Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [2]:

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

from mlxtend.classifier import StackingClassifier

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

```
/home/chitra/.local/lib/python3.6/site-packages/matplotlib/__init__.py:886:
MatplotlibDeprecationWarning:
examples.directory is deprecated; in the future, examples will be found relative to the 'datapath'
directory.
  "found relative to the 'datapath' directory.".format(key))
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [3]:

```
data = pd.read_csv('training/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[3]:

	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 [4]:

```
# note the seprator in this file
data_text =pd.read_csv("training/training_text",sep="\|\|",engine="python",names=["ID","TEXT"],skip
rows=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[4]:

	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 [5]:

```
# 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 [6]:

```
#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 : 171.472309 seconds
```

In [7]:

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

Out[7]:

	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 [8]:

```
result.shape
```

Out[8]:

```
(3321, 5)
```

In [9]:

```
result.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3321 entries, 0 to 3320
Data columns (total 5 columns):
ID           3321 non-null int64
Gene         3321 non-null object
Variation    3321 non-null object
Class        3321 non-null int64
TEXT         3316 non-null object
dtypes: int64(2), object(3)
memory usage: 155.7+ KB
```

In [10]:

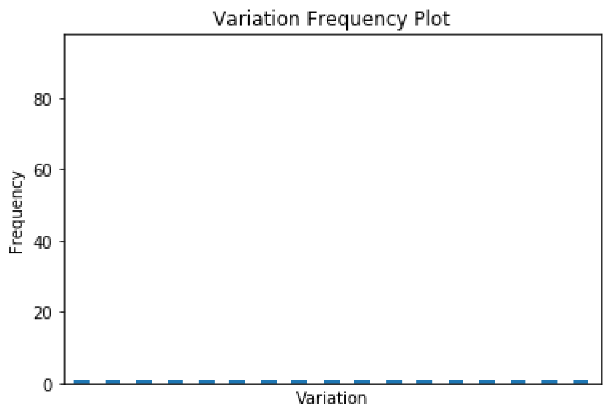
```
result["Variation"].describe()
```

Out[10]:

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

In [11]:

```
## gene frequency plot
plt.figure()
ax = result['Variation'].value_counts().plot(kind='bar')

ax.get_xaxis().set_ticks([])
ax.set_title('Variation Frequency Plot')
ax.set_xlabel('Variation')
ax.set_ylabel('Frequency')

plt.show()
```



Observation: Variation contains maximum independent unique values.

In [12]:

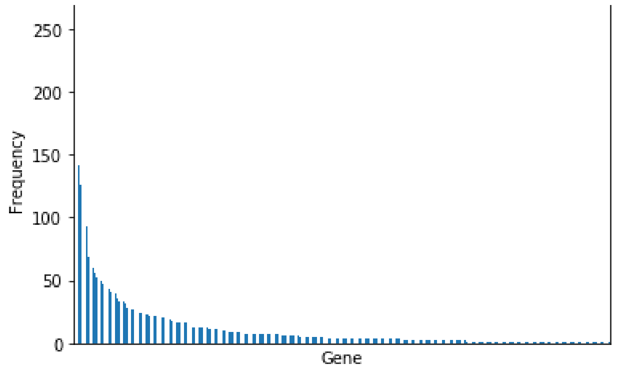
```
result["Gene"].describe()
```

Out[12]:

```
count      3321
unique      264
top       BRCA1
freq        264
Name: Gene, dtype: object
```

In [13]:

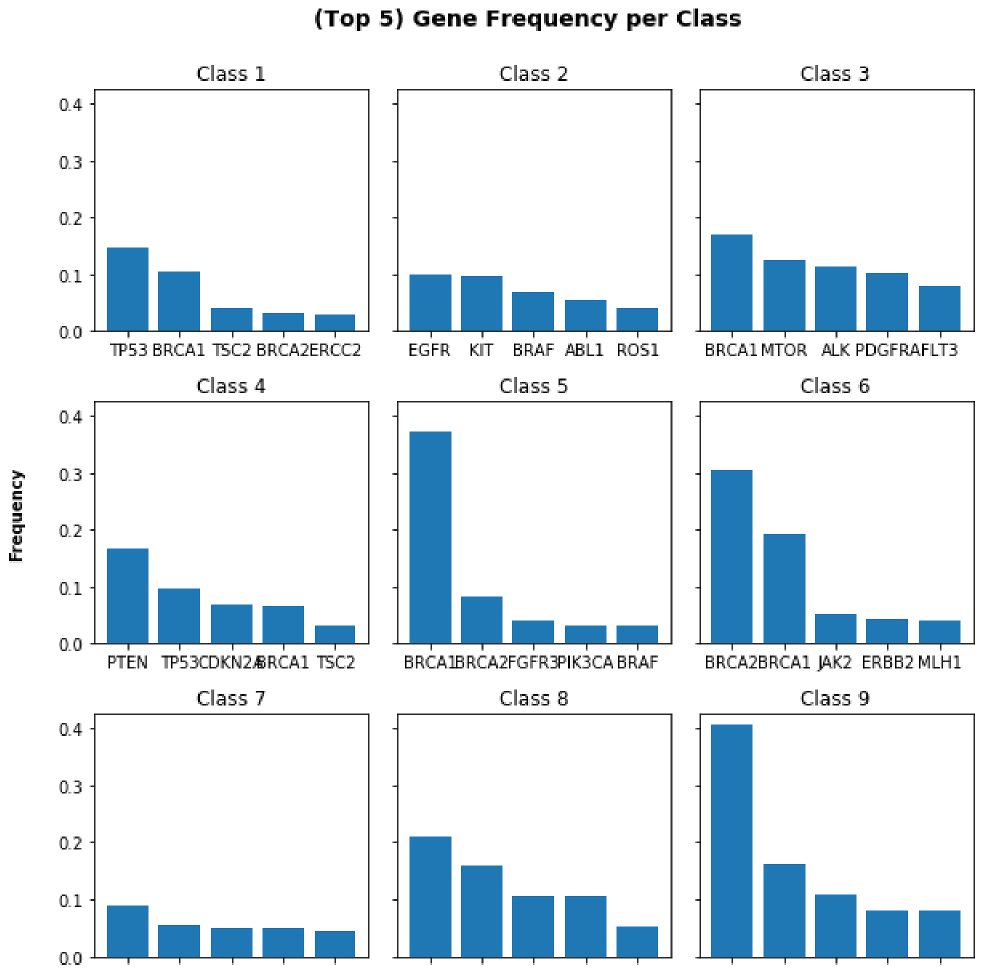
```
## gene frequency plot
plt.figure()
ax = result['Gene'].value_counts().plot(kind='bar')

ax.get_xaxis().set_ticks([])
ax.set_title('Gene Frequency Plot')
ax.set_xlabel('Gene')
ax.set_ylabel('Frequency')

plt.show()
```

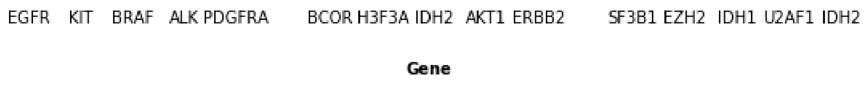


Observation: Gene contains minimum independent unique values.

```
In [15]:
```

```
fig, axes = plt.subplots(nrows=3, ncols=3, sharey=True, figsize=(9,9))

# Normalize value counts for better comparison
def normalize_group(x):
    label, repetition = x.index, x
    t = sum(repetition)
    r = [n/t for n in repetition]
    return label, r

for idx, g in enumerate(result.groupby('Class')):
    label, val = normalize_group(g[1]["Gene"].value_counts())
    ax = axes.flat[idx]
    ax.bar(np.arange(5), val[:5],
           tick_label=label[:5])
    ax.set_title("Class {}".format(g[0]))

fig.text(0.5, 0.97, '(Top 5) Gene Frequency per Class', ha='center', fontsize=14, fontweight='bold'
)
fig.text(0.5, 0, 'Gene', ha='center', fontweight='bold')
fig.text(0, 0.5, 'Frequency', va='center', rotation='vertical', fontweight='bold')
fig.tight_layout(rect=[0.03, 0.03, 0.95, 0.95])
```





In [16]:

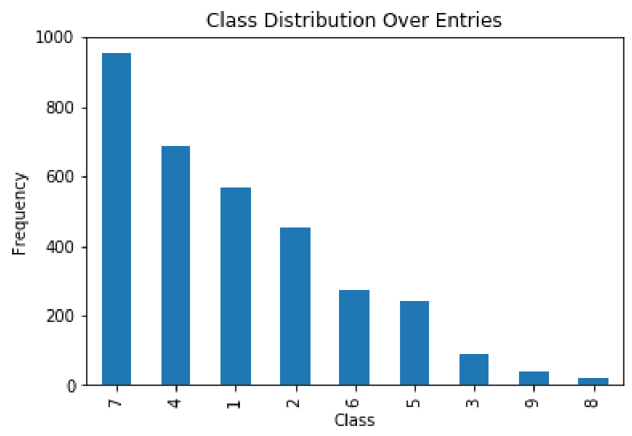
```
result.Class.unique()
```

Out[16]:

```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [17]:

```
#### class distribution
plt.figure()
ax = result['Class'].value_counts().plot(kind='bar')

ax.set_title('Class Distribution Over Entries')
ax.set_xlabel('Class')
ax.set_ylabel('Frequency')

plt.show()
```



Observation: Class 8 and class 9 have less examples.

In [18]:

```
result[result.isnull().any(axis=1)]
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

In [19]:

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

Out[19]:

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

3.1.4. Test, Train and Cross Validation Split

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

In [10]:

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

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

In [11]:

```
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])
```

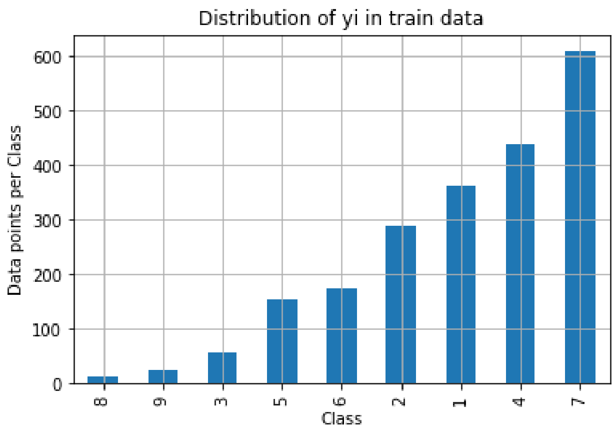
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

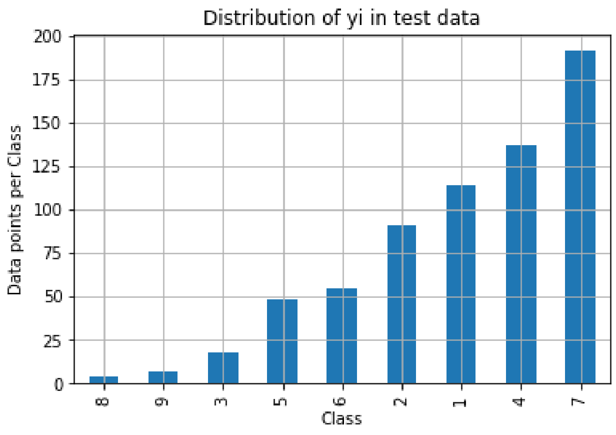
In [12]:

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

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.ro
und((train_class_distribution.values[i]/train_df.shape[0]*100), 3), '%)')


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

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

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
```

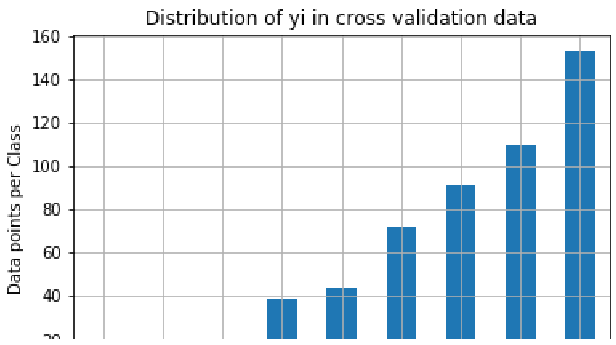
```
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.values[i], '(', np.round
((cv_class_distribution.values[i]/cv_df.shape[0]*100), 3), '%)')
```

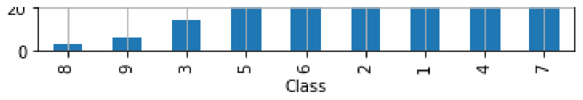


```
Number of data points in class 9 : 609 ( 28.672 %)
Number of data points in class 8 : 439 ( 20.669 %)
Number of data points in class 7 : 363 ( 17.09 %)
Number of data points in class 6 : 289 ( 13.606 %)
Number of data points in class 5 : 176 ( 8.286 %)
Number of data points in class 4 : 155 ( 7.298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



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





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

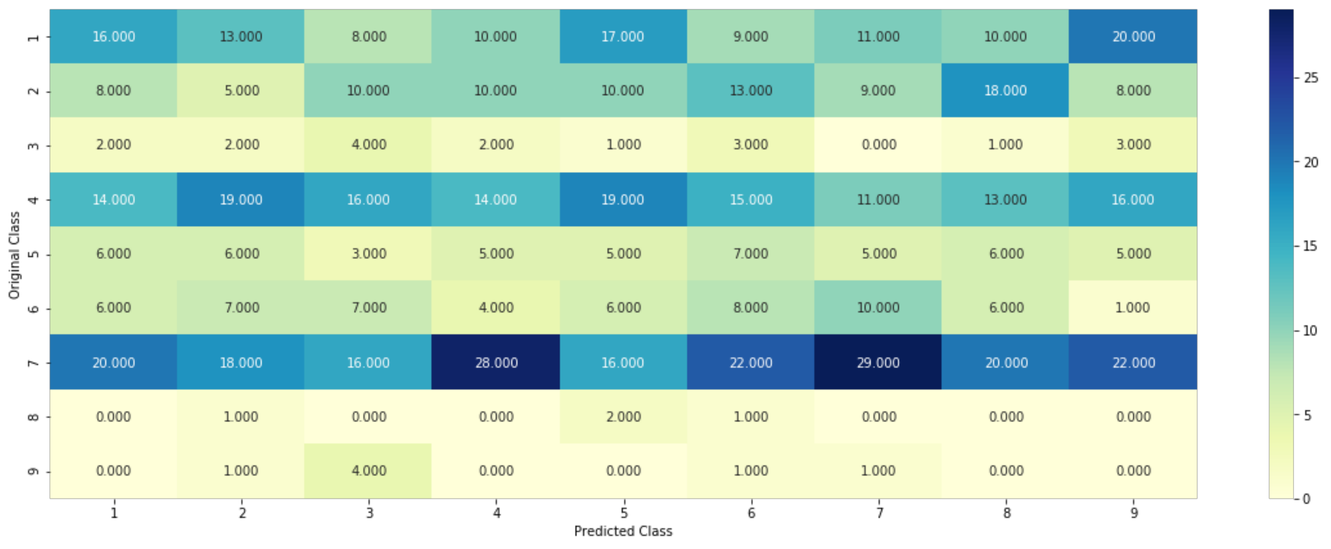
In [13]:

```
# 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()
```

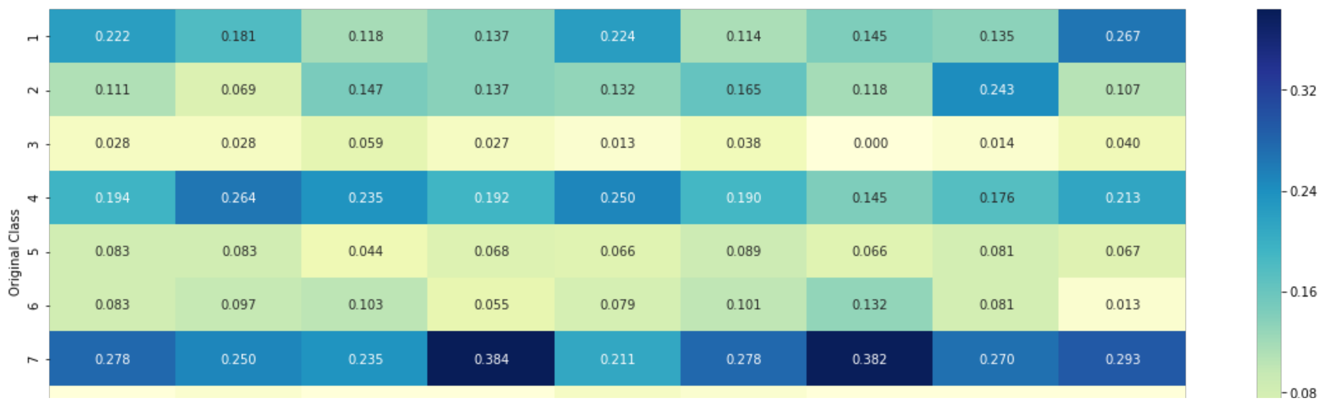
In [14]:

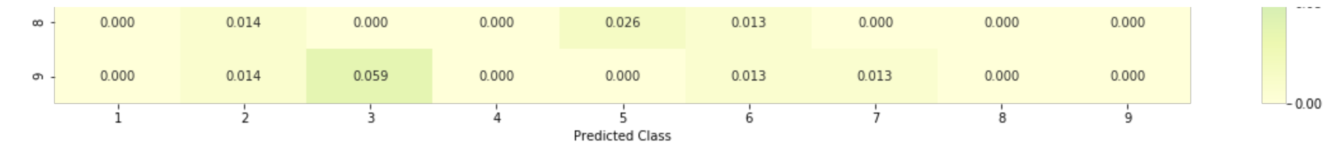
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# 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 2.4981758364316256
Log loss on Test Data using Random Model 2.4950372749043987
-------------------- Confusion matrix --------------------
```

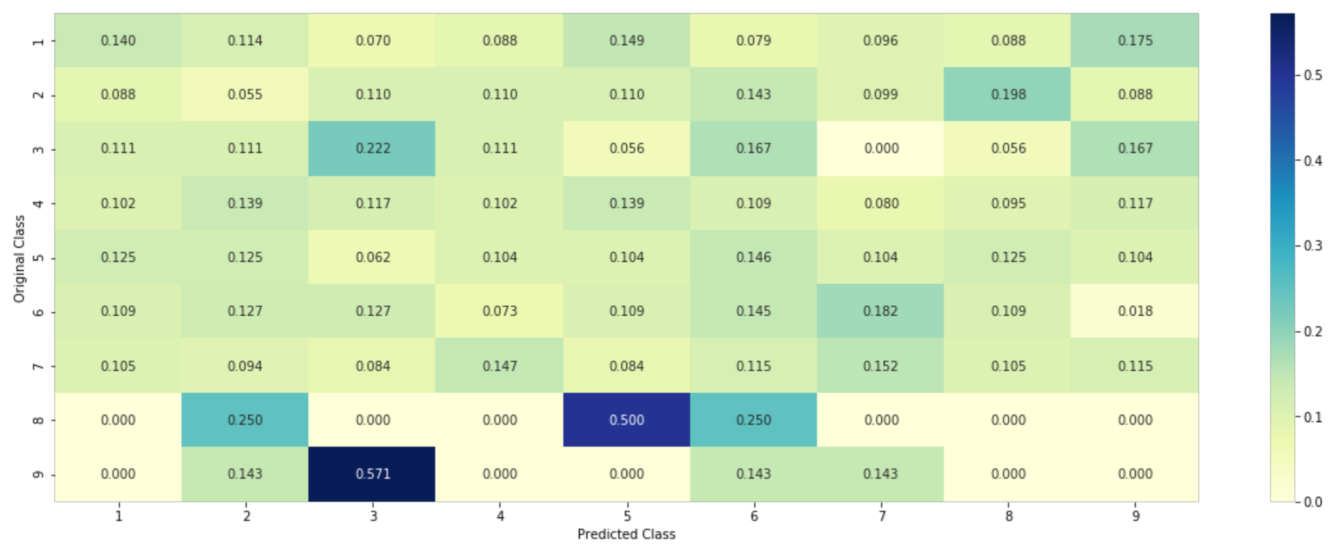


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



3.3 Univariate Analysis

In [15]:

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

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

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

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

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

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

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

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

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

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

In [16]:

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

```
Number of Unique Genes : 234
BRCA1     164
TP53      113
EGFR       83
BRCA2      78
PTEN       77
KIT        67
BRAF       54
ERBB2      45
ALK        43
PDGFRA     40
Name: Gene, dtype: int64
```

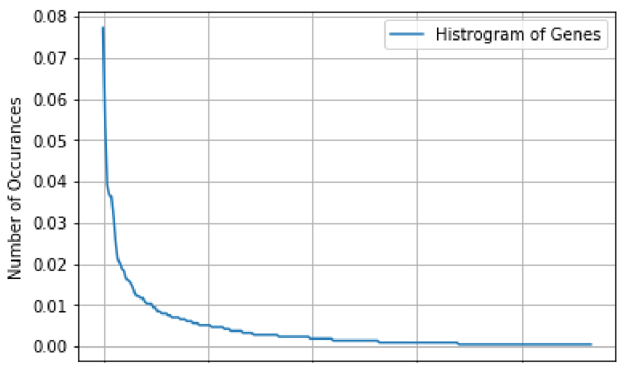
In [17]:

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

```
Ans: There are 234 different categories of genes in the train data, and they are distibuted as fol
lows
```

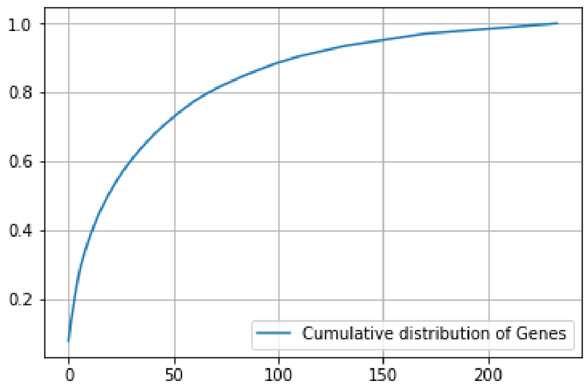
In [18]:

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



In [19]:

```
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 [20]:

```
#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 [21]:

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

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

In [22]:

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

In [23]:

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

Out[23]:

```
2701      BRAF
3137      KRAS
2585     BRCA1
683     CDKN2A
1426     FGFR3
Name: Gene, dtype: object
```

In [24]:

```
gene_vectorizer.get_feature_names()
```

Out[24]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl2',
 'atm',
 'atrx',
 'aurka',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcl2',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd2',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk6',
 'cdk8',
 'cdkn1a',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cebpa',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctnnb1',
 'ddr2',
 'dicer1',
 'dnmt3a',
 'dnmt3b',
 'dusp4',
 'egfr',
 'elf3',
 'ep300',
 'epas1',
 'epcam',
 'erbb2',
 'erbb3',
 'erbb4',
 'ercc2',
 'ercc3',
 'ercc4',
 'erg',
 'errfi1',
 'esr1',
 'etv1',
 'etv6',
 'ewsr1',
 'ezh2',
 'fanca',
 'fat1',
 'fbxw7',
 'fgf19',
 'fgf3',
 'fgfr1',
 'fgfr2',
 'fgfr3',
 'fgfr4',
 'flt1',
 'flt3',
 'foxa1',
 'foxl2',
 'foxo1',
 'foxp1',
 'fubp1',
 'gata3',
 'gli1',
 'gna11',
 'gnas',
 'h3f3a',
 'hist1h1c',
 'hla',
 'hnf1a',
 'hras',
 'idh1',
 'idh2',
 'igf1r',
 'ikzf1',
 'il7r',
 'jak1',
 'jak2',
 'kdm5a',
 'kdm5c',
 'kdm6a',
 'kdr',
 'keap1',
 'kit',
 'klf4',
 'kmt2a',
 'kmt2c',
 'kmt2d',
 'knstrn',
 'kras',
 'lats1',
 'lats2',
 'map2k1',
 'map2k2',
 'map2k4',
 'map3k1',
 'mapk1',
 'mdm2',
 'mdm4',
 'med12',
 'mef2b',
 'men1',
 'met',
 'mlh1',
 'mpl',
 'msh2',
 'msh6',
 'mtor',
 'myc',
 'mycn',
 'myd88',
 'myod1',
 'ncor1',
 'nf1',
 'nf2',
 'nfe2l2',
 'nfkbia',
 'nkx2',
 'notch1',
 'notch2',
 'npm1',
 'nras',
 'nsd1',
 'ntrk1',
 'ntrk2',
 'ntrk3',
 'nup93',
 'pak1',
 'pax8',
 'pbrm1',
 'pdgfra',
 'pdgfrb',
 'pik3ca',
 'pik3cb',
 'pik3cd',
 'pik3r1',
 'pik3r2',
 'pim1',
 'pms1',
 'pms2',
 'pole',
 'ppp2r1a',
 'ppp6c',
 'prdm1',
 'ptch1',
 'pten',
 'ptpn11',
 'ptprd',
 'ptprt',
 'rab35',
 'rac1',
 'rad21',
 'rad50',
 'rad51b',
 'rad51c',
 'raf1',
 'rara',
 'rasa1',
 'rb1',
 'rbm10',
 'ret',
 'rheb',
 'rhoa',
 'rit1',
 'ros1',
 'rras2',
 'runx1',
 'rxra',
 'rybp',
 'setd2',
 'sf3b1',
 'shoc2',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'srsf2',
 'stag2',
 'stat3',
 'stk11',
 'tcf3',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'whsc1',
 'whsc1l1',
 'xpo1',
 'yap1']
```

In [25]:

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of g
ene feature: (2124, 233)
```

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

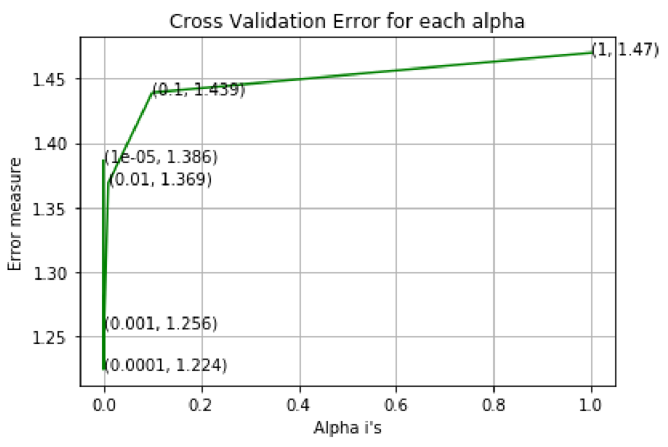
There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

In [26]:

```
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.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=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.clas
ses_, eps=1e-15))

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

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.386115214066996
For values of alpha =  0.0001 The log loss is: 1.223823320901138
For values of alpha =  0.001 The log loss is: 1.256482076017744
For values of alpha =  0.01 The log loss is: 1.3688601513821663
For values of alpha =  0.1 The log loss is: 1.4390610118982288
For values of alpha =  1 The log loss is: 1.4701674548050543
```



```
For values of best alpha =  0.0001 The train log loss is: 1.0395584472478763
For values of best alpha =  0.0001 The cross validation log loss is: 1.223823320901138
For values of best alpha =  0.0001 The test log loss is: 1.1662474420899918
```

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 [27]:

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

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  234  genes in train dataset?
Ans
1. In test data 646 out of 665 : 97.14285714285714
2. In cross validation data 517 out of  532 : 97.18045112781954
```

