Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
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

import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\dell\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[1]: True

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:
```
data = pd.read_csv(r'C:\Users\dell\Desktop\ai class\case studies\cancer\training_varia
print('Number of data points : ', data.shape[0])
print('Number of features : ', data.shape[1])
print('Features : ', data.columns.values)
```

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

Out[2]:

	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 [3]:
```
# note the seprator in this file
data_text =pd.read_csv(r"C:\Users\dell\Desktop\ai class\case studies\cancer\training_t
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[3]:

	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 [4]:
```
# 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 + " "
```

In [5]:
```
#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)
```

```
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 : 196.9051398 seconds
```

In [6]:
```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
```

Out[6]:

	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 [7]:

Out[7]:

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

In [8]:

In [9]:

Out[9]:

	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 [15]:
```
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 varaib
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true,
# split the train data into train and cross validation by maintaining same distributio
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train,
```

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

In [96]:
```
train_df.columns
train_variation=train_df['Variation'].values;test_variation=test_df['Variation'].value
train_gene=train_df['Gene'].values;test_gene=test_df['Gene'].values;cv_gene=cv_df['Ger
train_text=train_df['TEXT'].values;test_text=test_df['TEXT'].values;cv_text=cv_df['TEX

from sklearn.feature_extraction.text import CountVectorizer
encode=CountVectorizer()
train_variation=encode.fit_transform(train_variation);test_variation=encode.transform(
train_gene=encode.fit_transform(train_gene);test_gene=encode.transform(test_gene);cv_g
#train_text=encode.fit_transform(train_gene);test_gene=encode.transform(test_gene);cv_

print(train_gene.shape)
print(train_gene[1,:])
print(train_variation.shape)
print(train_variation[100,:])


train_text=encode.fit_transform(train_text);test_text=encode.transform(test_text);cv_g
print(train_text.shape)
```

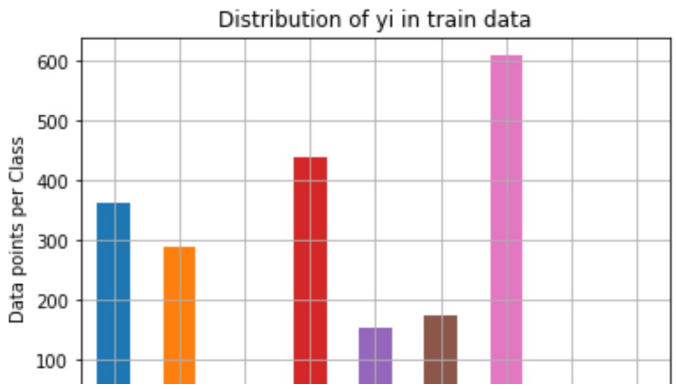
```
(2124, 232)
  (0, 228)      1
(2124, 1949)
  (0, 1682)     1
(2124, 127270)
```

In [97]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_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 [13]:
```
# it returns a dict, keys as class labels and values as the number of data points in t
train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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

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


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]

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],
```



3.2 Prediction using a 'Random' Model

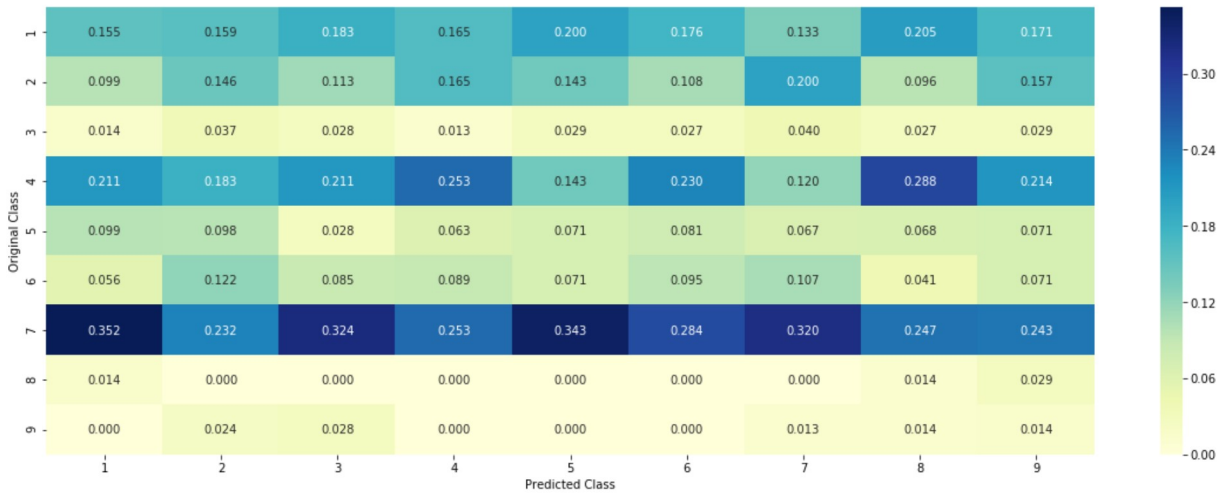
In a 'Random' Model, we generate the NINE class probabilites randomly such that they sum to 1.

In [31]:
```
# 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 predi

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

    # 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
    # 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
    # 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, yticklabe
    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, yticklabe
    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, yticklabe
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
```

In [15]:
```
# 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_predicte


# 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

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

```
Log loss on Cross Validation Data using Random Model 2.44565054571383
Log loss on Test Data using Random Model 2.4939955737943955
-------------------- Confusion matrix --------------------
```


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



3.3 Univariate Analysis

In [16]:
```
# 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 da
# build a vector (1*9) , the first element = (number of times it occured in class1 + 1
# gv_dict is like a look up table, for every gene it store a (1*9) representation of i
# 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 gen
    gv_dict = dict()

    # denominator will contain the number of time that particular feature occured in w
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to perti
        # 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
```

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?

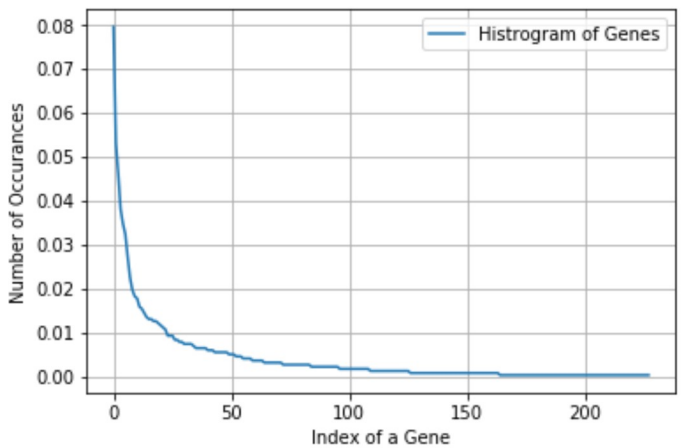
In [98]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
```

```
Number of Unique Genes : 232
BRCA1     168
TP53      106
EGFR       91
BRCA2      84
PTEN       83
KIT        61
BRAF       55
ERBB2      51
ALK        42
PIK3CA     37
Name: Gene, dtype: int64
```

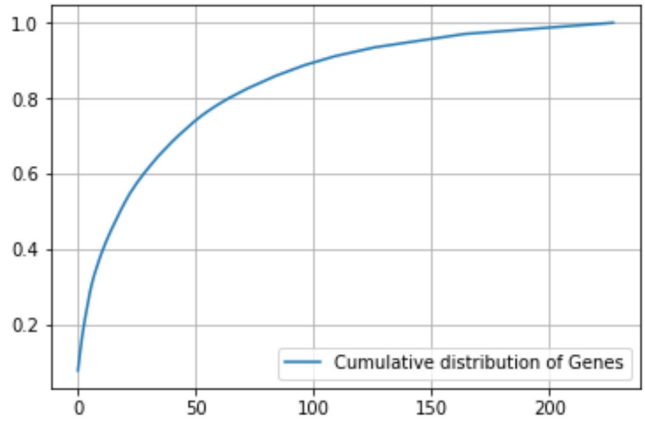
In [85]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the t
```

```
Ans: There are 224 different categories of genes in the train data, and they are d
istibuted as follows
```

In [19]:
```
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 [20]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
```



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/ (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 [21]:
```
#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
```

In [22]:
```
print("train_gene_feature_responseCoding is converted feature using respone coding met
```

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

In [23]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
```

In [24]:

```
2690      BRAF
2539     BRCA1
1304      MLH1
2256      PTEN
1998    MAP2K1
Name: Gene, dtype: object
```

In [25]:

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1b',
 'arid2',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'aurkb',
 'axin1',
 'b2m',
```

In [26]:

```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding met
```

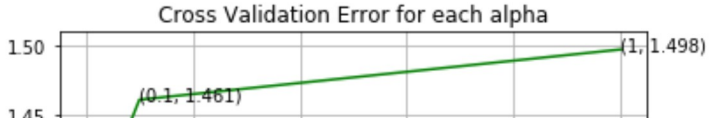
```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method
. The shape of gene feature: (2124, 227)
```

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 [27]:
```
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/
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='op
# 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
# 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, la

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_l
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
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_lo
```

```
For values of alpha =  1e-05 The log loss is: 1.396915158200963
For values of alpha =  0.0001 The log loss is: 1.2429075833353194
For values of alpha =  0.001 The log loss is: 1.266348282130333
For values of alpha =  0.01 The log loss is: 1.3729065860076568
For values of alpha =  0.1 The log loss is: 1.461130664504177
For values of alpha =  1 The log loss is: 1.4975502137999273
```



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 [28]:
```
print("Q6. How many data points in Test and CV datasets are covered by the ", unique_g

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_cover
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_cov
```

```
Q6. How many data points in Test and CV datasets are covered by the  228  genes in
train dataset?
Ans
1. In test data 647 out of 665 : 97.29323308270676
2. In cross validation data 509 out of  532 : 95.67669172932331
```

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?

