Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [294]:
```
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
#!pip install sklearn.cross_validation
#from sklearn.cross_validation import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")
!pip install mlxtend
from mlxtend.classifier import StackingClassifier

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

```
Requirement already satisfied: mlxtend in c:\users\user\anaconda3\lib\site-packages (0.15.0.0)
Requirement already satisfied: scipy>=0.17 in c:\users\user\anaconda3\lib\site-packages (from mlxtend) (1.0.0)
Requirement already satisfied: matplotlib>=1.5.1 in c:\users\user\anaconda3\lib\site-packages (from mlxtend) (2.
1.2)
Requirement already satisfied: scikit-learn>=0.18 in c:\users\user\anaconda3\lib\site-packages (from mlxtend)
(0.19.1)
Requirement already satisfied: pandas>=0.17.1 in c:\users\user\anaconda3\lib\site-packages (from mlxtend) (0.22.
0)
Requirement already satisfied: setuptools in c:\users\user\anaconda3\lib\site-packages (from mlxtend) (39.2.0)
Requirement already satisfied: numpy>=1.10.4 in c:\users\user\anaconda3\lib\site-packages (from mlxtend) (1.14.
3)
Requirement already satisfied: six>=1.10 in c:\users\user\anaconda3\lib\site-packages (from matplotlib>=1.5.1->m
lxtend) (1.11.0)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\user\anaconda3\lib\site-packages (from matplotli
b>=1.5.1->mlxtend) (2.6.1)
Requirement already satisfied: pytz in c:\users\user\anaconda3\lib\site-packages (from matplotlib>=1.5.1->mlxten
d) (2017.3)
Requirement already satisfied: cycler>=0.10 in c:\users\user\anaconda3\lib\site-packages (from matplotlib>=1.5.1
->mlxtend) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in c:\users\user\anaconda3\lib\site-pack
ages (from matplotlib>=1.5.1->mlxtend) (2.2.0)

You are using pip version 18.0, however version 19.0.3 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [295]:
```
data = pd.read_csv('training_variants')
print('Number of data points : ', data.shape[0])
print('Number of features : ', data.shape[1])
print('Features : ', data.columns.values)
data.head()
```

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

Out[295]:

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

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

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

3.1.2. Reading Text Data

In [296]:
```
# note the seprator in this file
data_text =pd.read_csv("training_text",sep="\|\|",engine="python",names=["ID","TEXT"],skiprows=1)
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

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

Out[296]:

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

3.1.3. Preprocessing of text

In [297]:
```python
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))


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

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

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

In [298]:
```python
#text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_time, "seconds")
```

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 933.7852256206243 seconds
```

In [299]:
```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[299]:

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

In [300]:
```
result[result.isnull().any(axis=1)]
```

Out[300]:

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

In [301]:
```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

In [302]:
```
result[result['ID']==1109]
```

Out[302]:

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

In [303]:
```
result.describe()
```

Out[303]:

	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

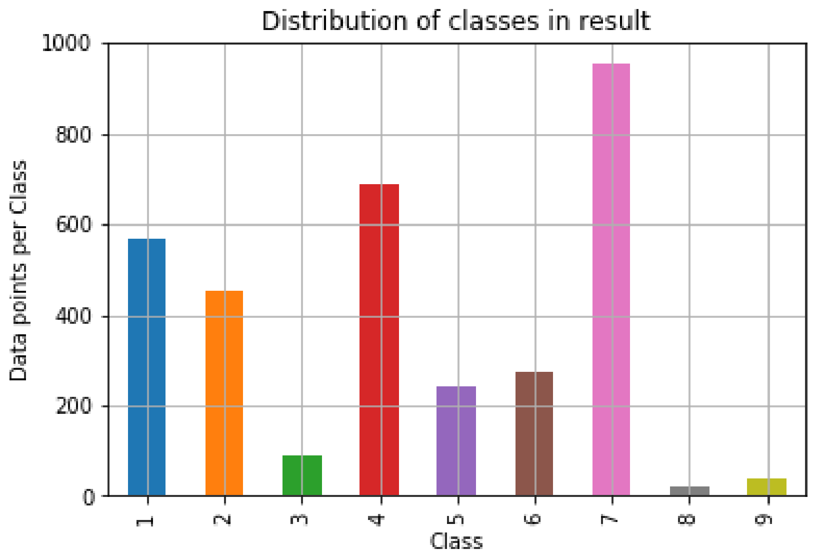
In [304]:
```
result["Gene"].describe()
```

Out[304]:
```
count      3321
unique      264
top       BRCA1
freq        264
Name: Gene, dtype: object
```

```
In [305]: result["Variation"].describe()
```

```
Out[305]: count                     3321
          unique                    2996
          top       Truncating Mutations
          freq                        93
          Name: Variation, dtype: object
```

In [306]:
```
result_class = result['Class'].value_counts().sortlevel()

my_colors = 'rgbkymc'
result_class.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of classes in result')
plt.grid()
plt.show()

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



```
Number of data points in class 7 : 953 ( 28.696 %)
Number of data points in class 4 : 686 ( 20.656 %)
Number of data points in class 1 : 568 ( 17.103 %)
Number of data points in class 2 : 452 ( 13.61 %)
Number of data points in class 6 : 275 ( 8.281 %)
Number of data points in class 5 : 242 ( 7.287 %)
Number of data points in class 3 : 89 ( 2.68 %)
Number of data points in class 9 : 37 ( 1.114 %)
Number of data points in class 8 : 19 ( 0.572 %)
```

3.1.4. Test, Train and Cross Validation Split

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

In [307]:
```
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

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

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

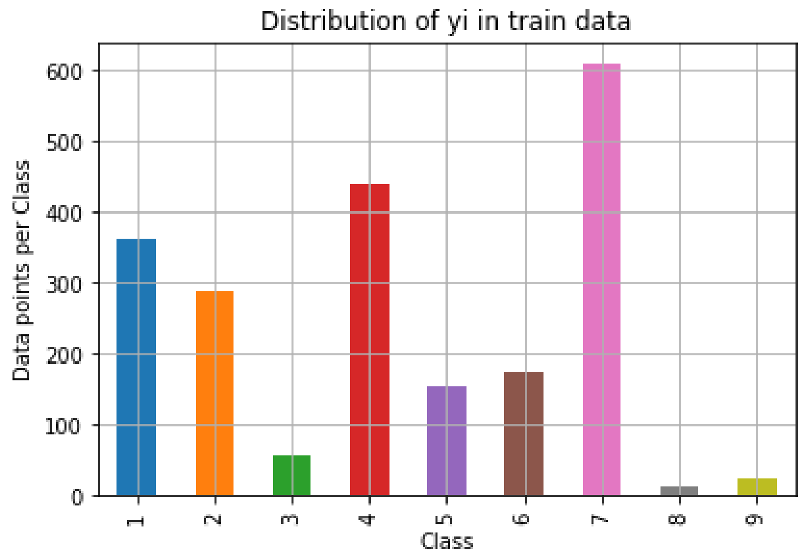
In [308]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

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

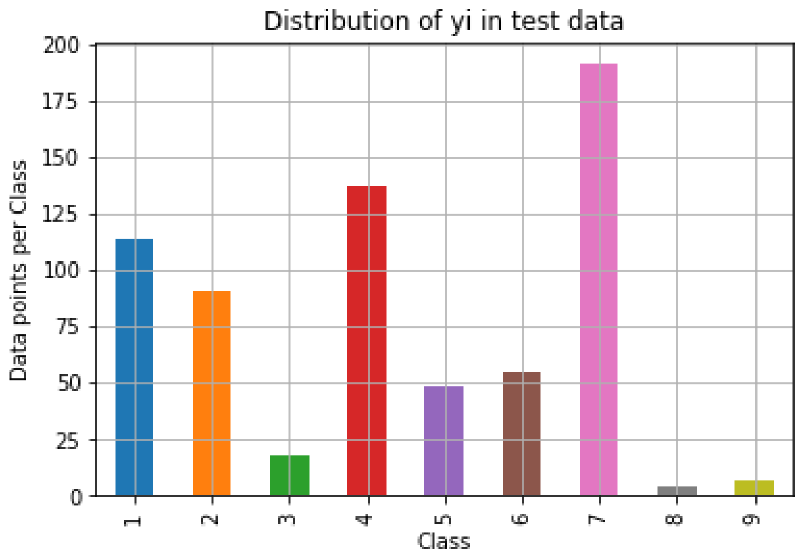
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

In [309]:
```
# it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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

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


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

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

print('-'*80)
my_colors = 'rgbkymc'
```

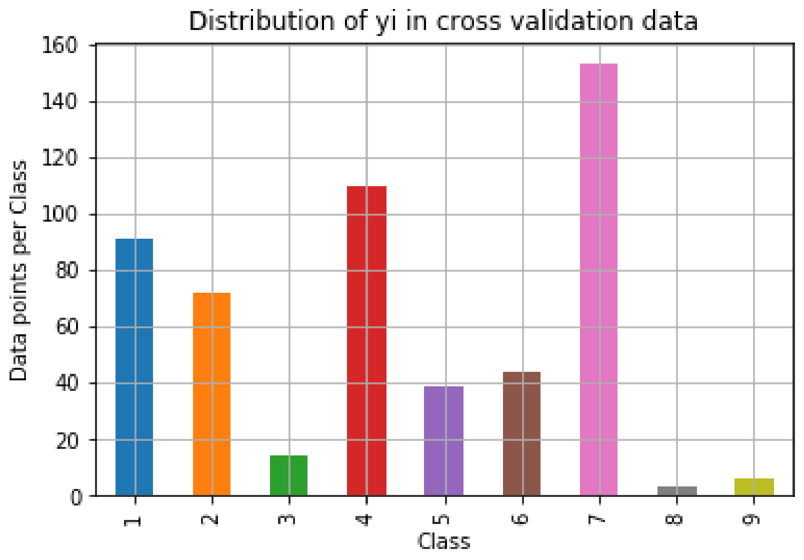
```
cv_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

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



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



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



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

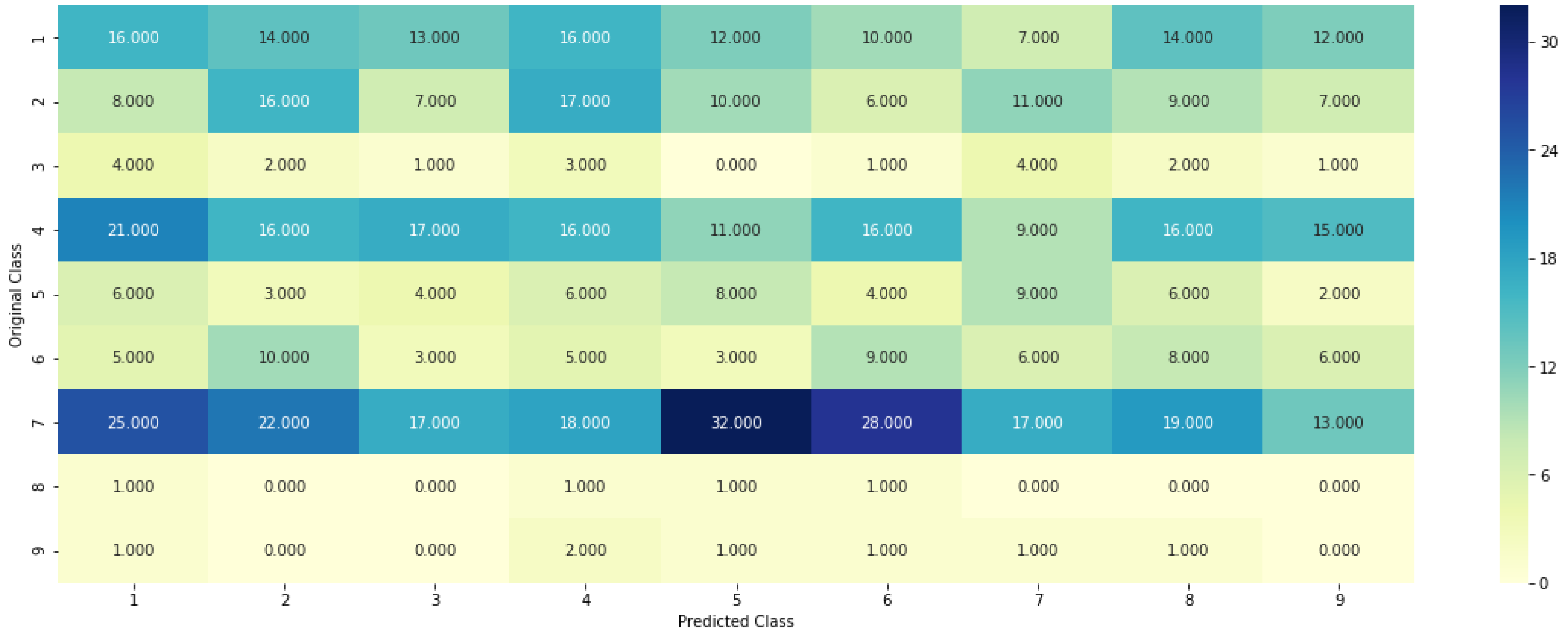
3.2 Prediction using a 'Random' Model

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

In [310]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

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

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

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

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

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

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

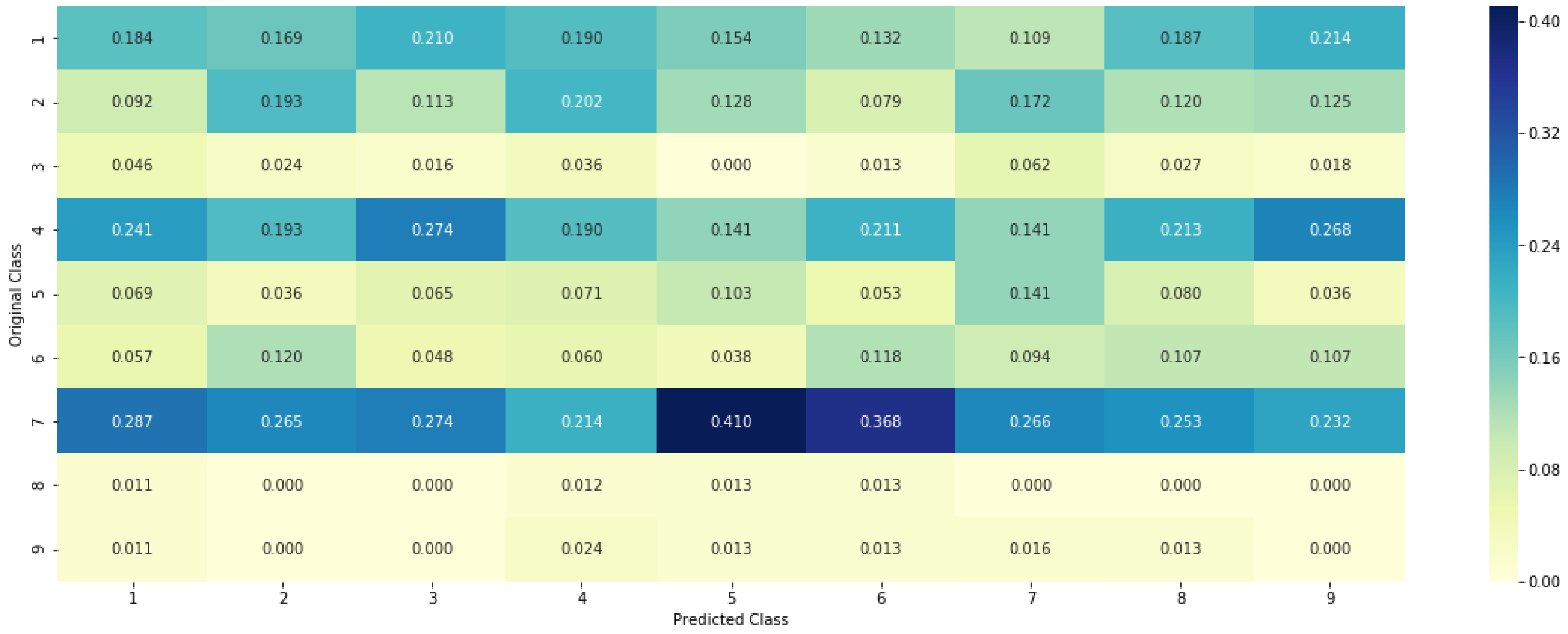
In [311]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

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


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

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

```
Log loss on Cross Validation Data using Random Model 2.481495323004776
Log loss on Test Data using Random Model 2.5151914943272855
-------------------- Confusion matrix --------------------
```

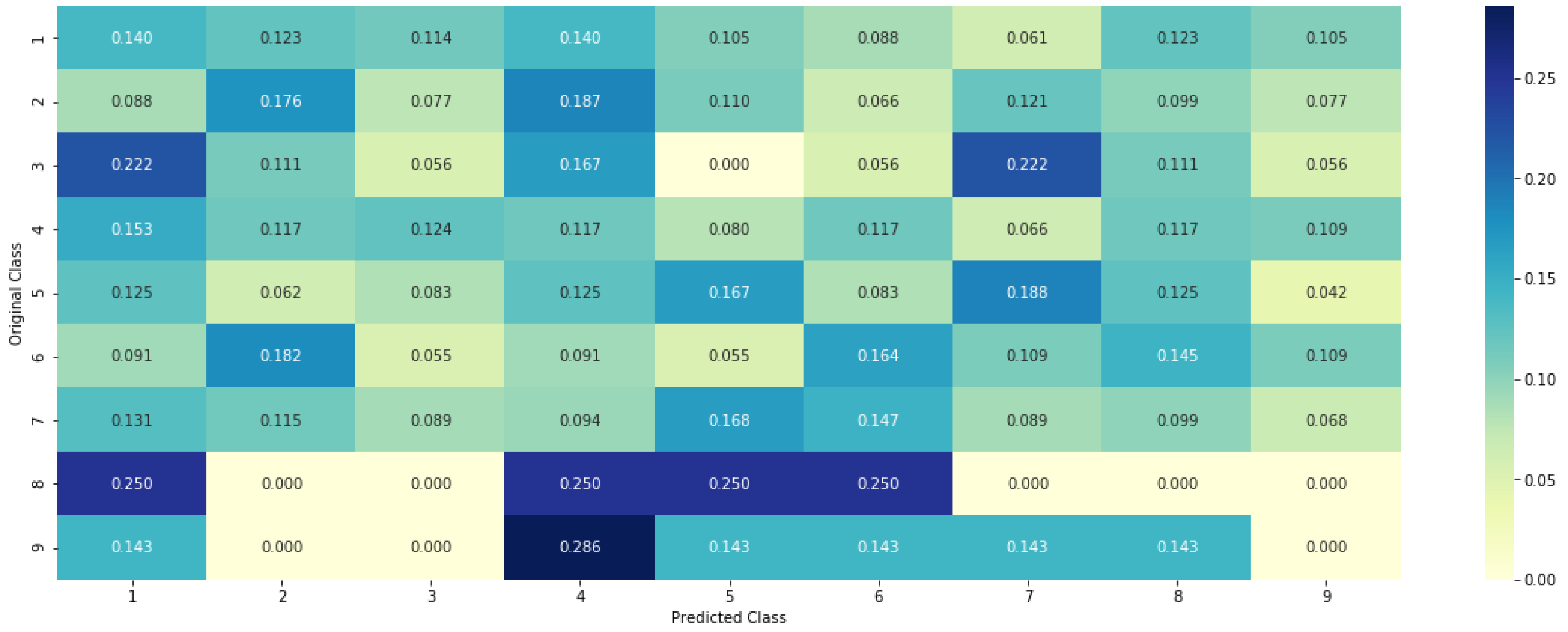


Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	16.000	14.000	13.000	16.000	12.000	10.000	7.000	14.000	12.000
2	8.000	16.000	7.000	17.000	10.000	6.000	11.000	9.000	7.000
3	4.000	2.000	1.000	3.000	0.000	1.000	4.000	2.000	1.000
4	21.000	16.000	17.000	16.000	11.000	16.000	9.000	16.000	15.000
5	6.000	3.000	4.000	6.000	8.000	4.000	9.000	6.000	2.000
6	5.000	10.000	3.000	5.000	3.000	9.000	6.000	8.000	6.000
7	25.000	22.000	17.000	18.000	32.000	28.000	17.000	19.000	13.000
8	1.000	0.000	0.000	1.000	1.000	1.000	0.000	0.000	0.000
9	1.000	0.000	0.000	2.000	1.000	1.000	1.000	1.000	0.000

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



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



3.3 Univariate Analysis

In [312]:
```
## 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 / number of time i
t 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 occured in who
le data
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))

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

```
# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177, 0.13636363636363635, 0.25,
 0.19318181818181818, 0.03787878787878788, 0.03787878787878788, 0.03787878787878788],
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.27040816326530615, 0.0612
24489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408163265307, 0.056122448979591837],
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177, 0.06818181818181817
7, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.078787878787878782, 0.13
93939393939394, 0.34545454545454546, 0.060606060606060608, 0.060606060606060608, 0.060606060606060608],
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.46540880503144655, 0.075
471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081761006289, 0.062893081761006289],
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.072847682119205295, 0.0662
25165562913912, 0.066225165562913912, 0.27152317880794702, 0.066225165562913912, 0.066225165562913912],
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.073333333333333334, 0.093
333333333333338, 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 data
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there in the train data then we wi
ll add the feature to gv_fea
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_fea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#             gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea
```

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

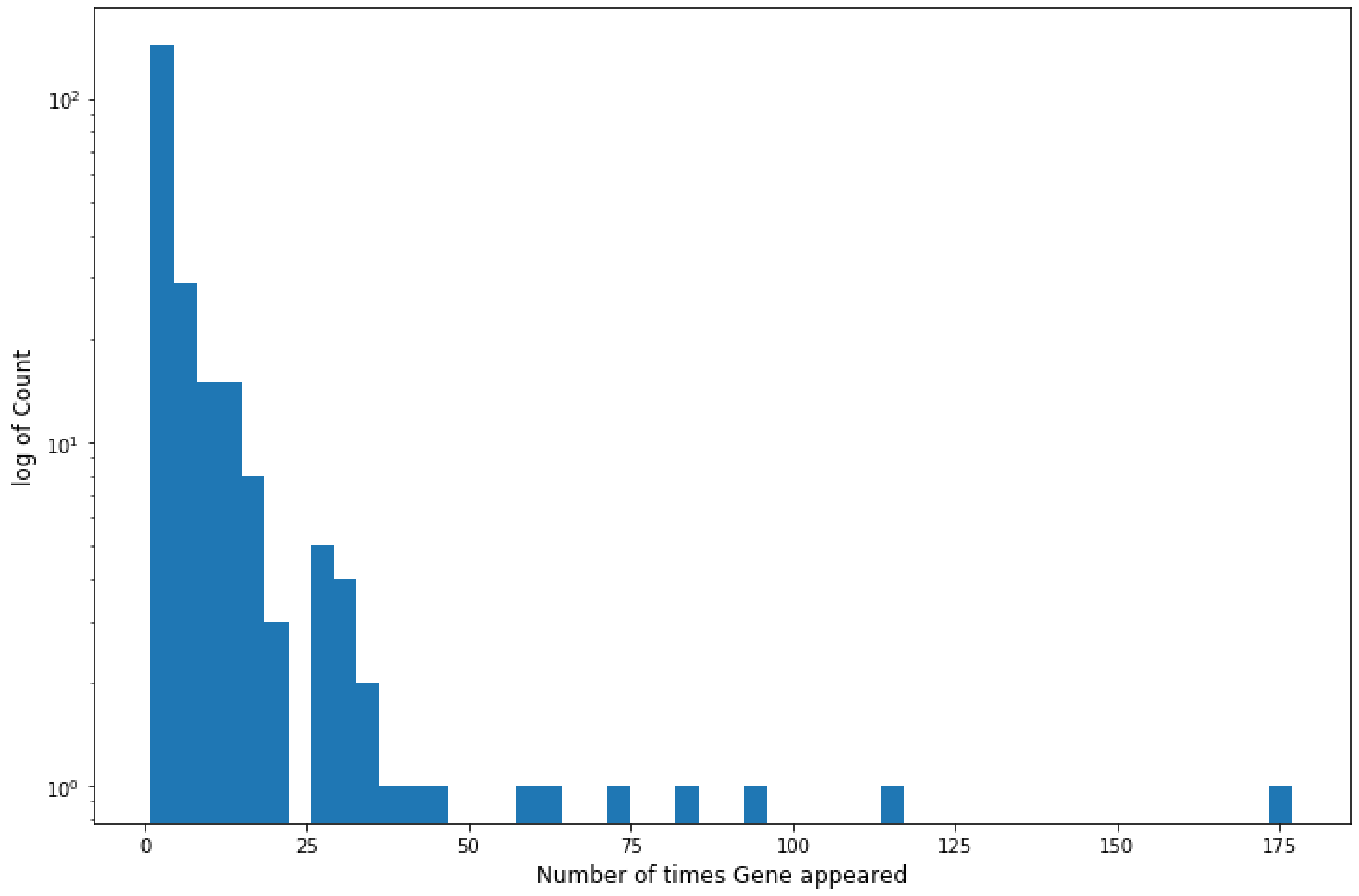
Q2. How many categories are in train,cv,test data and How they are distributed?

In [313]:
```
unique_genes_train = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes_train.shape[0])
# the top 10 genes that occured most
print(unique_genes_train.head(10))
```

```
Number of Unique Genes : 235
BRCA1     177
TP53      117
EGFR       94
PTEN       85
BRCA2      74
KIT        63
BRAF       59
ALK        44
ERBB2      41
PDGFRA     37
Name: Gene, dtype: int64
```

In [314]:
```
train_df_genes = train_df.groupby('Gene')['Gene'].count()

for i in [2, 5, 10, 20, 50, 100, 300, 500]:
    print('{}% of Genes appeared less than {} times'.format(round((train_df_genes < i).mean() * 100, 2),i))

plt.figure(figsize=(12, 8))
plt.hist(train_df_genes.values, bins=50, log=True)
plt.xlabel('Number of times Gene appeared', fontsize=12)
plt.ylabel('log of Count', fontsize=12)
plt.show()
```

```
31.06% of Genes appeared less than 2 times
61.28% of Genes appeared less than 5 times
76.17% of Genes appeared less than 10 times
90.21% of Genes appeared less than 20 times
97.02% of Genes appeared less than 50 times
99.15% of Genes appeared less than 100 times
100.0% of Genes appeared less than 300 times
100.0% of Genes appeared less than 500 times
```



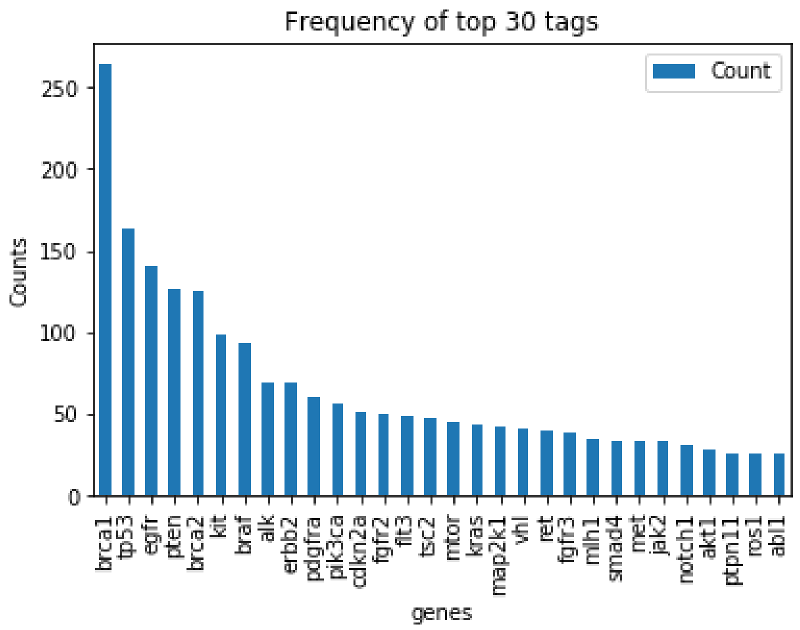
In [315]:
```
cnt_vectorizer = CountVectorizer()
gene_count = cnt_vectorizer.fit_transform(result["Gene"])

genes = cnt_vectorizer.get_feature_names()
freqs = gene_count.sum(axis=0).A1
gene_count = dict(zip(genes, freqs))

genes_count = pd.DataFrame.from_dict(gene_count,orient='index')
```

In [316]:
```
genes_count.columns = ["Count"]
genes_count = genes_count.sort_values("Count",ascending=False)
```

In [317]:
```
gene_values = genes_count.index
genes_count["genes"] = gene_values
```

```
In [318]: genes_count.head()

          i=np.arange(30)

          genes_count.head(30).plot(kind='bar')
          plt.title('Frequency of top 30 tags')
          plt.xticks(i, genes_count['genes'])
          plt.xlabel('genes')
          plt.ylabel('Counts')
```

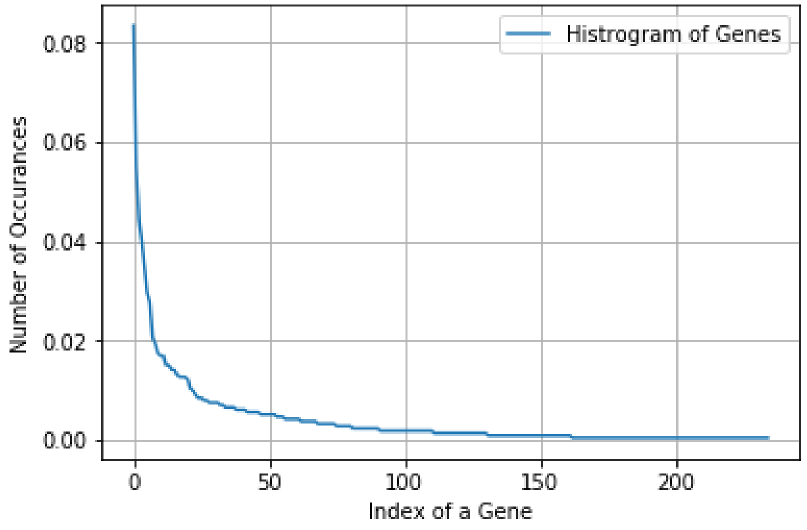
```
Out[318]: Text(0,0.5,'Counts')
```



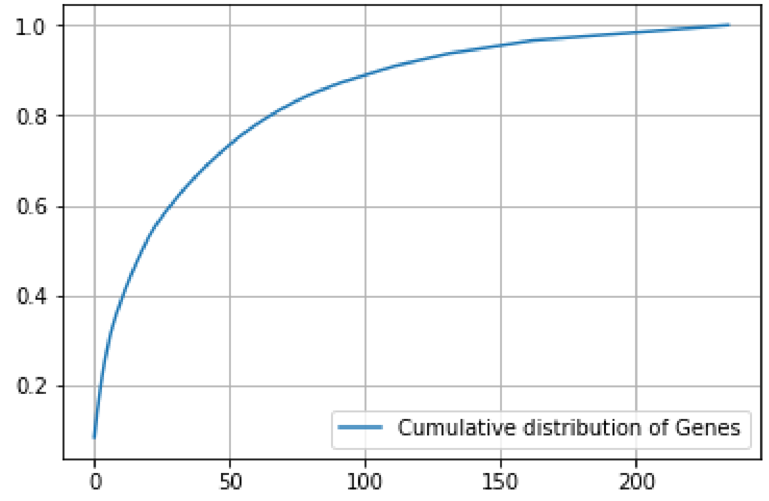
```
In [319]: print("Ans: There are", unique_genes_train.shape[0] ,"different categories of genes in the train data, and they
           are distibuted as follows",)
```

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

In [320]:
```
s = sum(unique_genes_train.values);
h = unique_genes_train.values/s;
plt.plot(h, label="Histrogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [321]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```

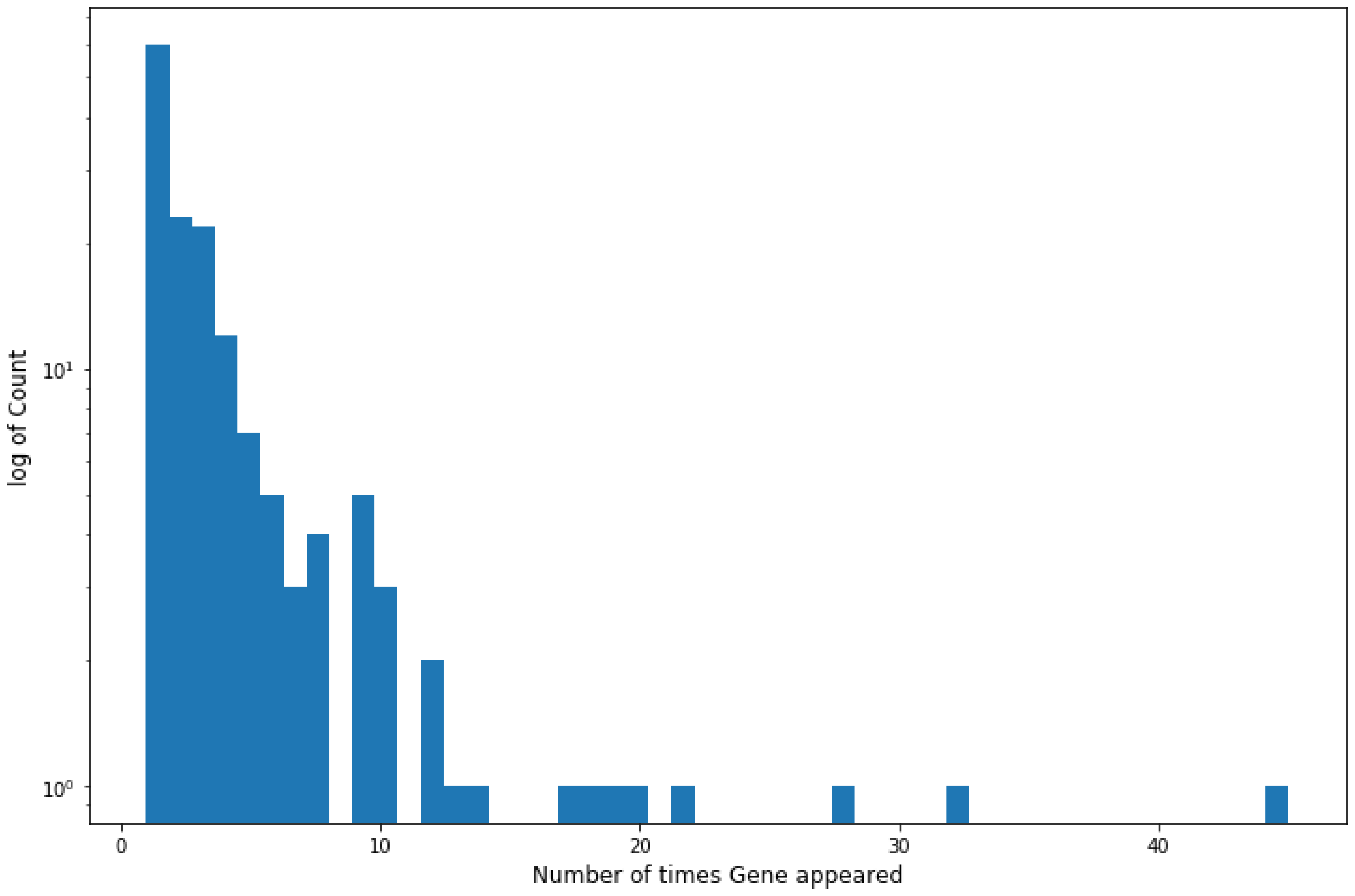


In [322]:
```
unique_genes_test = test_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes_test.shape[0])
# the top 10 genes that occured most
print(unique_genes_test.head(10))
```

```
Number of Unique Genes : 156
BRCA1     45
TP53      32
BRCA2     28
EGFR      22
BRAF      20
ALK       19
KIT       18
PTEN      17
ERBB2     14
CDKN2A    13
Name: Gene, dtype: int64
```

In [323]:
```
test_df_genes = test_df.groupby('Gene')['Gene'].count()

for i in [2, 5, 10, 20, 50, 100, 300, 500]:
    print('{}% of Genes appeared less than {} times'.format(round((test_df_genes < i).mean() * 100, 2),i))

plt.figure(figsize=(12, 8))
plt.hist(test_df_genes.values, bins=50, log=True)
plt.xlabel('Number of times Gene appeared', fontsize=12)
plt.ylabel('log of Count', fontsize=12)
plt.show()
```

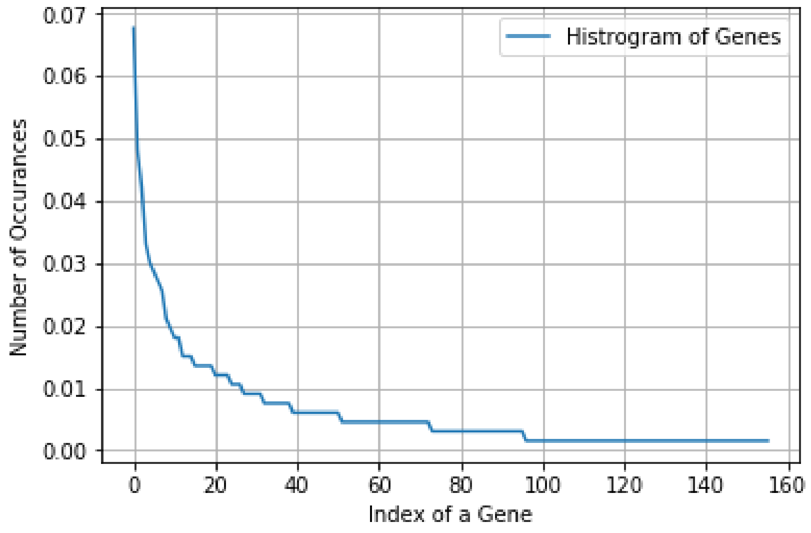
```
38.46% of Genes appeared less than 2 times
75.0% of Genes appeared less than 5 times
90.38% of Genes appeared less than 10 times
96.79% of Genes appeared less than 20 times
100.0% of Genes appeared less than 50 times
100.0% of Genes appeared less than 100 times
100.0% of Genes appeared less than 300 times
100.0% of Genes appeared less than 500 times
```



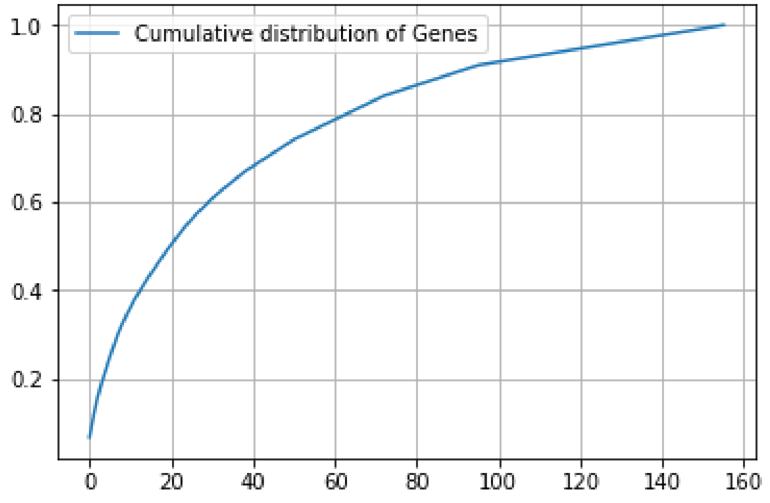
In [324]:
```
print("Ans: There are", unique_genes_test.shape[0] ,"different categories of genes in the test data, and they ar
e distibuted as follows",)
```

```
Ans: There are 156 different categories of genes in the test data, and they are distibuted as follows
```

In [325]:
```
s = sum(unique_genes_test.values);
h = unique_genes_test.values/s;
plt.plot(h, label="Histrogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [326]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```

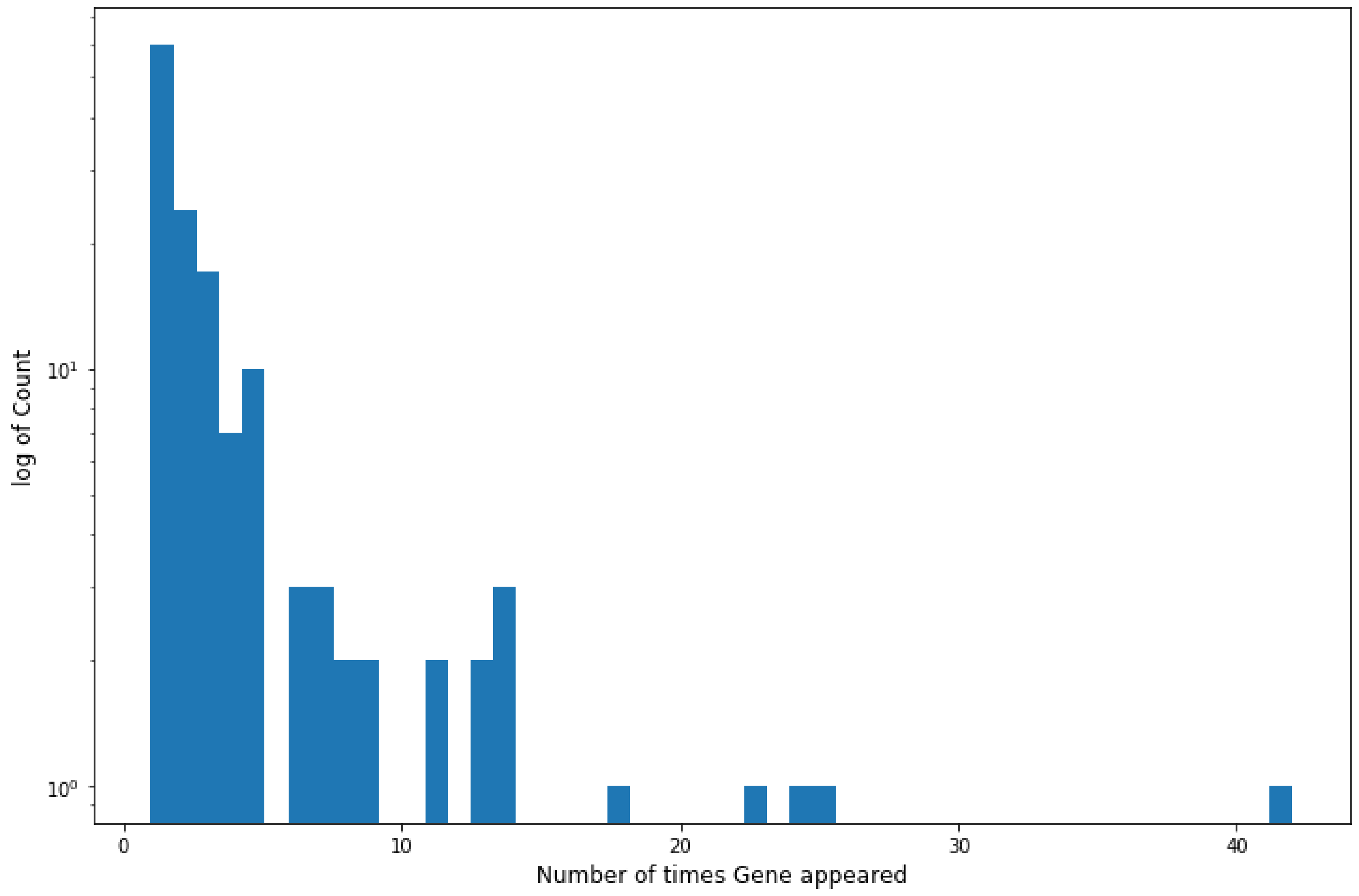


In [327]:
```
unique_genes_cv = cv_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes_cv.shape[0])
# the top 10 genes that occured most
print(unique_genes_cv.head(10))
```

