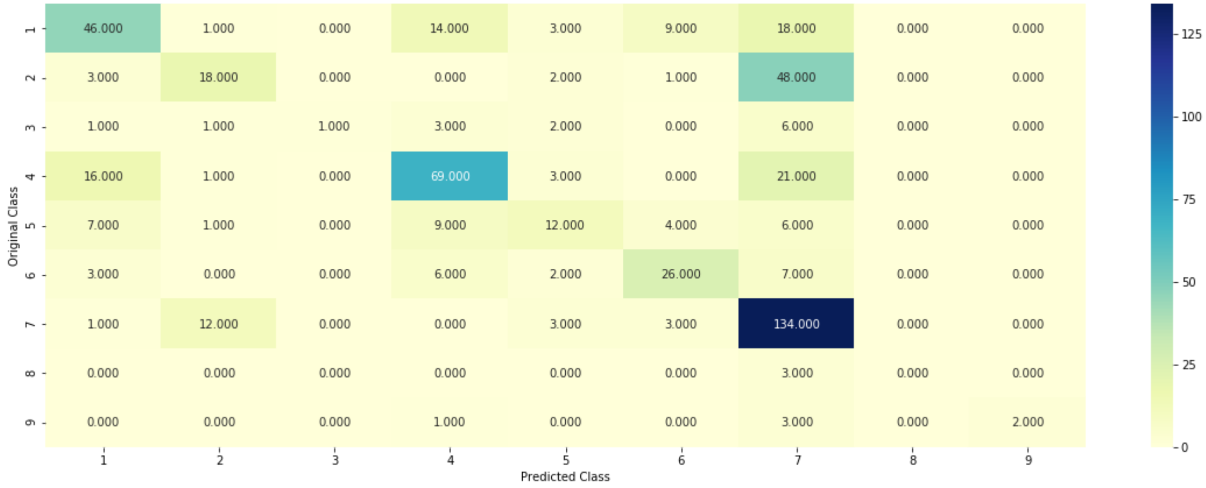



```
For values of best alpha =  0.01 The train log loss is: 0.841663367953
For values of best alpha =  0.01 The cross validation log loss is: 1.30
987283581
For values of best alpha =  0.01 The test log loss is: 1.21761367131
```

In [200]:

```
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_o
nehotCoding, cv_y, clf)
```

```
Log loss : 1.30987283581
Number of mis-classified points : 0.42105263157894735
-------------------- Confusion matrix --------------------
```




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


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

In [201]:

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

```
Predicted Class : 2
Predicted Class Probabilities: [[ 0.0704  0.4893  0.0083  0.146   0.028
2  0.0442  0.1972  0.0139  0.0024]]
Actual Class : 6
```

```
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

In [202]:

```
test_point_index = 10
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.1876  0.1592  0.0316  0.1378  0.068
4  0.0627  0.3372  0.0066  0.0089]]
Actual Class : 7
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

conclusion

1. We have applied the tfidf featurisation on text column and extracted top words.
2. we checked the this techniques on all above model to get the best log-loss.
3. we have also applied some feature engineering technique.
4. we have also applied bag of words with bigrams and checked the result on logistic regression.

Results on Models

- log-loss on Naive Bayes (best alpha = 0.0001)
  - Train = 0.560
  - CV = 1.196
  - Test = 1.209
- log-loss on KNN (best alpha = 11)
  - Train = 0.634
  - CV = 1.113
  - Test = 1.054
- log-loss on Logistic Regression- balanced (best alpha = 0.0001)
  - Train = 0.435
  - CV = 1.051
  - **Test = 0.963**
- log-loss on Logistic Regression - unbalanced (best alpha = 0.0001)
  - Train = 0.436
  - CV = 1.108
  - **Test = 0.997**
- log-loss on Linear SVM (best alpha = 0.001)
  - Train = 0.549
  - CV = 1.057
  - Test = 1.019
- log-loss on RF (best n_estimator = 2000 and max_depth = 5)
  - Train = 0.867
  - CV = 1.191
  - Test = 1.194
- log-loss on Stacking Classifier
  - Train = 0.643
  - CV = 1.200
  - Test = 1.201
- log-loss on Voting Classifier
  - Train = 0.851
  - CV = 1.1801
  - Test = 1.1809

- log-loss on Logsitic Regresstion - bigrams(best alpha = 0.01) balanced
    - Train = 0.850
    - CV = 1.282
    - Test = 1.198
- log-loss on Logsitic Regresstion - bigrams(best alpha = 0.01) unbalanced
    - Train = 0.841
    - CV = 1.309
    - Test = 1.217