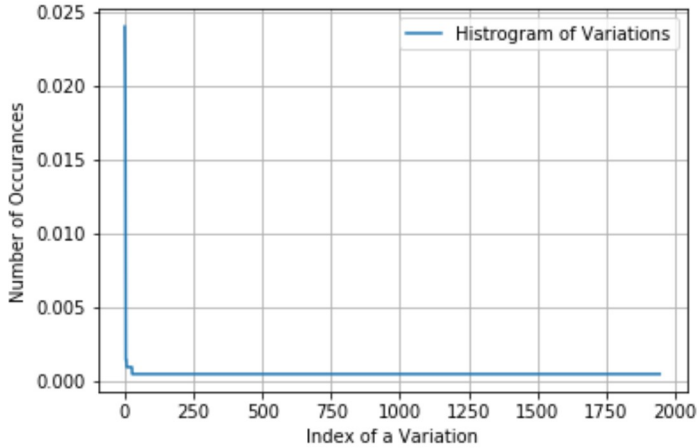
In [29]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
```

```
Number of Unique Variations : 1946
Truncating_Mutations         51
Deletion                     46
Amplification                40
Fusions                      19
Q61L                          3
Overexpression                3
G12V                          3
Q61R                          2
Promoter_Hypermethylation     2
EWSR1-ETV1_Fusion             2
Name: Variation, dtype: int64
```

In [30]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of variation
```

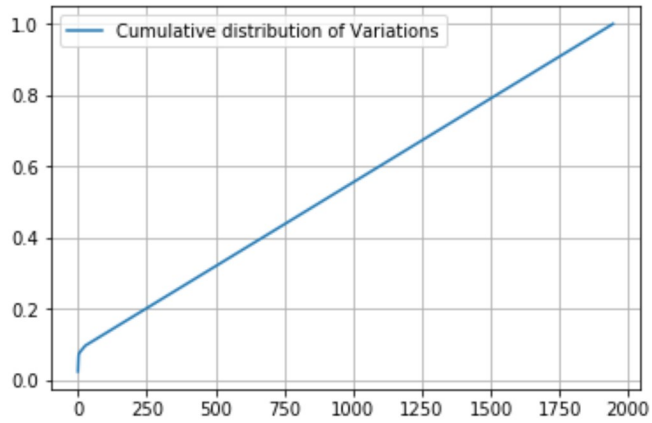
```
Ans: There are 1946 different categories of variations in the train data, and they
are distibuted as follows
```

In [31]:
```
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()
```



In [32]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
```

```
[0.0240113  0.04566855 0.06450094 ... 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/ (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 [33]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", t
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", te
# cross validation gene feature
```

In [34]:
```
print("train_variation_feature_responseCoding is a converted feature using the respons
```

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

In [35]:
```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Va
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variatio
```

In [36]:
```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot e
```

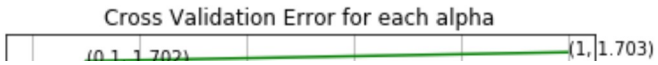
```
train_variation_feature_onehotEncoded is converted feature using the onne-hot enco
ding method. The shape of Variation feature: (2124, 1968)
```

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

Let's build a model just like the earlier!

In [37]:
```
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='op
# 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
# 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, la

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_l
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
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_lo
```

```
For values of alpha =  1e-05 The log loss is: 1.6968211058887468
For values of alpha =  0.0001 The log loss is: 1.6835290251627297
For values of alpha =  0.001 The log loss is: 1.6820651160020514
For values of alpha =  0.01 The log loss is: 1.6925368459444239
For values of alpha =  0.1 The log loss is: 1.7021217027094862
For values of alpha =  1 The log loss is: 1.7026452854266982
```



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 [38]:
```
print("Q12. How many data points are covered by total ", unique_variations.shape[0], "
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].sha
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_cover
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_cov
```

```
Q12. How many data points are covered by total  1946  genes in test and cross vali
dation data sets?
Ans
1. In test data 76 out of 665 : 11.428571428571429
2. In cross validation data 66 out of  532 : 12.406015037593985
```

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 [39]:
```
# 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
```

In [40]:
```
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(w
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEX
            row_index += 1
```

In [41]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in train
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*num
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
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))
```

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

In [42]:
```
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)
```

In [43]:
```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
test_text_feature_responseCoding  = get_text_responsecoding(test_df)
```

In [44]:
```
# 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_fe
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_featu
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_res
```

In [45]:
```
# 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
```

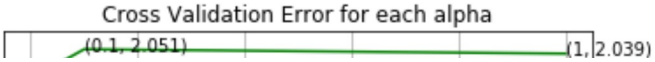
In [46]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , reverse
```

In [47]:
```
# Number of words for a given frequency.
```

```
Counter({253.26813346145497: 1, 179.47445982367256: 1, 133.19296321685073: 1, 132.
74162670149715: 1, 127.62595348326869: 1, 118.00168946661059: 1, 117.9411371020245
6: 1, 115.53615071203112: 1, 109.91764080541515: 1, 108.65809653229137: 1, 108.510
14730700803: 1, 93.7685996363791: 1, 90.28861575266002: 1, 89.10409998590228: 1, 8
3.86455917225236: 1, 82.68293278273036: 1, 81.9516378942476: 1, 78.7932454875583:
1, 77.68115528661973: 1, 77.2170101499124: 1, 76.55787326262731: 1, 75.04842976399
793: 1, 69.85598864393289: 1, 69.44835334566898: 1, 68.01449219123835: 1, 67.75326
208993692: 1, 67.15833674672778: 1, 64.49786074177975: 1, 64.04150958907309: 1, 64
.01138884897352: 1, 63.934785061898594: 1, 62.83828440055267: 1, 62.50016989989228
5: 1, 59.5773883899426: 1, 58.57118401590951: 1, 58.4137657289331: 1, 56.329687753
876385: 1, 56.00668550793676: 1, 54.370265771131734: 1, 53.46992571123765: 1, 53.0
4288234046608: 1, 50.6986581125913: 1, 49.12632874243556: 1, 48.642467319925615: 1
, 46.81796322481833: 1, 45.21616564546971: 1, 45.153596658832946: 1, 45.0126391885
4988: 1, 44.96747336965256: 1, 44.94951479974315: 1, 43.72917517156109: 1, 43.3740
1223665406: 1, 43.32294781887943: 1, 43.060730434983114: 1, 42.89277342089601: 1,
42.84943633591235: 1, 42.7036768318699: 1, 42.586648353379324: 1, 41.8565484946093
8: 1, 41.49539051752913: 1, 41.44996398260379: 1, 41.16835638883126: 1, 41.0795647
1386403: 1, 40.382320459235: 1, 40.008874267373855: 1, 39.80718022805432: 1, 39.77
398841973788: 1, 39.26887101364073: 1, 38.84897730514638: 1, 38.44540358417762: 1,
```

In [48]:
```
# Train a Logistic regression+Calibration model using text features whicha re on-hot e
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='op
# 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
# 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, la

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_l
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
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_lo
```

```
For values of alpha =  1e-05 The log loss is: 1.096326200788862
For values of alpha =  0.0001 The log loss is: 1.117321563280478
For values of alpha =  0.001 The log loss is: 1.4800977743720838
For values of alpha =  0.01 The log loss is: 1.9780161174731192
For values of alpha =  0.1 The log loss is: 2.0506972215624355
For values of alpha =  1 The log loss is: 2.038706044297784
```



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

Ans. Yes, it seems like!

In [49]:
```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(max_features=1000)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

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

In [50]:
```
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)
```

```
94.6 % of word of test data appeared in train data
93.1 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

In [29]:
```
#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
    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
```

In [52]:
```
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)
```

In [53]:
```
# 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(wo
        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 [{}]".form
        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(wo
```

Stacking the three types of features

In [54]:
```
# 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_
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_fea
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_c

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

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

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).t
cv_y = np.array(list(cv_df['Class']))


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_var
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variat
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_fe

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature_
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_res
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_responseC
```

In [55]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotC
print("(number of data points * number of features) in test data = ", test_x_onehotCod
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 3195)
(number of data points * number of features) in test data =  (665, 3195)
(number of data points * number of features) in cross validation data = (532, 3195
)
```

In [56]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_respons
print("(number of data points * number of features) in test data = ", test_x_responseC
```