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 [28]:

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

```
Number of Unique Variations : 1920
Truncating_Mutations     60
Deletion                 52
Amplification            51
Fusions                  24
Overexpression            4
S222D                     2
TMPRSS2-ETV1_Fusion       2
G12A                      2
E542K                     2
E330K                     2
Name: Variation, dtype: int64
```

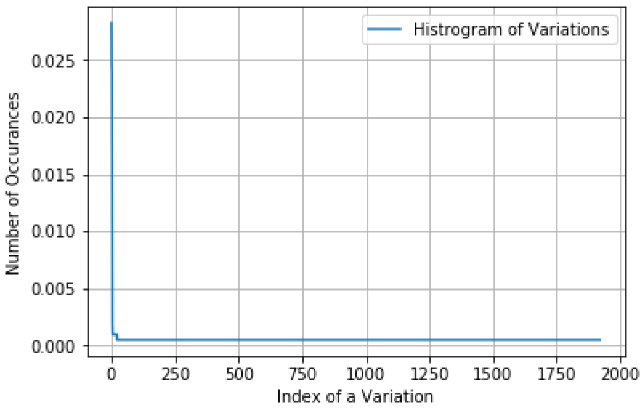
In [29]:

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

```
Ans: There are 1920 different categories of variations in the train data, and they are distibuted
as follows
```

In [30]:

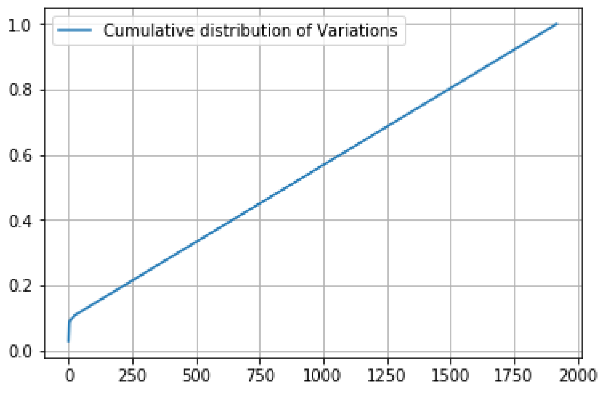
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [31]:

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

```
[0.02824859 0.0527307  0.076742   ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [32]:

```
# 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 [33]:

```
print("train_variation_feature_responseCoding is a converted feature using the response coding met
hod. The shape of Variation feature:", train_variation_feature_responseCoding.shape)
```

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

In [34]:

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

In [35]:

```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding meth
od. The shape of Variation feature:", train_variation_feature_onehotCoding.shape)
```

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The
shape of Variation feature: (2124, 1957)
```

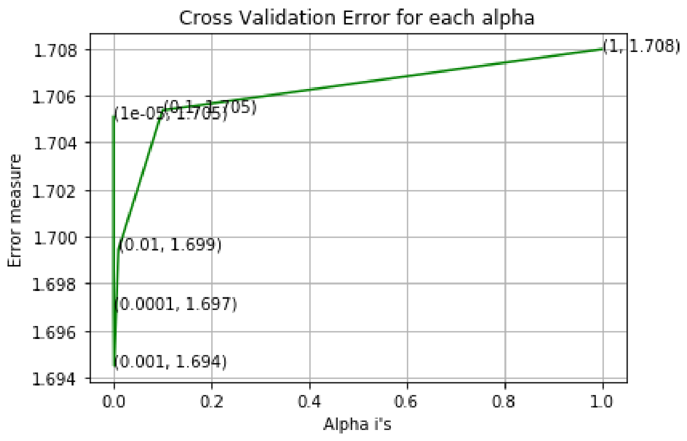
Q10. How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

In [36]:

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

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=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.clas
ses_, eps=1e-15))

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.7050910671569042
For values of alpha =  0.0001 The log loss is: 1.6969291943425067
For values of alpha =  0.001 The log loss is: 1.6944917304621145
For values of alpha =  0.01 The log loss is: 1.699453853566857
For values of alpha =  0.1 The log loss is: 1.7053670842932749
For values of alpha =  1 The log loss is: 1.7079626211260135
```



```
For values of best alpha =  0.001 The train log loss is: 1.131143494992806
For values of best alpha =  0.001 The cross validation log loss is: 1.6944917304621145
For values of best alpha =  0.001 The test log loss is: 1.7346765985485555
```

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 [37]:

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

```
Q12. How many data points are covered by total  1920  genes in test and cross validation data
sets?
Ans
1. In test data 61 out of 665 : 9.172932330827068
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 [38]:

```
# 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 [39]:

```
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 [40]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
from sklearn.feature_extraction.text import TfidfVectorizer
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)
train_text_features= text_vectorizer.get_feature_names()

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

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


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

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

In [41]:

```
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 [42]:

```
#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 [43]:

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

```
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_responseCoding.
sum(axis=1)).T
```

In [44]:

```
# 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 [45]:

```
#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 [46]:

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

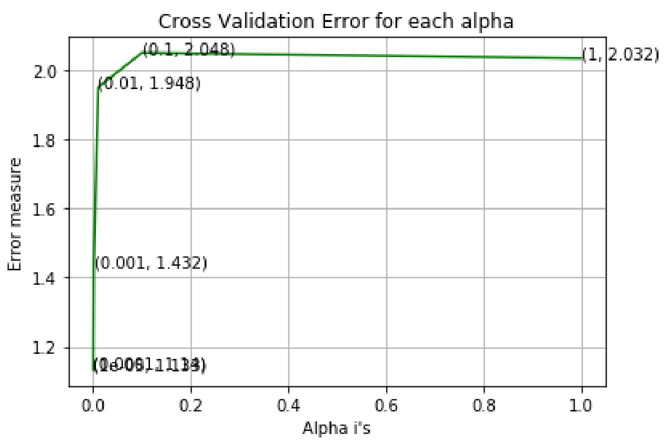
```
Counter({250.72892012740058: 1, 173.02654475783947: 1, 132.83511692489736: 1, 132.51742854093322:
1, 126.67472562947667: 1, 115.4972703462556: 1, 115.1516914380618: 1, 112.9882995857589: 1,
106.61117001544575: 1, 106.60202279942602: 1, 104.8642242353772: 1, 90.90300830733217: 1,
89.28140818886048: 1, 87.51559193259833: 1, 80.74177598295067: 1, 77.98733912227273: 1,
77.90805768730543: 1, 77.7016152593318: 1, 77.05252442567259: 1, 75.60544459010744: 1,
75.02149499445662: 1, 74.52520289784971: 1, 70.23458330686752: 1, 69.36512809005168: 1,
66.99246526039165: 1, 66.66894712694376: 1, 66.59858096700779: 1, 65.78921853781755: 1,
63.6069040469669: 1, 62.922053940753635: 1, 62.89245605494868: 1, 62.87654350048146: 1,
62.03299713932592: 1, 60.95070397971135: 1, 58.382412294514744: 1, 56.644365475565756: 1,
56.337136337365735: 1, 55.986787428209716: 1, 53.69216677927344: 1, 52.04756458641293: 1,
50.79667894226454: 1, 48.72811606908703: 1, 48.1268799597331: 1, 47.998823653415485: 1,
47.66698653661733: 1, 46.88938819417133: 1, 46.67715959372354: 1, 46.39658125327522: 1,
44.93323896635889: 1, 44.73893608908213: 1, 44.465196236374055: 1, 44.31306700395054: 1,
43.01381813161508: 1, 42.950137784541326: 1, 42.4174918031115: 1, 42.25063905763198: 1,
42.17004518216131: 1, 41.53575077506047: 1, 41.346898781176044: 1, 41.32273718544018: 1,
41.282150583152145: 1, 41.123774888416364: 1, 40.94458896270867: 1, 40.89435367455665: 1,
40.746912222419645: 1, 40.59258482141445: 1, 40.14338702667497: 1, 39.637651182793356: 1,
39.223224323840675: 1, 39.16460823584628: 1, 39.16230402419611: 1, 38.81088032328349: 1, 37.8840012
4925128: 1, 37.81209719518143: 1, 37.31366533012048: 1, 36.62312664078101: 1, 36.48044930059815: 1,
36.03328230502792: 1, 35.97082147905315: 1, 35.522614896414744: 1, 35.02363509294514: 1,
34.90789134051607: 1, 34.338827580948696: 1, 34.28118998603672: 1, 34.22775406528479: 1,
34.198042313744594: 1, 34.18996776392416: 1, 33.89938311962212: 1, 33.590064171295694: 1,
32.978896581729444: 1, 32.87225783540174: 1, 32.64280675732358: 1, 32.41799740963321: 1, 32.2227842
7359146: 1, 31.980997471538124: 1, 31.770632890681885: 1, 31.726131477734544: 1,
31.62157924672979: 1, 31.567085905211957: 1, 31.519968246807586: 1, 31.503550159186954: 1,
31.434805036174435: 1, 31.399710395998405: 1, 31.38758491018343: 1, 31.34478797778394: 1,
31.252835982492137: 1, 30.994438394386673: 1, 30.936429837808614: 1, 30.84615446963033: 1,
30.68248610755983: 1, 30.58426664366885: 1, 30.557670659108922: 1, 30.519013233554674: 1,
30.397270761756708: 1, 30.0879739526453: 1, 30.062333680709894: 1, 29.986433005455353: 1,
29.930611712960246: 1, 29.76032644969494: 1, 29.702373766776613: 1, 29.554095967229717: 1,
29.348313250712327: 1, 29.24751349389275: 1, 29.195960164271053: 1, 29.076505505950568: 1,
29.03327500551699: 1, 28.834103058887006: 1, 28.614274405224737: 1, 28.542245249048893: 1,
28.401205998235074: 1, 28.395111392436274: 1, 28.36810298572516: 1, 28.336578625509553: 1,
27.988877038621137: 1, 27.963089113850934: 1, 27.49019941733742: 1, 27.305775266693384: 1,
26.917376462483457: 1, 26.78662322914078: 1, 26.751848529281673: 1, 26.696881393877565: 1,
26.667245960877523: 1, 26.638082499336736: 1, 26.59217377914485: 1, 26.493726221470624: 1,
26.346583500275617: 1, 26.260555797815957: 1, 26.197947075109372: 1, 25.99508079587591: 1,
25.962781620470828: 1, 25.917007967321453: 1, 25.809085648827008: 1, 25.78655493413791: 1,
25.582060122957557: 1, 25.211776119101923: 1, 25.182516758917952: 1, 25.099574894885947: 1,
24.819012525645253: 1, 24.74371941307247: 1, 24.678046799283287: 1, 24.62360892309767: 1,
24.615421063085563: 1, 24.556378642250994: 1, 24.47011056270515: 1, 24.33192549426153: 1,
24.11667104252579: 1, 24.002220099835714: 1, 23.95282825660231: 1, 23.9483856206389: 1,
23.91677168227274: 1, 23.67604485733478: 1, 23.61488622955364: 1, 23.59583818991423: 1,
23.51273069516064: 1, 23.435692455867795: 1, 23.37461374166828: 1, 23.239471399918077: 1,
23.13843467403239: 1, 23.051747342554087: 1, 22.966591416004814: 1, 22.9567437709388: 1,
22.926442584243805: 1, 22.926366812114406: 1, 22.91146789338073: 1, 22.870137026543695: 1,
22.7582320244653: 1, 22.688691122244702: 1, 22.647694134919796: 1, 22.57702872770281: 1,
22.505932025937835: 1, 22.486070143557065: 1, 22.457298785735336: 1, 22.343350517596413: 1,
22.329520702907505: 1, 22.265738860946918: 1, 22.256423546684378: 1, 22.22450513201834: 1,
22.202666216815583: 1, 22.186647199296296: 1, 22.183308310695615: 1, 22.135072175023847: 1,
22.060937949117434: 1, 22.020016990336583: 1, 21.98633019704359: 1, 21.949831168027686: 1,
21.941106347763235: 1, 21.872670554224513: 1, 21.807938628828374: 1, 21.740315539008996: 1,
21.67775477183256: 1, 21.676880186808443: 1, 21.66022808944805: 1, 21.52595555269898: 1,
21.4810404318854: 1, 21.449379605619246: 1, 21.394886124543447: 1, 21.379996300366887: 1,
21.300571703550716: 1, 21.23682528494671: 1, 21.166320015506038: 1, 21.153281897260467: 1,
21.139116629905292: 1, 21.124833111477535: 1, 21.12270940694842: 1, 21.078436956540017: 1,
21.066392212891675: 1, 20.999378314971295: 1, 20.963744607425678: 1, 20.879472746879916: 1,
20.806045337222194: 1, 20.731266027460688: 1, 20.626578937380692: 1, 20.624633684745483: 1,
20.612071926823653: 1, 20.541917473051104: 1, 20.48504115593196: 1, 20.473843588683987: 1,
20.394428520590566: 1, 20.365994641889923: 1, 20.351959722875158: 1, 20.310964832051397: 1,
20.10694337998202: 1, 20.060220217381755: 1, 20.057111998746905: 1, 20.042123504935525: 1,
19.92532012749812: 1, 19.774122393652018: 1, 19.7562567270541: 1, 19.72820858767237: 1,
19.668026675265583: 1, 19.577730929010844: 1, 19.504395749140116: 1, 19.50367861965839: 1,
19.482061058733237: 1, 19.431300823164957: 1, 19.425706528836084: 1, 19.41908205017532: 1,
19.35421498453597: 1, 19.32817540089041: 1, 19.299830072679264: 1, 19.28351897408968: 1,
19.219660695307507: 1, 19.13984551753521: 1, 19.098840569065754: 1, 19.09797763993735: 1,
19.089843996939805: 1, 19.083030717880646: 1, 18.974599187356247: 1, 18.875290389252683: 1,
18.86237842560734: 1, 18.837745148157214: 1, 18.82438106443291: 1, 18.82050035941562: 1,
18.811638401840934: 1, 18.78545912687123: 1, 18.777985671244902: 1, 18.7758897986611: 1,
18.773689776912057: 1, 18.77298420225855: 1, 18.72799922116098: 1, 18.703566387697734: 1,
18.68790911383825: 1, 18.68631945250023: 1, 18.603645000499355: 1, 18.572954334857794: 1,
18.568092808856147: 1, 18.514426512526136: 1, 18.510447271768456: 1, 18.49997905719071: 1,
18.37317947370149: 1, 18.365382334237683: 1, 18.350314982065502: 1, 18.288572676473283: 1,
18.26111515737223: 1, 18.251816114859547: 1, 18.153073377803974: 1, 18.105449376589664: 1,
18.089245159873965: 1, 18.085283551562725: 1, 17.964064673541642: 1, 17.912463565557534: 1,
17.907978344394856: 1, 17.85315123451996: 1, 17.752420435432708: 1, 17.648579302706196: 1,
17.64741298638044: 1, 17.621971455694155: 1, 17.60950196735289: 1, 17.597712790089894: 1,
17.538977209811208: 1, 17.460564201953936: 1, 17.432037745537293: 1, 17.423841433882572: 1,
17.415858105555646: 1, 17.405872487200774: 1, 17.404624937274967: 1, 17.34673544725411: 1,
17.24465657924252: 1, 17.238455877641286: 1, 17.197330681916586: 1, 17.19688421086158: 1,
17.191896189404027: 1, 17.17077150666667: 1, 17.167679881970372: 1, 17.1661251356079: 1,
17.14255549097211: 1, 17.13284138208226: 1, 17.103568389779976: 1, 17.091962074709866: 1,
17.083524209719926: 1, 17.082690407574848: 1, 17.07325852123292: 1, 17.07169640164028: 1,
17.041887962196775: 1, 17.035538070085376: 1, 17.02029159976989: 1, 17.007042064222304: 1,
16.909457016664913: 1, 16.86348592873118: 1, 16.81196789717608: 1, 16.795180888853174: 1,
16.785539812757623: 1, 16.756099855805914: 1, 16.703749139861625: 1, 16.664770490047832: 1,
16.634438355223466: 1, 16.573365139599954: 1, 16.560552651924546: 1, 16.529118480248105: 1,
16.50999685148673: 1, 16.50444054229648: 1, 16.49389929147697: 1, 16.48406330970097: 1,
16.470492476930385: 1, 16.442516385847917: 1, 16.357178762007905: 1, 16.33618698767071: 1,
16.316544548391402: 1, 16.276677381461678: 1, 16.226697855831326: 1, 16.22034553574188: 1,
16.160995653158995: 1, 16.1447340813142: 1, 16.140803347190012: 1, 16.12050248367105: 1,
16.089969840269777: 1, 16.081666753194455: 1, 16.058710715295806: 1, 16.05308887647753: 1,
16.051376048871695: 1, 16.03670211888805: 1, 16.007897459749394: 1, 16.002246364670004: 1,
15.979798155095414: 1, 15.96434217044256: 1, 15.958195294531363: 1, 15.932766889186842: 1,
15.879432995942734: 1, 15.85216543073452: 1, 15.850665525475959: 1, 15.809393949647365: 1,
15.764805529292552: 1, 15.733452014059845: 1, 15.676659382576595: 1, 15.62483985839969: 1,
15.582316324017665: 1, 15.543155273781014: 1, 15.538207138015357: 1, 15.516484618282696: 1,
15.478792249275962: 1, 15.463390681483597: 1, 15.45024941306209: 1, 15.442760818195026: 1,
15.414162266415405: 1, 15.399066956618853: 1, 15.382229253813815: 1, 15.356082228147484: 1,
15.316323020520008: 1, 15.275499489676356: 1, 15.254539691439904: 1, 15.250806041526442: 1,
15.22667905301481: 1, 15.20867723832659: 1, 15.176528037811853: 1, 15.172026974173447: 1,
15.127311976887613: 1, 15.112495298768387: 1, 15.106040606191245: 1, 15.073234669186663: 1,
15.071746871503082: 1, 15.049133250310307: 1, 15.025455426384944: 1, 14.993088743451974: 1,
14.979269616122235: 1, 14.971731755769266: 1, 14.971479587799404: 1, 14.951002188315098: 1,
14.929343533918408: 1, 14.89115102140043: 1, 14.89070249786333: 1, 14.885221237140847: 1,
14.868137442319288: 1, 14.858458829151333: 1, 14.81050312265224: 1, 14.75847615226466: 1,
14.723514871811554: 1, 14.719454871864304: 1, 14.709820444746594: 1, 14.708449023690829: 1,
14.639382971595904: 1, 14.608375763176626: 1, 14.584917602398866: 1, 14.582153860441869: 1,
14.569884982022172: 1, 14.53935829270886: 1, 14.486927242942949: 1, 14.474302867524106: 1,
14.465233675078556: 1, 14.456537096649548: 1, 14.449475450331905: 1, 14.448108728597735: 1,
14.44561046395975: 1, 14.44082760873599: 1, 14.407822057645298: 1, 14.377167099120294: 1,
14.368957927211694: 1, 14.355637220990932: 1, 14.354205947542603: 1, 14.311133539446432: 1,
14.302810778616752: 1, 14.287132834187465: 1, 14.27770452027496: 1, 14.265208829238226: 1,
14.210011426178925: 1, 14.207974116898614: 1, 14.204816837008458: 1, 14.18823616892243: 1,
14.182997893358587: 1, 14.175036358636065: 1, 14.147066179615702: 1, 14.128944941167005: 1,
14.07805266421375: 1, 14.053155378236141: 1, 13.994260479444707: 1, 13.940463410658738: 1,
13.932317302154376: 1, 13.915445432757451: 1, 13.896576469432166: 1, 13.820064858177192: 1,
13.785016070364547: 1, 13.783857522671468: 1, 13.781575038867452: 1, 13.779181169276848: 1,
13.731939910184511: 1, 13.731350445532062: 1, 13.72623304625936: 1, 13.725200033311795: 1,
13.664946141316234: 1, 13.629756744799563: 1, 13.62070336404959: 1, 13.617036200516145: 1,
13.591069391863217: 1, 13.555310936171105: 1, 13.54960924867181: 1, 13.483481771522625: 1,
13.467909032283808: 1, 13.409786659846295: 1, 13.408782829469178: 1, 13.380301931345526: 1,
13.377367495591377: 1, 13.360035987107713: 1, 13.309525860663415: 1, 13.291140710856665: 1,
13.284885825422272: 1, 13.248818627895236: 1, 13.178337867798529: 1, 13.16365967137824: 1,
13.148002446829668: 1, 13.146050182117486: 1, 13.136896456463406: 1, 13.095691583754402: 1,
13.084902129919248: 1, 13.082098822611073: 1, 13.047595337547943: 1, 13.013406377995766: 1,
12.981135639809796: 1, 12.953449106304351: 1, 12.94253728336158: 1, 12.940389095157496: 1,
12.909942305470445: 1, 12.876879355478808: 1, 12.837453392266843: 1, 12.812652911244705: 1,
12.807749574719445: 1, 12.806751570916015: 1, 12.78461402721252: 1, 12.770379349769739: 1,
12.710502229640202: 1, 12.685078087464976: 1, 12.66789561115512: 1, 12.649124758371084: 1,
12.641792495943257: 1, 12.625220995653992: 1, 12.592147034396735: 1, 12.569519352503725: 1,
12.555909299305062: 1, 12.526451783062681: 1, 12.503835924378278: 1, 12.492933683755485: 1,
12.477745703267734: 1, 12.46404113257104: 1, 12.457888729579174: 1, 12.440534458827313: 1,
12.424302612792092: 1, 12.419374634893869: 1, 12.407502014856142: 1, 12.38690072877005: 1,
12.374634330040566: 1, 12.341093174903966: 1, 12.338902014590582: 1, 12.323600853539316: 1,
12.319867259186156: 1, 12.301232031204735: 1, 12.301058343956713: 1, 12.284066247300164: 1,
12.27525548039195: 1, 12.2084718134638: 1, 12.194035161892904: 1, 12.174317635278852: 1,
12.164884212766896: 1, 12.156683463389507: 1, 12.137147273596176: 1, 12.082374148777125: 1,
12.079727260035932: 1, 12.046235815177981: 1, 12.042098324305268: 1, 12.032449074211721: 1,
12.020549182425274: 1, 12.006838939507396: 1, 11.991094083737096: 1, 11.98131643268054: 1,
11.95296593695067: 1, 11.951011359135116: 1, 11.950362023949088: 1, 11.928969827454802: 1,
11.92286180911054: 1, 11.92138661018125: 1, 11.918234670379679: 1, 11.907180060158163: 1,
11.896376964299288: 1, 11.894501851447195: 1, 11.862081691554385: 1, 11.851047366865062: 1,
11.844045660040644: 1, 11.836534468981037: 1, 11.826364497101482: 1, 11.807566627840945: 1,
11.804883355251855: 1, 11.70405979421024: 1, 11.699881570650852: 1, 11.699433262223438: 1,
11.693165961910667: 1, 11.69012913056158: 1, 11.673249546174652: 1, 11.641885773160384: 1,
11.591596212317574: 1, 11.525516294827355: 1, 11.508837197016257: 1, 11.497500981303352: 1,
11.496278933904266: 1, 11.496075581620866: 1, 11.455039883348668: 1, 11.453109132966615: 1,
11.438816345069721: 1, 11.433781201622836: 1, 11.39430954788206: 1, 11.393623926375984: 1,
11.3920951033488: 1, 11.391398488991497: 1, 11.391161488873612: 1, 11.390747744851877: 1,
11.301685969975336: 1, 11.291155379258216: 1, 11.281624093960575: 1, 11.278217119774487: 1,
11.267864019705039: 1, 11.263587259468098: 1, 11.259369043369155: 1, 11.248003984252504: 1,
11.242904125880548: 1, 11.241784095923864: 1, 11.223716127568693: 1, 11.21590674544294: 1,
11.197526092799498: 1, 11.184232448292377: 1, 11.170002329008728: 1, 11.157856308408654: 1,
11.143525047127289: 1, 11.130213274303662: 1, 11.112883075741708: 1, 11.110702519158611: 1,
11.084226384851634: 1, 11.066798573880623: 1, 11.061373349464459: 1, 11.011677646040052: 1,
11.009164131268792: 1, 10.991953901387834: 1, 10.989442535213943: 1, 10.988946146584182: 1,
10.98530364060654: 1, 10.984279380133687: 1, 10.982979916829333: 1, 10.974095815642093: 1,
10.970242244264021: 1, 10.962743229177441: 1, 10.948522706992106: 1, 10.939724308041319: 1,
10.935926629024163: 1, 10.890493477900431: 1, 10.888991866520701: 1, 10.887894773247739: 1,
10.883414750751639: 1, 10.874670440719175: 1, 10.870146704997488: 1, 10.857774428103152: 1,
10.852876554383755: 1, 10.848560950434376: 1, 10.810745783078527: 1, 10.789894120198714: 1,
10.775875879442204: 1, 10.771435407586484: 1, 10.762960002380495: 1, 10.738544971358612: 1,
10.731381634589935: 1, 10.724783301742237: 1, 10.723157488718334: 1, 10.718419935170107: 1,
10.692724280630788: 1, 10.67418967108905: 1, 10.661292371638261: 1, 10.659156778891461: 1,
10.655819779263835: 1, 10.651903655427542: 1, 10.649699116996247: 1, 10.642395769864063: 1,
10.63835349367963: 1, 10.629472226102921: 1, 10.623600953469062: 1, 10.614523155822011: 1,
10.60959444727696: 1, 10.576360064256399: 1, 10.55791690483244: 1, 10.54750360159239: 1,
10.543773285787688: 1, 10.540323833982137: 1, 10.535370393334409: 1, 10.526794094567588: 1,
10.508175065915367: 1, 10.498727104014225: 1, 10.490473529183035: 1, 10.486102234347841: 1,
10.471305148003788: 1, 10.467813848207292: 1, 10.454252219617954: 1, 10.452648817547274: 1,
10.449023953941404: 1, 10.44179045485367: 1, 10.413552617165292: 1, 10.401183720983358: 1,
10.338780215541371: 1, 10.323124351423289: 1, 10.322457982947329: 1, 10.313649362559763: 1,
10.239919771391879: 1, 10.234618687273029: 1, 10.234436310116347: 1, 10.221481376748844: 1,
10.196628289470592: 1, 10.178724909934795: 1, 10.150794712380181: 1, 10.143680494780755: 1,
10.119991298504967: 1, 10.116003595980978: 1, 10.112249145373163: 1, 10.103999718553446: 1,
10.101617159440481: 1, 10.098003181683533: 1, 10.09757445730222: 1, 10.089670164210416: 1,
10.058171289659308: 1, 10.05077862600889: 1, 10.040254815045028: 1, 10.039209634202553: 1,
10.033229677483583: 1, 10.01823160892173: 1, 10.016334260900377: 1, 10.015634985683219: 1,
10.012003965141734: 1, 9.966939320025517: 1, 9.945191165221575: 1, 9.942590710531533: 1, 9.91872437
8890811: 1, 9.913748227041015: 1, 9.907385050819322: 1, 9.8989053132552: 1, 9.890922125926037: 1,
9.88994594898675: 1, 9.879949900216515: 1, 9.87629564950896: 1, 9.8406341511784: 1,
9.826530807538159: 1, 9.818979459209595: 1, 9.812769202695783: 1, 9.775318531346192: 1,
9.750997625867324: 1, 9.74670309629028: 1, 9.729553564792178: 1, 9.715517902071245: 1,
9.708775981252195: 1, 9.70366897327671: 1, 9.699118861509485: 1, 9.690521318836346: 1,
9.684671222717471: 1, 9.679609208381812: 1, 9.67610441580127: 1, 9.66562816169733: 1,
9.642474132590456: 1, 9.634936473221616: 1, 9.626805029119579: 1, 9.620471658103343: 1,
9.616339991575384: 1, 9.615093851273548: 1, 9.608399420940232: 1, 9.593702442896008: 1,
9.591930674102999: 1, 9.589406968497947: 1, 9.588241456380556: 1, 9.577780954351375: 1,
9.576807441694235: 1, 9.572446393138662: 1, 9.559914865675657: 1, 9.552343751858562: 1,
9.535130472839569: 1, 9.517587497259326: 1, 9.487355863356669: 1, 9.479915263181644: 1,
9.479794121251793: 1, 9.453554902070957: 1, 9.445286248013309: 1, 9.44333136766591: 1,
9.441268286178449: 1, 9.426185008454654: 1, 9.425637213129695: 1, 9.425556837483322: 1,
9.424933269798027: 1, 9.409865499837556: 1, 9.403519145100029: 1, 9.40114369180485: 1,
9.382406446414508: 1, 9.377746300255845: 1, 9.377505824167237: 1, 9.376717931321966: 1,
9.364879376425657: 1, 9.364623321833035: 1, 9.352264598646386: 1, 9.33267291918576: 1,
9.32819430623982: 1, 9.316801202983605: 1, 9.310503332032475: 1, 9.292667882982444: 1,
9.29197263611198: 1, 9.281333337067831: 1, 9.279972646363321: 1, 9.271466099798158: 1,
9.248860661191106: 1, 9.243667318135383: 1, 9.222293454082077: 1, 9.201994378781094: 1,
9.185658288149687: 1, 9.184602150704684: 1, 9.160172715544665: 1, 9.143107604021136: 1,
9.104050133255575: 1, 9.10340112123645: 1, 9.101868653671685: 1, 9.09701721680546: 1,
9.076246320202493: 1, 9.064764344712849: 1, 9.061642973958836: 1, 9.053206564290807: 1,
9.044505367814455: 1, 9.042055415500814: 1, 9.03773111955766: 1, 9.03329575655308: 1,
9.030823195094394: 1, 9.022687809974439: 1, 9.007626220854416: 1, 9.002725100991112: 1,
8.944051649270344: 1, 8.936071833985253: 1, 8.922234252688838: 1, 8.916473188177703: 1,
8.912451368823582: 1, 8.900122770795065: 1, 8.899663406544255: 1, 8.898976667056322: 1,
8.894365278363308: 1, 8.88852604516879: 1, 8.886329472776433: 1, 8.872471658988852: 1,
8.854228090326494: 1, 8.833481669477736: 1, 8.829302526652192: 1, 8.823018196743753: 1,
8.82086153412141: 1, 8.817199513177792: 1, 8.81458039601528: 1, 8.814197879469633: 1,
8.811438720332614: 1, 8.808542607289626: 1, 8.794301362484678: 1, 8.792733537077623: 1,
8.790947112141405: 1, 8.772234798683186: 1, 8.763687955677117: 1, 8.742707528257924: 1,
8.72570908992237: 1, 8.722626156087243: 1, 8.721210163209456: 1, 8.711673226932263: 1,
8.702746680560756: 1, 8.689932301132249: 1, 8.682616368297547: 1, 8.681222191802439: 1,
8.667825718465643: 1, 8.66008138685594: 1, 8.642914926681824: 1, 8.635632124534451: 1,
8.635516882184584: 1, 8.605998531395088: 1, 8.583807048936455: 1, 8.581990056143722: 1,
8.57403974958824: 1, 8.57171726917381: 1, 8.563310689139294: 1, 8.560470522245621: 1,
8.550515845745004: 1, 8.538607834375789: 1, 8.52167243930193: 1, 8.479824773609836: 1,
8.46398149614268: 1, 8.463513071989443: 1, 8.455252241154126: 1, 8.453208272900543: 1,
8.430475166343752: 1, 8.415382175897772: 1, 8.40593390918901: 1, 8.401466938806202: 1,
8.388076967718032: 1, 8.381011302884154: 1, 8.34337332431706: 1, 8.335126123207237: 1,
8.328690221881581: 1, 8.324982115775976: 1, 8.308418048413234: 1, 8.274303558586274: 1,
8.259255874586128: 1, 8.257601945076711: 1, 8.257219537371272: 1, 8.254674062034976: 1,
8.253493058872909: 1, 8.240591368271682: 1, 8.203835372503205: 1, 8.19404409139461: 1,
8.181452931933071: 1, 8.18109593483456: 1, 8.162089433704015: 1, 8.157175054577076: 1,
8.152045803909997: 1, 8.14827567321849: 1, 8.14586929248552: 1, 8.140315816663804: 1,
8.13339928808078: 1, 8.130390974389002: 1, 8.112804196545497: 1, 8.110208952470328: 1,
8.053234629995254: 1, 8.044225898274696: 1, 8.028873586384195: 1, 7.971386070238061: 1,
7.971307789955359: 1, 7.96777239913361: 1, 7.9635362816303745: 1, 7.958423198387063: 1,
7.95685957793533: 1, 7.952586202879537: 1, 7.951448563541705: 1, 7.950014572650517: 1,
7.935168374948315: 1, 7.911145190795048: 1, 7.907992332694968: 1, 7.900381147849715: 1,
7.89666822602443: 1, 7.894477606723113: 1, 7.871487486240053: 1, 7.821995591214573: 1,
7.811517177339199: 1, 7.8093706288269695: 1, 7.792969964025926: 1, 7.789977459831916: 1,
7.771099599719812: 1, 7.76710110380573: 1, 7.755809733412548: 1, 7.755054088867965: 1,
7.752428550293298: 1, 7.7352741772040625: 1, 7.717224409683708: 1, 7.692596752208869: 1,
7.688658212601434: 1, 7.666797680297411: 1, 7.6652825139557095: 1, 7.661673944776881: 1,
7.625892995783899: 1, 7.61996491396425: 1, 7.611818575325672: 1, 7.60497513974875: 1,
7.603186481164297: 1, 7.60195398783184: 1, 7.595283255693254: 1, 7.5877687960031: 1,
7.580795774315748: 1, 7.5663766165204045: 1, 7.565267045375169: 1, 7.559781304311897: 1,
7.511306316490987: 1, 7.503955753237551: 1, 7.490892499459127: 1, 7.484275483314977: 1,
7.483015733434967: 1, 7.478002384009957: 1, 7.473924704272374: 1, 7.43904952066073: 1,
7.388760017158587: 1, 7.38655401206979: 1, 7.373495937776988: 1, 7.327596119895092: 1,
7.3075714815995525: 1, 7.292678259853253: 1, 7.254833958364922: 1, 7.218872552425893: 1, 7.19368866
4435462: 1, 7.181881317173743: 1, 7.17950915080772: 1, 7.172238350943627: 1, 7.162607422029784: 1,
7.1502968587324: 1, 7.1404185499756085: 1, 7.08354367236268: 1, 7.074274440120067: 1,
7.064738514235847: 1, 7.0438901864930585: 1, 7.043818358071893: 1, 6.994274895378726: 1,
6.972246483465532: 1, 6.950761562156878: 1, 6.909671874886062: 1, 6.778242808840888: 1,
6.778204653823729: 1, 6.759318208839193: 1, 6.73456824811035: 1, 6.7308022577029485: 1,
6.70079903854279: 1, 6.633956761096837: 1, 6.604799176598486: 1, 6.57470195793869: 1,
6.547466939714414: 1, 6.537520600864914: 1, 6.455604879886807: 1, 6.430355277970063: 1,
6.220593942610076: 1})
```

In [47]:

```python
# 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.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=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.clas
ses_, eps=1e-15))

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


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

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

```
For values of alpha =  1e-05 The log loss is: 1.1332799753690965
For values of alpha =  0.0001 The log loss is: 1.1401701165617
For values of alpha =  0.001 The log loss is: 1.4320352626633477
For values of alpha =  0.01 The log loss is: 1.9484247797941776
For values of alpha =  0.1 The log loss is: 2.048307444386819
For values of alpha =  1 The log loss is: 2.03227229267448
```



```
For values of best alpha =  1e-05 The train log loss is: 0.7734900924635322
For values of best alpha =  1e-05 The cross validation log loss is: 1.1332799753690965
For values of best alpha =  1e-05 The test log loss is: 1.0675482165158825
```

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

Ans. Yes, it seems like!

In [48]:

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

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

In [49]:

```
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")
```

```
3.578 % of word of test data appeared in train data
3.715 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [50]:

```
#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 [51]:

```
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 [52]:

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

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

Stacking the three types of features

In [53]:

```
# merging gene, variance and text features

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

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

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

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

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


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

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

In [54]:

```
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, 3190)
(number of data points * number of features) in test data =  (665, 3190)
(number of data points * number of features) in cross validation data = (532, 3190)
```

In [55]:

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

```
 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

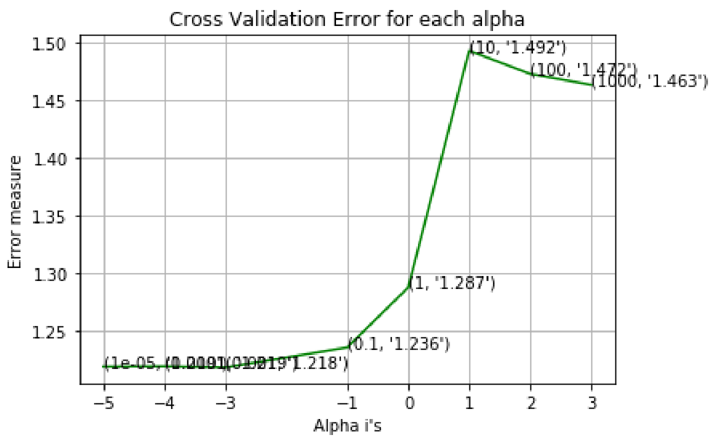
4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [56]:

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

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


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


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

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


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


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

```
for alpha = 1e-05
Log Loss : 1.2189753210138836
for alpha = 0.0001
Log Loss : 1.219234888454888
for alpha = 0.001
Log Loss : 1.2182657534624224
for alpha = 0.1
Log Loss : 1.2355641036322333
for alpha = 1
Log Loss : 1.2874930058911307
for alpha = 10
Log Loss : 1.4924748328746082
for alpha = 100
Log Loss : 1.4724086489080672
for alpha = 1000
Log Loss : 1.4629175983792337
```



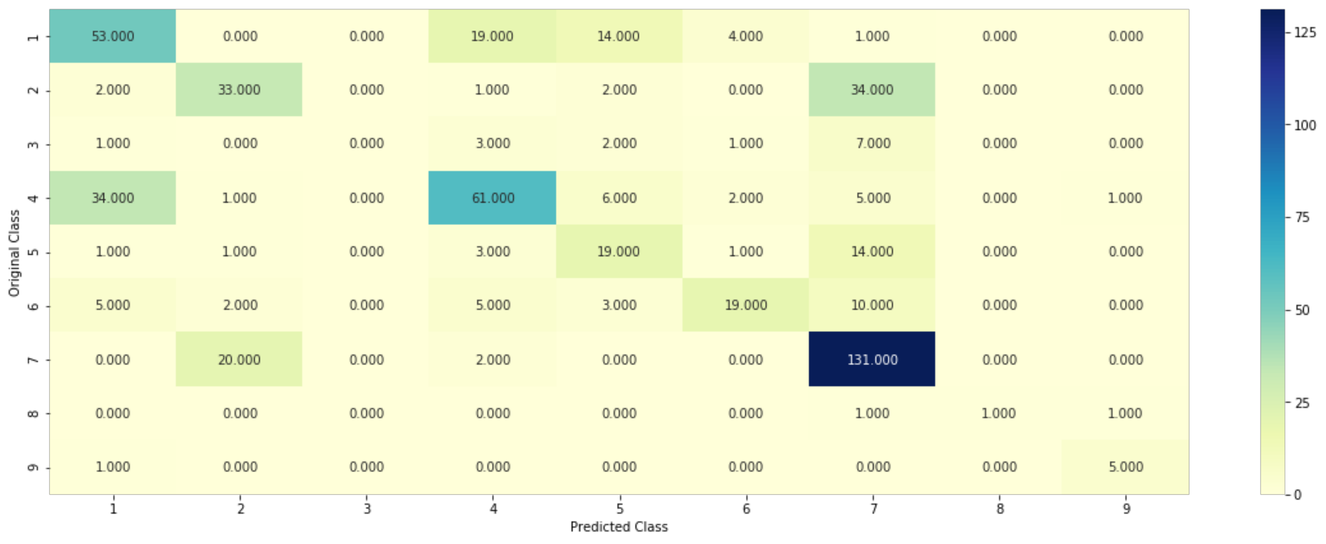
```
For values of best alpha =  0.001 The train log loss is: 0.5282886647203119
For values of best alpha =  0.001 The cross validation log loss is: 1.2182657534624224
For values of best alpha =  0.001 The test log loss is: 1.1980873453282335
```

4.1.1.2. Testing the model with best hyper paramters

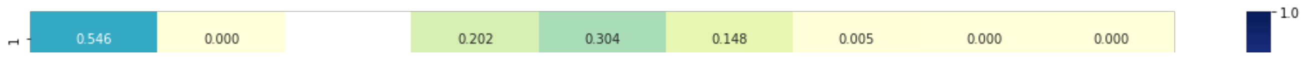
In [57]:

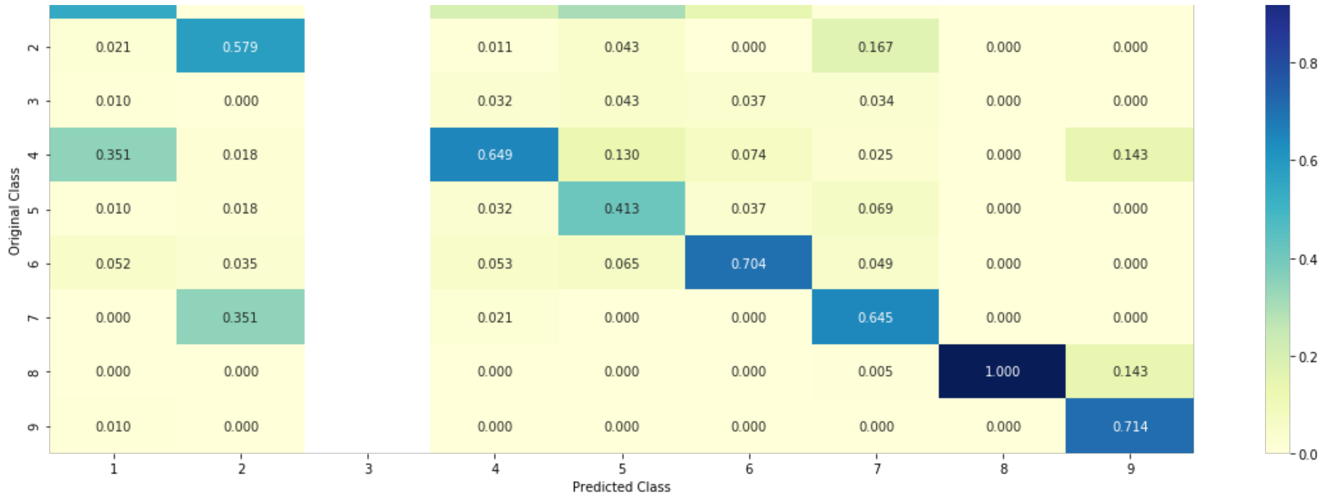
```
# find more about Multinomial Naive base function here http://scikit-
learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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

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

```
Log Loss : 1.2182657534624224
Number of missclassified point : 0.39473684210526316
-------------------- Confusion matrix --------------------
```

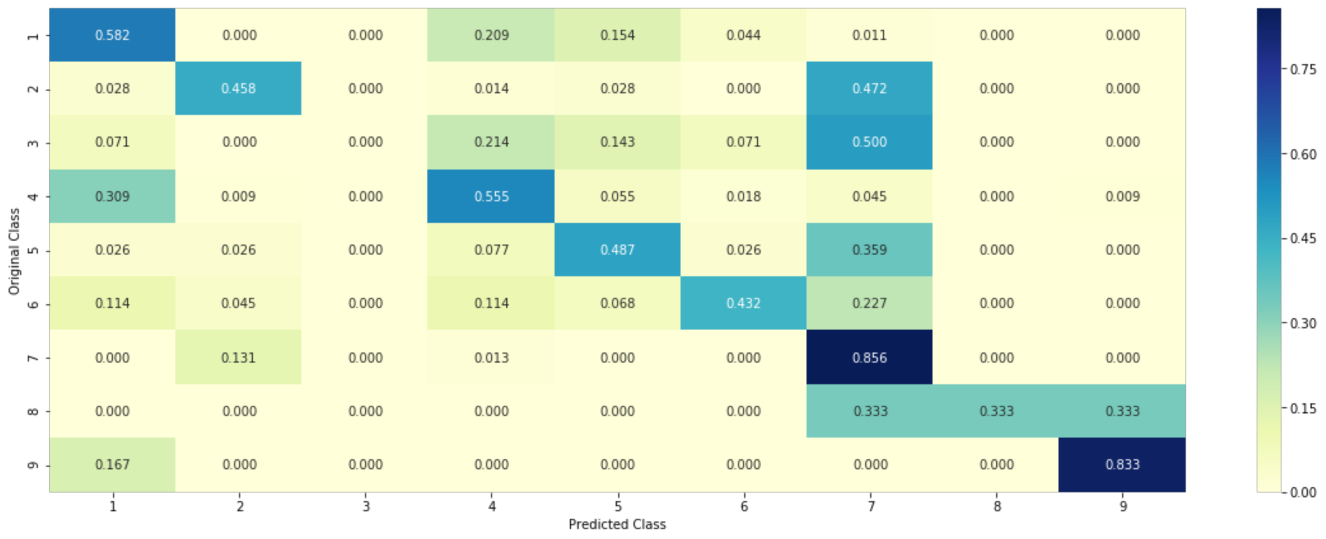


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```





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



4.1.1.3. Feature Importance, Correctly classified point

In [58]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0553 0.041  0.01   0.0659 0.032  0.03   0.76   0.0028 0.003 ]]
Actual Class : 7
--------------------------------------------------
16 Text feature [activation] present in test data point [True]
17 Text feature [activated] present in test data point [True]
18 Text feature [cells] present in test data point [True]
21 Text feature [kinase] present in test data point [True]
22 Text feature [signaling] present in test data point [True]
23 Text feature [expressing] present in test data point [True]
24 Text feature [downstream] present in test data point [True]
25 Text feature [contrast] present in test data point [True]
26 Text feature [presence] present in test data point [True]
27 Text feature [independent] present in test data point [True]
```

```
28 Text feature [inhibitor] present in test data point [True]
29 Text feature [growth] present in test data point [True]
30 Text feature [also] present in test data point [True]
31 Text feature [10] present in test data point [True]
32 Text feature [constitutive] present in test data point [True]
35 Text feature [factor] present in test data point [True]
36 Text feature [shown] present in test data point [True]
37 Text feature [however] present in test data point [True]
38 Text feature [treated] present in test data point [True]
41 Text feature [well] present in test data point [True]
42 Text feature [addition] present in test data point [True]
43 Text feature [previously] present in test data point [True]
44 Text feature [higher] present in test data point [True]
45 Text feature [similar] present in test data point [True]
46 Text feature [sensitive] present in test data point [True]
47 Text feature [mutations] present in test data point [True]
48 Text feature [phosphorylation] present in test data point [True]
49 Text feature [cell] present in test data point [True]
50 Text feature [suggest] present in test data point [True]
51 Text feature [increased] present in test data point [True]
52 Text feature [found] present in test data point [True]
53 Text feature [treatment] present in test data point [True]
54 Text feature [3b] present in test data point [True]
55 Text feature [recently] present in test data point [True]
56 Text feature [may] present in test data point [True]
57 Text feature [enhanced] present in test data point [True]
58 Text feature [mutation] present in test data point [True]
61 Text feature [inhibitors] present in test data point [True]
62 Text feature [oncogenic] present in test data point [True]
63 Text feature [results] present in test data point [True]
64 Text feature [showed] present in test data point [True]
65 Text feature [observed] present in test data point [True]
66 Text feature [absence] present in test data point [True]
67 Text feature [inhibited] present in test data point [True]
68 Text feature [potential] present in test data point [True]
69 Text feature [consistent] present in test data point [True]
70 Text feature [described] present in test data point [True]
71 Text feature [fig] present in test data point [True]
72 Text feature [although] present in test data point [True]
73 Text feature [concentrations] present in test data point [True]
74 Text feature [constitutively] present in test data point [True]
75 Text feature [proliferation] present in test data point [True]
76 Text feature [inhibition] present in test data point [True]
77 Text feature [pathway] present in test data point [True]
78 Text feature [using] present in test data point [True]
79 Text feature [examined] present in test data point [True]
80 Text feature [without] present in test data point [True]
81 Text feature [figure] present in test data point [True]
82 Text feature [tyrosine] present in test data point [True]
83 Text feature [study] present in test data point [True]
85 Text feature [followed] present in test data point [True]
86 Text feature [either] present in test data point [True]
87 Text feature [total] present in test data point [True]
89 Text feature [receptor] present in test data point [True]
90 Text feature [increase] present in test data point [True]
91 Text feature [reported] present in test data point [True]
92 Text feature [pathways] present in test data point [True]
93 Text feature [two] present in test data point [True]
94 Text feature [three] present in test data point [True]
95 Text feature [studies] present in test data point [True]
96 Text feature [mutant] present in test data point [True]
97 Text feature [activate] present in test data point [True]
98 Text feature [previous] present in test data point [True]
99 Text feature [13] present in test data point [True]
Out of the top  100  features  74 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [59]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0553 0.041  0.01   0.0659 0.032  0.03   0.76   0.0028 0.003 ]]
Actual Class : 7
--------------------------------------------------
16 Text feature [activation] present in test data point [True]
17 Text feature [activated] present in test data point [True]
18 Text feature [cells] present in test data point [True]
21 Text feature [kinase] present in test data point [True]
22 Text feature [signaling] present in test data point [True]
23 Text feature [expressing] present in test data point [True]
24 Text feature [downstream] present in test data point [True]
25 Text feature [contrast] present in test data point [True]
26 Text feature [presence] present in test data point [True]
27 Text feature [independent] present in test data point [True]
28 Text feature [inhibitor] present in test data point [True]
29 Text feature [growth] present in test data point [True]
30 Text feature [also] present in test data point [True]
31 Text feature [10] present in test data point [True]
32 Text feature [constitutive] present in test data point [True]
35 Text feature [factor] present in test data point [True]
36 Text feature [shown] present in test data point [True]
37 Text feature [however] present in test data point [True]
38 Text feature [treated] present in test data point [True]
41 Text feature [well] present in test data point [True]
42 Text feature [addition] present in test data point [True]
43 Text feature [previously] present in test data point [True]
44 Text feature [higher] present in test data point [True]
45 Text feature [similar] present in test data point [True]
46 Text feature [sensitive] present in test data point [True]
47 Text feature [mutations] present in test data point [True]
48 Text feature [phosphorylation] present in test data point [True]
49 Text feature [cell] present in test data point [True]
50 Text feature [suggest] present in test data point [True]
51 Text feature [increased] present in test data point [True]
52 Text feature [found] present in test data point [True]
53 Text feature [treatment] present in test data point [True]
54 Text feature [3b] present in test data point [True]
55 Text feature [recently] present in test data point [True]
56 Text feature [may] present in test data point [True]
57 Text feature [enhanced] present in test data point [True]
58 Text feature [mutation] present in test data point [True]
61 Text feature [inhibitors] present in test data point [True]
62 Text feature [oncogenic] present in test data point [True]
63 Text feature [results] present in test data point [True]
64 Text feature [showed] present in test data point [True]
65 Text feature [observed] present in test data point [True]
66 Text feature [absence] present in test data point [True]
67 Text feature [inhibited] present in test data point [True]
68 Text feature [potential] present in test data point [True]
69 Text feature [consistent] present in test data point [True]
70 Text feature [described] present in test data point [True]
71 Text feature [fig] present in test data point [True]
72 Text feature [although] present in test data point [True]
73 Text feature [concentrations] present in test data point [True]
74 Text feature [constitutively] present in test data point [True]
75 Text feature [proliferation] present in test data point [True]
76 Text feature [inhibition] present in test data point [True]
77 Text feature [pathway] present in test data point [True]
78 Text feature [using] present in test data point [True]
79 Text feature [examined] present in test data point [True]
80 Text feature [without] present in test data point [True]
81 Text feature [figure] present in test data point [True]
82 Text feature [tyrosine] present in test data point [True]
83 Text feature [study] present in test data point [True]
85 Text feature [followed] present in test data point [True]
86 Text feature [either] present in test data point [True]
87 Text feature [total] present in test data point [True]
89 Text feature [receptor] present in test data point [True]
```

```
90 Text feature [increase] present in test data point [True]
91 Text feature [reported] present in test data point [True]
92 Text feature [pathways] present in test data point [True]
93 Text feature [two] present in test data point [True]
94 Text feature [three] present in test data point [True]
95 Text feature [studies] present in test data point [True]
96 Text feature [mutant] present in test data point [True]
97 Text feature [activate] present in test data point [True]
98 Text feature [previous] present in test data point [True]
99 Text feature [13] present in test data point [True]
Out of the top  100  features  74 are present in query point
```

4.2. K Nearest Neighbour Classification

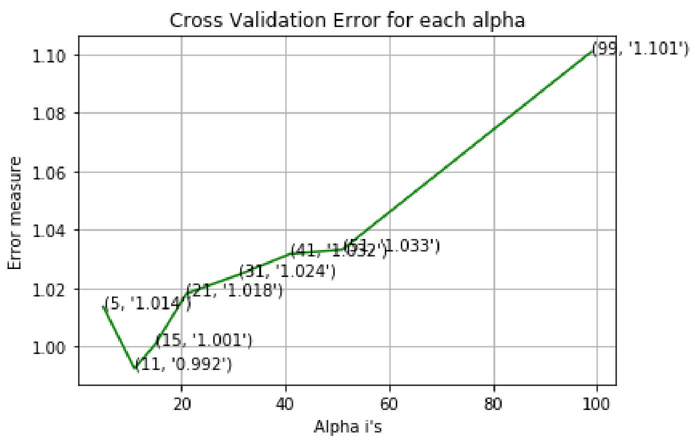
4.2.1. Hyper parameter tuning

In [60]:

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

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


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


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

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

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

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

```
for alpha = 5
Log Loss : 1.0135078123961825
for alpha = 11
Log Loss : 0.9924201892354696
for alpha = 15
Log Loss : 1.0006856221531295
for alpha = 21
Log Loss : 1.0180775103394368
for alpha = 31
Log Loss : 1.0244685923365353
for alpha = 41
Log Loss : 1.0316702410055811
for alpha = 51
Log Loss : 1.0331243694967758
for alpha = 99
Log Loss : 1.1009534167108281
```