```
Number of Unique Genes : 140
BRCA1     42
EGFR      25
PTEN      24
BRCA2     23
KIT       18
BRAF      14
ERBB2     14
TP53      14
PDGFRA    13
PIK3CA    13
Name: Gene, dtype: int64
```

In [328]:
```
cv_df_genes = cv_df.groupby('Gene')['Gene'].count()

for i in [2, 5, 10, 20, 50, 100, 300, 500]:
    print('{}% of Genes appeared less than {} times'.format(round((cv_df_genes < i).mean() * 100, 2),i))

plt.figure(figsize=(12, 8))
plt.hist(cv_df_genes.values, bins=50, log=True)
plt.xlabel('Number of times Gene appeared', fontsize=12)
plt.ylabel('log of Count', fontsize=12)
plt.show()
```

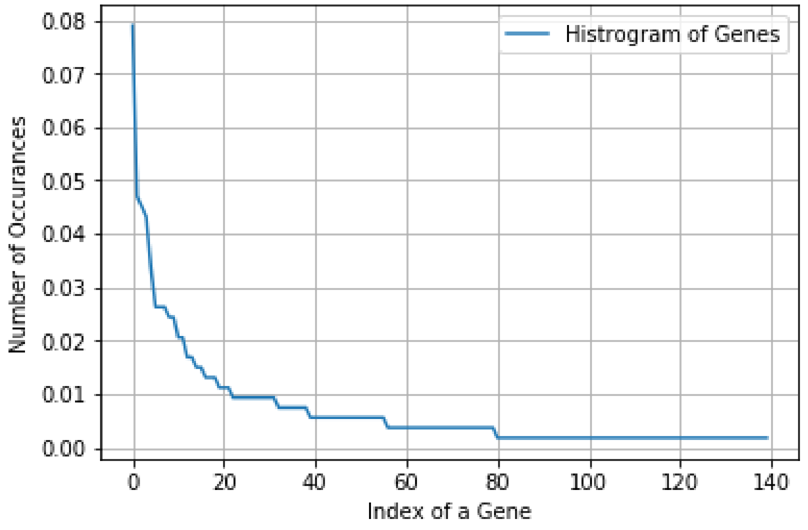
```
42.86% of Genes appeared less than 2 times
77.14% of Genes appeared less than 5 times
91.43% of Genes appeared less than 10 times
97.14% of Genes appeared less than 20 times
100.0% of Genes appeared less than 50 times
100.0% of Genes appeared less than 100 times
100.0% of Genes appeared less than 300 times
100.0% of Genes appeared less than 500 times
```



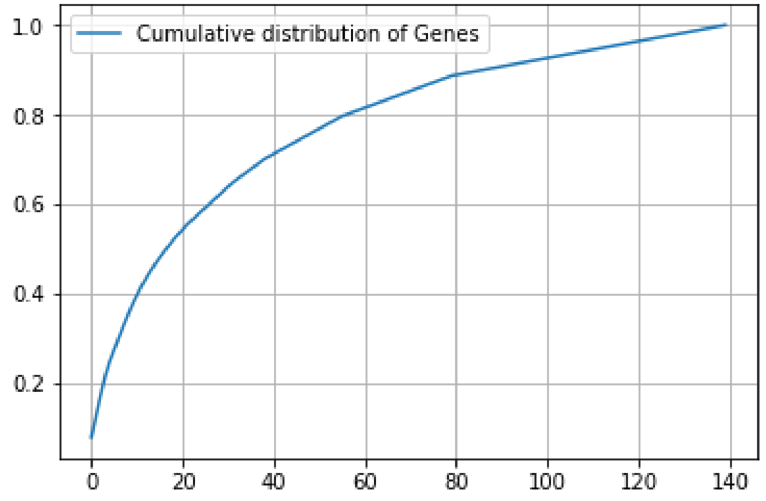
In [329]:
```
print("Ans: There are", unique_genes_cv.shape[0] ,"different categories of genes in the cv data, and they are di
stibuted as follows",)
```

```
Ans: There are 140 different categories of genes in the cv data, and they are distibuted as follows
```

In [330]:
```
s = sum(unique_genes_cv.values);
h = unique_genes_cv.values/s;
plt.plot(h, label="Histrogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [331]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

In [332]:
```
#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 [333]:
```
print("train_gene_feature_responseCoding is converted feature using respone coding method. The shape of gene fea
ture:", train_gene_feature_responseCoding.shape)
```

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

In [334]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [335]:
```
train_df['Gene'].head()
```

Out[335]:
```
1985    CTNNB1
452       TP53
3136      KRAS
24         CBL
1638     RRAS2
Name: Gene, dtype: object
```

In [336]:
```
gene_vectorizer.get_feature_names()
```

```
Out[336]: ['abl1',
            'acvr1',
            'ago2',
            'akt1',
            'akt2',
            'akt3',
            'alk',
            'apc',
            'ar',
            'araf',
            'arid1a',
            'arid1b',
            'arid2',
            'arid5b',
            'asxl1',
            'atm',
            'atr',
            'atrx',
            'aurka',
            'aurkb',
            'axin1',
            'axl',
            'b2m',
            'bap1',
            'bard1',
            'bcl10',
            'bcl2',
            'bcl2l11',
            'bcor',
            'braf',
            'brca1',
            'brca2',
            'brd4',
            'brip1',
            'btk',
            'card11',
            'carm1',
            'casp8',
            'cbl',
            'ccnd1',
            'ccnd2',
            'ccnd3',
            'ccne1',
            'cdh1',
            'cdk12',
            'cdk4',
            'cdk6',
            'cdk8',
            'cdkn1a',
            'cdkn1b',
            'cdkn2a',
            'cdkn2b',
            'cebpa',
            'chek2',
            'cic',
            'crebbp',
            'ctcf',
            'ctnnb1',
            'ddr2',
            'dicer1',
            'dnmt3a',
            'dusp4',
            'egfr',
            'eif1ax',
            'elf3',
            'ep300',
            'epas1',
            'epcam',
            'erbb2',
            'erbb3',
            'erbb4',
            'ercc2',
            'ercc3',
            'ercc4',
            'erg',
            'errfi1',
            'esr1',
            'etv1',
            'etv6',
            'ewsr1',
            'ezh2',
            'fanca',
            'fancc',
            'fat1',
            'fbxw7',
            'fgf19',
            'fgfr1',
            'fgfr2',
            'fgfr3',
            'fgfr4',
            'flt1',
            'flt3',
            'foxa1',
            'foxl2',
            'foxp1',
            'fubp1',
            'gata3',
            'gli1',
            'gna11',
            'gnaq',
            'gnas',
            'h3f3a',
            'hla',
            'hnf1a',
            'hras',
            'idh1',
            'idh2',
            'igf1r',
            'ikbke',
            'il7r',
            'inpp4b',
            'jak1',
            'jak2',
            'jun',
            'kdm5c',
            'kdm6a',
            'kdr',
            'keap1',
            'kit',
            'klf4',
            'kmt2a',
            'kmt2b',
            'kmt2c',
            'kmt2d',
            'knstrn',
            'kras',
            'lats1',
            'lats2',
            'map2k1',
            'map2k2',
            'map2k4',
            'map3k1',
            'mapk1',
            'mdm2',
            'mdm4',
            'med12',
            'mef2b',
            'men1',
            'met',
            'mga',
            'mlh1',
            'mpl',
            'msh2',
            'msh6',
            'mtor',
            'myc',
            'mycn',
            'myd88',
            'ncor1',
            'nf1',
            'nf2',
            'nfe2l2',
            'nfkbia',
            'nkx2',
            'notch1',
            'notch2',
            'npm1',
            'nras',
            'nsd1',
            'ntrk1',
            'ntrk2',
            'ntrk3',
            'nup93',
            'pak1',
            'pbrm1',
            'pdgfra',
            'pdgfrb',
            'pik3ca',
            'pik3cb',
            'pik3cd',
            'pik3r1',
            'pik3r2',
            'pim1',
            'pms1',
            'pms2',
            'pole',
            'ppp2r1a',
            'ppp6c',
            'prdm1',
            'pten',
            'ptpn11',
            'ptprd',
            'ptprt',
            'rac1',
            'rad21',
            'rad50',
            'rad51b',
            'rad51c',
            'raf1',
            'rasa1',
            'rb1',
            'rbm10',
            'ret',
            'rheb',
            'rhoa',
            'rictor',
            'rit1',
            'ros1',
            'rras2',
            'runx1',
            'rxra',
            'sdhb',
            'sdhc',
            'setd2',
            'sf3b1',
            'shoc2',
            'shq1',
            'smad2',
            'smad3',
            'smad4',
            'smarca4',
            'smarcb1',
            'smo',
            'sos1',
            'sox9',
            'spop',
            'src',
            'srsf2',
            'stat3',
            'stk11',
            'tert',
            'tet2',
            'tgfbr1',
            'tgfbr2',
            'tmprss2',
            'tp53',
            'tp53bp1',
            'tsc1',
            'tsc2',
            'u2af1',
            'vhl',
            'whsc1',
            'whsc1l1',
            'xrcc2',
            'yap1']
```

In [337]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene fea
ture:", train_gene_feature_onehotCoding.shape)
```

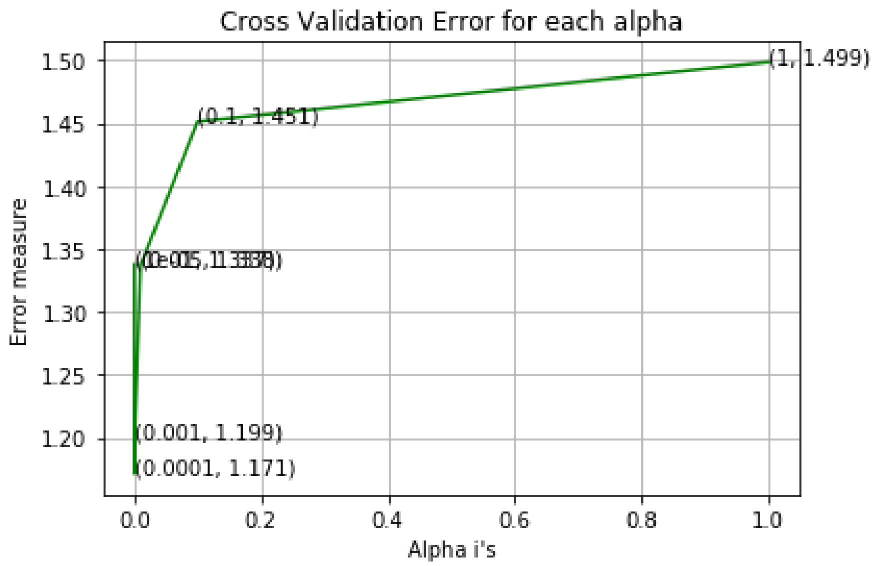
```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature:
(2124, 235)
```

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

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

In [338]:
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

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

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

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



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.classes_, eps=1e-
15))

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.3376102865548798
For values of alpha =  0.0001 The log loss is: 1.1712085707572386
For values of alpha =  0.001 The log loss is: 1.1993264741882734
For values of alpha =  0.01 The log loss is: 1.33665518099885
For values of alpha =  0.1 The log loss is: 1.4513096373718724
For values of alpha =  1 The log loss is: 1.498553684719974
```



```
For values of best alpha =  0.0001 The train log loss is: 1.063736765436026
For values of best alpha =  0.0001 The cross validation log loss is: 1.1712085707572386
For values of best alpha =  0.0001 The test log loss is: 1.2149213371407512
```

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

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

In [339]:
```
print("Q6. How many data points in Test and CV datasets are covered by the ", unique_genes.shape[0], " genes in
 train dataset?")

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  242  genes in train dataset?
Ans
1. In test data 644 out of 665 : 96.84210526315789
2. In cross validation data 518 out of  532 : 97.36842105263158
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

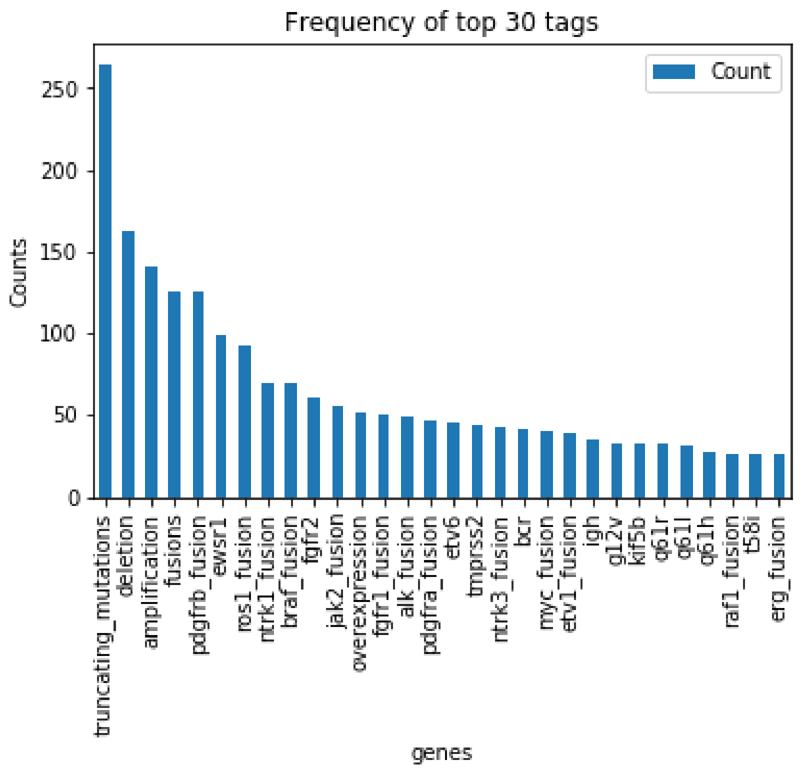
Ans. Variation is a categorical variable

Q8. How many categories are there?

In [340]:
```
cnt_vectorizer = CountVectorizer()
variation_count = cnt_vectorizer.fit_transform(result["Variation"])

variations = cnt_vectorizer.get_feature_names()
freqs = variation_count.sum(axis=0).A1
variations_count = dict(zip(variations, freqs))
```

In [341]:
```
variations_count = pd.DataFrame.from_dict(variations_count,orient='index')

variations_count.columns = ["Count"]
variations_count = variations_count.sort_values("Count",ascending=False)

variation_values = variations_count.index
variations_count["variation"] = variation_values
```

```
In [342]: i=np.arange(30)

          genes_count.head(30).plot(kind='bar')
          plt.title('Frequency of top 30 tags')
          plt.xticks(i, variations_count['variation'])
          plt.xlabel('genes')
          plt.ylabel('Counts')

Out[342]: Text(0,0.5,'Counts')
```

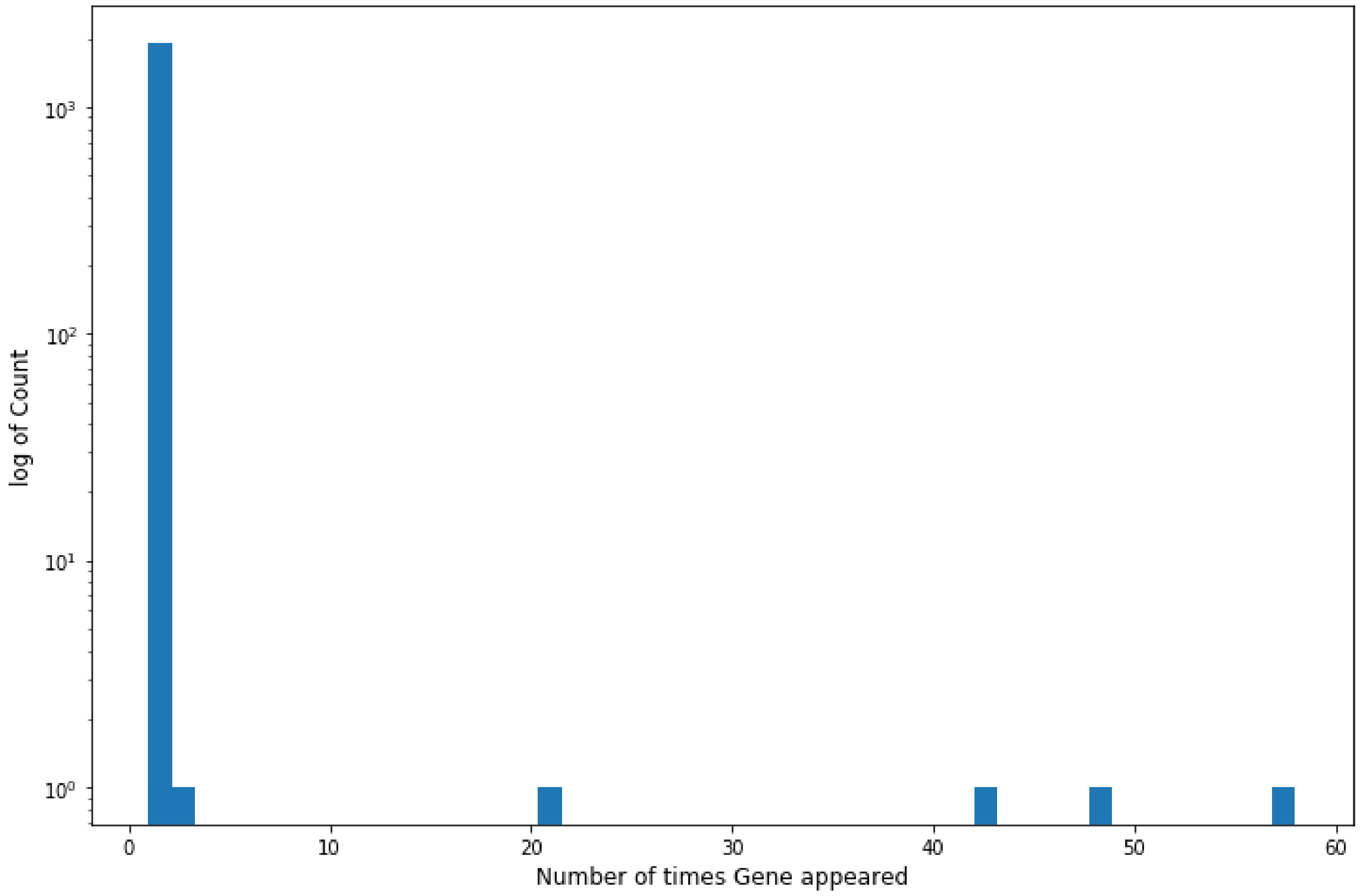


In [343]:
```
unique_train_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_train_variations.shape[0])
# the top 10 variations that occured most
print(unique_train_variations.head(10))
```

```
Number of Unique Variations : 1931
Truncating_Mutations    58
Deletion                48
Amplification           43
Fusions                 21
Q61L                     3
E17K                     2
E330K                    2
Overexpression           2
TMPRSS2-ETV1_Fusion      2
Q22K                     2
Name: Variation, dtype: int64
```

In [344]:
```
train_df_variation = train_df.groupby('Variation')['Variation'].count()

for i in [2, 5, 10, 20, 50, 100, 300, 500]:
    print('{}% of Genes appeared less than {} times'.format(round((train_df_variation < i).mean() * 100, 2),i))

plt.figure(figsize=(12, 8))
plt.hist(train_df_variation.values, bins=50, log=True)
plt.xlabel('Number of times Gene appeared', fontsize=12)
plt.ylabel('log of Count', fontsize=12)
plt.show()
```

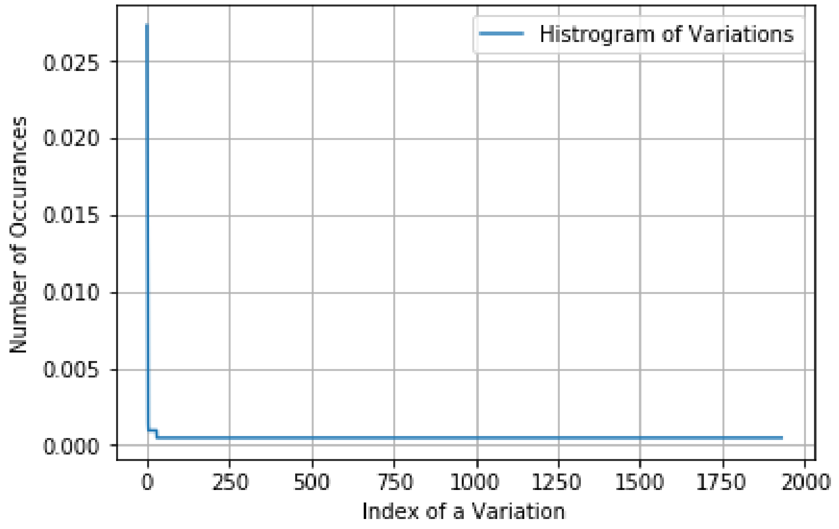
```
98.45% of Genes appeared less than 2 times
99.79% of Genes appeared less than 5 times
99.79% of Genes appeared less than 10 times
99.79% of Genes appeared less than 20 times
99.95% of Genes appeared less than 50 times
100.0% of Genes appeared less than 100 times
100.0% of Genes appeared less than 300 times
100.0% of Genes appeared less than 500 times
```



In [345]:
```
print("Ans: There are", unique_train_variations.shape[0] ,"different categories of variations in the train data, 
and they are distibuted as follows",)
```

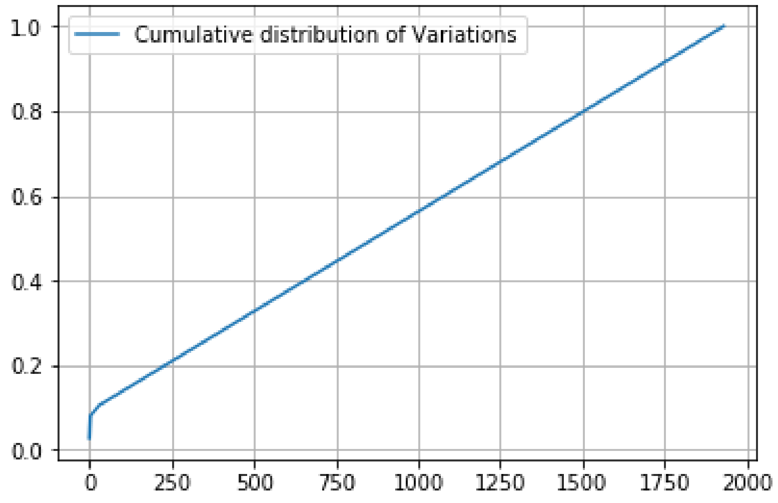
```
Ans: There are 1931 different categories of variations in the train data, and they are distibuted as follows
```

In [346]:
```
s = sum(unique_train_variations.values);
h = unique_train_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [347]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.02730697 0.04990584 0.07015066 ... 0.99905838 0.99952919 1.        ]
```

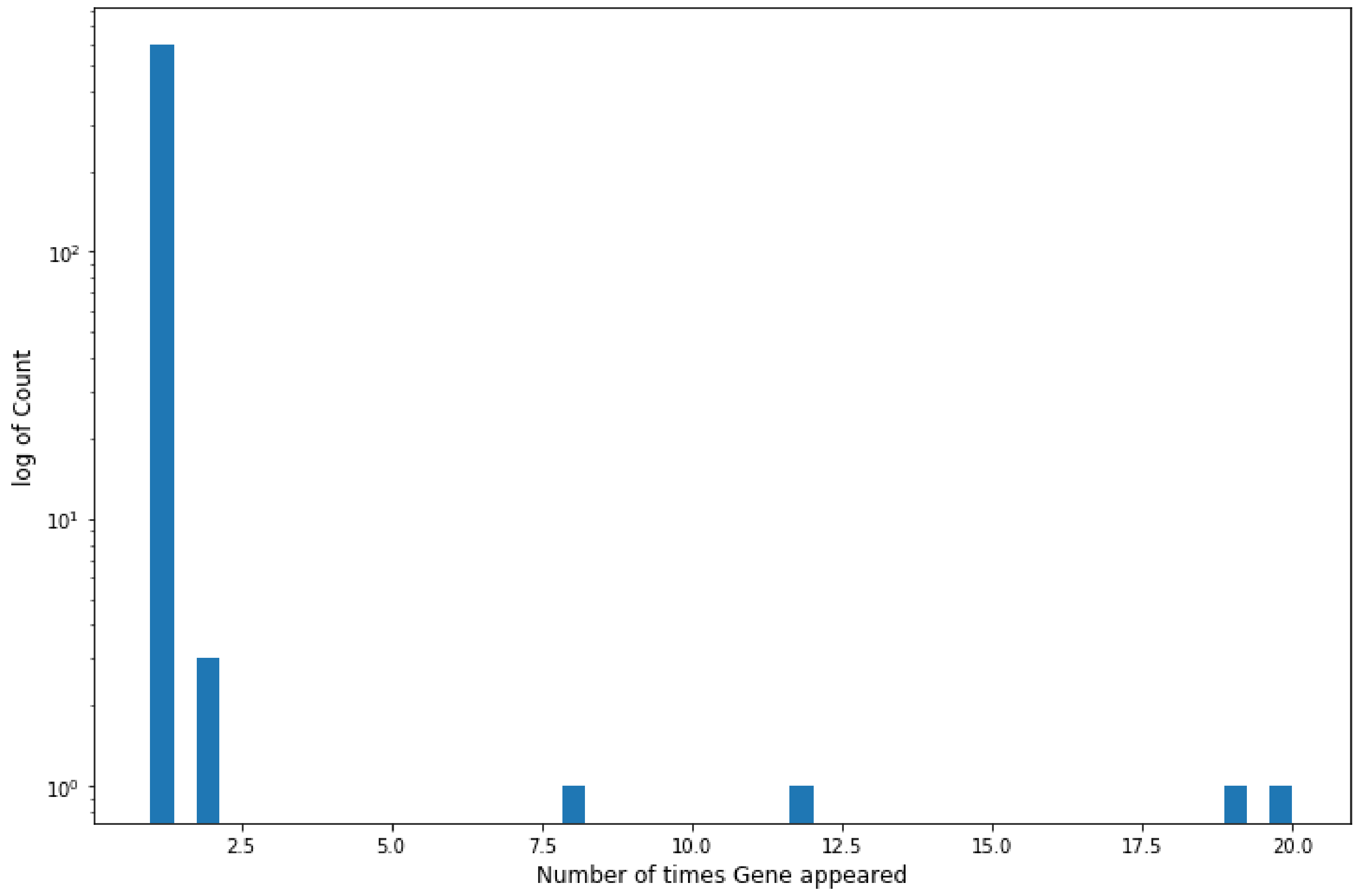


In [348]:
```
unique_test_variations = test_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_test_variations.shape[0])
# the top 10 variations that occured most
print(unique_test_variations.head(10))
```

```
Number of Unique Variations : 607
Truncating_Mutations    20
Amplification           19
Deletion                12
Fusions                  8
G35R                     2
G12V                     2
Q61H                     2
K539L                    1
R886W                    1
I89T                     1
Name: Variation, dtype: int64
```

In [349]:
```
test_df_variation = test_df.groupby('Variation')['Variation'].count()

for i in [2, 5, 10, 20, 50, 100, 300, 500]:
    print('{}% of Genes appeared less than {} times'.format(round((test_df_variation < i).mean() * 100, 2),i))

plt.figure(figsize=(12, 8))
plt.hist(test_df_variation.values, bins=50, log=True)
plt.xlabel('Number of times Gene appeared', fontsize=12)
plt.ylabel('log of Count', fontsize=12)
plt.show()
```

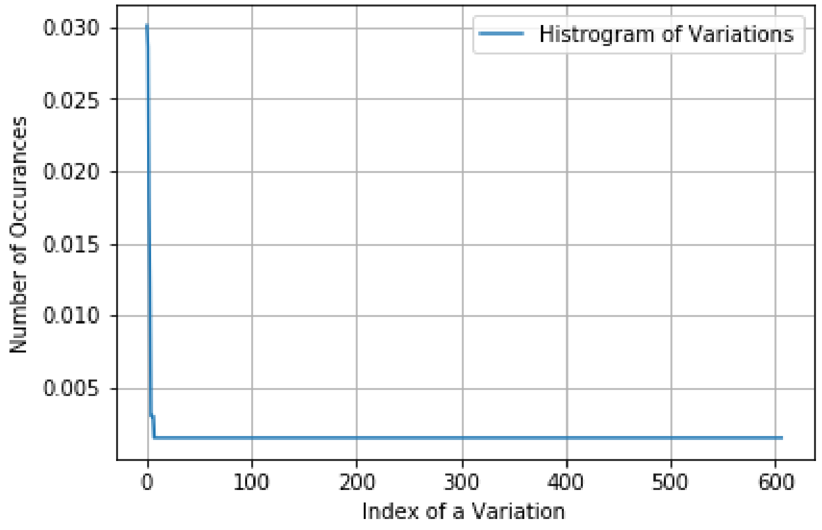
```
98.85% of Genes appeared less than 2 times
99.34% of Genes appeared less than 5 times
99.51% of Genes appeared less than 10 times
99.84% of Genes appeared less than 20 times
100.0% of Genes appeared less than 50 times
100.0% of Genes appeared less than 100 times
100.0% of Genes appeared less than 300 times
100.0% of Genes appeared less than 500 times
```



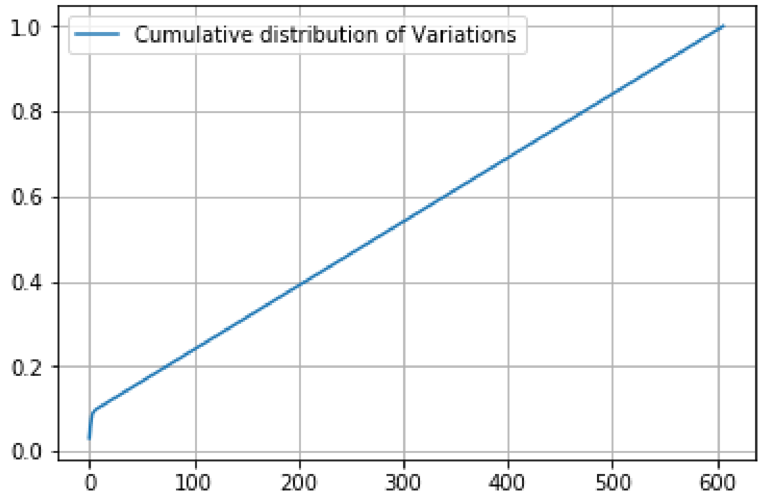
In [350]:
```
print("Ans: There are", unique_test_variations.shape[0] ,"different categories of variations in the test data, a
nd they are distibuted as follows",)
```

```
Ans: There are 607 different categories of variations in the test data, and they are distibuted as follows
```

In [351]:
```
s = sum(unique_test_variations.values);
h = unique_test_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [352]:
```
c = np.cumsum(h)
#print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

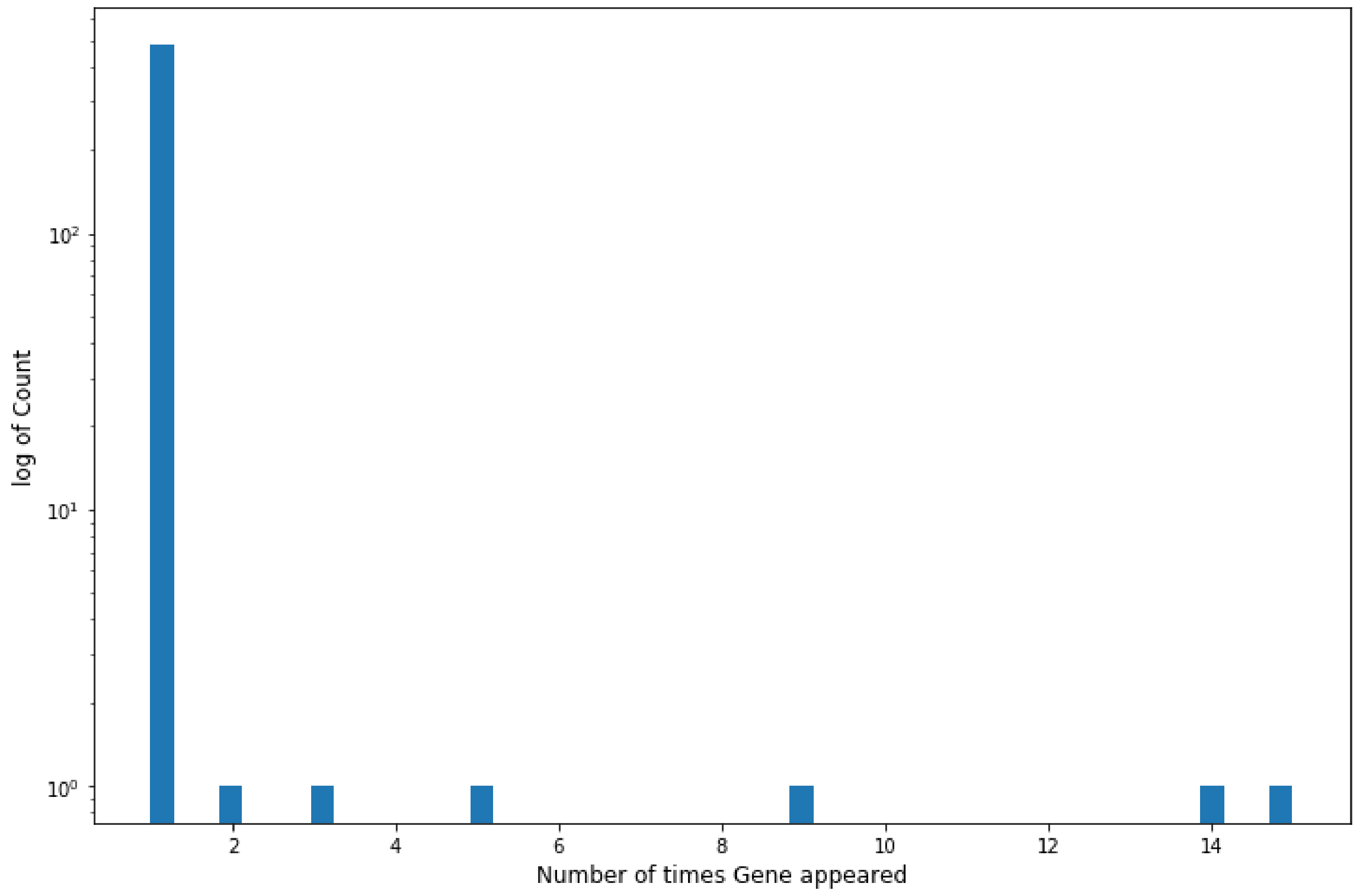


In [353]:
```
unique_cv_variations = cv_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_cv_variations.shape[0])
# the top 10 variations that occured most
print(unique_cv_variations.head(10))
```

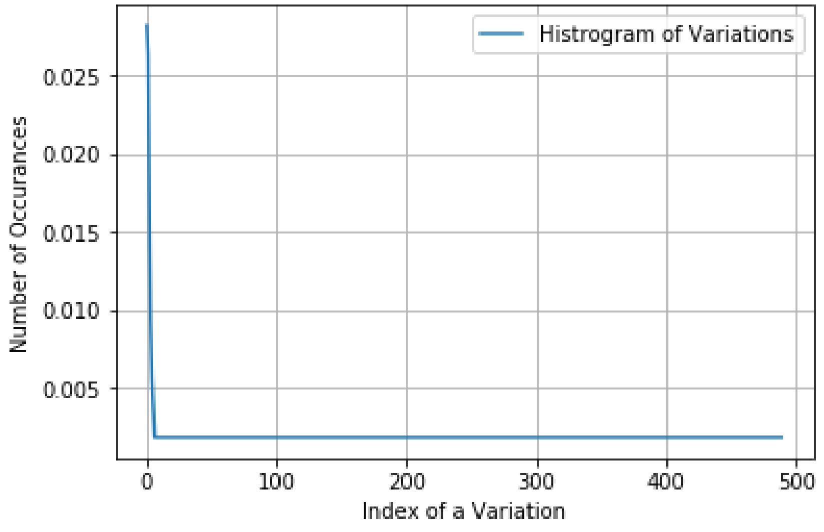
```
Number of Unique Variations : 490
Truncating_Mutations    15
Deletion                14
Amplification            9
Fusions                  5
Overexpression           3
G67R                     2
Q22R                     1
H191D                    1
L747F                    1
V84L                     1
Name: Variation, dtype: int64
```

In [354]:
```
cv_df_variation = cv_df.groupby('Variation')['Variation'].count()

for i in [2, 5, 10, 20, 50, 100, 300, 500]:
    print('{}% of Genes appeared less than {} times'.format(round((cv_df_variation < i).mean() * 100, 2),i))

plt.figure(figsize=(12, 8))
plt.hist(cv_df_variation.values, bins=50, log=True)
plt.xlabel('Number of times Gene appeared', fontsize=12)
plt.ylabel('log of Count', fontsize=12)
plt.show()
```

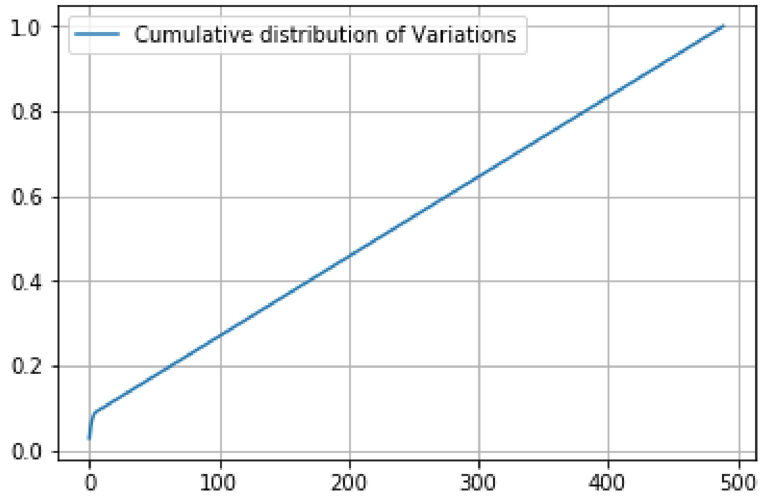
```
98.78% of Genes appeared less than 2 times
99.18% of Genes appeared less than 5 times
99.59% of Genes appeared less than 10 times
100.0% of Genes appeared less than 20 times
100.0% of Genes appeared less than 50 times
100.0% of Genes appeared less than 100 times
100.0% of Genes appeared less than 300 times
100.0% of Genes appeared less than 500 times
```



In [355]:
```
s = sum(unique_cv_variations.values);
h = unique_cv_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [356]:
```
c = np.cumsum(h)
#print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [357]:
```
# 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 [358]:
```
print("train_variation_feature_responseCoding is a converted feature using the response coding method. The shape
of Variation feature:", train_variation_feature_responseCoding.shape)
```

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

In [359]:
```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [360]:
```
print("train_variation_feature_onehotEncoded is converted feature using the one-hot encoding method. The shape o
f Variation feature:", train_variation_feature_onehotCoding.shape)
```

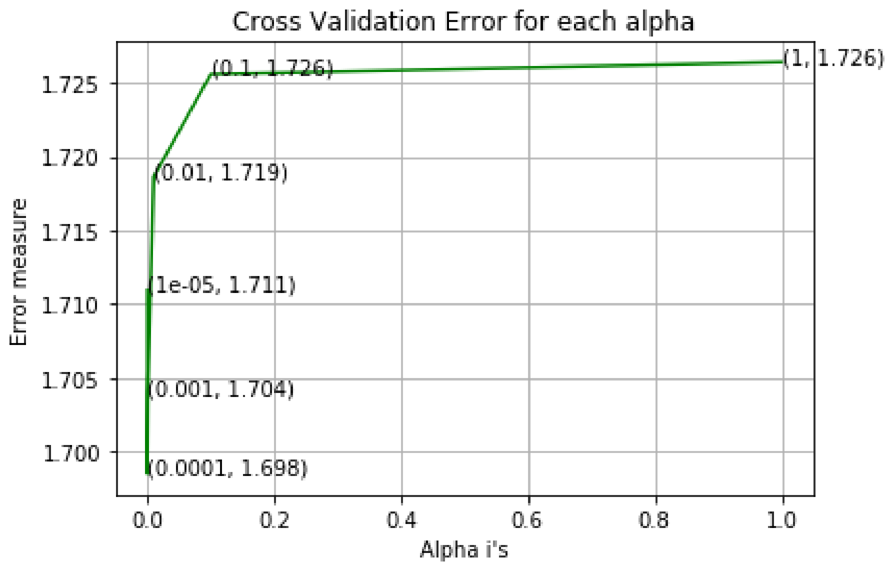
```
train_variation_feature_onehotEncoded is converted feature using the one-hot encoding method. The shape of Varia
tion feature: (2124, 1962)
```

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

Let's build a model just like the earlier!