```
 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

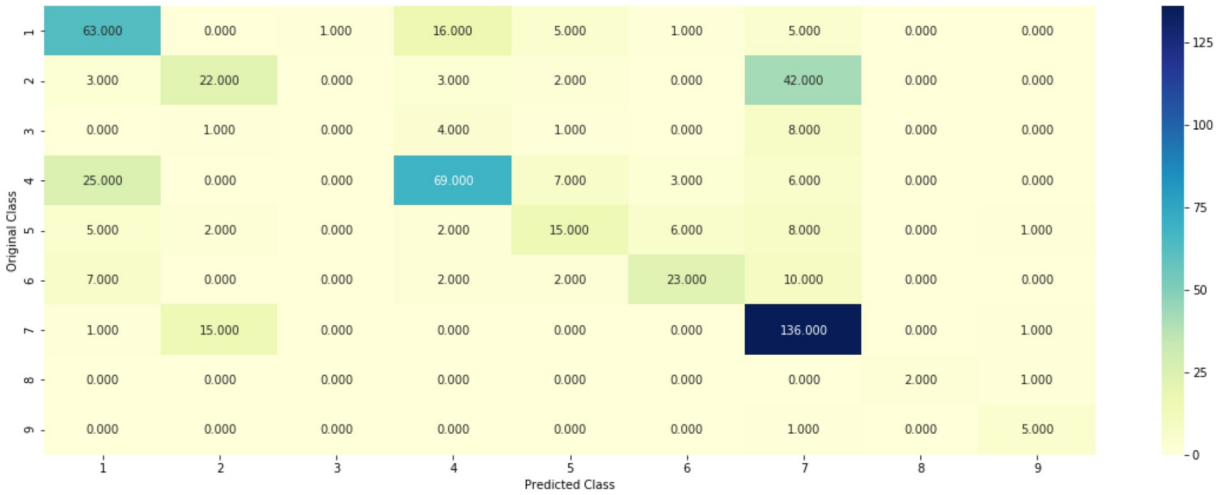
4.1.1.1. Hyper parameter tuning

In [57]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/
# -------------------------
# 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/
# -----------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/module
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv
#
# 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/
# -----------------------


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=1
    # to avoid rounding error while multiplying probabilites we use log-probability es
    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_l
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
```

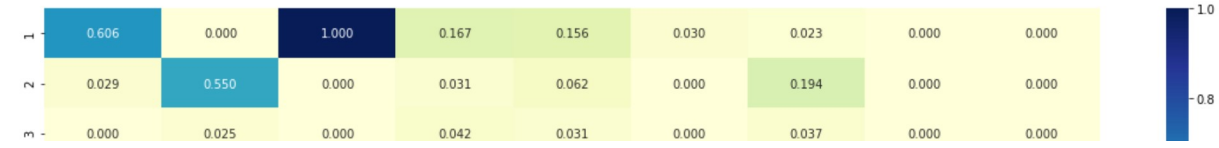
4.1.1.2. Testing the model with best hyper paramters

In [58]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/
# -------------------------
# 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/
# -----------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/module
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv
#
# 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 estima
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_oneho
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

```
Log Loss : 1.166092413987597
Number of missclassified point : 0.37030075187969924
-------------------- Confusion matrix --------------------
```




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



4.1.1.3. Feature Importance, Correctly classified point

In [62]:
```
test_point_index = 10
no_feature = 1000
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_onehotCo
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'
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0593 0.057  0.0113 0.0562 0.034  0.0321 0.7447
0.0033 0.0021]]
Actual Class : 7
--------------------------------------------------
13 Text feature [activation] present in test data point [True]
17 Text feature [kinase] present in test data point [True]
18 Text feature [inhibitor] present in test data point [True]
21 Text feature [downstream] present in test data point [True]
22 Text feature [cells] present in test data point [True]
23 Text feature [contrast] present in test data point [True]
24 Text feature [independent] present in test data point [True]
25 Text feature [expressing] present in test data point [True]
26 Text feature [growth] present in test data point [True]
27 Text feature [signaling] present in test data point [True]
28 Text feature [also] present in test data point [True]
29 Text feature [addition] present in test data point [True]
30 Text feature [treatment] present in test data point [True]
31 Text feature [activating] present in test data point [True]
35 Text feature [treated] present in test data point [True]
36 Text feature [however] present in test data point [True]
38 Text feature [mutations] present in test data point [True]
39 Text feature [10] present in test data point [True]
40 Text feature [shown] present in test data point [True]
41 Text feature [higher] present in test data point [True]
42 Text feature [sensitive] present in test data point [True]
43 Text feature [similar] present in test data point [True]
44 Text feature [inhibitors] present in test data point [True]
45 Text feature [previously] present in test data point [True]
46 Text feature [well] present in test data point [True]
47 Text feature [phosphorylation] present in test data point [True]
48 Text feature [factor] present in test data point [True]
51 Text feature [cell] present in test data point [True]
52 Text feature [presence] present in test data point [True]
53 Text feature [recently] present in test data point [True]
54 Text feature [constitutive] present in test data point [True]
55 Text feature [found] present in test data point [True]
56 Text feature [may] present in test data point [True]
58 Text feature [oncogenic] present in test data point [True]
59 Text feature [tyrosine] present in test data point [True]
60 Text feature [suggest] present in test data point [True]
62 Text feature [potential] present in test data point [True]
64 Text feature [although] present in test data point [True]
65 Text feature [showed] present in test data point [True]
67 Text feature [fig] present in test data point [True]
68 Text feature [mutation] present in test data point [True]
69 Text feature [consistent] present in test data point [True]
70 Text feature [reported] present in test data point [True]
71 Text feature [demonstrated] present in test data point [True]
72 Text feature [therapeutic] present in test data point [True]
74 Text feature [occur] present in test data point [True]
75 Text feature [total] present in test data point [True]
77 Text feature [results] present in test data point [True]
79 Text feature [figure] present in test data point [True]
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [61]:
```
test_point_index = 100
no_feature = 1000
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_onehotCo
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'
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.0718 0.0547 0.0142 0.0696 0.407  0.2598 0.1163
0.0041 0.0025]]
Actual Class : 1
--------------------------------------------------
8 Text feature [assays] present in test data point [True]
9 Text feature [functional] present in test data point [True]
10 Text feature [variants] present in test data point [True]
11 Text feature [assay] present in test data point [True]
17 Text feature [based] present in test data point [True]
18 Text feature [variant] present in test data point [True]
20 Text feature [effect] present in test data point [True]
22 Text feature [likely] present in test data point [True]
23 Text feature [results] present in test data point [True]
26 Text feature [tested] present in test data point [True]
27 Text feature [introduction] present in test data point [True]
29 Text feature [used] present in test data point [True]
36 Text feature [methods] present in test data point [True]
37 Text feature [discussion] present in test data point [True]
39 Text feature [information] present in test data point [True]
40 Text feature [available] present in test data point [True]
42 Text feature [remaining] present in test data point [True]
44 Text feature [vitro] present in test data point [True]
45 Text feature [function] present in test data point [True]
48 Text feature [published] present in test data point [True]
53 Text feature [table] present in test data point [True]
54 Text feature [genetic] present in test data point [True]
56 Text feature [addition] present in test data point [True]
59 Text feature [sensitivity] present in test data point [True]
61 Text feature [several] present in test data point [True]
63 Text feature [activities] present in test data point [True]
65 Text feature [provide] present in test data point [True]
68 Text feature [cancer] present in test data point [True]
72 Text feature [known] present in test data point [True]
74 Text feature [strong] present in test data point [True]
76 Text feature [sequence] present in test data point [True]
77 Text feature [type] present in test data point [True]
78 Text feature [also] present in test data point [True]
79 Text feature [activity] present in test data point [True]
80 Text feature [although] present in test data point [True]
81 Text feature [previously] present in test data point [True]
82 Text feature [possible] present in test data point [True]
85 Text feature [wild] present in test data point [True]
87 Text feature [include] present in test data point [True]
88 Text feature [containing] present in test data point [True]
89 Text feature [analysis] present in test data point [True]
90 Text feature [protein] present in test data point [True]
92 Text feature [therefore] present in test data point [True]
94 Text feature [three] present in test data point [True]
95 Text feature [relatively] present in test data point [True]
97 Text feature [fact] present in test data point [True]
98 Text feature [however] present in test data point [True]
99 Text feature [org] present in test data point [True]
102 Text feature [one] present in test data point [True]
```

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

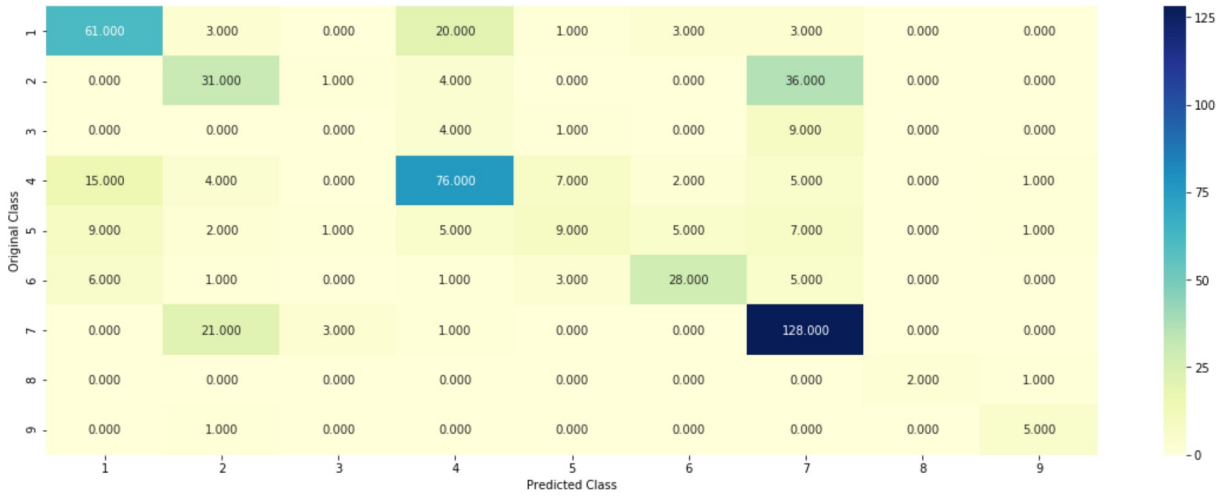
In [63]:

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/g
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=3
# 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/
#-------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/module
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv
#
# 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=1
    # to avoid rounding error while multiplying probabilites we use log-probability es
    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_l
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
```

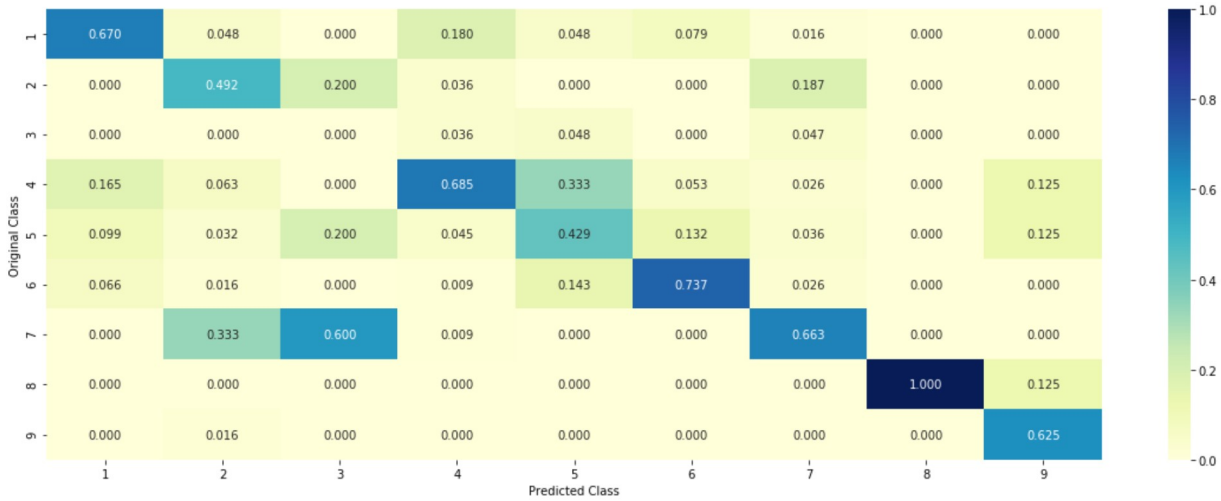
4.2.2. Testing the model with best hyper paramters

In [64]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/g
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=3
# 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/
#-------------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding
```

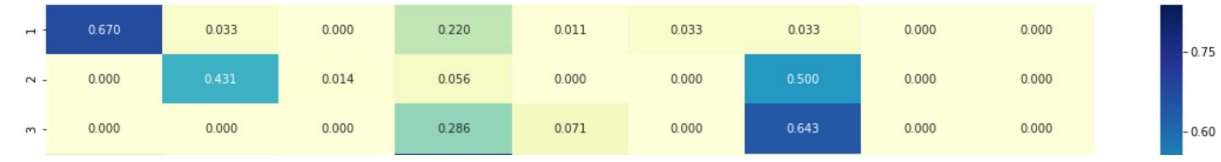
```
Log loss : 1.0560783490858177
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```




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




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



4.2.3.Sample Query point -1

In [65]:
```
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), alp
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to clas
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 3
Actual Class : 7
The  15  nearest neighbours of the test points belongs to classes [2 7 7 2 7 7 2 2
7 7 2 2 2 7 7]
Fequency of nearest points : Counter({7: 8, 2: 7})
```

4.2.4. Sample Query Point-2

In [66]:
```
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), alp
print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of the te
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 1
the k value for knn is 15 and the nearest neighbours of the test points belongs to
classes [5 7 7 6 1 7 7 2 7 7 7 2 2 7 7]
Fequency of nearest points : Counter({7: 9, 2: 3, 5: 1, 6: 1, 1: 1})
```

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

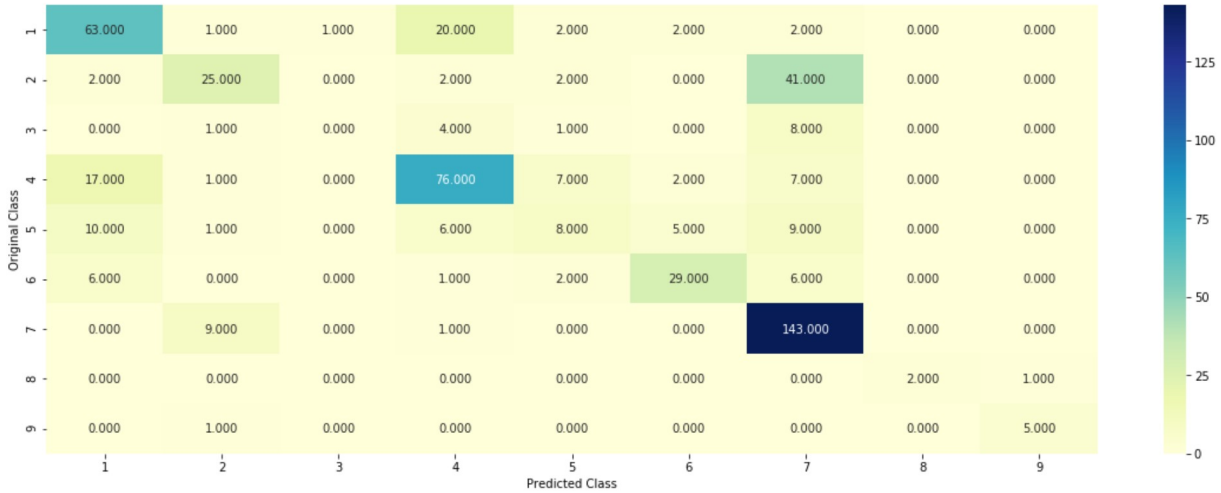
In [67]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='op
# 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
# predict(X)    Predict class labels for samples in X.

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/module
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv
#
# 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', ra
    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=1
    # to avoid rounding error while multiplying probabilites we use log-probability es
    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', lo
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_l
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
```

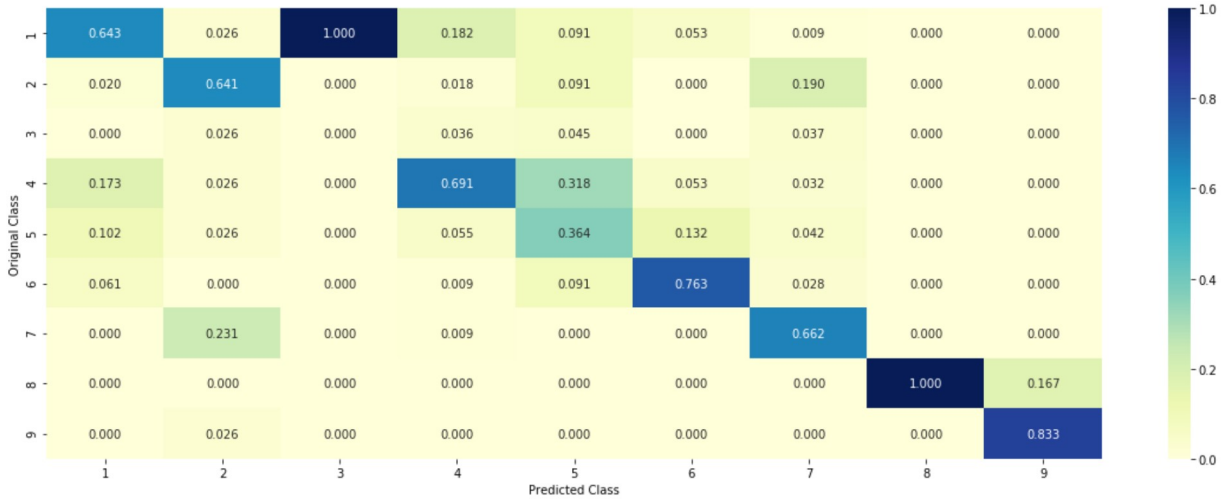
4.3.1.2. Testing the model with best hyper paramters

