Project : Personalized Cancer Diagnosis

Problem Statement:

A lot has been said during the past several years about how precision medicine and, more concretely, how genetic testing is going to disrupt the way diseases like cancer are treated.

But this is only partially happening due to the huge amount of manual work still required. Once sequenced, a cancer tumor can have thousands of genetic mutations. But the challenge is distinguishing the mutations that contribute to tumor growth (drivers) from the neutral mutations (passengers).

Currently this interpretation of genetic mutations is being done manually. This is a very time-consuming task where a clinical pathologist has to manually review and classify every single genetic mutation based on evidence from text-based clinical literature.

We need to develop a Machine Learning algorithm that, using this knowledge base as a baseline, automatically classifies genetic variations.

This problem was a competition posted on Kaggle with a award of $15,000. This was launched by Memorial Sloan Kettering Cancer Center (MSKCC), accepted by NIPS 2017 Competition Track, because we need your help to take personalized medicine to its full potential.

Analysis of the problem statement

Lets first understand the data set provided and using that dataset we will try to understand the above problem in Machine Learning world.

In [140]:

```
import pandas as pd
```

In [141]:

```
data_variants = pd.read_csv('training_variants')
# Loading training_text dataset. This is seperated by ||
data_text =pd.read_csv("training_text",sep="\|\|",engine="python",names=["ID","TEXT"],skiprows=1)
```

In [142]:

```
data_variants.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3321 entries, 0 to 3320
Data columns (total 4 columns):
ID           3321 non-null int64
Gene         3321 non-null object
Variation    3321 non-null object
Class        3321 non-null int64
dtypes: int64(2), object(2)
memory usage: 103.9+ KB
```

In [143]:

```
data_variants.head()
```

Out[143]:

	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

There are 4 fields above:

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

In [144]:

```
data_variants.describe()
```

Out[144]:

	ID	Class
count	3321.000000	3321.000000
mean	1660.000000	4.365854
std	958.834449	2.309781
min	0.000000	1.000000
25%	830.000000	2.000000
50%	1660.000000	4.000000
75%	2490.000000	7.000000
max	3320.000000	9.000000

In [145]:

```
data_text.head()
```

Out[145]:

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

So above dataset have 2 columns. ID and Text column. We can also observe column ID which is common in both the dataset.

In [146]:

```
data_text.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3321 entries, 0 to 3320
Data columns (total 2 columns):
ID      3321 non-null int64
TEXT    3316 non-null object
dtypes: int64(1), object(1)
memory usage: 52.0+ KB
```

In [147]:

```
data_variants.Variation.unique
```

Out[147]:

```
<bound method Series.unique of 0       Truncating Mutations
1                       W802*
2                       Q249E
3                       N454D
4                       L399V
5                       V391I
6                       V430M
7                    Deletion
8                       Y371H
9                       C384R
10                      P395A
11                      K382E
12                      R420Q
13                      C381A
14                      P428L
15                      D390Y
16      Truncating Mutations
17                      Q367P
18                      M374V
19                      Y371S
20                       H94Y
21                      C396R
22                      G375P
23                      S376F
24                      P417A
25                      H398Y
26                        S2G
27                      Y846C
28                      C228T
29                      H412Y
                ...
3291                    C620R
3292                    C634Y
3293                    V804G
3294                    R886W
3295                    F893L
3296                    Y791F
3297                    R177*
3298                    Y113*
3299                    R139G
3300                     K83N
3301                    R177Q
3302                    R166Q
3303                    P173S
3304                    R201Q
3305                 S70fsX93
3306                    W279*
3307    Truncating Mutations
3308                    R174*
3309                    D171G
3310             Amplification
3311       RUNX1-EVI1 Fusion
3312        TEL-RUNX1 Fusion
3313                     H78Q
3314                     G42R
3315    RUNX1-RUNX1T1 Fusion
3316                    D171N
3317                    A122*
3318                  Fusions
3319                     R80C
3320                     K83E
Name: Variation, Length: 3321, dtype: object>
```

In [148]:

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
```

```
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV


from mlxtend.classifier import StackingClassifier

from sklearn import model_selection
from sklearn.linear_model import LogisticRegression

import math
from sklearn.metrics import normalized_mutual_info_score
```

In [149]:

```
from sklearn.ensemble import RandomForestClassifier
```

This is descrete data so it is classification problem and since there are multiple descrete output possible so we can call it Multi class classification problem

Important note :

This is medical related problem so correct results are very important. Error can be really costly here so we would like to have result for each class in terms of Probablity. We might not be much bothered about time taken by ML algorithm as far as it is reasonable.

We also want our model to be highly interpritable because a medical practitionar want to also give proper reasonining on why ML algorithm is predicting any class.

We will evaluate our model using Confution matrix and Multi class log-loss

Ok, now we understood the problem statement. Let's work on the solution.

In [150]:

```
stop_words = set(stopwords.words('english'))
```

In [151]:

```
def data_text_preprocess(total_text, ind, col):
    # Remove int values from text data as that might not be imp
    if type(total_text) is not int:
        string = ""
        # replacing all special char with space
        total_text = re.sub('[^a-zA-Z0-9\n]', ' ', str(total_text))
        # replacing multiple spaces with single space
        total_text = re.sub('\s+',' ', str(total_text))
        # bring whole text to same lower-case scale.
        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 text
            if not word in stop_words:
                string += word + " "
```

```
        data_text[col][ind] = string
```

In [152]:

```
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        data_text_preprocess(row['TEXT'], index, 'TEXT')
```

Let's merge both the dataset. Remember that ID was common column. So lets use it to merge.

In [153]:

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

Out[153]:

	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...

It's very important to look for missing values. Else they create problem in final analysis.

In [154]:

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

Out[154]:

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

We can see many rows with missing data. Now the question is what to do with this missing value. One way could be that we can drop these rows having missing values or we can do some imputation in it. Let's go with imputation only. But question is what to impute here :

How about merging Gene and Variation column. Let's do it:

In [155]:

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

In [156]:

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

Out[156]:

	ID	Gene	Variation	Class	TEXT

Creating Training, Test and Validation data

Before we split the data into taining, test and validation data set. We want to ensure that all spaces in Gene and Variation column to be replaced by _ (underscore).

In [157]:

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

So, we can now start our split process in train, test and validation data set.

In [158]:

```
# Splitting the data into train and test set
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true, test_size=0.2
)
# split the train data now into train validation and cross validation
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test_size=0.2
)
```

In [159]:

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

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

In [160]:

```
import warnings
warnings.filterwarnings('ignore')

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()
```

Let's look at the distribution of data in train, test and validation set.

In [161]:

```
train_class_distribution
```

Out[161]:

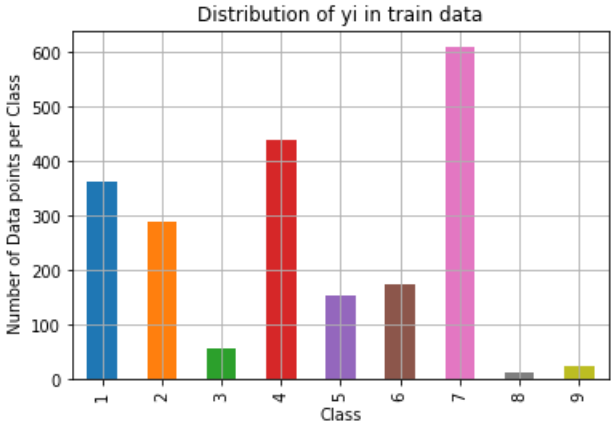
```
1    363
2    289
3     57
4    439
5    155
6    176
7    609
8     12
9     24
Name: Class, dtype: int64
```

So, what does above variable suggest us. This means in my train dataset we have class 1 values with count of 363, class 2 values having count of 289 and so on. It will be better idea to visualise it in graph format.

Visualizing for train class distribution:

In [162]:

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



Distribution in form of percentage:

In [163]:

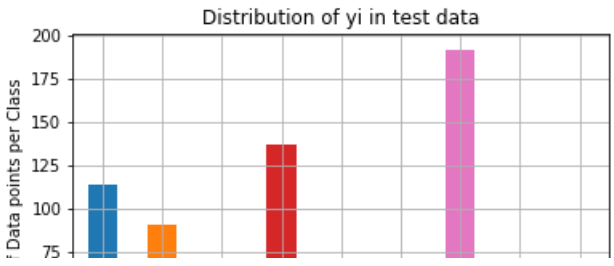
```
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distribution.values[i],
          '(', np.round((train_class_distribution.values[i]/train_df.shape[0]*100), 3), '%)')
```

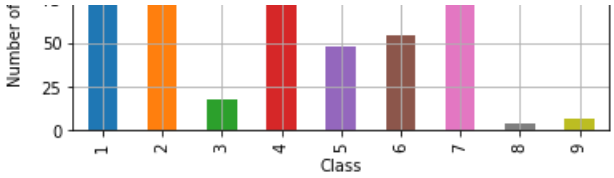
```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298 %)
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 %)
```

Testset

In [164]:

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





In [165]:

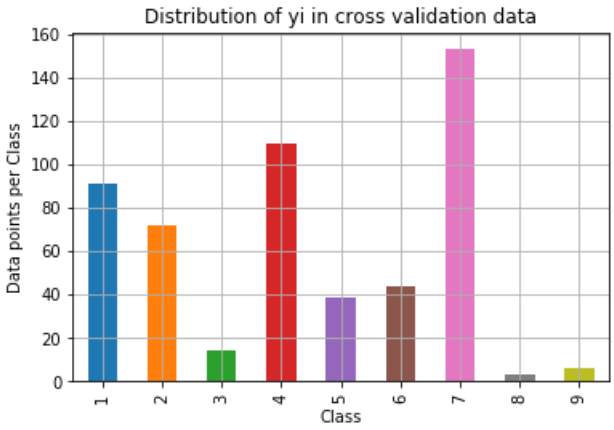
```
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.values[i],
          '(', np.round((test_class_distribution.values[i]/test_df.shape[0]*100), 3), '%)')
```

```
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 %)
```

CV Set

In [166]:

```
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()
```



In [167]:

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

```
Number of data points in class 7 : 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 %)
```

Now question is because we need log-loss as final evaluation metrics how do we say that model we are going to build will be good model. For doing this we will build a random model and will evaluate log loss. Our model should return lower log loss value than this.

Building a Random model:

So we need to generate 9 random numbers because we have 9 class such that their sum must be equal to 1 because sum of Probablity of all 9 classes must be equivalent to 1.

In [168]:

```
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]
test_data_len,cv_data_len
```

Out[168]:

```
(665, 532)
```

In [169]:

```
# 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))
```

```
Log loss on Cross Validation Data using Random Model 2.5215675494758534
```

In [170]:

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

```
Log loss on Test Data using Random Model 2.442413820625655
```

In [171]:

```
# Lets get the index of max probablity
predicted_y =np.argmax(test_predicted_y, axis=1)
```

In [172]:

```
# Lets see the output. these will be 665 values present in test dataset
predicted_y
```

Out[172]:

```
array([5, 5, 6, 8, 6, 6, 0, 3, 3, 4, 1, 0, 0, 1, 2, 8, 4, 4, 1, 7, 5, 6,
       3, 7, 3, 4, 1, 1, 3, 3, 4, 8, 1, 4, 6, 1, 6, 4, 2, 2, 2, 5, 1, 5,
       7, 8, 2, 7, 8, 4, 3, 8, 1, 7, 5, 6, 7, 1, 3, 6, 8, 2, 6, 8, 2, 0,
       7, 0, 2, 1, 0, 6, 1, 6, 6, 0, 3, 1, 6, 5, 3, 1, 5, 5, 6, 0, 0, 1,
       2, 5, 1, 7, 2, 6, 0, 5, 0, 8, 3, 8, 8, 3, 5, 1, 8, 8, 4, 5, 6, 3,
       3, 2, 1, 3, 3, 0, 0, 3, 2, 5, 4, 2, 2, 3, 0, 3, 1, 6, 8, 0, 8, 0,
       7, 4, 2, 7, 7, 1, 4, 3, 3, 1, 0, 6, 0, 2, 4, 7, 5, 6, 5, 0, 6, 3,
       7, 0, 3, 5, 8, 1, 0, 2, 4, 4, 2, 4, 6, 3, 8, 6, 0, 2, 0, 4, 7, 7,
       5, 5, 4, 2, 4, 6, 3, 6, 5, 4, 5, 1, 3, 7, 8, 8, 6, 2, 5, 3, 7, 2,
       8, 1, 7, 6, 7, 5, 5, 2, 5, 4, 0, 7, 2, 6, 2, 5, 5, 4, 4, 2, 4, 2,
       4, 7, 3, 8, 4, 1, 4, 4, 2, 5, 7, 8, 0, 3, 8, 8, 3, 5, 4, 5, 8, 0,
       2, 6, 7, 1, 2, 3, 5, 3, 6, 5, 0, 7, 1, 0, 1, 7, 1, 3, 7, 3, 0, 1,
       5, 1, 8, 3, 4, 8, 3, 3, 1, 4, 5, 6, 0, 7, 2, 7, 7, 6, 7, 2, 5, 0,
       2, 1, 7, 4, 1, 3, 0, 2, 2, 2, 0, 8, 7, 0, 4, 7, 3, 3, 8, 0, 8, 2,
```

```
1, 1, 6, 8, 8, 1, 1, 8, 7, 5, 0, 1, 6, 1, 0, 5, 5, 6, 0, 0, 7, 3,
6, 1, 8, 2, 5, 1, 3, 8, 4, 1, 8, 0, 0, 5, 8, 0, 6, 7, 2, 7, 5, 1,
8, 1, 5, 1, 8, 8, 6, 3, 5, 4, 1, 0, 5, 7, 6, 4, 0, 4, 5, 6, 7, 1,
4, 2, 2, 0, 4, 1, 6, 3, 7, 0, 0, 7, 2, 3, 7, 5, 1, 5, 6, 3, 2, 5,
3, 2, 4, 1, 5, 8, 8, 4, 8, 4, 1, 1, 8, 3, 3, 2, 0, 8, 3, 6, 1, 8,
1, 7, 8, 0, 7, 7, 4, 0, 0, 3, 7, 0, 2, 8, 3, 3, 6, 4, 6, 3, 0, 2,
0, 2, 0, 3, 4, 3, 3, 8, 3, 8, 0, 2, 5, 5, 6, 5, 5, 2, 4, 2, 8, 0,
3, 1, 6, 0, 8, 3, 4, 8, 2, 3, 8, 8, 5, 5, 0, 5, 0, 4, 4, 0, 6, 8,
6, 7, 2, 8, 0, 2, 3, 2, 5, 5, 1, 8, 1, 2, 0, 2, 1, 3, 1, 4, 1, 8,
7, 7, 4, 6, 5, 8, 2, 6, 7, 3, 3, 7, 3, 2, 8, 0, 0, 3, 1, 1, 8, 4,
1, 6, 7, 8, 4, 6, 1, 2, 6, 5, 6, 2, 7, 7, 1, 1, 0, 3, 6, 0, 5, 2,
7, 3, 8, 0, 1, 6, 7, 2, 8, 1, 2, 1, 1, 8, 6, 8, 1, 3, 3, 5, 3, 6,
2, 7, 7, 3, 3, 2, 8, 4, 3, 5, 4, 4, 2, 4, 0, 2, 7, 3, 8, 4, 7, 2,
0, 2, 7, 3, 6, 8, 4, 3, 5, 7, 2, 1, 6, 8, 2, 7, 0, 4, 1, 0, 7, 7,
4, 3, 6, 2, 1, 0, 1, 4, 8, 3, 8, 3, 8, 8, 7, 5, 5, 1, 5, 8, 0, 3,
7, 3, 8, 6, 6, 1, 0, 8, 7, 2, 6, 3, 5, 6, 2, 5, 8, 0, 7, 5, 4, 0,
4, 2, 3, 6, 2], dtype=int64)
```

So you can see the index value ranging from 0 to 8. So, lets make it as 1 to 9 we will increase this value by 1.

In [173]:

```
predicted_y = predicted_y + 1
```

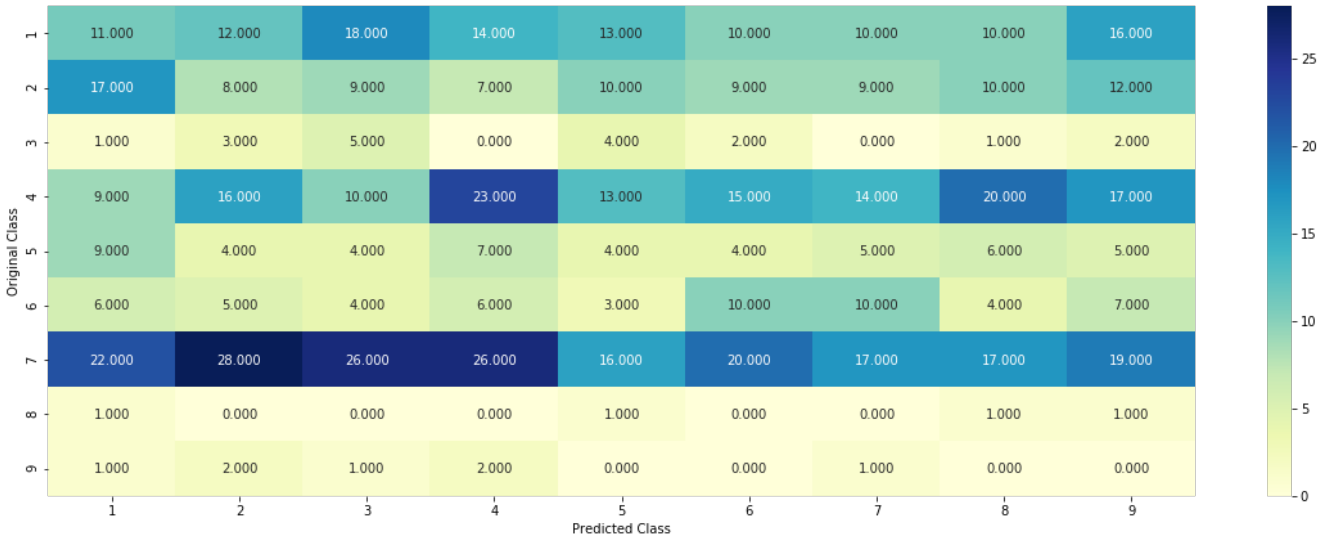
Confusion Matrix

In [174]:

```
C = confusion_matrix(y_test, predicted_y)
```

In [175]:

```
labels = [1,2,3,4,5,6,7,8,9]
plt.figure(figsize=(20,7))
sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
```



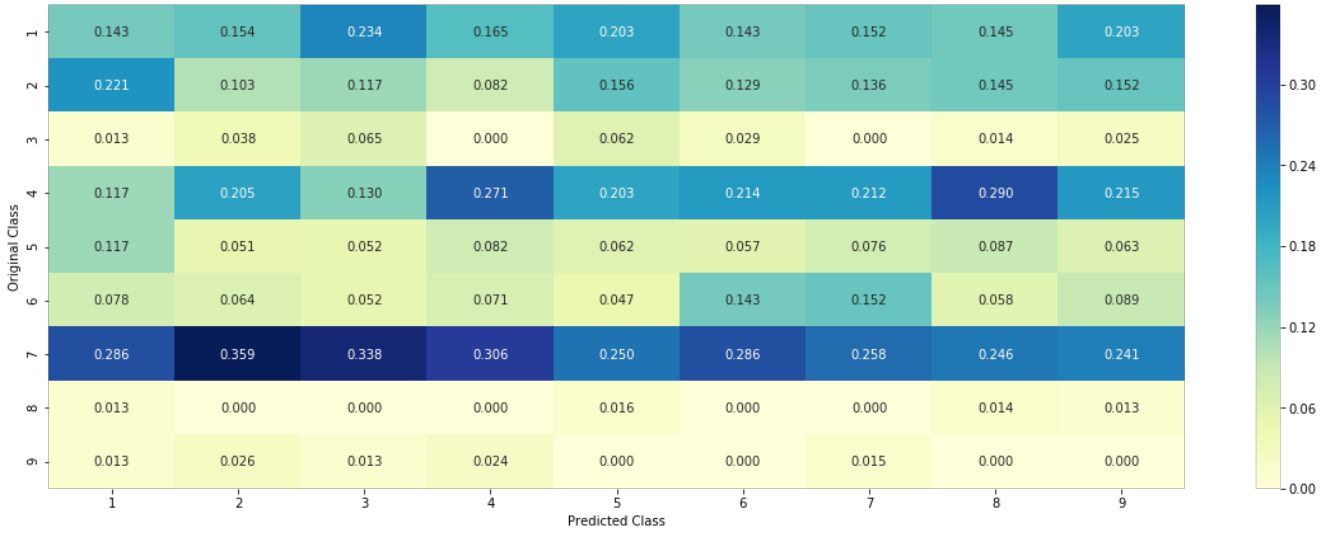
Precision matrix:

In [176]:

```
B =(C/C.sum(axis=0))
```

In [177]:

```
plt.figure(figsize=(20,7))
sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
```



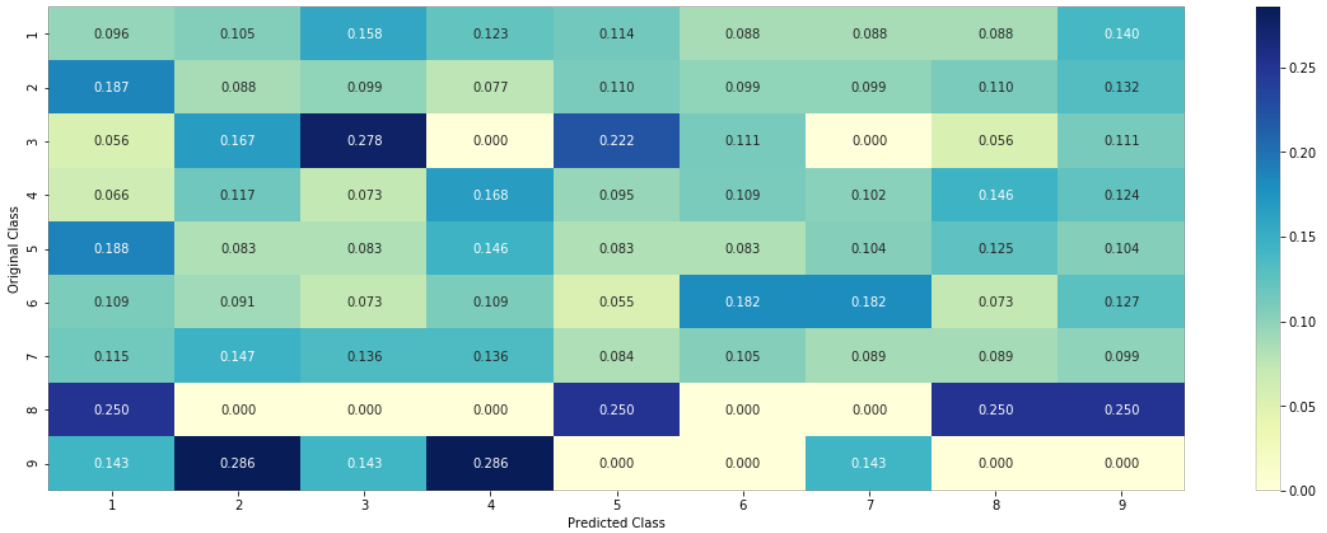
Recall matrix:

In [178]:

```
A =(((C.T)/(C.sum(axis=1))).T)
```

In [179]:

```
plt.figure(figsize=(20,7))
sns.heatmap(A, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
```



Evaluating Gene Column:

Now we will look at each independent column to make sure its relavent for my target variable but the question is, how? Let's understand with our first column Gene which is categorial in nature.

In [180]:

```
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    190
TP53      95
EGFR      85
BRCA2     80
PTEN      70
BRAF      66
KIT       64
ALK       46
ERBB2     44
FLT3      41
Name: Gene, dtype: int64
```

unique values present in gene

In [181]:

```
unique_genes.shape[0]
```

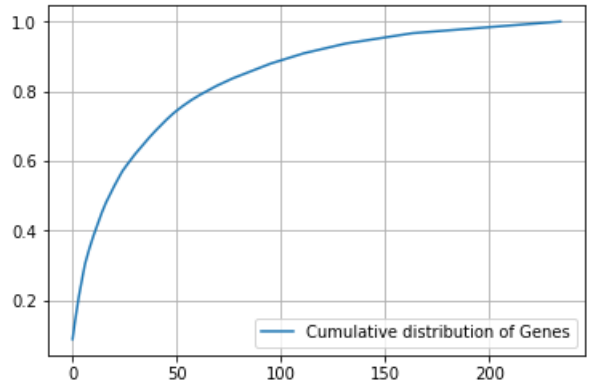
Out[181]:

```
235
```

Cumulative distribution of unique Genes values

In [182]:

```
s = sum(unique_genes.values);
h = unique_genes.values/s;
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



So, now we need to convert these categorical variable to appropirate format which my machine learning algorithm will be able to take as an input.

So we have 2 techniques to deal with it.

```
One-hot encoding
Response Encoding (Mean imputation)
```

Let's use both of them to see which one work the best. So lets start encoding using one hot encoder

In [183]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
```

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

In [184]:

```
train_gene_feature_onehotCoding.shape
```

Out[184]:

```
(2124, 234)
```

In [185]:

```
#column names after one-hot encoding for Gene column
gene_vectorizer.get_feature_names()
```

Out[185]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid5b',
 'asxl1',
 'atm',
 'atr',
 'aurka',
 'aurkb',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk6',
 'cdkn1a',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cdkn2c',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctla4',
 'ctnnb1',
 'ddr2',
 'dicer1',
 'dnmt3a',
 'dnmt3b',
 'dusp4',
```

```
'egfr',
'eif1ax',
'elf3',
'ep300',
'epas1',
'epcam',
'erbb2',
'erbb3',
'erbb4',
'ercc2',
'ercc3',
'ercc4',
'erg',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fam58a',
'fanca',
'fat1',
'fbxw7',
'fgf19',
'fgf3',
'fgf4',
'fgfr1',
'fgfr2',
'fgfr3',
'fgfr4',
'flt1',
'flt3',
'foxa1',
'foxl2',
'foxo1',
'foxp1',
'fubp1',
'gata3',
'gna11',
'gnaq',
'gnas',
'h3f3a',
'hist1h1c',
'hla',
'hnf1a',
'hras',
'idh1',
'idh2',
'igf1r',
'ikbke',
'ikzf1',
'il7r',
'jak1',
'jak2',
'jun',
'kdm5c',
'kdm6a',
'kdr',
'keap1',
'kit',
'kmt2a',
'kmt2c',
'knstrn',
'kras',
'lats1',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
'mapk1',
'mdm2',
'mdm4',
'med12',
'mef2b',
'men1',
'met',
'mga',
'mlh1',
```

```
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'mycn',
'myd88',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',
'nkx2',
'notch1',
'notch2',
'npm1',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'nup93',
'pak1',
'pax8',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pik3r3',
'pim1',
'pms2',
'pole',
'ppm1d',
'ppp2r1a',
'ppp6c',
'prdm1',
'ptch1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rab35',
'rac1',
'rad21',
'rad50',
'rad51b',
'rad51c',
'rad51d',
'rad54l',
'raf1',
'rara',
'rasa1',
'rb1',
'rbm10',
'ret',
'rhoa',
'rit1',
'rnf43',
'ros1',
'rras2',
'runx1',
'rxra',
'rybp',
'sdhb',
'sdhc',
'setd2',
'sf3b1',
'shoc2',
'shq1',
'smad2',
'smad3',
'smad4',
'smarca4',
'smo',
```

```
 'sos1',
 'sox9',
 'spop',
 'src',
 'srsf2',
 'stag2',
 'stat3',
 'stk11',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'whsc1',
 'whsc1l1',
 'xpo1',
 'yap1']
```

Response encoding columns for Gene column

In [186]:

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

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

```
    # ...
    # }
    value_count = train_df[feature].value_counts()

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

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

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

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.0378787878787878, 0.068181818181818177,
0.13636363636363635, 0.25, 0.19318181818181818, 0.0378787878787878, 0.0378787878787878,
0.0378787878787878],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366,
0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408
163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177,
0.068181818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608,
0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608,
0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917,
0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081
761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295,
0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702,
0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334,
0.073333333333333334, 0.093333333333333338, 0.080000000000000002, 0.29999999999999999,
0.066666666666666666, 0.066666666666666666],
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

In [187]:

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

In [188]:

```
train_gene_feature_responseCoding.shape
```

Out[188]:

```
(2124, 9)
```

Now, question is how good is Gene column feature to predict my 9 classes. One idea could be that we will build model having only gene column with one hot encoder with simple model like Logistic regression. If log loss with only one column Gene comes out to be better than random model, than this feature is important.

In [189]:

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

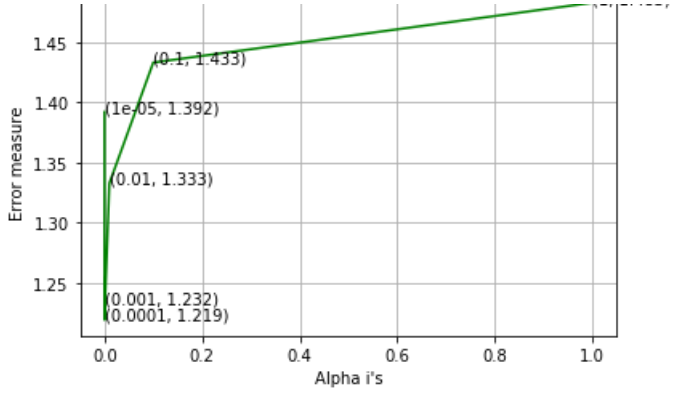
In [190]:

```
# We will be using SGD classifier
# http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# We will also be using Calibrated Classifier to get the result into probablity format t be used f
or log loss
import warnings
warnings.filterwarnings('ignore')
cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_gene_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.clas
ses_, eps=1e-15))
```

```
For values of alpha =  1e-05 The log loss is: 1.392194358482969
For values of alpha =  0.0001 The log loss is: 1.2193389192411614
For values of alpha =  0.001 The log loss is: 1.232401394919295
For values of alpha =  0.01 The log loss is: 1.3328727540964693
For values of alpha =  0.1 The log loss is: 1.4332725904007557
For values of alpha =  1 The log loss is: 1.482992658949991
```

In [191]:

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



In [192]:

```
# Lets use best alpha value as we can see from above graph and compute log loss
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_gene_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_gene_feature_onehotCoding, y_train)

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

```
For values of best alpha =  0.0001 The train log loss is: 1.039994274353958
For values of best alpha =  0.0001 The cross validation log loss is: 1.2193389192411614
For values of best alpha =  0.0001 The test log loss is: 1.2075505186176765
```

Now lets check how many values are overlapping between train, test or between CV and train

In [193]:

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

In [194]:

```
print('1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/test_df.shape[
0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.s
hape[0])*100)
```

```
1. In test data 642 out of 665 : 96.54135338345866
2. In cross validation data 517 out of  532 : 97.18045112781954
```

Evaluating Variation column

Variation is also a categorical variable so we have to deal in same way like we have done for Gene column. We will again get the one hot encoder and response enoding variable for variation column.

In [195]:

```
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))
```

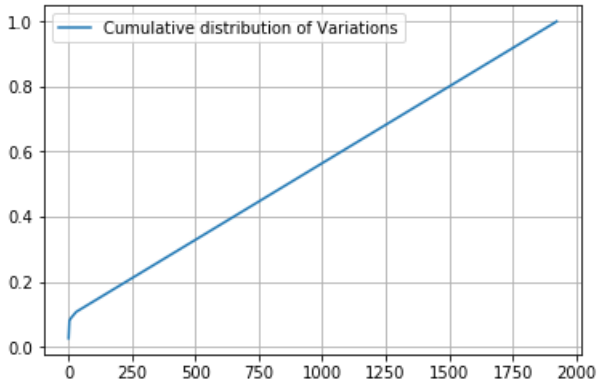
```
Number of Unique Variations : 1925
Truncating Mutations    57
```

```
Deletion                 50
Amplification            45
Fusions                  21
T58I                      3
Q61R                      3
Q61H                      3
Overexpression            3
G13V                      2
G12C                      2
Name: Variation, dtype: int64
```

In [196]:

```
s = sum(unique_variations.values);
h = unique_variations.values/s;
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.02683616 0.05037665 0.07156309 ... 0.99905838 0.99952919 1.        ]
```



In [197]:

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

In [198]:

```
train_variation_feature_onehotCoding.shape
```

Out[198]:

```
(2124, 1961)
```

Generate response encoding for the same.

In [199]:

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

In [200]:

```
train_variation_feature_responseCoding.shape
```

Out[200]:

```
(2124, 9)
```

Build the model with only column name of variation column

In [201]:

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

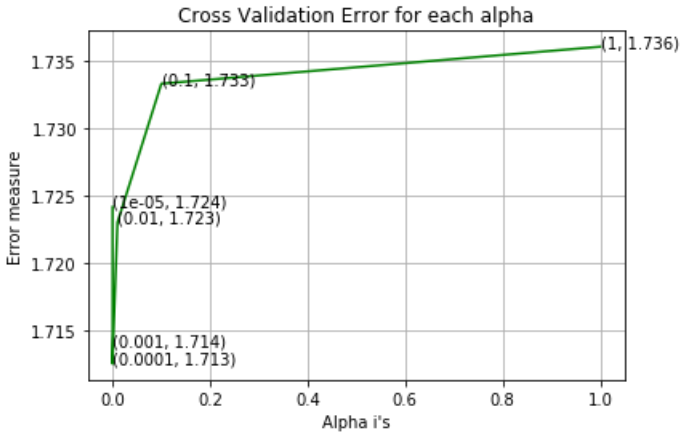
In [202]:

```
# We will be using SGD classifier
# http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# We will also be using Calibrated Classifier to get the result into probablity format t be used f
or log loss
cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_variation_feature_onehotCoding, y_train)

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

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

```
For values of alpha =  1e-05 The log loss is: 1.724183882592712
For values of alpha =  0.0001 The log loss is: 1.7125148640461212
For values of alpha =  0.001 The log loss is: 1.7138784423000655
For values of alpha =  0.01 The log loss is: 1.7230354671365309
For values of alpha =  0.1 The log loss is: 1.7333536748921587
For values of alpha =  1 The log loss is: 1.7361025287007095
```

In [203]:

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



In [204]:

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

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

```
For values of best alpha =  0.0001 The train log loss is: 0.7351459225938518
For values of best alpha =  0.0001 The cross validation log loss is: 1.7125148640461212
For values of best alpha =  0.0001 The test log loss is: 1.7065906179187307
```

In [205]:

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

In [206]:

```
print('1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/test_df.shape[
0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.s
hape[0])*100)
```

```
1. In test data 71 out of 665 : 10.676691729323307
2. In cross validation data 53 out of  532 : 9.962406015037594
```

Evaluating Text column:

In [207]:

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

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

In [208]:

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

In [209]:

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

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

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


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

```
Total number of unique words in train data : 53261
```

In [210]:

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

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


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

In [211]:

```
#response encoding 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 [212]:

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

In [213]:

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

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

# we use the same vectorizer that was trained on train data
```

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

In [214]:

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

In [215]:

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