In [361]:
```
alpha = [10 ** x for x in range(-5, 1)]

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

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

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



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.classes_, eps=1e-
15))

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.7109317239336137
For values of alpha =  0.0001 The log loss is: 1.69847036419656
For values of alpha =  0.001 The log loss is: 1.703932621558029
For values of alpha =  0.01 The log loss is: 1.7186039939517292
For values of alpha =  0.1 The log loss is: 1.7255738015730941
For values of alpha =  1 The log loss is: 1.7263942907206098
```



```
For values of best alpha =  0.0001 The train log loss is: 0.7820687239476903
For values of best alpha =  0.0001 The cross validation log loss is: 1.69847036419656
For values of best alpha =  0.0001 The test log loss is: 1.6829173294289694
```

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

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

In [362]:
```
print("Q12. How many data points are covered by total ", unique_train_variations.shape[0], " genes in test and c
ross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.shape[0])*100)
```

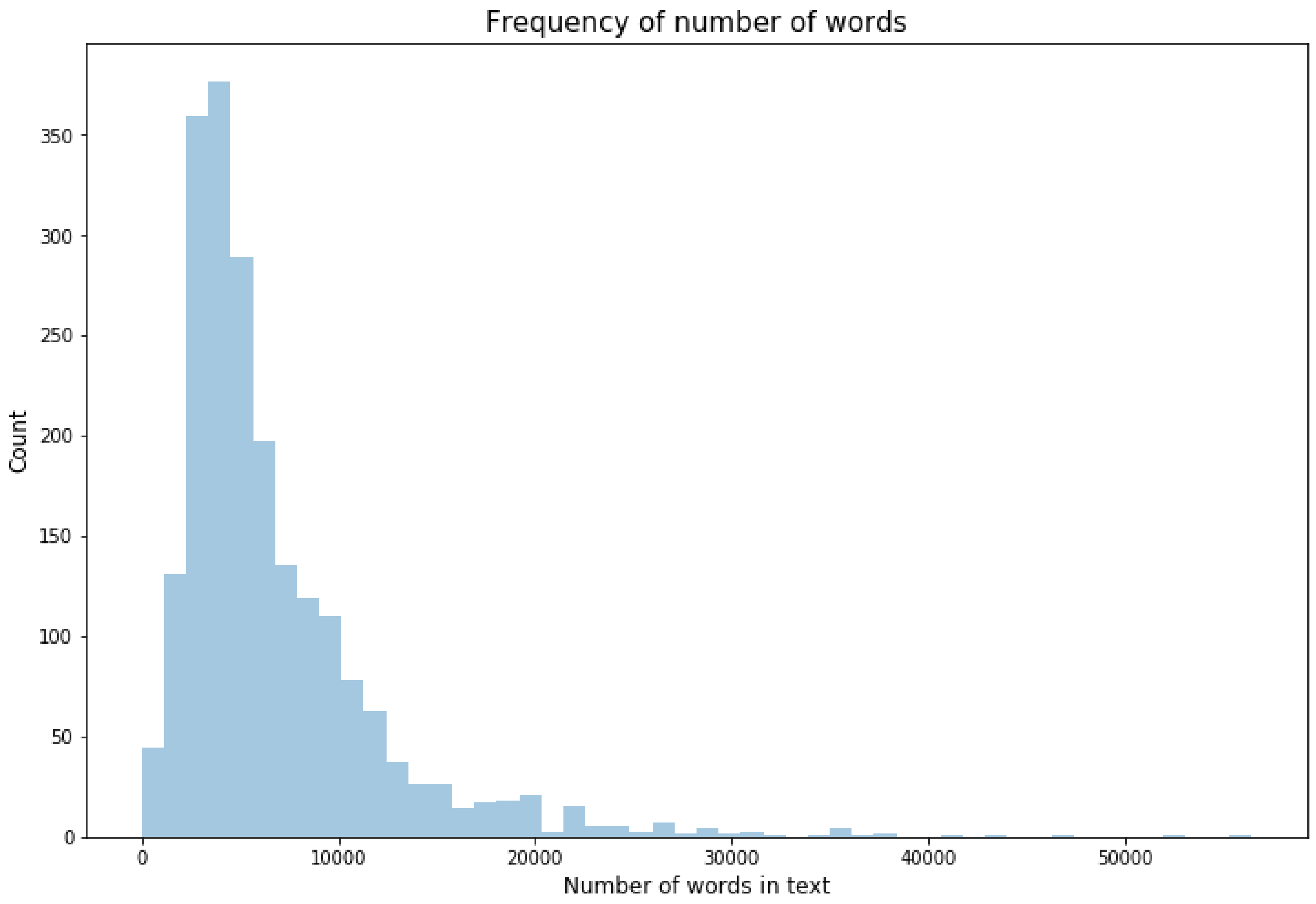
```
Q12. How many data points are covered by total  1931  genes in test and cross validation data sets?
Ans
1. In test data 76 out of 665 : 11.428571428571429
2. In cross validation data 52 out of  532 : 9.774436090225564
```

3.2.3 Univariate Analysis on Text Feature

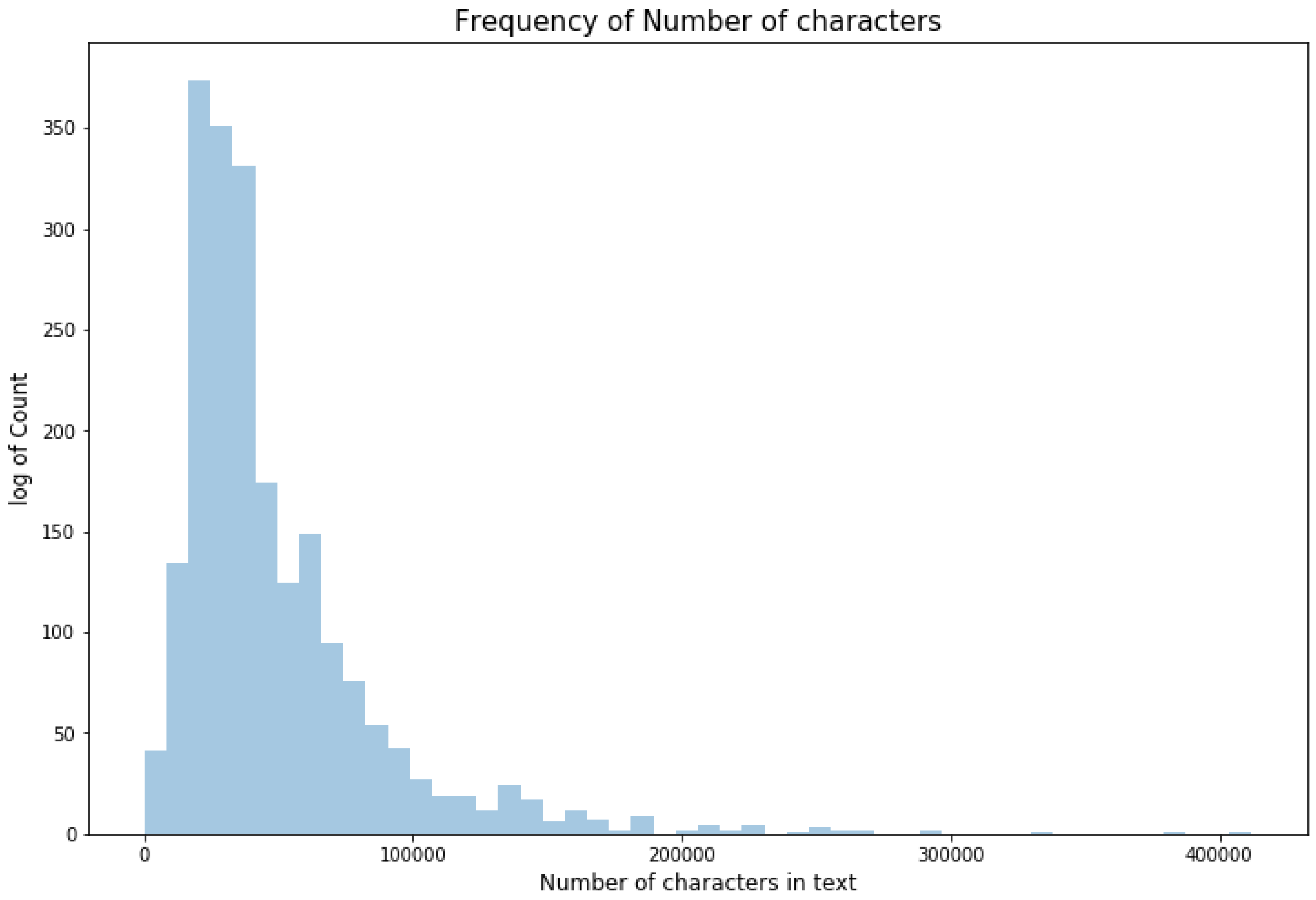
1. How many unique words are present in train data?
2. How are word frequencies distributed?
3. How to featurize text field?
4. Is the text feature useful in predicitng y_i?
5. Is the text feature stable across train, test and CV datasets?

In [363]:
```
no_of_words = train_df["TEXT"].apply(lambda x: len(str(x).split()) )
text_len = train_df["TEXT"].apply(lambda x: len(str(x)) )
```

In [364]:
```
plt.figure(figsize=(12, 8))
sns.distplot(no_of_words, bins=50, kde=False)
plt.xlabel('Number of words in text', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title("Frequency of number of words", fontsize=15)
plt.show()
```



In [365]:
```
plt.figure(figsize=(12, 8))
sns.distplot(text_len, bins=50, kde=False)
plt.xlabel('Number of characters in text', fontsize=12)
plt.ylabel('log of Count', fontsize=12)
plt.title("Frequency of Number of characters", fontsize=15)
plt.show()
```



In [366]:
```
# cls_text is a data frame
# for every row in data fram consider the 'TEXT'
# split the words by space
# make a dict with those words
# increment its count whenever we see that word

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

In [367]:
```
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 [368]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = TfidfVectorizer(min_df=3,max_features=1000)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
#tfeatures= text_vectorizer.get_feature_names()

words = text_vectorizer.get_feature_names()
# Displaying top 5 features
tfidf_mean = np.mean(train_text_feature_onehotCoding, axis = 0)
tfidf_mean = np.array(tfidf_mean)[0].tolist()

topn_ids = np.argsort(tfidf_mean)[::-1][:1000]
top_feats = []
for i in topn_ids:
    top_feats.append([words[i], tfidf_mean[i]])
train_text_features = pd.DataFrame(top_feats)
train_text_features.columns = ['feature', 'tfidf_score']
train_text_features = train_text_features['feature'].tolist()

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

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


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

```
Total number of unique words in train data : 1000
```

In [369]:
```
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 [370]:
```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
test_text_feature_responseCoding  = get_text_responsecoding(test_df)
cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)
```

In [371]:
```
# 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(a
xis=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 [372]:
```
# 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 [373]:
```
#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 [374]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({250.72763098357632: 1, 179.5255008265037: 1, 138.45989779510006: 1, 129.41618730704417: 1, 129.05550141
434912: 1, 115.39492908559812: 1, 115.24058076312042: 1, 115.06499053035995: 1, 110.3158004860868 9: 1, 106.34411
519054224: 1, 106.00437366314769: 1, 96.85341775028583: 1, 90.40663982505758: 1, 89.44853323905289: 1, 86.307714
66636787: 1, 82.71613478512045: 1, 81.81202303717974: 1, 80.50889849029772: 1, 77.86795676040984: 1, 77.55528096
500218: 1, 76.43525894340348: 1, 74.00886290474652: 1, 71.7724947189641: 1, 68.43345972809843: 1, 66.27579886194
172: 1, 66.1825602643916: 1, 65.83137040442911: 1, 65.60397977495369: 1, 65.47312685649042: 1, 63.2610513944409:
1, 63.038057759994075: 1, 62.46628376897927: 1, 62.14819855404199: 1, 62.072515596145: 1, 58.34848820451745: 1,
57.704770734094: 1, 56.00836636745287: 1, 55.726833825708674: 1, 53.40661302310928: 1, 51.880120659549874: 1, 5
0.850385516238696: 1, 50.50827463885715: 1, 49.164388137392166: 1, 48.69466030439487: 1, 48.018086809344375: 1,
47.53725461251578: 1, 45.36310895233383: 1, 45.29014222124111: 1, 45.06863024168037: 1, 44.03488632212769: 1, 4
3.893932505782786: 1, 43.42573583442284: 1, 43.04271979899348: 1, 42.555861230628686: 1, 42.46657852300239: 1, 4
2.45430329729527: 1, 42.3619705613652: 1, 42.221542225718004: 1, 41.01668038661078: 1, 40.9176152626886 8: 1, 40.
393729889945746: 1, 40.107003180863714: 1, 40.0957737068394: 1, 39.94416269368405: 1, 39.56298342261544: 1, 39.1
72813434182814: 1, 39.10235935377768: 1, 38.75625945283307: 1, 38.73163621191552: 1, 38.43158611309212: 1, 37.71
372186077893: 1, 37.57419626658479: 1, 37.55744314345285: 1, 37.27445219330268: 1, 36.96642621951516: 1, 36.9471
8207681225: 1, 36.63196558429721: 1, 36.56104085061595: 1, 35.89900687876188: 1, 35.50673403154524: 1, 35.304894
691286535: 1, 35.2212410344381: 1, 35.16782399342116: 1, 35.15962923087714: 1, 34.89948414334318: 1, 34.29906400
282229: 1, 34.19332828576665: 1, 33.85110693273544: 1, 33.37528566931936: 1, 33.10817521490745: 1, 32.8809138718
54744: 1, 32.53344759462224: 1, 32.35353742888153: 1, 32.260276831694526: 1, 31.980181565499667: 1, 31.969458362
192455: 1, 31.91380246463338 5: 1, 31.86635503269549: 1, 31.85946545843029: 1, 31.78463252046494: 1, 31.771319183
01086: 1, 31.40801600080692: 1, 31.334695482092904: 1, 31.171356064672725: 1, 31.166959055912567: 1, 30.93011824
552321: 1, 30.82517003435556: 1, 30.73437282329362: 1, 30.670282670983802: 1, 30.64808436008649: 1, 30.576776432
705234: 1, 30.279579144721737: 1, 30.24734926708067: 1, 30.193745687842465: 1, 30.091928179584137: 1, 30.0599896
4194817: 1, 30.056484004975236: 1, 29.756615335640593: 1, 29.697835057384474: 1, 29.513696787708742: 1, 29.51347
7835095575: 1, 29.493883850029018: 1, 29.476161700695314: 1, 29.420101685074773: 1, 29.129282276626647: 1, 29.00
60859656436: 1, 28.875587542760513: 1, 28.655123073801086: 1, 28.53646517921174: 1, 28.326976873319957: 1, 28.08
783765952294: 1, 27.987730394956746: 1, 27.862160169875214: 1, 27.798122413903847: 1, 27.584717109284817: 1, 27.
563065663042508: 1, 27.552289828550037: 1, 27.30136375858492: 1, 26.893668389018686: 1, 26.858464809290066: 1, 2
6.774222242062176: 1, 26.73035159232457: 1, 26.62853870752232: 1, 26.31580046750259: 1, 26.291405745969122: 1, 2
6.182948022361014: 1, 26.14262616241894: 1, 26.132155490861972: 1, 25.94609469592265: 1, 25.575903905757: 1, 25.
423351223897836: 1, 25.36469441895981: 1, 25.34138880498353: 1, 25.3230297831802: 1, 25.299247693199536: 1, 25.2
4670984834756: 1, 25.22830015313863: 1, 24.852632718158926: 1, 24.798239092881687: 1, 24.7490725290887 5: 1, 24.7
4163966718364: 1, 24.6172705031042: 1, 24.60610904587499: 1, 24.370643056249943: 1, 24.286658901855898: 1, 24.20
7877361009317: 1, 24.11021125017693: 1, 23.898554098735282: 1, 23.872439588829476: 1, 23.828794606484717: 1, 23.
80038056011981: 1, 23.79503273977186: 1, 23.65067046891399: 1, 23.62672350153865: 1, 23.49207170035395: 1, 23.48
724193124555: 1, 23.45828447717811: 1, 23.435283536073356: 1, 23.421356997404825: 1, 23.23912168261072: 1, 23.22
336907969196: 1, 22.979144399869593: 1, 22.970803121419365: 1, 22.900672913324073: 1, 22.838480668888646: 1, 22.
837587856271757: 1, 22.74266630083903: 1, 22.723046506167208: 1, 22.64897177700476: 1, 22.635704751195156: 1, 2
2.516729357309103: 1, 22.488954421948037: 1, 22.47640013109744: 1, 22.474148932692597: 1, 22.376411563782774: 1,
22.31202404186532: 1, 22.245216357814737: 1, 22.18249633421697: 1, 22.08213973019711: 1, 22.007975114185765: 1,
21.98987052035315: 1, 21.985164108184154: 1, 21.934722863708146: 1, 21.933291330620897: 1, 21.932318774561086:
1, 21.88619508739867: 1, 21.686952288932385: 1, 21.67191509519617: 1, 21.657504724348193: 1, 21.48837803396553:
1, 21.467310745050842: 1, 21.43990208937264: 1, 21.369064239146944: 1, 21.35081732677666: 1, 21.319094559624226:
1, 21.2498149302735: 1, 21.24426380230708: 1, 21.2370622386425: 1, 21.229285721319393: 1, 21.189951020620175: 1,
21.171184293974687: 1, 21.12436834785333: 1, 21.077075662651293: 1, 21.026199247704778: 1, 21.006116495036057:
1, 20.955448763946936: 1, 20.952009260234945: 1, 20.92748032375003: 1, 20.90896465340115: 1, 20.89351185671354:
1, 20.83462187740024: 1, 20.829820507612467: 1, 20.762567425438018: 1, 20.670103499442224: 1, 20.65896742071378:
1, 20.652105900116055: 1, 20.6074222038947: 1, 20.440173002829685: 1, 20.377347037768565: 1, 20.314387912276505:
1, 20.299944568914647: 1, 20.177065510173716: 1, 20.066448105704556: 1, 20.062652012371323: 1, 20.0616076177529
9: 1, 20.03488884152631: 1, 20.00708161753659: 1, 20.001069066901856: 1, 19.880699596988403: 1, 19.8219496802917
1: 1, 19.791913563451462: 1, 19.750781700512835: 1, 19.69854509138638: 1, 19.691051165435027: 1, 19.669176334590
254: 1, 19.61115721147714: 1, 19.597078242516236: 1, 19.464652279499088: 1, 19.45832655714815: 1, 19.45795078002
3854: 1, 19.428018535143455: 1, 19.363320007257173: 1, 19.29288478244963: 1, 19.237390247377075: 1, 19.204994471
9013: 1, 19.1797324585765: 1, 19.16971553013208: 1, 19.121880022545437: 1, 19.110592515510245: 1, 19.07196738294
758: 1, 19.033877113474016: 1, 19.02937541443437: 1, 18.999227694947404: 1, 18.97144671311694: 1, 18.91928156406
512: 1, 18.840318390061395: 1, 18.818695565774618: 1, 18.78763515415584: 1, 18.76308139860332: 1, 18.73989161121
2858: 1, 18.725640558293232: 1, 18.67609538779854: 1, 18.649345253964718: 1, 18.646145105718553: 1, 18.642745487
469348: 1, 18.626658541708547: 1, 18.610551310824267: 1, 18.60916859740142: 1, 18.51607032578995: 1, 18.48191113
76752: 1, 18.428643036990913: 1, 18.342955027203327: 1, 18.30367591577206: 1, 18.30311922405231: 1, 18.294414963
002705: 1, 18.26261088193725: 1, 18.23257864818012: 1, 18.2004468567305: 1, 18.18666941972467: 1, 18.15500959457
031: 1, 18.09425982201249: 1, 18.07407001356429: 1, 18.058445378119085: 1, 18.03017232748758: 1, 17.971923246082
035: 1, 17.95973218075607: 1, 17.958192602342674: 1, 17.874625550690585: 1, 17.80813622885757: 1, 17.77636373500
8665: 1, 17.772208226230575: 1, 17.759255991568583: 1, 17.750152711344825: 1, 17.747905930903336: 1, 17.74230356
723754: 1, 17.64282994691834: 1, 17.597240733687453: 1, 17.573552387096456: 1, 17.566838571187887: 1, 17.5573563
46410785: 1, 17.49741256510799: 1, 17.467177157467077: 1, 17.326925259472397: 1, 17.3150249703825: 1, 17.2952009
790365: 1, 17.267236109328273: 1, 17.230669218854427: 1, 17.2012918087547: 1, 17.20065005727575: 1, 17.158831263
502602: 1, 17.15160960662643: 1, 17.13930051227068: 1, 17.092934810128188: 1, 17.03417148715952: 1, 16.998742558
637808: 1, 16.96133906781007: 1, 16.926964274149523: 1, 16.906567744158863: 1, 16.879884347842168: 1, 16.8784226
80765194: 1, 16.869964016838495: 1, 16.864106290562557: 1, 16.851609784259985: 1, 16.73049226983154: 1, 16.69531
135143078: 1, 16.68625912966429: 1, 16.648718450967223: 1, 16.642930395839986: 1, 16.62267076146576: 1, 16.57650
5777079827: 1, 16.522008291964962: 1, 16.51879549454129: 1, 16.504470628671072: 1, 16.49898656904974: 1, 16.4941
0698455016: 1, 16.439585934182958: 1, 16.427006984684912: 1, 16.41207266967564: 1, 16.395221052460087: 1, 16.347
627845086926: 1, 16.320436027646274: 1, 16.19825190480459: 1, 16.18419121568417: 1, 16.177194936684675: 1, 16.14
817440742112: 1, 16.114031906914523: 1, 16.1108507868502: 1, 16.10635322763364: 1, 16.10274480870874: 1, 16.0991
03431350265: 1, 16.090409849285695: 1, 16.00868174337802: 1, 15.985862603276335: 1, 15.976108318532361: 1, 15.93
924844322395: 1, 15.889124367793066: 1, 15.868471896601564: 1, 15.861596886524165: 1, 15.828187894843689: 1, 15.
79647019764008: 1, 15.769508444025504: 1, 15.682117844906239: 1, 15.649661458346788: 1, 15.572006084646635: 1, 1
5.54859579477599: 1, 15.544132629580545: 1, 15.539775348430965: 1, 15.509923360430605: 1, 15.493499920689779: 1,
15.427163381028453: 1, 15.394748611380905: 1, 15.372777608652957: 1, 15.361857604764516: 1, 15.345674193508609:
1, 15.303492575254236: 1, 15.168201584055131: 1, 15.138628134043666: 1, 15.124858230987845: 1, 15.12090285448656
```

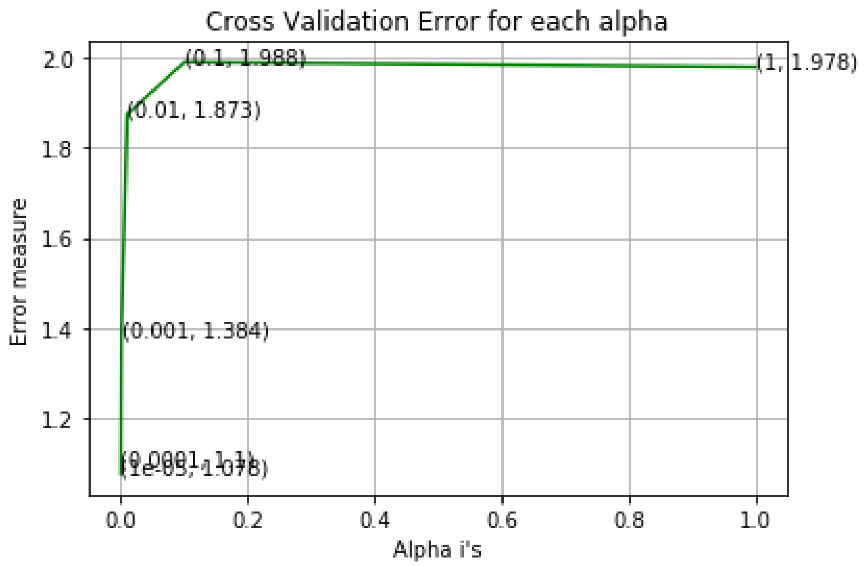
```
8: 1, 15.115299254785654: 1, 15.025365249920494: 1, 15.023900738390958: 1, 15.017124578065834: 1, 15.01317642994
0874: 1, 15.004141651056278: 1, 14.965648952849085: 1, 14.95565447811964: 1, 14.931013569756473: 1, 14.925751748
409233: 1, 14.895607876675083: 1, 14.836009087043847: 1, 14.808063450978876: 1, 14.791639848605536: 1, 14.783283
083873341: 1, 14.742352223010945: 1, 14.730790644966273: 1, 14.73048865808426: 1, 14.723946761849788: 1, 14.7051
41719288623: 1, 14.659934396300335: 1, 14.648717515470803: 1, 14.648362580589371: 1, 14.645054977953507: 1, 14.5
80831620721554: 1, 14.57742149012579: 1, 14.574484973090232: 1, 14.570374019139546: 1, 14.565566539955148: 1, 1
4.550785631921151: 1, 14.544706457697936: 1, 14.500858962582267: 1, 14.470507241025398: 1, 14.44953508335837: 1,
14.4482805282553: 1, 14.446763540042888: 1, 14.445051638738482: 1, 14.444337401004931: 1, 14.434591398132383: 1,
14.417966276535205: 1, 14.390374395081714: 1, 14.38056635418232: 1, 14.31651993708069: 1, 14.275489680916854: 1,
14.269653552777473: 1, 14.260913597009422: 1, 14.259409408339483: 1, 14.24946406545267: 1, 14.238294609600787:
1, 14.2133965720536: 1, 14.202440768971117: 1, 14.197393844566479: 1, 14.192959482892382: 1, 14.179834876485673:
1, 14.167877847834115: 1, 14.154102595774301: 1, 14.135556781652001: 1, 14.071453945913074: 1, 14.03898737387353
7: 1, 14.025238316677445: 1, 14.025088316995923: 1, 13.988537830051312: 1, 13.973887349644478: 1, 13.96196096594
2241: 1, 13.9428218687359: 1, 13.926951112529736: 1, 13.886446927521595: 1, 13.885233825356764: 1, 13.8837797369
75532: 1, 13.880439247718622: 1, 13.856924725061768: 1, 13.846860499865263: 1, 13.777620690271961: 1, 13.7755766
70610084: 1, 13.7478460619102: 1, 13.717851550167458: 1, 13.67891616200589: 1, 13.674148936767262: 1, 13.6198636
53942723: 1, 13.598524408010046: 1, 13.564651426175535: 1, 13.518165231452889: 1, 13.503206038836547: 1, 13.4835
6533236529: 1, 13.461499351092876: 1, 13.454371089885472: 1, 13.405491765068664: 1, 13.400312268328122: 1, 13.36
241289006254: 1, 13.35841979152609: 1, 13.358180601658635: 1, 13.326337965808394: 1, 13.258379240689344: 1, 13.1
90133810744657: 1, 13.16002899712209: 1, 13.142073372980423: 1, 13.093175485665718: 1, 13.081871111300687: 1, 1
3.022832829042985: 1, 13.01379663530376: 1, 12.997750671538514: 1, 12.984977786558126: 1, 12.934291020160478: 1,
12.91461881050302: 1, 12.905882847774715: 1, 12.900742899711345: 1, 12.890158378043445: 1, 12.843479286991425:
1, 12.842145024038192: 1, 12.834923596788766: 1, 12.828717938328333: 1, 12.79370746878482: 1, 12.77187439950969
5: 1, 12.767961551891279: 1, 12.767055305175191: 1, 12.75876226604579: 1, 12.7229317463763: 1, 12.70267176804438
8: 1, 12.666565093215661: 1, 12.666104423384377: 1, 12.659904933204514: 1, 12.642763237678285: 1, 12.59475107924
6125: 1, 12.580891121455418: 1, 12.56458850448097: 1, 12.560106633836579: 1, 12.548758085232524: 1, 12.542536819
989307: 1, 12.540108231085073: 1, 12.51990686505701: 1, 12.50478640634645: 1, 12.504161482909128: 1, 12.49611704
4588287: 1, 12.485181924304184: 1, 12.468568548078997: 1, 12.44446969265212: 1, 12.437985878395736: 1, 12.397798
447031388: 1, 12.37378318701037: 1, 12.353846547754934: 1, 12.349215308891003: 1, 12.300284266257426: 1, 12.2730
52702341046: 1, 12.268529175009618: 1, 12.267022215347007: 1, 12.263874529186719: 1, 12.256266003819517: 1, 12.2
53676091614015: 1, 12.227062135400093: 1, 12.224966051105786: 1, 12.222670042549023: 1, 12.163729084973149: 1, 1
2.16075029498224: 1, 12.159209748563883: 1, 12.123686959636265: 1, 12.101210490534411: 1, 12.065782321998437: 1,
12.063823837355013: 1, 12.057255170263518: 1, 12.028955964510583: 1, 12.021644222781344: 1, 12.000751246834529:
1, 11.979417137106447: 1, 11.975579816649256: 1, 11.948337595370116: 1, 11.921673035866204: 1, 11.90364301480704
7: 1, 11.860204355818253: 1, 11.858813431144112: 1, 11.85759535039639: 1, 11.831055532072252: 1, 11.826269514569
224: 1, 11.82506145730606: 1, 11.821775510449323: 1, 11.7775557638019: 1, 11.772613094645147: 1, 11.771957090092
037: 1, 11.762045062636156: 1, 11.732259075221599: 1, 11.722765942669712: 1, 11.7214893694628: 1, 11.69051794594
4435: 1, 11.680578784045538: 1, 11.673606291743765: 1, 11.67246136424945: 1, 11.659654175498833: 1, 11.614026010
240464: 1, 11.610016511576907: 1, 11.608294412267346: 1, 11.568407262458418: 1, 11.553227520629706: 1, 11.509685
579726293: 1, 11.490943302698154: 1, 11.484961553861426: 1, 11.474333956817656: 1, 11.466580036304455: 1, 11.457
```

```
744699396644: 1, 11.454764729475787: 1, 11.434870033746972: 1, 11.42490826362254: 1, 11.422276636757273: 1, 11.4
16375427564585: 1, 11.416253284256298: 1, 11.38585140210466: 1, 11.377019950174056: 1, 11.376202388526078: 1, 1
1.374379676498501: 1, 11.365352305716842: 1, 11.350101280633504: 1, 11.338861655022155: 1, 11.325994194754928:
1, 11.299722998262386: 1, 11.29693921652356: 1, 11.289485681254366: 1, 11.260316428887537: 1, 11.24994718709102
8: 1, 11.21506688160708: 1, 11.180800438230081: 1, 11.180600977402975: 1, 11.160569152681147: 1, 11.119834406356
308: 1, 11.080898315369117: 1, 11.068314782967304: 1, 11.05896098400603: 1, 11.050482075806359: 1, 11.0287057604
39264: 1, 11.024606684393685: 1, 11.015099940658665: 1, 11.013136454226002: 1, 11.010298280152492: 1, 10.9850063
78725696: 1, 10.981392770290201: 1, 10.975163998708855: 1, 10.953360790711374: 1, 10.94583517023661: 1, 10.91624
1035687177: 1, 10.911105256864351: 1, 10.907921009642116: 1, 10.889867765905503: 1, 10.864994464338377: 1, 10.85
7766692123421: 1, 10.853252051305263: 1, 10.847126833128051: 1, 10.83851256754278: 1, 10.835022656491907: 1, 10.
826988961800645: 1, 10.823714571131411: 1, 10.821745448254797: 1, 10.818271076395439: 1, 10.817356648985278: 1,
10.814601046578698: 1, 10.766669162699674: 1, 10.766586087225791: 1, 10.756747061298444: 1, 10.750877308880908:
1, 10.749466442305286: 1, 10.740977670197365: 1, 10.733184109549022: 1, 10.717685926491143: 1, 10.71691839818792
9: 1, 10.680655580003256: 1, 10.661628761320165: 1, 10.620026311174902: 1, 10.601479227305303: 1, 10.59316195365
4237: 1, 10.570616434502684: 1, 10.555204301144856: 1, 10.54732331638825: 1, 10.542734197635939: 1, 10.538023849
96284: 1, 10.532855325844864: 1, 10.5286851894222: 1, 10.517561656268594: 1, 10.498646876731286: 1, 10.490153350
790704: 1, 10.483996605245947: 1, 10.482972802064056: 1, 10.47565820615753: 1, 10.4391316243572: 1, 10.421591464
030907: 1, 10.418717308492587: 1, 10.418160169513554: 1, 10.400284374058222: 1, 10.38950160945033: 1, 10.3860327
07655293: 1, 10.385694078589136: 1, 10.375588522390371: 1, 10.360841094942698: 1, 10.359107412908807: 1, 10.3541
28013157084: 1, 10.331313073901324: 1, 10.3117427325553: 1, 10.302947086048464: 1, 10.28954797872161: 1, 10.2794
10696824474: 1, 10.27177312877801: 1, 10.269535679429811: 1, 10.241702202012155: 1, 10.236914077649878: 1, 10.19
3087855008924: 1, 10.178138297718881: 1, 10.171971031244956: 1, 10.168502053196498: 1, 10.143690171604126: 1, 1
0.136991452496636: 1, 10.133005571993213: 1, 10.122501880371022: 1, 10.103305053730423: 1, 10.086552250934178:
1, 10.067314343887345: 1, 10.039370478187031: 1, 10.037065850855276: 1, 10.036289289933125: 1, 10.02371398041918
1: 1, 10.016947629566701: 1, 9.995712986858184: 1, 9.994278153855555: 1, 9.98990094440999: 1, 9.98131127234761:
1, 9.967559342358173: 1, 9.96618467607766: 1, 9.937020795772899: 1, 9.930768424382576: 1, 9.922372207029946: 1,
9.900572895995786: 1, 9.894860893223976: 1, 9.886592063695044: 1, 9.883706626599885: 1, 9.866676601167931: 1, 9.
846728187380108: 1, 9.830549741903907: 1, 9.82874127404943: 1, 9.81528775704133: 1, 9.809988054289201: 1, 9.8076
36414496509: 1, 9.782010813006664: 1, 9.771403903080092: 1, 9.770507335649947: 1, 9.7662620754472 9: 1, 9.7603508
25219643: 1, 9.759011120237616: 1, 9.677536674138041: 1, 9.676719001479654: 1, 9.661938299295088: 1, 9.661725470
646175: 1, 9.65841886112682: 1, 9.649940526921421: 1, 9.648930829202298: 1, 9.644731001174168: 1, 9.639933524639
4: 1, 9.611042524653037: 1, 9.58181118729836: 1, 9.56649012985495: 1, 9.546996824282122: 1, 9.53581808953782: 1,
9.531709395303468: 1, 9.52840188011146: 1, 9.486962643889774: 1, 9.484093856543968: 1, 9.464168169728683: 1, 9.4
63323920233165: 1, 9.458619654689704: 1, 9.454849800248319: 1, 9.4524824035604: 1, 9.438421521177405: 1, 9.41909
1209182925: 1, 9.411831159423542: 1, 9.408662668458005: 1, 9.389700074247777: 1, 9.37794552641875: 1, 9.37447318
5703595: 1, 9.372270590662437: 1, 9.369058008901234: 1, 9.346387002967903: 1, 9.333348572322208: 1, 9.3184661643
9376: 1, 9.31388907479819: 1, 9.305694992304911: 1, 9.294581979954916: 1, 9.286357104702832: 1, 9.27645408295282
8: 1, 9.264483851738325: 1, 9.264230401645461: 1, 9.25432449234352: 1, 9.252976156549012: 1, 9.25258685355668:
1, 9.247629088087438: 1, 9.244608472152292: 1, 9.244028783314292: 1, 9.2074915815753: 1, 9.206908811948425: 1,
9.200248373382456: 1, 9.195437799668166: 1, 9.194078774617834: 1, 9.178765030481538: 1, 9.172466822846124: 1, 9.
```

```
144320999465737: 1, 9.144120293652554: 1, 9.139296760613714: 1, 9.124737573224554: 1, 9.122945557119925: 1, 9.11
6987238729312: 1, 9.116087588036208: 1, 9.114885331636327: 1, 9.100260251993426: 1, 9.097326388176675: 1, 9.0941
90818991542: 1, 9.082147565461726: 1, 9.08068000096498: 1, 9.070272334879482: 1, 9.061266553991: 1, 9.0605521407
33616: 1, 9.050237648993868: 1, 9.04615062095357: 1, 9.033425535438605: 1, 9.03062679174733: 1, 9.02882365956463
3: 1, 9.013414554761361: 1, 9.00583382358591: 1, 8.998995827225974: 1, 8.989202652670542: 1, 8.986273802531926:
1, 8.97583534200499: 1, 8.96363278708198: 1, 8.961413597321405: 1, 8.953389514546195: 1, 8.952560252217387: 1,
8.950304882100893: 1, 8.94266250359515: 1, 8.941326638512377: 1, 8.910360397550082: 1, 8.910250655097485: 1, 8.9
09563250084938: 1, 8.891464656361938: 1, 8.88448818077858: 1, 8.88024215544865: 1, 8.879249350914169: 1, 8.86783
9530071766: 1, 8.861303917150112: 1, 8.858027135363454: 1, 8.84400156483756: 1, 8.833979565245748: 1, 8.82897554
9831991: 1, 8.813609219497451: 1, 8.802088216323188: 1, 8.789861090342539: 1, 8.786656428792693: 1, 8.7855845137
65654: 1, 8.769318306120537: 1, 8.752617164993644: 1, 8.735593096274608: 1, 8.730455646236628: 1, 8.715514164944
44: 1, 8.713740249543555: 1, 8.704576341439648: 1, 8.704486842308627: 1, 8.668682740511034: 1, 8.65082575601675
3: 1, 8.650470487373974: 1, 8.64125807850648: 1, 8.636377055788095: 1, 8.632055433634616: 1, 8.626980316795429:
1, 8.624178565649196: 1, 8.599136811150508: 1, 8.596232919961441: 1, 8.594387856385513: 1, 8.575191940769288: 1,
8.54840885856775: 1, 8.541333367743409: 1, 8.49988026743655: 1, 8.48233686027317: 1, 8.469675479711785: 1, 8.469
021270165348: 1, 8.466428001680312: 1, 8.45251595367837: 1, 8.437139083080515: 1, 8.436656862695585: 1, 8.420648
232390715: 1, 8.369611899411089: 1, 8.369492015945989: 1, 8.348465629391702: 1, 8.342319725369423: 1, 8.33206572
3595306: 1, 8.330232343599791: 1, 8.326517888545041: 1, 8.322245413155073: 1, 8.306401590891573: 1, 8.2843727460
01221: 1, 8.283087750247363: 1, 8.261854658612057: 1, 8.259579928315608: 1, 8.258119067399576: 1, 8.241577460172
847: 1, 8.235858104731093: 1, 8.225006999344792: 1, 8.209656705405246: 1, 8.14206921094283: 1, 8.13182357301515
6: 1, 8.102025162103825: 1, 8.099300592581137: 1, 8.098390328735004: 1, 8.096108573445376: 1, 8.090717676736805:
1, 8.086011838053226: 1, 8.084819941311117: 1, 8.083252747425039: 1, 8.073861655909834: 1, 8.069865765804101: 1,
8.061836536589768: 1, 8.020806360855142: 1, 8.019852055715349: 1, 8.01631063764562: 1, 8.01599883736217: 1, 8.00
848093239325: 1, 8.00044325679398: 1, 8.000006964166067: 1, 7.9995681807740295: 1, 7.969396893628262: 1, 7.95507
54002868524: 1, 7.952623788394996: 1, 7.951312392746429: 1, 7.937763938168826: 1, 7.913858234853997: 1, 7.907223
7003517385: 1, 7.895575254549794: 1, 7.873299427583269: 1, 7.8590460358514465: 1, 7.8478849098794115: 1, 7.82834
4011856274: 1, 7.8253576467225585: 1, 7.817605890939118: 1, 7.8047683182405025: 1, 7.796286483601219: 1, 7.78308
6578374863: 1, 7.7788034082400115: 1, 7.748412591630108: 1, 7.745244602222587: 1, 7.7373126677788715: 1, 7.73360
423592991: 1, 7.714136395263808: 1, 7.708980739738527: 1, 7.707949344773889: 1, 7.7041486359224445: 1, 7.6959681
82173223: 1, 7.679519181403692: 1, 7.665703353184878: 1, 7.6379696957686685: 1, 7.587803349175346: 1, 7.58454736
9854338: 1, 7.5784748735938505: 1, 7.5656095779816415: 1, 7.558747547461828: 1, 7.557861332215591: 1, 7.52706623
4635643: 1, 7.522775660639606: 1, 7.515908816187682: 1, 7.505891675793847: 1, 7.449189837726829: 1, 7.4291122032
94294: 1, 7.415394363232918: 1, 7.383409332935221: 1, 7.382969176501172: 1, 7.381449684558363: 1, 7.373933780240
052: 1, 7.311799088146715: 1, 7.310878629083388: 1, 7.297587348733911: 1, 7.295425619811468: 1, 7.27954027776126
3: 1, 7.277422402020437: 1, 7.260628976110228: 1, 7.247003797233445: 1, 7.2435414477210545: 1, 7.22772719777084
8: 1, 7.2019539087541: 1, 7.193159990484473: 1, 7.136524544321367: 1, 7.079788466981915: 1, 7.0793934451619505:
1, 7.065725883633098: 1, 7.062832392868323: 1, 7.0034828876148625: 1, 6.978821742276303: 1, 6.963353237799388:
1, 6.961708834040785: 1, 6.91130220582727: 1, 6.900166189925014: 1, 6.875233510896341: 1, 6.837080991638119: 1,
6.813362630293442: 1, 6.803339320897403: 1, 6.7805183467784795: 1, 6.769570311986878: 1, 6.733510714351341: 1,
6.704507933038503: 1, 6.6851261446627035: 1, 6.597762769072989: 1, 6.486750012155753: 1, 6.407559663461865: 1,
5.848306426960806: 1})
```

In [375]:
```
# Train a Logistic regression+Calibration model using text features whicha re on-hot encoded
alpha = [10 ** x for x in range(-5, 1)]

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

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

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



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.classes_, eps=1e-
15))

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.0778869952347918
For values of alpha =  0.0001 The log loss is: 1.0997223099472682
For values of alpha =  0.001 The log loss is: 1.3836520903122105
For values of alpha =  0.01 The log loss is: 1.872819204234293
For values of alpha =  0.1 The log loss is: 1.9882175722308055
For values of alpha =  1 The log loss is: 1.977680798121613
```



```
For values of best alpha =  1e-05 The train log loss is: 0.7844816473552603
For values of best alpha =  1e-05 The cross validation log loss is: 1.0778869952347918
For values of best alpha =  1e-05 The test log loss is: 1.120260237189313
```

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

Ans. Yes, it seems like!

In [376]:
```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=3,max_features=1000)
    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 [377]:
```
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")
```

```
94.2 % of word of test data appeared in train data
93.9 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [378]:
```
#Data preparation for ML models.