In [68]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='op
# 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
# predict(X)    Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/
#-------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', lo
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv
```

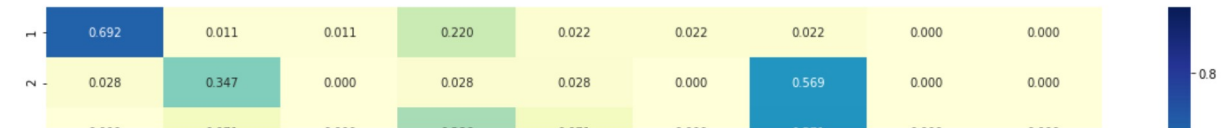
```
Log loss : 1.0182234577351874
Number of mis-classified points : 0.34022556390977443
-------------------- Confusion matrix --------------------
```




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




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



4.3.1.3. Feature Importance

In [69]:
```
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:")
```

4.3.1.3.1. Correctly Classified point

In [72]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', lo
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 10
no_feature = 1000
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_onehotCo
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'
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0098 0.0373 0.0118 0.0079 0.0486 0.0103 0.8684
0.0039 0.002 ]]
Actual Class : 7
--------------------------------------------------
17 Text feature [downstream] present in test data point [True]
18 Text feature [activation] present in test data point [True]
21 Text feature [constitutive] present in test data point [True]
24 Text feature [activating] present in test data point [True]
28 Text feature [insertion] present in test data point [True]
33 Text feature [inhibited] present in test data point [True]
36 Text feature [occur] present in test data point [True]
47 Text feature [leukemia] present in test data point [True]
57 Text feature [2a] present in test data point [True]
74 Text feature [inhibitor] present in test data point [True]
77 Text feature [versus] present in test data point [True]
103 Text feature [ligand] present in test data point [True]
108 Text feature [demonstrated] present in test data point [True]
117 Text feature [codon] present in test data point [True]
121 Text feature [2b] present in test data point [True]
127 Text feature [genomic] present in test data point [True]
128 Text feature [fusion] present in test data point [True]
166 Text feature [advanced] present in test data point [True]
169 Text feature [oncogenic] present in test data point [True]
170 Text feature [signaling] present in test data point [True]
177 Text feature [derived] present in test data point [True]
186 Text feature [treated] present in test data point [True]
207 Text feature [effective] present in test data point [True]
208 Text feature [added] present in test data point [True]
218 Text feature [volume] present in test data point [True]
232 Text feature [higher] present in test data point [True]
251 Text feature [f3] present in test data point [True]
252 Text feature [factor] present in test data point [True]
257 Text feature [ba] present in test data point [True]
300 Text feature [extracellular] present in test data point [True]
309 Text feature [malignant] present in test data point [True]
320 Text feature [amplification] present in test data point [True]
331 Text feature [frequent] present in test data point [True]
342 Text feature [gists] present in test data point [True]
347 Text feature [harboring] present in test data point [True]
354 Text feature [serum] present in test data point [True]
378 Text feature [resulting] present in test data point [True]
384 Text feature [include] present in test data point [True]
392 Text feature [contrast] present in test data point [True]
402 Text feature [pathways] present in test data point [True]
405 Text feature [phosphorylation] present in test data point [True]
409 Text feature [stat3] present in test data point [True]
426 Text feature [24] present in test data point [True]
436 Text feature [various] present in test data point [True]
442 Text feature [hours] present in test data point [True]
443 Text feature [addition] present in test data point [True]
```

4.3.1.3.2. Incorrectly Classified point

In [71]:
```
test_point_index = 100
no_feature = 1000
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_onehotCo
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'
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.0153 0.0986 0.0062 0.106  0.2936 0.2253 0.247
0.0054 0.0027]]
Actual Class : 1
--------------------------------------------------
135 Text feature [efficacy] present in test data point [True]
161 Text feature [studied] present in test data point [True]
164 Text feature [assay] present in test data point [True]
175 Text feature [alterations] present in test data point [True]
178 Text feature [assays] present in test data point [True]
186 Text feature [numbers] present in test data point [True]
189 Text feature [vitro] present in test data point [True]
190 Text feature [rare] present in test data point [True]
192 Text feature [members] present in test data point [True]
195 Text feature [discovery] present in test data point [True]
201 Text feature [functional] present in test data point [True]
203 Text feature [addition] present in test data point [True]
204 Text feature [terminal] present in test data point [True]
206 Text feature [effect] present in test data point [True]
```

4.3.2. Without Class balancing

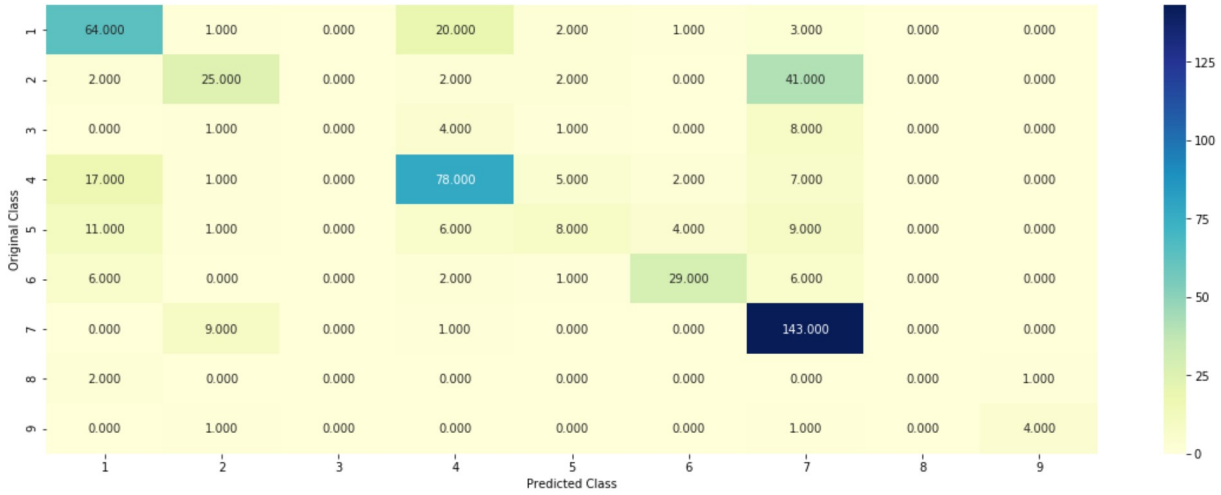
4.3.2.1. Hyper paramter tuning

In [73]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='op
# 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
# predict(X)    Predict class labels for samples in X.

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



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/module
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv
#
# 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=1
    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_l
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
```

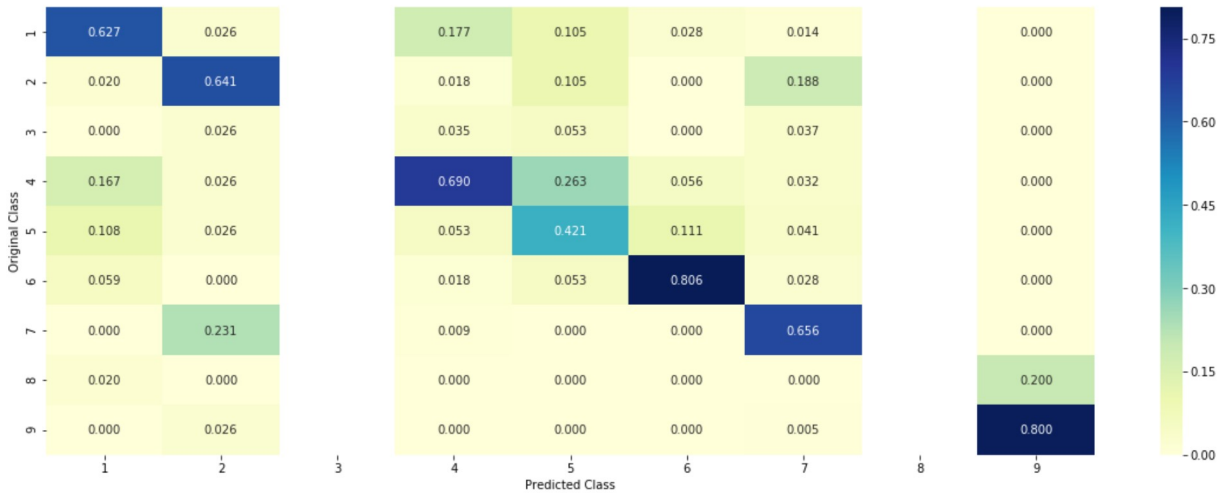
4.3.2.2. Testing model with best hyper parameters

In [74]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='op
# 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
# 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
```

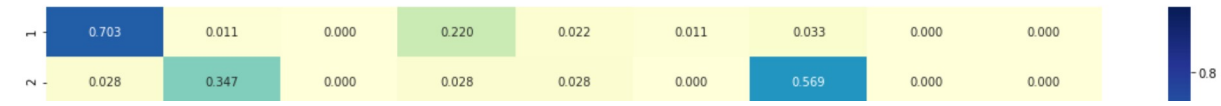
```
Log loss : 1.0514230560372393
Number of mis-classified points : 0.34022556390977443
-------------------- Confusion matrix --------------------
```




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




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



4.3.2.3. Feature Importance, Correctly Classified point

In [75]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 10
no_feature = 1000
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_onehotCo
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'
```