```
Counter({3: 5092, 4: 3712, 5: 2841, 6: 2645, 7: 2031, 9: 1881, 8: 1740, 10: 1715, 12: 1536, 11: 113
1, 15: 1029, 14: 870, 13: 831, 16: 737, 18: 679, 17: 676, 24: 646, 20: 644, 19: 548, 21: 505, 22:
456, 23: 449, 30: 383, 28: 354, 41: 347, 26: 345, 25: 342, 27: 318, 32: 283, 36: 281, 29: 277, 33:
273, 35: 271, 55: 269, 31: 268, 40: 251, 38: 233, 34: 220, 42: 208, 37: 204, 39: 190, 45: 188, 43:
179, 48: 176, 44: 172, 46: 166, 60: 155, 50: 155, 56: 152, 49: 152, 51: 146, 47: 141, 54: 139, 57:
136, 52: 132, 58: 130, 53: 130, 76: 120, 64: 120, 61: 119, 66: 114, 63: 113, 65: 112, 59: 110, 72:
107, 81: 98, 80: 98, 70: 98, 77: 97, 67: 94, 68: 90, 78: 89, 69: 88, 62: 86, 75: 82, 73: 80, 74:
79, 71: 79, 90: 74, 98: 72, 84: 70, 85: 69, 92: 68, 105: 65, 99: 65, 86: 65, 83: 65, 91: 64, 82:
64, 79: 63, 100: 62, 88: 62, 101: 61, 93: 61, 89: 60, 110: 59, 87: 58, 120: 57, 114: 55, 107: 55,
95: 55, 94: 53, 108: 52, 104: 52, 103: 52, 96: 51, 109: 50, 112: 49, 115: 48, 97: 48, 141: 45,
102: 45, 139: 44, 136: 42, 118: 42, 133: 41, 171: 40, 127: 40, 124: 40, 111: 40, 106: 40, 125: 39,
116: 39, 113: 39, 134: 38, 126: 37, 117: 37, 143: 36, 129: 36, 121: 36, 144: 35, 130: 35, 119: 35,
123: 34, 122: 34, 165: 33, 152: 33, 153: 32, 150: 32, 148: 32, 140: 32, 170: 31, 169: 31, 135: 31,
131: 31, 128: 31, 172: 30, 160: 30, 149: 30, 145: 30, 132: 30, 194: 29, 146: 29, 200: 28, 181: 28,
164: 28, 163: 28, 147: 28, 137: 28, 187: 27, 161: 27, 159: 27, 158: 27, 156: 27, 138: 27, 174: 26,
154: 26, 151: 26, 198: 25, 173: 25, 168: 25, 157: 25, 142: 25, 225: 24, 197: 24, 162: 24, 223: 23,
213: 23, 207: 23, 196: 23, 180: 23, 176: 23, 166: 23, 155: 23, 230: 22, 221: 22, 209: 22, 199: 22,
185: 22, 182: 22, 175: 22, 254: 21, 220: 21, 204: 21, 192: 21, 189: 21, 184: 21, 253: 20, 250: 20,
202: 20, 183: 20, 237: 19, 228: 19, 217: 19, 216: 19, 206: 19, 203: 19, 195: 19, 193: 19, 179: 19,
167: 19, 279: 18, 264: 18, 243: 18, 226: 18, 208: 18, 205: 18, 188: 18, 311: 17, 300: 17, 260: 17,
259: 17, 246: 17, 240: 17, 231: 17, 218: 17, 215: 17, 191: 17, 186: 17, 328: 16, 269: 16, 251: 16,
239: 16, 224: 16, 212: 16, 363: 15, 309: 15, 289: 15, 286: 15, 280: 15, 275: 15, 273: 15, 272: 15,
244: 15, 242: 15, 233: 15, 229: 15, 222: 15, 177: 15, 344: 14, 332: 14, 331: 14, 318: 14, 301: 14,
297: 14, 285: 14, 267: 14, 262: 14, 249: 14, 234: 14, 201: 14, 333: 13, 316: 13, 313: 13, 294: 13,
278: 13, 277: 13, 265: 13, 258: 13, 257: 13, 252: 13, 247: 13, 245: 13, 219: 13, 214: 13, 190: 13,
379: 12, 375: 12, 364: 12, 343: 12, 317: 12, 315: 12, 295: 12, 282: 12, 281: 12, 274: 12, 271: 12,
268: 12, 261: 12, 248: 12, 241: 12, 238: 12, 235: 12, 232: 12, 211: 12, 352: 11, 345: 11, 340: 11,
327: 11, 324: 11, 308: 11, 303: 11, 263: 11, 236: 11, 227: 11, 210: 11, 583: 10, 411: 10, 401: 10,
398: 10, 390: 10, 368: 10, 365: 10, 361: 10, 351: 10, 346: 10, 325: 10, 314: 10, 305: 10, 299: 10,
298: 10, 256: 10, 255: 10, 178: 10, 524: 9, 509: 9, 505: 9, 489: 9, 472: 9, 443: 9, 441: 9, 418: 9
, 417: 9, 414: 9, 403: 9, 396: 9, 373: 9, 372: 9, 371: 9, 360: 9, 359: 9, 353: 9, 342: 9, 334: 9,
326: 9, 320: 9, 310: 9, 306: 9, 304: 9, 293: 9, 292: 9, 291: 9, 290: 9, 276: 9, 270: 9, 724: 8,
704: 8, 605: 8, 575: 8, 542: 8, 533: 8, 515: 8, 512: 8, 504: 8, 486: 8, 470: 8, 468: 8, 458: 8,
457: 8, 456: 8, 455: 8, 453: 8, 446: 8, 444: 8, 433: 8, 431: 8, 416: 8, 413: 8, 384: 8, 362: 8,
358: 8, 355: 8, 349: 8, 341: 8, 336: 8, 335: 8, 330: 8, 323: 8, 321: 8, 319: 8, 287: 8, 284: 8,
283: 8, 266: 8, 682: 7, 632: 7, 572: 7, 497: 7, 484: 7, 475: 7, 461: 7, 440: 7, 437: 7, 426: 7,
425: 7, 420: 7, 399: 7, 397: 7, 386: 7, 380: 7, 378: 7, 376: 7, 369: 7, 366: 7, 350: 7, 348: 7,
339: 7, 307: 7, 302: 7, 296: 7, 288: 7, 899: 6, 761: 6, 736: 6, 718: 6, 668: 6, 663: 6, 653: 6,
649: 6, 641: 6, 636: 6, 630: 6, 615: 6, 582: 6, 567: 6, 564: 6, 554: 6, 526: 6, 518: 6, 507: 6,
503: 6, 502: 6, 501: 6, 495: 6, 481: 6, 477: 6, 476: 6, 466: 6, 460: 6, 429: 6, 423: 6, 422: 6,
419: 6, 412: 6, 409: 6, 408: 6, 400: 6, 393: 6, 392: 6, 391: 6, 389: 6, 388: 6, 383: 6, 377: 6,
367: 6, 1636: 5, 1374: 5, 1150: 5, 973: 5, 958: 5, 923: 5, 891: 5, 874: 5, 834: 5, 809: 5, 803: 5,
801: 5, 789: 5, 778: 5, 774: 5, 773: 5, 766: 5, 764: 5, 748: 5, 746: 5, 740: 5, 738: 5, 730: 5,
694: 5, 681: 5, 676: 5, 674: 5, 656: 5, 651: 5, 646: 5, 645: 5, 644: 5, 638: 5, 633: 5, 628: 5,
617: 5, 613: 5, 594: 5, 588: 5, 581: 5, 580: 5, 573: 5, 569: 5, 563: 5, 556: 5, 549: 5, 536: 5,
528: 5, 523: 5, 520: 5, 516: 5, 500: 5, 496: 5, 490: 5, 487: 5, 483: 5, 480: 5, 478: 5, 471: 5,
469: 5, 464: 5, 449: 5, 447: 5, 445: 5, 442: 5, 436: 5, 430: 5, 410: 5, 404: 5, 402: 5, 394: 5,
385: 5, 357: 5, 354: 5, 347: 5, 338: 5, 312: 5, 1875: 4, 1872: 4, 1656: 4, 1655: 4, 1626: 4, 1527:
4, 1499: 4, 1454: 4, 1408: 4, 1373: 4, 1365: 4, 1329: 4, 1326: 4, 1304: 4, 1248: 4, 1226: 4, 1218:
4, 1153: 4, 1152: 4, 1116: 4, 1097: 4, 1066: 4, 1037: 4, 1017: 4, 1005: 4, 979: 4, 964: 4, 963: 4,
960: 4, 956: 4, 954: 4, 922: 4, 919: 4, 915: 4, 887: 4, 871: 4, 868: 4, 858: 4, 854: 4, 826: 4,
818: 4, 815: 4, 813: 4, 805: 4, 799: 4, 792: 4, 787: 4, 781: 4, 775: 4, 763: 4, 743: 4, 742: 4,
728: 4, 722: 4, 719: 4, 715: 4, 712: 4, 711: 4, 705: 4, 693: 4, 691: 4, 686: 4, 685: 4, 683: 4,
679: 4, 672: 4, 666: 4, 648: 4, 647: 4, 637: 4, 629: 4, 624: 4, 620: 4, 619: 4, 614: 4, 607: 4,
595: 4, 591: 4, 586: 4, 585: 4, 584: 4, 571: 4, 570: 4, 565: 4, 559: 4, 558: 4, 551: 4, 547: 4,
538: 4, 534: 4, 530: 4, 529: 4, 527: 4, 522: 4, 514: 4, 510: 4, 508: 4, 499: 4, 498: 4, 485: 4,
479: 4, 474: 4, 463: 4, 454: 4, 452: 4, 451: 4, 448: 4, 438: 4, 435: 4, 428: 4, 415: 4, 407: 4,
387: 4, 382: 4, 381: 4, 370: 4, 337: 4, 329: 4, 322: 4, 3935: 3, 2662: 3, 2574: 3, 2558: 3, 2460: 3
, 2348: 3, 2285: 3, 2244: 3, 2198: 3, 2188: 3, 2153: 3, 2073: 3, 2038: 3, 2037: 3, 1995: 3, 1967: 3
```

, 1951: 3, 1950: 3, 1894: 3, 1851: 3, 1810: 3, 1719: 3, 1692: 3, 1677: 3, 1657: 3, 1645: 3, 1607: 3
, 1601: 3, 1538: 3, 1536: 3, 1535: 3, 1529: 3, 1528: 3, 1497: 3, 1495: 3, 1489: 3, 1481: 3, 1474: 3
, 1462: 3, 1457: 3, 1431: 3, 1410: 3, 1378: 3, 1377: 3, 1360: 3, 1337: 3, 1334: 3, 1322: 3, 1315: 3
, 1302: 3, 1301: 3, 1282: 3, 1267: 3, 1262: 3, 1244: 3, 1231: 3, 1216: 3, 1213: 3, 1206: 3, 1188: 3
, 1186: 3, 1176: 3, 1170: 3, 1155: 3, 1146: 3, 1125: 3, 1121: 3, 1106: 3, 1103: 3, 1102: 3, 1094: 3
, 1087: 3, 1084: 3, 1083: 3, 1063: 3, 1049: 3, 1044: 3, 1040: 3, 1036: 3, 1027: 3, 1019: 3, 1009: 3
, 1002: 3, 1001: 3, 995: 3, 991: 3, 988: 3, 987: 3, 984: 3, 983: 3, 982: 3, 981: 3, 980: 3, 965: 3,
957: 3, 947: 3, 942: 3, 941: 3, 931: 3, 918: 3, 911: 3, 904: 3, 895: 3, 883: 3, 882: 3, 867: 3,
865: 3, 861: 3, 860: 3, 859: 3, 852: 3, 849: 3, 848: 3, 842: 3, 839: 3, 831: 3, 828: 3, 816: 3,
814: 3, 808: 3, 806: 3, 793: 3, 788: 3, 784: 3, 776: 3, 772: 3, 767: 3, 762: 3, 760: 3, 751: 3,
750: 3, 747: 3, 741: 3, 739: 3, 734: 3, 731: 3, 721: 3, 717: 3, 713: 3, 709: 3, 706: 3, 695: 3,
690: 3, 687: 3, 684: 3, 680: 3, 664: 3, 661: 3, 652: 3, 650: 3, 640: 3, 623: 3, 622: 3, 618: 3,
608: 3, 604: 3, 603: 3, 600: 3, 592: 3, 578: 3, 577: 3, 574: 3, 568: 3, 566: 3, 561: 3, 557: 3,
553: 3, 552: 3, 548: 3, 546: 3, 541: 3, 537: 3, 532: 3, 525: 3, 517: 3, 506: 3, 493: 3, 492: 3,
491: 3, 462: 3, 459: 3, 439: 3, 424: 3, 421: 3, 395: 3, 356: 3, 6953: 2, 6743: 2, 6085: 2, 6021: 2,
5540: 2, 5352: 2, 5072: 2, 4734: 2, 4470: 2, 4326: 2, 4311: 2, 4270: 2, 4204: 2, 4152: 2, 3986: 2,
3907: 2, 3887: 2, 3823: 2, 3797: 2, 3791: 2, 3777: 2, 3727: 2, 3706: 2, 3689: 2, 3612: 2, 3596: 2,
3541: 2, 3539: 2, 3443: 2, 3430: 2, 3408: 2, 3386: 2, 3330: 2, 3297: 2, 3266: 2, 3249: 2, 3197: 2,
3188: 2, 3178: 2, 3159: 2, 3136: 2, 3071: 2, 3063: 2, 3044: 2, 3018: 2, 3000: 2, 2963: 2, 2943: 2,
2810: 2, 2799: 2, 2779: 2, 2748: 2, 2735: 2, 2728: 2, 2725: 2, 2717: 2, 2716: 2, 2666: 2, 2643: 2,
2628: 2, 2623: 2, 2621: 2, 2600: 2, 2597: 2, 2593: 2, 2578: 2, 2573: 2, 2534: 2, 2517: 2, 2502: 2,
2482: 2, 2471: 2, 2456: 2, 2449: 2, 2436: 2, 2356: 2, 2310: 2, 2300: 2, 2265: 2, 2254: 2, 2230: 2,
2214: 2, 2174: 2, 2172: 2, 2136: 2, 2135: 2, 2129: 2, 2114: 2, 2081: 2, 2072: 2, 2068: 2, 2041: 2,
2023: 2, 1991: 2, 1986: 2, 1956: 2, 1937: 2, 1932: 2, 1927: 2, 1922: 2, 1905: 2, 1868: 2, 1863: 2,
1858: 2, 1850: 2, 1838: 2, 1833: 2, 1829: 2, 1820: 2, 1811: 2, 1788: 2, 1787: 2, 1785: 2, 1779: 2,
1778: 2, 1775: 2, 1764: 2, 1744: 2, 1738: 2, 1737: 2, 1730: 2, 1704: 2, 1699: 2, 1690: 2, 1682: 2,
1675: 2, 1661: 2, 1651: 2, 1649: 2, 1643: 2, 1641: 2, 1638: 2, 1635: 2, 1632: 2, 1622: 2, 1621: 2,
1620: 2, 1619: 2, 1608: 2, 1595: 2, 1585: 2, 1582: 2, 1581: 2, 1571: 2, 1566: 2, 1562: 2, 1558: 2,
1557: 2, 1547: 2, 1545: 2, 1520: 2, 1518: 2, 1511: 2, 1510: 2, 1508: 2, 1501: 2, 1490: 2, 1483: 2,
1479: 2, 1465: 2, 1450: 2, 1445: 2, 1444: 2, 1440: 2, 1437: 2, 1429: 2, 1422: 2, 1412: 2, 1411: 2,
1407: 2, 1405: 2, 1404: 2, 1401: 2, 1399: 2, 1382: 2, 1379: 2, 1370: 2, 1368: 2, 1363: 2, 1358: 2,
1357: 2, 1355: 2, 1346: 2, 1344: 2, 1342: 2, 1341: 2, 1327: 2, 1325: 2, 1320: 2, 1319: 2, 1318: 2,
1317: 2, 1316: 2, 1311: 2, 1305: 2, 1299: 2, 1298: 2, 1296: 2, 1294: 2, 1290: 2, 1288: 2, 1285: 2,
1279: 2, 1278: 2, 1277: 2, 1264: 2, 1241: 2, 1238: 2, 1237: 2, 1235: 2, 1229: 2, 1212: 2, 1210: 2,
1207: 2, 1202: 2, 1200: 2, 1199: 2, 1197: 2, 1194: 2, 1192: 2, 1189: 2, 1181: 2, 1180: 2, 1178: 2,
1173: 2, 1169: 2, 1147: 2, 1144: 2, 1129: 2, 1120: 2, 1118: 2, 1115: 2, 1114: 2, 1111: 2, 1110: 2,
1107: 2, 1098: 2, 1096: 2, 1093: 2, 1092: 2, 1086: 2, 1081: 2, 1078: 2, 1075: 2, 1074: 2, 1072: 2,
1070: 2, 1065: 2, 1062: 2, 1054: 2, 1047: 2, 1043: 2, 1039: 2, 1033: 2, 1026: 2, 1024: 2, 1015: 2,
1006: 2, 1000: 2, 998: 2, 996: 2, 992: 2, 990: 2, 976: 2, 975: 2, 972: 2, 962: 2, 953: 2, 950: 2,
949: 2, 948: 2, 945: 2, 943: 2, 939: 2, 934: 2, 932: 2, 930: 2, 929: 2, 925: 2, 924: 2, 921: 2,
917: 2, 916: 2, 914: 2, 910: 2, 908: 2, 901: 2, 897: 2, 894: 2, 892: 2, 890: 2, 884: 2, 880: 2,
879: 2, 878: 2, 876: 2, 873: 2, 862: 2, 857: 2, 856: 2, 853: 2, 851: 2, 850: 2, 847: 2, 845: 2,
843: 2, 841: 2, 837: 2, 832: 2, 830: 2, 829: 2, 825: 2, 824: 2, 821: 2, 819: 2, 817: 2, 810: 2,
807: 2, 804: 2, 800: 2, 796: 2, 794: 2, 790: 2, 786: 2, 783: 2, 771: 2, 770: 2, 769: 2, 759: 2,
756: 2, 754: 2, 753: 2, 749: 2, 745: 2, 744: 2, 737: 2, 732: 2, 710: 2, 703: 2, 702: 2, 701: 2,
700: 2, 697: 2, 696: 2, 677: 2, 675: 2, 673: 2, 665: 2, 662: 2, 660: 2, 659: 2, 658: 2, 643: 2,
642: 2, 639: 2, 634: 2, 631: 2, 627: 2, 625: 2, 621: 2, 616: 2, 612: 2, 610: 2, 606: 2, 599: 2,
598: 2, 597: 2, 596: 2, 590: 2, 589: 2, 576: 2, 562: 2, 560: 2, 555: 2, 545: 2, 544: 2, 543: 2,
540: 2, 535: 2, 521: 2, 519: 2, 511: 2, 494: 2, 488: 2, 482: 2, 473: 2, 450: 2, 434: 2, 432: 2,
427: 2, 406: 2, 405: 2, 374: 2, 154116: 1, 119884: 1, 82275: 1, 68546: 1, 68543: 1, 68455: 1,
66154: 1, 65032: 1, 63704: 1, 58273: 1, 55181: 1, 50758: 1, 49221: 1, 47400: 1, 47221: 1, 44469: 1
, 44189: 1, 43582: 1, 43469: 1, 43065: 1, 42073: 1, 41547: 1, 40220: 1, 40070: 1, 38860: 1, 38770:
1, 37038: 1, 36873: 1, 36198: 1, 34385: 1, 34148: 1, 33903: 1, 33839: 1, 33261: 1, 33220: 1, 32122
: 1, 30222: 1, 29981: 1, 28341: 1, 26365: 1, 26357: 1, 26017: 1, 25930: 1, 25250: 1, 25154: 1, 250
89: 1, 24978: 1, 24536: 1, 24489: 1, 24413: 1, 24371: 1, 24283: 1, 23882: 1, 23180: 1, 22567: 1, 2
2282: 1, 22245: 1, 22235: 1, 22009: 1, 21522: 1, 21166: 1, 20748: 1, 20309: 1, 20261: 1, 20209: 1,
19945: 1, 19675: 1, 19594: 1, 19353: 1, 19226: 1, 19084: 1, 18883: 1, 18860: 1, 18846: 1, 18811: 1
, 18596: 1, 18546: 1, 18514: 1, 18419: 1, 18395: 1, 18110: 1, 18082: 1, 18056: 1, 17990: 1, 17943:
1, 17907: 1, 17788: 1, 17732: 1, 17696: 1, 17592: 1, 17556: 1, 17437: 1, 17090: 1, 17086: 1, 17083
: 1, 17054: 1, 16704: 1, 16627: 1, 16573: 1, 15972: 1, 15955: 1, 15952: 1, 15950: 1, 15894: 1, 157
89: 1, 15740: 1, 15689: 1, 15440: 1, 15305: 1, 15271: 1, 15240: 1, 15128: 1, 15044: 1, 14985: 1, 1
4973: 1, 14871: 1, 14825: 1, 14494: 1, 14483: 1, 14374: 1, 14330: 1, 14273: 1, 14230: 1, 13945: 1,
13919: 1, 13893: 1, 13799: 1, 13792: 1, 13712: 1, 13570: 1, 13522: 1, 13512: 1, 13310: 1, 13308: 1
, 13220: 1, 13214: 1, 13186: 1, 13183: 1, 13102: 1, 13022: 1, 12996: 1, 12917: 1, 12875: 1, 12845:
1, 12749: 1, 12723: 1, 12712: 1, 12702: 1, 12664: 1, 12594: 1, 12587: 1, 12575: 1, 12564: 1, 12504
: 1, 12443: 1, 12400: 1, 12373: 1, 12366: 1, 12328: 1, 12325: 1, 12296: 1, 12264: 1, 12247: 1, 121
55: 1, 12074: 1, 12035: 1, 12008: 1, 11997: 1, 11984: 1, 11922: 1, 11860: 1, 11797: 1, 11747: 1, 1
1716: 1, 11704: 1, 11638: 1, 11489: 1, 11482: 1, 11478: 1, 11409: 1, 11343: 1, 11284: 1, 11278: 1,
11251: 1, 11235: 1, 11212: 1, 11155: 1, 11041: 1, 11011: 1, 10964: 1, 10878: 1, 10796: 1, 10706: 1
, 10620: 1, 10534: 1, 10476: 1, 10409: 1, 10349: 1, 10346: 1, 10286: 1, 10242: 1, 10224: 1, 10162:
1, 10158: 1, 10106: 1, 10105: 1, 10074: 1, 10046: 1, 10007: 1, 9934: 1, 9927: 1, 9859: 1, 9796: 1,
9792: 1, 9773: 1, 9717: 1, 9689: 1, 9590: 1, 9564: 1, 9487: 1, 9469: 1, 9464: 1, 9459: 1, 9384: 1,
9370: 1, 9358: 1, 9356: 1, 9351: 1, 9312: 1, 9292: 1, 9277: 1, 9275: 1, 9270: 1, 9245: 1, 9240: 1,
9205: 1, 9182: 1, 9074: 1, 9011: 1, 9009: 1, 8985: 1, 8979: 1, 8965: 1, 8924: 1, 8863: 1, 8860: 1,
8831: 1, 8829: 1, 8770: 1, 8730: 1, 8697: 1, 8592: 1, 8578: 1, 8571: 1, 8538: 1, 8521: 1, 8516: 1,
8514: 1, 8507: 1, 8469: 1, 8459: 1, 8435: 1, 8414: 1, 8402: 1, 8399: 1, 8367: 1, 8353: 1, 8255: 1,
8251: 1, 8227: 1, 8219: 1, 8217: 1, 8186: 1, 8164: 1, 8161: 1, 8080: 1, 8055: 1, 8054: 1, 8008: 1,

8005: 1, 7985: 1, 7970: 1, 7963: 1, 7931: 1, 7911: 1, 7849: 1, 7835: 1, 7825: 1, 7811: 1, 7805: 1,
7799: 1, 7789: 1, 7779: 1, 7707: 1, 7695: 1, 7689: 1, 7667: 1, 7660: 1, 7646: 1, 7577: 1, 7563: 1,
7556: 1, 7548: 1, 7545: 1, 7479: 1, 7457: 1, 7407: 1, 7383: 1, 7378: 1, 7370: 1, 7361: 1, 7358: 1,
7303: 1, 7294: 1, 7277: 1, 7235: 1, 7223: 1, 7203: 1, 7193: 1, 7191: 1, 7182: 1, 7181: 1, 7168: 1,
7161: 1, 7154: 1, 7124: 1, 7085: 1, 7081: 1, 7075: 1, 7066: 1, 7058: 1, 7054: 1, 7024: 1, 6975: 1,
6929: 1, 6925: 1, 6924: 1, 6919: 1, 6908: 1, 6851: 1, 6835: 1, 6783: 1, 6775: 1, 6767: 1, 6757: 1,
6718: 1, 6714: 1, 6701: 1, 6695: 1, 6685: 1, 6676: 1, 6647: 1, 6635: 1, 6629: 1, 6621: 1, 6615: 1,
6607: 1, 6572: 1, 6538: 1, 6520: 1, 6510: 1, 6502: 1, 6493: 1, 6486: 1, 6483: 1, 6478: 1, 6470: 1,
6446: 1, 6374: 1, 6373: 1, 6367: 1, 6359: 1, 6358: 1, 6341: 1, 6340: 1, 6321: 1, 6320: 1, 6319: 1,
6312: 1, 6308: 1, 6283: 1, 6278: 1, 6261: 1, 6239: 1, 6212: 1, 6178: 1, 6175: 1, 6172: 1, 6165: 1,
6150: 1, 6148: 1, 6141: 1, 6140: 1, 6128: 1, 6125: 1, 6084: 1, 6050: 1, 6041: 1, 6032: 1, 6018: 1,
6009: 1, 6000: 1, 5996: 1, 5986: 1, 5984: 1, 5974: 1, 5968: 1, 5946: 1, 5934: 1, 5927: 1, 5868: 1,
5865: 1, 5834: 1, 5817: 1, 5810: 1, 5773: 1, 5770: 1, 5767: 1, 5752: 1, 5717: 1, 5715: 1, 5713: 1,
5710: 1, 5683: 1, 5682: 1, 5670: 1, 5667: 1, 5656: 1, 5647: 1, 5607: 1, 5593: 1, 5590: 1, 5559: 1,
5545: 1, 5538: 1, 5537: 1, 5530: 1, 5516: 1, 5507: 1, 5502: 1, 5498: 1, 5476: 1, 5464: 1, 5447: 1,
5427: 1, 5426: 1, 5418: 1, 5391: 1, 5377: 1, 5365: 1, 5361: 1, 5337: 1, 5333: 1, 5318: 1, 5304: 1,
5293: 1, 5275: 1, 5244: 1, 5237: 1, 5230: 1, 5223: 1, 5222: 1, 5219: 1, 5197: 1, 5183: 1, 5181: 1,
5179: 1, 5176: 1, 5171: 1, 5165: 1, 5158: 1, 5157: 1, 5155: 1, 5152: 1, 5112: 1, 5082: 1, 5065: 1,
5062: 1, 5054: 1, 5053: 1, 5051: 1, 5046: 1, 5038: 1, 5010: 1, 5008: 1, 4998: 1, 4995: 1, 4994: 1,
4990: 1, 4969: 1, 4968: 1, 4953: 1, 4942: 1, 4934: 1, 4931: 1, 4929: 1, 4922: 1, 4899: 1, 4896: 1,
4886: 1, 4882: 1, 4879: 1, 4876: 1, 4874: 1, 4856: 1, 4837: 1, 4829: 1, 4828: 1, 4824: 1, 4811: 1,
4806: 1, 4803: 1, 4796: 1, 4788: 1, 4775: 1, 4774: 1, 4752: 1, 4727: 1, 4709: 1, 4708: 1, 4695: 1,
4694: 1, 4658: 1, 4654: 1, 4652: 1, 4651: 1, 4636: 1, 4601: 1, 4590: 1, 4544: 1, 4543: 1, 4540: 1,
4537: 1, 4533: 1, 4531: 1, 4503: 1, 4494: 1, 4475: 1, 4460: 1, 4459: 1, 4456: 1, 4453: 1, 4452: 1,
4445: 1, 4444: 1, 4441: 1, 4430: 1, 4424: 1, 4423: 1, 4422: 1, 4417: 1, 4410: 1, 4406: 1, 4400: 1,
4396: 1, 4388: 1, 4384: 1, 4365: 1, 4360: 1, 4359: 1, 4357: 1, 4340: 1, 4334: 1, 4330: 1, 4323: 1,
4319: 1, 4308: 1, 4298: 1, 4282: 1, 4276: 1, 4272: 1, 4264: 1, 4262: 1, 4258: 1, 4232: 1, 4231: 1,
4226: 1, 4224: 1, 4218: 1, 4214: 1, 4210: 1, 4197: 1, 4195: 1, 4192: 1, 4172: 1, 4161: 1, 4150: 1,
4145: 1, 4144: 1, 4135: 1, 4120: 1, 4116: 1, 4111: 1, 4104: 1, 4101: 1, 4095: 1, 4084: 1, 4080: 1,
4073: 1, 4071: 1, 4070: 1, 4066: 1, 4060: 1, 4059: 1, 4041: 1, 4035: 1, 4033: 1, 4026: 1, 4024: 1,
4018: 1, 4017: 1, 4001: 1, 3995: 1, 3983: 1, 3982: 1, 3979: 1, 3964: 1, 3953: 1, 3948: 1, 3941: 1,
3940: 1, 3930: 1, 3928: 1, 3914: 1, 3913: 1, 3896: 1, 3892: 1, 3885: 1, 3880: 1, 3871: 1, 3856: 1,
3847: 1, 3825: 1, 3820: 1, 3804: 1, 3801: 1, 3799: 1, 3793: 1, 3792: 1, 3790: 1, 3780: 1, 3751: 1,
3748: 1, 3745: 1, 3743: 1, 3739: 1, 3732: 1, 3731: 1, 3728: 1, 3725: 1, 3724: 1, 3721: 1, 3714: 1,
3713: 1, 3695: 1, 3690: 1, 3684: 1, 3682: 1, 3681: 1, 3680: 1, 3672: 1, 3669: 1, 3664: 1, 3656: 1,
3641: 1, 3639: 1, 3634: 1, 3632: 1, 3628: 1, 3613: 1, 3610: 1, 3608: 1, 3607: 1, 3604: 1, 3583: 1,
3582: 1, 3579: 1, 3578: 1, 3562: 1, 3558: 1, 3557: 1, 3549: 1, 3547: 1, 3545: 1, 3543: 1, 3533: 1,
3530: 1, 3524: 1, 3519: 1, 3509: 1, 3506: 1, 3501: 1, 3499: 1, 3498: 1, 3490: 1, 3487: 1, 3481: 1,
3480: 1, 3478: 1, 3471: 1, 3469: 1, 3465: 1, 3459: 1, 3455: 1, 3452: 1, 3450: 1, 3445: 1, 3437: 1,
3433: 1, 3432: 1, 3424: 1, 3415: 1, 3406: 1, 3395: 1, 3391: 1, 3390: 1, 3385: 1, 3372: 1, 3370: 1,
3369: 1, 3367: 1, 3360: 1, 3358: 1, 3357: 1, 3356: 1, 3353: 1, 3347: 1, 3346: 1, 3344: 1, 3343: 1,
3341: 1, 3340: 1, 3329: 1, 3328: 1, 3322: 1, 3311: 1, 3309: 1, 3296: 1, 3293: 1, 3292: 1, 3291: 1,
3289: 1, 3278: 1, 3274: 1, 3269: 1, 3268: 1, 3261: 1, 3259: 1, 3257: 1, 3254: 1, 3252: 1, 3251: 1,
3248: 1, 3238: 1, 3229: 1, 3226: 1, 3206: 1, 3194: 1, 3192: 1, 3187: 1, 3181: 1, 3176: 1, 3173: 1,
3170: 1, 3167: 1, 3157: 1, 3153: 1, 3143: 1, 3139: 1, 3135: 1, 3130: 1, 3125: 1, 3124: 1, 3122: 1,
3120: 1, 3117: 1, 3115: 1, 3106: 1, 3100: 1, 3098: 1, 3090: 1, 3085: 1, 3082: 1, 3074: 1, 3073: 1,
3064: 1, 3051: 1, 3036: 1, 3032: 1, 3026: 1, 3022: 1, 3019: 1, 3016: 1, 3015: 1, 3013: 1, 3012: 1,
3011: 1, 3008: 1, 3007: 1, 3001: 1, 2995: 1, 2990: 1, 2988: 1, 2984: 1, 2982: 1, 2974: 1, 2948: 1,
2946: 1, 2945: 1, 2938: 1, 2933: 1, 2913: 1, 2907: 1, 2905: 1, 2902: 1, 2899: 1, 2893: 1, 2884: 1,
2882: 1, 2880: 1, 2870: 1, 2863: 1, 2862: 1, 2858: 1, 2856: 1, 2854: 1, 2853: 1, 2848: 1, 2843: 1,
2841: 1, 2833: 1, 2831: 1, 2830: 1, 2829: 1, 2828: 1, 2827: 1, 2815: 1, 2801: 1, 2800: 1, 2786: 1,
2782: 1, 2774: 1, 2768: 1, 2761: 1, 2760: 1, 2759: 1, 2755: 1, 2753: 1, 2751: 1, 2750: 1, 2743: 1,
2739: 1, 2736: 1, 2734: 1, 2733: 1, 2732: 1, 2723: 1, 2719: 1, 2712: 1, 2709: 1, 2706: 1, 2703: 1,
2688: 1, 2684: 1, 2676: 1, 2664: 1, 2660: 1, 2649: 1, 2648: 1, 2644: 1, 2642: 1, 2635: 1, 2634: 1,
2631: 1, 2629: 1, 2620: 1, 2619: 1, 2617: 1, 2615: 1, 2614: 1, 2613: 1, 2610: 1, 2601: 1, 2599: 1,
2594: 1, 2590: 1, 2588: 1, 2585: 1, 2580: 1, 2565: 1, 2562: 1, 2561: 1, 2560: 1, 2559: 1, 2557: 1,
2555: 1, 2545: 1, 2544: 1, 2535: 1, 2529: 1, 2527: 1, 2523: 1, 2514: 1, 2513: 1, 2510: 1, 2508: 1,
2505: 1, 2497: 1, 2496: 1, 2491: 1, 2486: 1, 2485: 1, 2481: 1, 2473: 1, 2472: 1, 2469: 1, 2464: 1,
2462: 1, 2457: 1, 2455: 1, 2453: 1, 2446: 1, 2444: 1, 2440: 1, 2435: 1, 2434: 1, 2432: 1, 2424: 1,
2423: 1, 2418: 1, 2415: 1, 2411: 1, 2404: 1, 2403: 1, 2402: 1, 2400: 1, 2396: 1, 2390: 1, 2381: 1,
2377: 1, 2376: 1, 2375: 1, 2365: 1, 2361: 1, 2360: 1, 2358: 1, 2357: 1, 2354: 1, 2342: 1, 2340: 1,
2336: 1, 2333: 1, 2332: 1, 2331: 1, 2326: 1, 2319: 1, 2317: 1, 2315: 1, 2311: 1, 2306: 1, 2304: 1,
2301: 1, 2294: 1, 2291: 1, 2287: 1, 2286: 1, 2284: 1, 2280: 1, 2279: 1, 2276: 1, 2273: 1, 2272: 1,
2268: 1, 2267: 1, 2266: 1, 2262: 1, 2258: 1, 2257: 1, 2253: 1, 2249: 1, 2248: 1, 2247: 1, 2246: 1,
2241: 1, 2240: 1, 2239: 1, 2232: 1, 2225: 1, 2222: 1, 2219: 1, 2213: 1, 2211: 1, 2209: 1, 2201: 1,
2199: 1, 2197: 1, 2196: 1, 2193: 1, 2186: 1, 2185: 1, 2184: 1, 2183: 1, 2182: 1, 2179: 1, 2176: 1,
2175: 1, 2173: 1, 2170: 1, 2169: 1, 2166: 1, 2165: 1, 2164: 1, 2160: 1, 2158: 1, 2156: 1, 2147: 1,
2142: 1, 2141: 1, 2134: 1, 2133: 1, 2131: 1, 2128: 1, 2126: 1, 2122: 1, 2119: 1, 2117: 1, 2113: 1,
2108: 1, 2103: 1, 2101: 1, 2100: 1, 2099: 1, 2097: 1, 2093: 1, 2091: 1, 2089: 1, 2083: 1, 2082: 1,
2079: 1, 2077: 1, 2076: 1, 2075: 1, 2074: 1, 2071: 1, 2070: 1, 2069: 1, 2064: 1, 2056: 1, 2054: 1,
2053: 1, 2052: 1, 2049: 1, 2047: 1, 2044: 1, 2039: 1, 2036: 1, 2033: 1, 2028: 1, 2025: 1, 2024: 1,
2022: 1, 2021: 1, 2020: 1, 2019: 1, 2017: 1, 2010: 1, 2009: 1, 2008: 1, 2006: 1, 2003: 1, 2000: 1,
1999: 1, 1998: 1, 1997: 1, 1993: 1, 1992: 1, 1989: 1, 1983: 1, 1978: 1, 1975: 1, 1972: 1, 1968: 1,
1965: 1, 1964: 1, 1963: 1, 1961: 1, 1960: 1, 1957: 1, 1955: 1, 1954: 1, 1953: 1, 1949: 1, 1943: 1,
1939: 1, 1938: 1, 1931: 1, 1929: 1, 1924: 1, 1917: 1, 1910: 1, 1909: 1, 1907: 1, 1906: 1, 1904: 1,
1903: 1, 1902: 1, 1901: 1, 1900: 1, 1898: 1, 1896: 1, 1885: 1, 1878: 1, 1874: 1, 1873: 1, 1866: 1,
1862: 1, 1861: 1, 1857: 1, 1856: 1, 1848: 1, 1847: 1, 1844: 1, 1843: 1, 1832: 1, 1831: 1, 1828: 1,

1827: 1, 1825: 1, 1824: 1, 1822: 1, 1819: 1, 1818: 1, 1816: 1, 1814: 1, 1809: 1, 1808: 1, 1805: 1,
1799: 1, 1798: 1, 1795: 1, 1791: 1, 1783: 1, 1782: 1, 1781: 1, 1780: 1, 1776: 1, 1774: 1, 1773: 1,
1769: 1, 1768: 1, 1763: 1, 1761: 1, 1758: 1, 1756: 1, 1752: 1, 1751: 1, 1749: 1, 1748: 1, 1747: 1,
1746: 1, 1745: 1, 1743: 1, 1741: 1, 1734: 1, 1732: 1, 1731: 1, 1728: 1, 1726: 1, 1724: 1, 1722: 1,
1717: 1, 1716: 1, 1715: 1, 1713: 1, 1711: 1, 1710: 1, 1709: 1, 1708: 1, 1702: 1, 1698: 1, 1694: 1,
1693: 1, 1691: 1, 1688: 1, 1684: 1, 1678: 1, 1676: 1, 1674: 1, 1671: 1, 1670: 1, 1669: 1, 1668: 1,
1653: 1, 1647: 1, 1646: 1, 1642: 1, 1640: 1, 1639: 1, 1637: 1, 1618: 1, 1617: 1, 1616: 1, 1614: 1,
1613: 1, 1605: 1, 1604: 1, 1600: 1, 1599: 1, 1589: 1, 1587: 1, 1586: 1, 1584: 1, 1579: 1, 1576: 1,
1575: 1, 1574: 1, 1569: 1, 1567: 1, 1564: 1, 1561: 1, 1560: 1, 1559: 1, 1556: 1, 1552: 1, 1550: 1,
1549: 1, 1548: 1, 1546: 1, 1540: 1, 1533: 1, 1531: 1, 1530: 1, 1515: 1, 1509: 1, 1505: 1, 1503: 1,
1502: 1, 1500: 1, 1496: 1, 1492: 1, 1487: 1, 1486: 1, 1480: 1, 1476: 1, 1473: 1, 1471: 1, 1467: 1,
1464: 1, 1463: 1, 1460: 1, 1458: 1, 1449: 1, 1448: 1, 1447: 1, 1442: 1, 1439: 1, 1430: 1, 1428: 1,
1427: 1, 1426: 1, 1425: 1, 1419: 1, 1418: 1, 1416: 1, 1414: 1, 1409: 1, 1402: 1, 1398: 1, 1397: 1,
1394: 1, 1392: 1, 1391: 1, 1390: 1, 1386: 1, 1385: 1, 1384: 1, 1381: 1, 1376: 1, 1375: 1, 1371: 1,
1369: 1, 1366: 1, 1364: 1, 1362: 1, 1359: 1, 1353: 1, 1352: 1, 1347: 1, 1343: 1, 1340: 1, 1339: 1,
1338: 1, 1336: 1, 1335: 1, 1333: 1, 1331: 1, 1328: 1, 1324: 1, 1323: 1, 1321: 1, 1314: 1, 1313: 1,
1300: 1, 1293: 1, 1292: 1, 1291: 1, 1287: 1, 1286: 1, 1284: 1, 1283: 1, 1276: 1, 1275: 1, 1273: 1,
1272: 1, 1266: 1, 1263: 1, 1261: 1, 1260: 1, 1258: 1, 1256: 1, 1255: 1, 1254: 1, 1251: 1, 1250: 1,
1246: 1, 1245: 1, 1243: 1, 1242: 1, 1240: 1, 1239: 1, 1234: 1, 1232: 1, 1227: 1, 1225: 1, 1224: 1,
1223: 1, 1215: 1, 1211: 1, 1209: 1, 1208: 1, 1205: 1, 1203: 1, 1201: 1, 1198: 1, 1196: 1, 1195: 1,
1193: 1, 1191: 1, 1187: 1, 1184: 1, 1182: 1, 1175: 1, 1172: 1, 1171: 1, 1168: 1, 1167: 1, 1165: 1,
1163: 1, 1162: 1, 1161: 1, 1159: 1, 1157: 1, 1154: 1, 1151: 1, 1145: 1, 1140: 1, 1137: 1, 1136: 1,
1135: 1, 1134: 1, 1133: 1, 1132: 1, 1131: 1, 1127: 1, 1126: 1, 1124: 1, 1119: 1, 1117: 1, 1113: 1,
1112: 1, 1108: 1, 1105: 1, 1100: 1, 1099: 1, 1091: 1, 1088: 1, 1085: 1, 1082: 1, 1080: 1, 1079: 1,
1077: 1, 1076: 1, 1069: 1, 1068: 1, 1067: 1, 1064: 1, 1061: 1, 1060: 1, 1056: 1, 1055: 1, 1052: 1,
1051: 1, 1050: 1, 1046: 1, 1042: 1, 1041: 1, 1032: 1, 1030: 1, 1028: 1, 1025: 1, 1023: 1, 1022: 1,
1018: 1, 1014: 1, 1012: 1, 1007: 1, 1003: 1, 997: 1, 994: 1, 993: 1, 985: 1, 977: 1, 974: 1, 971: 1
, 968: 1, 967: 1, 966: 1, 952: 1, 946: 1, 944: 1, 940: 1, 936: 1, 935: 1, 933: 1, 928: 1, 927: 1,
926: 1, 920: 1, 913: 1, 912: 1, 909: 1, 907: 1, 903: 1, 898: 1, 893: 1, 889: 1, 885: 1, 877: 1,
875: 1, 872: 1, 869: 1, 864: 1, 846: 1, 840: 1, 835: 1, 833: 1, 827: 1, 823: 1, 822: 1, 820: 1,
812: 1, 811: 1, 802: 1, 797: 1, 785: 1, 782: 1, 780: 1, 779: 1, 768: 1, 765: 1, 758: 1, 755: 1,
752: 1, 729: 1, 727: 1, 725: 1, 723: 1, 716: 1, 714: 1, 708: 1, 707: 1, 699: 1, 698: 1, 692: 1,
671: 1, 667: 1, 657: 1, 654: 1, 611: 1, 609: 1, 601: 1, 593: 1, 587: 1, 579: 1, 550: 1, 539: 1,
513: 1, 467: 1, 465: 1})

Building the model with only text column:

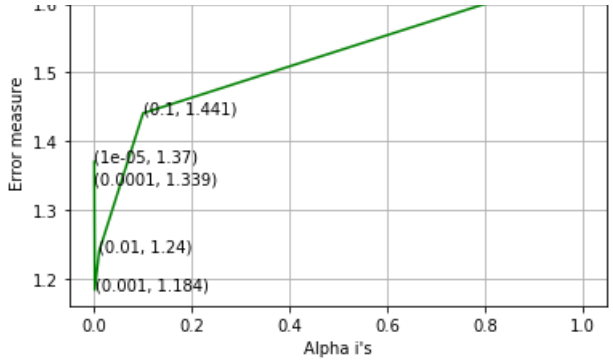
In [216]:

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

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