#Misc. functionns for ML models


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

    # for calculating log_loss we willl provide the array of probabilities belongs to 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 [379]:
```
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 [380]:
```
# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3,max_features=1000)

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

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

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

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

Stacking the three types of features

In [381]:
```
# merging gene, variance and text features

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

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

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

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

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


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_variation_feature_responseCod
ing))
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 [382]:
```
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, 3197)
(number of data points * number of features) in test data =  (665, 3197)
(number of data points * number of features) in cross validation data = (532, 3197)
```

In [383]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_responseCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_responseCoding.shape)
```

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

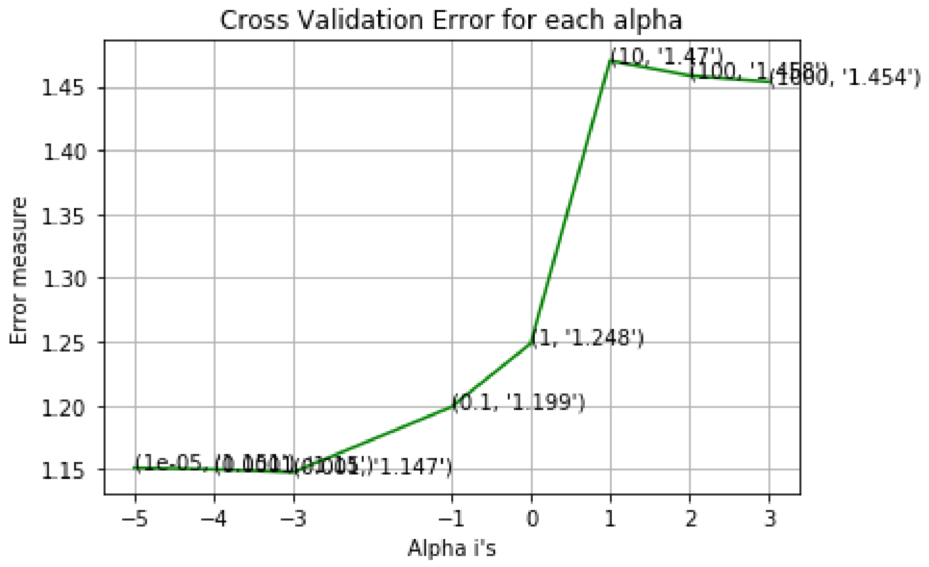
4.1. Base Line Model

4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [384]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.
naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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


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

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


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


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

```
for alpha = 1e-05
Log Loss : 1.1511292200784728
for alpha = 0.0001
Log Loss : 1.1498816566161665
for alpha = 0.001
Log Loss : 1.1473977533450992
for alpha = 0.1
Log Loss : 1.1986197146861954
for alpha = 1
Log Loss : 1.248277288617506
for alpha = 10
Log Loss : 1.4700829893631393
for alpha = 100
Log Loss : 1.4583973801119625
for alpha = 1000
Log Loss : 1.4535854663227001
```

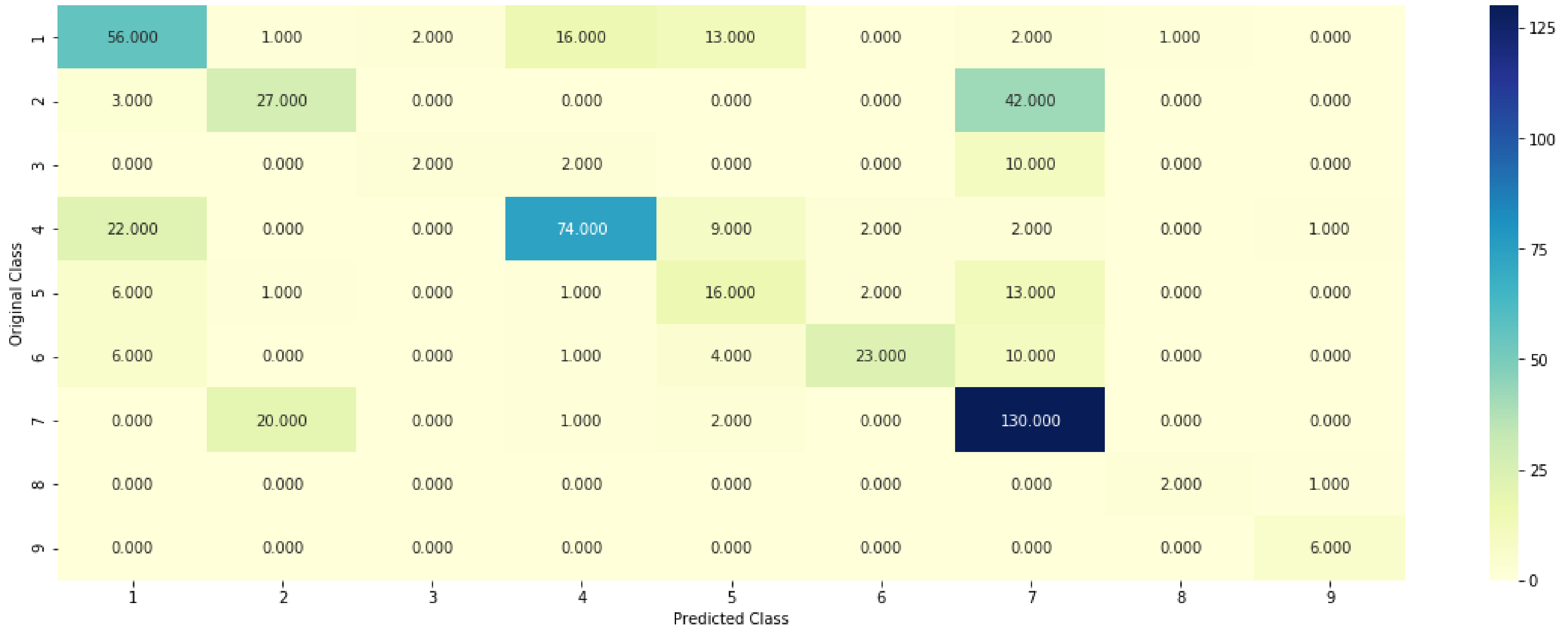


```
For values of best alpha =  0.001 The train log loss is: 0.5363341707740532
For values of best alpha =  0.001 The cross validation log loss is: 1.1473977533450992
For values of best alpha =  0.001 The test log loss is: 1.1714342464361676
```

4.1.1.2. Testing the model with best hyper paramters

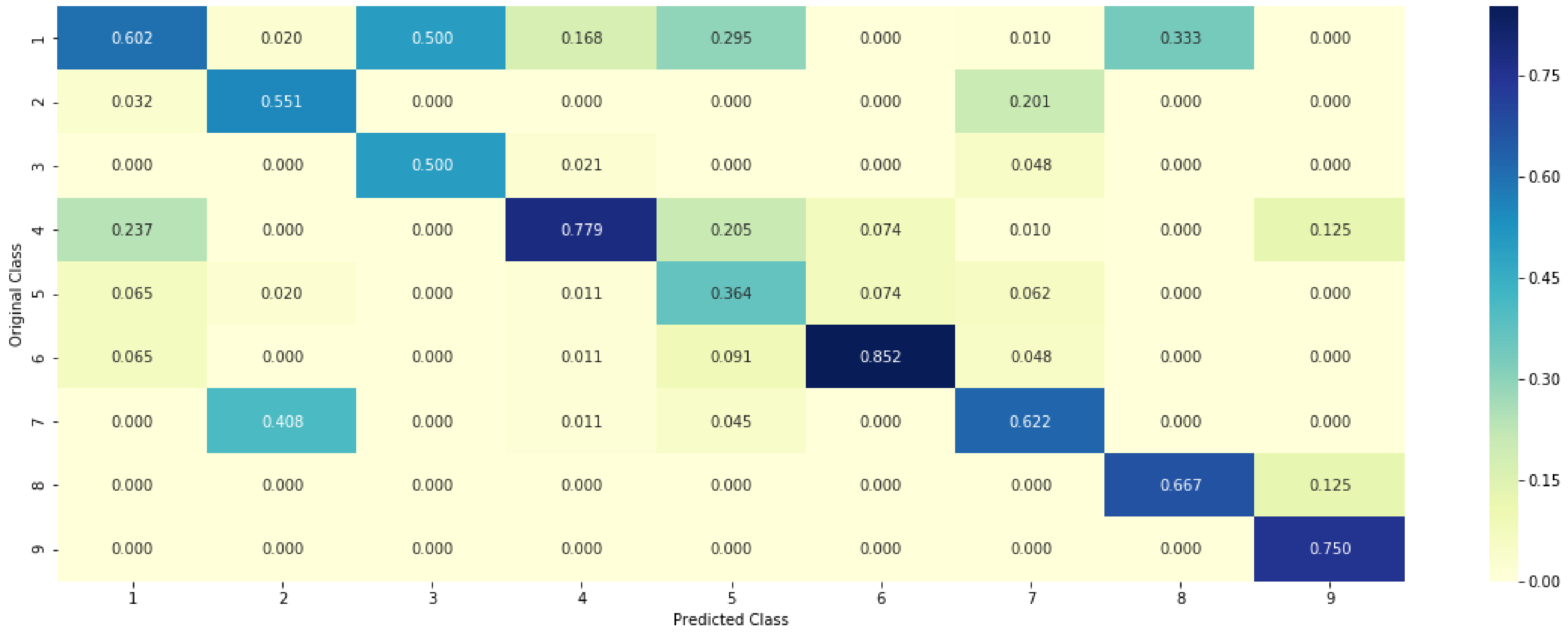
In [385]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.
naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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

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

```
Log Loss : 1.1473977533450992
Number of missclassified point : 0.3684210526315789
-------------------- Confusion matrix --------------------
```



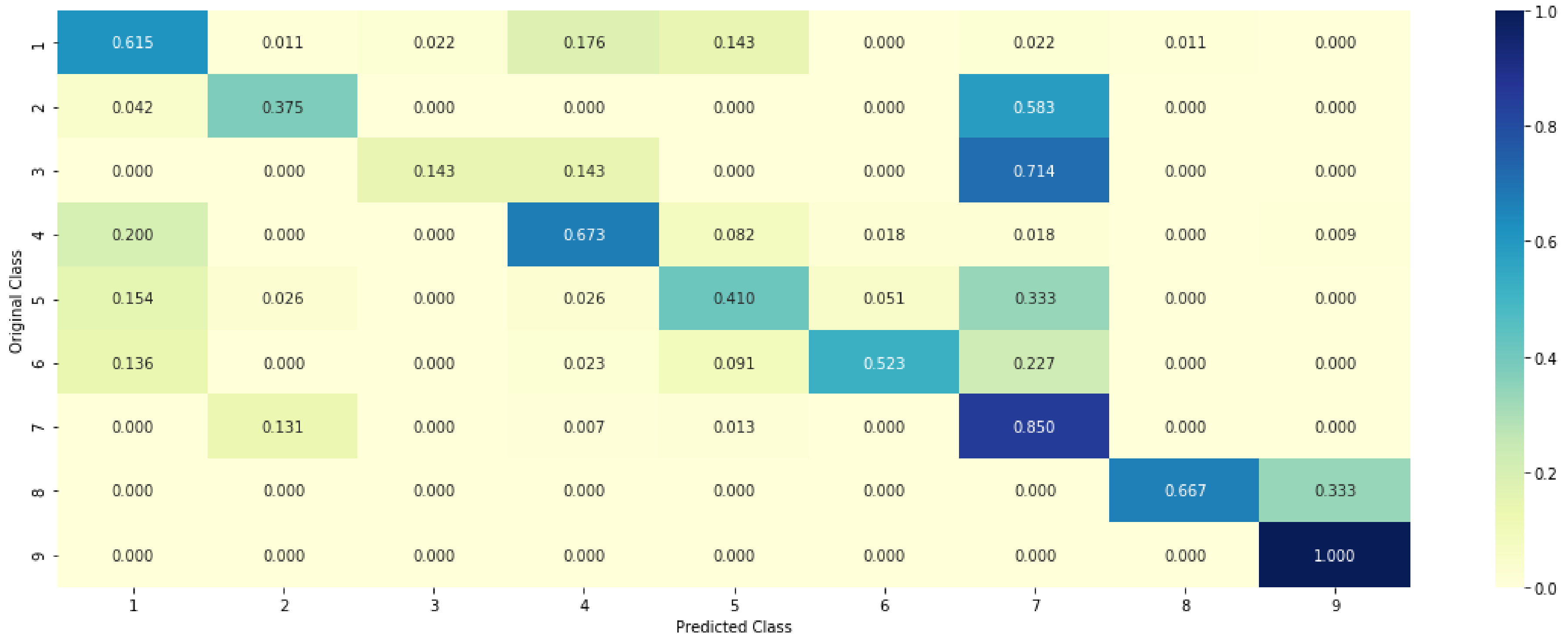
Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	56.000	1.000	2.000	16.000	13.000	0.000	2.000	1.000	0.000
2	3.000	27.000	0.000	0.000	0.000	0.000	42.000	0.000	0.000
3	0.000	0.000	2.000	2.000	0.000	0.000	10.000	0.000	0.000
4	22.000	0.000	0.000	74.000	9.000	2.000	2.000	0.000	1.000
5	6.000	1.000	0.000	1.000	16.000	2.000	13.000	0.000	0.000
6	6.000	0.000	0.000	1.000	4.000	23.000	10.000	0.000	0.000
7	0.000	20.000	0.000	1.000	2.000	0.000	130.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	2.000	1.000
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	6.000

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



	1	2	3	4	5	6	7	8	9
1	0.602	0.020	0.500	0.168	0.295	0.000	0.010	0.333	0.000
2	0.032	0.551	0.000	0.000	0.000	0.000	0.201	0.000	0.000
3	0.000	0.000	0.500	0.021	0.000	0.000	0.048	0.000	0.000
4	0.237	0.000	0.000	0.779	0.205	0.074	0.010	0.000	0.125
5	0.065	0.020	0.000	0.011	0.364	0.074	0.062	0.000	0.000
6	0.065	0.000	0.000	0.011	0.091	0.852	0.048	0.000	0.000
7	0.000	0.408	0.000	0.011	0.045	0.000	0.622	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.667	0.125
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.750

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



	1	2	3	4	5	6	7	8	9
1	0.615	0.011	0.022	0.176	0.143	0.000	0.022	0.011	0.000
2	0.042	0.375	0.000	0.000	0.000	0.000	0.583	0.000	0.000
3	0.000	0.000	0.143	0.143	0.000	0.000	0.714	0.000	0.000
4	0.200	0.000	0.000	0.673	0.082	0.018	0.018	0.000	0.009
5	0.154	0.026	0.000	0.026	0.410	0.051	0.333	0.000	0.000
6	0.136	0.000	0.000	0.023	0.091	0.523	0.227	0.000	0.000
7	0.000	0.131	0.000	0.007	0.013	0.000	0.850	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.667	0.333
9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000

4.1.1.3. Feature Importance, Correctly classified point

In [386]:
```
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],t
est_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0673 0.0583 0.0132 0.0706 0.0346 0.3923 0.3559 0.0039 0.0039]]
Actual Class : 6
--------------------------------------------------
16 Text feature [predicted] present in test data point [True]
17 Text feature [family] present in test data point [True]
18 Text feature [substitutions] present in test data point [True]
19 Text feature [models] present in test data point [True]
20 Text feature [49] present in test data point [True]
22 Text feature [expected] present in test data point [True]
23 Text feature [43] present in test data point [True]
24 Text feature [evidence] present in test data point [True]
25 Text feature [model] present in test data point [True]
29 Text feature [variant] present in test data point [True]
30 Text feature [use] present in test data point [True]
33 Text feature [likely] present in test data point [True]
34 Text feature [known] present in test data point [True]
38 Text feature [variants] present in test data point [True]
39 Text feature [31] present in test data point [True]
43 Text feature [substitution] present in test data point [True]
44 Text feature [given] present in test data point [True]
45 Text feature [used] present in test data point [True]
46 Text feature [conserved] present in test data point [True]
48 Text feature [interaction] present in test data point [True]
49 Text feature [23] present in test data point [True]
50 Text feature [would] present in test data point [True]
51 Text feature [proportion] present in test data point [True]
52 Text feature [data] present in test data point [True]
58 Text feature [significant] present in test data point [True]
61 Text feature [site] present in test data point [True]
63 Text feature [values] present in test data point [True]
67 Text feature [thus] present in test data point [True]
68 Text feature [45] present in test data point [True]
69 Text feature [25] present in test data point [True]
71 Text feature [although] present in test data point [True]
72 Text feature [50] present in test data point [True]
74 Text feature [analysis] present in test data point [True]
75 Text feature [46] present in test data point [True]
80 Text feature [32] present in test data point [True]
81 Text feature [three] present in test data point [True]
```

```
82 Text feature [provide] present in test data point [True]
83 Text feature [developed] present in test data point [True]
84 Text feature [affect] present in test data point [True]
87 Text feature [based] present in test data point [True]
88 Text feature [many] present in test data point [True]
89 Text feature [56] present in test data point [True]
93 Text feature [genetic] present in test data point [True]
94 Text feature [studies] present in test data point [True]
95 Text feature [two] present in test data point [True]
98 Text feature [residues] present in test data point [True]
Out of the top  100  features  46 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [387]:
```
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],t
est_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.059  0.0421 0.0116 0.0618 0.0299 0.0332 0.7556 0.0033 0.0034]]
Actual Class : 7
--------------------------------------------------
18 Text feature [activation] present in test data point [True]
19 Text feature [activated] present in test data point [True]
20 Text feature [downstream] present in test data point [True]
21 Text feature [kinase] present in test data point [True]
22 Text feature [cells] present in test data point [True]
23 Text feature [signaling] present in test data point [True]
24 Text feature [inhibitor] present in test data point [True]
25 Text feature [expressing] present in test data point [True]
26 Text feature [also] present in test data point [True]
27 Text feature [independent] present in test data point [True]
28 Text feature [contrast] present in test data point [True]
29 Text feature [10] present in test data point [True]
31 Text feature [recently] present in test data point [True]
32 Text feature [growth] present in test data point [True]
33 Text feature [however] present in test data point [True]
35 Text feature [factor] present in test data point [True]
36 Text feature [addition] present in test data point [True]
37 Text feature [shown] present in test data point [True]
38 Text feature [activating] present in test data point [True]
39 Text feature [constitutive] present in test data point [True]
40 Text feature [compared] present in test data point [True]
42 Text feature [previously] present in test data point [True]
43 Text feature [similar] present in test data point [True]
44 Text feature [mutations] present in test data point [True]
45 Text feature [presence] present in test data point [True]
46 Text feature [well] present in test data point [True]
47 Text feature [higher] present in test data point [True]
50 Text feature [treatment] present in test data point [True]
51 Text feature [found] present in test data point [True]
52 Text feature [3b] present in test data point [True]
54 Text feature [potential] present in test data point [True]
55 Text feature [may] present in test data point [True]
56 Text feature [suggest] present in test data point [True]
57 Text feature [cell] present in test data point [True]
58 Text feature [oncogenic] present in test data point [True]
59 Text feature [increased] present in test data point [True]
```

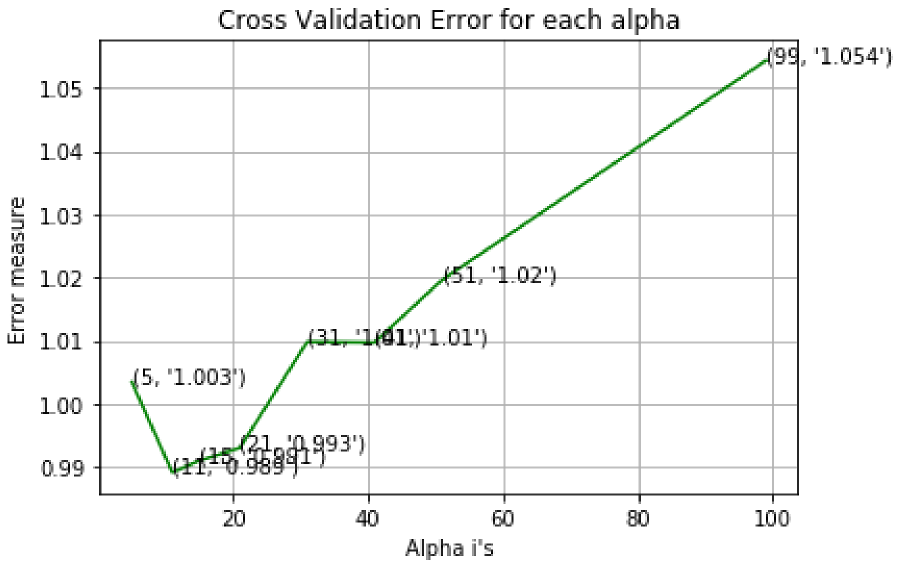
```
60 Text feature [described] present in test data point [True]
62 Text feature [inhibition] present in test data point [True]
63 Text feature [although] present in test data point [True]
64 Text feature [total] present in test data point [True]
65 Text feature [12] present in test data point [True]
66 Text feature [activate] present in test data point [True]
68 Text feature [results] present in test data point [True]
69 Text feature [without] present in test data point [True]
71 Text feature [mutation] present in test data point [True]
73 Text feature [tyrosine] present in test data point [True]
75 Text feature [two] present in test data point [True]
77 Text feature [3a] present in test data point [True]
78 Text feature [enhanced] present in test data point [True]
79 Text feature [observed] present in test data point [True]
80 Text feature [respectively] present in test data point [True]
81 Text feature [20] present in test data point [True]
82 Text feature [identified] present in test data point [True]
83 Text feature [pathways] present in test data point [True]
84 Text feature [including] present in test data point [True]
85 Text feature [mechanism] present in test data point [True]
86 Text feature [constitutively] present in test data point [True]
87 Text feature [pathway] present in test data point [True]
88 Text feature [effective] present in test data point [True]
89 Text feature [reported] present in test data point [True]
90 Text feature [study] present in test data point [True]
91 Text feature [discussion] present in test data point [True]
93 Text feature [13] present in test data point [True]
95 Text feature [therapeutic] present in test data point [True]
96 Text feature [proliferation] present in test data point [True]
97 Text feature [survival] present in test data point [True]
98 Text feature [15] present in test data point [True]
99 Text feature [fig] present in test data point [True]
Out of the top  100  features  68 are present in query point
```

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [388]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbor
s.KNeighborsClassifier.html
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights=’uniform’, algorithm=’auto’, leaf_size=30, p=2,
# metric=’minkowski’, metric_params=None, n_jobs=1, **kwargs)

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


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


alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i,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))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

```
for alpha = 5
Log Loss : 1.0034472315235965
for alpha = 11
Log Loss : 0.9891188435109726
for alpha = 15
Log Loss : 0.9909643241962542
for alpha = 21
Log Loss : 0.992965884600427
for alpha = 31
Log Loss : 1.0097818243321601
for alpha = 41
Log Loss : 1.0096866946324106
for alpha = 51
Log Loss : 1.0195829209109826
for alpha = 99
Log Loss : 1.0543009220411574
```

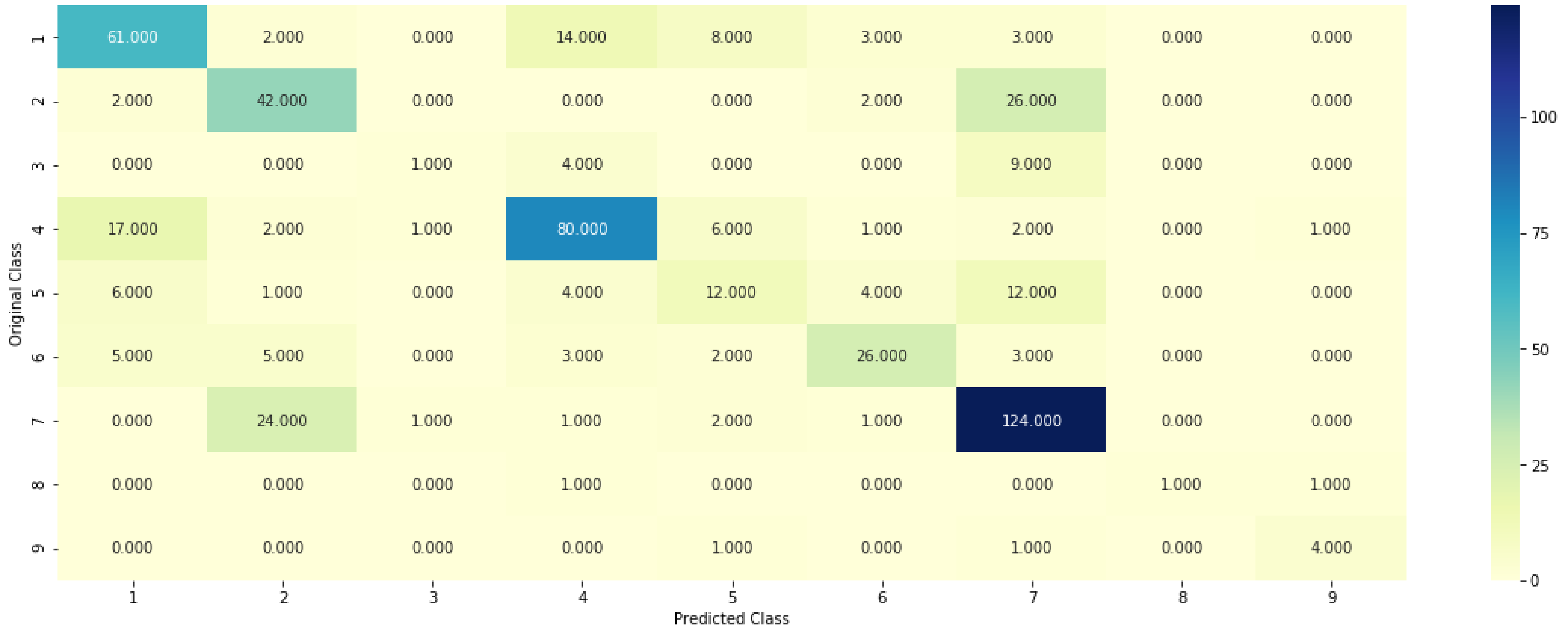


```
For values of best alpha =  11 The train log loss is: 0.6365580949459336
For values of best alpha =  11 The cross validation log loss is: 0.9891188435109726
For values of best alpha =  11 The test log loss is: 1.0010088465702234
```

4.2.2. Testing the model with best hyper paramters

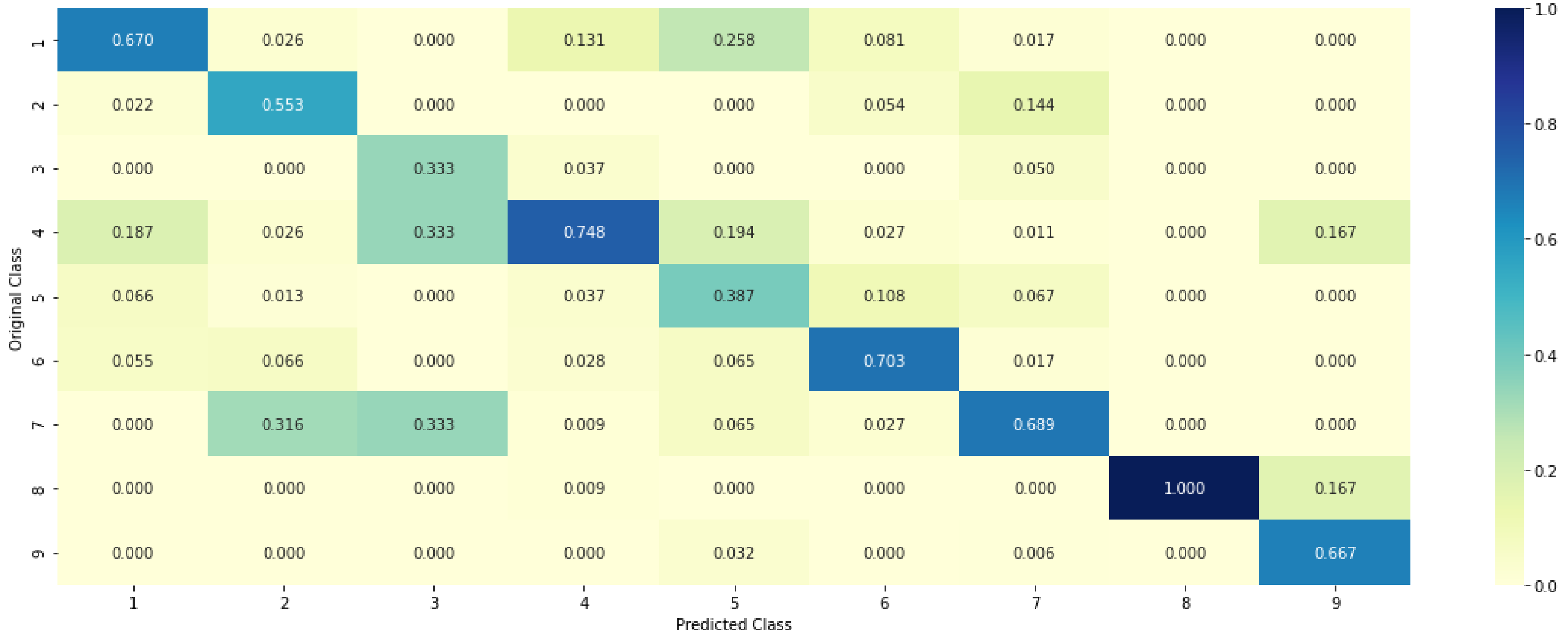
In [389]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbor
s.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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

```
Log loss : 0.9891188435109726
Number of mis-classified points : 0.34022556390977443
-------------------- Confusion matrix --------------------
```

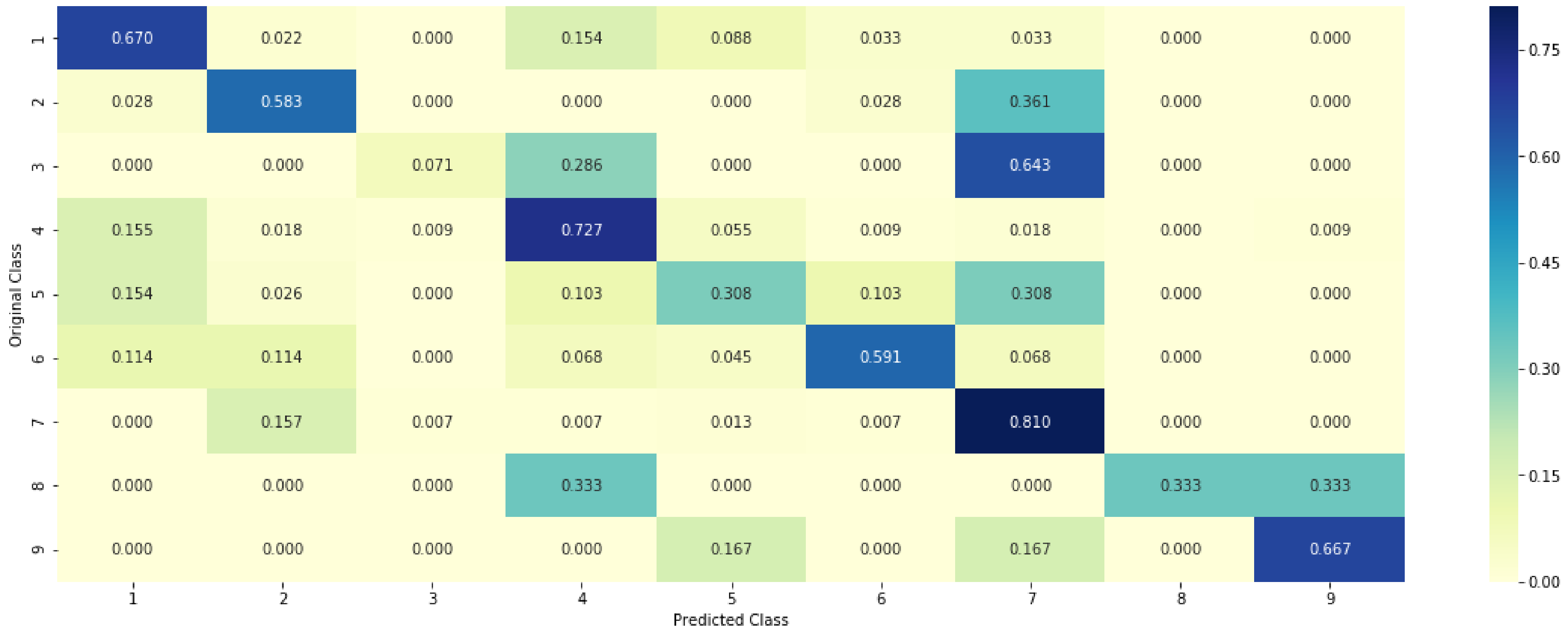


	1	2	3	4	5	6	7	8	9
1	61.000	2.000	0.000	14.000	8.000	3.000	3.000	0.000	0.000
2	2.000	42.000	0.000	0.000	0.000	2.000	26.000	0.000	0.000
3	0.000	0.000	1.000	4.000	0.000	0.000	9.000	0.000	0.000
4	17.000	2.000	1.000	80.000	6.000	1.000	2.000	0.000	1.000
5	6.000	1.000	0.000	4.000	12.000	4.000	12.000	0.000	0.000
6	5.000	5.000	0.000	3.000	2.000	26.000	3.000	0.000	0.000
7	0.000	24.000	1.000	1.000	2.000	1.000	124.000	0.000	0.000
8	0.000	0.000	0.000	1.000	0.000	0.000	0.000	1.000	1.000
9	0.000	0.000	0.000	0.000	1.000	0.000	1.000	0.000	4.000

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



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



	1	2	3	4	5	6	7	8	9
1	0.670	0.022	0.000	0.154	0.088	0.033	0.033	0.000	0.000
2	0.028	0.583	0.000	0.000	0.000	0.028	0.361	0.000	0.000
3	0.000	0.000	0.071	0.286	0.000	0.000	0.643	0.000	0.000
4	0.155	0.018	0.009	0.727	0.055	0.009	0.018	0.000	0.009
5	0.154	0.026	0.000	0.103	0.308	0.103	0.308	0.000	0.000
6	0.114	0.114	0.000	0.068	0.045	0.591	0.068	0.000	0.000
7	0.000	0.157	0.007	0.007	0.013	0.007	0.810	0.000	0.000
8	0.000	0.000	0.000	0.333	0.000	0.000	0.000	0.333	0.333
9	0.000	0.000	0.000	0.000	0.167	0.000	0.167	0.000	0.667

4.2.3.Sample Query point -1

In [390]:
```
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 : 7
Actual Class : 6
The  11  nearest neighbours of the test points belongs to classes [6 6 6 6 6 6 6 6 6 2 2]
Fequency of nearest points : Counter({6: 9, 2: 2})
```

4.2.4. Sample Query Point-2

In [391]:
```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 100

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

```
Predicted Class : 7
Actual Class : 7
the k value for knn is 11 and the nearest neighbours of the test points belongs to classes [7 7 7 2 7 7 2 7 2 7
2]
Fequency of nearest points : Counter({7: 7, 2: 4})
```

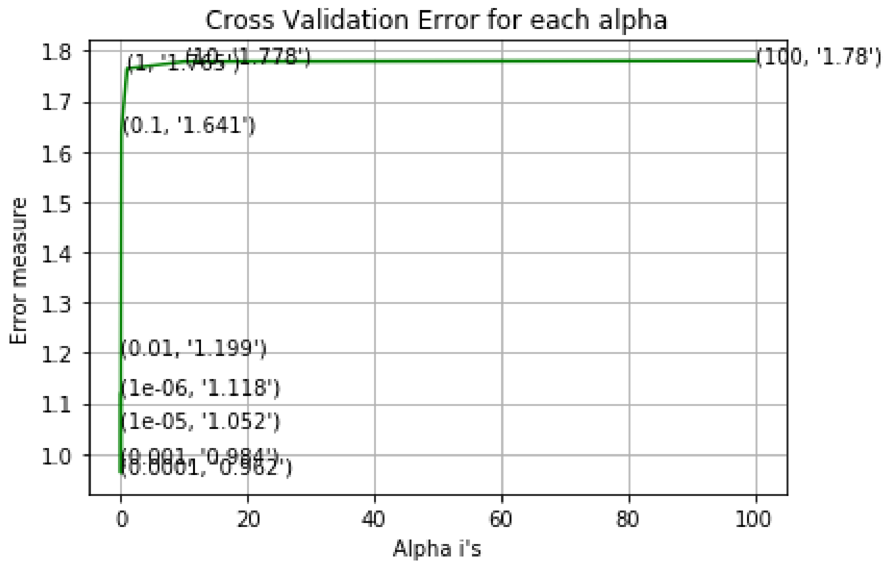
4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [392]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDCl
assifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=
None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=
0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



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

alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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



best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], 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, la
bels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predi
ct_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labe
ls=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.1182472225577926
for alpha = 1e-05
Log Loss : 1.0521355666324703
for alpha = 0.0001
Log Loss : 0.9618846222372404
for alpha = 0.001
Log Loss : 0.9840918639645416
for alpha = 0.01
Log Loss : 1.1986826801117376
for alpha = 0.1
Log Loss : 1.6409340149135736
for alpha = 1
Log Loss : 1.7646424412803479
for alpha = 10
Log Loss : 1.7784424009948525
for alpha = 100
Log Loss : 1.7800611591148183
```

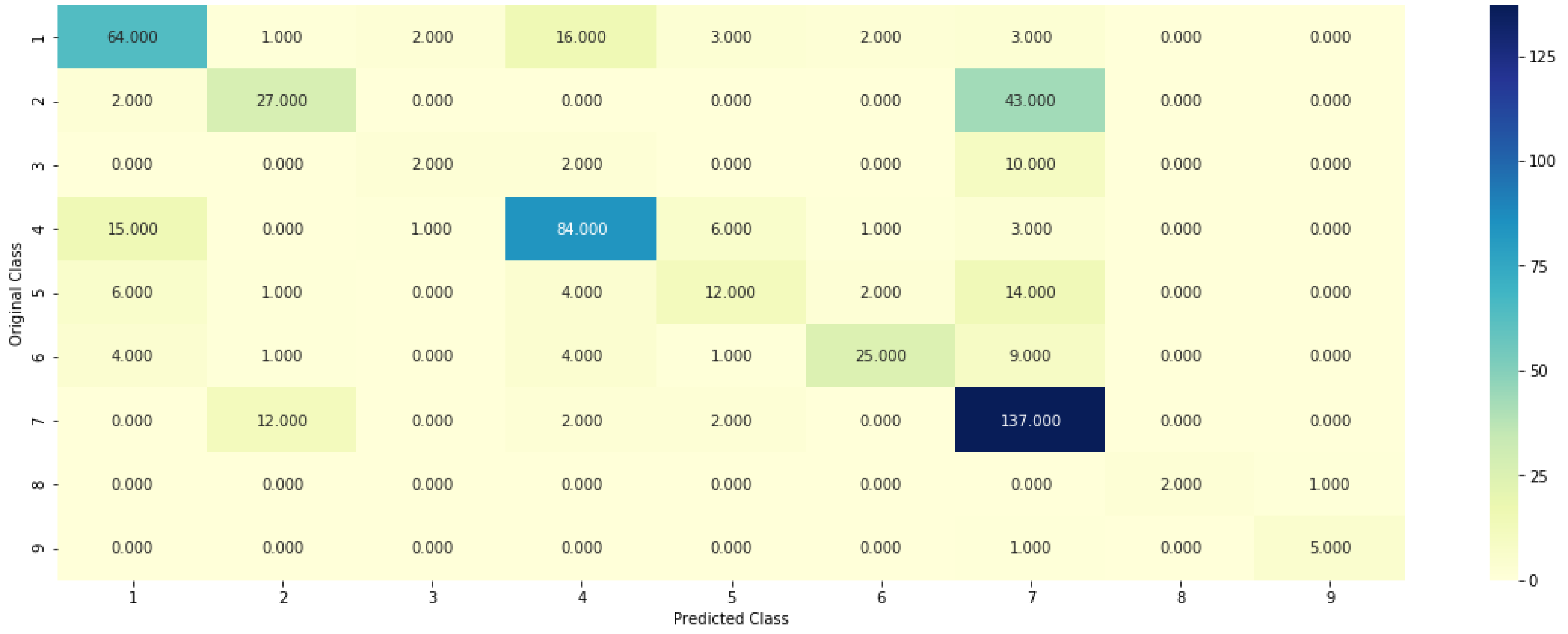


```
For values of best alpha =  0.0001 The train log loss is: 0.4641060292292926
For values of best alpha =  0.0001 The cross validation log loss is: 0.9618846222372404
For values of best alpha =  0.0001 The test log loss is: 0.9746716124635243
```

4.3.1.2. Testing the model with best hyper paramters

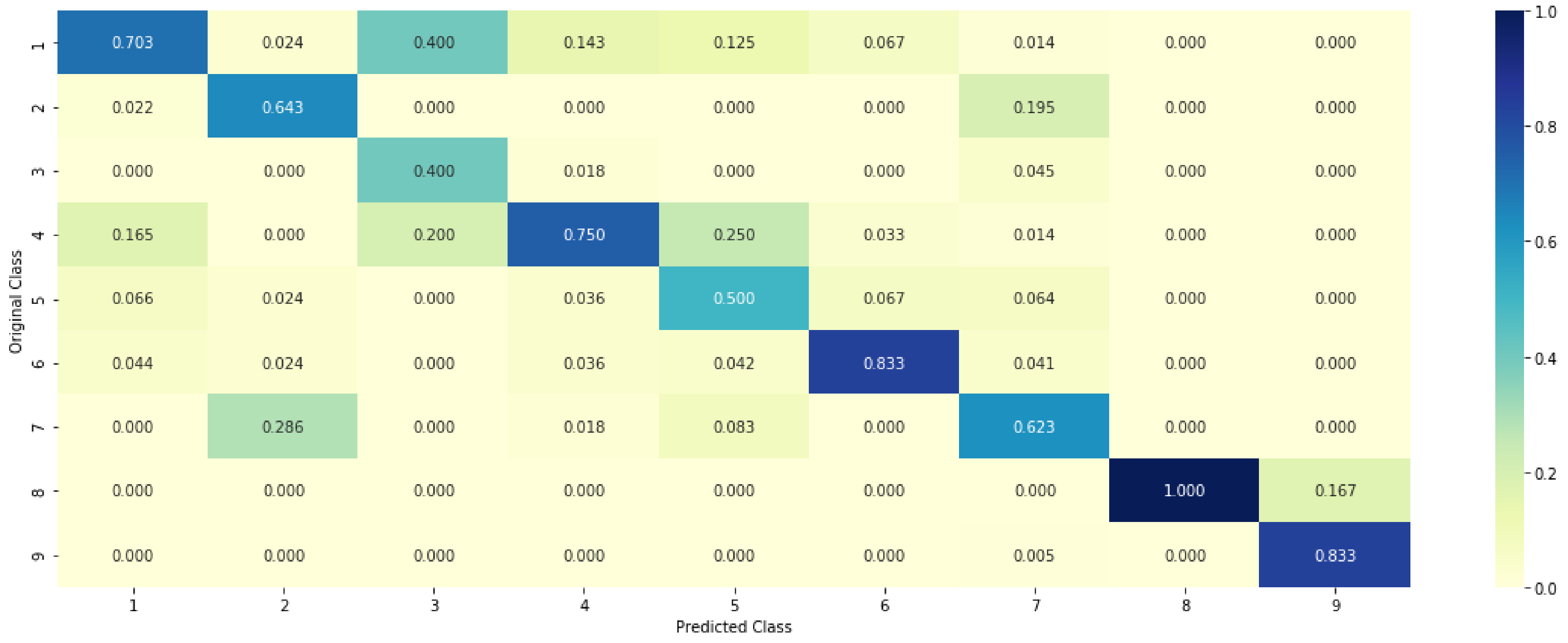
In [393]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDCl
assifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=
None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=
0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

```
Log loss : 0.9618846222372404
Number of mis-classified points : 0.32706766917293234
-------------------- Confusion matrix --------------------
```

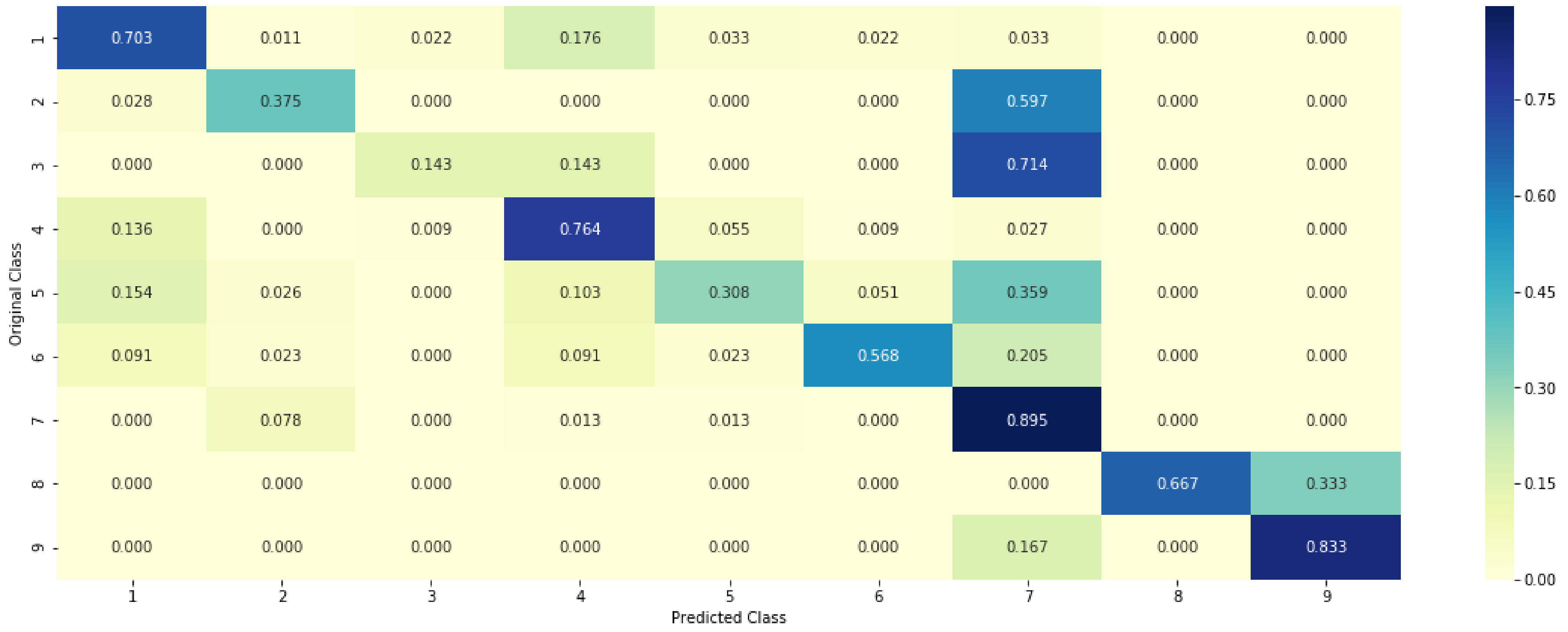


Original \ Predicted	1	2	3	4	5	6	7	8	9
1	64.000	1.000	2.000	16.000	3.000	2.000	3.000	0.000	0.000
2	2.000	27.000	0.000	0.000	0.000	0.000	43.000	0.000	0.000
3	0.000	0.000	2.000	2.000	0.000	0.000	10.000	0.000	0.000
4	15.000	0.000	1.000	84.000	6.000	1.000	3.000	0.000	0.000
5	6.000	1.000	0.000	4.000	12.000	2.000	14.000	0.000	0.000
6	4.000	1.000	0.000	4.000	1.000	25.000	9.000	0.000	0.000
7	0.000	12.000	0.000	2.000	2.000	0.000	137.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	2.000	1.000
9	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	5.000

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



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



4.3.1.3. Feature Importance