```
Predicted Class : 7
Predicted Class Probabilities: [[9.300e-03 3.770e-02 1.580e-02 8.200e-03 4.660e-02
9.500e-03 8.691e-01
  3.100e-03 8.000e-04]]
Actual Class : 7
--------------------------------------------------
24 Text feature [activation] present in test data point [True]
25 Text feature [constitutive] present in test data point [True]
29 Text feature [activating] present in test data point [True]
30 Text feature [downstream] present in test data point [True]
36 Text feature [inhibited] present in test data point [True]
44 Text feature [insertion] present in test data point [True]
54 Text feature [occur] present in test data point [True]
71 Text feature [leukemia] present in test data point [True]
76 Text feature [2a] present in test data point [True]
107 Text feature [derived] present in test data point [True]
125 Text feature [genomic] present in test data point [True]
129 Text feature [inhibitor] present in test data point [True]
130 Text feature [versus] present in test data point [True]
164 Text feature [demonstrated] present in test data point [True]
166 Text feature [advanced] present in test data point [True]
170 Text feature [codon] present in test data point [True]
187 Text feature [oncogenic] present in test data point [True]
190 Text feature [fusion] present in test data point [True]
193 Text feature [2b] present in test data point [True]
203 Text feature [ligand] present in test data point [True]
223 Text feature [added] present in test data point [True]
230 Text feature [treated] present in test data point [True]
233 Text feature [factor] present in test data point [True]
242 Text feature [volume] present in test data point [True]
247 Text feature [effective] present in test data point [True]
269 Text feature [higher] present in test data point [True]
279 Text feature [harboring] present in test data point [True]
299 Text feature [signaling] present in test data point [True]
307 Text feature [f3] present in test data point [True]
310 Text feature [ba] present in test data point [True]
328 Text feature [frequent] present in test data point [True]
334 Text feature [include] present in test data point [True]
364 Text feature [phosphorylation] present in test data point [True]
380 Text feature [amplification] present in test data point [True]
384 Text feature [24] present in test data point [True]
390 Text feature [gists] present in test data point [True]
421 Text feature [contrast] present in test data point [True]
422 Text feature [extracellular] present in test data point [True]
423 Text feature [resistant] present in test data point [True]
443 Text feature [concentrations] present in test data point [True]
450 Text feature [stat3] present in test data point [True]
459 Text feature [transfection] present in test data point [True]
461 Text feature [resulting] present in test data point [True]
472 Text feature [addition] present in test data point [True]
481 Text feature [signals] present in test data point [True]
482 Text feature [stably] present in test data point [True]
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [76]:
```
test_point_index = 100
no_feature = 1000
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_onehotCo
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'
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.0197 0.1098 0.0068 0.1212 0.2911 0.2009 0.2449
0.0048 0.0008]]
Actual Class : 1
--------------------------------------------------
136 Text feature [efficacy] present in test data point [True]
167 Text feature [studied] present in test data point [True]
168 Text feature [assay] present in test data point [True]
178 Text feature [assays] present in test data point [True]
179 Text feature [alterations] present in test data point [True]
186 Text feature [rare] present in test data point [True]
188 Text feature [numbers] present in test data point [True]
191 Text feature [vitro] present in test data point [True]
196 Text feature [members] present in test data point [True]
197 Text feature [discovery] present in test data point [True]
201 Text feature [variants] present in test data point [True]
202 Text feature [functional] present in test data point [True]
207 Text feature [construct] present in test data point [True]
208 Text feature [effect] present in test data point [True]
```

4.4. Linear Support Vector Machines

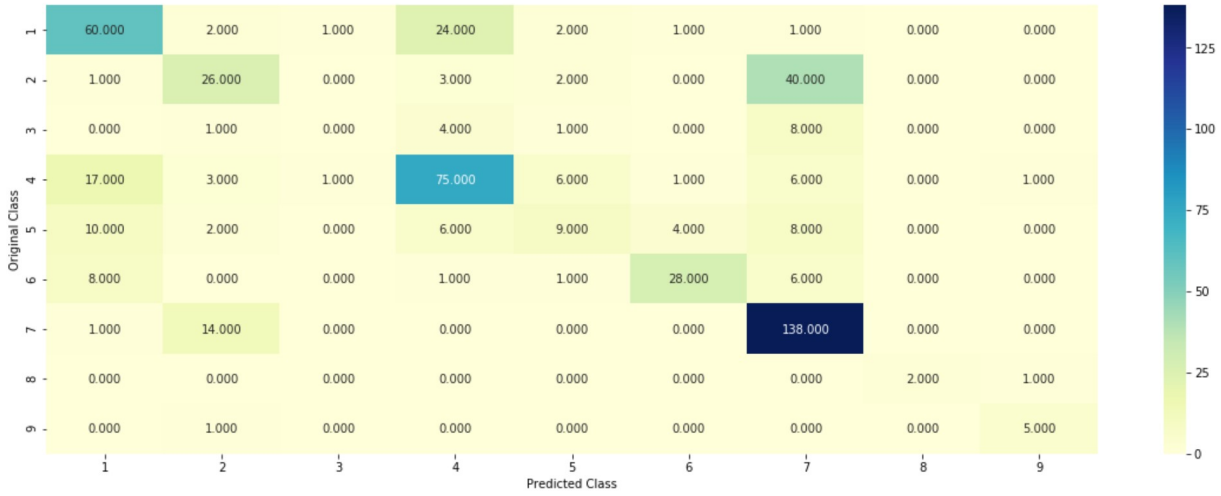
4.4.1. Hyper paramter tuning

In [77]:
```
# read more about support vector machines with linear kernals here http://scikit-learn

# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probabil
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_sha

# 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/
# --------------------------------



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/module
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv
#
# 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',
    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=1
    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', lo
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_l
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
```

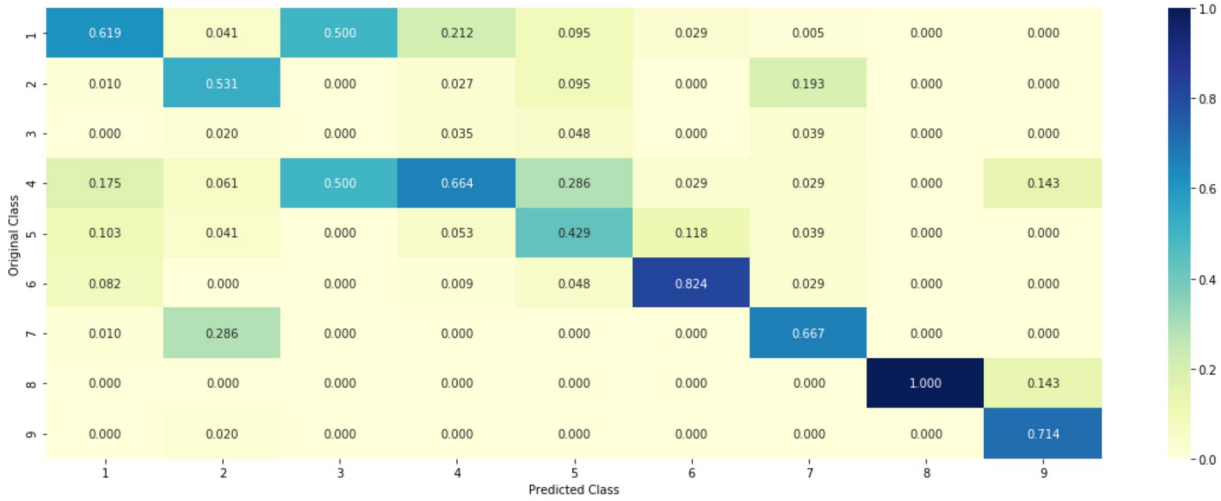
4.4.2. Testing model with best hyper parameters

In [78]:
```
# read more about support vector machines with linear kernals here http://scikit-learn
# ------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probabil
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_sha
# 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/
# ------------------------------
# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='balanc
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y
```

```
Log loss : 1.0323447347430585
Number of mis-classified points : 0.35526315789473684
-------------------- Confusion matrix --------------------
```