```
For values of alpha =  1e-05 The log loss is: 1.3704140564726335
For values of alpha =  0.0001 The log loss is: 1.3393383682590894
For values of alpha =  0.001 The log loss is: 1.1841352941447698
For values of alpha =  0.01 The log loss is: 1.2404874385740101
For values of alpha =  0.1 The log loss is: 1.440903934622484
For values of alpha =  1 The log loss is: 1.6450070898698594
```

In [217]:

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



In [218]:

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

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

```
For values of best alpha =  0.001 The train log loss is: 0.7652139935683602
For values of best alpha =  0.001 The cross validation log loss is: 1.1841352941447698
For values of best alpha =  0.001 The test log loss is: 1.1923836881749634
```

Lets check the overlap of text data

In [219]:

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

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

In [220]:

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

```
96.925 % of word of test data appeared in train data
98.106 % of word of Cross Validation appeared in train data
```

So, all 3 columns are going to be important.

Data prepration for Machine Learning models

Lets create few functions which we will be using later

In [221]:

```
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 [222]:

```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)

    A =(((C.T)/(C.sum(axis=1))).T)

    B =(C/C.sum(axis=0))
    labels = [1,2,3,4,5,6,7,8,9]
    # representing A in heatmap format
    print("-"*20, "Confusion matrix", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

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

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


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

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

In [223]:

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

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

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

    word_present = 0
    for i,v in enumerate(indices):
        if (v < fea1_len):
            word = gene_vec.get_feature_names()[v]
            yes_no = True if word == gene else False
```

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

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

Combining all 3 features together

In [224]:

```
# merging gene, variance and text features

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

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

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

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

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


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

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

In [225]:

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

```
One hot encoding features :
```

```
(number of data points * number of features) in train data =  (2124, 55456)
(number of data points * number of features) in test data =  (665, 55456)
(number of data points * number of features) in cross validation data = (532, 55456)
```

In [226]:

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

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

Building Machine Learning model

Lets start the first model which is most suitable when we have lot of text column data. So, we will start with Naive Bayes.

Naive Bayes

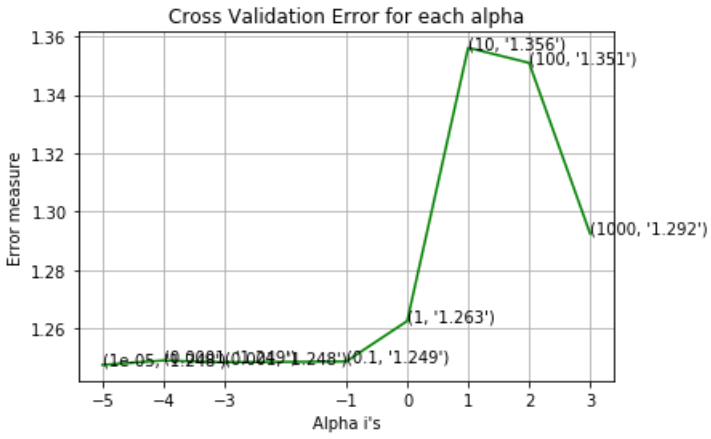
In [227]:

```
# http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
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))
```

```
for alpha = 1e-05
Log Loss : 1.2475208132340923
for alpha = 0.0001
Log Loss : 1.2490210551822714
for alpha = 0.001
Log Loss : 1.2483920506383104
for alpha = 0.1
Log Loss : 1.2486899718707631
for alpha = 1
Log Loss : 1.2625367647832204
for alpha = 10
Log Loss : 1.3560771144259305
for alpha = 100
Log Loss : 1.3508954502469588
for alpha = 1000
Log Loss : 1.292473562311196
```

In [228]:

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



In [229]:

```
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_lo
ss(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, p
redict_y, labels=clf.classes_, eps=1e-15))
```

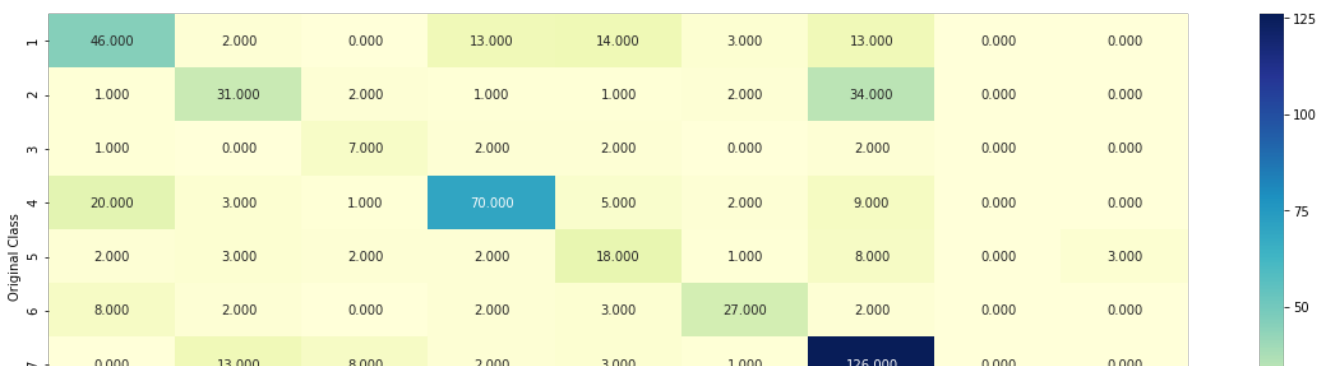
```
For values of best alpha =  1e-05 The train log loss is: 0.8681774554463136
For values of best alpha =  1e-05 The cross validation log loss is: 1.2475208132340923
For values of best alpha =  1e-05 The test log loss is: 1.2990146488643595
```

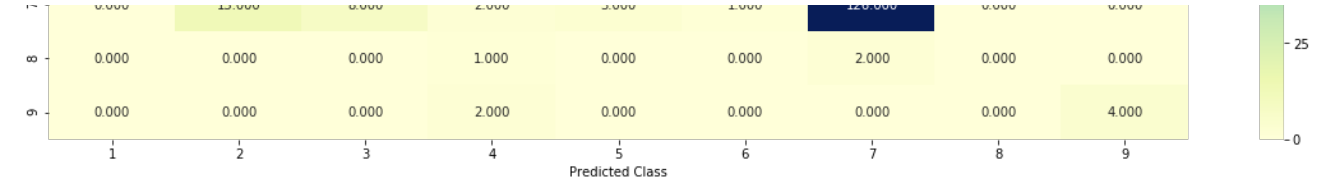
Testing our Naive Bayes model with best found value of alpha on testing data

In [230]:

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

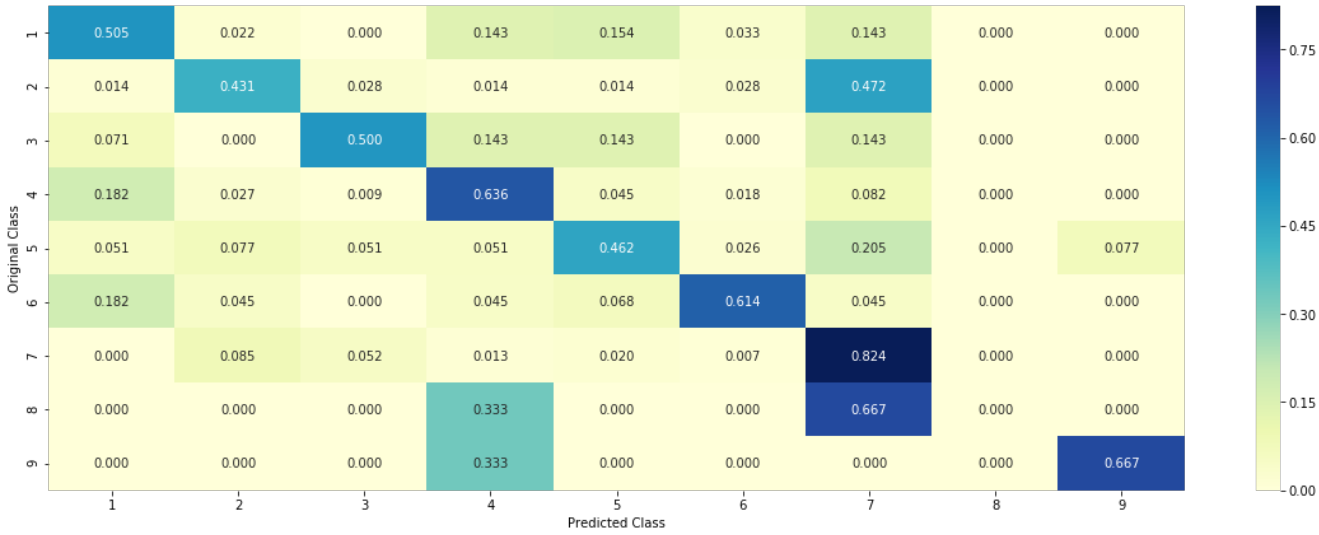
```
Log Loss : 1.2475208132340923
Number of missclassified point : 0.3815789473684211
-------------------- Confusion matrix --------------------
```






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


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



Interpretability of our model

In [231]:

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

test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 2
```

```
Predicted Class Probabilities: [[0.1125 0.4024 0.0127 0.121  0.0417 0.0371 0.2628 0.0054 0.0045]]
Actual Class : 7
--------------------------------------------------
14 Text feature [identified] present in test data point [True]
20 Text feature [molecular] present in test data point [True]
21 Text feature [including] present in test data point [True]
22 Text feature [harbored] present in test data point [True]
23 Text feature [novel] present in test data point [True]
24 Text feature [another] present in test data point [True]
25 Text feature [clinical] present in test data point [True]
26 Text feature [identification] present in test data point [True]
27 Text feature [case] present in test data point [True]
28 Text feature [discovered] present in test data point [True]
29 Text feature [confirmed] present in test data point [True]
30 Text feature [rearrangements] present in test data point [True]
31 Text feature [harbor] present in test data point [True]
32 Text feature [recently] present in test data point [True]
33 Text feature [mutations] present in test data point [True]
34 Text feature [may] present in test data point [True]
35 Text feature [new] present in test data point [True]
36 Text feature [well] present in test data point [True]
37 Text feature [using] present in test data point [True]
40 Text feature [revealed] present in test data point [True]
41 Text feature [qiagen] present in test data point [True]
42 Text feature [different] present in test data point [True]
43 Text feature [present] present in test data point [True]
44 Text feature [partners] present in test data point [True]
45 Text feature [patient] present in test data point [True]
46 Text feature [potential] present in test data point [True]
47 Text feature [go] present in test data point [True]
48 Text feature [12] present in test data point [True]
49 Text feature [subsequently] present in test data point [True]
50 Text feature [identify] present in test data point [True]
51 Text feature [distinct] present in test data point [True]
52 Text feature [sequencing] present in test data point [True]
53 Text feature [studies] present in test data point [True]
54 Text feature [kinase] present in test data point [True]
55 Text feature [characterized] present in test data point [True]
56 Text feature [gene] present in test data point [True]
57 Text feature [need] present in test data point [True]
58 Text feature [subsequent] present in test data point [True]
59 Text feature [activating] present in test data point [True]
60 Text feature [found] present in test data point [True]
61 Text feature [treatment] present in test data point [True]
62 Text feature [highly] present in test data point [True]
63 Text feature [observed] present in test data point [True]
64 Text feature [also] present in test data point [True]
65 Text feature [15] present in test data point [True]
67 Text feature [33] present in test data point [True]
68 Text feature [previously] present in test data point [True]
70 Text feature [however] present in test data point [True]
72 Text feature [currently] present in test data point [True]
73 Text feature [fusion] present in test data point [True]
74 Text feature [informed] present in test data point [True]
75 Text feature [40] present in test data point [True]
76 Text feature [mutational] present in test data point [True]
77 Text feature [tumor] present in test data point [True]
78 Text feature [harboring] present in test data point [True]
79 Text feature [time] present in test data point [True]
80 Text feature [pcr] present in test data point [True]
81 Text feature [therapeutic] present in test data point [True]
82 Text feature [higher] present in test data point [True]
83 Text feature [involving] present in test data point [True]
84 Text feature [respectively] present in test data point [True]
87 Text feature [significance] present in test data point [True]
88 Text feature [study] present in test data point [True]
89 Text feature [detection] present in test data point [True]
90 Text feature [number] present in test data point [True]
92 Text feature [one] present in test data point [True]
93 Text feature [included] present in test data point [True]
95 Text feature [findings] present in test data point [True]
96 Text feature [types] present in test data point [True]
97 Text feature [similar] present in test data point [True]
98 Text feature [small] present in test data point [True]
99 Text feature [reported] present in test data point [True]
Out of the top  100  features  72 are present in query point
```

In [232]:

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

,test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.403  0.0981 0.0131 0.1265 0.0435 0.0386 0.2669 0.0056 0.0046]]
Actual Class : 1
--------------------------------------------------
12 Text feature [function] present in test data point [True]
13 Text feature [affect] present in test data point [True]
14 Text feature [protein] present in test data point [True]
15 Text feature [binding] present in test data point [True]
16 Text feature [type] present in test data point [True]
17 Text feature [wild] present in test data point [True]
18 Text feature [two] present in test data point [True]
19 Text feature [dna] present in test data point [True]
20 Text feature [one] present in test data point [True]
21 Text feature [amino] present in test data point [True]
22 Text feature [region] present in test data point [True]
23 Text feature [remaining] present in test data point [True]
24 Text feature [acids] present in test data point [True]
25 Text feature [containing] present in test data point [True]
26 Text feature [effect] present in test data point [True]
29 Text feature [sequence] present in test data point [True]
30 Text feature [four] present in test data point [True]
31 Text feature [loss] present in test data point [True]
32 Text feature [least] present in test data point [True]
33 Text feature [large] present in test data point [True]
34 Text feature [indicating] present in test data point [True]
36 Text feature [surface] present in test data point [True]
37 Text feature [conserved] present in test data point [True]
38 Text feature [therefore] present in test data point [True]
39 Text feature [possible] present in test data point [True]
40 Text feature [indicate] present in test data point [True]
41 Text feature [reduced] present in test data point [True]
42 Text feature [likely] present in test data point [True]
43 Text feature [six] present in test data point [True]
44 Text feature [possibility] present in test data point [True]
45 Text feature [three] present in test data point [True]
46 Text feature [critical] present in test data point [True]
47 Text feature [structure] present in test data point [True]
48 Text feature [nonsense] present in test data point [True]
49 Text feature [corresponding] present in test data point [True]
50 Text feature [results] present in test data point [True]
51 Text feature [define] present in test data point [True]
52 Text feature [contains] present in test data point [True]
53 Text feature [used] present in test data point [True]
54 Text feature [sequences] present in test data point [True]
55 Text feature [indicated] present in test data point [True]
56 Text feature [terminal] present in test data point [True]
57 Text feature [present] present in test data point [True]
58 Text feature [five] present in test data point [True]
59 Text feature [domains] present in test data point [True]
60 Text feature [reveal] present in test data point [True]
61 Text feature [control] present in test data point [True]
62 Text feature [analysis] present in test data point [True]
63 Text feature [located] present in test data point [True]
64 Text feature [result] present in test data point [True]
65 Text feature [table] present in test data point [True]
66 Text feature [functions] present in test data point [True]
67 Text feature [form] present in test data point [True]
68 Text feature [data] present in test data point [True]
70 Text feature [genetic] present in test data point [True]
71 Text feature [length] present in test data point [True]
72 Text feature [determined] present in test data point [True]
```

```
73 Text feature [transcriptional] present in test data point [True]
74 Text feature [expected] present in test data point [True]
75 Text feature [change] present in test data point [True]
76 Text feature [structural] present in test data point [True]
77 Text feature [previous] present in test data point [True]
79 Text feature [also] present in test data point [True]
80 Text feature [page] present in test data point [True]
81 Text feature [whereas] present in test data point [True]
82 Text feature [identified] present in test data point [True]
83 Text feature [37] present in test data point [True]
84 Text feature [ability] present in test data point [True]
85 Text feature [defined] present in test data point [True]
86 Text feature [full] present in test data point [True]
87 Text feature [specific] present in test data point [True]
89 Text feature [central] present in test data point [True]
90 Text feature [multiple] present in test data point [True]
91 Text feature [using] present in test data point [True]
92 Text feature [proteins] present in test data point [True]
93 Text feature [either] present in test data point [True]
94 Text feature [involved] present in test data point [True]
95 Text feature [within] present in test data point [True]
96 Text feature [eight] present in test data point [True]
97 Text feature [shown] present in test data point [True]
98 Text feature [essential] present in test data point [True]
Out of the top  100  features  81 are present in query point
```

Naive Bayes is performing good but lets look at other models

K Nearest Neighbour Classification

In [233]:

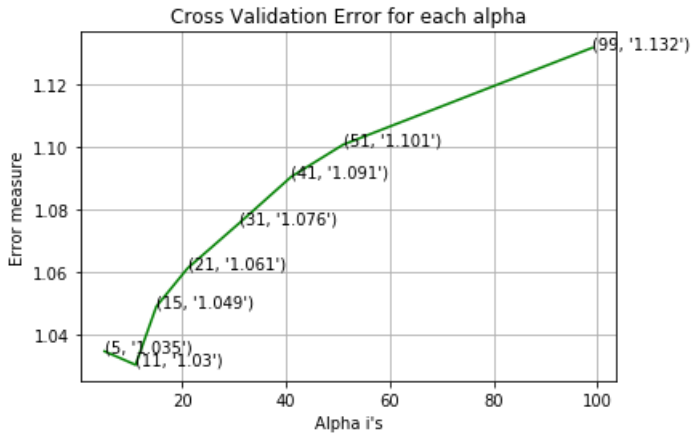
```
alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_responseCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_responseCoding, 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))
```

```
for alpha = 5
Log Loss : 1.0348121963956298
for alpha = 11
Log Loss : 1.0304604427129438
for alpha = 15
Log Loss : 1.0491074249435823
for alpha = 21
Log Loss : 1.0612952387631567
for alpha = 31
Log Loss : 1.0757757970235726
for alpha = 41
Log Loss : 1.0905807426127927
for alpha = 51
Log Loss : 1.1008247870508938
for alpha = 99
Log Loss : 1.131793689021609
```

In [234]:

```
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()
```



In [235]:

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

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

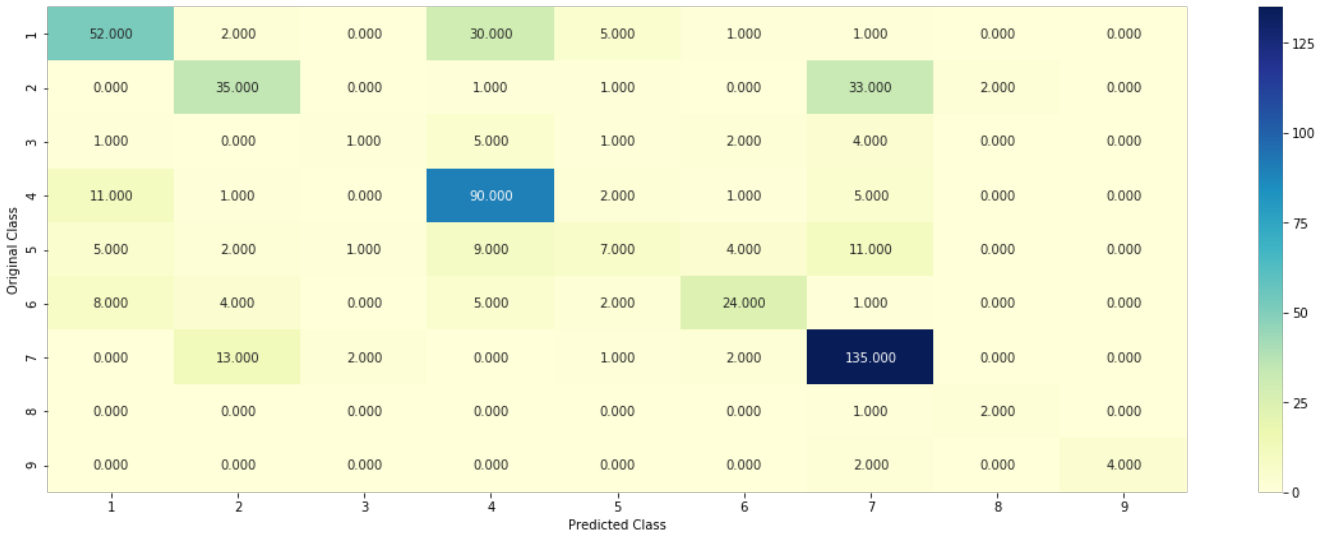
```
For values of best alpha =  11 The train log loss is: 0.6004644889100195
For values of best alpha =  11 The cross validation log loss is: 1.0304604427129438
For values of best alpha =  11 The test log loss is: 1.0575180443535317
```

Let's test it on testing dataset with our best alpha value

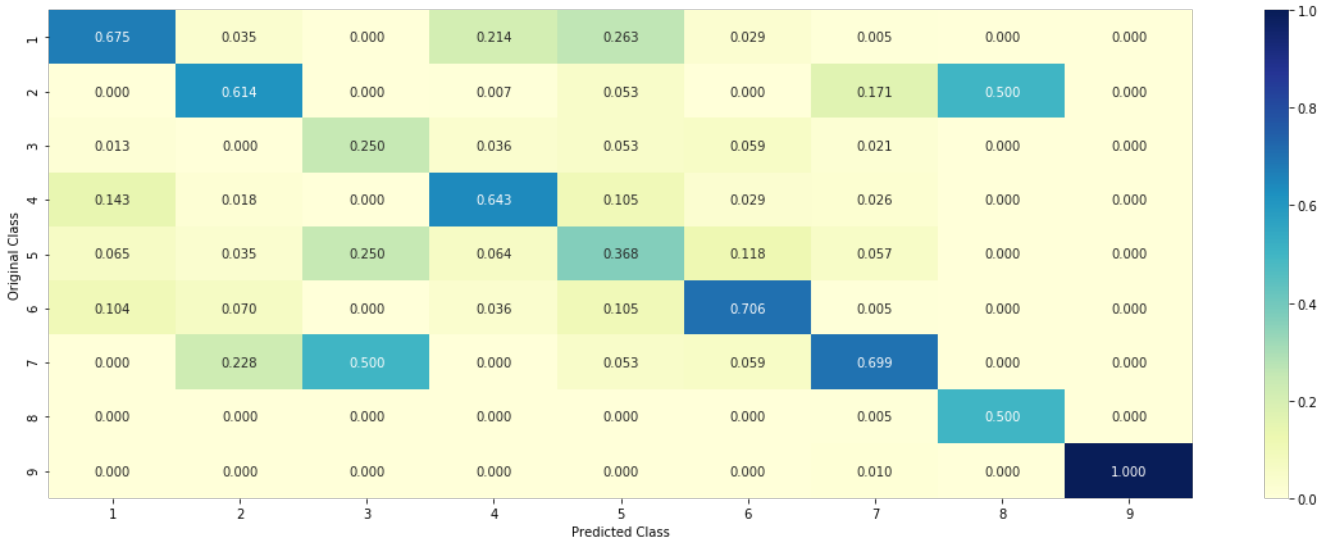
In [236]:

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

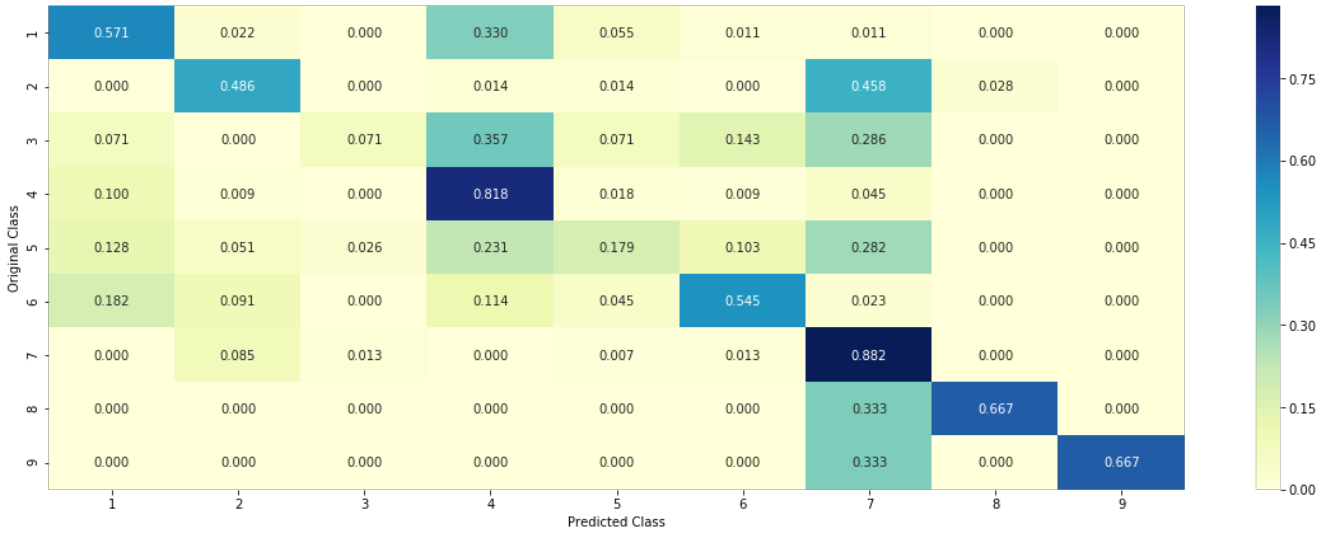
```
Log loss : 1.0304604427129438
Number of mis-classified points : 0.34210526315789475
-------------------- Confusion matrix --------------------
```



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




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



In [237]:

```
# Lets look at few test points
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

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

```
Predicted Class : 6
Actual Class : 7
The  11  nearest neighbours of the test points belongs to classes [2 2 7 7 2 7 7 7 7 7 5]
Fequency of nearest points : Counter({7: 7, 2: 3, 5: 1})
```

In [238]:

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

test_point_index = 100

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

```
Predicted Class : 4
Actual Class : 1
the k value for knn is 11 and the nearest neighbours of the test points belongs to classes [1 4 4
4 4 4 4 4 1 4 4]
Fequency of nearest points : Counter({4: 9, 1: 2})
```

Logistic Regression

Balancing all classes

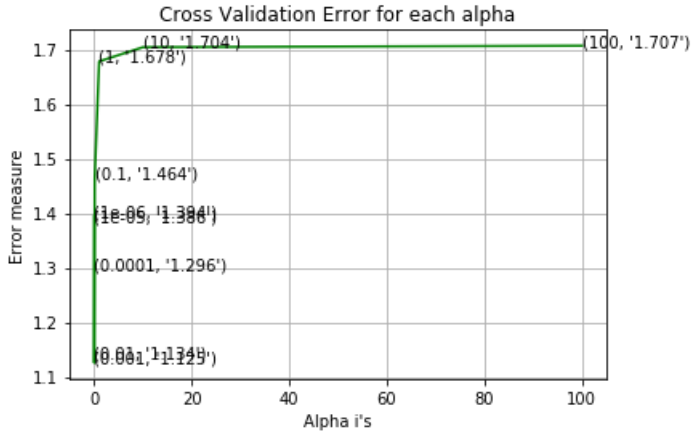
In [239]:

```
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))
```

```
for alpha = 1e-06
Log Loss : 1.3942145409032611
for alpha = 1e-05
Log Loss : 1.3855304656240663
for alpha = 0.0001
Log Loss : 1.296231589711205
for alpha = 0.001
Log Loss : 1.125268306575762
for alpha = 0.01
Log Loss : 1.1342016880199113
for alpha = 0.1
Log Loss : 1.4635560438301263
for alpha = 1
Log Loss : 1.6780405692377318
for alpha = 10
Log Loss : 1.7040761409848242
for alpha = 100
Log Loss : 1.7067588133280587
```

In [240]:

```
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()
```



In [241]:

```
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
clf.fit(train_x_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_lo
ss(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, p
redict_y, labels=clf.classes_, eps=1e-15))
```

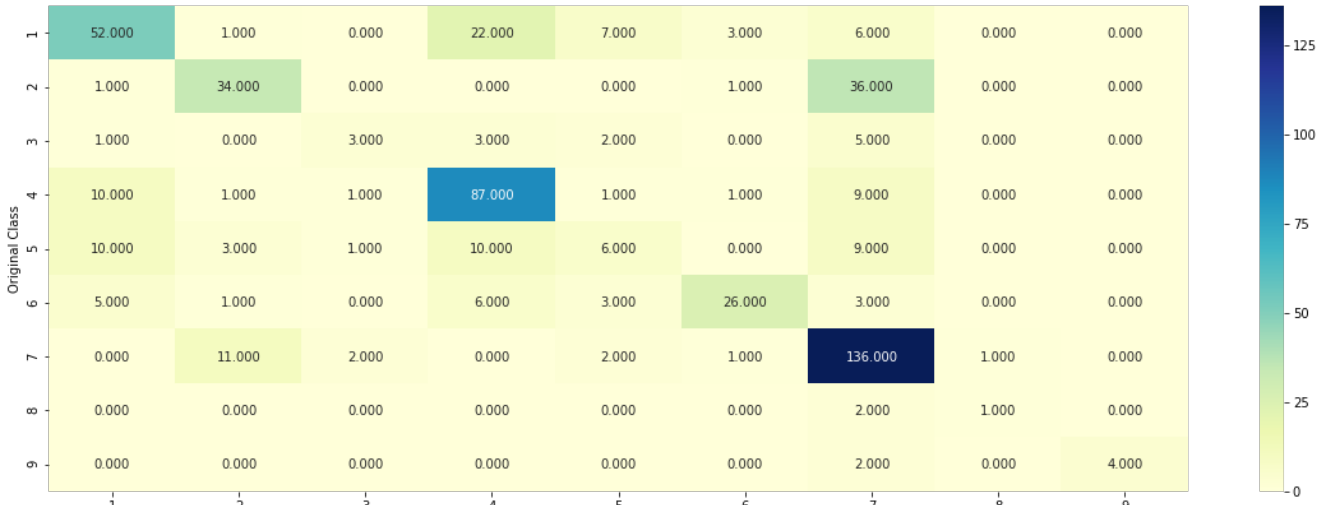
```
For values of best alpha =  0.001 The train log loss is: 0.6169109555505876
For values of best alpha =  0.001 The cross validation log loss is: 1.125268306575762
For values of best alpha =  0.001 The test log loss is: 1.1339637673544194
```

Lets test it on testing data using best alpha value

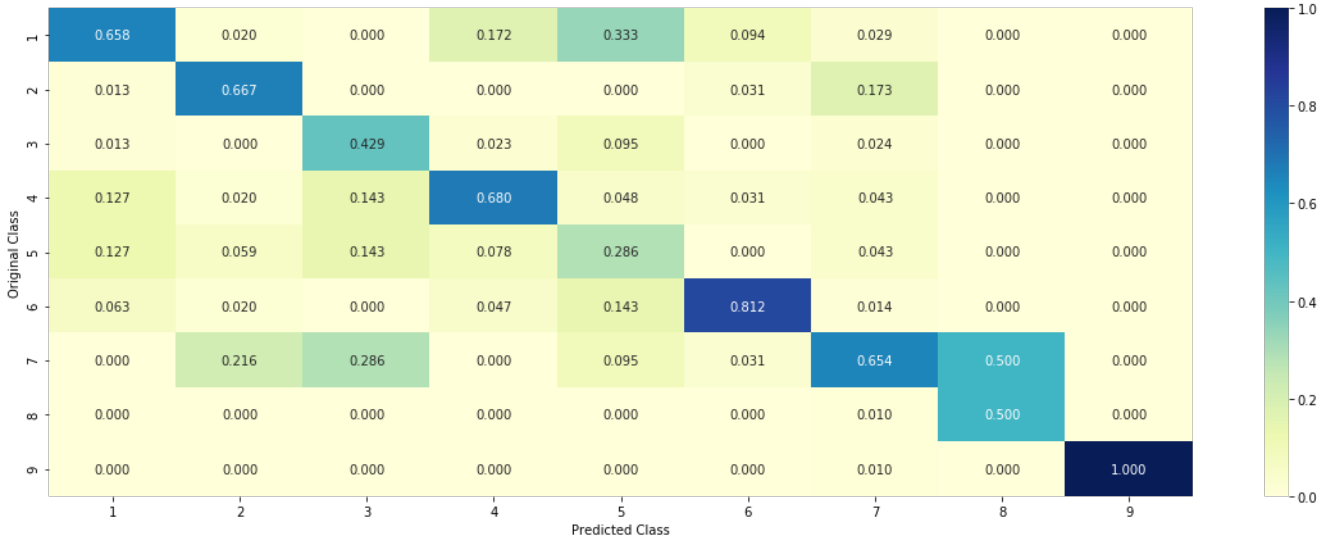
In [242]:

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

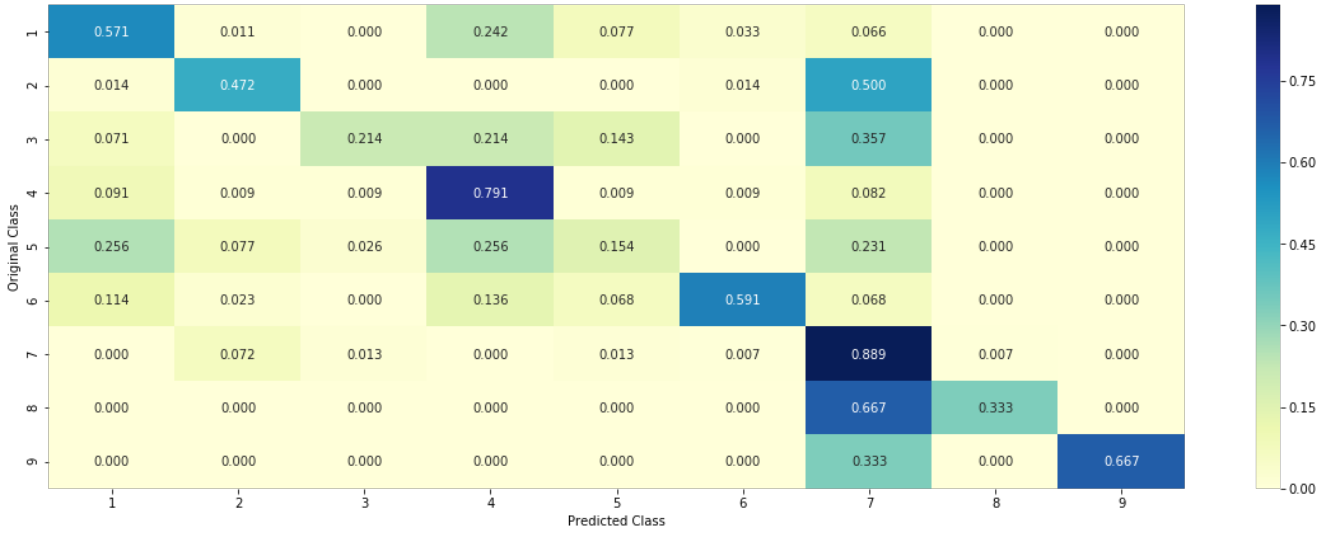
```
Log loss : 1.125268306575762
Number of mis-classified points : 0.34398496240601506
-------------------- Confusion matrix --------------------
```



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



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



Feature importance:

In [243]:

```
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']))
```

Testing query point and Interpretability:

In [244]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
clf.fit(train_x_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: [[0.0076 0.2786 0.0022 0.0048 0.0048 0.0019 0.6897 0.0065 0.0039]]
Actual Class : 7
--------------------------------------------------
57 Text feature [constitutive] present in test data point [True]
88 Text feature [transforming] present in test data point [True]
94 Text feature [oncogene] present in test data point [True]
121 Text feature [cdnas] present in test data point [True]
128 Text feature [activated] present in test data point [True]
130 Text feature [balb] present in test data point [True]
133 Text feature [technology] present in test data point [True]
137 Text feature [subcutaneously] present in test data point [True]
142 Text feature [oncogenes] present in test data point [True]
156 Text feature [recurrence] present in test data point [True]
160 Text feature [houston] present in test data point [True]
168 Text feature [doses] present in test data point [True]
175 Text feature [nude] present in test data point [True]
181 Text feature [downstream] present in test data point [True]
182 Text feature [overexpression] present in test data point [True]
187 Text feature [4695] present in test data point [True]
204 Text feature [tyr204] present in test data point [True]
205 Text feature [iccs] present in test data point [True]
210 Text feature [flex] present in test data point [True]
264 Text feature [138] present in test data point [True]
265 Text feature [pmxs] present in test data point [True]
294 Text feature [inhibited] present in test data point [True]
306 Text feature [tyr705] present in test data point [True]
315 Text feature [specimen] present in test data point [True]
316 Text feature [icc] present in test data point [True]
322 Text feature [anesthetized] present in test data point [True]
350 Text feature [injection] present in test data point [True]
383 Text feature [expressing] present in test data point [True]
386 Text feature [thr202] present in test data point [True]
391 Text feature [rarely] present in test data point [True]
400 Text feature [therapeutics] present in test data point [True]
403 Text feature [activating] present in test data point [True]
482 Text feature [transduced] present in test data point [True]
Out of the top  500  features  33 are present in query point
```

In [245]:

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

```
                    ,test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[6.754e-01 2.750e-02 1.800e-03 1.508e-01 2.000e-03 5.000e-04 1.078
e-01
  2.160e-02 1.260e-02]]
Actual Class : 1
--------------------------------------------------
215 Text feature [meningioma] present in test data point [True]
265 Text feature [boundary] present in test data point [True]
294 Text feature [c8] present in test data point [True]
317 Text feature [mev] present in test data point [True]
375 Text feature [stop] present in test data point [True]
410 Text feature [urls] present in test data point [True]
416 Text feature [v0] present in test data point [True]
431 Text feature [families] present in test data point [True]
433 Text feature [ebss] present in test data point [True]
465 Text feature [abrogating] present in test data point [True]
494 Text feature [paralogs] present in test data point [True]
Out of the top  500  features  11 are present in query point
```

Without Class Balancing

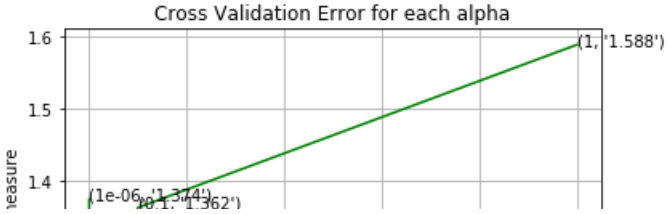
In [246]:

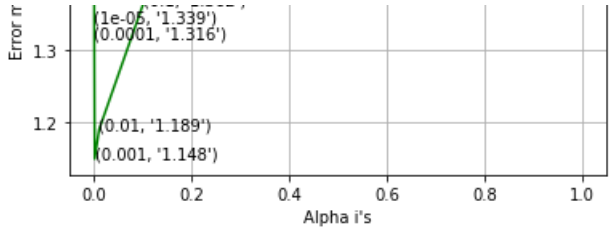
```
alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, 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))
```

```
for alpha = 1e-06
Log Loss : 1.3736065062892857
for alpha = 1e-05
Log Loss : 1.3385260058961135
for alpha = 0.0001
Log Loss : 1.3156940365111653
for alpha = 0.001
Log Loss : 1.1483026568834127
for alpha = 0.01
Log Loss : 1.1888855110924894
for alpha = 0.1
Log Loss : 1.3620289780362065
for alpha = 1
Log Loss : 1.5877397901838608
```

In [247]:

```
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()
```





In [248]:

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

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

```
For values of best alpha =  0.001 The train log loss is: 0.6175165599189917
For values of best alpha =  0.001 The cross validation log loss is: 1.1483026568834127
For values of best alpha =  0.001 The test log loss is: 1.143644063411625
```

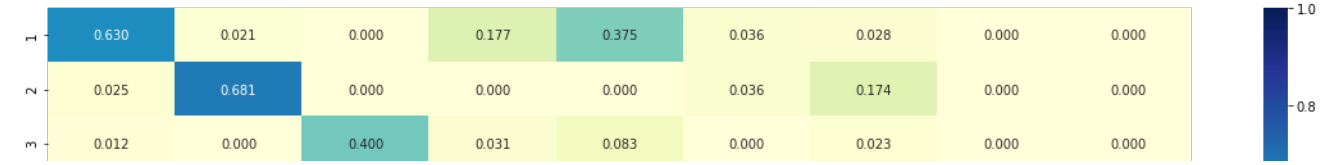
Lets test our model with best hyper param

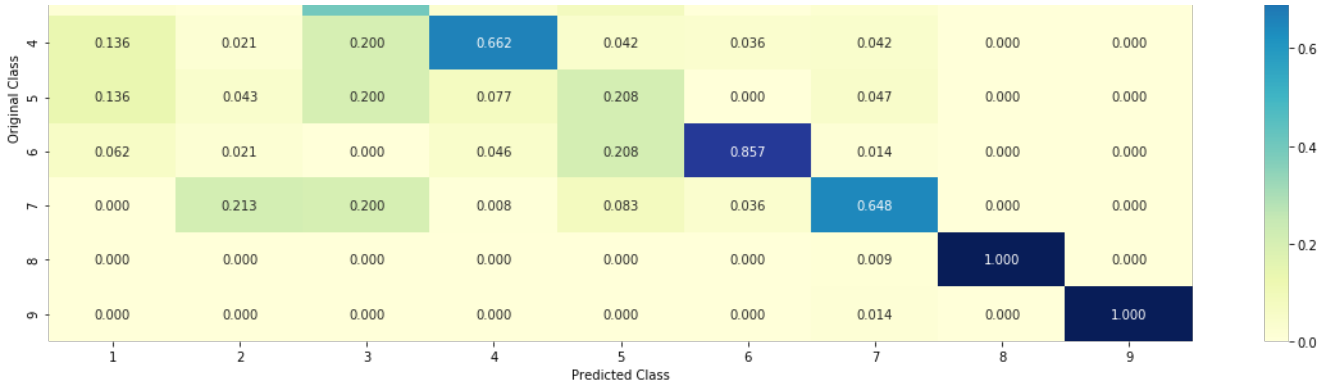
In [249]:

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

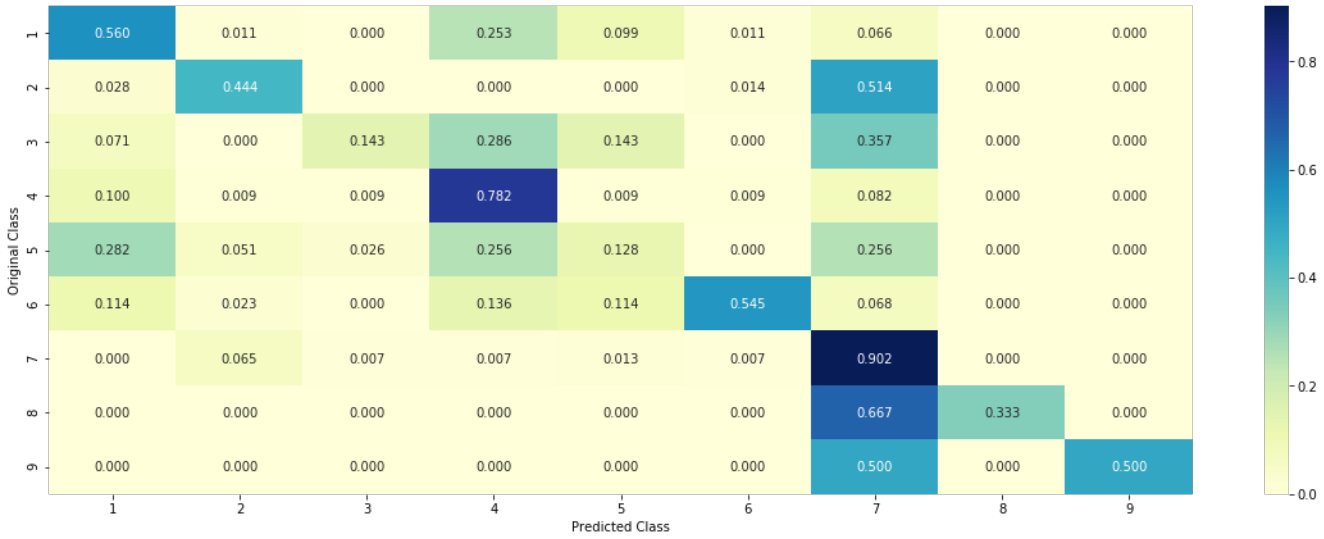
```
Log loss : 1.1483026568834127
Number of mis-classified points : 0.35714285714285715
-------------------- Confusion matrix --------------------
```


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






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



Testing query point and interpretability

In [250]:

```
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: [[8.000e-03 2.524e-01 5.000e-04 7.000e-03 3.200e-03 1.200e-03 7.238
e-01
  3.800e-03 1.000e-04]]
Actual Class : 7
--------------------------------------------------
145 Text feature [transforming] present in test data point [True]
162 Text feature [iccs] present in test data point [True]
164 Text feature [oncogene] present in test data point [True]
169 Text feature [constitutive] present in test data point [True]
180 Text feature [4695] present in test data point [True]
192 Text feature [houston] present in test data point [True]
201 Text feature [subcutaneously] present in test data point [True]
218 Text feature [mir] present in test data point [True]
224 Text feature [cdnas] present in test data point [True]
231 Text feature [technology] present in test data point [True]
```

```
258 Text feature [recurrence] present in test data point [True]
293 Text feature [doses] present in test data point [True]
313 Text feature [activated] present in test data point [True]
326 Text feature [oncogenes] present in test data point [True]
335 Text feature [pmxs] present in test data point [True]
360 Text feature [overexpression] present in test data point [True]
361 Text feature [downstream] present in test data point [True]
369 Text feature [icc] present in test data point [True]
384 Text feature [balb] present in test data point [True]
425 Text feature [swab] present in test data point [True]
427 Text feature [138] present in test data point [True]
430 Text feature [activating] present in test data point [True]
436 Text feature [conventional] present in test data point [True]
455 Text feature [nude] present in test data point [True]
481 Text feature [transduced] present in test data point [True]
488 Text feature [anesthetized] present in test data point [True]
489 Text feature [expressing] present in test data point [True]
498 Text feature [inhibited] present in test data point [True]
Out of the top  500  features  28 are present in query point
```

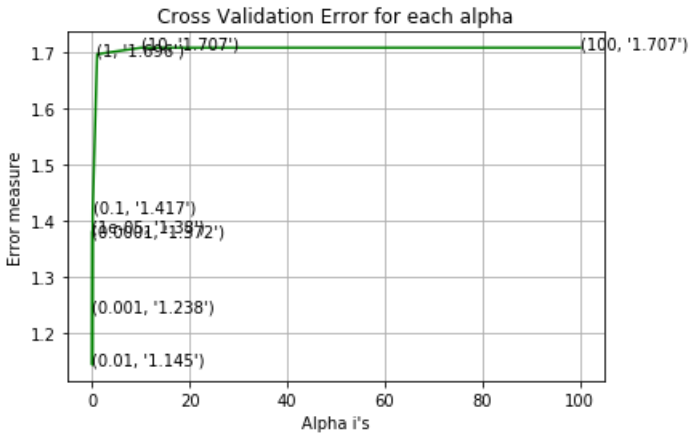
Linear Support Vector Machines

In [251]:

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

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


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

predict_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_lo
ss(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, p
redict_y, labels=clf.classes_, eps=1e-15))
```

```
for C = 1e-05
Log Loss : 1.3802639479271654
for C = 0.0001
Log Loss : 1.3717633861438547
for C = 0.001
Log Loss : 1.238353952522779
for C = 0.01
```

```
Log Loss : 1.144530064704165
for C = 0.1
Log Loss : 1.4169809617362963
for C = 1
Log Loss : 1.6957808741770433
for C = 10
Log Loss : 1.7071785843528657
for C = 100
Log Loss : 1.7071975593595048
```