```
In [394]: def get_imp_feature_names(text, indices, removed_ind = []):
              word_present = 0
              tabulte_list = []
              incresingorder_ind = 0
              for i in indices:
                  if i < train_gene_feature_onehotCoding.shape[1]:
                      tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
                  elif i< 18:
                      tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
                  if ((i > 17) & (i not in removed_ind)) :
                      word = train_text_features[i]
                      yes_no = True if word in text.split() else False
                      if yes_no:
                          word_present += 1
                      tabulte_list.append([incresingorder_ind,train_text_features[i], yes_no])
                  incresingorder_ind += 1
              print(word_present, "most importent features are present in our query point")
              print("-"*50)
              print("The features that are most importent of the ",predicted_cls[0]," class:")
              print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or Not']))
```

4.3.1.3.1. Correctly Classified point

In [395]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4
))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],t
est_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0127 0.0458 0.0037 0.005  0.0053 0.7794 0.1418 0.0036 0.0028]]
Actual Class : 6
--------------------------------------------------
70 Text feature [blue] present in test data point [True]
90 Text feature [values] present in test data point [True]
95 Text feature [mek1] present in test data point [True]
105 Text feature [substrate] present in test data point [True]
110 Text feature [binds] present in test data point [True]
120 Text feature [right] present in test data point [True]
141 Text feature [concentration] present in test data point [True]
142 Text feature [substitutions] present in test data point [True]
146 Text feature [significant] present in test data point [True]
147 Text feature [interaction] present in test data point [True]
148 Text feature [suppression] present in test data point [True]
151 Text feature [resistance] present in test data point [True]
152 Text feature [free] present in test data point [True]
154 Text feature [conformation] present in test data point [True]
157 Text feature [substitution] present in test data point [True]
158 Text feature [expected] present in test data point [True]
161 Text feature [inhibition] present in test data point [True]
162 Text feature [site] present in test data point [True]
163 Text feature [helix] present in test data point [True]
164 Text feature [enzyme] present in test data point [True]
172 Text feature [distinct] present in test data point [True]
173 Text feature [inhibitors] present in test data point [True]
180 Text feature [family] present in test data point [True]
190 Text feature [low] present in test data point [True]
191 Text feature [loss] present in test data point [True]
192 Text feature [identified] present in test data point [True]
193 Text feature [predicted] present in test data point [True]
194 Text feature [showing] present in test data point [True]
196 Text feature [models] present in test data point [True]
198 Text feature [observation] present in test data point [True]
200 Text feature [active] present in test data point [True]
201 Text feature [ic50] present in test data point [True]
203 Text feature [selection] present in test data point [True]
205 Text feature [ability] present in test data point [True]
206 Text feature [decreased] present in test data point [True]
211 Text feature [atp] present in test data point [True]
```

```
212 Text feature [state] present in test data point [True]
213 Text feature [important] present in test data point [True]
214 Text feature [evidence] present in test data point [True]
216 Text feature [frequent] present in test data point [True]
217 Text feature [mek] present in test data point [True]
218 Text feature [model] present in test data point [True]
219 Text feature [phase] present in test data point [True]
221 Text feature [driver] present in test data point [True]
223 Text feature [cause] present in test data point [True]
226 Text feature [thus] present in test data point [True]
227 Text feature [gel] present in test data point [True]
236 Text feature [associated] present in test data point [True]
237 Text feature [development] present in test data point [True]
238 Text feature [mutagenesis] present in test data point [True]
239 Text feature [49] present in test data point [True]
246 Text feature [potential] present in test data point [True]
248 Text feature [43] present in test data point [True]
256 Text feature [nm] present in test data point [True]
260 Text feature [drug] present in test data point [True]
261 Text feature [affect] present in test data point [True]
275 Text feature [structural] present in test data point [True]
278 Text feature [lower] present in test data point [True]
279 Text feature [kinase] present in test data point [True]
281 Text feature [therapeutic] present in test data point [True]
286 Text feature [resistant] present in test data point [True]
289 Text feature [confer] present in test data point [True]
290 Text feature [acids] present in test data point [True]
292 Text feature [60] present in test data point [True]
294 Text feature [phospho] present in test data point [True]
297 Text feature [interactions] present in test data point [True]
299 Text feature [groups] present in test data point [True]
307 Text feature [type] present in test data point [True]
308 Text feature [40] present in test data point [True]
310 Text feature [time] present in test data point [True]
311 Text feature [due] present in test data point [True]
314 Text feature [46] present in test data point [True]
317 Text feature [factors] present in test data point [True]
326 Text feature [bound] present in test data point [True]
328 Text feature [single] present in test data point [True]
332 Text feature [bind] present in test data point [True]
```

```
336 Text feature [dose] present in test data point [True]
338 Text feature [regulated] present in test data point [True]
340 Text feature [clinically] present in test data point [True]
341 Text feature [greater] present in test data point [True]
344 Text feature [mechanism] present in test data point [True]
345 Text feature [second] present in test data point [True]
346 Text feature [20] present in test data point [True]
347 Text feature [inactive] present in test data point [True]
350 Text feature [value] present in test data point [True]
353 Text feature [34] present in test data point [True]
354 Text feature [regulation] present in test data point [True]
356 Text feature [demonstrated] present in test data point [True]
357 Text feature [residues] present in test data point [True]
358 Text feature [catalytic] present in test data point [True]
366 Text feature [variant] present in test data point [True]
374 Text feature [breast] present in test data point [True]
377 Text feature [four] present in test data point [True]
378 Text feature [relevant] present in test data point [True]
379 Text feature [survival] present in test data point [True]
380 Text feature [combination] present in test data point [True]
384 Text feature [targets] present in test data point [True]
389 Text feature [screening] present in test data point [True]
390 Text feature [group] present in test data point [True]
392 Text feature [well] present in test data point [True]
395 Text feature [leading] present in test data point [True]
396 Text feature [trials] present in test data point [True]
400 Text feature [proportion] present in test data point [True]
402 Text feature [region] present in test data point [True]
403 Text feature [studies] present in test data point [True]
404 Text feature [direct] present in test data point [True]
405 Text feature [determine] present in test data point [True]
410 Text feature [amino] present in test data point [True]
411 Text feature [factor] present in test data point [True]
412 Text feature [use] present in test data point [True]
414 Text feature [indeed] present in test data point [True]
417 Text feature [44] present in test data point [True]
420 Text feature [hydrophobic] present in test data point [True]
421 Text feature [acid] present in test data point [True]
423 Text feature [45] present in test data point [True]
426 Text feature [conserved] present in test data point [True]
```

```
429 Text feature [inhibitory] present in test data point [True]
432 Text feature [loop] present in test data point [True]
433 Text feature [members] present in test data point [True]
434 Text feature [required] present in test data point [True]
436 Text feature [proliferation] present in test data point [True]
438 Text feature [given] present in test data point [True]
439 Text feature [whether] present in test data point [True]
440 Text feature [concentrations] present in test data point [True]
448 Text feature [therapy] present in test data point [True]
450 Text feature [multiple] present in test data point [True]
453 Text feature [first] present in test data point [True]
455 Text feature [change] present in test data point [True]
457 Text feature [observations] present in test data point [True]
458 Text feature [since] present in test data point [True]
464 Text feature [red] present in test data point [True]
470 Text feature [targeting] present in test data point [True]
471 Text feature [affected] present in test data point [True]
472 Text feature [56] present in test data point [True]
473 Text feature [upon] present in test data point [True]
475 Text feature [fig] present in test data point [True]
476 Text feature [involved] present in test data point [True]
477 Text feature [src] present in test data point [True]
478 Text feature [targeted] present in test data point [True]
479 Text feature [side] present in test data point [True]
484 Text feature [1a] present in test data point [True]
486 Text feature [23] present in test data point [True]
489 Text feature [form] present in test data point [True]
490 Text feature [hydrogen] present in test data point [True]
491 Text feature [egfr] present in test data point [True]
493 Text feature [higher] present in test data point [True]
494 Text feature [receptor] present in test data point [True]
495 Text feature [sequencing] present in test data point [True]
497 Text feature [measured] present in test data point [True]
499 Text feature [binding] present in test data point [True]
Out of the top  500  features  150 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [396]:
```
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],t
est_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0124 0.0107 0.0021 0.0144 0.0178 0.0093 0.9296 0.0026 0.0011]]
Actual Class : 7
--------------------------------------------------
4 Text feature [downstream] present in test data point [True]
12 Text feature [activation] present in test data point [True]
16 Text feature [oncogene] present in test data point [True]
22 Text feature [3b] present in test data point [True]
33 Text feature [constitutive] present in test data point [True]
34 Text feature [factor] present in test data point [True]
40 Text feature [enhanced] present in test data point [True]
51 Text feature [activating] present in test data point [True]
54 Text feature [activated] present in test data point [True]
56 Text feature [transforming] present in test data point [True]
72 Text feature [leukemia] present in test data point [True]
74 Text feature [mapk] present in test data point [True]
92 Text feature [activate] present in test data point [True]
104 Text feature [pathways] present in test data point [True]
113 Text feature [61] present in test data point [True]
157 Text feature [lung] present in test data point [True]
160 Text feature [inhibited] present in test data point [True]
161 Text feature [epithelial] present in test data point [True]
163 Text feature [3t3] present in test data point [True]
198 Text feature [gtp] present in test data point [True]
219 Text feature [inhibitor] present in test data point [True]
222 Text feature [specimens] present in test data point [True]
238 Text feature [expressing] present in test data point [True]
244 Text feature [conditions] present in test data point [True]
259 Text feature [signaling] present in test data point [True]
264 Text feature [3a] present in test data point [True]
269 Text feature [total] present in test data point [True]
290 Text feature [occur] present in test data point [True]
296 Text feature [colony] present in test data point [True]
358 Text feature [increased] present in test data point [True]
362 Text feature [high] present in test data point [True]
409 Text feature [positive] present in test data point [True]
416 Text feature [per] present in test data point [True]
419 Text feature [presence] present in test data point [True]
435 Text feature [somatic] present in test data point [True]
476 Text feature [signals] present in test data point [True]
```

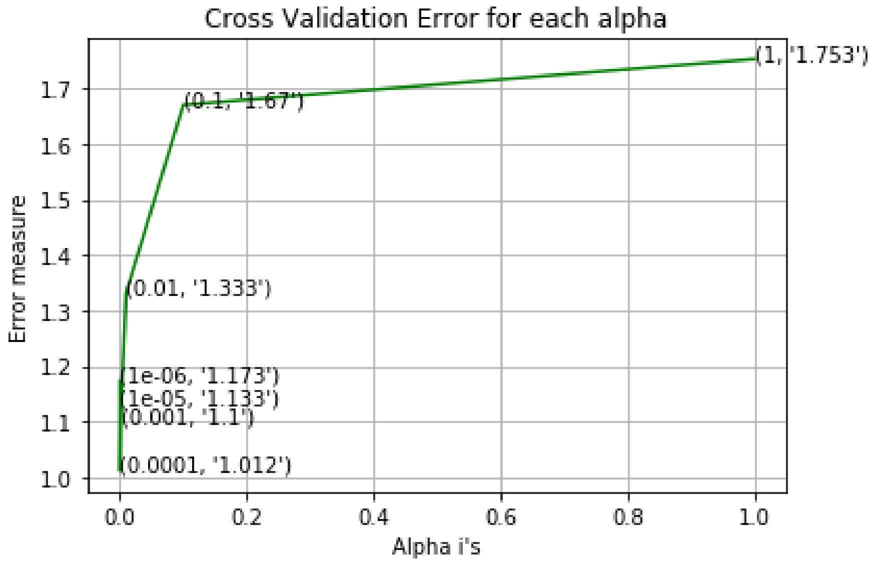
```
477 Text feature [recently] present in test data point [True]
495 Text feature [oncogenic] present in test data point [True]
498 Text feature [mechanism] present in test data point [True]
Out of the top  500  features  39 are present in query point
```

4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

In [397]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDCl
assifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=
None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=
0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



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

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.1726602891034956
for alpha = 1e-05
Log Loss : 1.132877605102943
for alpha = 0.0001
Log Loss : 1.0115297918022363
for alpha = 0.001
Log Loss : 1.1000078458811038
for alpha = 0.01
Log Loss : 1.3326497094076821
for alpha = 0.1
Log Loss : 1.670408060261667
for alpha = 1
Log Loss : 1.7526538749640215
```

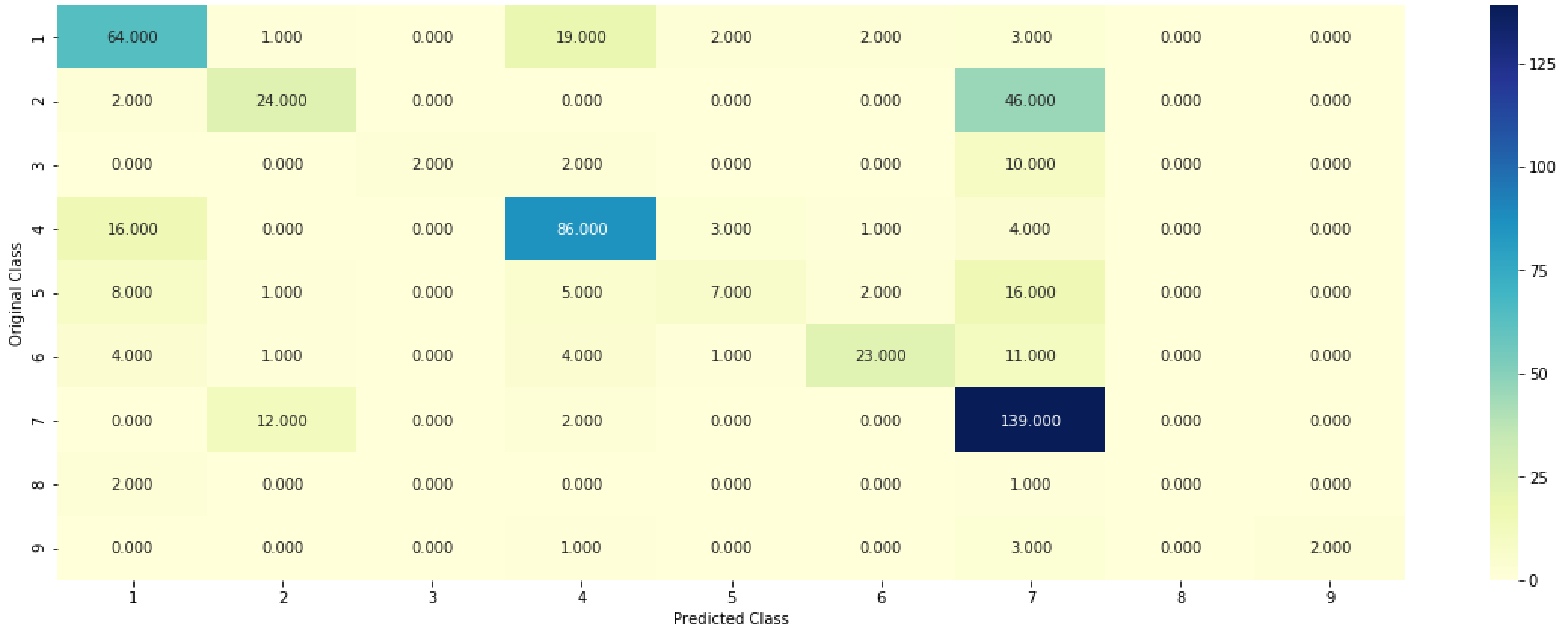


```
For values of best alpha =  0.0001 The train log loss is: 0.45275319705807254
For values of best alpha =  0.0001 The cross validation log loss is: 1.0115297918022363
For values of best alpha =  0.0001 The test log loss is: 1.0032592578024464
```

4.3.2.2. Testing model with best hyper parameters

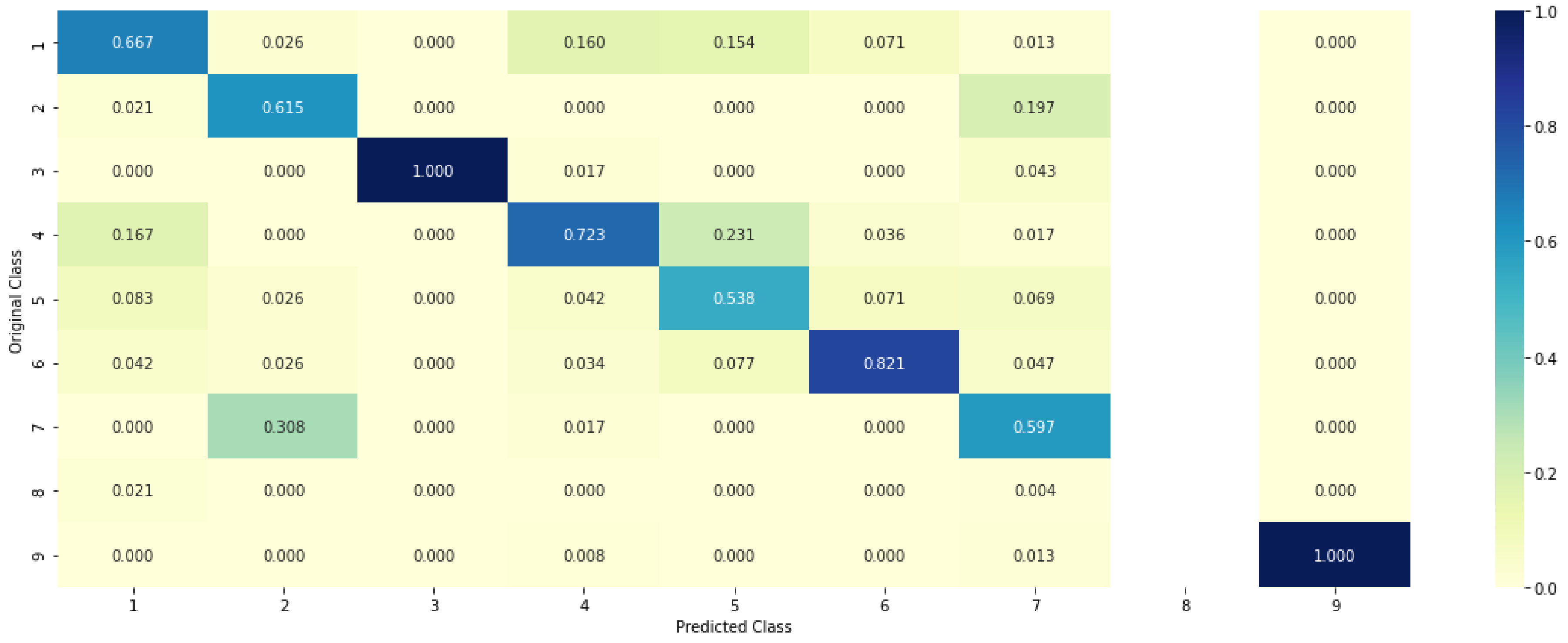
In [398]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDCl
assifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=
None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=
0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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 : 1.0115297918022363
Number of mis-classified points : 0.3477443609022556З
-------------------- Confusion matrix --------------------
```

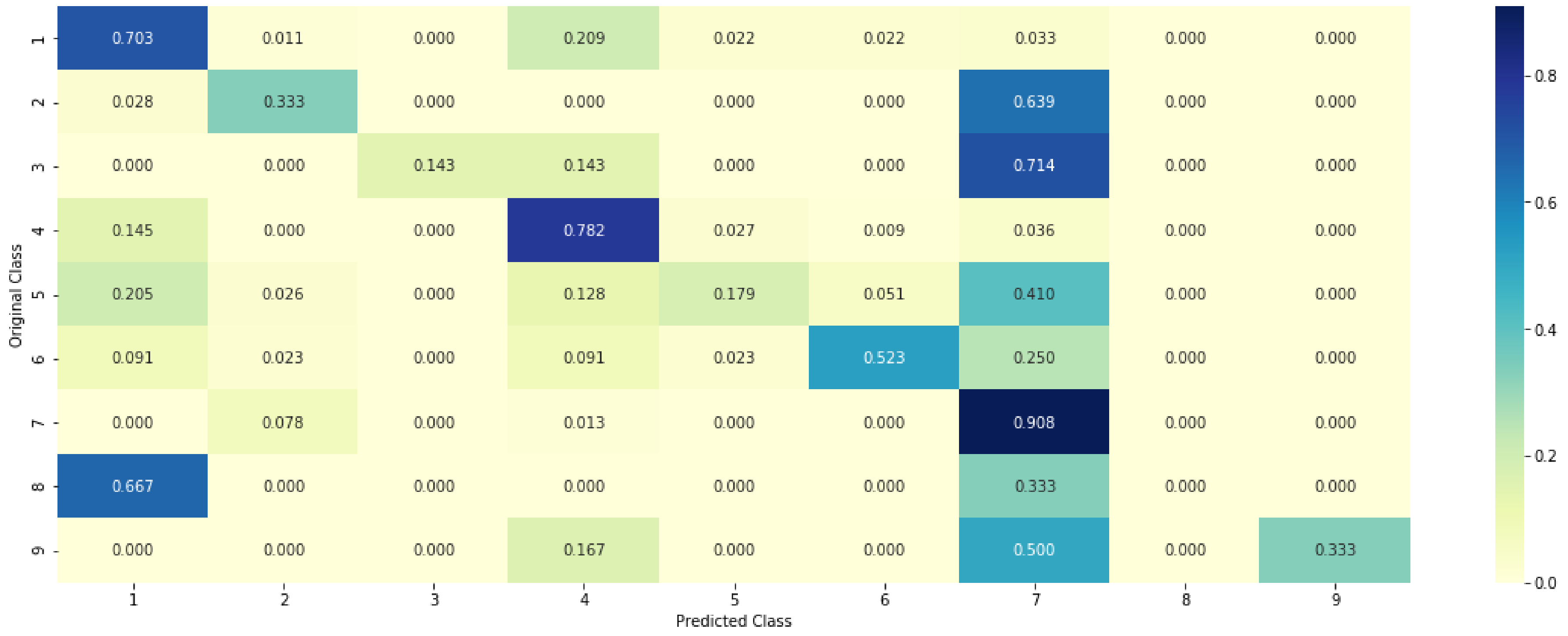


	1	2	3	4	5	6	7	8	9
1	64.000	1.000	0.000	19.000	2.000	2.000	3.000	0.000	0.000
2	2.000	24.000	0.000	0.000	0.000	0.000	46.000	0.000	0.000
3	0.000	0.000	2.000	2.000	0.000	0.000	10.000	0.000	0.000
4	16.000	0.000	0.000	86.000	3.000	1.000	4.000	0.000	0.000
5	8.000	1.000	0.000	5.000	7.000	2.000	16.000	0.000	0.000
6	4.000	1.000	0.000	4.000	1.000	23.000	11.000	0.000	0.000
7	0.000	12.000	0.000	2.000	0.000	0.000	139.000	0.000	0.000
8	2.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000
9	0.000	0.000	0.000	1.000	0.000	0.000	3.000	0.000	2.000

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



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



4.3.2.3. Feature Importance, Correctly Classified point

```
In [399]: 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],t
          est_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0145 0.0476 0.0025 0.0059 0.0048 0.7827 0.1372 0.0039 0.0009]]
Actual Class : 6
--------------------------------------------------
73 Text feature [blue] present in test data point [True]
89 Text feature [values] present in test data point [True]
94 Text feature [mek1] present in test data point [True]
99 Text feature [binds] present in test data point [True]
104 Text feature [substrate] present in test data point [True]
120 Text feature [right] present in test data point [True]
136 Text feature [concentration] present in test data point [True]
137 Text feature [substitutions] present in test data point [True]
140 Text feature [suppression] present in test data point [True]
144 Text feature [interaction] present in test data point [True]
145 Text feature [conformation] present in test data point [True]
146 Text feature [resistance] present in test data point [True]
149 Text feature [significant] present in test data point [True]
155 Text feature [substitution] present in test data point [True]
156 Text feature [expected] present in test data point [True]
159 Text feature [enzyme] present in test data point [True]
160 Text feature [free] present in test data point [True]
162 Text feature [site] present in test data point [True]
163 Text feature [helix] present in test data point [True]
165 Text feature [inhibition] present in test data point [True]
175 Text feature [inhibitors] present in test data point [True]
180 Text feature [distinct] present in test data point [True]
183 Text feature [family] present in test data point [True]
187 Text feature [predicted] present in test data point [True]
189 Text feature [models] present in test data point [True]
193 Text feature [low] present in test data point [True]
197 Text feature [state] present in test data point [True]
198 Text feature [identified] present in test data point [True]
200 Text feature [ic50] present in test data point [True]
201 Text feature [loss] present in test data point [True]
202 Text feature [showing] present in test data point [True]
203 Text feature [observation] present in test data point [True]
205 Text feature [active] present in test data point [True]
211 Text feature [ability] present in test data point [True]
212 Text feature [atp] present in test data point [True]
213 Text feature [selection] present in test data point [True]
```

```
214 Text feature [evidence] present in test data point [True]
218 Text feature [49] present in test data point [True]
222 Text feature [cause] present in test data point [True]
223 Text feature [mek] present in test data point [True]
224 Text feature [gel] present in test data point [True]
225 Text feature [model] present in test data point [True]
226 Text feature [important] present in test data point [True]
229 Text feature [phase] present in test data point [True]
232 Text feature [frequent] present in test data point [True]
233 Text feature [decreased] present in test data point [True]
236 Text feature [thus] present in test data point [True]
237 Text feature [development] present in test data point [True]
243 Text feature [driver] present in test data point [True]
246 Text feature [43] present in test data point [True]
252 Text feature [potential] present in test data point [True]
253 Text feature [associated] present in test data point [True]
257 Text feature [drug] present in test data point [True]
258 Text feature [nm] present in test data point [True]
259 Text feature [mutagenesis] present in test data point [True]
261 Text feature [affect] present in test data point [True]
273 Text feature [lower] present in test data point [True]
277 Text feature [resistant] present in test data point [True]
279 Text feature [structural] present in test data point [True]
283 Text feature [interactions] present in test data point [True]
286 Text feature [confer] present in test data point [True]
288 Text feature [therapeutic] present in test data point [True]
290 Text feature [kinase] present in test data point [True]
294 Text feature [acids] present in test data point [True]
303 Text feature [60] present in test data point [True]
304 Text feature [phospho] present in test data point [True]
307 Text feature [due] present in test data point [True]
308 Text feature [groups] present in test data point [True]
310 Text feature [40] present in test data point [True]
312 Text feature [time] present in test data point [True]
313 Text feature [46] present in test data point [True]
317 Text feature [factors] present in test data point [True]
318 Text feature [single] present in test data point [True]
320 Text feature [bind] present in test data point [True]
323 Text feature [dose] present in test data point [True]
328 Text feature [type] present in test data point [True]
```

```
329 Text feature [bound] present in test data point [True]
338 Text feature [greater] present in test data point [True]
344 Text feature [inactive] present in test data point [True]
346 Text feature [breast] present in test data point [True]
350 Text feature [clinically] present in test data point [True]
351 Text feature [second] present in test data point [True]
354 Text feature [20] present in test data point [True]
355 Text feature [mechanism] present in test data point [True]
356 Text feature [catalytic] present in test data point [True]
358 Text feature [regulation] present in test data point [True]
360 Text feature [value] present in test data point [True]
361 Text feature [residues] present in test data point [True]
364 Text feature [regulated] present in test data point [True]
365 Text feature [variant] present in test data point [True]
367 Text feature [leading] present in test data point [True]
370 Text feature [34] present in test data point [True]
372 Text feature [group] present in test data point [True]
374 Text feature [demonstrated] present in test data point [True]
380 Text feature [targets] present in test data point [True]
382 Text feature [trials] present in test data point [True]
383 Text feature [combination] present in test data point [True]
384 Text feature [well] present in test data point [True]
386 Text feature [region] present in test data point [True]
389 Text feature [44] present in test data point [True]
391 Text feature [four] present in test data point [True]
393 Text feature [screening] present in test data point [True]
398 Text feature [hydrophobic] present in test data point [True]
400 Text feature [amino] present in test data point [True]
401 Text feature [direct] present in test data point [True]
402 Text feature [relevant] present in test data point [True]
405 Text feature [use] present in test data point [True]
406 Text feature [determine] present in test data point [True]
407 Text feature [proportion] present in test data point [True]
409 Text feature [studies] present in test data point [True]
412 Text feature [factor] present in test data point [True]
413 Text feature [therapy] present in test data point [True]
416 Text feature [acid] present in test data point [True]
418 Text feature [members] present in test data point [True]
420 Text feature [45] present in test data point [True]
422 Text feature [survival] present in test data point [True]
```

```
429 Text feature [loop] present in test data point [True]
430 Text feature [concentrations] present in test data point [True]
432 Text feature [given] present in test data point [True]
433 Text feature [proliferation] present in test data point [True]
434 Text feature [inhibitory] present in test data point [True]
435 Text feature [conserved] present in test data point [True]
437 Text feature [required] present in test data point [True]
439 Text feature [indeed] present in test data point [True]
441 Text feature [since] present in test data point [True]
446 Text feature [56] present in test data point [True]
447 Text feature [affected] present in test data point [True]
450 Text feature [red] present in test data point [True]
453 Text feature [targeted] present in test data point [True]
454 Text feature [targeting] present in test data point [True]
455 Text feature [involved] present in test data point [True]
459 Text feature [whether] present in test data point [True]
461 Text feature [multiple] present in test data point [True]
463 Text feature [first] present in test data point [True]
466 Text feature [side] present in test data point [True]
467 Text feature [change] present in test data point [True]
468 Text feature [src] present in test data point [True]
469 Text feature [observations] present in test data point [True]
472 Text feature [seven] present in test data point [True]
478 Text feature [hydrogen] present in test data point [True]
479 Text feature [upon] present in test data point [True]
483 Text feature [form] present in test data point [True]
486 Text feature [receptor] present in test data point [True]
488 Text feature [image] present in test data point [True]
489 Text feature [binding] present in test data point [True]
491 Text feature [biochemical] present in test data point [True]
493 Text feature [mice] present in test data point [True]
494 Text feature [23] present in test data point [True]
498 Text feature [fig] present in test data point [True]
499 Text feature [set] present in test data point [True]
Out of the top  500  features  150 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [400]:
```
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],t
est_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[1.320e-02 9.700e-03 1.900e-03 1.470e-02 1.390e-02 8.500e-03 9.344e-01
  3.400e-03 3.000e-04]]
Actual Class : 7
--------------------------------------------------
5 Text feature [downstream] present in test data point [True]
17 Text feature [activation] present in test data point [True]
22 Text feature [3b] present in test data point [True]
25 Text feature [oncogene] present in test data point [True]
40 Text feature [factor] present in test data point [True]
73 Text feature [activating] present in test data point [True]
75 Text feature [enhanced] present in test data point [True]
91 Text feature [constitutive] present in test data point [True]
96 Text feature [activated] present in test data point [True]
104 Text feature [activate] present in test data point [True]
142 Text feature [transforming] present in test data point [True]
147 Text feature [leukemia] present in test data point [True]
148 Text feature [pathways] present in test data point [True]
183 Text feature [mapk] present in test data point [True]
202 Text feature [lung] present in test data point [True]
211 Text feature [epithelial] present in test data point [True]
215 Text feature [inhibited] present in test data point [True]
223 Text feature [inhibitor] present in test data point [True]
245 Text feature [61] present in test data point [True]
276 Text feature [3t3] present in test data point [True]
294 Text feature [specimens] present in test data point [True]
296 Text feature [gtp] present in test data point [True]
314 Text feature [3a] present in test data point [True]
320 Text feature [occur] present in test data point [True]
325 Text feature [conditions] present in test data point [True]
358 Text feature [free] present in test data point [True]
360 Text feature [total] present in test data point [True]
366 Text feature [colony] present in test data point [True]
369 Text feature [recently] present in test data point [True]
373 Text feature [increased] present in test data point [True]
379 Text feature [signaling] present in test data point [True]
380 Text feature [expressing] present in test data point [True]
424 Text feature [somatic] present in test data point [True]
427 Text feature [presence] present in test data point [True]
432 Text feature [positive] present in test data point [True]
```

```
437 Text feature [high] present in test data point [True]
446 Text feature [signals] present in test data point [True]
470 Text feature [mechanism] present in test data point [True]
474 Text feature [tumor] present in test data point [True]
478 Text feature [sites] present in test data point [True]
482 Text feature [s3] present in test data point [True]
484 Text feature [per] present in test data point [True]
492 Text feature [position] present in test data point [True]
Out of the top  500  features  43 are present in query point
```

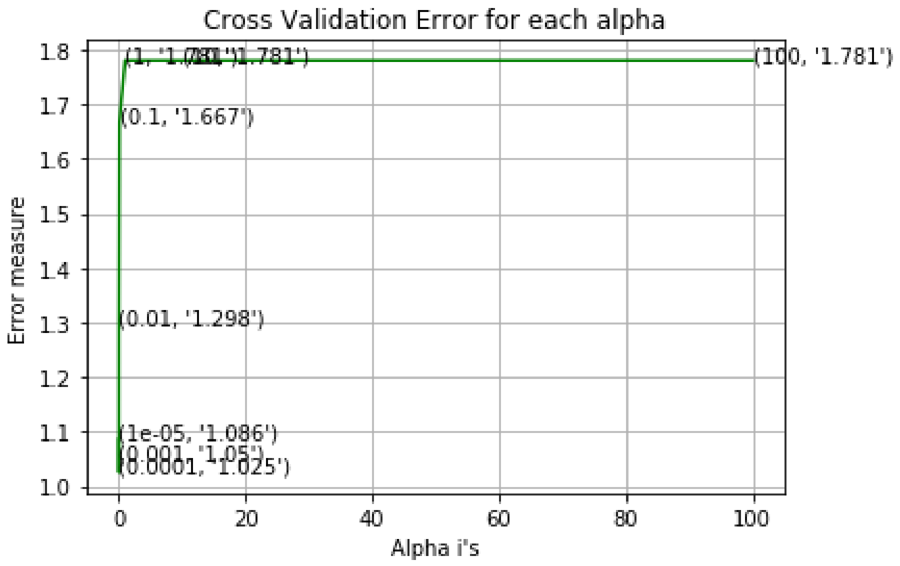
4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [401]:
```
# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/genera
ted/sklearn.svm.SVC.html

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

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



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

alpha = [10 ** x for x in range(-5, 3)]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
#     clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
```

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

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


best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=4
2)
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, la
bels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predi
ct_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labe
ls=clf.classes_, eps=1e-15))
```

```
for C = 1e-05
Log Loss : 1.0861904473724304
for C = 0.0001
Log Loss : 1.0251910291519057
for C = 0.001
Log Loss : 1.05027377742702
for C = 0.01
Log Loss : 1.2979655941285495
for C = 0.1
Log Loss : 1.6673270463087346
for C = 1
Log Loss : 1.7805196029669421
for C = 10
Log Loss : 1.7805195084549206
for C = 100
Log Loss : 1.7805196140020976
```

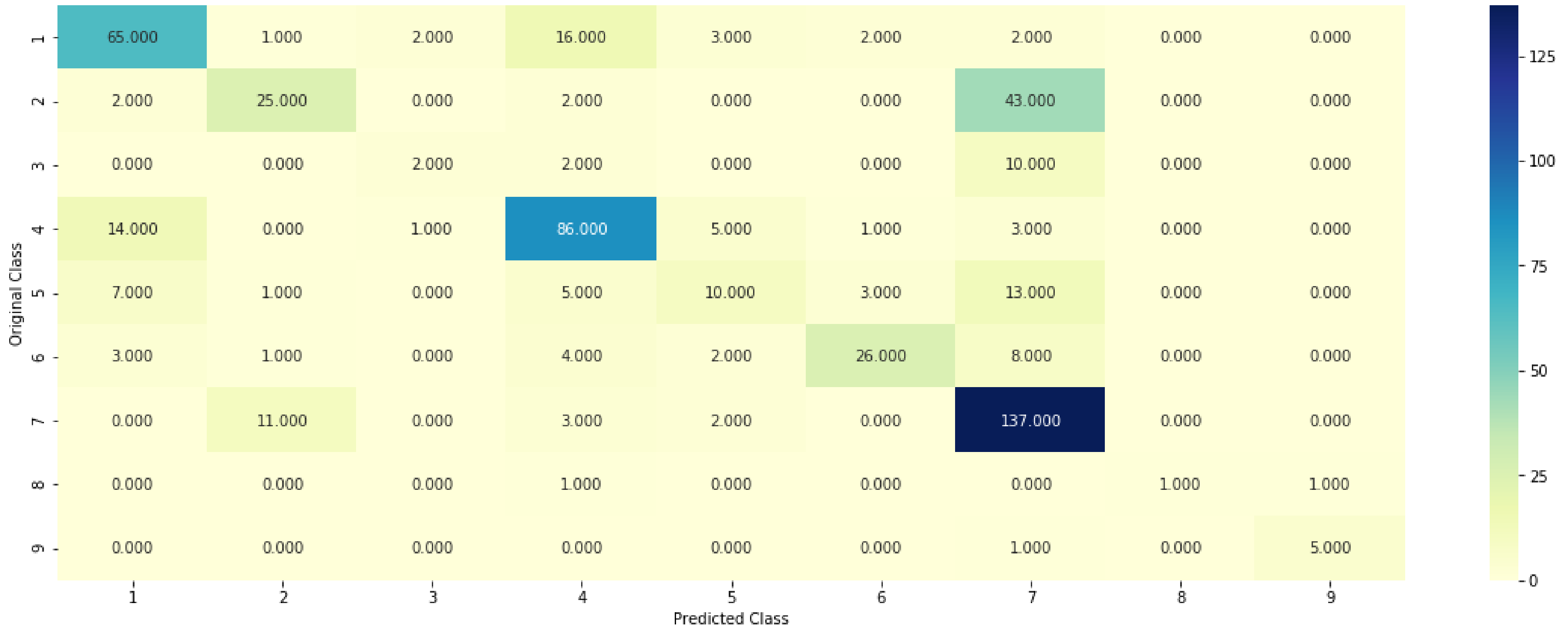


```
For values of best alpha =  0.0001 The train log loss is: 0.48108557634784666
For values of best alpha =  0.0001 The cross validation log loss is: 1.0251910291519057
For values of best alpha =  0.0001 The test log loss is: 1.0088254967787396
```

4.4.2. Testing model with best hyper parameters

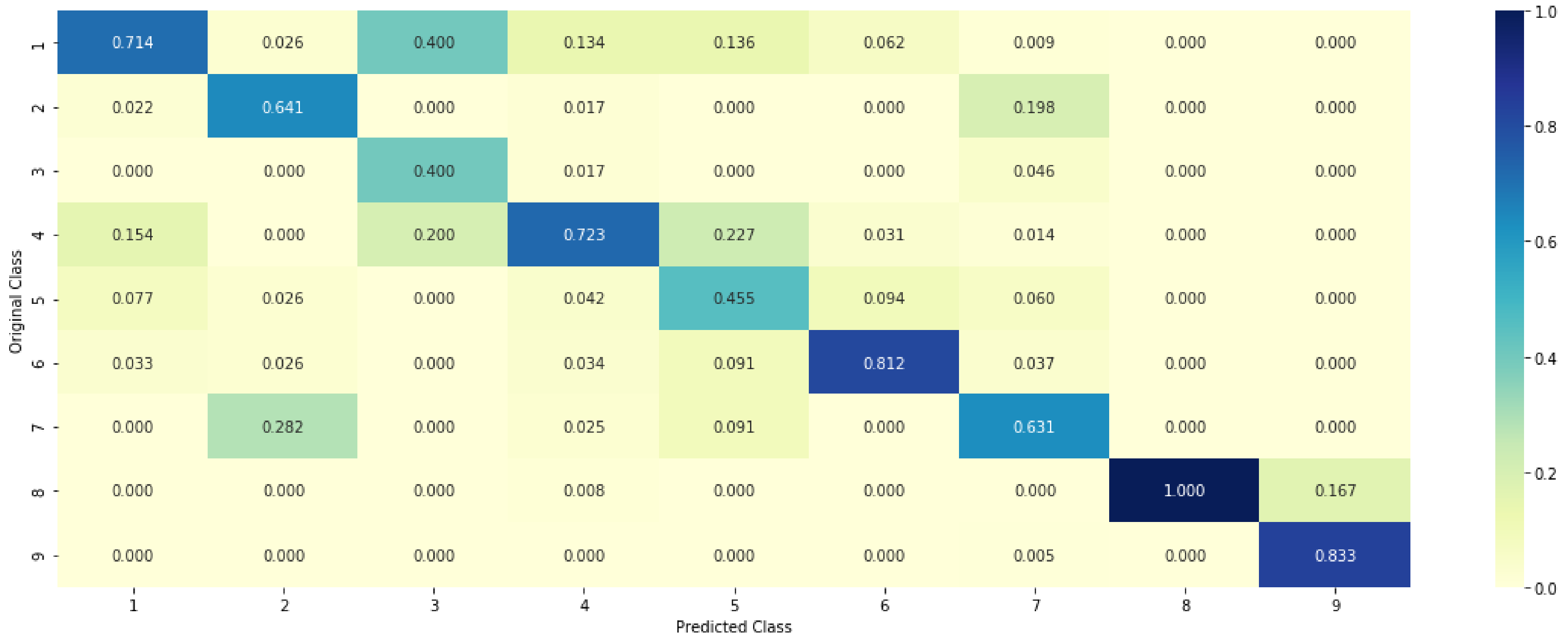
In [402]:
```
# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/genera
ted/sklearn.svm.SVC.html

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

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



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

```
Log loss : 1.0251910291519057
Number of mis-classified points : 0.32894736842105265
-------------------- Confusion matrix --------------------
```