```
For values of best alpha =  11 The train log loss is: 0.6314786511730951
For values of best alpha =  11 The cross validation log loss is: 0.9924201892354696
For values of best alpha =  11 The test log loss is: 1.0059073715392284
```

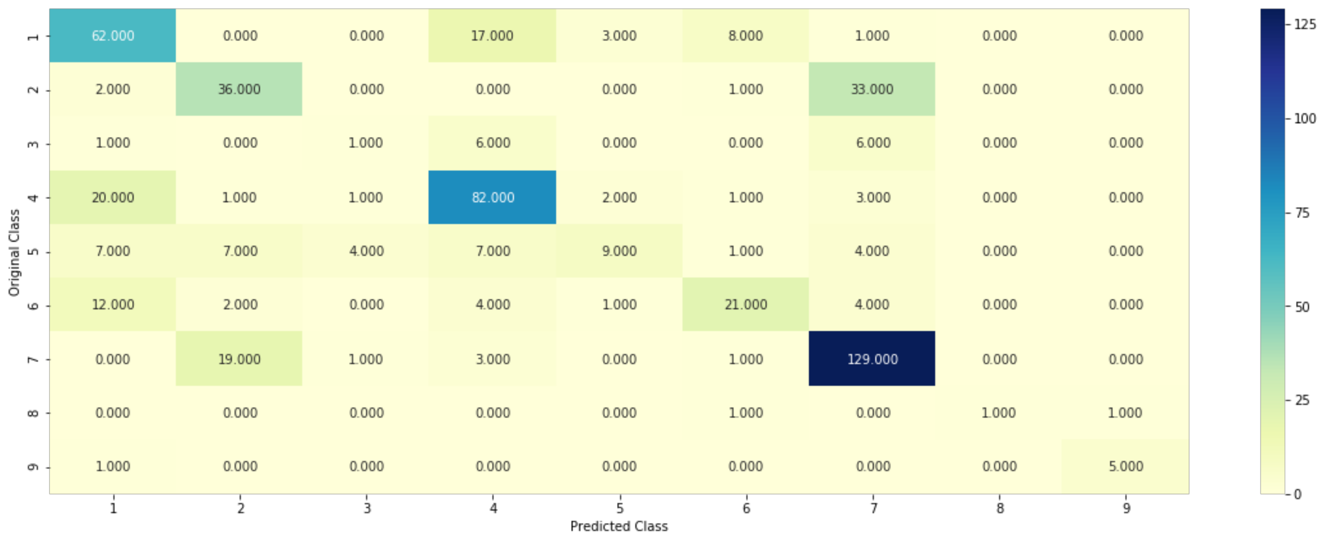
4.2.2. Testing the model with best hyper paramters

In [61]:

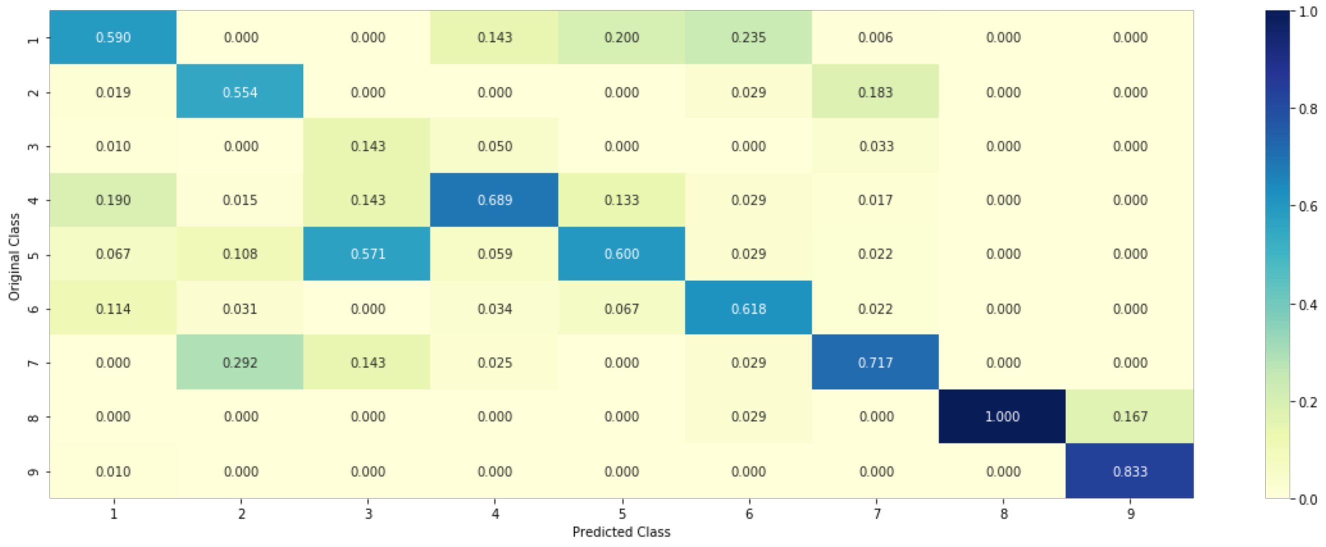
```
# find more about KNeighborsClassifier() here http://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski',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

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-------------------------------------
```

```
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-ne
ighbors-geometric-intuition-with-a-toy-example-1/
#-------------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y, clf)
```

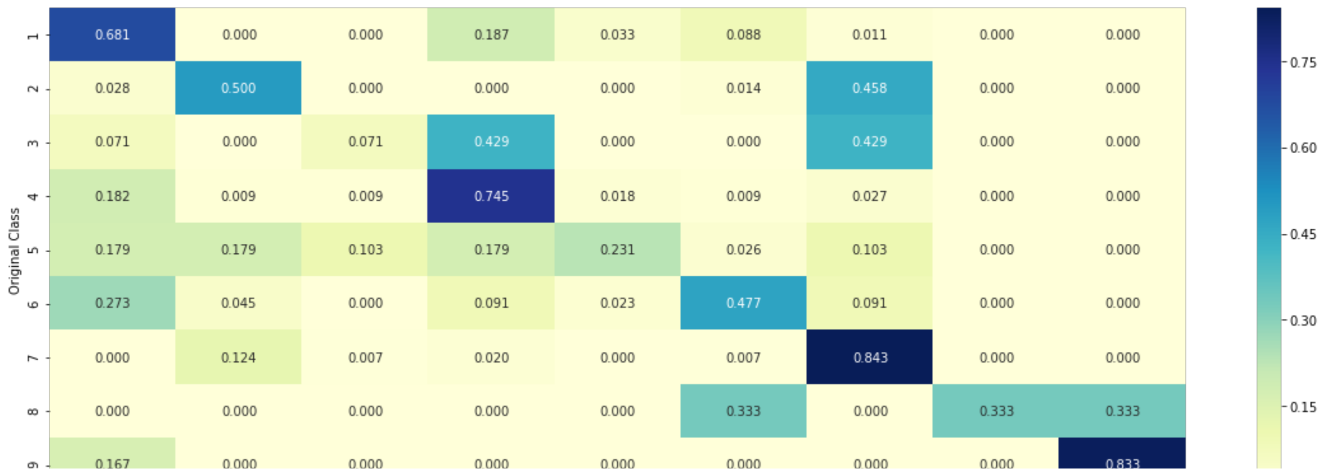
```
Log loss : 0.9924201892354696
Number of mis-classified points : 0.34962406015037595
-------------------- Confusion matrix --------------------
```

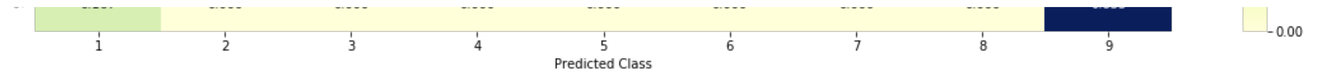


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





4.2.3.Sample Query point -1

In [62]:

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

4.2.4. Sample Query Point-2

In [63]:

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

test_point_index = 100

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

```
Predicted Class : 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
 7 7 7 7 2 7 7 7]
Fequency of nearest points : Counter({7: 10, 2: 1})
```

4.3. Logistic Regression

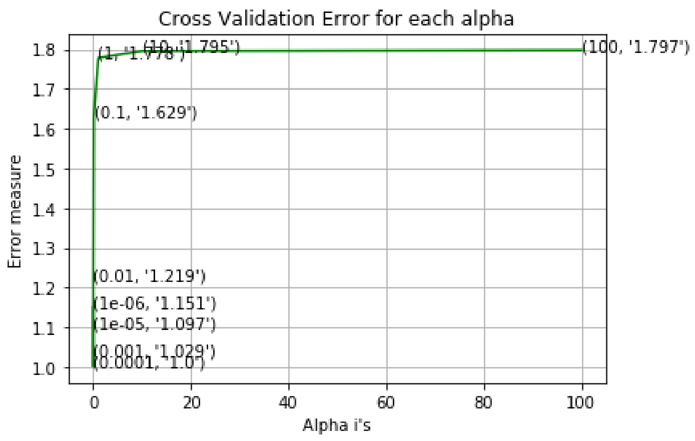
4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [64]:

```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=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-in
tuition-1/
#------------------------------


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

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', ran
dom_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```
for alpha = 1e-06
Log Loss : 1.1507472519411477
for alpha = 1e-05
Log Loss : 1.09679268243564
Log Loss : 1.09679208245504
for alpha = 0.0001
Log Loss : 1.000441942795181
for alpha = 0.001
Log Loss : 1.0289303735224615
for alpha = 0.01
Log Loss : 1.2187480579084689
for alpha = 0.1
Log Loss : 1.6292964049248162
for alpha = 1
Log Loss : 1.7780908869103667
for alpha = 10
Log Loss : 1.795341175139565
for alpha = 100
Log Loss : 1.7973772038939875
```



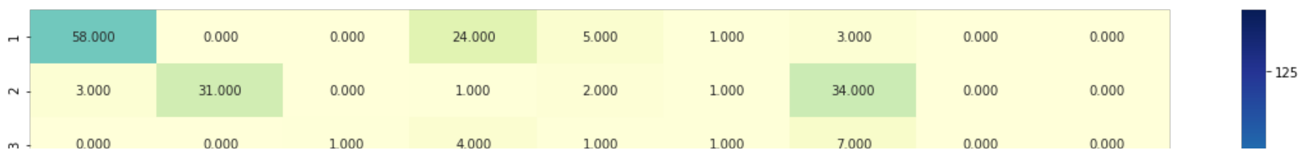
```
For values of best alpha =  0.0001 The train log loss is: 0.4445853332786352
For values of best alpha =  0.0001 The cross validation log loss is: 1.000441942795181
For values of best alpha =  0.0001 The test log loss is: 0.9713398242027661
```

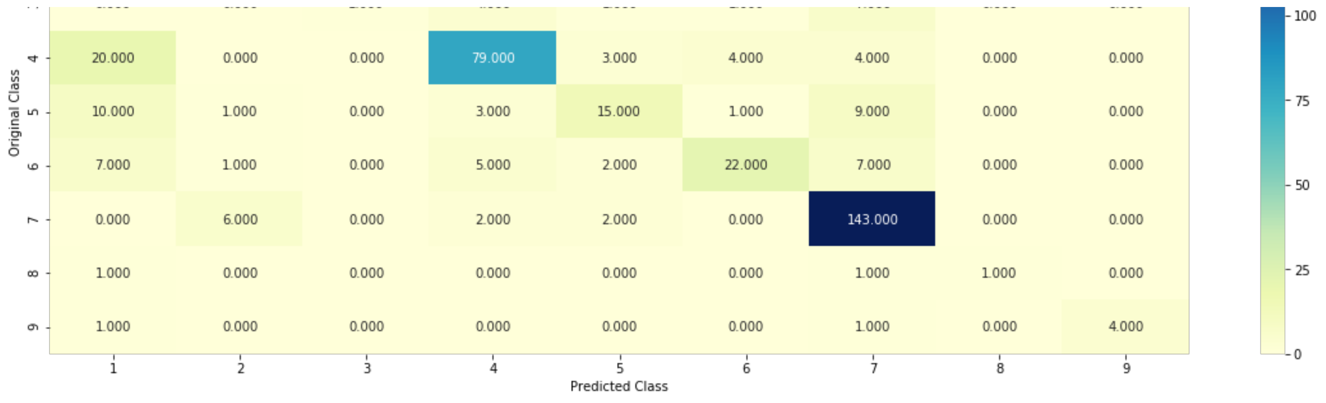
4.3.1.2. Testing the model with best hyper paramters

In [65]:

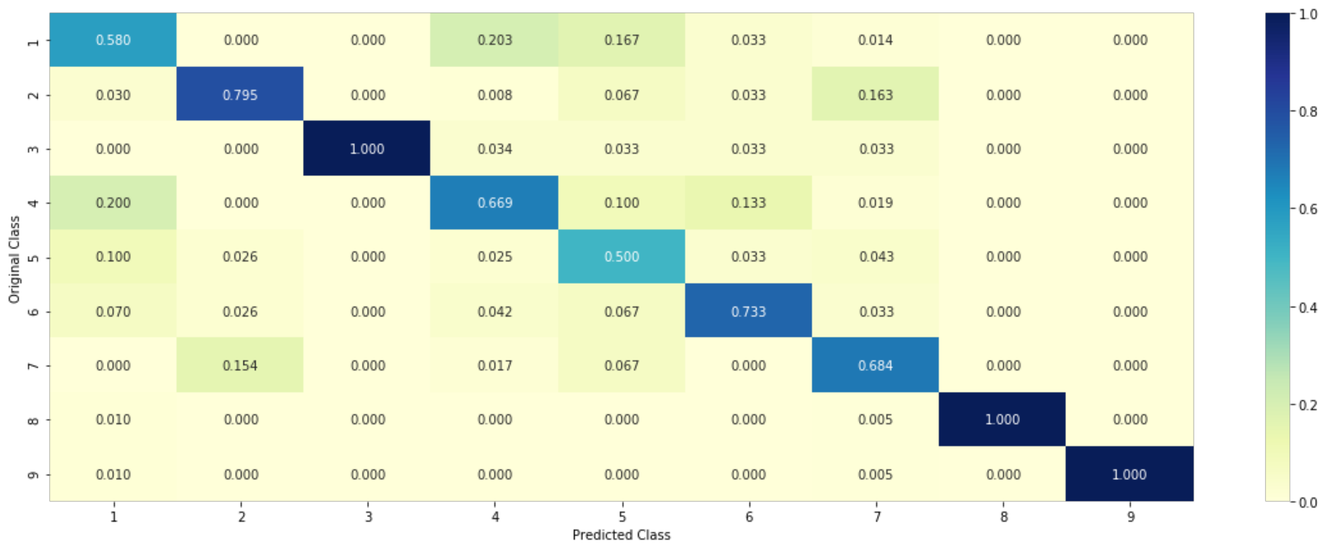
```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# 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-in
tuition-1/
#------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

```
Log loss : 1.000441942795181
Number of mis-classified points : 0.33458646616541354
-------------------- Confusion matrix --------------------
```

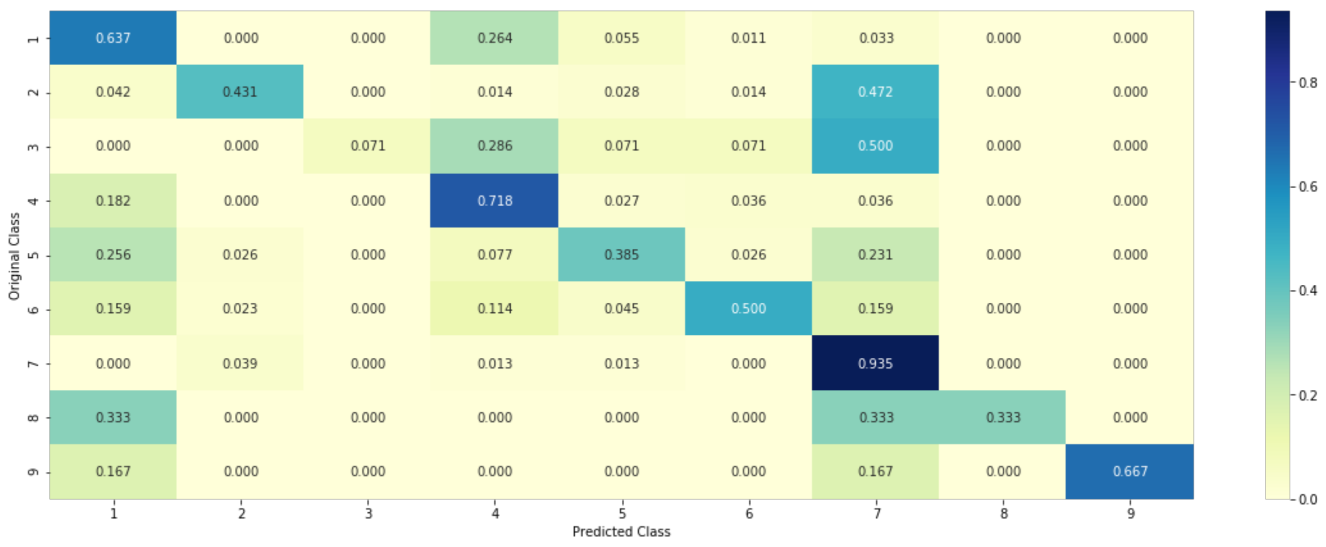




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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



4.3.1.3. Feature Importance

In [66]:

```
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 [67]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1359 0.0448 0.016  0.0436 0.1078 0.0357 0.6013 0.0055 0.0095]]
Actual Class : 7
--------------------------------------------------
8 Text feature [activated] present in test data point [True]
11 Text feature [constitutive] present in test data point [True]
17 Text feature [downstream] present in test data point [True]
18 Text feature [codon] present in test data point [True]
23 Text feature [overexpression] present in test data point [True]
29 Text feature [activation] present in test data point [True]
32 Text feature [enhanced] present in test data point [True]
34 Text feature [positive] present in test data point [True]
36 Text feature [2a] present in test data point [True]
43 Text feature [ligand] present in test data point [True]
54 Text feature [elevated] present in test data point [True]
59 Text feature [fold] present in test data point [True]
65 Text feature [3b] present in test data point [True]
67 Text feature [mapk] present in test data point [True]
75 Text feature [bone] present in test data point [True]
79 Text feature [concentrations] present in test data point [True]
84 Text feature [oncogene] present in test data point [True]
95 Text feature [inhibited] present in test data point [True]
101 Text feature [receptors] present in test data point [True]
102 Text feature [000] present in test data point [True]
103 Text feature [signaling] present in test data point [True]
108 Text feature [leukemia] present in test data point [True]
129 Text feature [approximately] present in test data point [True]
144 Text feature [activate] present in test data point [True]
148 Text feature [activating] present in test data point [True]
149 Text feature [transformed] present in test data point [True]
177 Text feature [mechanisms] present in test data point [True]
198 Text feature [lung] present in test data point [True]
201 Text feature [malignant] present in test data point [True]
203 Text feature [factor] present in test data point [True]
208 Text feature [presence] present in test data point [True]
215 Text feature [pathways] present in test data point [True]
217 Text feature [examined] present in test data point [True]
220 Text feature [advanced] present in test data point [True]
233 Text feature [expressing] present in test data point [True]
```

```
248 Text feature [2b] present in test data point [True]
250 Text feature [s3] present in test data point [True]
266 Text feature [versus] present in test data point [True]
271 Text feature [constitutively] present in test data point [True]
275 Text feature [wt] present in test data point [True]
278 Text feature [2003] present in test data point [True]
280 Text feature [lead] present in test data point [True]
307 Text feature [days] present in test data point [True]
313 Text feature [position] present in test data point [True]
317 Text feature [bp] present in test data point [True]
335 Text feature [occur] present in test data point [True]
370 Text feature [promote] present in test data point [True]
371 Text feature [colony] present in test data point [True]
392 Text feature [akt] present in test data point [True]
398 Text feature [cells] present in test data point [True]
422 Text feature [gfp] present in test data point [True]
423 Text feature [phospho] present in test data point [True]
442 Text feature [inhibitor] present in test data point [True]
445 Text feature [specimens] present in test data point [True]
468 Text feature [day] present in test data point [True]
474 Text feature [culture] present in test data point [True]
476 Text feature [gain] present in test data point [True]
481 Text feature [proliferation] present in test data point [True]
484 Text feature [effective] present in test data point [True]
496 Text feature [coding] present in test data point [True]
Out of the top  500  features  60 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [68]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[1.07e-02 6.45e-02 4.00e-04 5.90e-03 3.50e-03 1.40e-03 9.05e-01 8.
40e-03
  3.00e-04]]
Actual Class : 7
--------------------------------------------------
8 Text feature [activated] present in test data point [True]
11 Text feature [constitutive] present in test data point [True]
17 Text feature [downstream] present in test data point [True]
22 Text feature [insertion] present in test data point [True]
23 Text feature [overexpression] present in test data point [True]
29 Text feature [activation] present in test data point [True]
32 Text feature [enhanced] present in test data point [True]
34 Text feature [positive] present in test data point [True]
36 Text feature [2a] present in test data point [True]
43 Text feature [ligand] present in test data point [True]
54 Text feature [elevated] present in test data point [True]
65 Text feature [3b] present in test data point [True]
67 Text feature [mapk] present in test data point [True]
75 Text feature [bone] present in test data point [True]
79 Text feature [concentrations] present in test data point [True]
87 Text feature [phosphorylated] present in test data point [True]
95 Text feature [inhibited] present in test data point [True]
103 Text feature [signaling] present in test data point [True]
108 Text feature [leukemia] present in test data point [True]
129 Text feature [approximately] present in test data point [True]
144 Text feature [activate] present in test data point [True]
149 Text feature [transformed] present in test data point [True]
171 Text feature [transformation] present in test data point [True]
177 Text feature [mechanisms] present in test data point [True]
```

```
198 Text feature [lung] present in test data point [True]
203 Text feature [factor] present in test data point [True]
208 Text feature [presence] present in test data point [True]
215 Text feature [pathways] present in test data point [True]
217 Text feature [examined] present in test data point [True]
233 Text feature [expressing] present in test data point [True]
248 Text feature [2b] present in test data point [True]
257 Text feature [ras] present in test data point [True]
271 Text feature [constitutively] present in test data point [True]
280 Text feature [lead] present in test data point [True]
307 Text feature [days] present in test data point [True]
317 Text feature [bp] present in test data point [True]
335 Text feature [occur] present in test data point [True]
353 Text feature [transforming] present in test data point [True]
370 Text feature [promote] present in test data point [True]
392 Text feature [akt] present in test data point [True]
397 Text feature [72] present in test data point [True]
398 Text feature [cells] present in test data point [True]
423 Text feature [phospho] present in test data point [True]
442 Text feature [inhibitor] present in test data point [True]
445 Text feature [specimens] present in test data point [True]
467 Text feature [extracellular] present in test data point [True]
468 Text feature [day] present in test data point [True]
481 Text feature [proliferation] present in test data point [True]
484 Text feature [effective] present in test data point [True]
Out of the top  500  features  49 are present in query point
```

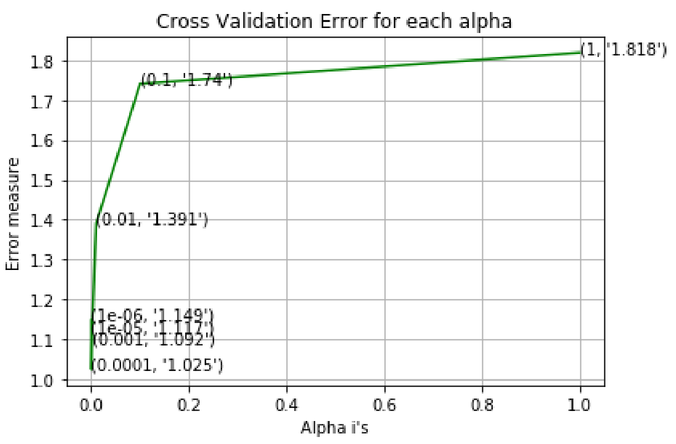
4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

In [69]:

```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=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-in
tuition-1/
#------------------------------



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

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.1491058758961354
for alpha = 1e-05
Log Loss : 1.1168028666910488
for alpha = 0.0001
Log Loss : 1.025147652417543
for alpha = 0.001
Log Loss : 1.0918739194685716
for alpha = 0.01
Log Loss : 1.3908666496584612
for alpha = 0.1
Log Loss : 1.7400918221098747
for alpha = 1
Log Loss : 1.8180840242686098
```



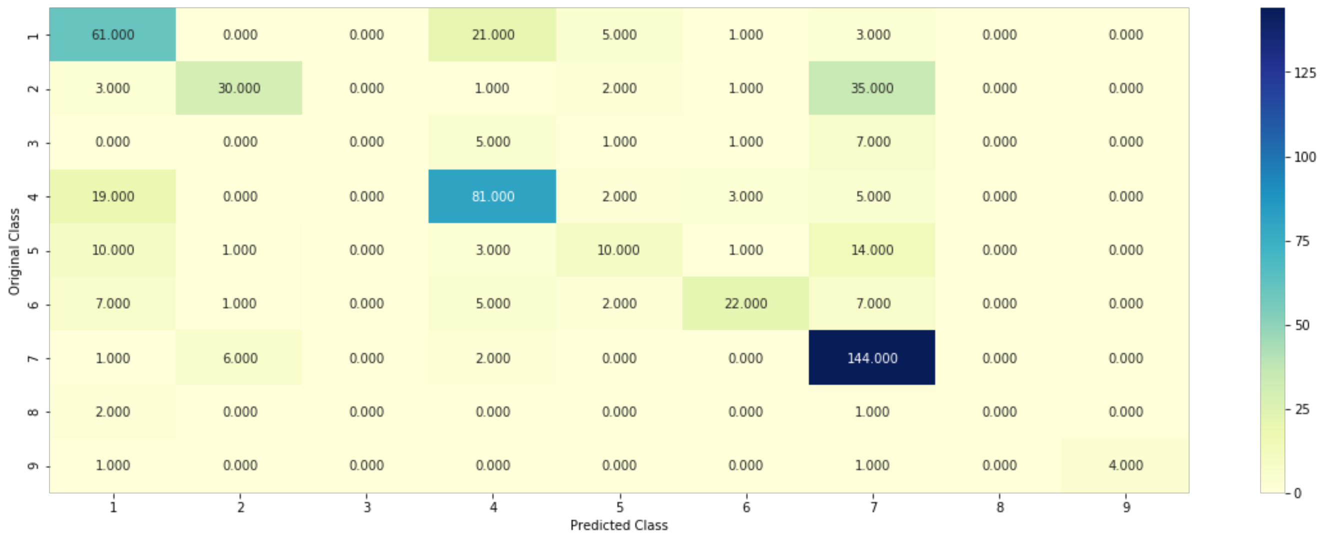
```
For values of best alpha =  0.0001 The train log loss is: 0.43317253163431446
For values of best alpha =  0.0001 The cross validation log loss is: 1.025147652417543
For values of best alpha =  0.0001 The test log loss is: 0.9874409474063465
```