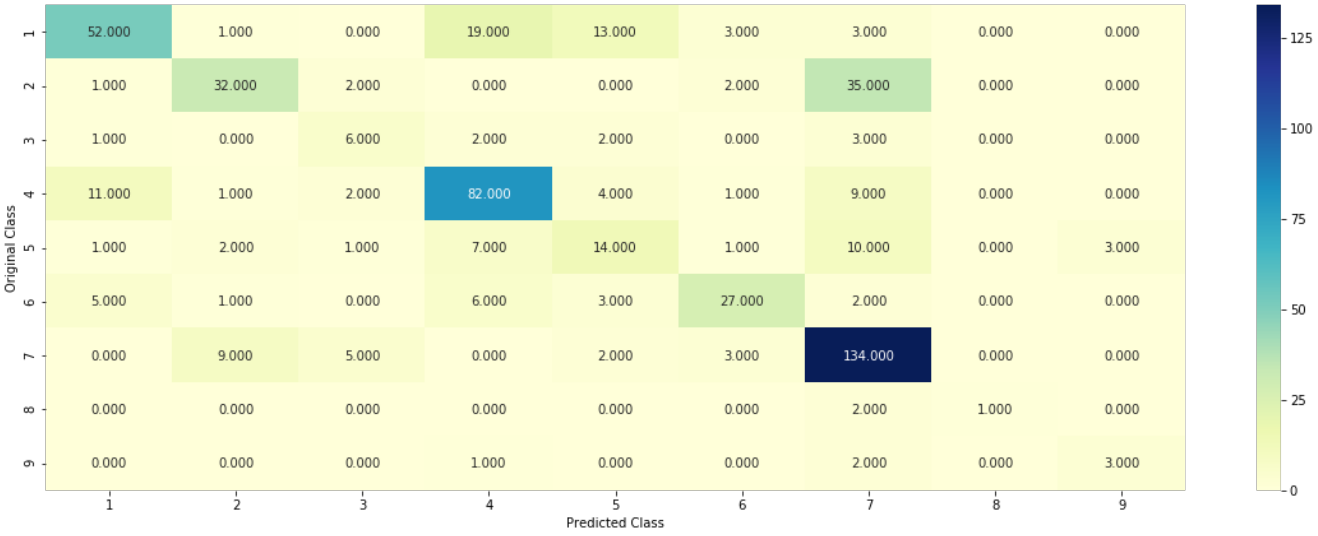

```
For values of best alpha =  0.01 The train log loss is: 0.7520612888538187
For values of best alpha =  0.01 The cross validation log loss is: 1.144530064704165
For values of best alpha =  0.01 The test log loss is: 1.1492073568872678
```

Testing model with best alpha values

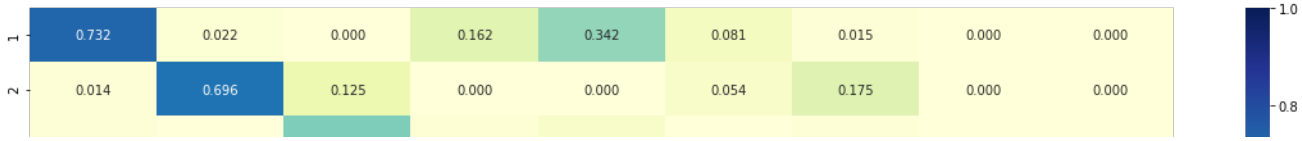
In [252]:

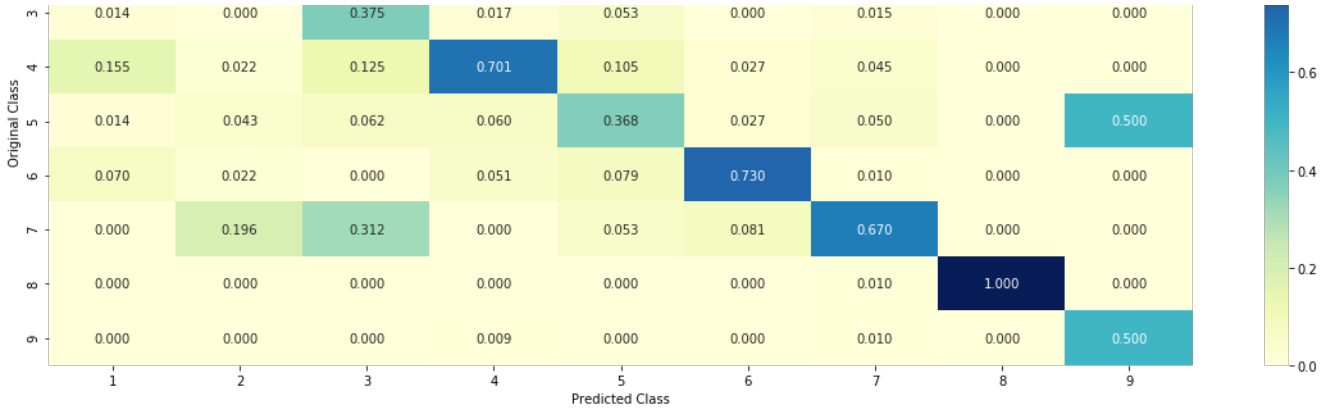
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge',
random_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

```
Log loss : 1.144530064704165
Number of mis-classified points : 0.34022556390977443
-------------------- Confusion matrix --------------------
```

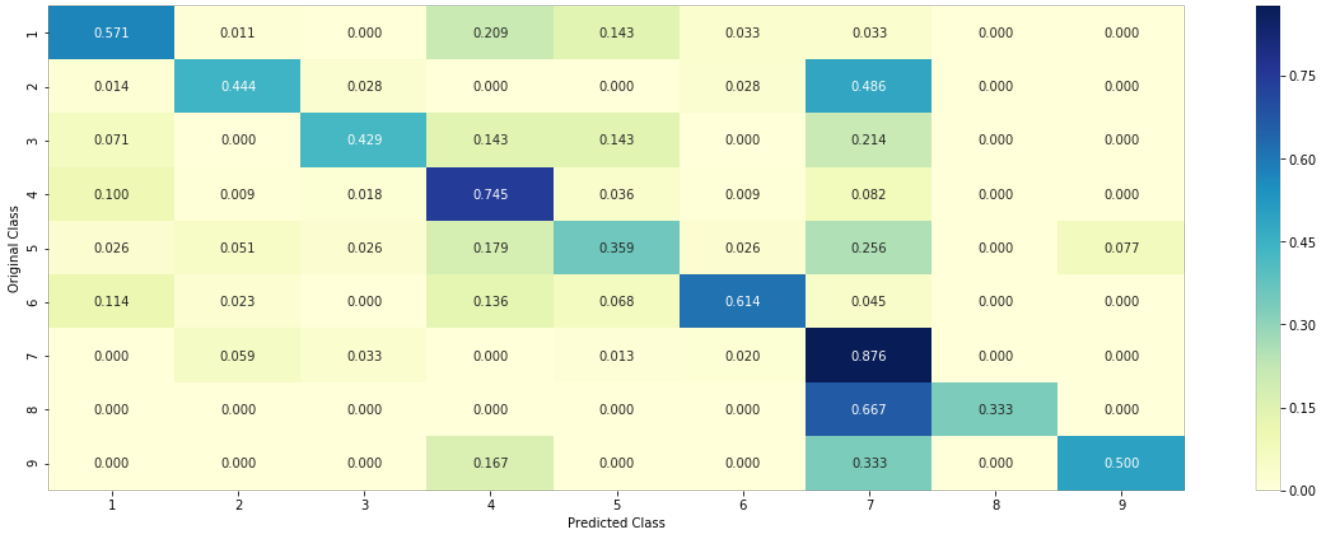



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



Querying some correctly classified point

In [253]:

```
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: [[0.0244 0.3674 0.0061 0.0296 0.0169 0.0069 0.5382 0.0074 0.0031]]
Actual Class : 7
--------------------------------------------------
32 Text feature [cdnas] present in test data point [True]
46 Text feature [subcutaneously] present in test data point [True]
53 Text feature [houston] present in test data point [True]
54 Text feature [transforming] present in test data point [True]
75 Text feature [technology] present in test data point [True]
82 Text feature [nude] present in test data point [True]
95 Text feature [constitutive] present in test data point [True]
96 Text feature [flex] present in test data point [True]
105 Text feature [pmxs] present in test data point [True]
127 Text feature [doses] present in test data point [True]
128 Text feature [oncogene] present in test data point [True]
```

```
142 Text feature [expressing] present in test data point [True]
161 Text feature [oncogenes] present in test data point [True]
166 Text feature [activated] present in test data point [True]
187 Text feature [transduced] present in test data point [True]
216 Text feature [tyr705] present in test data point [True]
243 Text feature [downstream] present in test data point [True]
244 Text feature [activating] present in test data point [True]
250 Text feature [intractable] present in test data point [True]
259 Text feature [injection] present in test data point [True]
264 Text feature [overexpression] present in test data point [True]
271 Text feature [epitomics] present in test data point [True]
288 Text feature [balb] present in test data point [True]
301 Text feature [icc] present in test data point [True]
317 Text feature [concentrations] present in test data point [True]
331 Text feature [tyr204] present in test data point [True]
332 Text feature [thyroid] present in test data point [True]
333 Text feature [138] present in test data point [True]
336 Text feature [inhibited] present in test data point [True]
346 Text feature [starved] present in test data point [True]
348 Text feature [selleck] present in test data point [True]
355 Text feature [atcc] present in test data point [True]
377 Text feature [phospho] present in test data point [True]
383 Text feature [specimen] present in test data point [True]
406 Text feature [therapeutics] present in test data point [True]
431 Text feature [coiled] present in test data point [True]
432 Text feature [invasive] present in test data point [True]
433 Text feature [ccdc6] present in test data point [True]
459 Text feature [4695] present in test data point [True]
Out of the top  500  features  39 are present in query point
```

Random Forest Classifier

Model with One hot encoder

In [254]:

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2468117501802753
for n_estimators = 100 and max depth =  10
Log Loss : 1.18443699159111
for n_estimators = 200 and max depth =  5
Log Loss : 1.235364192329643
for n_estimators = 200 and max depth =  10
Log Loss : 1.172324203551975
for n_estimators = 500 and max depth =  5
Log Loss : 1.2297019575070018
for n_estimators = 500 and max depth =  10
Log Loss : 1.168223394005894
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2292297831611891
for n_estimators = 1000 and max depth =  10
Log Loss : 1.164133526584798
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2255347277572484
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1642663007273435
```

In [255]:

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

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

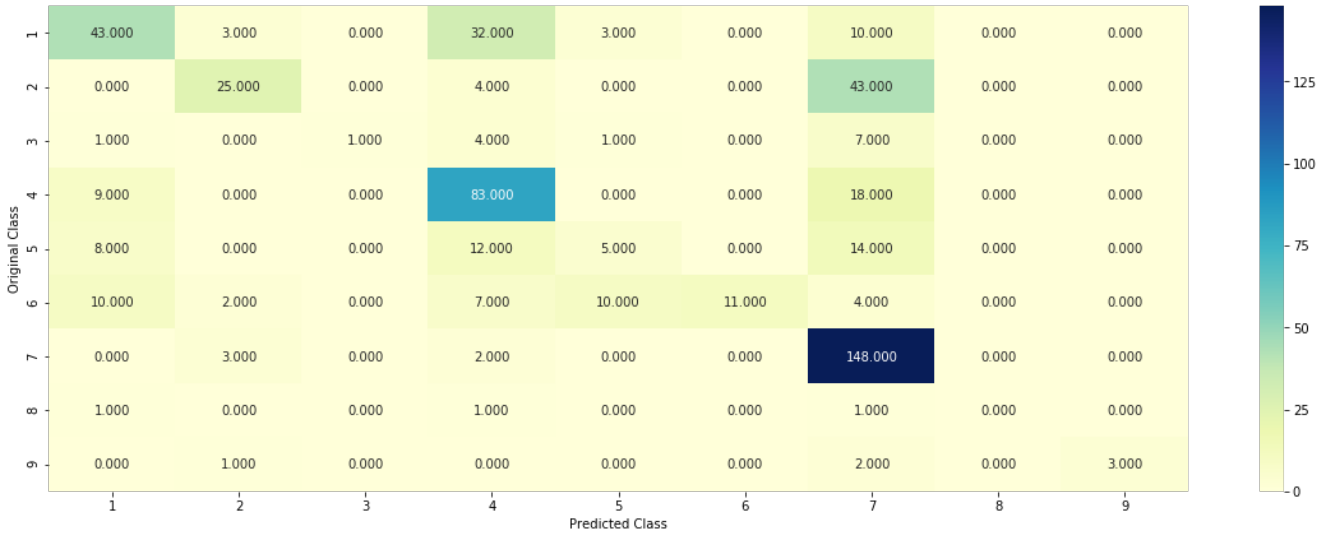
```
For values of best estimator =  1000 The train log loss is: 0.7021436289629714
For values of best estimator =  1000 The cross validation log loss is: 1.164133526584798
For values of best estimator =  1000 The test log loss is: 1.1697292594803708
```

Lets test it on testing data using best hyper param

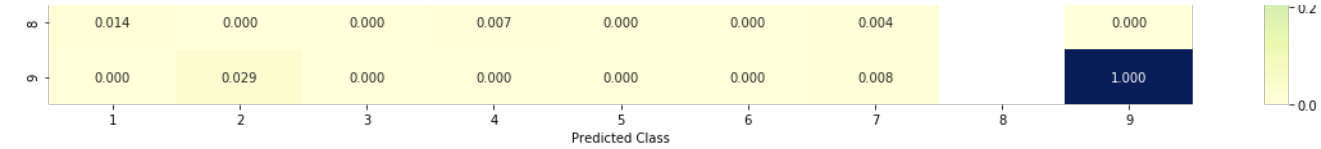
In [256]:

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

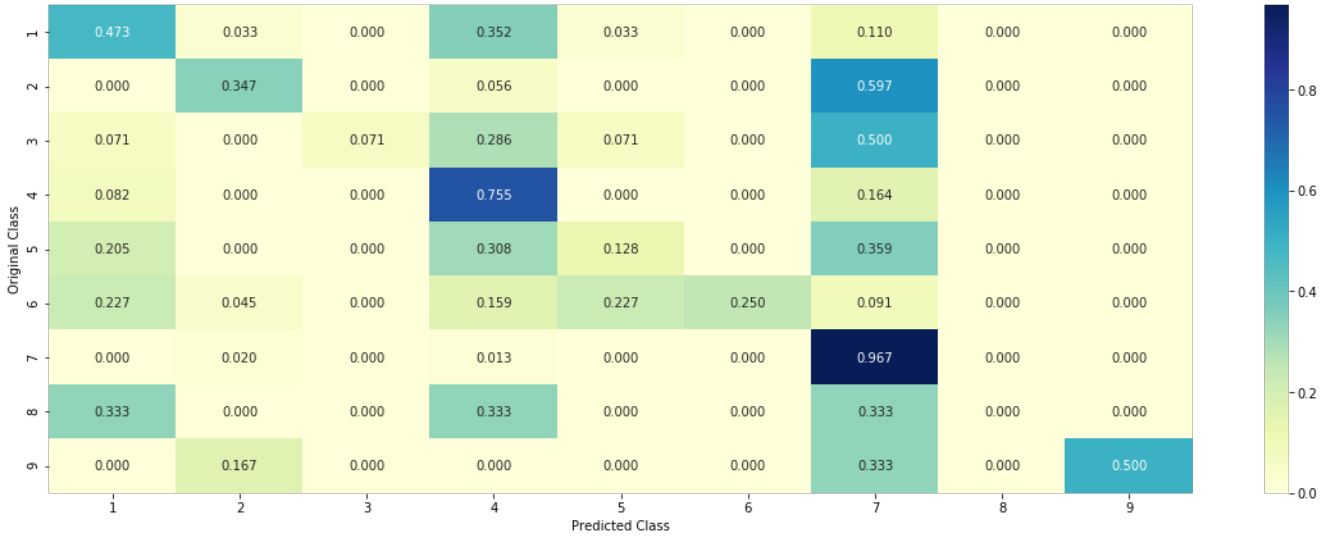
```
Log loss : 1.164133526584798
Number of mis-classified points : 0.40037593984962405
-------------------- Confusion matrix --------------------
```




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



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



In [257]:

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

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

test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0527 0.2464 0.0136 0.038  0.031  0.0299 0.5787 0.0049 0.0048]]
Actual Class : 7
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [tyrosine] present in test data point [True]
2 Text feature [constitutive] present in test data point [True]
3 Text feature [activation] present in test data point [True]
4 Text feature [activating] present in test data point [True]
5 Text feature [inhibitors] present in test data point [True]
8 Text feature [activated] present in test data point [True]
9 Text feature [phosphorylation] present in test data point [True]
10 Text feature [treatment] present in test data point [True]
12 Text feature [function] present in test data point [True]
13 Text feature [growth] present in test data point [True]
15 Text feature [signaling] present in test data point [True]
16 Text feature [cells] present in test data point [True]
17 Text feature [inhibitor] present in test data point [True]
18 Text feature [activate] present in test data point [True]
19 Text feature [suppressor] present in test data point [True]
21 Text feature [receptor] present in test data point [True]
22 Text feature [oncogenic] present in test data point [True]
23 Text feature [kinases] present in test data point [True]
```

```
25 Text feature [akt] present in test data point [True]
26 Text feature [months] present in test data point [True]
27 Text feature [loss] present in test data point [True]
28 Text feature [therapy] present in test data point [True]
30 Text feature [pathogenic] present in test data point [True]
31 Text feature [efficacy] present in test data point [True]
32 Text feature [transforming] present in test data point [True]
34 Text feature [trials] present in test data point [True]
35 Text feature [variants] present in test data point [True]
39 Text feature [protein] present in test data point [True]
41 Text feature [patients] present in test data point [True]
43 Text feature [cell] present in test data point [True]
45 Text feature [drug] present in test data point [True]
46 Text feature [downstream] present in test data point [True]
47 Text feature [functional] present in test data point [True]
49 Text feature [proliferation] present in test data point [True]
50 Text feature [expressing] present in test data point [True]
53 Text feature [inhibition] present in test data point [True]
57 Text feature [harboring] present in test data point [True]
58 Text feature [therapeutic] present in test data point [True]
60 Text feature [ras] present in test data point [True]
61 Text feature [tki] present in test data point [True]
64 Text feature [treated] present in test data point [True]
65 Text feature [mapk] present in test data point [True]
66 Text feature [advanced] present in test data point [True]
68 Text feature [clinical] present in test data point [True]
71 Text feature [expression] present in test data point [True]
72 Text feature [erk1] present in test data point [True]
74 Text feature [survival] present in test data point [True]
75 Text feature [sensitivity] present in test data point [True]
76 Text feature [lines] present in test data point [True]
77 Text feature [tkis] present in test data point [True]
78 Text feature [variant] present in test data point [True]
79 Text feature [potential] present in test data point [True]
83 Text feature [inactivation] present in test data point [True]
84 Text feature [imatinib] present in test data point [True]
86 Text feature [inhibited] present in test data point [True]
87 Text feature [assays] present in test data point [True]
89 Text feature [trial] present in test data point [True]
90 Text feature [sensitive] present in test data point [True]
92 Text feature [proteins] present in test data point [True]
93 Text feature [transformation] present in test data point [True]
95 Text feature [phospho] present in test data point [True]
99 Text feature [assay] present in test data point [True]
Out of the top  100  features  63 are present in query point
```

RF with Response Coding

In [258]:

```
alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42
, n_jobs=-1)
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))


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

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log loss is:",log_loss(y
```