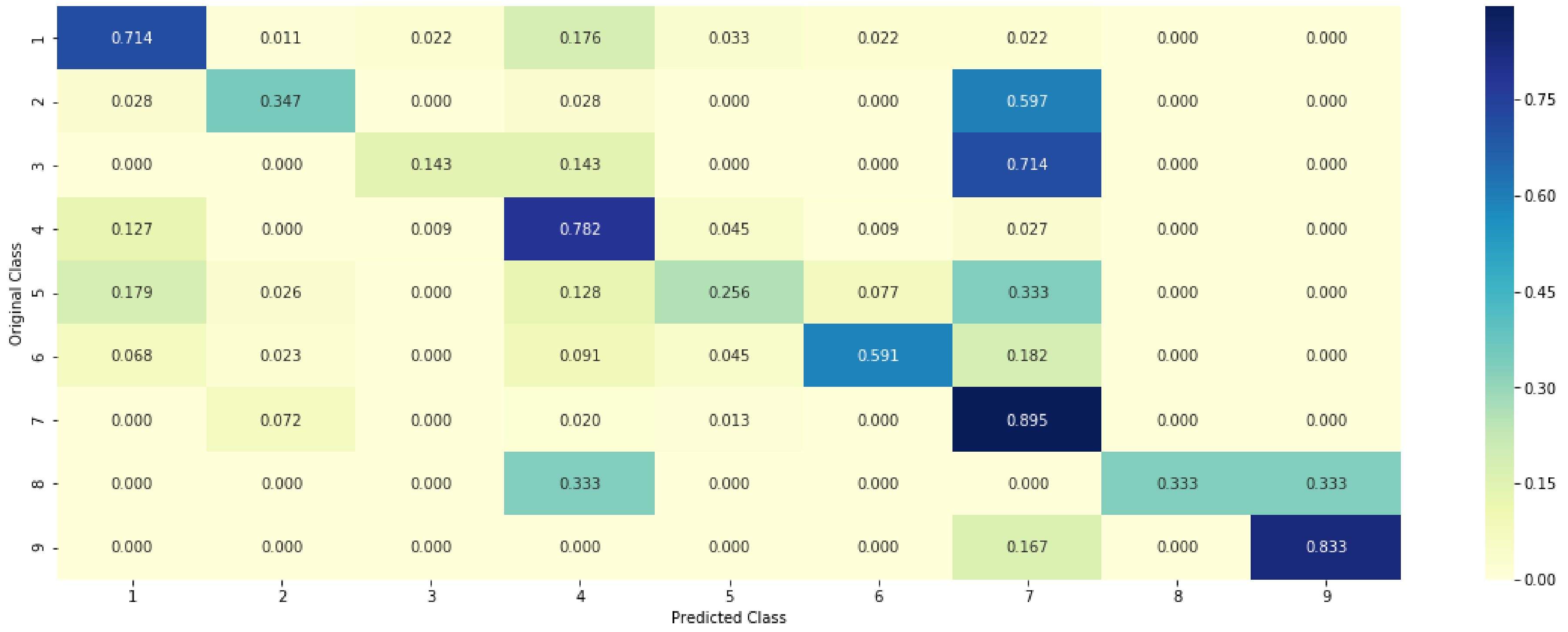


	1	2	3	4	5	6	7	8	9
1	65.000	1.000	2.000	16.000	3.000	2.000	2.000	0.000	0.000
2	2.000	25.000	0.000	2.000	0.000	0.000	43.000	0.000	0.000
3	0.000	0.000	2.000	2.000	0.000	0.000	10.000	0.000	0.000
4	14.000	0.000	1.000	86.000	5.000	1.000	3.000	0.000	0.000
5	7.000	1.000	0.000	5.000	10.000	3.000	13.000	0.000	0.000
6	3.000	1.000	0.000	4.000	2.000	26.000	8.000	0.000	0.000
7	0.000	11.000	0.000	3.000	2.000	0.000	137.000	0.000	0.000
8	0.000	0.000	0.000	1.000	0.000	0.000	0.000	1.000	1.000
9	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	5.000

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



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



	1	2	3	4	5	6	7	8	9
1	0.714	0.011	0.022	0.176	0.033	0.022	0.022	0.000	0.000
2	0.028	0.347	0.000	0.028	0.000	0.000	0.597	0.000	0.000
3	0.000	0.000	0.143	0.143	0.000	0.000	0.714	0.000	0.000
4	0.127	0.000	0.009	0.782	0.045	0.009	0.027	0.000	0.000
5	0.179	0.026	0.000	0.128	0.256	0.077	0.333	0.000	0.000
6	0.068	0.023	0.000	0.091	0.045	0.591	0.182	0.000	0.000
7	0.000	0.072	0.000	0.020	0.013	0.000	0.895	0.000	0.000
8	0.000	0.000	0.000	0.333	0.000	0.000	0.000	0.333	0.333
9	0.000	0.000	0.000	0.000	0.000	0.000	0.167	0.000	0.833

4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [403]:
```
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],t
est_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.035  0.0483 0.0097 0.0235 0.0157 0.7499 0.1127 0.0029 0.0025]]
Actual Class : 6
--------------------------------------------------
56 Text feature [values] present in test data point [True]
57 Text feature [mek1] present in test data point [True]
111 Text feature [blue] present in test data point [True]
115 Text feature [substrate] present in test data point [True]
117 Text feature [binds] present in test data point [True]
118 Text feature [substitutions] present in test data point [True]
120 Text feature [right] present in test data point [True]
123 Text feature [significant] present in test data point [True]
124 Text feature [expected] present in test data point [True]
126 Text feature [site] present in test data point [True]
127 Text feature [observation] present in test data point [True]
128 Text feature [interaction] present in test data point [True]
129 Text feature [substitution] present in test data point [True]
130 Text feature [conformation] present in test data point [True]
131 Text feature [suppression] present in test data point [True]
132 Text feature [phase] present in test data point [True]
136 Text feature [mutagenesis] present in test data point [True]
137 Text feature [inhibitors] present in test data point [True]
139 Text feature [selection] present in test data point [True]
141 Text feature [models] present in test data point [True]
142 Text feature [frequent] present in test data point [True]
143 Text feature [resistance] present in test data point [True]
145 Text feature [concentration] present in test data point [True]
146 Text feature [showing] present in test data point [True]
147 Text feature [gel] present in test data point [True]
223 Text feature [acids] present in test data point [True]
224 Text feature [distinct] present in test data point [True]
226 Text feature [phospho] present in test data point [True]
227 Text feature [atp] present in test data point [True]
228 Text feature [inhibition] present in test data point [True]
229 Text feature [model] present in test data point [True]
230 Text feature [ic50] present in test data point [True]
232 Text feature [mek] present in test data point [True]
234 Text feature [identified] present in test data point [True]
235 Text feature [low] present in test data point [True]
238 Text feature [helix] present in test data point [True]
```

```
239 Text feature [predicted] present in test data point [True]
240 Text feature [ability] present in test data point [True]
243 Text feature [family] present in test data point [True]
245 Text feature [development] present in test data point [True]
246 Text feature [trials] present in test data point [True]
248 Text feature [free] present in test data point [True]
250 Text feature [thus] present in test data point [True]
251 Text feature [driver] present in test data point [True]
255 Text feature [indeed] present in test data point [True]
260 Text feature [cause] present in test data point [True]
263 Text feature [acid] present in test data point [True]
266 Text feature [amino] present in test data point [True]
268 Text feature [drug] present in test data point [True]
270 Text feature [lower] present in test data point [True]
272 Text feature [enzyme] present in test data point [True]
273 Text feature [associated] present in test data point [True]
274 Text feature [resistant] present in test data point [True]
275 Text feature [combination] present in test data point [True]
277 Text feature [direct] present in test data point [True]
279 Text feature [type] present in test data point [True]
280 Text feature [potential] present in test data point [True]
282 Text feature [loss] present in test data point [True]
283 Text feature [observations] present in test data point [True]
285 Text feature [screening] present in test data point [True]
287 Text feature [therapeutic] present in test data point [True]
292 Text feature [decreased] present in test data point [True]
293 Text feature [kinase] present in test data point [True]
295 Text feature [confer] present in test data point [True]
307 Text feature [single] present in test data point [True]
310 Text feature [proportion] present in test data point [True]
314 Text feature [relevant] present in test data point [True]
315 Text feature [leading] present in test data point [True]
317 Text feature [active] present in test data point [True]
318 Text feature [factors] present in test data point [True]
320 Text feature [interactions] present in test data point [True]
321 Text feature [nm] present in test data point [True]
323 Text feature [since] present in test data point [True]
324 Text feature [affect] present in test data point [True]
327 Text feature [targets] present in test data point [True]
328 Text feature [evidence] present in test data point [True]
```

```
329 Text feature [important] present in test data point [True]
331 Text feature [groups] present in test data point [True]
338 Text feature [whether] present in test data point [True]
339 Text feature [34] present in test data point [True]
340 Text feature [20] present in test data point [True]
341 Text feature [studies] present in test data point [True]
344 Text feature [bind] present in test data point [True]
345 Text feature [phosphorylation] present in test data point [True]
348 Text feature [group] present in test data point [True]
350 Text feature [mutation] present in test data point [True]
351 Text feature [46] present in test data point [True]
353 Text feature [43] present in test data point [True]
356 Text feature [60] present in test data point [True]
359 Text feature [dose] present in test data point [True]
362 Text feature [due] present in test data point [True]
363 Text feature [mice] present in test data point [True]
364 Text feature [use] present in test data point [True]
366 Text feature [catalytic] present in test data point [True]
367 Text feature [image] present in test data point [True]
369 Text feature [bound] present in test data point [True]
376 Text feature [frequency] present in test data point [True]
377 Text feature [value] present in test data point [True]
378 Text feature [mechanism] present in test data point [True]
379 Text feature [different] present in test data point [True]
380 Text feature [loop] present in test data point [True]
382 Text feature [49] present in test data point [True]
386 Text feature [motif] present in test data point [True]
388 Text feature [clinically] present in test data point [True]
390 Text feature [required] present in test data point [True]
392 Text feature [first] present in test data point [True]
393 Text feature [demonstrated] present in test data point [True]
394 Text feature [factor] present in test data point [True]
399 Text feature [greater] present in test data point [True]
400 Text feature [100] present in test data point [True]
403 Text feature [unable] present in test data point [True]
406 Text feature [15] present in test data point [True]
407 Text feature [1a] present in test data point [True]
408 Text feature [induced] present in test data point [True]
410 Text feature [four] present in test data point [True]
411 Text feature [mediated] present in test data point [True]
```

```
413 Text feature [panel] present in test data point [True]
414 Text feature [survival] present in test data point [True]
416 Text feature [well] present in test data point [True]
417 Text feature [variant] present in test data point [True]
418 Text feature [open] present in test data point [True]
421 Text feature [binding] present in test data point [True]
422 Text feature [4b] present in test data point [True]
425 Text feature [normal] present in test data point [True]
426 Text feature [relatively] present in test data point [True]
429 Text feature [inhibitory] present in test data point [True]
430 Text feature [regulated] present in test data point [True]
431 Text feature [days] present in test data point [True]
432 Text feature [higher] present in test data point [True]
435 Text feature [breast] present in test data point [True]
437 Text feature [inactive] present in test data point [True]
438 Text feature [suggests] present in test data point [True]
439 Text feature [45] present in test data point [True]
440 Text feature [structural] present in test data point [True]
441 Text feature [state] present in test data point [True]
442 Text feature [dependent] present in test data point [True]
443 Text feature [fig] present in test data point [True]
445 Text feature [residues] present in test data point [True]
446 Text feature [egfr] present in test data point [True]
447 Text feature [may] present in test data point [True]
448 Text feature [receptor] present in test data point [True]
450 Text feature [second] present in test data point [True]
451 Text feature [mutant] present in test data point [True]
452 Text feature [time] present in test data point [True]
456 Text feature [general] present in test data point [True]
457 Text feature [also] present in test data point [True]
460 Text feature [purified] present in test data point [True]
461 Text feature [without] present in test data point [True]
464 Text feature [either] present in test data point [True]
467 Text feature [red] present in test data point [True]
468 Text feature [targeted] present in test data point [True]
472 Text feature [probe] present in test data point [True]
473 Text feature [determine] present in test data point [True]
474 Text feature [even] present in test data point [True]
475 Text feature [developed] present in test data point [True]
477 Text feature [proliferation] present in test data point [True]
```

```
479 Text feature [multiple] present in test data point [True]
480 Text feature [10] present in test data point [True]
484 Text feature [carried] present in test data point [True]
485 Text feature [clones] present in test data point [True]
487 Text feature [44] present in test data point [True]
490 Text feature [side] present in test data point [True]
493 Text feature [cell] present in test data point [True]
494 Text feature [phosphorylated] present in test data point [True]
495 Text feature [affected] present in test data point [True]
496 Text feature [discussion] present in test data point [True]
497 Text feature [finally] present in test data point [True]
498 Text feature [two] present in test data point [True]
Out of the top  500  features  168 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [404]:
```
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],t
est_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0183 0.0066 0.0046 0.0375 0.0857 0.0179 0.8259 0.0022 0.0013]]
Actual Class : 7
--------------------------------------------------
26 Text feature [downstream] present in test data point [True]
29 Text feature [3b] present in test data point [True]
35 Text feature [oncogene] present in test data point [True]
45 Text feature [activation] present in test data point [True]
233 Text feature [activate] present in test data point [True]
234 Text feature [lung] present in test data point [True]
238 Text feature [factor] present in test data point [True]
239 Text feature [enhanced] present in test data point [True]
241 Text feature [conditions] present in test data point [True]
243 Text feature [high] present in test data point [True]
244 Text feature [colony] present in test data point [True]
251 Text feature [epithelial] present in test data point [True]
255 Text feature [activating] present in test data point [True]
262 Text feature [61] present in test data point [True]
266 Text feature [free] present in test data point [True]
271 Text feature [pathways] present in test data point [True]
272 Text feature [constitutive] present in test data point [True]
281 Text feature [3t3] present in test data point [True]
282 Text feature [inhibited] present in test data point [True]
287 Text feature [3a] present in test data point [True]
290 Text feature [tumor] present in test data point [True]
291 Text feature [leukemia] present in test data point [True]
294 Text feature [22] present in test data point [True]
295 Text feature [gtp] present in test data point [True]
301 Text feature [recently] present in test data point [True]
308 Text feature [s3] present in test data point [True]
313 Text feature [activated] present in test data point [True]
314 Text feature [expressing] present in test data point [True]
318 Text feature [inhibitor] present in test data point [True]
322 Text feature [positive] present in test data point [True]
323 Text feature [increased] present in test data point [True]
324 Text feature [mapk] present in test data point [True]
326 Text feature [signals] present in test data point [True]
327 Text feature [specimens] present in test data point [True]
329 Text feature [mm] present in test data point [True]
330 Text feature [note] present in test data point [True]
```

```
331 Text feature [total] present in test data point [True]
332 Text feature [transforming] present in test data point [True]
333 Text feature [mutants] present in test data point [True]
340 Text feature [distinct] present in test data point [True]
343 Text feature [differentiation] present in test data point [True]
349 Text feature [domain] present in test data point [True]
Out of the top  500  features  42 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [405]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=
2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decre
ase=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=F
alse,
# class_weight=None)

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

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

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


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

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

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


best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(be
st_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, pr
edict_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))
```

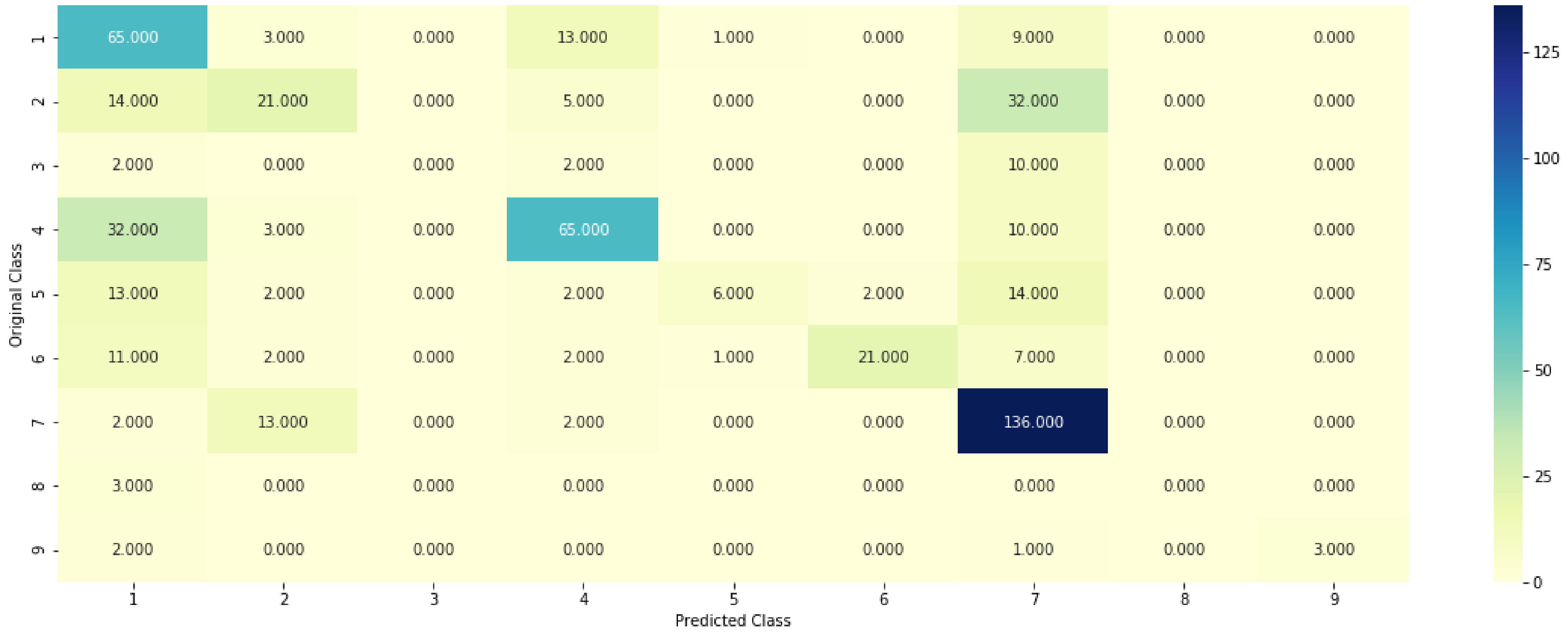
```
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss is:",log_loss(y_test, pred
ict_y, labels=clf.classes_, eps=1e-15))
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2144169880370983
for n_estimators = 100 and max depth =  10
Log Loss : 1.233972449299039
for n_estimators = 200 and max depth =  5
Log Loss : 1.201516888250428
for n_estimators = 200 and max depth =  10
Log Loss : 1.2282383764839266
for n_estimators = 500 and max depth =  5
Log Loss : 1.1909214940614425
for n_estimators = 500 and max depth =  10
Log Loss : 1.2226303354640131
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1878363368592595
for n_estimators = 1000 and max depth =  10
Log Loss : 1.218087967292333
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1849651802936925
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2145901435906123
For values of best estimator =  2000 The train log loss is: 0.8662215899166893
For values of best estimator =  2000 The cross validation log loss is: 1.1849651802936925
For values of best estimator =  2000 The test log loss is: 1.171482919005115
```

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

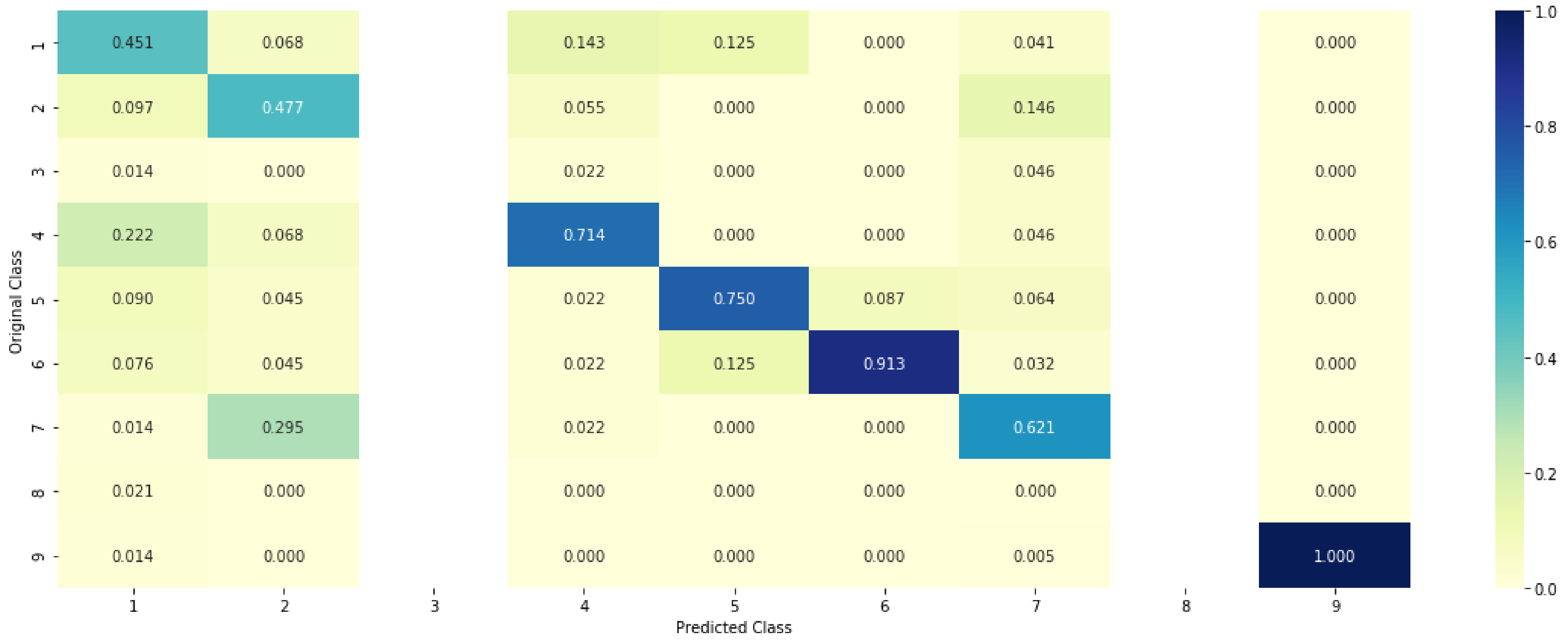
In [406]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=
2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decre
ase=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=F
alse,
# class_weight=None)

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

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

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

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(be
st_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 : 1.1849651802936925
Number of mis-classified points : 0.4041353383458647
-------------------- Confusion matrix --------------------
```

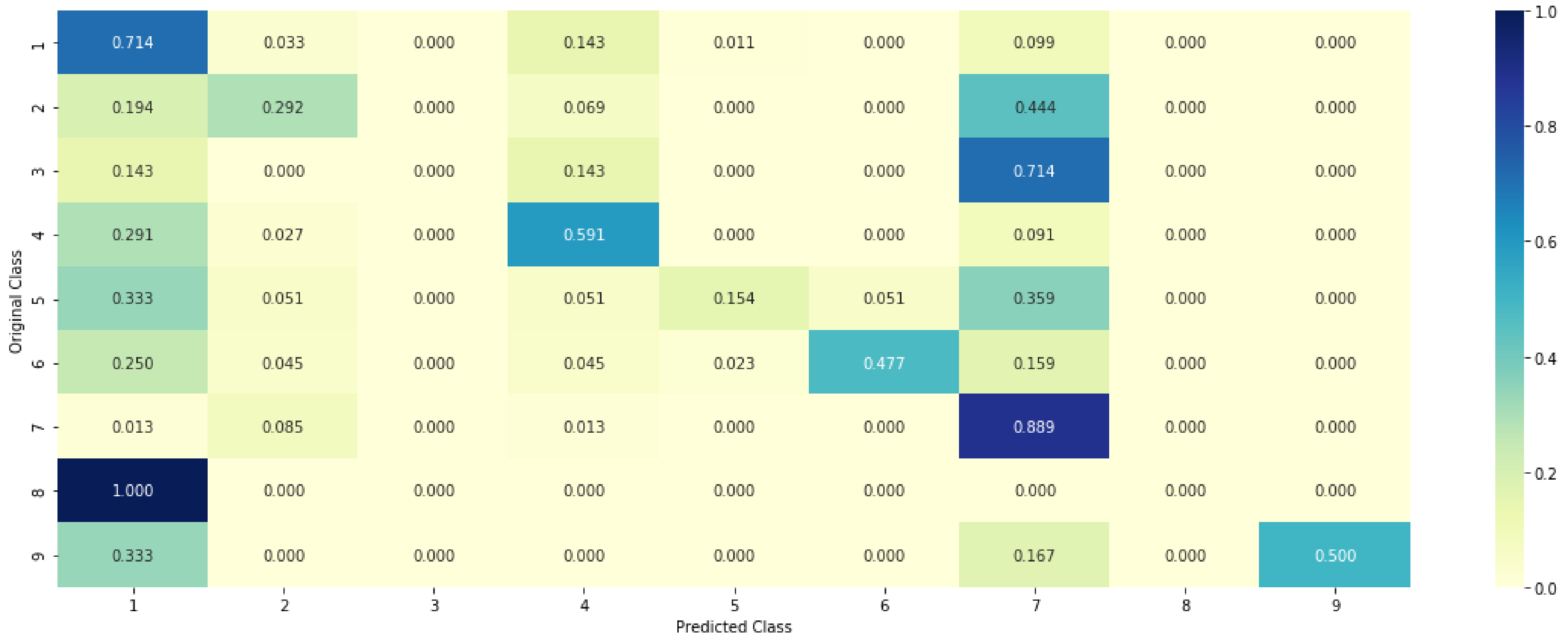


Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	65.000	3.000	0.000	13.000	1.000	0.000	9.000	0.000	0.000
2	14.000	21.000	0.000	5.000	0.000	0.000	32.000	0.000	0.000
3	2.000	0.000	0.000	2.000	0.000	0.000	10.000	0.000	0.000
4	32.000	3.000	0.000	65.000	0.000	0.000	10.000	0.000	0.000
5	13.000	2.000	0.000	2.000	6.000	2.000	14.000	0.000	0.000
6	11.000	2.000	0.000	2.000	1.000	21.000	7.000	0.000	0.000
7	2.000	13.000	0.000	2.000	0.000	0.000	136.000	0.000	0.000
8	3.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
9	2.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	3.000

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



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



	1	2	3	4	5	6	7	8	9
1	0.714	0.033	0.000	0.143	0.011	0.000	0.099	0.000	0.000
2	0.194	0.292	0.000	0.069	0.000	0.000	0.444	0.000	0.000
3	0.143	0.000	0.000	0.143	0.000	0.000	0.714	0.000	0.000
4	0.291	0.027	0.000	0.591	0.000	0.000	0.091	0.000	0.000
5	0.333	0.051	0.000	0.051	0.154	0.051	0.359	0.000	0.000
6	0.250	0.045	0.000	0.045	0.023	0.477	0.159	0.000	0.000
7	0.013	0.085	0.000	0.013	0.000	0.000	0.889	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
9	0.333	0.000	0.000	0.000	0.000	0.000	0.167	0.000	0.500

4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [407]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(be
st_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_poin
t_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0212 0.1138 0.0157 0.0213 0.0347 0.5535 0.2345 0.0028 0.0026]]
Actual Class : 6
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [tyrosine] present in test data point [True]
3 Text feature [inhibitors] present in test data point [True]
6 Text feature [activation] present in test data point [True]
7 Text feature [inhibitor] present in test data point [True]
8 Text feature [activated] present in test data point [True]
9 Text feature [loss] present in test data point [True]
10 Text feature [constitutive] present in test data point [True]
11 Text feature [phosphorylation] present in test data point [True]
12 Text feature [oncogenic] present in test data point [True]
13 Text feature [treatment] present in test data point [True]
15 Text feature [protein] present in test data point [True]
20 Text feature [cells] present in test data point [True]
21 Text feature [variants] present in test data point [True]
24 Text feature [receptor] present in test data point [True]
25 Text feature [growth] present in test data point [True]
26 Text feature [constitutively] present in test data point [True]
27 Text feature [treated] present in test data point [True]
29 Text feature [therapy] present in test data point [True]
30 Text feature [kinases] present in test data point [True]
31 Text feature [signaling] present in test data point [True]
32 Text feature [stability] present in test data point [True]
33 Text feature [functional] present in test data point [True]
36 Text feature [trials] present in test data point [True]
40 Text feature [patients] present in test data point [True]
41 Text feature [defective] present in test data point [True]
42 Text feature [mek] present in test data point [True]
43 Text feature [expression] present in test data point [True]
44 Text feature [phosphatase] present in test data point [True]
48 Text feature [cell] present in test data point [True]
50 Text feature [therapeutic] present in test data point [True]
51 Text feature [activate] present in test data point [True]
54 Text feature [inhibited] present in test data point [True]
55 Text feature [efficacy] present in test data point [True]
56 Text feature [inhibition] present in test data point [True]
```

```
57 Text feature [proteins] present in test data point [True]
58 Text feature [ic50] present in test data point [True]
60 Text feature [resistance] present in test data point [True]
64 Text feature [response] present in test data point [True]
65 Text feature [drug] present in test data point [True]
67 Text feature [clinical] present in test data point [True]
69 Text feature [variant] present in test data point [True]
76 Text feature [predicted] present in test data point [True]
77 Text feature [proliferation] present in test data point [True]
79 Text feature [ability] present in test data point [True]
81 Text feature [tki] present in test data point [True]
83 Text feature [conserved] present in test data point [True]
90 Text feature [imatinib] present in test data point [True]
91 Text feature [expected] present in test data point [True]
93 Text feature [activity] present in test data point [True]
96 Text feature [assays] present in test data point [True]
97 Text feature [sensitive] present in test data point [True]
98 Text feature [potential] present in test data point [True]
Out of the top  100  features  54 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [408]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4
))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_poin
t_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0973 0.2071 0.0211 0.1184 0.054  0.0639 0.3949 0.0309 0.0126]]
Actuall Class : 7
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [tyrosine] present in test data point [True]
4 Text feature [function] present in test data point [True]
6 Text feature [activation] present in test data point [True]
7 Text feature [inhibitor] present in test data point [True]
8 Text feature [activated] present in test data point [True]
9 Text feature [loss] present in test data point [True]
10 Text feature [constitutive] present in test data point [True]
12 Text feature [oncogenic] present in test data point [True]
13 Text feature [treatment] present in test data point [True]
14 Text feature [missense] present in test data point [True]
15 Text feature [protein] present in test data point [True]
20 Text feature [cells] present in test data point [True]
22 Text feature [transforming] present in test data point [True]
24 Text feature [receptor] present in test data point [True]
25 Text feature [growth] present in test data point [True]
26 Text feature [constitutively] present in test data point [True]
29 Text feature [therapy] present in test data point [True]
31 Text feature [signaling] present in test data point [True]
43 Text feature [expression] present in test data point [True]
48 Text feature [cell] present in test data point [True]
49 Text feature [functions] present in test data point [True]
50 Text feature [therapeutic] present in test data point [True]
51 Text feature [activate] present in test data point [True]
54 Text feature [inhibited] present in test data point [True]
56 Text feature [inhibition] present in test data point [True]
57 Text feature [proteins] present in test data point [True]
67 Text feature [clinical] present in test data point [True]
70 Text feature [3t3] present in test data point [True]
71 Text feature [ring] present in test data point [True]
74 Text feature [downstream] present in test data point [True]
77 Text feature [proliferation] present in test data point [True]
79 Text feature [ability] present in test data point [True]
80 Text feature [sequence] present in test data point [True]
83 Text feature [conserved] present in test data point [True]
```

```
84 Text feature [dna] present in test data point [True]
90 Text feature [imatinib] present in test data point [True]
91 Text feature [expected] present in test data point [True]
93 Text feature [activity] present in test data point [True]
94 Text feature [pathway] present in test data point [True]
95 Text feature [mapk] present in test data point [True]
96 Text feature [assays] present in test data point [True]
98 Text feature [potential] present in test data point [True]
Out of the top  100  features  44 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [409]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=
2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decre
ase=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=F
alse,
# class_weight=None)

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

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

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


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

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


best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=max_depth[int(be
st_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_train, predic
t_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))
```

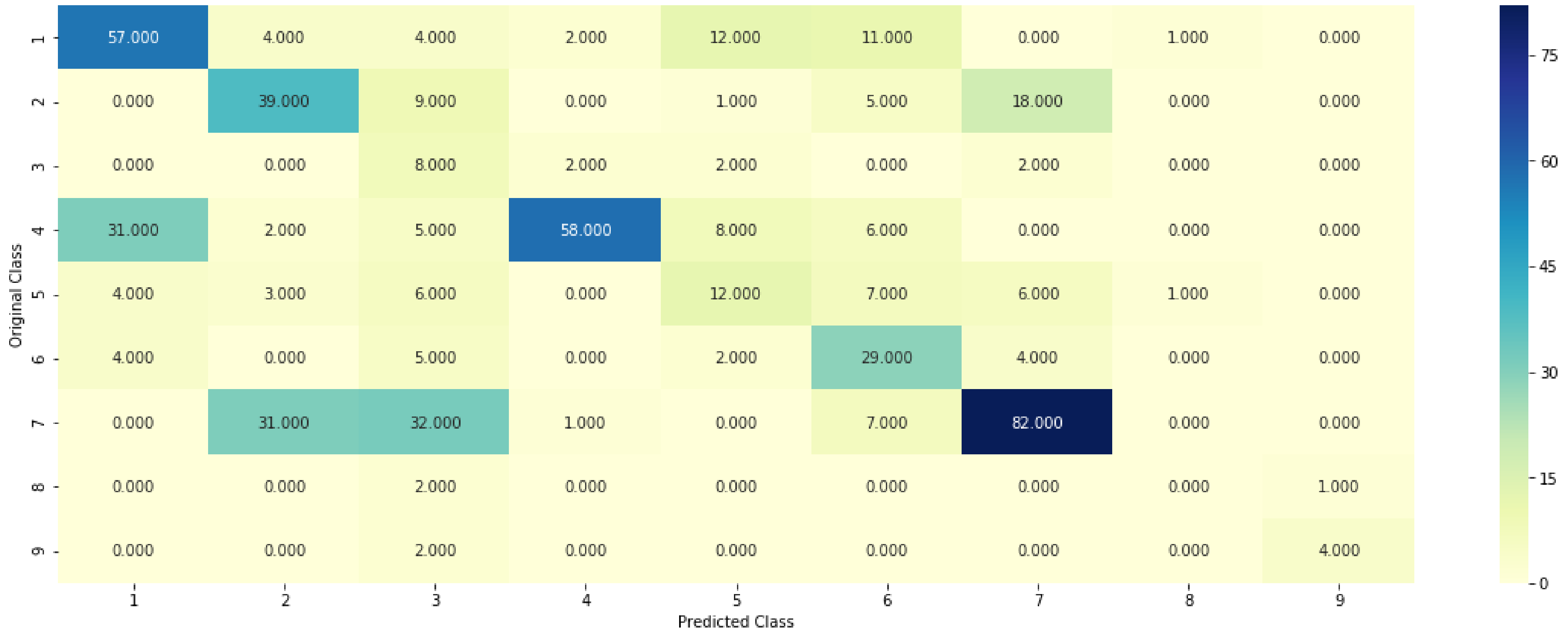
```
for n_estimators = 10 and max depth =  2
Log Loss : 2.043941590614312
for n_estimators = 10 and max depth =  3
Log Loss : 1.5695074323808598
for n_estimators = 10 and max depth =  5
Log Loss : 1.295517341793895
for n_estimators = 10 and max depth =  10
Log Loss : 1.7190858264799267
for n_estimators = 50 and max depth =  2
Log Loss : 1.6837054667629343
for n_estimators = 50 and max depth =  3
Log Loss : 1.4337747355020705
for n_estimators = 50 and max depth =  5
Log Loss : 1.4137401229901492
for n_estimators = 50 and max depth =  10
Log Loss : 1.5271809814063373
for n_estimators = 100 and max depth =  2
Log Loss : 1.6015986553489534
for n_estimators = 100 and max depth =  3
Log Loss : 1.500145822958219
for n_estimators = 100 and max depth =  5
Log Loss : 1.3686320851239344
for n_estimators = 100 and max depth =  10
Log Loss : 1.5857648284914814
for n_estimators = 200 and max depth =  2
Log Loss : 1.632741537322024
for n_estimators = 200 and max depth =  3
Log Loss : 1.5144126125388202
for n_estimators = 200 and max depth =  5
Log Loss : 1.4055962920203717
for n_estimators = 200 and max depth =  10
Log Loss : 1.6603229701363345
for n_estimators = 500 and max depth =  2
Log Loss : 1.713743025369839
for n_estimators = 500 and max depth =  3
Log Loss : 1.5654909791543352
for n_estimators = 500 and max depth =  5
Log Loss : 1.3562817301571208
for n_estimators = 500 and max depth =  10
Log Loss : 1.6736975753836105
```

```
for n_estimators = 1000 and max depth =  2
Log Loss : 1.676293530152117
for n_estimators = 1000 and max depth =  3
Log Loss : 1.5541473971631654
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3387035666236387
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6709512410143597
For values of best alpha =  10 The train log loss is: 0.0738695040940323
For values of best alpha =  10 The cross validation log loss is: 1.2955173417938948
For values of best alpha =  10 The test log loss is: 1.3964897970229424
```

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

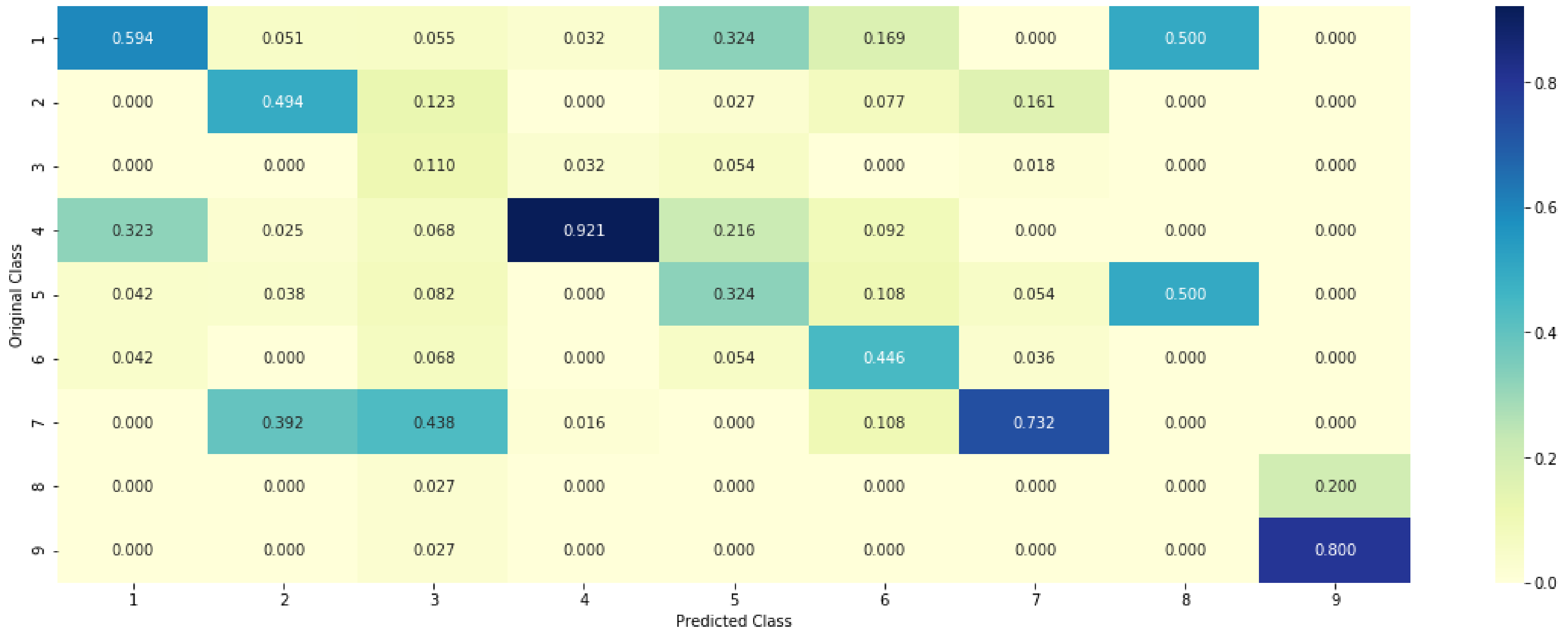
In [410]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=
2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decre
ase=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=F
alse,
# class_weight=None)

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

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

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

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimators=alpha[int(best_alpha/4)], crit
erion='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 : 1.295517341793895
Number of mis-classified points : 0.4567669172932331
-------------------- Confusion matrix --------------------
```



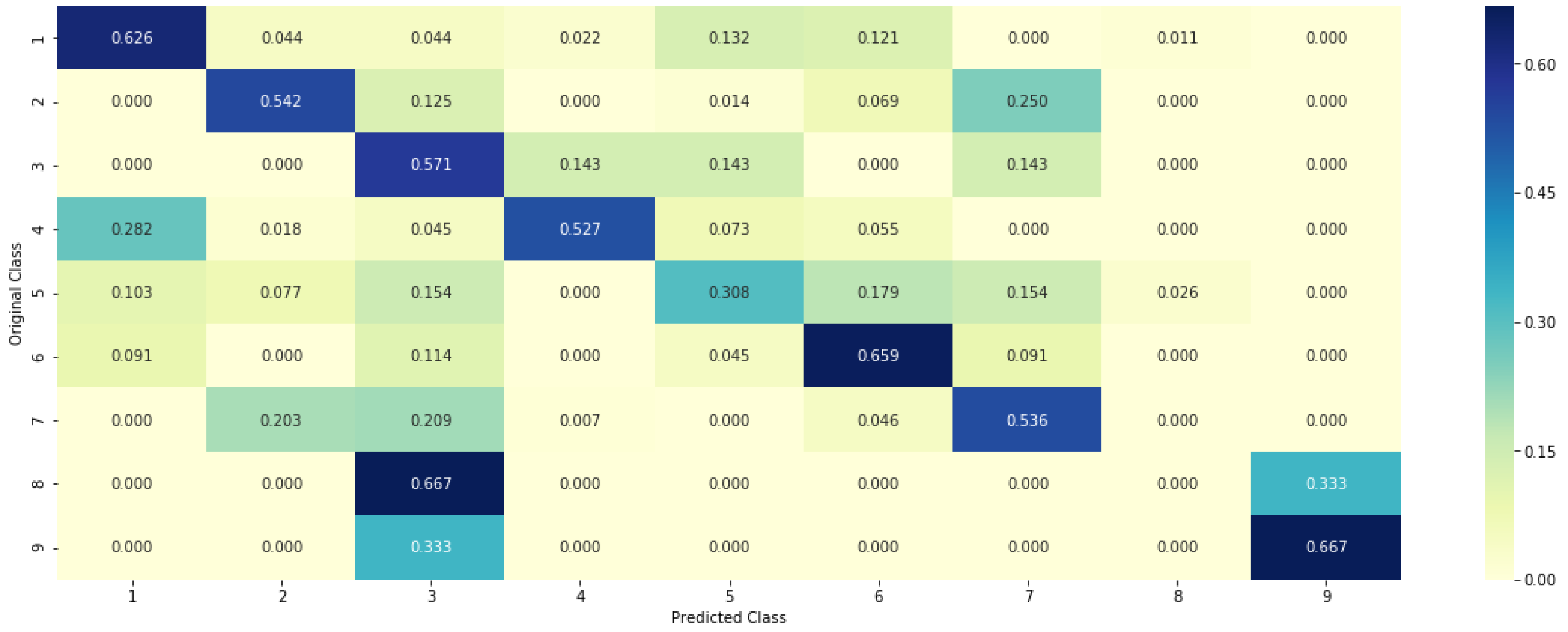
	1	2	3	4	5	6	7	8	9
1	57.000	4.000	4.000	2.000	12.000	11.000	0.000	1.000	0.000
2	0.000	39.000	9.000	0.000	1.000	5.000	18.000	0.000	0.000
3	0.000	0.000	8.000	2.000	2.000	0.000	2.000	0.000	0.000
4	31.000	2.000	5.000	58.000	8.000	6.000	0.000	0.000	0.000
5	4.000	3.000	6.000	0.000	12.000	7.000	6.000	1.000	0.000
6	4.000	0.000	5.000	0.000	2.000	29.000	4.000	0.000	0.000
7	0.000	31.000	32.000	1.000	0.000	7.000	82.000	0.000	0.000
8	0.000	0.000	2.000	0.000	0.000	0.000	0.000	0.000	1.000
9	0.000	0.000	2.000	0.000	0.000	0.000	0.000	0.000	4.000

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



	1	2	3	4	5	6	7	8	9
1	0.594	0.051	0.055	0.032	0.324	0.169	0.000	0.500	0.000
2	0.000	0.494	0.123	0.000	0.027	0.077	0.161	0.000	0.000
3	0.000	0.000	0.110	0.032	0.054	0.000	0.018	0.000	0.000
4	0.323	0.025	0.068	0.921	0.216	0.092	0.000	0.000	0.000
5	0.042	0.038	0.082	0.000	0.324	0.108	0.054	0.500	0.000
6	0.042	0.000	0.068	0.000	0.054	0.446	0.036	0.000	0.000
7	0.000	0.392	0.438	0.016	0.000	0.108	0.732	0.000	0.000
8	0.000	0.000	0.027	0.000	0.000	0.000	0.000	0.000	0.200
9	0.000	0.000	0.027	0.000	0.000	0.000	0.000	0.000	0.800