4.3.2.2. Testing model with best hyper parameters

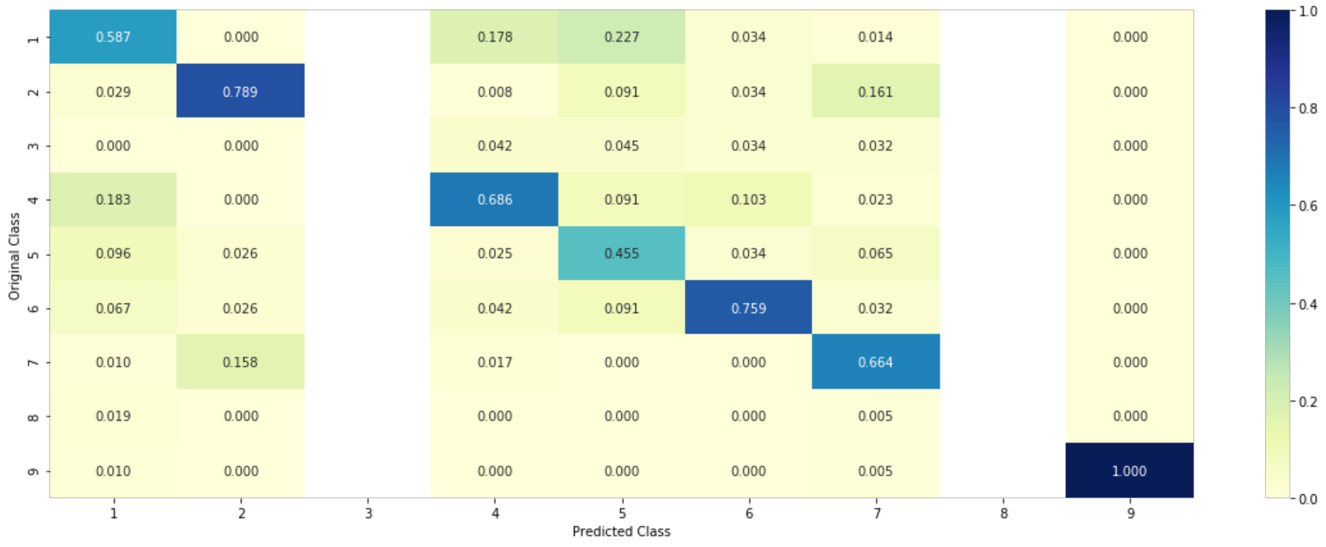
In [70]:

```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=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)
```

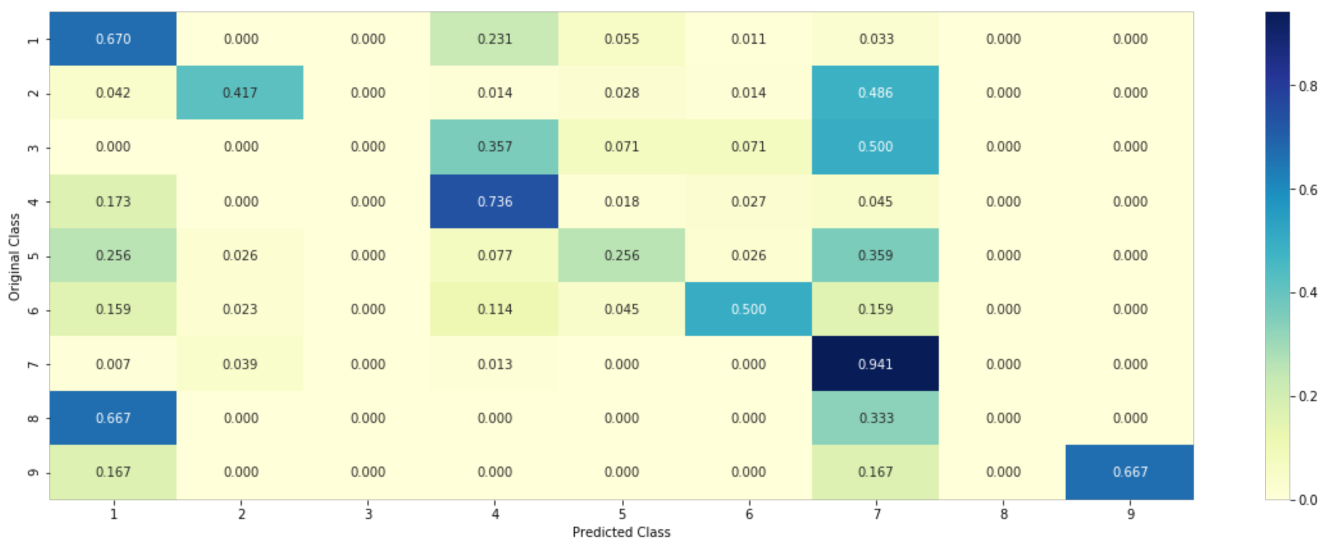
```
Log loss : 1.025147652417543
Number of mis-classified points : 0.3383458646616541
-------------------- Confusion matrix --------------------
```



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



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



4.3.2.3. Feature Importance, Correctly Classified point

In [71]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1351 0.0465 0.0139 0.0438 0.1103 0.0359 0.5983 0.0065 0.0097]]
Actual Class : 7
--------------------------------------------------
13 Text feature [activated] present in test data point [True]
18 Text feature [codon] present in test data point [True]
26 Text feature [constitutive] present in test data point [True]
28 Text feature [downstream] present in test data point [True]
38 Text feature [overexpression] present in test data point [True]
47 Text feature [2a] present in test data point [True]
57 Text feature [enhanced] present in test data point [True]
68 Text feature [positive] present in test data point [True]
72 Text feature [elevated] present in test data point [True]
76 Text feature [activation] present in test data point [True]
79 Text feature [ligand] present in test data point [True]
92 Text feature [concentrations] present in test data point [True]
96 Text feature [fold] present in test data point [True]
112 Text feature [3b] present in test data point [True]
115 Text feature [inhibited] present in test data point [True]
122 Text feature [bone] present in test data point [True]
123 Text feature [000] present in test data point [True]
126 Text feature [mapk] present in test data point [True]
133 Text feature [approximately] present in test data point [True]
149 Text feature [receptors] present in test data point [True]
166 Text feature [lung] present in test data point [True]
188 Text feature [mechanisms] present in test data point [True]
192 Text feature [leukemia] present in test data point [True]
193 Text feature [oncogene] present in test data point [True]
199 Text feature [activate] present in test data point [True]
213 Text feature [activating] present in test data point [True]
215 Text feature [signaling] present in test data point [True]
238 Text feature [transformed] present in test data point [True]
241 Text feature [factor] present in test data point [True]
247 Text feature [s3] present in test data point [True]
```

```
248 Text feature [examined] present in test data point [True]
278 Text feature [advanced] present in test data point [True]
282 Text feature [presence] present in test data point [True]
286 Text feature [2b] present in test data point [True]
297 Text feature [lead] present in test data point [True]
306 Text feature [wt] present in test data point [True]
307 Text feature [versus] present in test data point [True]
308 Text feature [expressing] present in test data point [True]
323 Text feature [malignant] present in test data point [True]
326 Text feature [bp] present in test data point [True]
336 Text feature [2003] present in test data point [True]
359 Text feature [occur] present in test data point [True]
360 Text feature [previously] present in test data point [True]
379 Text feature [proliferation] present in test data point [True]
382 Text feature [position] present in test data point [True]
383 Text feature [observations] present in test data point [True]
397 Text feature [promote] present in test data point [True]
400 Text feature [coding] present in test data point [True]
403 Text feature [colony] present in test data point [True]
405 Text feature [akt] present in test data point [True]
415 Text feature [constitutively] present in test data point [True]
427 Text feature [pathways] present in test data point [True]
430 Text feature [days] present in test data point [True]
436 Text feature [current] present in test data point [True]
443 Text feature [gain] present in test data point [True]
451 Text feature [epithelial] present in test data point [True]
454 Text feature [gfp] present in test data point [True]
456 Text feature [effective] present in test data point [True]
457 Text feature [inhibitor] present in test data point [True]
469 Text feature [provided] present in test data point [True]
470 Text feature [cells] present in test data point [True]
485 Text feature [regulated] present in test data point [True]
491 Text feature [properties] present in test data point [True]
496 Text feature [phospho] present in test data point [True]
Out of the top  500  features  64 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [72]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[1.020e-02 6.500e-02 3.000e-04 6.900e-03 3.100e-03 1.300e-03 9.111
e-01
  2.100e-03 0.000e+00]]
Actual Class : 7
--------------------------------------------------
13 Text feature [activated] present in test data point [True]
26 Text feature [constitutive] present in test data point [True]
28 Text feature [downstream] present in test data point [True]
32 Text feature [insertion] present in test data point [True]
38 Text feature [overexpression] present in test data point [True]
47 Text feature [2a] present in test data point [True]
57 Text feature [enhanced] present in test data point [True]
68 Text feature [positive] present in test data point [True]
72 Text feature [elevated] present in test data point [True]
76 Text feature [activation] present in test data point [True]
79 Text feature [ligand] present in test data point [True]
92 Text feature [concentrations] present in test data point [True]
112 Text feature [3b] present in test data point [True]
115 Text feature [inhibited] present in test data point [True]
122 Text feature [bone] present in test data point [True]
```

```
125 Text feature [phosphorylated] present in test data point [True]
126 Text feature [mapk] present in test data point [True]
133 Text feature [approximately] present in test data point [True]
166 Text feature [lung] present in test data point [True]
188 Text feature [mechanisms] present in test data point [True]
192 Text feature [leukemia] present in test data point [True]
199 Text feature [activate] present in test data point [True]
215 Text feature [signaling] present in test data point [True]
238 Text feature [transformed] present in test data point [True]
241 Text feature [factor] present in test data point [True]
248 Text feature [examined] present in test data point [True]
251 Text feature [transformation] present in test data point [True]
259 Text feature [ras] present in test data point [True]
282 Text feature [presence] present in test data point [True]
286 Text feature [2b] present in test data point [True]
297 Text feature [lead] present in test data point [True]
308 Text feature [expressing] present in test data point [True]
326 Text feature [bp] present in test data point [True]
359 Text feature [occur] present in test data point [True]
360 Text feature [previously] present in test data point [True]
379 Text feature [proliferation] present in test data point [True]
383 Text feature [observations] present in test data point [True]
397 Text feature [promote] present in test data point [True]
405 Text feature [akt] present in test data point [True]
415 Text feature [constitutively] present in test data point [True]
417 Text feature [carcinomas] present in test data point [True]
427 Text feature [pathways] present in test data point [True]
430 Text feature [days] present in test data point [True]
436 Text feature [current] present in test data point [True]
437 Text feature [72] present in test data point [True]
456 Text feature [effective] present in test data point [True]
457 Text feature [inhibitor] present in test data point [True]
469 Text feature [provided] present in test data point [True]
470 Text feature [cells] present in test data point [True]
485 Text feature [regulated] present in test data point [True]
491 Text feature [properties] present in test data point [True]
493 Text feature [ph] present in test data point [True]
496 Text feature [phospho] present in test data point [True]
Out of the top  500  features  53 are present in query point
```

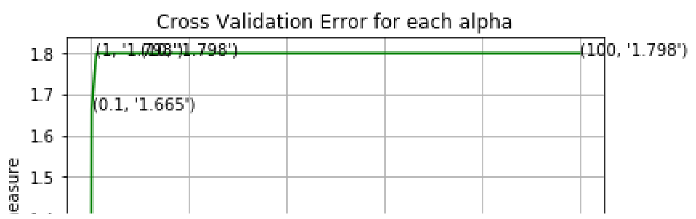
4.4. Linear Support Vector Machines

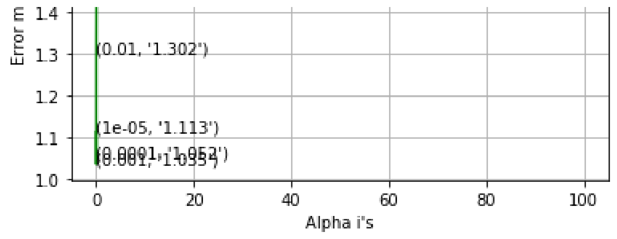
4.4.1. Hyper paramter tuning

In [73]:

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

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

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



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

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

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


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

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

```
for C = 1e-05
Log Loss : 1.1127115889802064
for C = 0.0001
Log Loss : 1.052082601599579
for C = 0.001
Log Loss : 1.0350029013925814
for C = 0.01
Log Loss : 1.3022828665341786
for C = 0.1
Log Loss : 1.6654604716465948
for C = 1
Log Loss : 1.797950705946721
for C = 10
Log Loss : 1.7979506780078758
for C = 100
Log Loss : 1.797950739227157
```





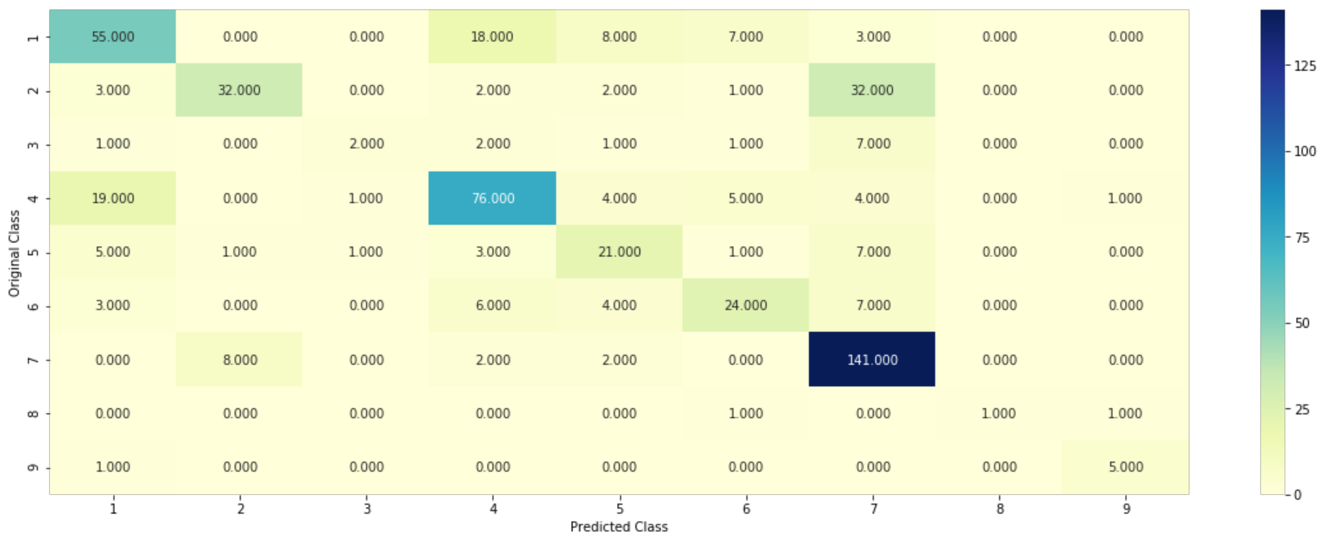
```
For values of best alpha =  0.001 The train log loss is: 0.5829917038128628
For values of best alpha =  0.001 The cross validation log loss is: 1.0350029013925814
For values of best alpha =  0.001 The test log loss is: 1.0438151454336577
```

4.4.2. Testing model with best hyper parameters

In [74]:

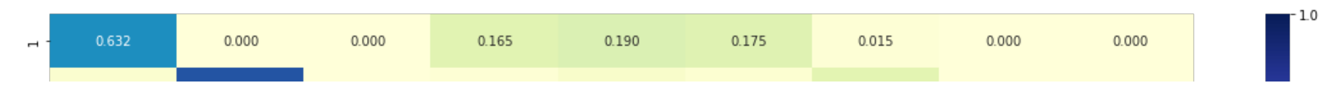
```python
# read more about support vector machines with linear kernals here http://scikit-
learn.org/stable/modules/generated/sklearn.svm.SVC.html

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, t
ol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', ra
ndom_state=None)

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


# 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 : 1.0350029013925814
Number of mis-classified points : 0.32894736842105265
-------------------- Confusion matrix --------------------
```

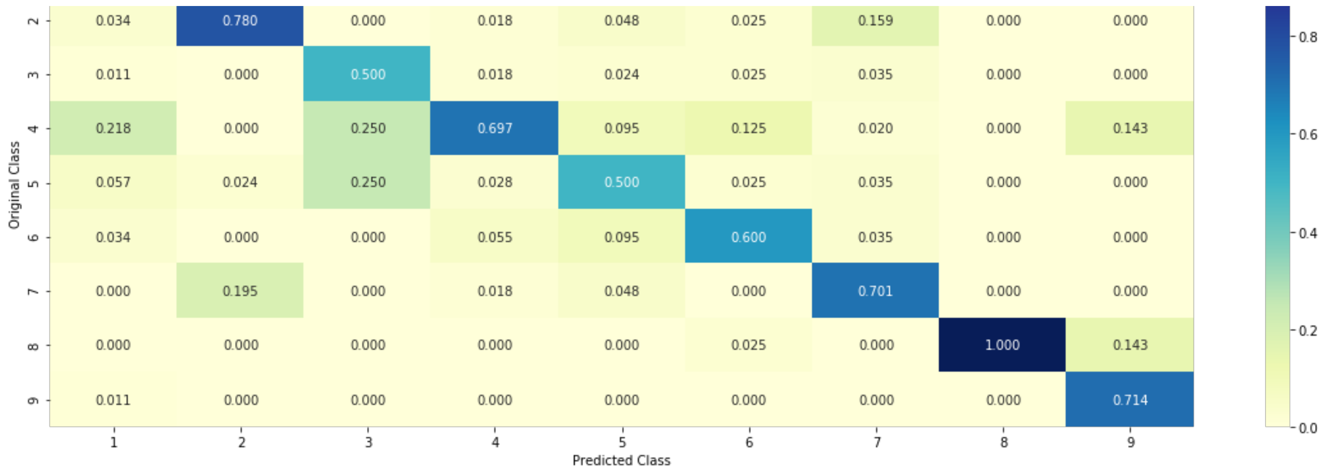


Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	55.000	0.000	0.000	18.000	8.000	7.000	3.000	0.000	0.000
2	3.000	32.000	0.000	2.000	2.000	1.000	32.000	0.000	0.000
3	1.000	0.000	2.000	2.000	1.000	1.000	7.000	0.000	0.000
4	19.000	0.000	1.000	76.000	4.000	5.000	4.000	0.000	1.000
5	5.000	1.000	1.000	3.000	21.000	1.000	7.000	0.000	0.000
6	3.000	0.000	0.000	6.000	4.000	24.000	7.000	0.000	0.000
7	0.000	8.000	0.000	2.000	2.000	0.000	141.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	1.000	0.000	1.000	1.000
9	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	5.000

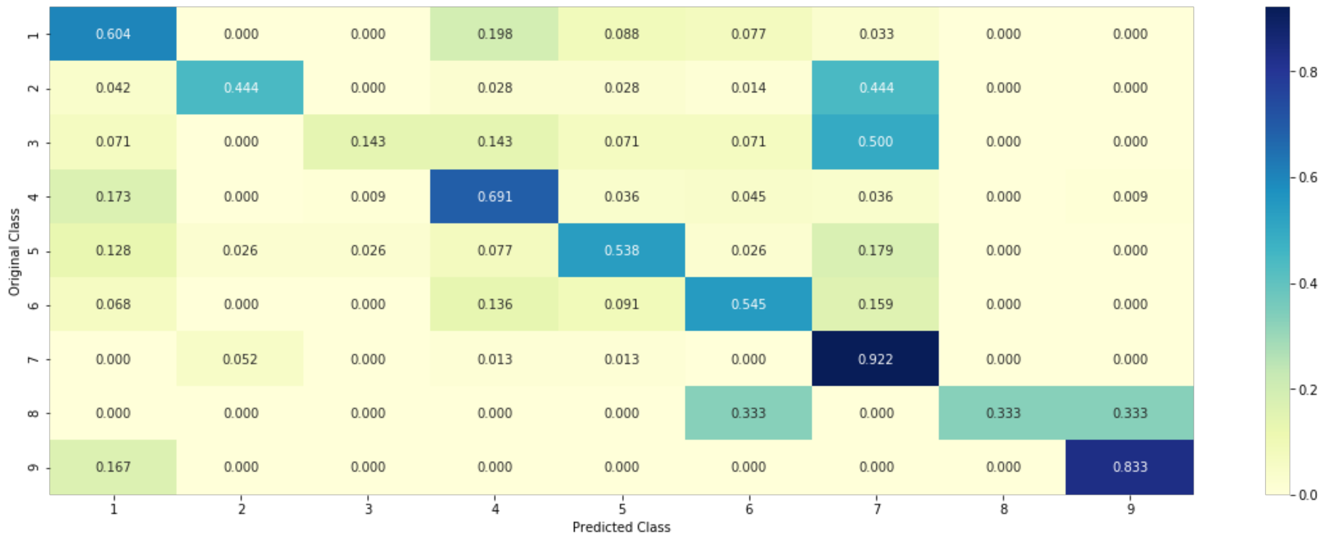
```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



	1	2	3	4	5	6	7	8	9
1	0.632	0.000	0.000	0.165	0.190	0.175	0.015	0.000	0.000



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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [75]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1234 0.0798 0.0157 0.0873 0.0868 0.0346 0.5569 0.0074 0.0081]]
Actual Class : 7
--------------------------------------------------
8 Text feature [constitutive] present in test data point [True]
19 Text feature [codon] present in test data point [True]
20 Text feature [ligand] present in test data point [True]
21 Text feature [activated] present in test data point [True]
```

```
22 Text feature [2a] present in test data point [True]
31 Text feature [activation] present in test data point [True]
32 Text feature [enhanced] present in test data point [True]
37 Text feature [leukemia] present in test data point [True]
38 Text feature [concentrations] present in test data point [True]
40 Text feature [inhibited] present in test data point [True]
42 Text feature [oncogene] present in test data point [True]
44 Text feature [downstream] present in test data point [True]
45 Text feature [positive] present in test data point [True]
46 Text feature [000] present in test data point [True]
47 Text feature [fold] present in test data point [True]
59 Text feature [signaling] present in test data point [True]
60 Text feature [mapk] present in test data point [True]
61 Text feature [3b] present in test data point [True]
62 Text feature [activate] present in test data point [True]
64 Text feature [akt] present in test data point [True]
65 Text feature [presence] present in test data point [True]
66 Text feature [factor] present in test data point [True]
67 Text feature [2b] present in test data point [True]
68 Text feature [egf] present in test data point [True]
69 Text feature [bone] present in test data point [True]
70 Text feature [approximately] present in test data point [True]
71 Text feature [expressing] present in test data point [True]
72 Text feature [wt] present in test data point [True]
73 Text feature [activating] present in test data point [True]
74 Text feature [overexpression] present in test data point [True]
75 Text feature [receptors] present in test data point [True]
76 Text feature [elevated] present in test data point [True]
77 Text feature [constitutively] present in test data point [True]
78 Text feature [lung] present in test data point [True]
79 Text feature [s3] present in test data point [True]
274 Text feature [advanced] present in test data point [True]
275 Text feature [effective] present in test data point [True]
276 Text feature [examined] present in test data point [True]
278 Text feature [2003] present in test data point [True]
279 Text feature [gfp] present in test data point [True]
281 Text feature [promote] present in test data point [True]
284 Text feature [versus] present in test data point [True]
285 Text feature [transformed] present in test data point [True]
286 Text feature [lead] present in test data point [True]
288 Text feature [position] present in test data point [True]
289 Text feature [malignant] present in test data point [True]
290 Text feature [pathways] present in test data point [True]
291 Text feature [sensitive] present in test data point [True]
445 Text feature [occur] present in test data point [True]
446 Text feature [regulated] present in test data point [True]
447 Text feature [properties] present in test data point [True]
448 Text feature [absence] present in test data point [True]
451 Text feature [cells] present in test data point [True]
453 Text feature [days] present in test data point [True]
454 Text feature [gain] present in test data point [True]
457 Text feature [bp] present in test data point [True]
459 Text feature [express] present in test data point [True]
460 Text feature [inhibitor] present in test data point [True]
462 Text feature [proliferation] present in test data point [True]
463 Text feature [culture] present in test data point [True]
465 Text feature [mutant] present in test data point [True]
466 Text feature [tyrosine] present in test data point [True]
467 Text feature [mechanisms] present in test data point [True]
468 Text feature [phase] present in test data point [True]
469 Text feature [stat3] present in test data point [True]
471 Text feature [suggest] present in test data point [True]
472 Text feature [colony] present in test data point [True]
473 Text feature [strong] present in test data point [True]
474 Text feature [membrane] present in test data point [True]
475 Text feature [reaction] present in test data point [True]
476 Text feature [carcinoma] present in test data point [True]
477 Text feature [provided] present in test data point [True]
478 Text feature [previously] present in test data point [True]
479 Text feature [phosphorylation] present in test data point [True]
481 Text feature [day] present in test data point [True]
482 Text feature [mutants] present in test data point [True]
483 Text feature [current] present in test data point [True]
484 Text feature [cdna] present in test data point [True]
485 Text feature [mm] present in test data point [True]
486 Text feature [serum] present in test data point [True]
489 Text feature [derived] present in test data point [True]
```

```
490 Text feature [increased] present in test data point [True]
492 Text feature [independent] present in test data point [True]
494 Text feature [51] present in test data point [True]
495 Text feature [high] present in test data point [True]
497 Text feature [medium] present in test data point [True]
498 Text feature [without] present in test data point [True]
499 Text feature [per] present in test data point [True]
Out of the top  500  features  88 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [76]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0383 0.127  0.0018 0.0167 0.0096 0.0039 0.7995 0.0016 0.0016]]
Actual Class : 7
--------------------------------------------------
8 Text feature [constitutive] present in test data point [True]
20 Text feature [ligand] present in test data point [True]
21 Text feature [activated] present in test data point [True]
22 Text feature [2a] present in test data point [True]
30 Text feature [insertion] present in test data point [True]
31 Text feature [activation] present in test data point [True]
32 Text feature [enhanced] present in test data point [True]
37 Text feature [leukemia] present in test data point [True]
38 Text feature [concentrations] present in test data point [True]
39 Text feature [phosphorylated] present in test data point [True]
40 Text feature [inhibited] present in test data point [True]
44 Text feature [downstream] present in test data point [True]
45 Text feature [positive] present in test data point [True]
59 Text feature [signaling] present in test data point [True]
60 Text feature [mapk] present in test data point [True]
61 Text feature [3b] present in test data point [True]
62 Text feature [activate] present in test data point [True]
64 Text feature [akt] present in test data point [True]
65 Text feature [presence] present in test data point [True]
66 Text feature [factor] present in test data point [True]
67 Text feature [2b] present in test data point [True]
69 Text feature [bone] present in test data point [True]
70 Text feature [approximately] present in test data point [True]
71 Text feature [expressing] present in test data point [True]
74 Text feature [overexpression] present in test data point [True]
76 Text feature [elevated] present in test data point [True]
77 Text feature [constitutively] present in test data point [True]
78 Text feature [lung] present in test data point [True]
275 Text feature [effective] present in test data point [True]
276 Text feature [examined] present in test data point [True]
277 Text feature [transforming] present in test data point [True]
281 Text feature [promote] present in test data point [True]
283 Text feature [ph] present in test data point [True]
285 Text feature [transformed] present in test data point [True]
286 Text feature [lead] present in test data point [True]
287 Text feature [ras] present in test data point [True]
290 Text feature [pathways] present in test data point [True]
291 Text feature [sensitive] present in test data point [True]
292 Text feature [transformation] present in test data point [True]
445 Text feature [occur] present in test data point [True]
446 Text feature [regulated] present in test data point [True]
447 Text feature [properties] present in test data point [True]
448 Text feature [absence] present in test data point [True]
451 Text feature [cells] present in test data point [True]
```