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

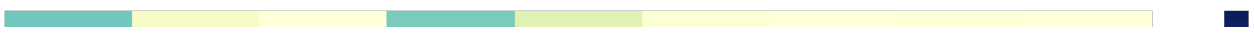
```
for n_estimators = 10 and max depth =  2
Log Loss : 1.9696326880410098
for n_estimators = 10 and max depth =  3
Log Loss : 1.6973923458447513
for n_estimators = 10 and max depth =  5
Log Loss : 1.575572335287424
for n_estimators = 10 and max depth =  10
Log Loss : 1.6354995588458099
for n_estimators = 50 and max depth =  2
Log Loss : 1.6978185436308586
for n_estimators = 50 and max depth =  3
Log Loss : 1.4304257486073455
for n_estimators = 50 and max depth =  5
Log Loss : 1.4502243888068964
for n_estimators = 50 and max depth =  10
Log Loss : 1.6971318415963736
for n_estimators = 100 and max depth =  2
Log Loss : 1.5298531050781068
for n_estimators = 100 and max depth =  3
Log Loss : 1.4874156127870042
for n_estimators = 100 and max depth =  5
Log Loss : 1.3584023556232825
for n_estimators = 100 and max depth =  10
Log Loss : 1.6697475129011614
for n_estimators = 200 and max depth =  2
Log Loss : 1.592471159201734
for n_estimators = 200 and max depth =  3
Log Loss : 1.527128109294702
for n_estimators = 200 and max depth =  5
Log Loss : 1.4001558468023685
for n_estimators = 200 and max depth =  10
Log Loss : 1.5997031486748738
for n_estimators = 500 and max depth =  2
Log Loss : 1.69358446303512
for n_estimators = 500 and max depth =  3
Log Loss : 1.6062167119628794
for n_estimators = 500 and max depth =  5
Log Loss : 1.3979650711704223
for n_estimators = 500 and max depth =  10
Log Loss : 1.6592440708447487
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6624585912465224
for n_estimators = 1000 and max depth =  3
Log Loss : 1.5993892702402066
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3983844555293026
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6278786176127698
For values of best alpha =  100 The train log loss is: 0.054351076743951725
For values of best alpha =  100 The cross validation log loss is: 1.3584023556232823
For values of best alpha =  100 The test log loss is: 1.389994653526336
```

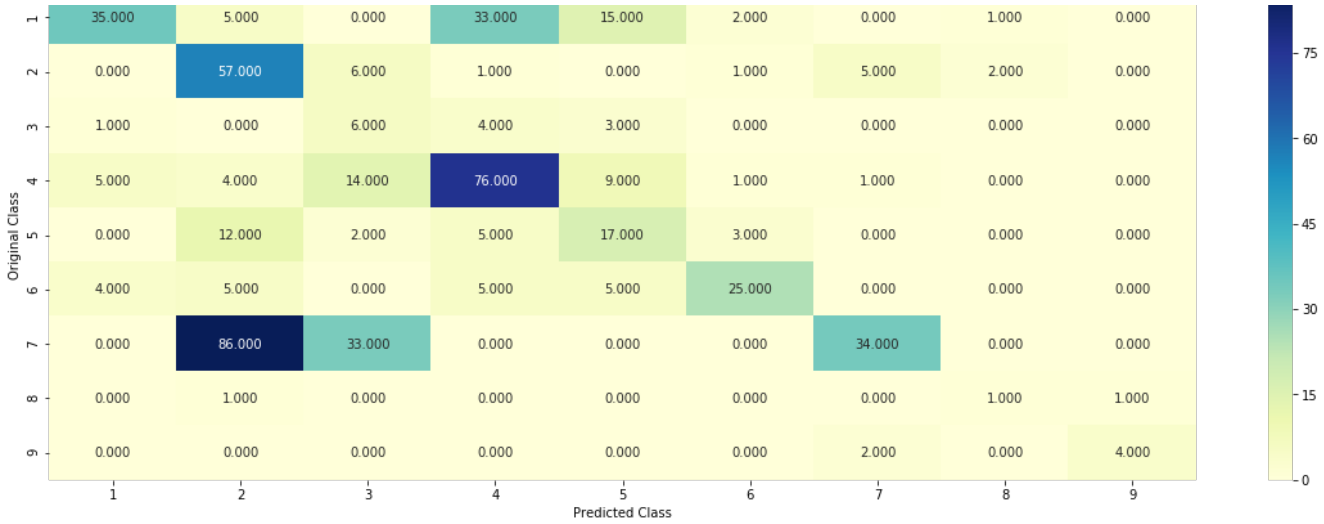
Testing model with best hyper param

In [259]:

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

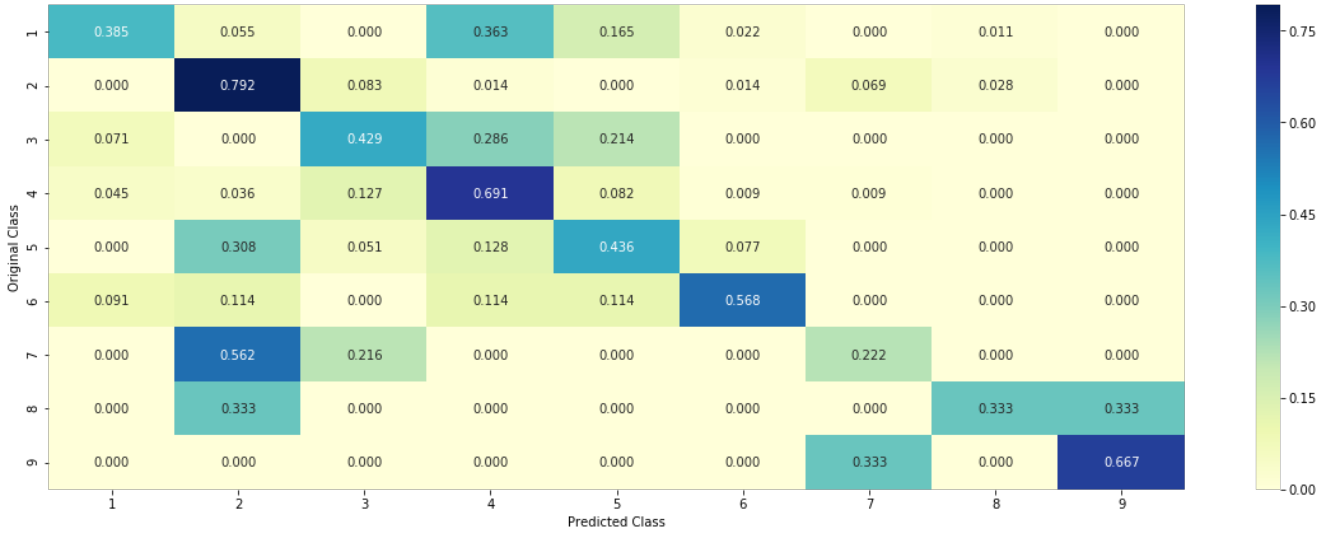
```
Log loss : 1.3584023556232823
Number of mis-classified points : 0.5206766917293233
-------------------- Confusion matrix --------------------
```





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

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



Query the classified point

In [ ]:

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

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


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

Stacking model

In [261]:

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

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


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

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

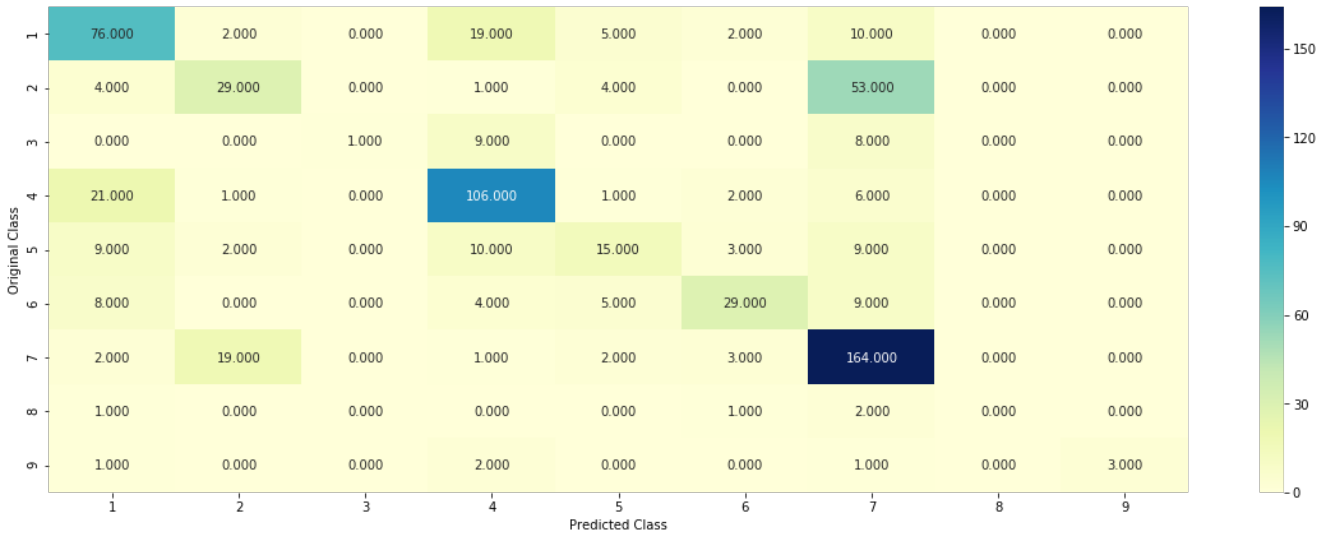
```
Logistic Regression :  Log Loss: 1.13
Support vector machines : Log Loss: 1.70
Naive Bayes : Log Loss: 1.25
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.179
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.042
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.528
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.125
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.216
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.477
```

Testing with best hyper param:

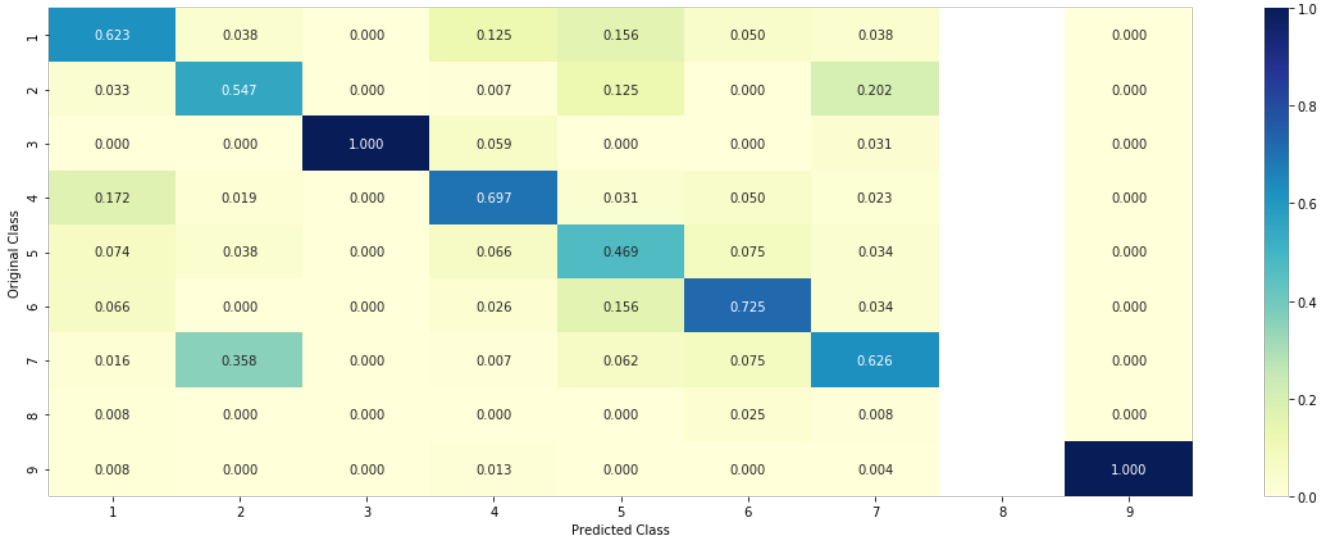
In [262]:

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

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

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

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

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

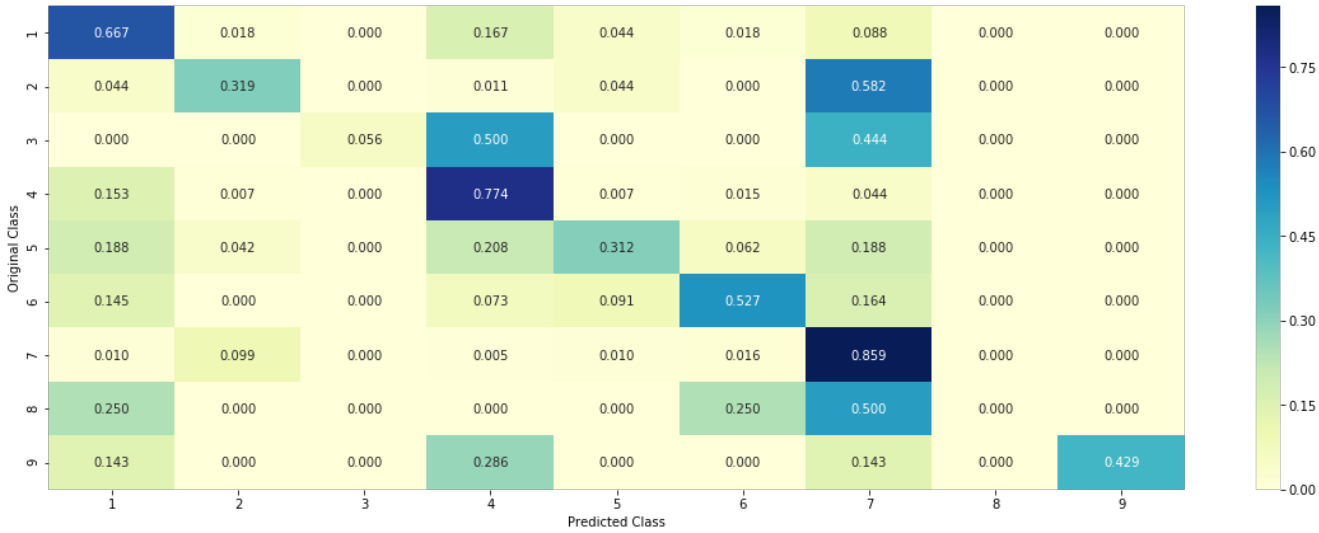
```
Log loss (train) on the stacking classifier : 0.6790802038161988
Log loss (CV) on the stacking classifier : 1.124512101363307
Log loss (test) on the stacking classifier : 1.1548091143045933
Number of missclassified point : 0.36390977443609024
-------------------- Confusion matrix --------------------
```




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




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

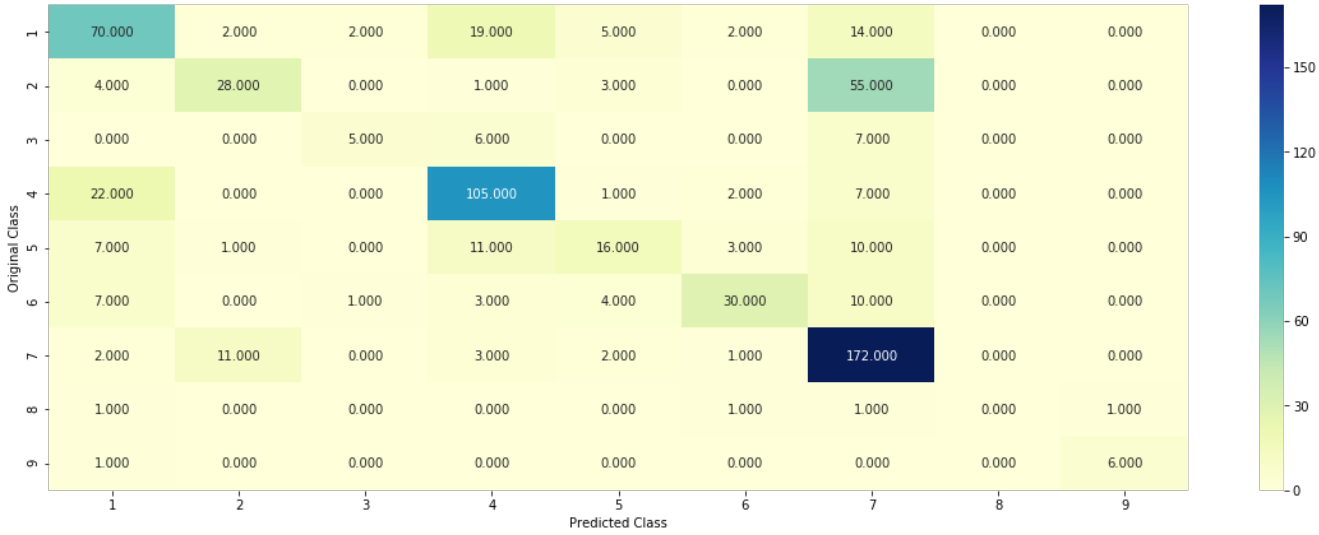


Maximum voting Classifier

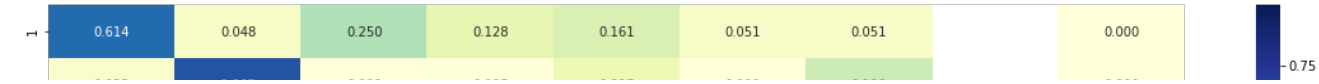
In [263]:

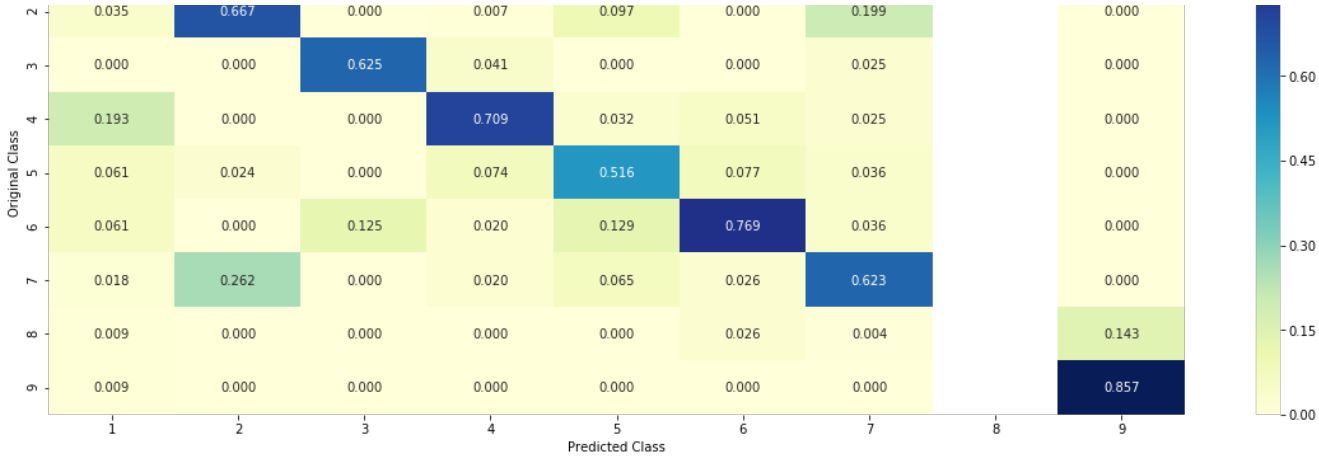
```
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting=
'soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y,
vclf.predict_proba(train_x_onehotCoding)))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

```
Log loss (train) on the VotingClassifier : 0.9319376669588304
Log loss (CV) on the VotingClassifier : 1.2160157662465771
Log loss (test) on the VotingClassifier : 1.2169491271467225
Number of missclassified point : 0.35037593984962406
-------------------- Confusion matrix --------------------
```

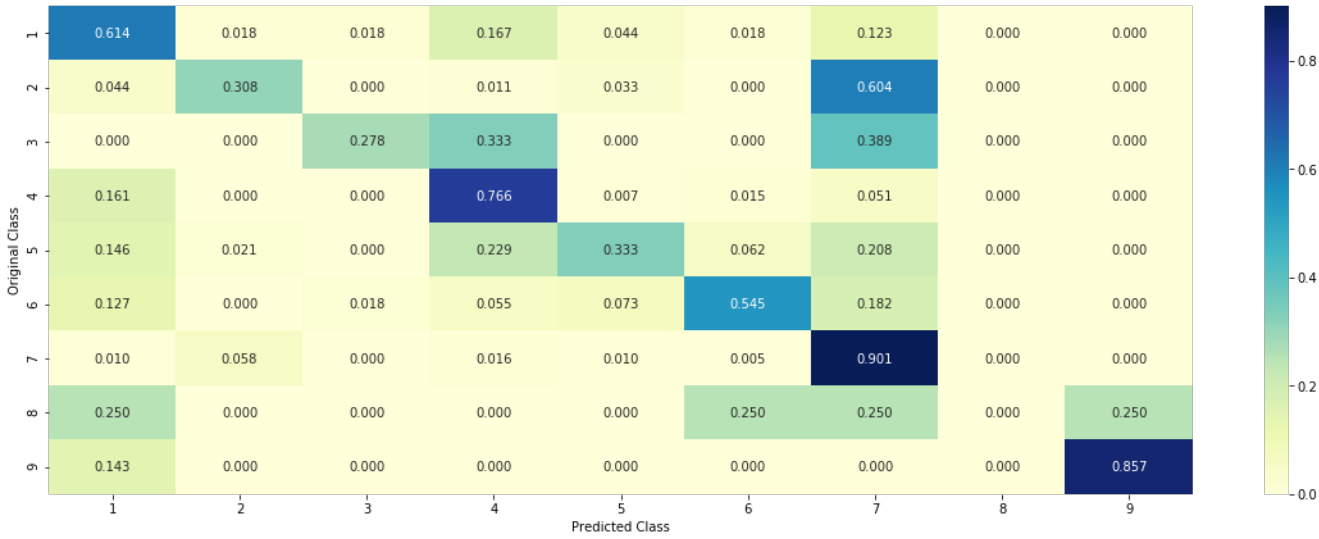



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



Observation:

Overall Linear SVM, Logistic Regression, kNearest Neighbour Models performed well with less number of misclassification points and low LogLoss Metric.