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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [411]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=max_depth[int(be
st_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].r
eshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0224 0.0535 0.1009 0.022  0.0423 0.4886 0.2231 0.0209 0.0265]]
Actual Class : 6
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [412]:
```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index].r
eshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0235 0.3812 0.1527 0.0233 0.1056 0.0636 0.1908 0.0255 0.0339]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [413]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDCl
assifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=
None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=
0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



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

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



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

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

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

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


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

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


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

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehotCoding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
```

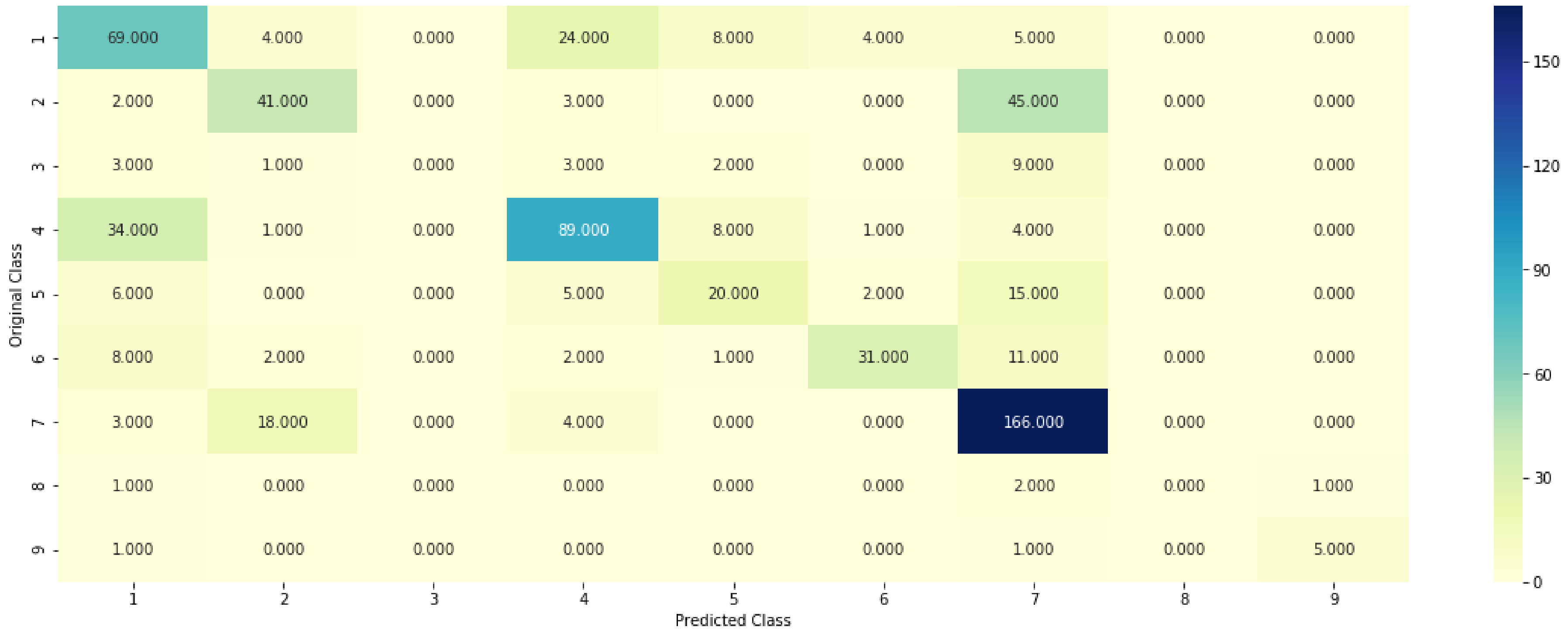
```
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding))))
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sclf.predict_pr
oba(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: 0.99
Support vector machines : Log Loss: 1.78
Naive Bayes : Log Loss: 1.15
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.032
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.490
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.132
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.330
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.803
```

4.7.2 testing the model with the best hyper parameters

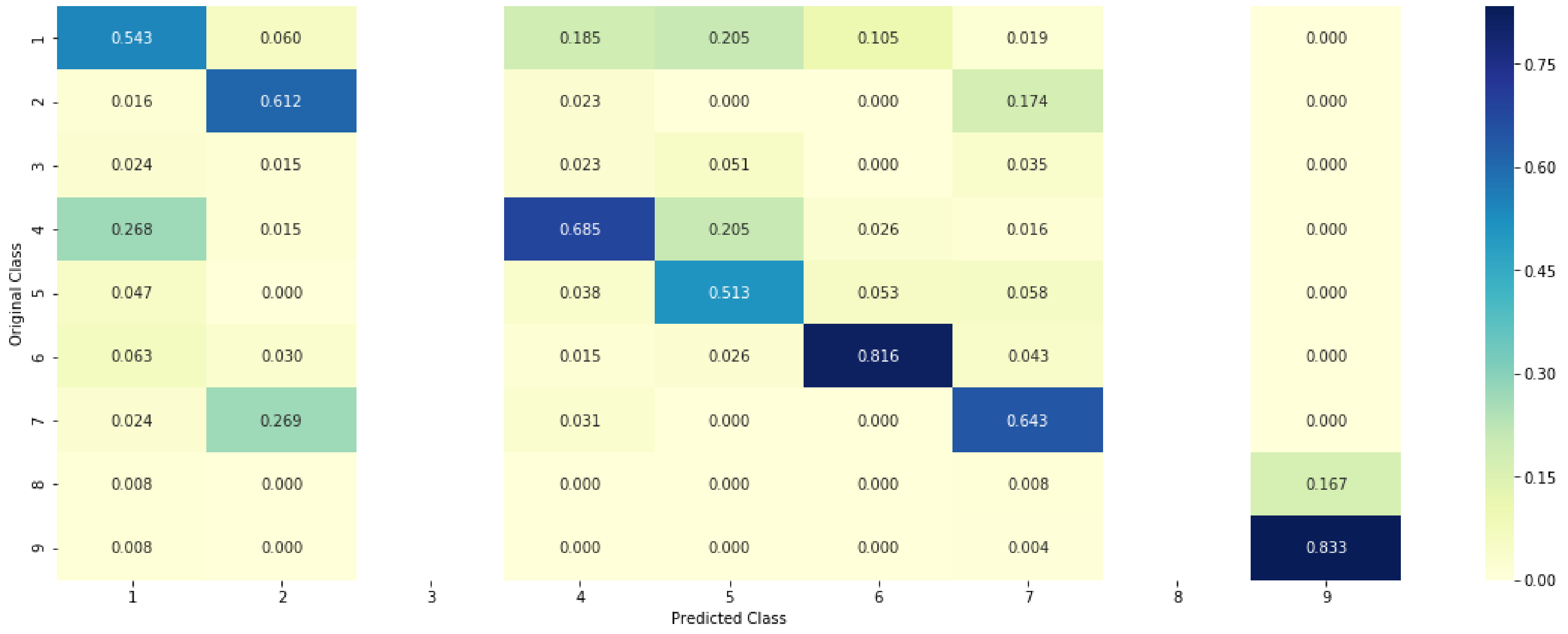
In [414]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

```
Log loss (train) on the stacking classifier : 0.5513807643128392
Log loss (CV) on the stacking classifier : 1.1321770265944153
Log loss (test) on the stacking classifier : 1.1413472730900858
Number of missclassified point : 0.3669172932330827
-------------------- Confusion matrix --------------------
```



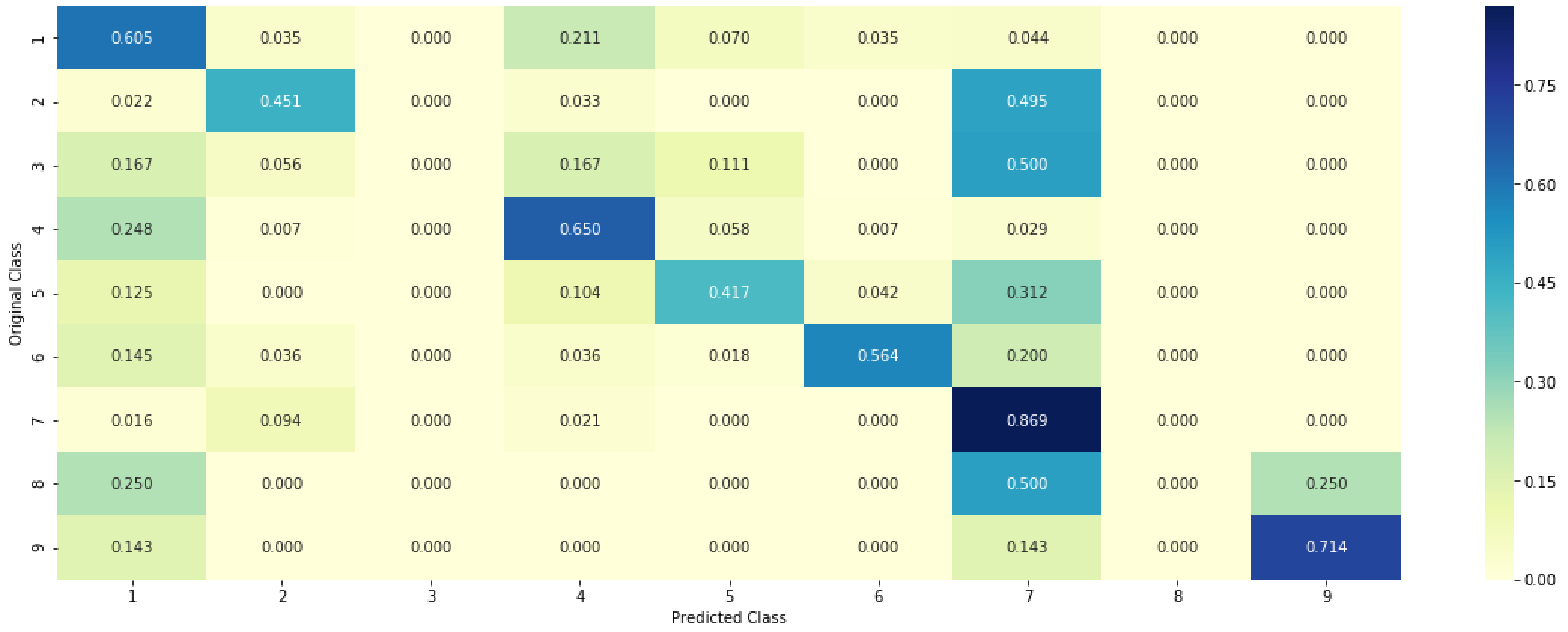
Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	69.000	4.000	0.000	24.000	8.000	4.000	5.000	0.000	0.000
2	2.000	41.000	0.000	3.000	0.000	0.000	45.000	0.000	0.000
3	3.000	1.000	0.000	3.000	2.000	0.000	9.000	0.000	0.000
4	34.000	1.000	0.000	89.000	8.000	1.000	4.000	0.000	0.000
5	6.000	0.000	0.000	5.000	20.000	2.000	15.000	0.000	0.000
6	8.000	2.000	0.000	2.000	1.000	31.000	11.000	0.000	0.000
7	3.000	18.000	0.000	4.000	0.000	0.000	166.000	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	0.000	2.000	0.000	1.000
9	1.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	5.000

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



Original Class	1	2	3	4	5	6	7	8	9
1	0.543	0.060		0.185	0.205	0.105	0.019		0.000
2	0.016	0.612		0.023	0.000	0.000	0.174		0.000
3	0.024	0.015		0.023	0.051	0.000	0.035		0.000
4	0.268	0.015		0.685	0.205	0.026	0.016		0.000
5	0.047	0.000		0.038	0.513	0.053	0.058		0.000
6	0.063	0.030		0.015	0.026	0.816	0.043		0.000
7	0.024	0.269		0.031	0.000	0.000	0.643		0.000
8	0.008	0.000		0.000	0.000	0.000	0.008		0.167
9	0.008	0.000		0.000	0.000	0.000	0.004		0.833

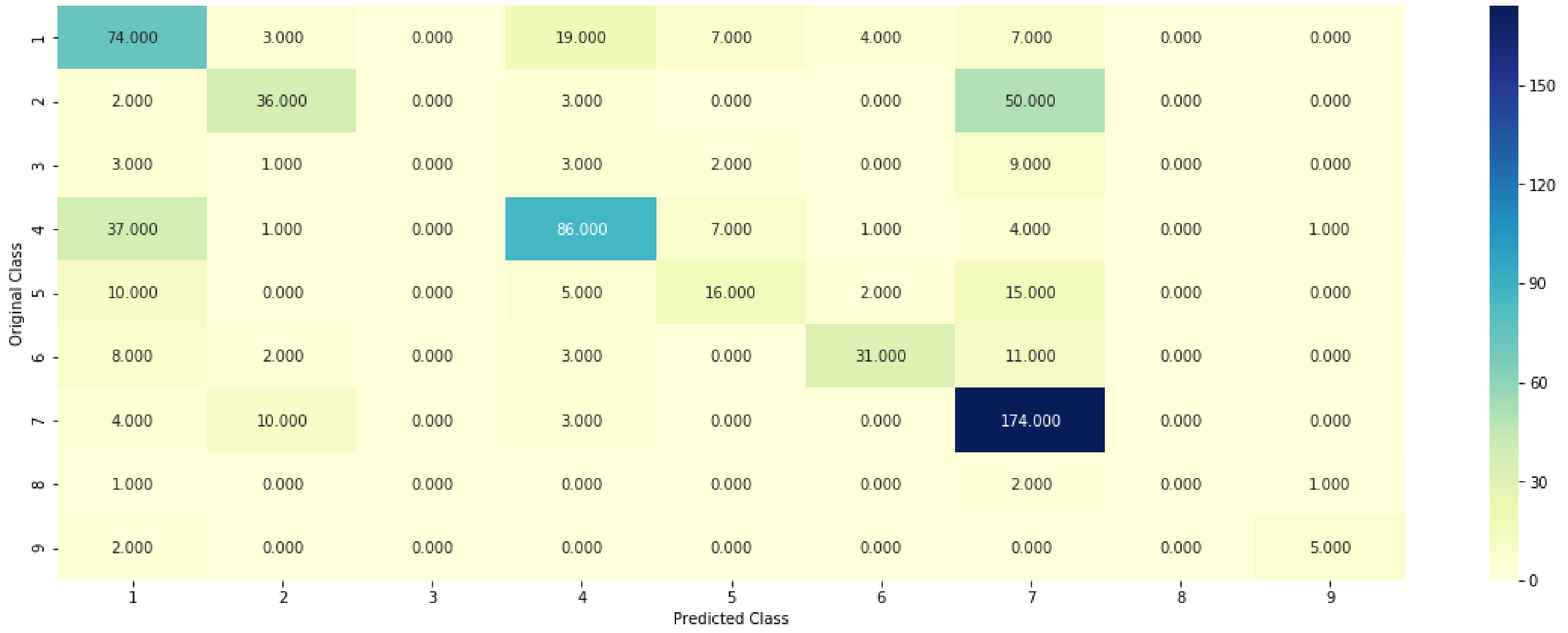
```
-------------------- Recall matrix (Row sum=1) --------------------
```



4.7.3 Maximum Voting classifier

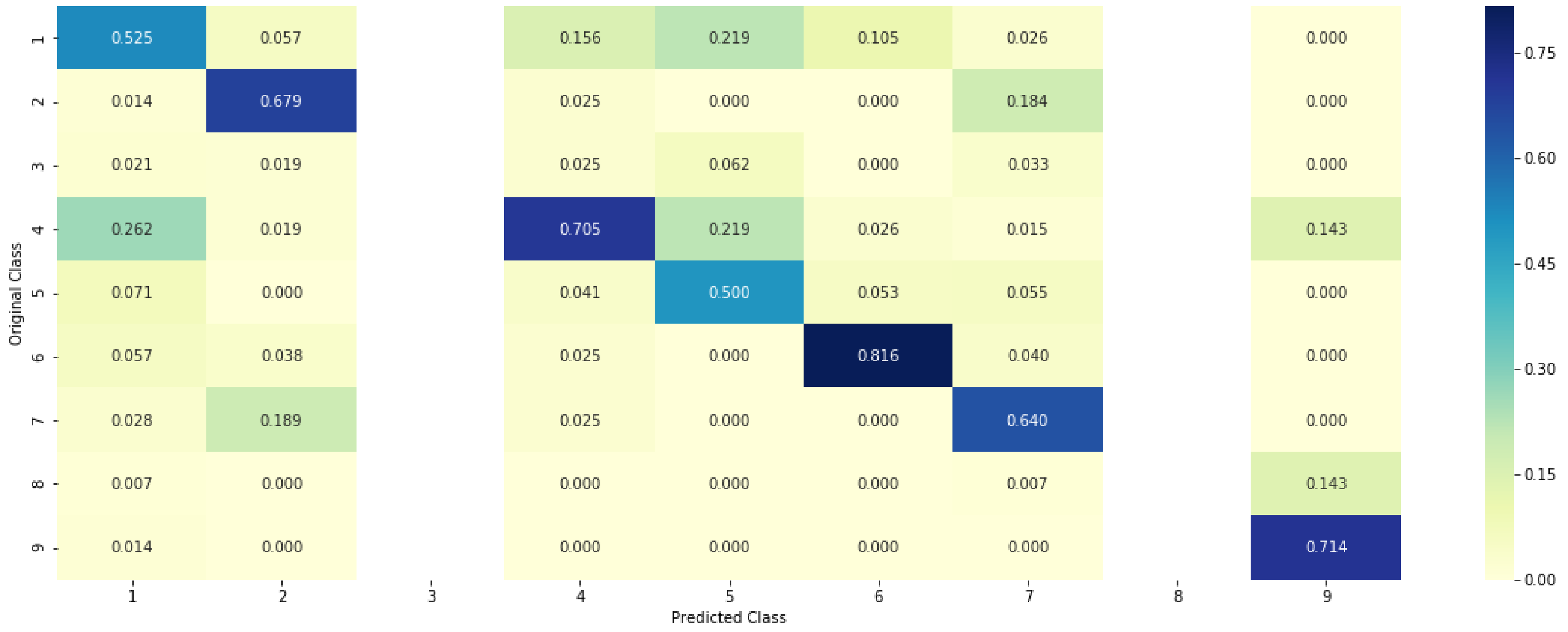
In [415]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCoding)- test_y))/test_y.s
hape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

```
Log loss (train) on the VotingClassifier : 0.8498888518797484
Log loss (CV) on the VotingClassifier : 1.1595406118923846
Log loss (test) on the VotingClassifier : 1.1731392187055079
Number of missclassified point : 0.36541353383458647
-------------------- Confusion matrix --------------------
```



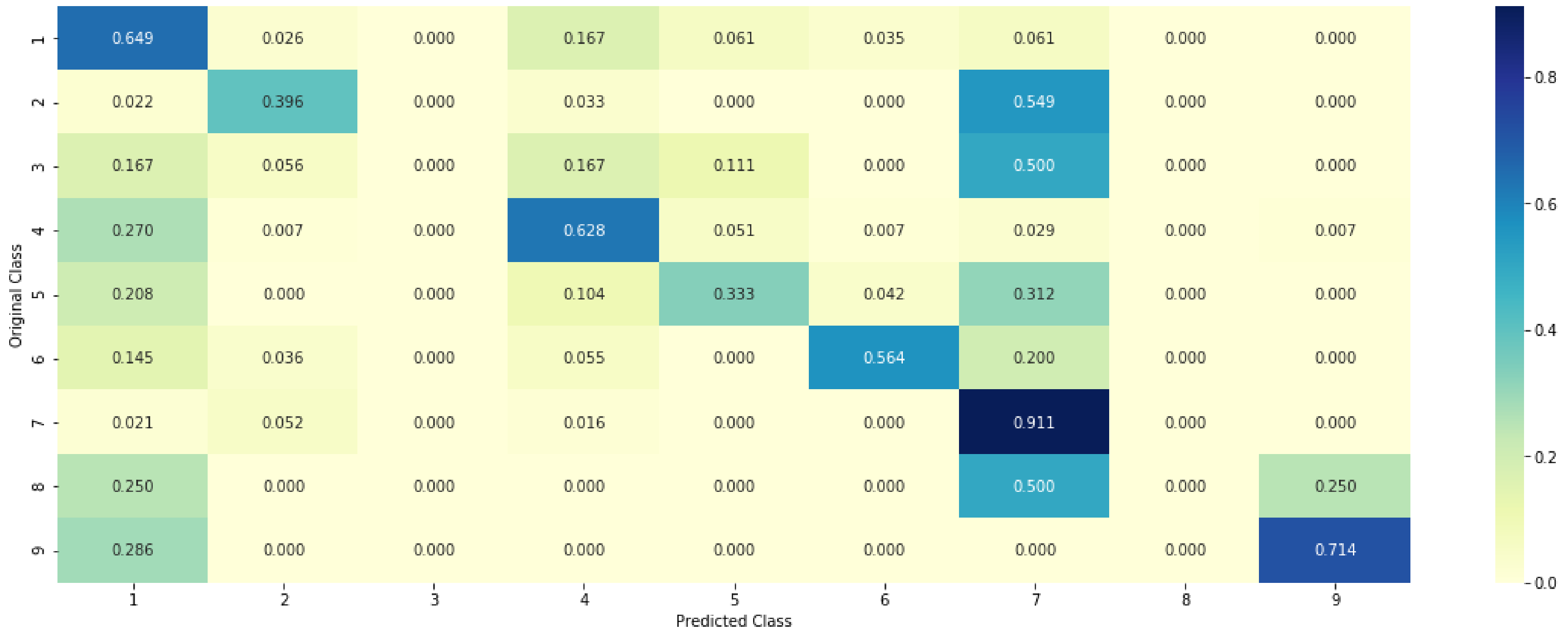
Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	74.000	3.000	0.000	19.000	7.000	4.000	7.000	0.000	0.000
2	2.000	36.000	0.000	3.000	0.000	0.000	50.000	0.000	0.000
3	3.000	1.000	0.000	3.000	2.000	0.000	9.000	0.000	0.000
4	37.000	1.000	0.000	86.000	7.000	1.000	4.000	0.000	1.000
5	10.000	0.000	0.000	5.000	16.000	2.000	15.000	0.000	0.000
6	8.000	2.000	0.000	3.000	0.000	31.000	11.000	0.000	0.000
7	4.000	10.000	0.000	3.000	0.000	0.000	174.000	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	0.000	2.000	0.000	1.000
9	2.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	5.000

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



	1	2	3	4	5	6	7	8	9
1	0.525	0.057		0.156	0.219	0.105	0.026		0.000
2	0.014	0.679		0.025	0.000	0.000	0.184		0.000
3	0.021	0.019		0.025	0.062	0.000	0.033		0.000
4	0.262	0.019		0.705	0.219	0.026	0.015		0.143
5	0.071	0.000		0.041	0.500	0.053	0.055		0.000
6	0.057	0.038		0.025	0.000	0.816	0.040		0.000
7	0.028	0.189		0.025	0.000	0.000	0.640		0.000
8	0.007	0.000		0.000	0.000	0.000	0.007		0.143
9	0.014	0.000		0.000	0.000	0.000	0.000		0.714

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



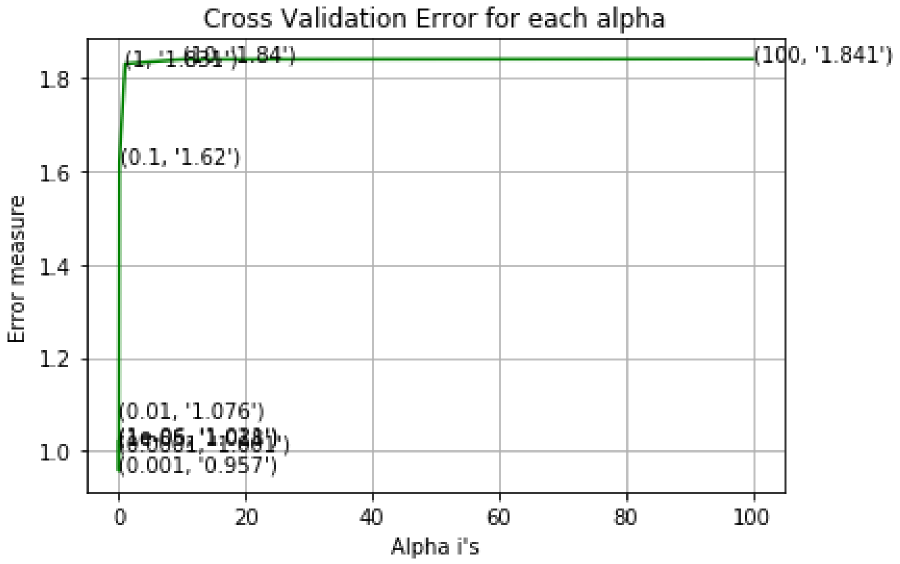
	1	2	3	4	5	6	7	8	9
1	0.649	0.026	0.000	0.167	0.061	0.035	0.061	0.000	0.000
2	0.022	0.396	0.000	0.033	0.000	0.000	0.549	0.000	0.000
3	0.167	0.056	0.000	0.167	0.111	0.000	0.500	0.000	0.000
4	0.270	0.007	0.000	0.628	0.051	0.007	0.029	0.000	0.007
5	0.208	0.000	0.000	0.104	0.333	0.042	0.312	0.000	0.000
6	0.145	0.036	0.000	0.055	0.000	0.564	0.200	0.000	0.000
7	0.021	0.052	0.000	0.016	0.000	0.000	0.911	0.000	0.000
8	0.250	0.000	0.000	0.000	0.000	0.000	0.500	0.000	0.250
9	0.286	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.714

LogisticRegression(with weight balance) after feature engineering

In [417]:
```
train_x_onehotCoding_fe=np.sqrt(train_x_onehotCoding)
test_x_onehotCoding_fe=np.sqrt(test_x_onehotCoding)
cv_x_onehotCoding_fe=np.sqrt(cv_x_onehotCoding)
```

In [418]:
```
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_fe, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding_fe, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding_fe)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCoding_fe)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, la
bels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding_fe)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predi
ct_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding_fe)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labe
ls=clf.classes_, eps=1e-15))
```

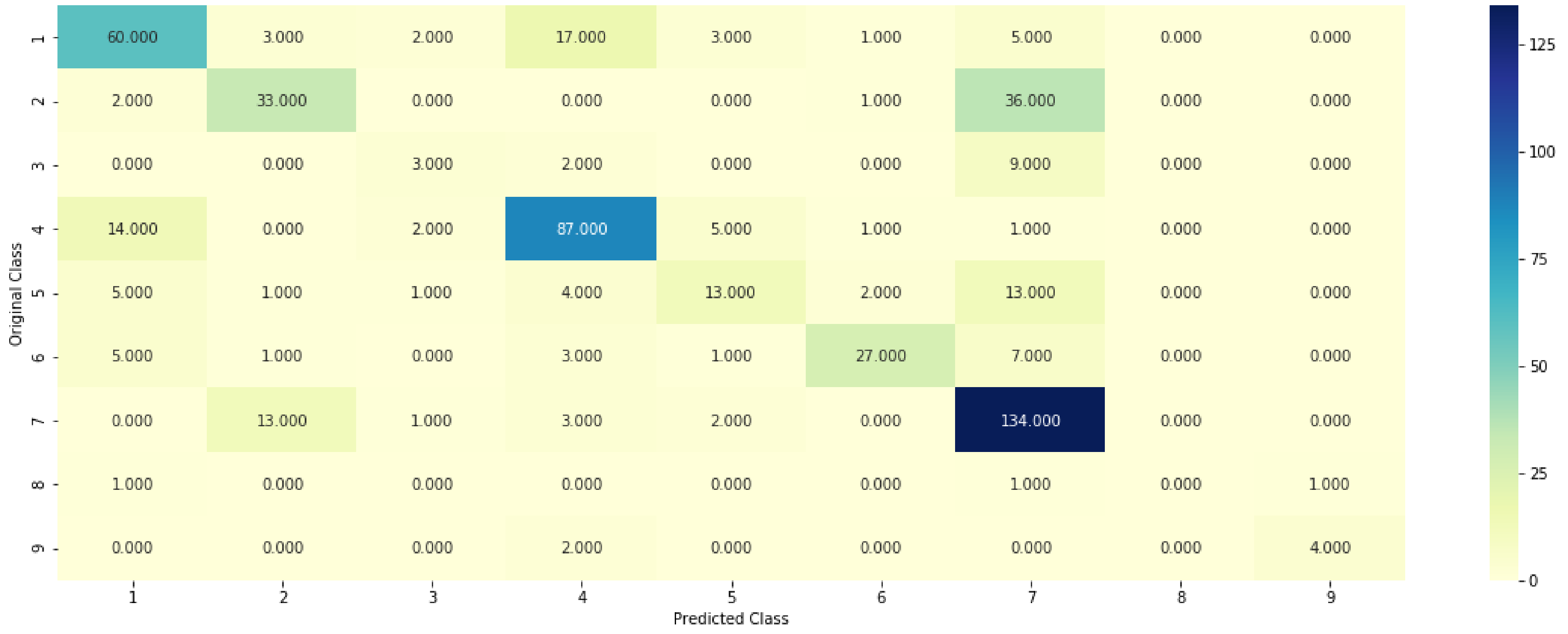
```
for alpha = 1e-06
Log Loss : 1.0206812607986404
for alpha = 1e-05
Log Loss : 1.0177041613143625
for alpha = 0.0001
Log Loss : 1.0007585730488204
for alpha = 0.001
Log Loss : 0.9569234363293772
for alpha = 0.01
Log Loss : 1.0760136568526453
for alpha = 0.1
Log Loss : 1.6202295953921209
for alpha = 1
Log Loss : 1.8311149580845778
for alpha = 10
Log Loss : 1.840341928348447
for alpha = 100
Log Loss : 1.8409331202385346
```



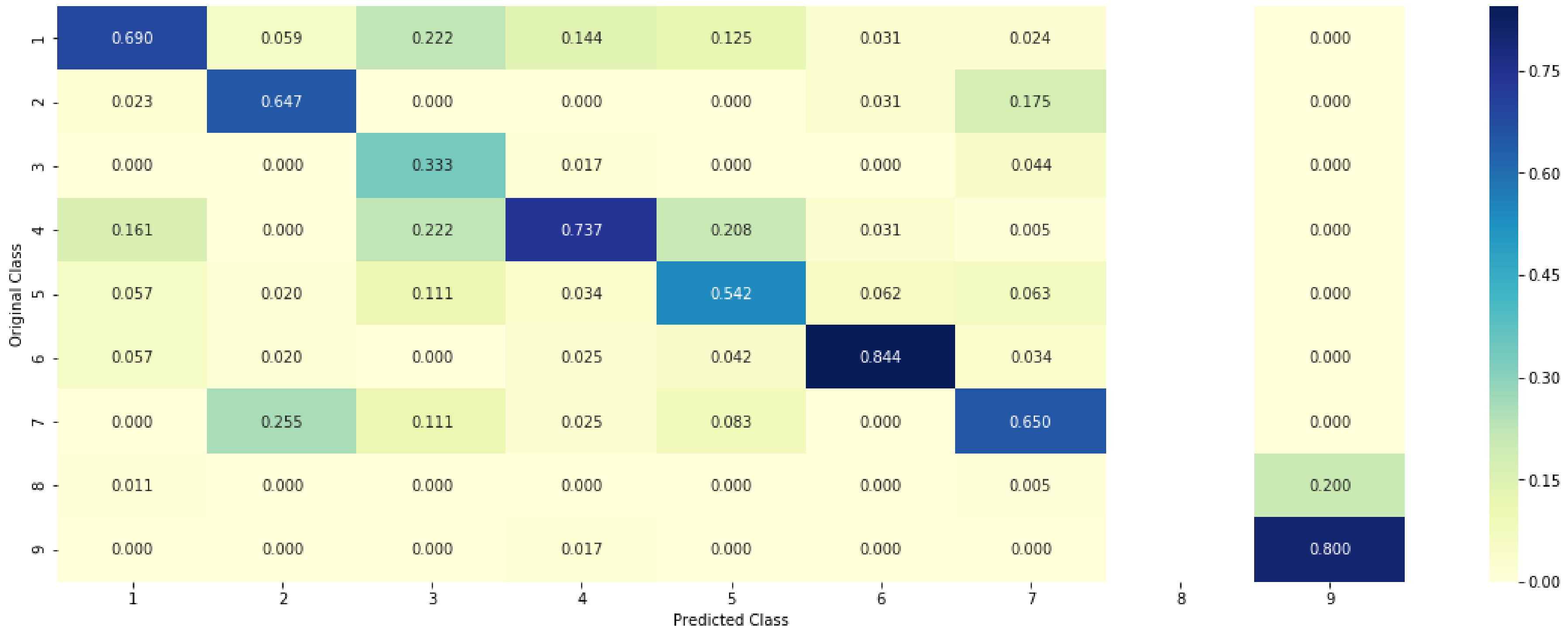
```
For values of best alpha =  0.001 The train log loss is: 0.6385987909984792
For values of best alpha =  0.001 The cross validation log loss is: 0.9569234363293772
For values of best alpha =  0.001 The test log loss is: 0.9987658104925244
```

In [419]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding_fe, train_y, cv_x_onehotCoding_fe, cv_y, clf)
```

```
Log loss : 0.95692343632937772
Number of mis-classified points : 0.32142857142857145
-------------------- Confusion matrix --------------------
```

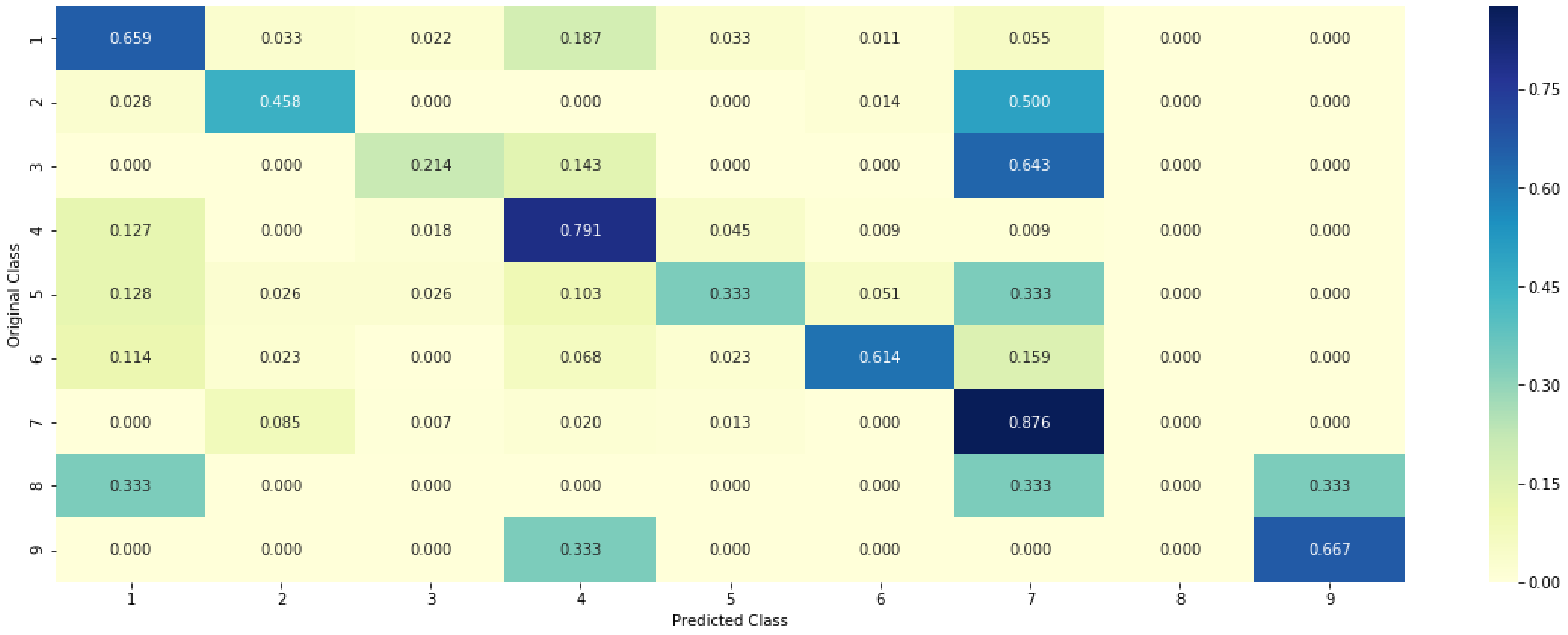


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



Original \ Predicted	1	2	3	4	5	6	7	8	9
1	0.690	0.059	0.222	0.144	0.125	0.031	0.024		0.000
2	0.023	0.647	0.000	0.000	0.000	0.031	0.175		0.000
3	0.000	0.000	0.333	0.017	0.000	0.000	0.044		0.000
4	0.161	0.000	0.222	0.737	0.208	0.031	0.005		0.000
5	0.057	0.020	0.111	0.034	0.542	0.062	0.063		0.000
6	0.057	0.020	0.000	0.025	0.042	0.844	0.034		0.000
7	0.000	0.255	0.111	0.025	0.083	0.000	0.650		0.000
8	0.011	0.000	0.000	0.000	0.000	0.000	0.005		0.200
9	0.000	0.000	0.000	0.017	0.000	0.000	0.000		0.800

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



	1	2	3	4	5	6	7	8	9
1	0.659	0.033	0.022	0.187	0.033	0.011	0.055	0.000	0.000
2	0.028	0.458	0.000	0.000	0.000	0.014	0.500	0.000	0.000
3	0.000	0.000	0.214	0.143	0.000	0.000	0.643	0.000	0.000
4	0.127	0.000	0.018	0.791	0.045	0.009	0.009	0.000	0.000
5	0.128	0.026	0.026	0.103	0.333	0.051	0.333	0.000	0.000
6	0.114	0.023	0.000	0.068	0.023	0.614	0.159	0.000	0.000
7	0.000	0.085	0.007	0.020	0.013	0.000	0.876	0.000	0.000
8	0.333	0.000	0.000	0.000	0.000	0.000	0.333	0.000	0.333
9	0.000	0.000	0.000	0.333	0.000	0.000	0.000	0.000	0.667

Incorrectly Classified point

In [420]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding_fe,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_fe[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding_fe[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],t
est_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[2.900e-03 7.850e-02 3.500e-03 3.600e-03 5.500e-03 7.975e-01 1.054e-01
  2.700e-03 5.000e-04]]
Actual Class : 6
--------------------------------------------------
5 Text feature [blue] present in test data point [True]
11 Text feature [significant] present in test data point [True]
12 Text feature [mek1] present in test data point [True]
14 Text feature [binds] present in test data point [True]
23 Text feature [selection] present in test data point [True]
27 Text feature [substrate] present in test data point [True]
30 Text feature [observation] present in test data point [True]
36 Text feature [frequent] present in test data point [True]
39 Text feature [regulated] present in test data point [True]
41 Text feature [breast] present in test data point [True]
58 Text feature [decreased] present in test data point [True]
60 Text feature [family] present in test data point [True]
63 Text feature [development] present in test data point [True]
65 Text feature [identified] present in test data point [True]
75 Text feature [loss] present in test data point [True]
79 Text feature [inhibitors] present in test data point [True]
88 Text feature [right] present in test data point [True]
89 Text feature [values] present in test data point [True]
97 Text feature [mek] present in test data point [True]
99 Text feature [phospho] present in test data point [True]
100 Text feature [targeted] present in test data point [True]
101 Text feature [altered] present in test data point [True]
103 Text feature [suppression] present in test data point [True]
104 Text feature [inhibition] present in test data point [True]
106 Text feature [cause] present in test data point [True]
111 Text feature [conformation] present in test data point [True]
112 Text feature [gel] present in test data point [True]
113 Text feature [concentration] present in test data point [True]
114 Text feature [phase] present in test data point [True]
115 Text feature [important] present in test data point [True]
117 Text feature [length] present in test data point [True]
119 Text feature [therapeutic] present in test data point [True]
120 Text feature [members] present in test data point [True]
122 Text feature [use] present in test data point [True]
123 Text feature [evidence] present in test data point [True]
```

```
124 Text feature [helix] present in test data point [True]
130 Text feature [thus] present in test data point [True]
132 Text feature [biochemical] present in test data point [True]
134 Text feature [bind] present in test data point [True]
135 Text feature [variant] present in test data point [True]
142 Text feature [involved] present in test data point [True]
146 Text feature [potential] present in test data point [True]
148 Text feature [structures] present in test data point [True]
150 Text feature [mechanism] present in test data point [True]
155 Text feature [distinct] present in test data point [True]
156 Text feature [red] present in test data point [True]
160 Text feature [groups] present in test data point [True]
164 Text feature [enzyme] present in test data point [True]
168 Text feature [region] present in test data point [True]
172 Text feature [time] present in test data point [True]
176 Text feature [resistance] present in test data point [True]
178 Text feature [frequency] present in test data point [True]
181 Text feature [src] present in test data point [True]
184 Text feature [40] present in test data point [True]
189 Text feature [group] present in test data point [True]
192 Text feature [express] present in test data point [True]
196 Text feature [factors] present in test data point [True]
197 Text feature [expected] present in test data point [True]
199 Text feature [44] present in test data point [True]
201 Text feature [1a] present in test data point [True]
212 Text feature [well] present in test data point [True]
214 Text feature [relevant] present in test data point [True]
216 Text feature [dose] present in test data point [True]
218 Text feature [normal] present in test data point [True]
220 Text feature [targeting] present in test data point [True]
221 Text feature [model] present in test data point [True]
224 Text feature [60] present in test data point [True]
225 Text feature [driver] present in test data point [True]
227 Text feature [lower] present in test data point [True]
229 Text feature [low] present in test data point [True]
231 Text feature [proliferation] present in test data point [True]
233 Text feature [substitution] present in test data point [True]
237 Text feature [left] present in test data point [True]
238 Text feature [leading] present in test data point [True]
241 Text feature [nm] present in test data point [True]
```

```
242 Text feature [second] present in test data point [True]
246 Text feature [whole] present in test data point [True]
250 Text feature [structural] present in test data point [True]
251 Text feature [clinically] present in test data point [True]
254 Text feature [direct] present in test data point [True]
258 Text feature [kinases] present in test data point [True]
260 Text feature [remains] present in test data point [True]
262 Text feature [kinase] present in test data point [True]
264 Text feature [receptors] present in test data point [True]
266 Text feature [value] present in test data point [True]
270 Text feature [jak2] present in test data point [True]
271 Text feature [factor] present in test data point [True]
272 Text feature [catalytic] present in test data point [True]
274 Text feature [substitutions] present in test data point [True]
279 Text feature [measured] present in test data point [True]
285 Text feature [multiple] present in test data point [True]
287 Text feature [ability] present in test data point [True]
290 Text feature [since] present in test data point [True]
292 Text feature [reported] present in test data point [True]
293 Text feature [hours] present in test data point [True]
295 Text feature [predicted] present in test data point [True]
300 Text feature [single] present in test data point [True]
301 Text feature [interactions] present in test data point [True]
302 Text feature [probe] present in test data point [True]
304 Text feature [receptor] present in test data point [True]
307 Text feature [greater] present in test data point [True]
309 Text feature [proportion] present in test data point [True]
310 Text feature [structure] present in test data point [True]
311 Text feature [initial] present in test data point [True]
313 Text feature [state] present in test data point [True]
315 Text feature [hydrogen] present in test data point [True]
317 Text feature [residues] present in test data point [True]
318 Text feature [confer] present in test data point [True]
321 Text feature [determine] present in test data point [True]
322 Text feature [individual] present in test data point [True]
323 Text feature [43] present in test data point [True]
329 Text feature [provide] present in test data point [True]
331 Text feature [interaction] present in test data point [True]
332 Text feature [45] present in test data point [True]
336 Text feature [variants] present in test data point [True]
```

```
337 Text feature [site] present in test data point [True]
347 Text feature [first] present in test data point [True]
348 Text feature [ic50] present in test data point [True]
349 Text feature [using] present in test data point [True]
351 Text feature [free] present in test data point [True]
352 Text feature [egfr] present in test data point [True]
354 Text feature [experiments] present in test data point [True]
356 Text feature [mechanisms] present in test data point [True]
361 Text feature [known] present in test data point [True]
363 Text feature [chemotherapy] present in test data point [True]
368 Text feature [grade] present in test data point [True]
371 Text feature [clones] present in test data point [True]
372 Text feature [il] present in test data point [True]
373 Text feature [active] present in test data point [True]
375 Text feature [conserved] present in test data point [True]
376 Text feature [suggests] present in test data point [True]
378 Text feature [associated] present in test data point [True]
381 Text feature [previous] present in test data point [True]
382 Text feature [four] present in test data point [True]
391 Text feature [days] present in test data point [True]
394 Text feature [analyses] present in test data point [True]
395 Text feature [mice] present in test data point [True]
398 Text feature [would] present in test data point [True]
399 Text feature [whether] present in test data point [True]
403 Text feature [seven] present in test data point [True]
404 Text feature [relatively] present in test data point [True]
408 Text feature [concentrations] present in test data point [True]
412 Text feature [number] present in test data point [True]
415 Text feature [20] present in test data point [True]
416 Text feature [demonstrated] present in test data point [True]
419 Text feature [novel] present in test data point [True]
421 Text feature [indeed] present in test data point [True]
423 Text feature [set] present in test data point [True]
428 Text feature [atp] present in test data point [True]
429 Text feature [sequencing] present in test data point [True]
430 Text feature [image] present in test data point [True]
432 Text feature [secondary] present in test data point [True]
436 Text feature [4b] present in test data point [True]
441 Text feature [essential] present in test data point [True]
442 Text feature [studies] present in test data point [True]
```

```
443 Text feature [17] present in test data point [True]
446 Text feature [due] present in test data point [True]
447 Text feature [therapy] present in test data point [True]
450 Text feature [bound] present in test data point [True]
452 Text feature [strong] present in test data point [True]
462 Text feature [experimental] present in test data point [True]
466 Text feature [colonies] present in test data point [True]
468 Text feature [increased] present in test data point [True]
471 Text feature [11] present in test data point [True]
475 Text feature [result] present in test data point [True]
482 Text feature [via] present in test data point [True]
483 Text feature [genetic] present in test data point [True]
490 Text feature [loop] present in test data point [True]
492 Text feature [new] present in test data point [True]
493 Text feature [10] present in test data point [True]
495 Text feature [may] present in test data point [True]
496 Text feature [sites] present in test data point [True]
Out of the top  500  features  172 are present in query point
```