```
453 Text feature [days] present in test data point [True]
455 Text feature [raf] present in test data point [True]
456 Text feature [extracellular] present in test data point [True]
457 Text feature [bp] present in test data point [True]
460 Text feature [inhibitor] present in test data point [True]
462 Text feature [proliferation] present in test data point [True]
465 Text feature [mutant] present in test data point [True]
466 Text feature [tyrosine] present in test data point [True]
467 Text feature [mechanisms] present in test data point [True]
468 Text feature [phase] present in test data point [True]
471 Text feature [suggest] present in test data point [True]
474 Text feature [membrane] present in test data point [True]
475 Text feature [reaction] present in test data point [True]
477 Text feature [provided] present in test data point [True]
478 Text feature [previously] present in test data point [True]
479 Text feature [phosphorylation] present in test data point [True]
481 Text feature [day] present in test data point [True]
483 Text feature [current] present in test data point [True]
486 Text feature [serum] present in test data point [True]
487 Text feature [72] present in test data point [True]
489 Text feature [derived] present in test data point [True]
490 Text feature [increased] present in test data point [True]
491 Text feature [ba] present in test data point [True]
492 Text feature [independent] present in test data point [True]
494 Text feature [51] present in test data point [True]
495 Text feature [high] present in test data point [True]
496 Text feature [f3] present in test data point [True]
497 Text feature [medium] present in test data point [True]
498 Text feature [without] present in test data point [True]
Out of the top  500  features  73 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [77]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
verbose=0, warm_start=False,
# class_weight=None)

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

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

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


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

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

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

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

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

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

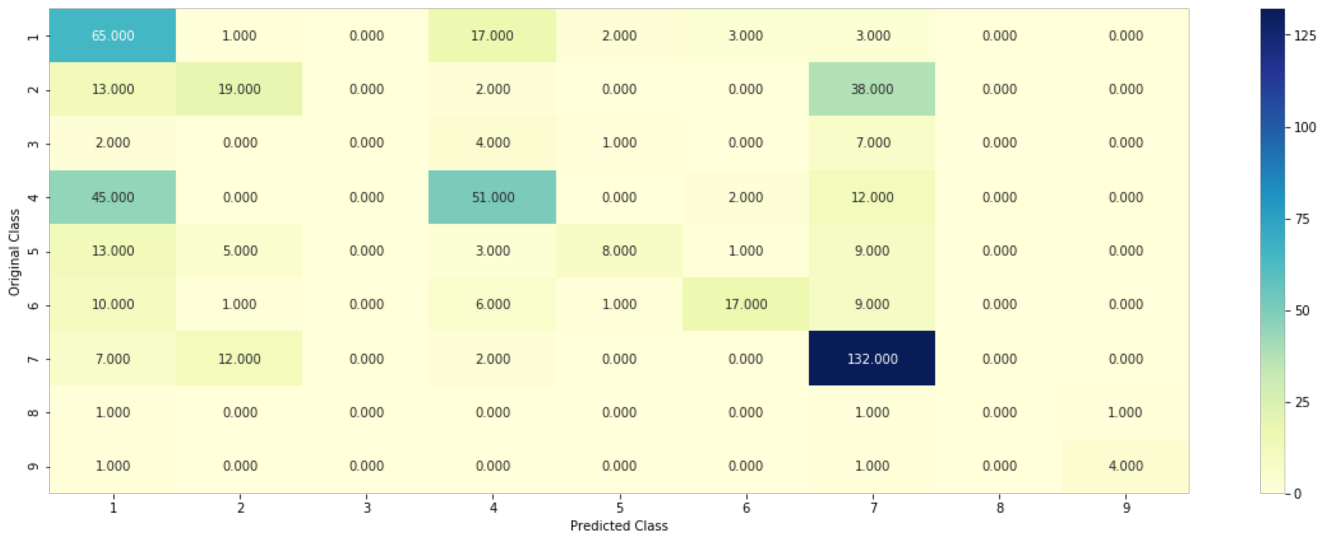
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2352835385617826
for n_estimators = 100 and max depth =  10
Log Loss : 1.2561755601394726
for n_estimators = 200 and max depth =  5
Log Loss : 1.2201457180000816
for n_estimators = 200 and max depth =  10
Log Loss : 1.2497704639105234
for n_estimators = 500 and max depth =  5
Log Loss : 1.2086034285708824
for n_estimators = 500 and max depth =  10
Log Loss : 1.2415584374992874
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2041099750748856
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2407150153425046
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2005955895402045
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2393323016147653
For values of best estimator =  2000 The train log loss is: 0.8558260547625659
For values of best estimator =  2000 The cross validation log loss is: 1.2005955895402045
For values of best estimator =  2000 The test log loss is: 1.193747883227259
```

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

In [78]:

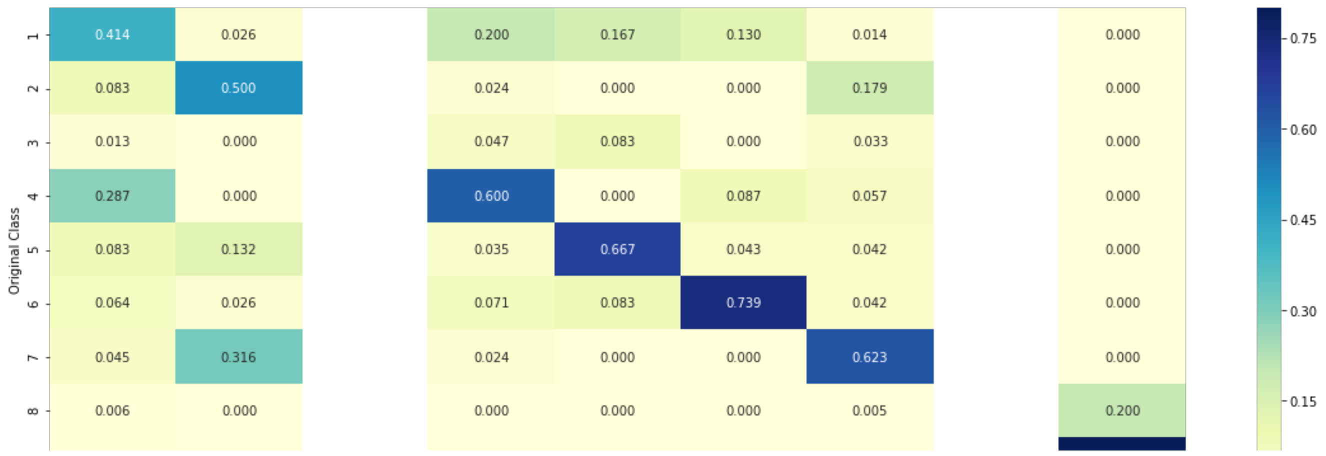
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
verbose=0, warm_start=False,
# class_weight=None)

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

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

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

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

```
Log loss : 1.2005955895402045
Number of mis-classified points : 0.44360902255639095
-------------------- Confusion matrix --------------------
```

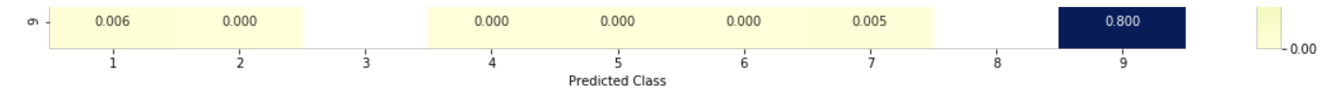


Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	65.000	1.000	0.000	17.000	2.000	3.000	3.000	0.000	0.000
2	13.000	19.000	0.000	2.000	0.000	0.000	38.000	0.000	0.000
3	2.000	0.000	0.000	4.000	1.000	0.000	7.000	0.000	0.000
4	45.000	0.000	0.000	51.000	0.000	2.000	12.000	0.000	0.000
5	13.000	5.000	0.000	3.000	8.000	1.000	9.000	0.000	0.000
6	10.000	1.000	0.000	6.000	1.000	17.000	9.000	0.000	0.000
7	7.000	12.000	0.000	2.000	0.000	0.000	132.000	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	1.000
9	1.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	4.000

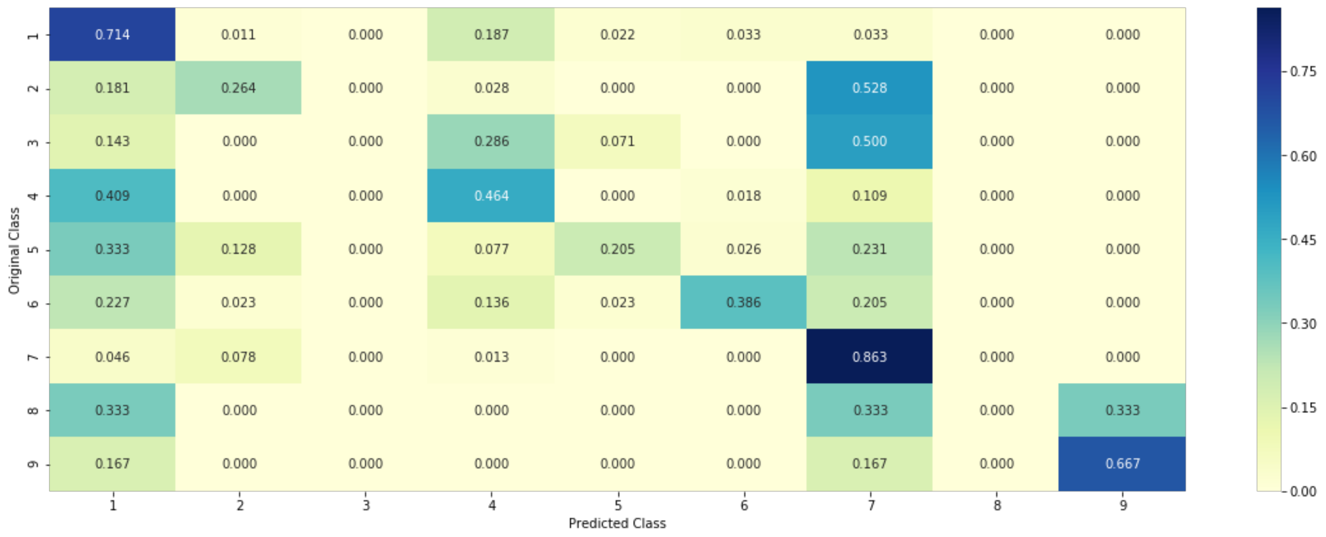
```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



Original Class \ Predicted Class	1	2	3	4	5	6	7	8	9
1	0.414	0.026		0.200	0.167	0.130	0.014		0.000
2	0.083	0.500		0.024	0.000	0.000	0.179		0.000
3	0.013	0.000		0.047	0.083	0.000	0.033		0.000
4	0.287	0.000		0.600	0.000	0.087	0.057		0.000
5	0.083	0.132		0.035	0.667	0.043	0.042		0.000
6	0.064	0.026		0.071	0.083	0.739	0.042		0.000
7	0.045	0.316		0.024	0.000	0.000	0.623		0.000
8	0.006	0.000		0.000	0.000	0.000	0.005		0.200



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [79]:

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

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1347 0.1078 0.0212 0.1223 0.0581 0.0467 0.4865 0.0091 0.0136]]
Actual Class : 7
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [inhibitors] present in test data point [True]
2 Text feature [activating] present in test data point [True]
3 Text feature [activation] present in test data point [True]
4 Text feature [constitutive] present in test data point [True]
5 Text feature [function] present in test data point [True]
6 Text feature [tyrosine] present in test data point [True]
7 Text feature [suppressor] present in test data point [True]
8 Text feature [phosphorylation] present in test data point [True]
9 Text feature [loss] present in test data point [True]
10 Text feature [activated] present in test data point [True]
12 Text feature [treatment] present in test data point [True]
16 Text feature [inhibitor] present in test data point [True]
17 Text feature [protein] present in test data point [True]
18 Text feature [signaling] present in test data point [True]
```

```
19 Text feature [receptor] present in test data point [True]
20 Text feature [constitutively] present in test data point [True]
21 Text feature [variants] present in test data point [True]
23 Text feature [cell] present in test data point [True]
24 Text feature [expression] present in test data point [True]
25 Text feature [functional] present in test data point [True]
26 Text feature [pten] present in test data point [True]
27 Text feature [growth] present in test data point [True]
28 Text feature [cells] present in test data point [True]
29 Text feature [kinases] present in test data point [True]
31 Text feature [therapy] present in test data point [True]
32 Text feature [therapeutic] present in test data point [True]
33 Text feature [stability] present in test data point [True]
36 Text feature [neutral] present in test data point [True]
37 Text feature [akt] present in test data point [True]
40 Text feature [clinical] present in test data point [True]
42 Text feature [proteins] present in test data point [True]
44 Text feature [patients] present in test data point [True]
45 Text feature [phosphatase] present in test data point [True]
48 Text feature [treated] present in test data point [True]
49 Text feature [months] present in test data point [True]
51 Text feature [inhibited] present in test data point [True]
52 Text feature [trials] present in test data point [True]
53 Text feature [predicted] present in test data point [True]
55 Text feature [activate] present in test data point [True]
61 Text feature [null] present in test data point [True]
66 Text feature [response] present in test data point [True]
67 Text feature [functions] present in test data point [True]
68 Text feature [resistance] present in test data point [True]
69 Text feature [advanced] present in test data point [True]
70 Text feature [downstream] present in test data point [True]
73 Text feature [splice] present in test data point [True]
74 Text feature [drug] present in test data point [True]
75 Text feature [inhibition] present in test data point [True]
76 Text feature [lines] present in test data point [True]
77 Text feature [dna] present in test data point [True]
78 Text feature [proliferation] present in test data point [True]
79 Text feature [p53] present in test data point [True]
80 Text feature [expected] present in test data point [True]
81 Text feature [database] present in test data point [True]
82 Text feature [survival] present in test data point [True]
83 Text feature [variant] present in test data point [True]
84 Text feature [potential] present in test data point [True]
85 Text feature [sensitive] present in test data point [True]
89 Text feature [sequencing] present in test data point [True]
90 Text feature [oncogene] present in test data point [True]
91 Text feature [ligand] present in test data point [True]
92 Text feature [harboring] present in test data point [True]
94 Text feature [expressing] present in test data point [True]
95 Text feature [activity] present in test data point [True]
96 Text feature [phospho] present in test data point [True]
98 Text feature [assays] present in test data point [True]
Out of the top  100  features  67 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [80]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0435 0.3127 0.0147 0.0295 0.0394 0.0409 0.5094 0.0076 0.0024]]
Actuall Class : 7
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
```

```
0 Text feature [kinase] present in test data point [True]
1 Text feature [inhibitors] present in test data point [True]
3 Text feature [activation] present in test data point [True]
4 Text feature [constitutive] present in test data point [True]
5 Text feature [function] present in test data point [True]
6 Text feature [tyrosine] present in test data point [True]
8 Text feature [phosphorylation] present in test data point [True]
9 Text feature [loss] present in test data point [True]
10 Text feature [activated] present in test data point [True]
12 Text feature [treatment] present in test data point [True]
14 Text feature [oncogenic] present in test data point [True]
16 Text feature [inhibitor] present in test data point [True]
17 Text feature [protein] present in test data point [True]
18 Text feature [signaling] present in test data point [True]
19 Text feature [receptor] present in test data point [True]
20 Text feature [constitutively] present in test data point [True]
21 Text feature [variants] present in test data point [True]
23 Text feature [cell] present in test data point [True]
24 Text feature [expression] present in test data point [True]
27 Text feature [growth] present in test data point [True]
28 Text feature [cells] present in test data point [True]
29 Text feature [kinases] present in test data point [True]
30 Text feature [erk] present in test data point [True]
31 Text feature [therapy] present in test data point [True]
32 Text feature [therapeutic] present in test data point [True]
34 Text feature [classified] present in test data point [True]
37 Text feature [akt] present in test data point [True]
40 Text feature [clinical] present in test data point [True]
42 Text feature [proteins] present in test data point [True]
44 Text feature [patients] present in test data point [True]
45 Text feature [phosphatase] present in test data point [True]
47 Text feature [transforming] present in test data point [True]
48 Text feature [treated] present in test data point [True]
49 Text feature [months] present in test data point [True]
51 Text feature [inhibited] present in test data point [True]
52 Text feature [trials] present in test data point [True]
53 Text feature [predicted] present in test data point [True]
54 Text feature [57] present in test data point [True]
55 Text feature [activate] present in test data point [True]
59 Text feature [ba] present in test data point [True]
60 Text feature [extracellular] present in test data point [True]
63 Text feature [inactivation] present in test data point [True]
65 Text feature [imatinib] present in test data point [True]
66 Text feature [response] present in test data point [True]
68 Text feature [resistance] present in test data point [True]
70 Text feature [downstream] present in test data point [True]
71 Text feature [mek] present in test data point [True]
73 Text feature [splice] present in test data point [True]
75 Text feature [inhibition] present in test data point [True]
76 Text feature [lines] present in test data point [True]
77 Text feature [dna] present in test data point [True]
78 Text feature [proliferation] present in test data point [True]
80 Text feature [expected] present in test data point [True]
82 Text feature [survival] present in test data point [True]
83 Text feature [variant] present in test data point [True]
84 Text feature [potential] present in test data point [True]
85 Text feature [sensitive] present in test data point [True]
86 Text feature [ic50] present in test data point [True]
87 Text feature [f3] present in test data point [True]
89 Text feature [sequencing] present in test data point [True]
91 Text feature [ligand] present in test data point [True]
94 Text feature [expressing] present in test data point [True]
95 Text feature [activity] present in test data point [True]
96 Text feature [phospho] present in test data point [True]
98 Text feature [assays] present in test data point [True]
Out of the top  100  features  65 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [81]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
verbose=0, warm_start=False,
# class_weight=None)

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

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

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


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

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

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

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

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

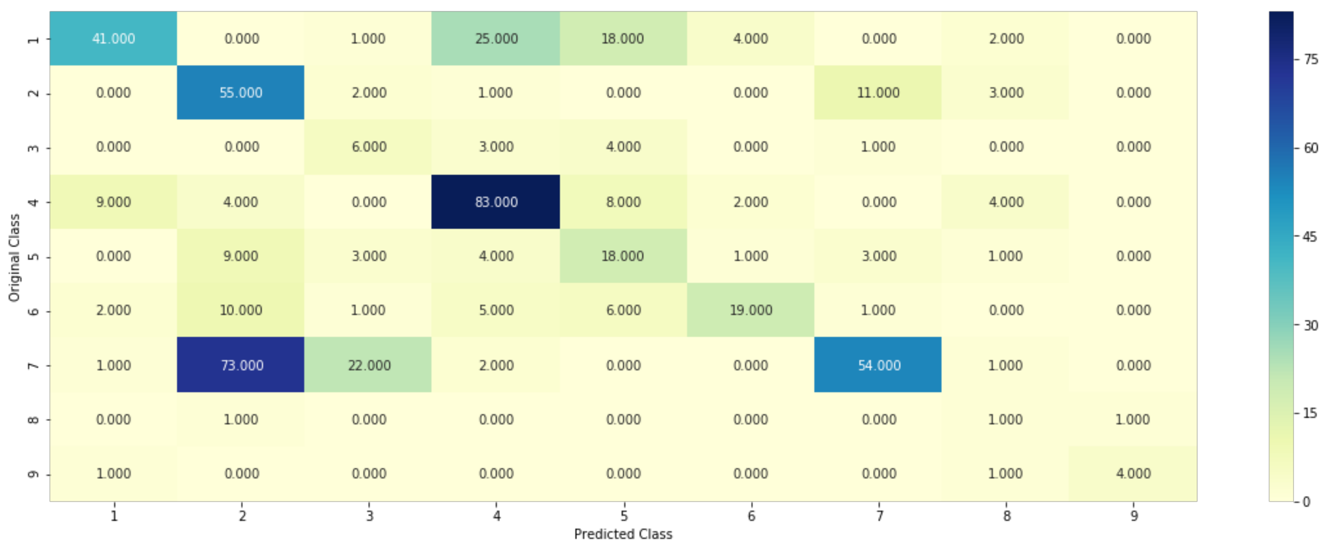
```
for n_estimators = 10 and max depth =  2
Log Loss : 2.0785756107879925
for n_estimators = 10 and max depth =  3
Log Loss : 1.7796114115816195
for n_estimators = 10 and max depth =  5
Log Loss : 1.5694449342981143
for n_estimators = 10 and max depth =  10
Log Loss : 1.657816298826748
for n_estimators = 50 and max depth =  2
Log Loss : 1.6617004025996724
for n_estimators = 50 and max depth =  3
Log Loss : 1.4800792162013208
for n_estimators = 50 and max depth =  5
Log Loss : 1.3513609488357476
for n_estimators = 50 and max depth =  10
Log Loss : 1.5211250630935944
for n_estimators = 100 and max depth =  2
Log Loss : 1.527619885776052
for n_estimators = 100 and max depth =  3
Log Loss : 1.5047685332655452
for n_estimators = 100 and max depth =  5
Log Loss : 1.2849039948988856
for n_estimators = 100 and max depth =  10
Log Loss : 1.547251816192383
for n_estimators = 200 and max depth =  2
Log Loss : 1.6040861532378936
for n_estimators = 200 and max depth =  3
Log Loss : 1.5458649287252697
for n_estimators = 200 and max depth =  5
Log Loss : 1.3414045603977711
for n_estimators = 200 and max depth =  10
Log Loss : 1.580066209186053
for n_estimators = 500 and max depth =  2
Log Loss : 1.6752845392819173
for n_estimators = 500 and max depth =  3
Log Loss : 1.6030874490387306
for n_estimators = 500 and max depth =  5
Log Loss : 1.377347599572653
for n_estimators = 500 and max depth =  10
Log Loss : 1.6193253552989664
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6365851633691797
for n_estimators = 1000 and max depth =  3
Log Loss : 1.5807434029803829
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3784217198878619
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6167620272181216
For values of best alpha =  100 The train log loss is: 0.053272816858339025
For values of best alpha =  100 The cross validation log loss is: 1.2849039948988856
For values of best alpha =  100 The test log loss is: 1.335529535482945
```

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

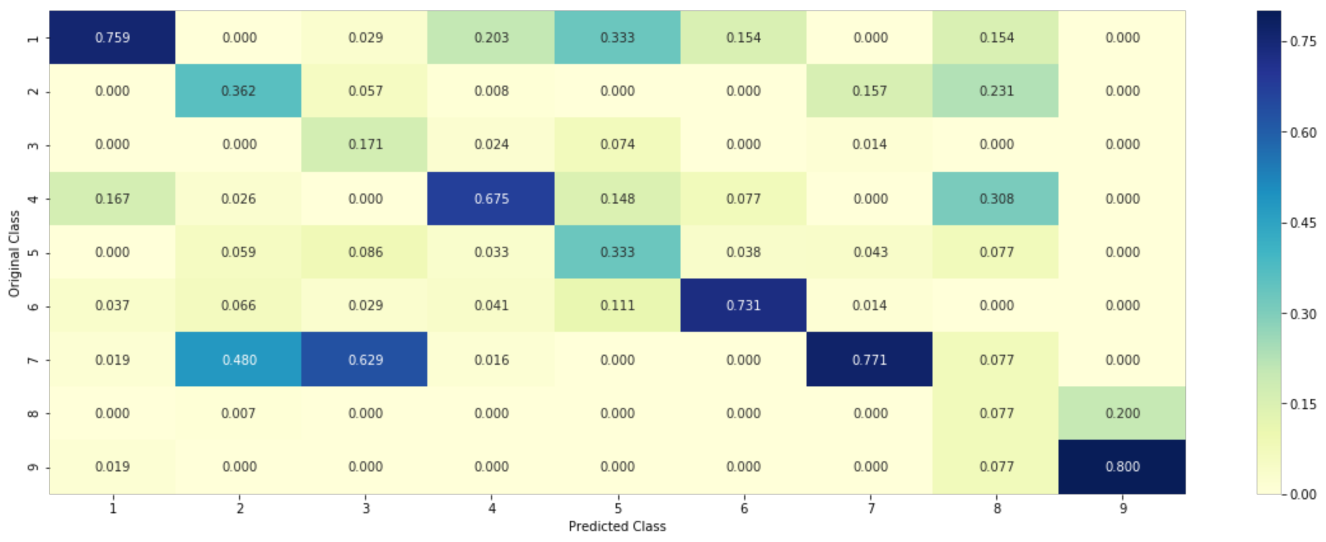
In [82]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# predict_proba (X) Perform classification on samples in X.
# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

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

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

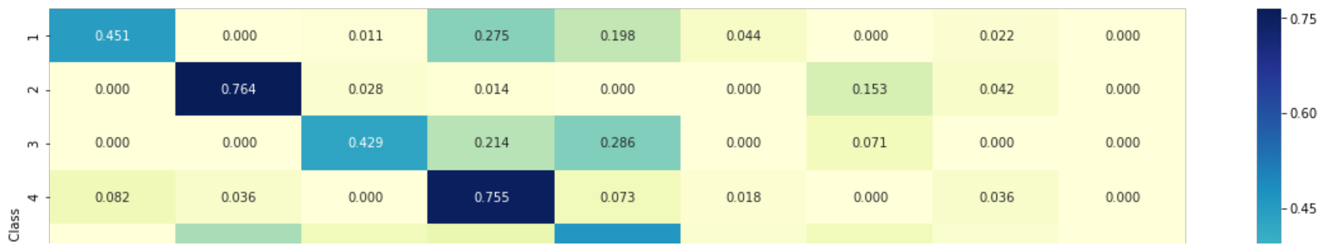
```
Log loss : 1.2849039948988856
Number of mis-classified points : 0.4718045112781955
-------------------- Confusion matrix --------------------
```

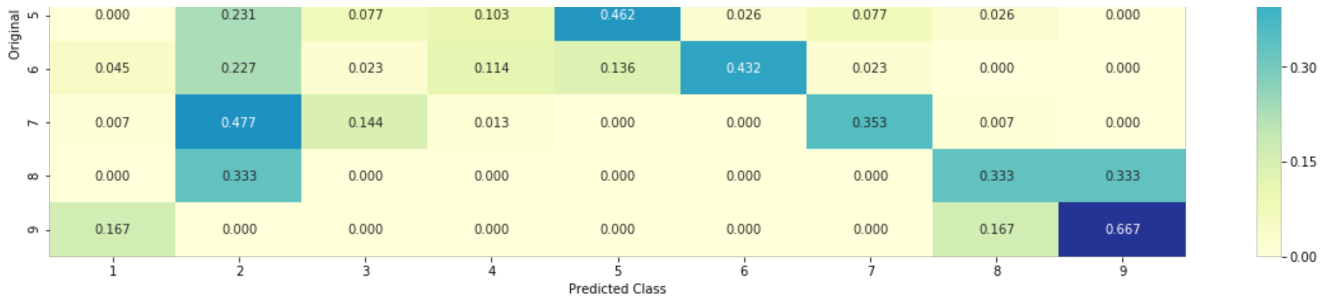


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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