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




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

4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

In [80]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 10
# test_point_index = 100
no_feature = 1000
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_onehotCo
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'
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.027  0.0309 0.014  0.0162 0.0423 0.0486 0.8146
0.0035 0.0029]]
Actual Class : 7
--------------------------------------------------
35 Text feature [insertion] present in test data point [True]
36 Text feature [downstream] present in test data point [True]
38 Text feature [activation] present in test data point [True]
40 Text feature [constitutive] present in test data point [True]
45 Text feature [versus] present in test data point [True]
49 Text feature [occur] present in test data point [True]
50 Text feature [demonstrated] present in test data point [True]
52 Text feature [volume] present in test data point [True]
221 Text feature [derived] present in test data point [True]
223 Text feature [inhibited] present in test data point [True]
226 Text feature [genomic] present in test data point [True]
229 Text feature [added] present in test data point [True]
231 Text feature [2a] present in test data point [True]
236 Text feature [activating] present in test data point [True]
```

4.3.3.2. For Incorrectly classified point

In [81]:
```
test_point_index = 100
no_feature = 1000
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_onehotCo
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'
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0431 0.1338 0.01   0.2626 0.2467 0.2206 0.0753
0.0049 0.003 ]]
Actual Class : 1
--------------------------------------------------
57 Text feature [suppressor] present in test data point [True]
200 Text feature [indicate] present in test data point [True]
201 Text feature [show] present in test data point [True]
202 Text feature [2010] present in test data point [True]
205 Text feature [transfection] present in test data point [True]
206 Text feature [germline] present in test data point [True]
216 Text feature [1998] present in test data point [True]
218 Text feature [transfected] present in test data point [True]
221 Text feature [protein] present in test data point [True]
225 Text feature [skin] present in test data point [True]
227 Text feature [anti] present in test data point [True]
228 Text feature [considered] present in test data point [True]
232 Text feature [inactivation] present in test data point [True]
233 Text feature [antibodies] present in test data point [True]
235 Text feature [lower] present in test data point [True]
237 Text feature [induced] present in test data point [True]
239 Text feature [recent] present in test data point [True]
244 Text feature [western] present in test data point [True]
248 Text feature [linked] present in test data point [True]
250 Text feature [kinases] present in test data point [True]
252 Text feature [functional] present in test data point [True]
450 Text feature [open] present in test data point [True]
453 Text feature [high] present in test data point [True]
454 Text feature [consequences] present in test data point [True]
459 Text feature [characterized] present in test data point [True]
460 Text feature [leukemia] present in test data point [True]
461 Text feature [loss] present in test data point [True]
462 Text feature [test] present in test data point [True]
463 Text feature [catalytic] present in test data point [True]
464 Text feature [flag] present in test data point [True]
466 Text feature [occur] present in test data point [True]
470 Text feature [half] present in test data point [True]
477 Text feature [observed] present in test data point [True]
478 Text feature [larger] present in test data point [True]
485 Text feature [information] present in test data point [True]
489 Text feature [times] present in test data point [True]
490 Text feature [cases] present in test data point [True]
494 Text feature [resulting] present in test data point [True]
496 Text feature [alone] present in test data point [True]
499 Text feature [spectrum] present in test data point [True]
502 Text feature [likely] present in test data point [True]
503 Text feature [min] present in test data point [True]
505 Text feature [genetic] present in test data point [True]
506 Text feature [remaining] present in test data point [True]
507 Text feature [regions] present in test data point [True]
510 Text feature [suggesting] present in test data point [True]
511 Text feature [low] present in test data point [True]
512 Text feature [antibody] present in test data point [True]
521 Text feature [cause] present in test data point [True]
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

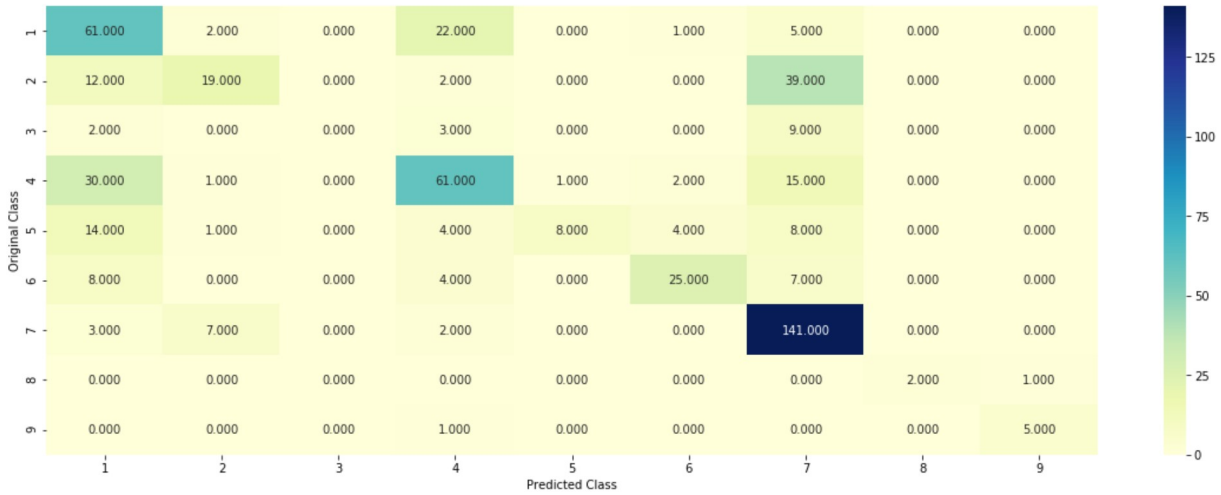
In [82]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_node
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=Non
# 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/
# --------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/module
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv
#
# 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, ra
        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_, e
        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_er
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',
clf.fit(train_x_onehotCoding, train_y)
```

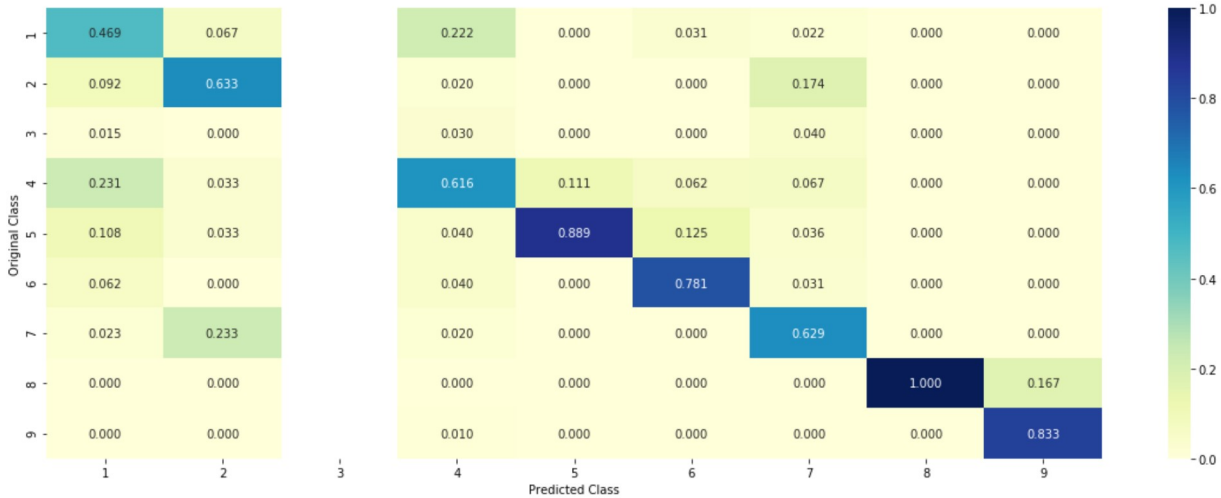
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

In [83]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_node
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=Non
# 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/
# --------------------------------

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini',
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y
```

```
Log loss : 1.185549925244811
Number of mis-classified points : 0.39473684210526316
-------------------- Confusion matrix --------------------
```




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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [84]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini',
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