Correctly Classified point

In [421]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_fe[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding_fe[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],t
est_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0075 0.0445 0.0064 0.003  0.0726 0.0197 0.8406 0.0039 0.0018]]
Actual Class : 7
--------------------------------------------------
2 Text feature [activating] present in test data point [True]
3 Text feature [constitutive] present in test data point [True]
5 Text feature [3b] present in test data point [True]
7 Text feature [activate] present in test data point [True]
9 Text feature [downstream] present in test data point [True]
10 Text feature [activated] present in test data point [True]
11 Text feature [activation] present in test data point [True]
14 Text feature [enhanced] present in test data point [True]
18 Text feature [transforming] present in test data point [True]
22 Text feature [oncogene] present in test data point [True]
23 Text feature [epithelial] present in test data point [True]
27 Text feature [interestingly] present in test data point [True]
29 Text feature [3a] present in test data point [True]
37 Text feature [free] present in test data point [True]
43 Text feature [somatic] present in test data point [True]
45 Text feature [3t3] present in test data point [True]
46 Text feature [signals] present in test data point [True]
48 Text feature [pathways] present in test data point [True]
53 Text feature [recurrent] present in test data point [True]
55 Text feature [nucleotide] present in test data point [True]
59 Text feature [example] present in test data point [True]
60 Text feature [sites] present in test data point [True]
63 Text feature [pathway] present in test data point [True]
70 Text feature [leukemia] present in test data point [True]
73 Text feature [residue] present in test data point [True]
74 Text feature [occur] present in test data point [True]
77 Text feature [oncogenic] present in test data point [True]
79 Text feature [s3] present in test data point [True]
81 Text feature [inhibited] present in test data point [True]
84 Text feature [recently] present in test data point [True]
91 Text feature [state] present in test data point [True]
98 Text feature [total] present in test data point [True]
102 Text feature [fact] present in test data point [True]
103 Text feature [observations] present in test data point [True]
104 Text feature [frequent] present in test data point [True]
109 Text feature [cluster] present in test data point [True]
```

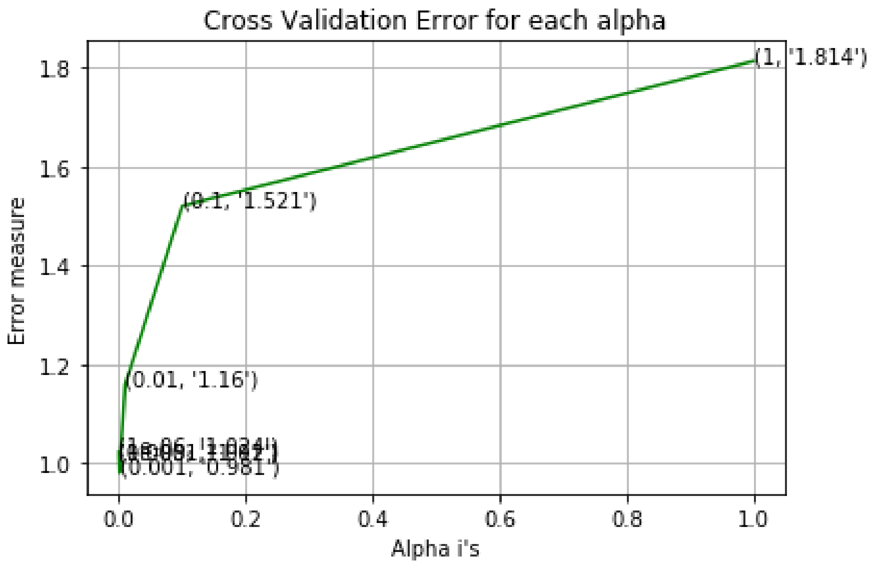
```
110 Text feature [specimens] present in test data point [True]
112 Text feature [per] present in test data point [True]
117 Text feature [position] present in test data point [True]
120 Text feature [proliferation] present in test data point [True]
124 Text feature [mean] present in test data point [True]
129 Text feature [mm] present in test data point [True]
137 Text feature [signaling] present in test data point [True]
138 Text feature [distinct] present in test data point [True]
140 Text feature [expressing] present in test data point [True]
141 Text feature [mice] present in test data point [True]
142 Text feature [mapk] present in test data point [True]
148 Text feature [therapeutic] present in test data point [True]
154 Text feature [effective] present in test data point [True]
158 Text feature [go] present in test data point [True]
160 Text feature [potential] present in test data point [True]
164 Text feature [gtp] present in test data point [True]
165 Text feature [association] present in test data point [True]
166 Text feature [residues] present in test data point [True]
170 Text feature [higher] present in test data point [True]
171 Text feature [harboring] present in test data point [True]
172 Text feature [ras] present in test data point [True]
175 Text feature [inhibitor] present in test data point [True]
181 Text feature [culture] present in test data point [True]
184 Text feature [mutants] present in test data point [True]
188 Text feature [mutated] present in test data point [True]
190 Text feature [nras] present in test data point [True]
195 Text feature [antibodies] present in test data point [True]
198 Text feature [tumors] present in test data point [True]
199 Text feature [lymphoma] present in test data point [True]
200 Text feature [factor] present in test data point [True]
203 Text feature [demonstrated] present in test data point [True]
207 Text feature [differentiation] present in test data point [True]
209 Text feature [22] present in test data point [True]
212 Text feature [driven] present in test data point [True]
217 Text feature [reverse] present in test data point [True]
220 Text feature [frequently] present in test data point [True]
226 Text feature [properties] present in test data point [True]
229 Text feature [positive] present in test data point [True]
230 Text feature [increased] present in test data point [True]
234 Text feature [five] present in test data point [True]
```

```
236 Text feature [likely] present in test data point [True]
238 Text feature [isolated] present in test data point [True]
241 Text feature [conditions] present in test data point [True]
242 Text feature [bound] present in test data point [True]
244 Text feature [lines] present in test data point [True]
248 Text feature [alk] present in test data point [True]
251 Text feature [kras] present in test data point [True]
264 Text feature [survival] present in test data point [True]
273 Text feature [next] present in test data point [True]
274 Text feature [present] present in test data point [True]
277 Text feature [manner] present in test data point [True]
278 Text feature [note] present in test data point [True]
282 Text feature [cdna] present in test data point [True]
288 Text feature [various] present in test data point [True]
299 Text feature [resulting] present in test data point [True]
316 Text feature [nm] present in test data point [True]
319 Text feature [mechanism] present in test data point [True]
320 Text feature [progression] present in test data point [True]
322 Text feature [lung] present in test data point [True]
327 Text feature [suggest] present in test data point [True]
334 Text feature [colonies] present in test data point [True]
346 Text feature [kinase] present in test data point [True]
369 Text feature [constitutively] present in test data point [True]
392 Text feature [cases] present in test data point [True]
394 Text feature [tumor] present in test data point [True]
403 Text feature [raf] present in test data point [True]
417 Text feature [presence] present in test data point [True]
430 Text feature [inhibition] present in test data point [True]
478 Text feature [sirna] present in test data point [True]
481 Text feature [affinity] present in test data point [True]
483 Text feature [targets] present in test data point [True]
491 Text feature [level] present in test data point [True]
496 Text feature [ratio] present in test data point [True]
498 Text feature [biochemical] present in test data point [True]
Out of the top  500  features  110 are present in query point
```

LogisticRegression(without weights) after feature engineering

In [422]:
```
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_fe, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding_fe, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding_fe)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCoding_fe)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, la
bels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding_fe)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predi
ct_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding_fe)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labe
ls=clf.classes_, eps=1e-15))
```

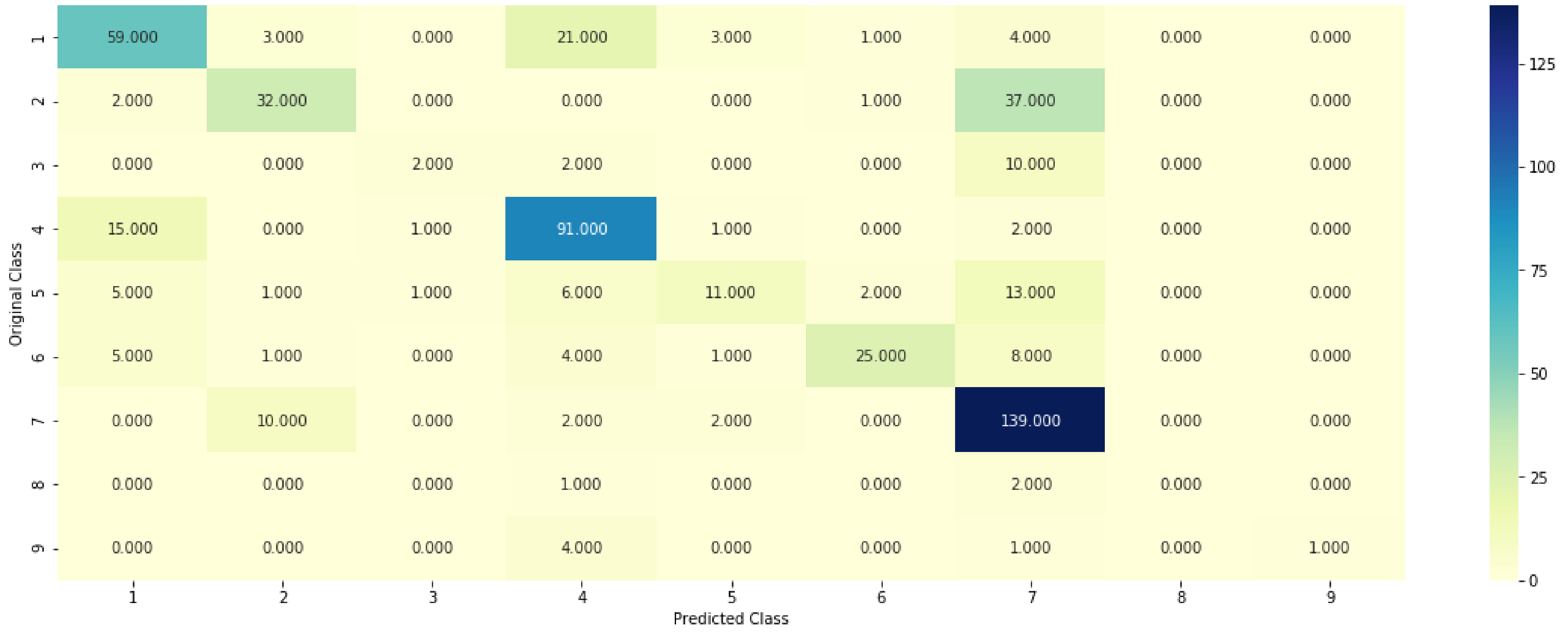
```
for alpha = 1e-06
Log Loss : 1.0239923267625084
for alpha = 1e-05
Log Loss : 1.0119790720300836
for alpha = 0.0001
Log Loss : 1.0101154293338257
for alpha = 0.001
Log Loss : 0.9812590877789893
for alpha = 0.01
Log Loss : 1.160055391040069
for alpha = 0.1
Log Loss : 1.5211459550265043
for alpha = 1
Log Loss : 1.813667376377312
```



```
For values of best alpha =  0.001 The train log loss is: 0.6406720827911725
For values of best alpha =  0.001 The cross validation log loss is: 0.9812590877789893
For values of best alpha =  0.001 The test log loss is: 1.012898899301796
```

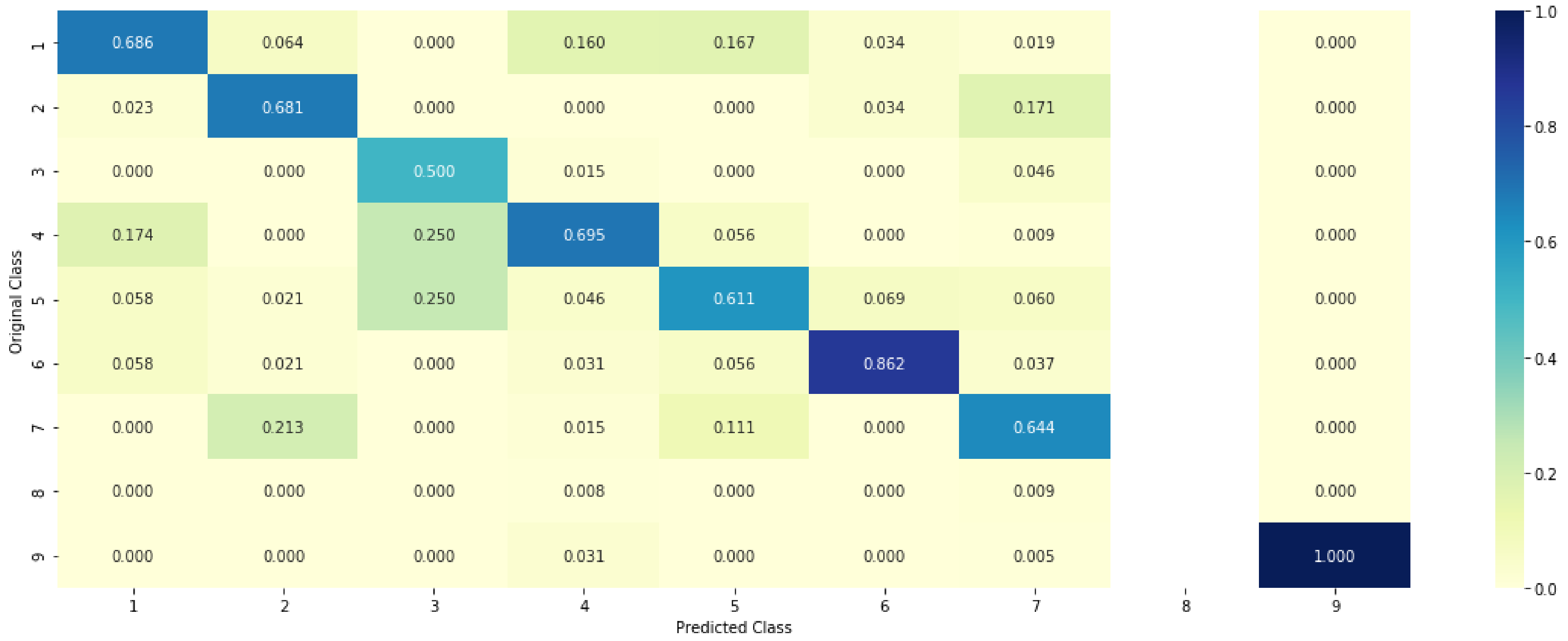
In [423]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding_fe, train_y, cv_x_onehotCoding_fe, cv_y, clf)
```

```
Log loss : 0.98125908777789893
Number of mis-classified points : 0.3233082706766917
-------------------- Confusion matrix --------------------
```



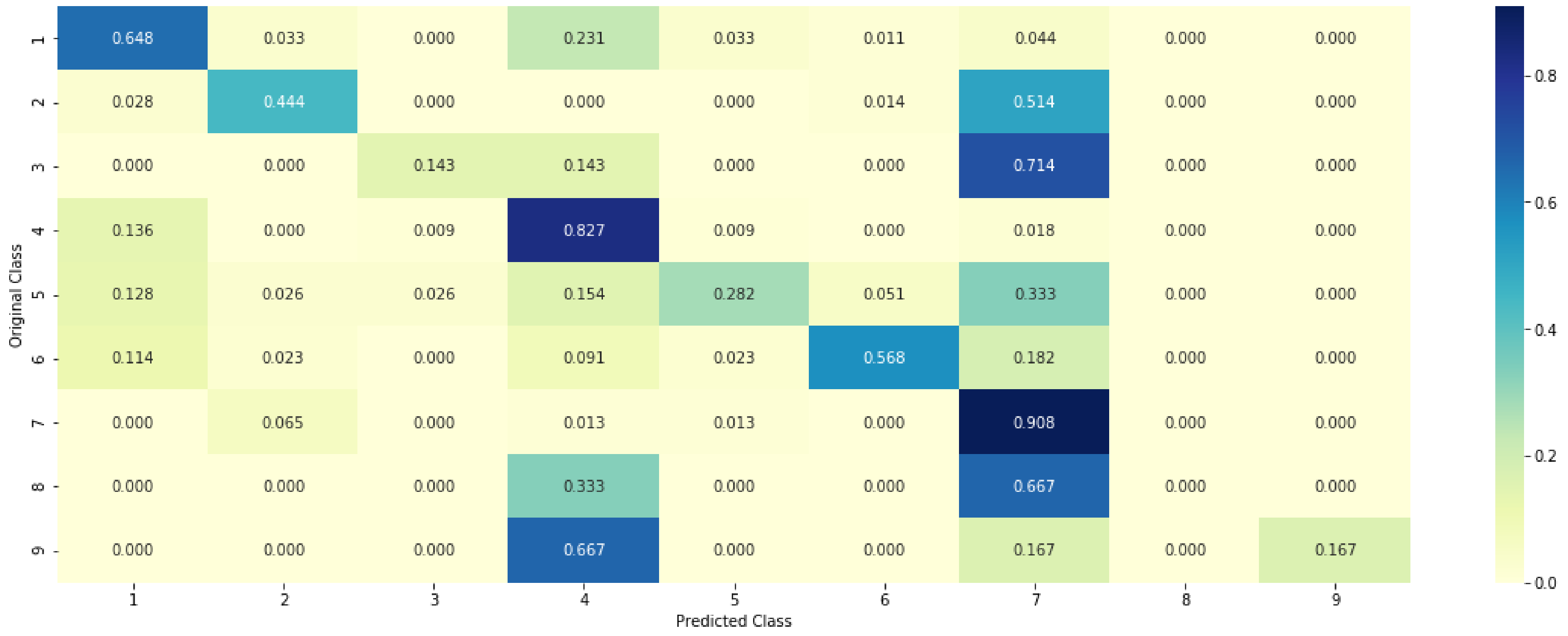
	1	2	3	4	5	6	7	8	9
1	59.000	3.000	0.000	21.000	3.000	1.000	4.000	0.000	0.000
2	2.000	32.000	0.000	0.000	0.000	1.000	37.000	0.000	0.000
3	0.000	0.000	2.000	2.000	0.000	0.000	10.000	0.000	0.000
4	15.000	0.000	1.000	91.000	1.000	0.000	2.000	0.000	0.000
5	5.000	1.000	1.000	6.000	11.000	2.000	13.000	0.000	0.000
6	5.000	1.000	0.000	4.000	1.000	25.000	8.000	0.000	0.000
7	0.000	10.000	0.000	2.000	2.000	0.000	139.000	0.000	0.000
8	0.000	0.000	0.000	1.000	0.000	0.000	2.000	0.000	0.000
9	0.000	0.000	0.000	4.000	0.000	0.000	1.000	0.000	1.000

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



	1	2	3	4	5	6	7	8	9
1	0.686	0.064	0.000	0.160	0.167	0.034	0.019		0.000
2	0.023	0.681	0.000	0.000	0.000	0.034	0.171		0.000
3	0.000	0.000	0.500	0.015	0.000	0.000	0.046		0.000
4	0.174	0.000	0.250	0.695	0.056	0.000	0.009		0.000
5	0.058	0.021	0.250	0.046	0.611	0.069	0.060		0.000
6	0.058	0.021	0.000	0.031	0.056	0.862	0.037		0.000
7	0.000	0.213	0.000	0.015	0.111	0.000	0.644		0.000
8	0.000	0.000	0.000	0.008	0.000	0.000	0.009		0.000
9	0.000	0.000	0.000	0.031	0.000	0.000	0.005		1.000

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



Feature Importance, Incorrectly Classified point

In [424]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding_fe,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_fe[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding_fe[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],t
est_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 6
Predicted Class Probabilities: [[0.0031 0.0824 0.0045 0.0042 0.0056 0.7997 0.0945 0.005  0.001 ]]
Actual Class : 6
--------------------------------------------------
6 Text feature [blue] present in test data point [True]
10 Text feature [binds] present in test data point [True]
11 Text feature [mek1] present in test data point [True]
14 Text feature [significant] present in test data point [True]
15 Text feature [selection] present in test data point [True]
20 Text feature [breast] present in test data point [True]
21 Text feature [frequent] present in test data point [True]
23 Text feature [observation] present in test data point [True]
26 Text feature [substrate] present in test data point [True]
29 Text feature [family] present in test data point [True]
31 Text feature [regulated] present in test data point [True]
39 Text feature [development] present in test data point [True]
47 Text feature [decreased] present in test data point [True]
52 Text feature [loss] present in test data point [True]
55 Text feature [values] present in test data point [True]
56 Text feature [inhibitors] present in test data point [True]
60 Text feature [suppression] present in test data point [True]
70 Text feature [identified] present in test data point [True]
76 Text feature [therapeutic] present in test data point [True]
80 Text feature [conformation] present in test data point [True]
84 Text feature [mek] present in test data point [True]
91 Text feature [cause] present in test data point [True]
92 Text feature [targeted] present in test data point [True]
94 Text feature [phospho] present in test data point [True]
101 Text feature [altered] present in test data point [True]
105 Text feature [right] present in test data point [True]
111 Text feature [members] present in test data point [True]
114 Text feature [inhibition] present in test data point [True]
117 Text feature [helix] present in test data point [True]
118 Text feature [use] present in test data point [True]
119 Text feature [gel] present in test data point [True]
120 Text feature [concentration] present in test data point [True]
125 Text feature [structures] present in test data point [True]
126 Text feature [important] present in test data point [True]
129 Text feature [thus] present in test data point [True]
130 Text feature [evidence] present in test data point [True]
```

```
133 Text feature [length] present in test data point [True]
136 Text feature [variant] present in test data point [True]
139 Text feature [phase] present in test data point [True]
140 Text feature [biochemical] present in test data point [True]
143 Text feature [enzyme] present in test data point [True]
147 Text feature [groups] present in test data point [True]
149 Text feature [involved] present in test data point [True]
153 Text feature [red] present in test data point [True]
155 Text feature [potential] present in test data point [True]
157 Text feature [frequency] present in test data point [True]
159 Text feature [resistance] present in test data point [True]
160 Text feature [bind] present in test data point [True]
165 Text feature [expected] present in test data point [True]
168 Text feature [distinct] present in test data point [True]
174 Text feature [region] present in test data point [True]
179 Text feature [mechanism] present in test data point [True]
180 Text feature [44] present in test data point [True]
181 Text feature [group] present in test data point [True]
183 Text feature [src] present in test data point [True]
188 Text feature [40] present in test data point [True]
190 Text feature [model] present in test data point [True]
197 Text feature [factors] present in test data point [True]
198 Text feature [substitution] present in test data point [True]
199 Text feature [lower] present in test data point [True]
202 Text feature [express] present in test data point [True]
205 Text feature [dose] present in test data point [True]
210 Text feature [structural] present in test data point [True]
219 Text feature [well] present in test data point [True]
222 Text feature [time] present in test data point [True]
223 Text feature [substitutions] present in test data point [True]
227 Text feature [leading] present in test data point [True]
229 Text feature [targeting] present in test data point [True]
231 Text feature [left] present in test data point [True]
236 Text feature [relevant] present in test data point [True]
238 Text feature [1a] present in test data point [True]
239 Text feature [predicted] present in test data point [True]
241 Text feature [nm] present in test data point [True]
242 Text feature [normal] present in test data point [True]
246 Text feature [60] present in test data point [True]
249 Text feature [clinically] present in test data point [True]
```

```
255 Text feature [kinase] present in test data point [True]
256 Text feature [jak2] present in test data point [True]
258 Text feature [remains] present in test data point [True]
259 Text feature [driver] present in test data point [True]
260 Text feature [direct] present in test data point [True]
264 Text feature [whole] present in test data point [True]
265 Text feature [receptors] present in test data point [True]
266 Text feature [second] present in test data point [True]
270 Text feature [state] present in test data point [True]
271 Text feature [kinases] present in test data point [True]
272 Text feature [low] present in test data point [True]
273 Text feature [catalytic] present in test data point [True]
275 Text feature [free] present in test data point [True]
277 Text feature [confer] present in test data point [True]
280 Text feature [proliferation] present in test data point [True]
281 Text feature [value] present in test data point [True]
282 Text feature [factor] present in test data point [True]
284 Text feature [43] present in test data point [True]
286 Text feature [hydrogen] present in test data point [True]
289 Text feature [variants] present in test data point [True]
294 Text feature [45] present in test data point [True]
295 Text feature [multiple] present in test data point [True]
296 Text feature [interactions] present in test data point [True]
297 Text feature [reported] present in test data point [True]
298 Text feature [receptor] present in test data point [True]
302 Text feature [structure] present in test data point [True]
304 Text feature [ic50] present in test data point [True]
307 Text feature [proportion] present in test data point [True]
310 Text feature [days] present in test data point [True]
312 Text feature [site] present in test data point [True]
315 Text feature [probe] present in test data point [True]
316 Text feature [hours] present in test data point [True]
317 Text feature [suggests] present in test data point [True]
319 Text feature [single] present in test data point [True]
320 Text feature [would] present in test data point [True]
321 Text feature [interaction] present in test data point [True]
323 Text feature [since] present in test data point [True]
325 Text feature [clones] present in test data point [True]
327 Text feature [residues] present in test data point [True]
329 Text feature [measured] present in test data point [True]
```

```
331 Text feature [greater] present in test data point [True]
335 Text feature [determine] present in test data point [True]
337 Text feature [known] present in test data point [True]
343 Text feature [mechanisms] present in test data point [True]
345 Text feature [provide] present in test data point [True]
348 Text feature [individual] present in test data point [True]
351 Text feature [mice] present in test data point [True]
356 Text feature [grade] present in test data point [True]
357 Text feature [conserved] present in test data point [True]
359 Text feature [atp] present in test data point [True]
364 Text feature [using] present in test data point [True]
371 Text feature [first] present in test data point [True]
372 Text feature [chemotherapy] present in test data point [True]
374 Text feature [active] present in test data point [True]
377 Text feature [egfr] present in test data point [True]
379 Text feature [four] present in test data point [True]
383 Text feature [seven] present in test data point [True]
388 Text feature [associated] present in test data point [True]
389 Text feature [il] present in test data point [True]
390 Text feature [initial] present in test data point [True]
392 Text feature [studies] present in test data point [True]
394 Text feature [ability] present in test data point [True]
395 Text feature [relatively] present in test data point [True]
397 Text feature [previous] present in test data point [True]
401 Text feature [whether] present in test data point [True]
403 Text feature [concentrations] present in test data point [True]
405 Text feature [genetic] present in test data point [True]
409 Text feature [image] present in test data point [True]
422 Text feature [bound] present in test data point [True]
423 Text feature [17] present in test data point [True]
427 Text feature [experiments] present in test data point [True]
428 Text feature [secondary] present in test data point [True]
436 Text feature [sequencing] present in test data point [True]
440 Text feature [set] present in test data point [True]
441 Text feature [essential] present in test data point [True]
446 Text feature [analyses] present in test data point [True]
455 Text feature [20] present in test data point [True]
457 Text feature [strong] present in test data point [True]
459 Text feature [indeed] present in test data point [True]
461 Text feature [sites] present in test data point [True]
```

```
465 Text feature [experimental] present in test data point [True]
467 Text feature [11] present in test data point [True]
474 Text feature [number] present in test data point [True]
476 Text feature [via] present in test data point [True]
477 Text feature [combination] present in test data point [True]
480 Text feature [survival] present in test data point [True]
481 Text feature [therapy] present in test data point [True]
482 Text feature [interestingly] present in test data point [True]
487 Text feature [loop] present in test data point [True]
490 Text feature [transformation] present in test data point [True]
492 Text feature [novel] present in test data point [True]
493 Text feature [cellular] present in test data point [True]
496 Text feature [due] present in test data point [True]
497 Text feature [4b] present in test data point [True]
Out of the top  500  features  170 are present in query point
```

Incorrectly Classified point

In [425]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding_fe[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding_fe[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],t
est_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0083 0.0399 0.0043 0.0024 0.0427 0.0185 0.876  0.0068 0.0011]]
Actual Class : 7
--------------------------------------------------
1 Text feature [activating] present in test data point [True]
3 Text feature [3b] present in test data point [True]
5 Text feature [constitutive] present in test data point [True]
7 Text feature [activate] present in test data point [True]
8 Text feature [activation] present in test data point [True]
13 Text feature [downstream] present in test data point [True]
14 Text feature [transforming] present in test data point [True]
15 Text feature [enhanced] present in test data point [True]
22 Text feature [activated] present in test data point [True]
23 Text feature [3a] present in test data point [True]
25 Text feature [free] present in test data point [True]
31 Text feature [oncogene] present in test data point [True]
34 Text feature [epithelial] present in test data point [True]
35 Text feature [signals] present in test data point [True]
37 Text feature [example] present in test data point [True]
45 Text feature [somatic] present in test data point [True]
46 Text feature [interestingly] present in test data point [True]
49 Text feature [sites] present in test data point [True]
51 Text feature [recently] present in test data point [True]
58 Text feature [pathway] present in test data point [True]
59 Text feature [residue] present in test data point [True]
63 Text feature [3t3] present in test data point [True]
65 Text feature [inhibited] present in test data point [True]
67 Text feature [pathways] present in test data point [True]
69 Text feature [nucleotide] present in test data point [True]
71 Text feature [occur] present in test data point [True]
72 Text feature [observations] present in test data point [True]
79 Text feature [recurrent] present in test data point [True]
82 Text feature [state] present in test data point [True]
92 Text feature [mm] present in test data point [True]
94 Text feature [fact] present in test data point [True]
99 Text feature [s3] present in test data point [True]
102 Text feature [total] present in test data point [True]
108 Text feature [position] present in test data point [True]
115 Text feature [cluster] present in test data point [True]
124 Text feature [oncogenic] present in test data point [True]
```

```
125 Text feature [five] present in test data point [True]
127 Text feature [leukemia] present in test data point [True]
133 Text feature [per] present in test data point [True]
137 Text feature [distinct] present in test data point [True]
141 Text feature [frequent] present in test data point [True]
143 Text feature [effective] present in test data point [True]
144 Text feature [increased] present in test data point [True]
145 Text feature [higher] present in test data point [True]
146 Text feature [mean] present in test data point [True]
147 Text feature [residues] present in test data point [True]
149 Text feature [antibodies] present in test data point [True]
152 Text feature [proliferation] present in test data point [True]
159 Text feature [specimens] present in test data point [True]
160 Text feature [differentiation] present in test data point [True]
170 Text feature [go] present in test data point [True]
171 Text feature [expressing] present in test data point [True]
175 Text feature [mutants] present in test data point [True]
177 Text feature [lymphoma] present in test data point [True]
184 Text feature [potential] present in test data point [True]
185 Text feature [presence] present in test data point [True]
193 Text feature [22] present in test data point [True]
194 Text feature [therapeutic] present in test data point [True]
195 Text feature [inhibitor] present in test data point [True]
198 Text feature [bound] present in test data point [True]
200 Text feature [mice] present in test data point [True]
203 Text feature [harboring] present in test data point [True]
206 Text feature [mutated] present in test data point [True]
210 Text feature [likely] present in test data point [True]
212 Text feature [positive] present in test data point [True]
219 Text feature [association] present in test data point [True]
230 Text feature [factor] present in test data point [True]
242 Text feature [culture] present in test data point [True]
248 Text feature [tumor] present in test data point [True]
249 Text feature [gtp] present in test data point [True]
252 Text feature [manner] present in test data point [True]
263 Text feature [mapk] present in test data point [True]
275 Text feature [tumors] present in test data point [True]
277 Text feature [alk] present in test data point [True]
281 Text feature [frequently] present in test data point [True]
299 Text feature [conditions] present in test data point [True]
```

```
307 Text feature [ratio] present in test data point [True]
309 Text feature [signaling] present in test data point [True]
312 Text feature [nras] present in test data point [True]
319 Text feature [suggest] present in test data point [True]
336 Text feature [1a] present in test data point [True]
339 Text feature [ras] present in test data point [True]
346 Text feature [90] present in test data point [True]
353 Text feature [isolated] present in test data point [True]
381 Text feature [kras] present in test data point [True]
382 Text feature [demonstrated] present in test data point [True]
384 Text feature [mechanism] present in test data point [True]
385 Text feature [cdna] present in test data point [True]
393 Text feature [various] present in test data point [True]
406 Text feature [properties] present in test data point [True]
408 Text feature [driven] present in test data point [True]
411 Text feature [four] present in test data point [True]
430 Text feature [luciferase] present in test data point [True]
436 Text feature [note] present in test data point [True]
438 Text feature [would] present in test data point [True]
441 Text feature [present] present in test data point [True]
442 Text feature [reverse] present in test data point [True]
444 Text feature [next] present in test data point [True]
446 Text feature [nm] present in test data point [True]
448 Text feature [resulting] present in test data point [True]
450 Text feature [melanomas] present in test data point [True]
458 Text feature [lines] present in test data point [True]
481 Text feature [fig] present in test data point [True]
493 Text feature [progression] present in test data point [True]
Out of the top  500  features  104 are present in query point
```

LogisticRegression with unigrams and bigrams

In [426]:
```
vectorizer = CountVectorizer(min_df=3, ngram_range=(1,2))

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

In [427]:
```
train_variation_feature_onehotCoding = vectorizer.fit_transform(train_df["Variation"])
test_variation_feature_onehotCoding = vectorizer.transform(test_df["Variation"])
cv_variation_feature_onehotCoding = vectorizer.transform(cv_df["Variation"])
```

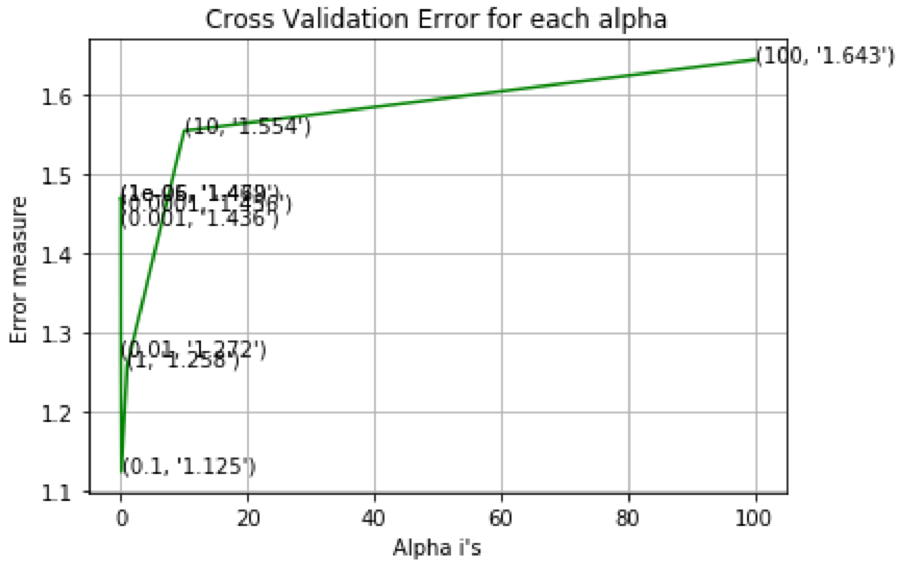
In [428]:
```
train_text_feature_onehotCoding = vectorizer.fit_transform(train_df['TEXT'])

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

In [429]:
```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding))

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

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

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

In [430]:
```
train_x_onehotCoding_fe = np.sqrt(train_x_onehotCoding)
test_x_onehotCoding_fe = np.sqrt(test_x_onehotCoding)
cv_x_onehotCoding_fe = np.sqrt(cv_x_onehotCoding)
```

In [431]:
```
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_fe, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding_fe, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding_fe)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

predict_y = sig_clf.predict_proba(train_x_onehotCoding_fe)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y, la
bels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding_fe)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predi
ct_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding_fe)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y, labe
ls=clf.classes_, eps=1e-15))
```

```
for alpha = 1e-06
Log Loss : 1.468613218209733
for alpha = 1e-05
Log Loss : 1.4701568266267726
for alpha = 0.0001
Log Loss : 1.4556461216987016
for alpha = 0.001
Log Loss : 1.4363615888173689
for alpha = 0.01
Log Loss : 1.271678430433688
for alpha = 0.1
Log Loss : 1.1246473490200204
for alpha = 1
Log Loss : 1.258298158709285
for alpha = 10
Log Loss : 1.5536668345332583
for alpha = 100
Log Loss : 1.6430558990659025
```



```
For values of best alpha =  0.1 The train log loss is: 0.8152007401037437
For values of best alpha =  0.1 The cross validation log loss is: 1.1246473490200204
For values of best alpha =  0.1 The test log loss is: 1.1340428361629944
```

Observation:

- By looking into the above performed models LogisticRegression with balanced weights is giving very less logloss value.
- The Logloss value is less than 1.

In [ ]:
```
 0.6406720827911725
For values of best alpha =  0.001 The cross validation log loss is: 0.9812590877789893
For values of best alpha =  0.001 The test log loss is: 1.012898899301796
```

In [434]:
```
from prettytable import PrettyTable
table = PrettyTable()
table.field_names = ["model","hyperparameter","train_score","cv_score","test_score","missclasfied_points"]
table.add_row(["NavieBayes(Baseline model)","0.001","0.536","1.147","1.171","0.368"])
table.add_row(["K-NearestNeighbour","11","0.636","0.989","1.001","0.340"])
table.add_row(["LogisticRegression(Balancedweights)","0.0001","0.464","0.961","0.974","0.327"])
table.add_row(["LogisticRegression(without BalancedWeights)","0.0001","0.452","1.011","1.003","0.347"])
table.add_row(["LinearSVM","0.0001","0.481","1.025","1.008","0.328"])
table.add_row(["RandomForest","n_estimators=2000,max_depth=5","0.866","1.184","1.171","0.404"])
table.add_row(["RandomForest with response coding","n_estimators=10,max_depth=5","0.073","1.295","1.396","0.456"
])
table.add_row(["Stacking LR,NB,SVM models","0.1","0.551","1.132","1.141","0.366"])
table.add_row(["VottingClassifier"," ","0.849","1.159","1.173","0.368"])
table.add_row(["LR with balanced weights after FE","0.001","0.638","0.956","0.998","0.321"])
table.add_row(["LR without balanced weights after FE","0.001","0.640","0.981","1.012","0.323"])
table.add_row(["LR with unigrams,bigrams","0.1","0.815","1.124","1.134",""])
print(table)
```

```
+----------------------------------------------+--------------------------------+-------------+-----------+--------
----+---------------------+
|                    model                     |         hyperparameter         | train_score | cv_score | test_sc
ore | missclasfied_points |
+----------------------------------------------+--------------------------------+-------------+-----------+--------
----+---------------------+
|          NavieBayes(Baseline model)          |             0.001              |    0.536    |  1.147   |   1.171
|        0.368        |
|              K-NearestNeighbour              |               11               |    0.636    |  0.989   |   1.001
|        0.340        |
|     LogisticRegression(Balancedweights)      |             0.0001             |    0.464    |  0.961   |   0.974
|        0.327        |
| LogisticRegression(without BalancedWeights) |             0.0001             |    0.452    |  1.011   |   1.003
|        0.347        |
|                  LinearSVM                   |             0.0001             |    0.481    |  1.025   |   1.008
|        0.328        |
|                 RandomForest                 | n_estimators=2000,max_depth=5 |    0.866    |  1.184   |   1.171
|        0.404        |
|      RandomForest with response coding       |  n_estimators=10,max_depth=5  |    0.073    |  1.295   |   1.396
|        0.456        |
|          Stacking LR,NB,SVM models           |              0.1               |    0.551    |  1.132   |   1.141
|        0.366        |
|              VottingClassifier               |                                |    0.849    |  1.159   |   1.173
|        0.368        |
|      LR with balanced weights after FE       |             0.001              |    0.638    |  0.956   |   0.998
|        0.321        |
|     LR without balanced weights after FE     |             0.001              |    0.640    |  0.981   |   1.012
|        0.323        |
|           LR with unigrams,bigrams           |              0.1               |    0.815    |  1.124   |   1.134
|                     |
+----------------------------------------------+--------------------------------+-------------+-----------+--------
----+---------------------+
```

- From the above table we can see that LogisticRegression with balanced weights is performing well compared to all the other models and it has interpretablity. As we can see in the table the gap between train and cv scores is more thus it may leads to overfittng.
- After doning feature engineering, LR with balanced weights turned out well as the gap between train and cv is less by keeping the logloss less than 1.

Summary

step-by-step procedure:

- Read the data in dataframe.
- Merge text and variants data.
- Split the train,cv,test data in the ratio 64:16:20.
- Analyze each and every feature to check what is the impotance of the feature and how well the data is distributed into train,test,cv. To check that use LogisticRegression as a model.
- Finally, take NavieBayes as a baseline model and apply various models to check with model is performing well.
- Table all the log loss values for each models to check which model is performed well.
- In my case SGDClassifier with logloss performed well, applied feature engineered the sparse matrix of train,test and cv.
- It turned out well the loss is reduced lessthan 1.
- Then tried with unigrams and bigrams which turned not well.