4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [83]:

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


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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0251 0.1392 0.242  0.025  0.0344 0.0533 0.4303 0.0344 0.0163]]
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
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [84]:

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

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0236 0.4162 0.1169 0.0175 0.0214 0.0502 0.322  0.0196 0.0127]]
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
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [85]:

```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=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-in
tuition-1/
#------------------------------


# read more about support vector machines with linear kernals here http://scikit-
learn.org/stable/modules/generated/sklearn.svm.SVC.html
# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, t
ol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', ra
ndom_state=None)

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


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

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

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

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


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

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


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

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehot
Coding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y,
sig_clf2.predict_proba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding)))
)
```

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

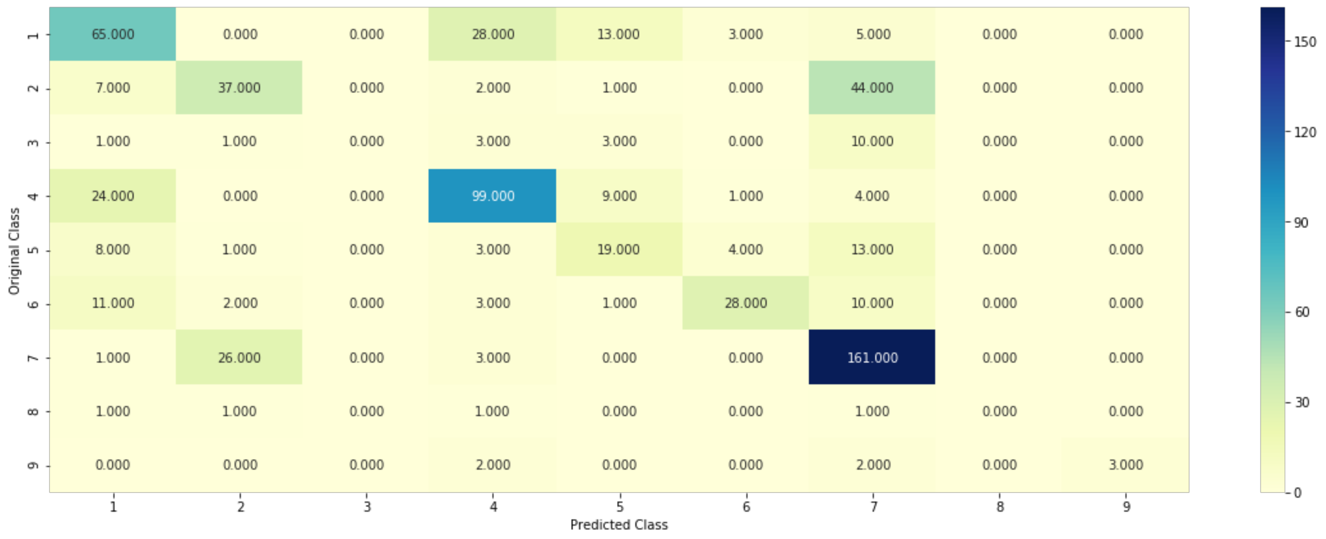
```
Logistic Regression :  Log Loss: 1.03
Support vector machines : Log Loss: 1.80
Naive Bayes : Log Loss: 1.22
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.177
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.031
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.496
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.170
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.418
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.929
```

4.7.2 testing the model with the best hyper parameters

In [86]:

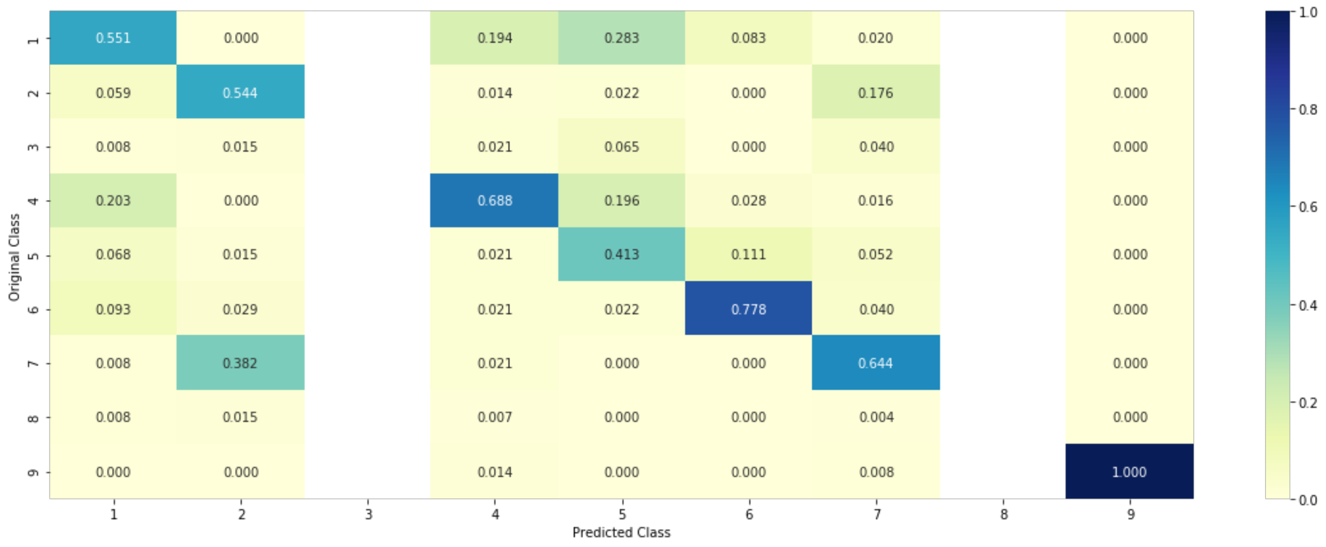
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba
s=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

```
Log loss (train) on the stacking classifier : 0.5408578433796639
Log loss (CV) on the stacking classifier : 1.170348879345548
Log loss (test) on the stacking classifier : 1.1809138319674743
Number of missclassified point : 0.3804511278195489
-------------------- Confusion matrix --------------------
```

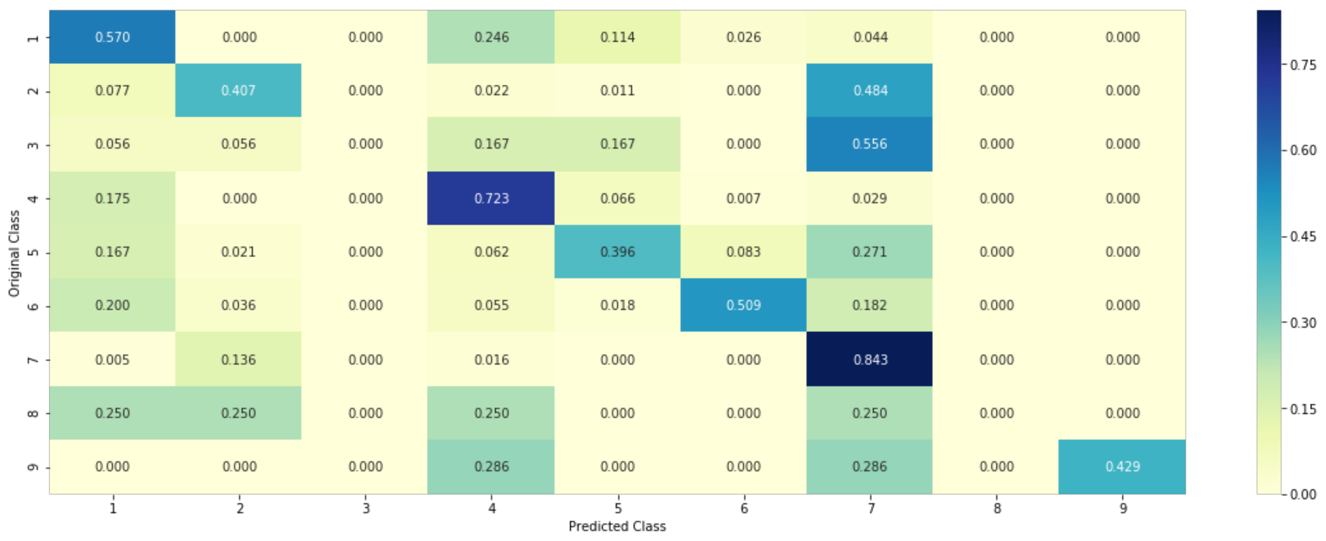


Original \ Predicted	1	2	3	4	5	6	7	8	9
1	65.000	0.000	0.000	28.000	13.000	3.000	5.000	0.000	0.000
2	7.000	37.000	0.000	2.000	1.000	0.000	44.000	0.000	0.000
3	1.000	1.000	0.000	3.000	3.000	0.000	10.000	0.000	0.000
4	24.000	0.000	0.000	99.000	9.000	1.000	4.000	0.000	0.000
5	8.000	1.000	0.000	3.000	19.000	4.000	13.000	0.000	0.000
6	11.000	2.000	0.000	3.000	1.000	28.000	10.000	0.000	0.000
7	1.000	26.000	0.000	3.000	0.000	0.000	161.000	0.000	0.000
8	1.000	1.000	0.000	1.000	0.000	0.000	1.000	0.000	0.000
9	0.000	0.000	0.000	2.000	0.000	0.000	2.000	0.000	3.000

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



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

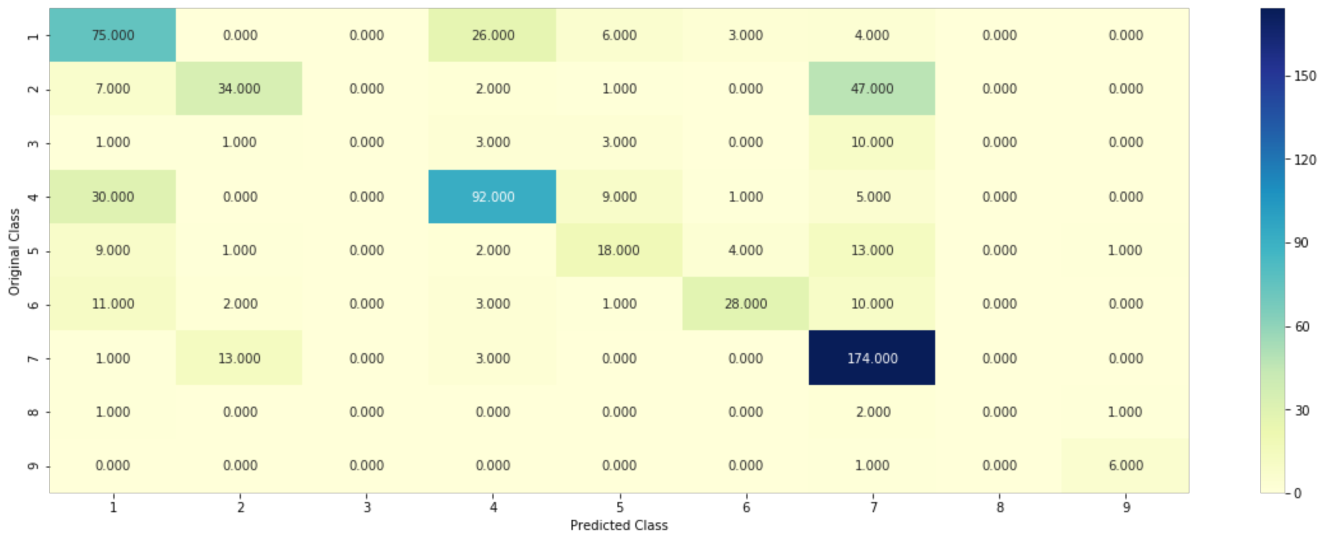


4.7.3 Maximum Voting classifier

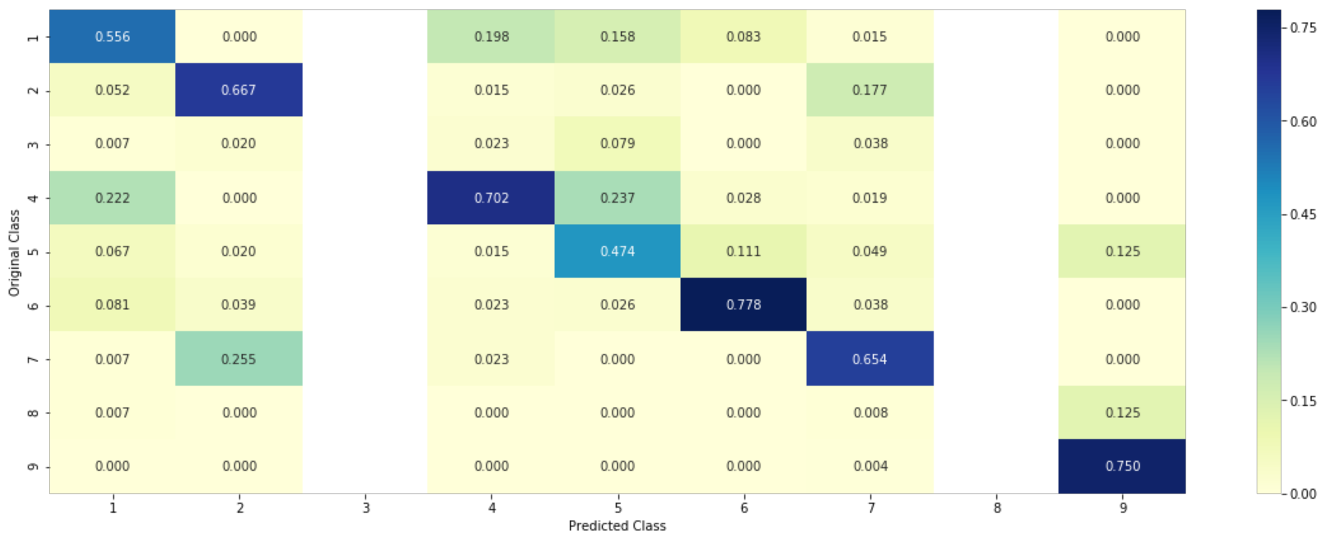
In [87]:

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

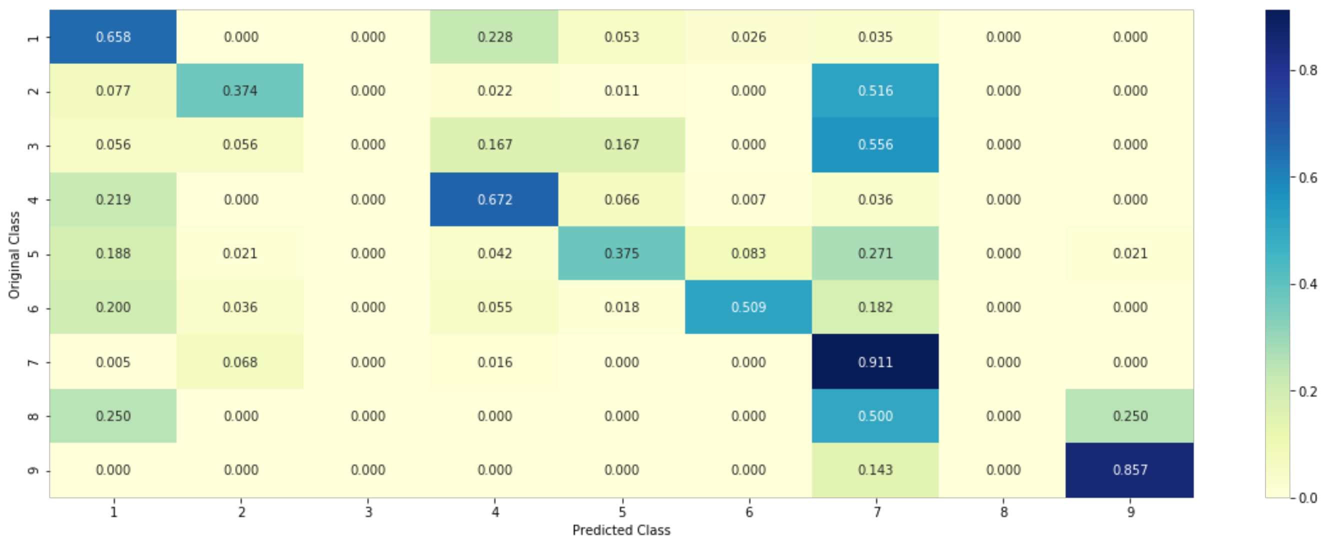
```
Log loss (train) on the VotingClassifier : 0.8345714707758667
Log loss (CV) on the VotingClassifier : 1.1869920170002433
Log loss (test) on the VotingClassifier : 1.18940847457613
Number of missclassified point : 0.35789473684210527
-------------------- Confusion matrix --------------------
```



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



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



Logistic regression with CountVectorizer Features, including both unigrams and bigrams

In [88]:

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = CountVectorizer(min_df=3,ngram_range=(1,2))
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

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


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

```
Total number of unique words in train data : 776065
```

In [89]:

```
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 [90]:

```
#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 [91]:

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

In [92]:

```
# 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 [93]:

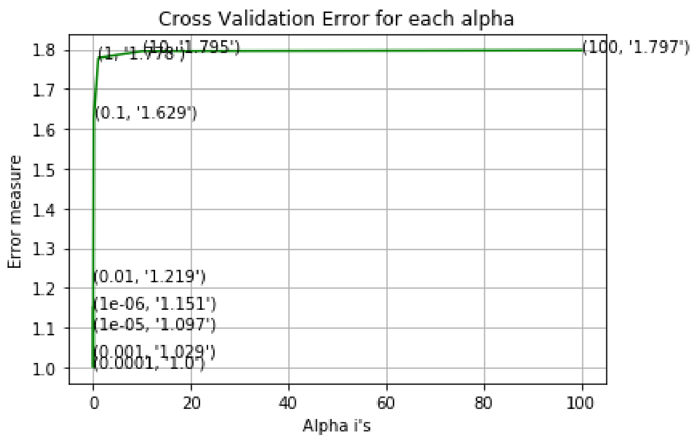
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

With class balancing

In [94]:

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

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

```
for alpha = 1e-06
Log Loss : 1.1507472519411477
for alpha = 1e-05
Log Loss : 1.09679268243564
for alpha = 0.0001
Log Loss : 1.000441942795181
for alpha = 0.001
Log Loss : 1.0289303735224615
for alpha = 0.01
Log Loss : 1.2187480579084689
for alpha = 0.1
Log Loss : 1.6292964049248162
for alpha = 1
Log Loss : 1.7780908869103667
for alpha = 10
Log Loss : 1.795341175139565
for alpha = 100
Log Loss : 1.7973772038939875
```

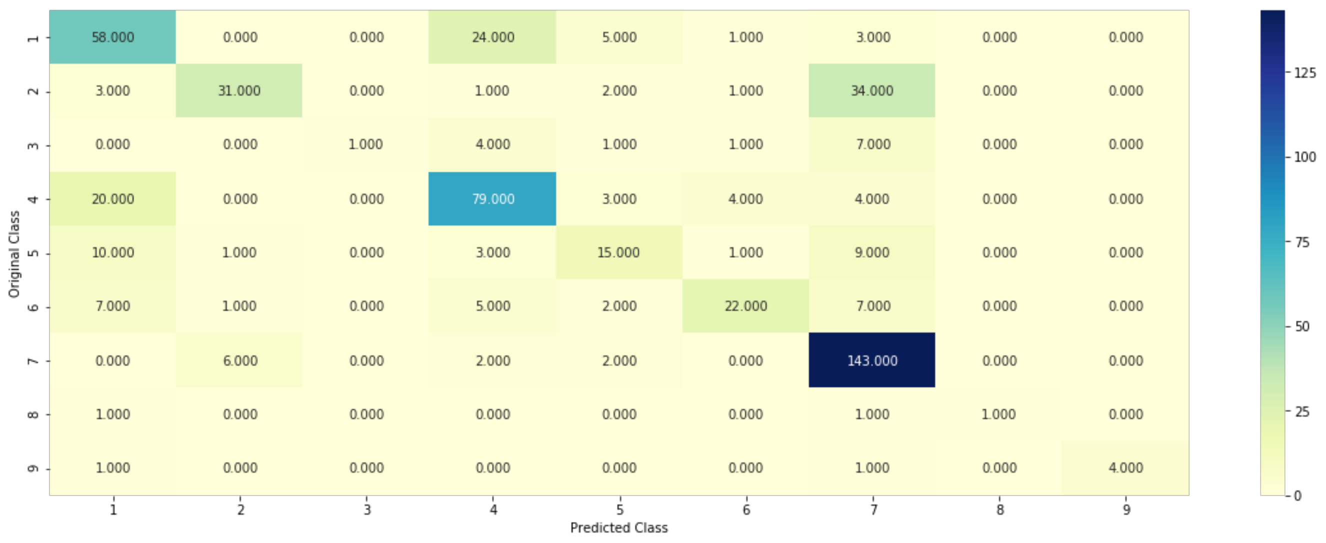


```
For values of best alpha =  0.0001 The train log loss is: 0.4445853332786352
For values of best alpha =  0.0001 The cross validation log loss is: 1.000441942795181
For values of best alpha =  0.0001 The test log loss is: 0.9713398242027661
```

In [95]:

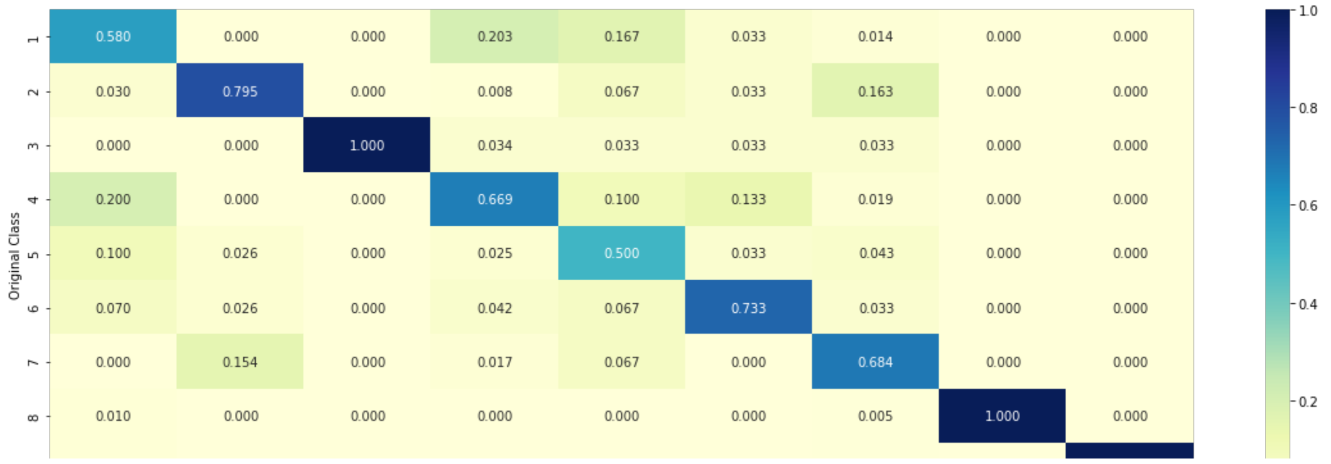
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

```
Log loss : 1.000441942795181
Number of mis-classified points : 0.33458646616541354
-------------------- Confusion matrix --------------------
```

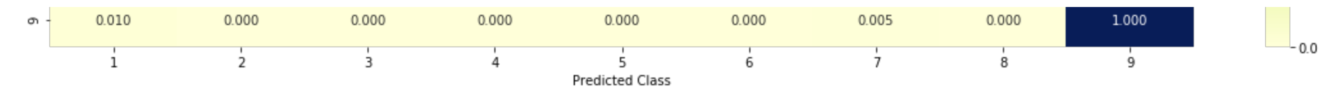


	1	2	3	4	5	6	7	8	9
1	58.000	0.000	0.000	24.000	5.000	1.000	3.000	0.000	0.000
2	3.000	31.000	0.000	1.000	2.000	1.000	34.000	0.000	0.000
3	0.000	0.000	1.000	4.000	1.000	1.000	7.000	0.000	0.000
4	20.000	0.000	0.000	79.000	3.000	4.000	4.000	0.000	0.000
5	10.000	1.000	0.000	3.000	15.000	1.000	9.000	0.000	0.000
6	7.000	1.000	0.000	5.000	2.000	22.000	7.000	0.000	0.000
7	0.000	6.000	0.000	2.000	2.000	0.000	143.000	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	0.000	1.000	1.000	0.000
9	1.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	4.000

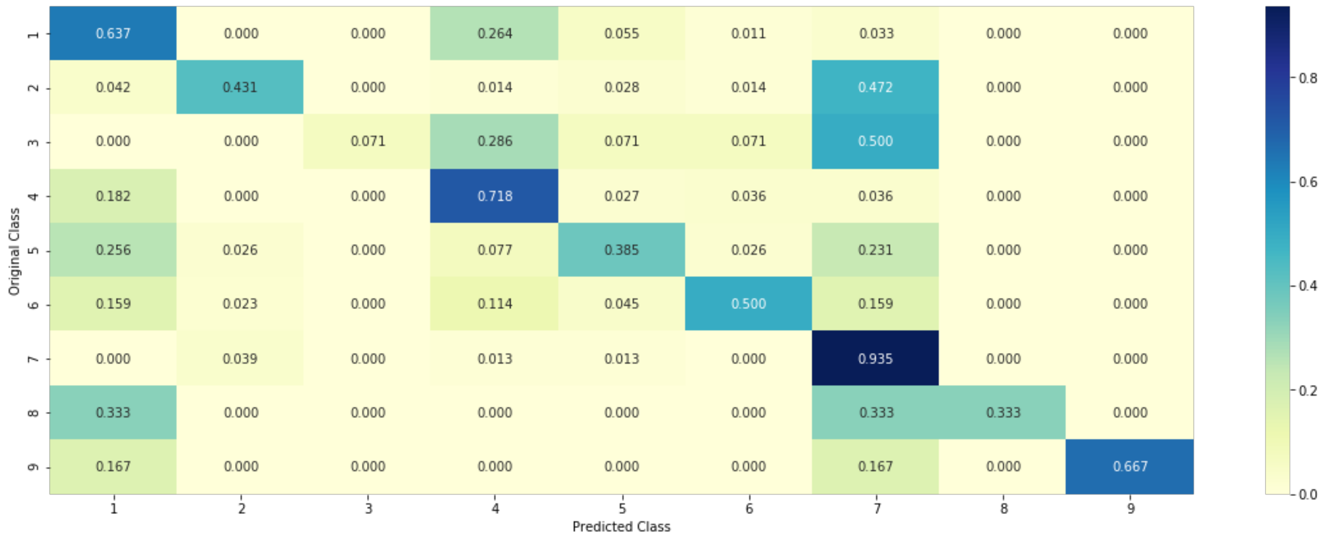
```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



1	0.580	0.000	0.000	0.203	0.167	0.033	0.014	0.000	0.000
2	0.030	0.795	0.000	0.008	0.067	0.033	0.163	0.000	0.000
3	0.000	0.000	1.000	0.034	0.033	0.033	0.033	0.000	0.000
4	0.200	0.000	0.000	0.669	0.100	0.133	0.019	0.000	0.000
5	0.100	0.026	0.000	0.025	0.500	0.033	0.043	0.000	0.000
6	0.070	0.026	0.000	0.042	0.067	0.733	0.033	0.000	0.000
7	0.000	0.154	0.000	0.017	0.067	0.000	0.684	0.000	0.000
8	0.010	0.000	0.000	0.000	0.000	0.000	0.005	1.000	0.000



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



In [96]:

```
## feature importance
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']))
```

In [97]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1359 0.0448 0.016  0.0436 0.1078 0.0357 0.6013 0.0055 0.0095]]
```

```
Actual Class : 7
--------------------------------------------------
8 Text feature [activated] present in test data point [True]
11 Text feature [constitutive] present in test data point [True]
17 Text feature [downstream] present in test data point [True]
18 Text feature [codon] present in test data point [True]
23 Text feature [overexpression] present in test data point [True]
29 Text feature [activation] present in test data point [True]
32 Text feature [enhanced] present in test data point [True]
34 Text feature [positive] present in test data point [True]
36 Text feature [2a] present in test data point [True]
43 Text feature [ligand] present in test data point [True]
54 Text feature [elevated] present in test data point [True]
59 Text feature [fold] present in test data point [True]
65 Text feature [3b] present in test data point [True]
67 Text feature [mapk] present in test data point [True]
75 Text feature [bone] present in test data point [True]
79 Text feature [concentrations] present in test data point [True]
84 Text feature [oncogene] present in test data point [True]
95 Text feature [inhibited] present in test data point [True]
101 Text feature [receptors] present in test data point [True]
102 Text feature [000] present in test data point [True]
103 Text feature [signaling] present in test data point [True]
108 Text feature [leukemia] present in test data point [True]
129 Text feature [approximately] present in test data point [True]
144 Text feature [activate] present in test data point [True]
148 Text feature [activating] present in test data point [True]
149 Text feature [transformed] present in test data point [True]
177 Text feature [mechanisms] present in test data point [True]
198 Text feature [lung] present in test data point [True]
201 Text feature [malignant] present in test data point [True]
203 Text feature [factor] present in test data point [True]
208 Text feature [presence] present in test data point [True]
215 Text feature [pathways] present in test data point [True]
217 Text feature [examined] present in test data point [True]
220 Text feature [advanced] present in test data point [True]
233 Text feature [expressing] present in test data point [True]
248 Text feature [2b] present in test data point [True]
250 Text feature [s3] present in test data point [True]
266 Text feature [versus] present in test data point [True]
271 Text feature [constitutively] present in test data point [True]
275 Text feature [wt] present in test data point [True]
278 Text feature [2003] present in test data point [True]
280 Text feature [lead] present in test data point [True]
307 Text feature [days] present in test data point [True]
313 Text feature [position] present in test data point [True]
317 Text feature [bp] present in test data point [True]
335 Text feature [occur] present in test data point [True]
370 Text feature [promote] present in test data point [True]
371 Text feature [colony] present in test data point [True]
392 Text feature [akt] present in test data point [True]
398 Text feature [cells] present in test data point [True]
422 Text feature [gfp] present in test data point [True]
423 Text feature [phospho] present in test data point [True]
442 Text feature [inhibitor] present in test data point [True]
445 Text feature [specimens] present in test data point [True]
468 Text feature [day] present in test data point [True]
474 Text feature [culture] present in test data point [True]
476 Text feature [gain] present in test data point [True]
481 Text feature [proliferation] present in test data point [True]
484 Text feature [effective] present in test data point [True]
496 Text feature [coding] present in test data point [True]
Out of the top  500  features  60 are present in query point
```

In [98]:

```
### Incorrectly Classified point
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:",
np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
get_impfeature_names(indices[0],
test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation']
.iloc[test_point_index], no_feature)
```

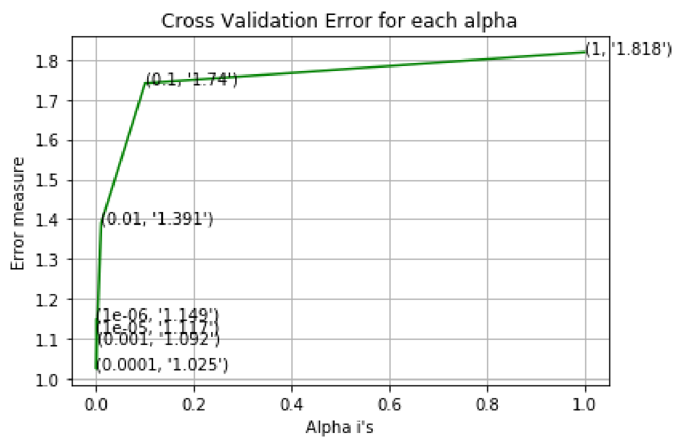
```
Predicted Class : 7
Predicted Class Probabilities: [[1.07e-02 6.45e-02 4.00e-04 5.90e-03 3.50e-03 1.40e-03 9.05e-01 8.
40e-03
  3.00e-04]]
Actual Class : 7
--------------------------------------------------
8 Text feature [activated] present in test data point [True]
11 Text feature [constitutive] present in test data point [True]
17 Text feature [downstream] present in test data point [True]
22 Text feature [insertion] present in test data point [True]
23 Text feature [overexpression] present in test data point [True]
29 Text feature [activation] present in test data point [True]
32 Text feature [enhanced] present in test data point [True]
34 Text feature [positive] present in test data point [True]
36 Text feature [2a] present in test data point [True]
43 Text feature [ligand] present in test data point [True]
54 Text feature [elevated] present in test data point [True]
65 Text feature [3b] present in test data point [True]
67 Text feature [mapk] present in test data point [True]
75 Text feature [bone] present in test data point [True]
79 Text feature [concentrations] present in test data point [True]
87 Text feature [phosphorylated] present in test data point [True]
95 Text feature [inhibited] present in test data point [True]
103 Text feature [signaling] present in test data point [True]
108 Text feature [leukemia] present in test data point [True]
129 Text feature [approximately] present in test data point [True]
144 Text feature [activate] present in test data point [True]
149 Text feature [transformed] present in test data point [True]
171 Text feature [transformation] present in test data point [True]
177 Text feature [mechanisms] present in test data point [True]
198 Text feature [lung] present in test data point [True]
203 Text feature [factor] present in test data point [True]
208 Text feature [presence] present in test data point [True]
215 Text feature [pathways] present in test data point [True]
217 Text feature [examined] present in test data point [True]
233 Text feature [expressing] present in test data point [True]
248 Text feature [2b] present in test data point [True]
257 Text feature [ras] present in test data point [True]
271 Text feature [constitutively] present in test data point [True]
280 Text feature [lead] present in test data point [True]
307 Text feature [days] present in test data point [True]
317 Text feature [bp] present in test data point [True]
335 Text feature [occur] present in test data point [True]
353 Text feature [transforming] present in test data point [True]
370 Text feature [promote] present in test data point [True]
392 Text feature [akt] present in test data point [True]
397 Text feature [72] present in test data point [True]
398 Text feature [cells] present in test data point [True]
423 Text feature [phospho] present in test data point [True]
442 Text feature [inhibitor] present in test data point [True]
445 Text feature [specimens] present in test data point [True]
467 Text feature [extracellular] present in test data point [True]
468 Text feature [day] present in test data point [True]
481 Text feature [proliferation] present in test data point [True]
484 Text feature [effective] present in test data point [True]
Out of the top  500  features  49 are present in query point
```

Without Class balancing

In [99]:

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

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


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

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

```
for alpha = 1e-06
Log Loss : 1.1491058758961354
for alpha = 1e-05
Log Loss : 1.1168028666910488
for alpha = 0.0001
Log Loss : 1.025147652417543
for alpha = 0.001
Log Loss : 1.0918739194685716
for alpha = 0.01
Log Loss : 1.3908666496584612
for alpha = 0.1
Log Loss : 1.7400918221098747
for alpha = 1
Log Loss : 1.8180840242686098
```



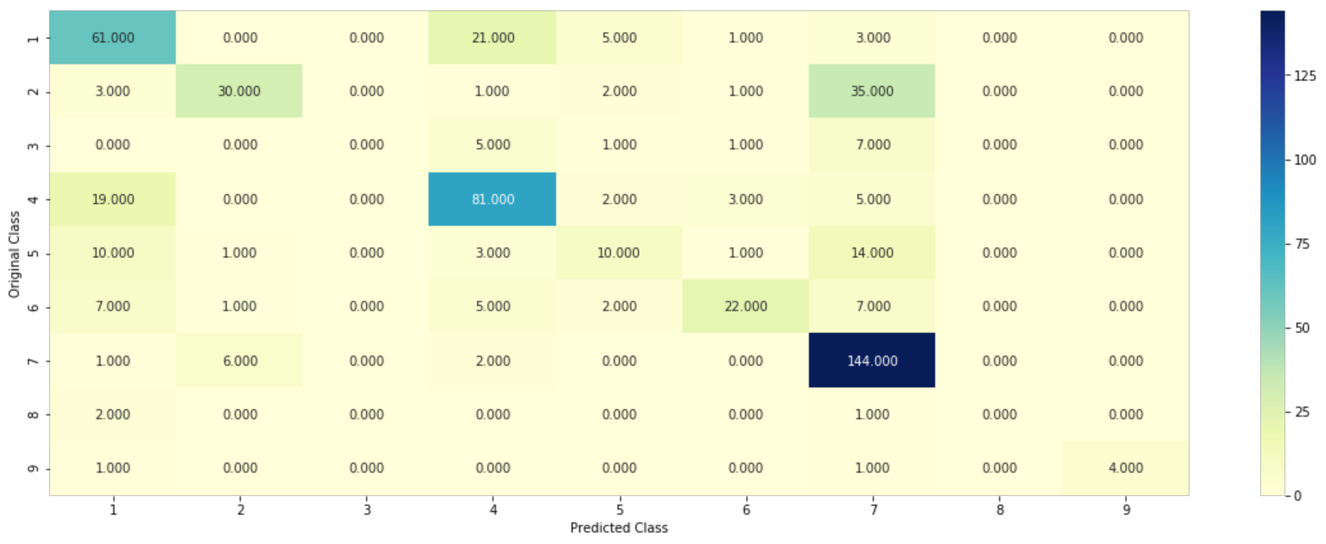
```
For values of best alpha =  0.0001 The train log loss is: 0.43317253163431446
For values of best alpha =  0.0001 The cross validation log loss is: 1.025147652417543
For values of best alpha =  0.0001 The test log loss is: 0.9874409474063465
```

In [100]:

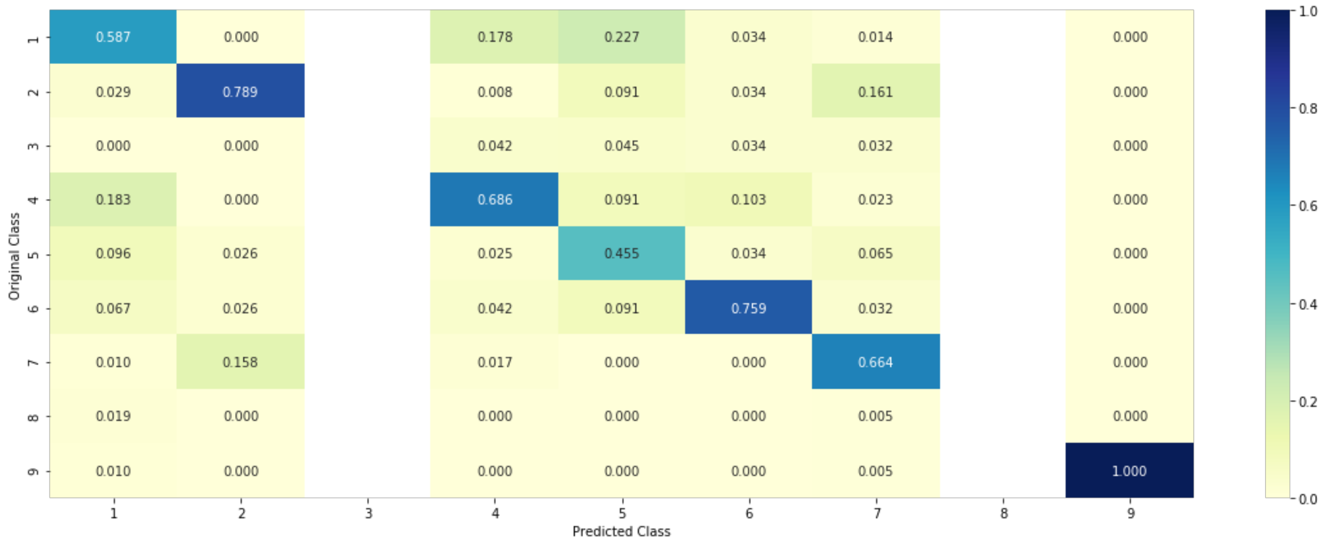
```
### Testing model with best hyper parameters
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

```
Log loss : 1.025147652417543
Number of mis-classified points : 0.3383458646616541
```

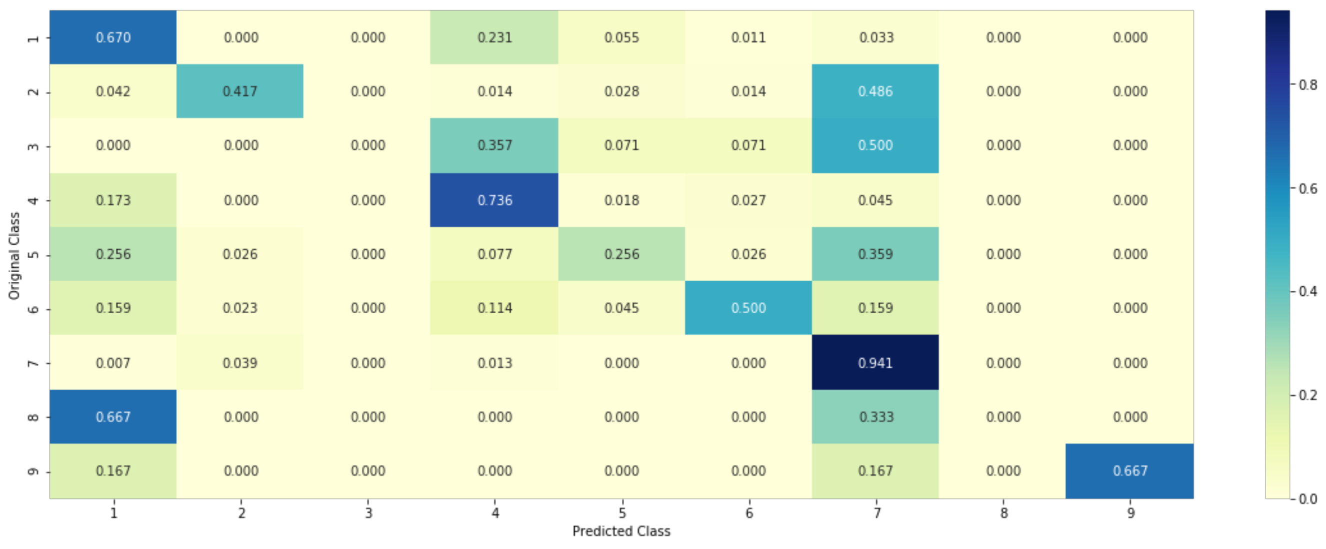
```
-------------------- Confusion matrix --------------------
```



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



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



In [101]:

```
####  Feature Importance, Correctly Classified point
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
```

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

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1351 0.0465 0.0139 0.0438 0.1103 0.0359 0.5983 0.0065 0.0097]]
Actual Class : 7
--------------------------------------------------
13 Text feature [activated] present in test data point [True]
18 Text feature [codon] present in test data point [True]
26 Text feature [constitutive] present in test data point [True]
28 Text feature [downstream] present in test data point [True]
38 Text feature [overexpression] present in test data point [True]
47 Text feature [2a] present in test data point [True]
57 Text feature [enhanced] present in test data point [True]
68 Text feature [positive] present in test data point [True]
72 Text feature [elevated] present in test data point [True]
76 Text feature [activation] present in test data point [True]
79 Text feature [ligand] present in test data point [True]
92 Text feature [concentrations] present in test data point [True]
96 Text feature [fold] present in test data point [True]
112 Text feature [3b] present in test data point [True]
115 Text feature [inhibited] present in test data point [True]
122 Text feature [bone] present in test data point [True]
123 Text feature [000] present in test data point [True]
126 Text feature [mapk] present in test data point [True]
133 Text feature [approximately] present in test data point [True]
149 Text feature [receptors] present in test data point [True]
166 Text feature [lung] present in test data point [True]
188 Text feature [mechanisms] present in test data point [True]
192 Text feature [leukemia] present in test data point [True]
193 Text feature [oncogene] present in test data point [True]
199 Text feature [activate] present in test data point [True]
213 Text feature [activating] present in test data point [True]
215 Text feature [signaling] present in test data point [True]
238 Text feature [transformed] present in test data point [True]
241 Text feature [factor] present in test data point [True]
247 Text feature [s3] present in test data point [True]
248 Text feature [examined] present in test data point [True]
278 Text feature [advanced] present in test data point [True]
282 Text feature [presence] present in test data point [True]
286 Text feature [2b] present in test data point [True]
297 Text feature [lead] present in test data point [True]
306 Text feature [wt] present in test data point [True]
307 Text feature [versus] present in test data point [True]
308 Text feature [expressing] present in test data point [True]
323 Text feature [malignant] present in test data point [True]
326 Text feature [bp] present in test data point [True]
336 Text feature [2003] present in test data point [True]
359 Text feature [occur] present in test data point [True]
360 Text feature [previously] present in test data point [True]
379 Text feature [proliferation] present in test data point [True]
382 Text feature [position] present in test data point [True]
383 Text feature [observations] present in test data point [True]
397 Text feature [promote] present in test data point [True]
400 Text feature [coding] present in test data point [True]
403 Text feature [colony] present in test data point [True]
405 Text feature [akt] present in test data point [True]
415 Text feature [constitutively] present in test data point [True]
427 Text feature [pathways] present in test data point [True]
430 Text feature [days] present in test data point [True]
436 Text feature [current] present in test data point [True]
443 Text feature [gain] present in test data point [True]
451 Text feature [epithelial] present in test data point [True]
454 Text feature [gfp] present in test data point [True]
456 Text feature [effective] present in test data point [True]
```

```
457 Text feature [inhibitor] present in test data point [True]
469 Text feature [provided] present in test data point [True]
470 Text feature [cells] present in test data point [True]
485 Text feature [regulated] present in test data point [True]
491 Text feature [properties] present in test data point [True]
496 Text feature [phospho] present in test data point [True]
Out of the top  500  features  64 are present in query point
```

In [102]:

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

```
Predicted Class : 7
Predicted Class Probabilities: [[1.020e-02 6.500e-02 3.000e-04 6.900e-03 3.100e-03 1.300e-03 9.111
e-01
  2.100e-03 0.000e+00]]
Actual Class : 7
--------------------------------------------------
13 Text feature [activated] present in test data point [True]
26 Text feature [constitutive] present in test data point [True]
28 Text feature [downstream] present in test data point [True]
32 Text feature [insertion] present in test data point [True]
38 Text feature [overexpression] present in test data point [True]
47 Text feature [2a] present in test data point [True]
57 Text feature [enhanced] present in test data point [True]
68 Text feature [positive] present in test data point [True]
72 Text feature [elevated] present in test data point [True]
76 Text feature [activation] present in test data point [True]
79 Text feature [ligand] present in test data point [True]
92 Text feature [concentrations] present in test data point [True]
112 Text feature [3b] present in test data point [True]
115 Text feature [inhibited] present in test data point [True]
122 Text feature [bone] present in test data point [True]
125 Text feature [phosphorylated] present in test data point [True]
126 Text feature [mapk] present in test data point [True]
133 Text feature [approximately] present in test data point [True]
166 Text feature [lung] present in test data point [True]
188 Text feature [mechanisms] present in test data point [True]
192 Text feature [leukemia] present in test data point [True]
199 Text feature [activate] present in test data point [True]
215 Text feature [signaling] present in test data point [True]
238 Text feature [transformed] present in test data point [True]
241 Text feature [factor] present in test data point [True]
248 Text feature [examined] present in test data point [True]
251 Text feature [transformation] present in test data point [True]
259 Text feature [ras] present in test data point [True]
282 Text feature [presence] present in test data point [True]
286 Text feature [2b] present in test data point [True]
297 Text feature [lead] present in test data point [True]
308 Text feature [expressing] present in test data point [True]
326 Text feature [bp] present in test data point [True]
359 Text feature [occur] present in test data point [True]
360 Text feature [previously] present in test data point [True]
379 Text feature [proliferation] present in test data point [True]
383 Text feature [observations] present in test data point [True]
397 Text feature [promote] present in test data point [True]
405 Text feature [akt] present in test data point [True]
415 Text feature [constitutively] present in test data point [True]
417 Text feature [carcinomas] present in test data point [True]
427 Text feature [pathways] present in test data point [True]
430 Text feature [days] present in test data point [True]
436 Text feature [current] present in test data point [True]
437 Text feature [72] present in test data point [True]
```

```
456 Text feature [effective] present in test data point [True]
457 Text feature [inhibitor] present in test data point [True]
469 Text feature [provided] present in test data point [True]
470 Text feature [cells] present in test data point [True]
485 Text feature [regulated] present in test data point [True]
491 Text feature [properties] present in test data point [True]
493 Text feature [ph] present in test data point [True]
496 Text feature [phospho] present in test data point [True]
Out of the top  500  features  53 are present in query point
```

Conclusion:

### Personalized Cancer Diagnosis Using Tfidfvectorizer 1) WE have taken training variants and training_text CSV files from kaggle site for personalized cancer diagnosis. #### EDA: 2) In Exploratory Data Analysis, visualize and plot Gene , variation and Class columns for understanding of data. 3) In EDA, for text preprocessing we remove the stopwords, lower the characters. 4) After EDA, we split the data for train,test and cross validation using train_test_split. 5)We plot the distribution of class in each train, test and cross validation, and it seems to equal in all three parts. #### CReate a Random MOdel 6)Next build a random model.In a 'Random' Model, we generate the NINE class probabilites randomly such that they sum to 1. 7) Create a function to plot confusion matrix for Confusion, recall, precision matrix. #### Univariate Analysis 8) For Univariate Analysis, create a function for Gene variation Dictionary, which contains the probability array for each gene/variation and create Gene variation feature, it will contain the feature for each feature value in the data. 9) To caculate the probability of a feature belongs to any particular class, we apply laplace smoothing. ### Univariate Analysis on Gene Feature 10) Observe Gene is a what tye of feature,how many categories are there and how they are distributed. 11) Plot a histogram and cumulative distributive of genes. 12) Featurize the Gene feature using response coding and one hot coding. 13) Observe how good is this gene feature in predicting y_i, here we build Logistic regrewssion for prediction. 14) And compute log loss for train ,cv and test. 14) Observe the Gene feature stable across all the data sets Test, Train, Cross validation. 15) Repete the steps for Variation and Text feture. ### Univariate Analysis on Text Feature 16) Observe, how many unique words are present in train data 17) How are word frequencies distributed? 18) Featurize text field, build a TFidfvectorizer with max 5000 features. 19) Is the text feature useful in predicitng y_i? 20) Is the text feature stable across train, test and CV datasets ### Machine lerning Models 21) Prepare a data for ML models define a function to predict and plot confusion matrix, use calibratedClassifierCV because we want probabilities. 22) Define a function for log loss and important features. 23) Shack the three features ##### Base line models 24) Here we have build naive bayes, knn, LOgistic regression, svm. random forest, stacking and voting classifier. 25) Do the hyperparameter tuning and test the model with best hyperparameter. 26) Note the log loss for train, test and cv.

In [4]:

```python
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Model","features","Train","CV","Test","% Missclassified"]

x.add_row(["Naive Bayes", "OneHOt","0.528","1.218","1.198","39.47%"])
x.add_row(["KNN","REsponse", "0.631","0.992","1.00","34.96%"])
x.add_row(["Logistic(with class balance)", "OneHOt","0.444","1.000","0.971","33.45%"])
x.add_row(["Logistic(without class balance)","OneHOt", "0.433","1.025","0.987","33.83%"])
x.add_row(["Linear SVM","OneHOt", "0.582","1.035","1.0438","32.89%"])
x.add_row(["Random Forest", "OneHOt","0.855","1.200","1.1937","44.36%"])
x.add_row(["Random Forest","Response", "0.053","1.284","1.335","47.18%"])
x.add_row(["Stacking Classifier", "OneHOt","0.540","1.170","1.180","38.04%"])
x.add_row(["Voting Classifier","OneHOt", "0.834","1.186","1.189","35.78%"])


print(x)
```

```
+---------------------------------+----------+-------+-------+--------+------------------+
|              Model              | features | Train |   CV  |  Test  | % Missclassified |
+---------------------------------+----------+-------+-------+--------+------------------+
|           Naive Bayes           |  OneHOt  | 0.528 | 1.218 | 1.198  |      39.47%      |
|               KNN               | REsponse | 0.631 | 0.992 |  1.00  |      34.96%      |
|   Logistic(with class balance)  |  OneHOt  | 0.444 | 1.000 | 0.971  |      33.45%      |
| Logistic(without class balance) |  OneHOt  | 0.433 | 1.025 | 0.987  |      33.83%      |
|            Linear SVM           |  OneHOt  | 0.582 | 1.035 | 1.0438 |      32.89%      |
|          Random Forest          |  OneHOt  | 0.855 | 1.200 | 1.1937 |      44.36%      |
|          Random Forest          | Response | 0.053 | 1.284 | 1.335  |      47.18%      |
|       Stacking Classifier       |  OneHOt  | 0.540 | 1.170 | 1.180  |      38.04%      |
|        Voting Classifier        |  OneHOt  | 0.834 | 1.186 | 1.189  |      35.78%      |
+---------------------------------+----------+-------+-------+--------+------------------+
```

1) For TFIDF vectorizer with top 1000 features we are getting best result with LOgistic regression with class balance with one hot coding.
2) For TFIDF vectorizer with top 1000 features we are getting worst result with Random Forest with response codingLogistic regression

with CountVectorizer Features, including both unigrams and bigrams

In [1]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Model","Train","CV","Test","% Missclassified"]

x.add_row(["Logistic unigram(classbalance)", "0.614","1.143","1.048","34.77%"])
x.add_row(["Logistic unigram(without classbalance)", "0.628","1.185","1.054","36.28%"])
x.add_row(["Logistic bigramms(class balance)", "0.444","1.000","0.971","33.45%"])
x.add_row(["Logistic bigrams(wothout class balance)", "0.433","1.025","0.987","33.83%"])


print(x)
```

```
+-----------------------------------------+-------+-------+-------+------------------+
|                  Model                  | Train |   CV  |  Test | % Missclassified |
+-----------------------------------------+-------+-------+-------+------------------+
|      Logistic unigram(classbalance)     | 0.614 | 1.143 | 1.048 |      34.77%      |
|  Logistic unigram(without classbalance) | 0.628 | 1.185 | 1.054 |      36.28%      |
|     Logistic bigramms(class balance)    | 0.444 | 1.000 | 0.971 |      33.45%      |
| Logistic bigrams(wothout class balance) | 0.433 | 1.025 | 0.987 |      33.83%      |
+-----------------------------------------+-------+-------+-------+------------------+
```

1)For Countvectorizer with bigrams class balancing features we are getting best result with LOgistic regression with one hot coding. 2) With bigrams loss decreases and number of missclassified points percentage for