test_point_index = 10
no_feature = 1000
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_onehotCo
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
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0189 0.2381 0.0151 0.0126 0.0354 0.0238 0.6518
0.0033 0.0009]]
Actual Class : 7
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [tyrosine] present in test data point [True]
4 Text feature [activation] present in test data point [True]
5 Text feature [inhibitors] present in test data point [True]
7 Text feature [phosphorylation] present in test data point [True]
8 Text feature [inhibitor] present in test data point [True]
10 Text feature [treatment] present in test data point [True]
13 Text feature [oncogenic] present in test data point [True]
14 Text feature [protein] present in test data point [True]
17 Text feature [constitutive] present in test data point [True]
21 Text feature [receptor] present in test data point [True]
23 Text feature [kinases] present in test data point [True]
25 Text feature [signaling] present in test data point [True]
27 Text feature [cells] present in test data point [True]
29 Text feature [therapy] present in test data point [True]
32 Text feature [growth] present in test data point [True]
33 Text feature [extracellular] present in test data point [True]
34 Text feature [trials] present in test data point [True]
37 Text feature [resistance] present in test data point [True]
40 Text feature [expression] present in test data point [True]
41 Text feature [therapeutic] present in test data point [True]
49 Text feature [ba] present in test data point [True]
52 Text feature [treated] present in test data point [True]
53 Text feature [inhibited] present in test data point [True]
54 Text feature [drug] present in test data point [True]
60 Text feature [proteins] present in test data point [True]
62 Text feature [cell] present in test data point [True]
63 Text feature [ic50] present in test data point [True]
64 Text feature [response] present in test data point [True]
66 Text feature [sensitivity] present in test data point [True]
67 Text feature [patients] present in test data point [True]
70 Text feature [advanced] present in test data point [True]
73 Text feature [downstream] present in test data point [True]
75 Text feature [clinical] present in test data point [True]
77 Text feature [predicted] present in test data point [True]
81 Text feature [f3] present in test data point [True]
85 Text feature [activity] present in test data point [True]
88 Text feature [imatinib] present in test data point [True]
90 Text feature [expected] present in test data point [True]
96 Text feature [proliferation] present in test data point [True]
97 Text feature [potential] present in test data point [True]
98 Text feature [sequence] present in test data point [True]
```

4.5.3.2. Inorrectly Classified point

In [85]:
```
test_point_index = 100
no_feature = 1000
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_onehotCo
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
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0951 0.2158 0.0248 0.0897 0.1154 0.0597 0.3821
0.0101 0.0073]]
Actuall Class : 1
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
3 Text feature [suppressor] present in test data point [True]
4 Text feature [activation] present in test data point [True]
5 Text feature [inhibitors] present in test data point [True]
6 Text feature [function] present in test data point [True]
7 Text feature [phosphorylation] present in test data point [True]
8 Text feature [inhibitor] present in test data point [True]
10 Text feature [treatment] present in test data point [True]
11 Text feature [activated] present in test data point [True]
12 Text feature [erk] present in test data point [True]
14 Text feature [protein] present in test data point [True]
15 Text feature [loss] present in test data point [True]
17 Text feature [constitutive] present in test data point [True]
22 Text feature [classified] present in test data point [True]
23 Text feature [kinases] present in test data point [True]
24 Text feature [variants] present in test data point [True]
25 Text feature [signaling] present in test data point [True]
27 Text feature [cells] present in test data point [True]
28 Text feature [mek] present in test data point [True]
30 Text feature [functional] present in test data point [True]
33 Text feature [extracellular] present in test data point [True]
34 Text feature [trials] present in test data point [True]
37 Text feature [resistance] present in test data point [True]
40 Text feature [expression] present in test data point [True]
41 Text feature [therapeutic] present in test data point [True]
42 Text feature [constitutively] present in test data point [True]
48 Text feature [activate] present in test data point [True]
52 Text feature [treated] present in test data point [True]
54 Text feature [drug] present in test data point [True]
56 Text feature [inhibition] present in test data point [True]
58 Text feature [inactivation] present in test data point [True]
60 Text feature [proteins] present in test data point [True]
62 Text feature [cell] present in test data point [True]
63 Text feature [ic50] present in test data point [True]
64 Text feature [response] present in test data point [True]
66 Text feature [sensitivity] present in test data point [True]
67 Text feature [patients] present in test data point [True]
71 Text feature [efficacy] present in test data point [True]
73 Text feature [downstream] present in test data point [True]
74 Text feature [phospho] present in test data point [True]
75 Text feature [clinical] present in test data point [True]
76 Text feature [mapk] present in test data point [True]
79 Text feature [variant] present in test data point [True]
85 Text feature [activity] present in test data point [True]
90 Text feature [expected] present in test data point [True]
93 Text feature [assays] present in test data point [True]
95 Text feature [raf] present in test data point [True]
96 Text feature [proliferation] present in test data point [True]
```

4.5.3. Hyper paramter tuning (With Response Coding)

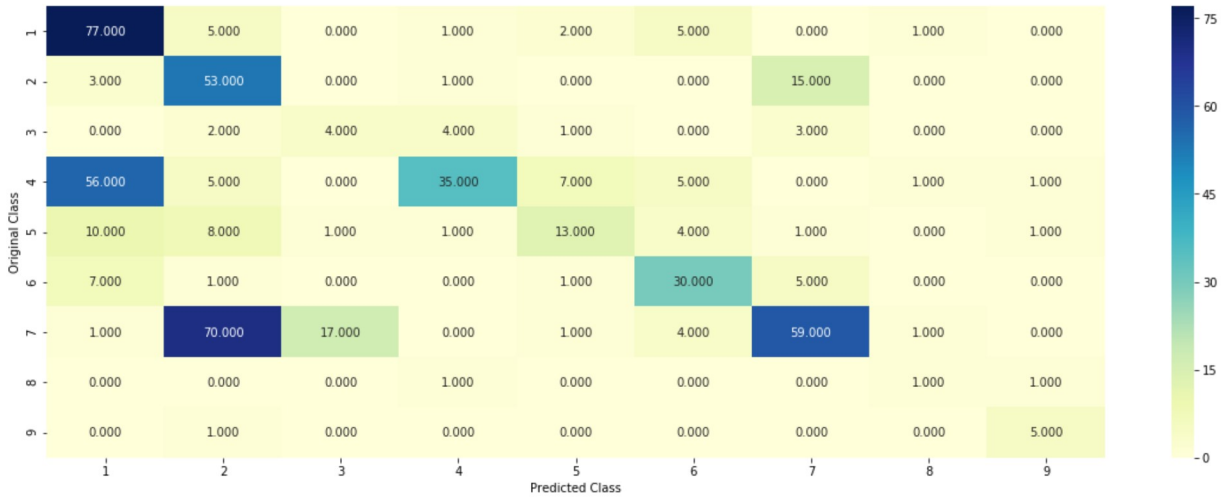
In [86]:

```
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_node
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=Non
# 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/
# -------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/module
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv
#
# 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, ra
        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_, e
        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_er
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',
clf.fit(train_x_responseCoding, train_y)
```

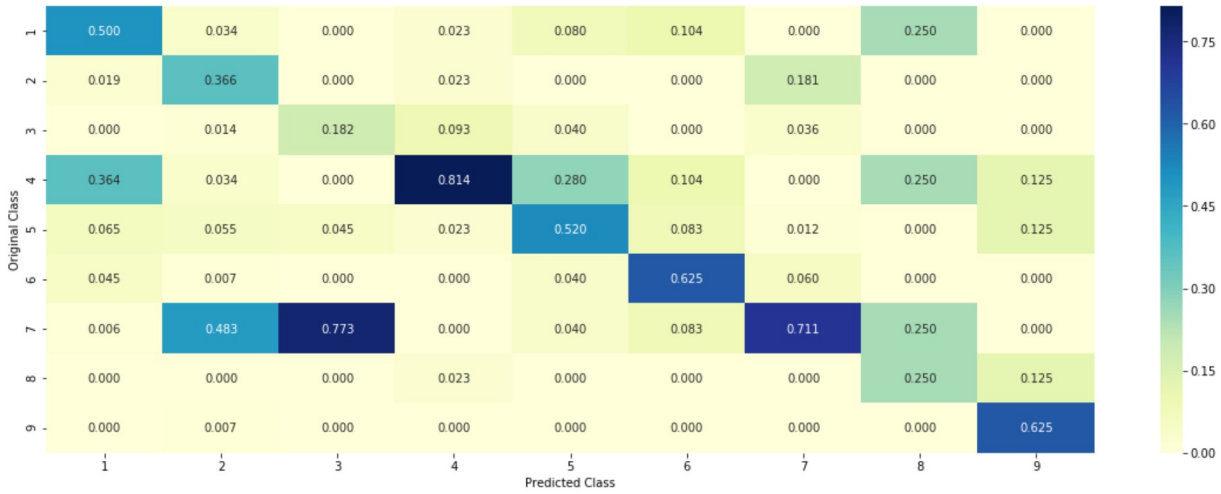
4.5.4. Testing model with best hyper parameters (Response Coding)

In [87]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_node
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=Non
# 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/
# --------------------------------

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimators=alph
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCoding,
```

```
Log loss : 1.2689163796676426
Number of mis-classified points : 0.4793233082706767
-------------------- Confusion matrix --------------------
```




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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [89]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini',
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)


test_point_index = 10
no_feature = 1000
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_response
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: [[0.0154 0.321  0.1189 0.0247 0.0298 0.0529 0.3832
0.0359 0.0182]]
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
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Gene 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 [90]:
```
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_response
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: [[0.0676 0.1827 0.0785 0.0575 0.1119 0.1633 0.0966
0.1743 0.0676]]
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Gene 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

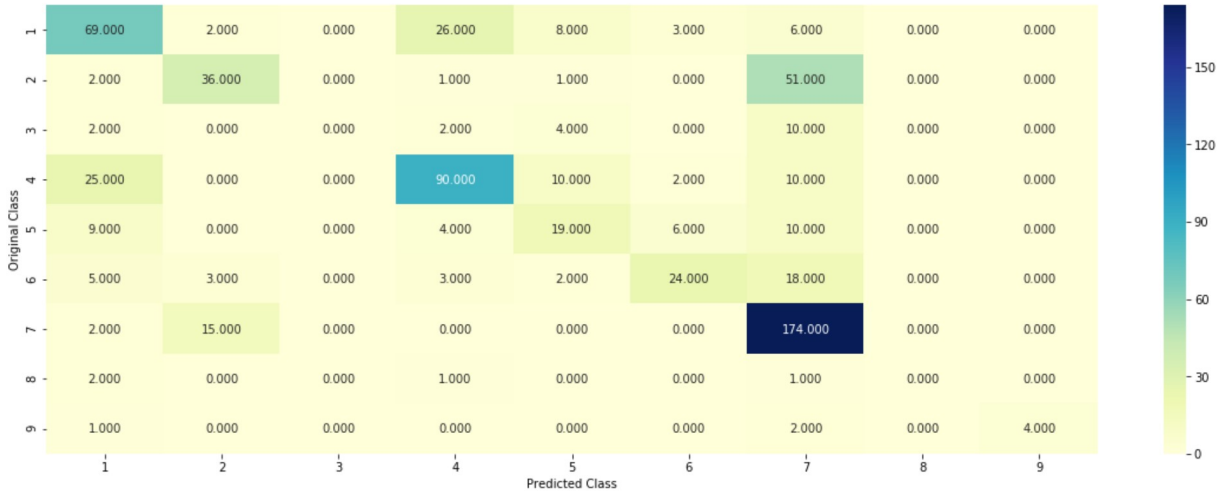
In [91]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='op
# 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
# predict(X)    Predict class labels for samples in X.

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


# read more about support vector machines with linear kernals here http://scikit-learn
# ---------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probabil
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_sha

# 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/
# ---------------------------------


# read more about support vector machines with linear kernals here http://scikit-learn
# ---------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_node
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=Non
# 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/
# ---------------------------------


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

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', ran
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")
```

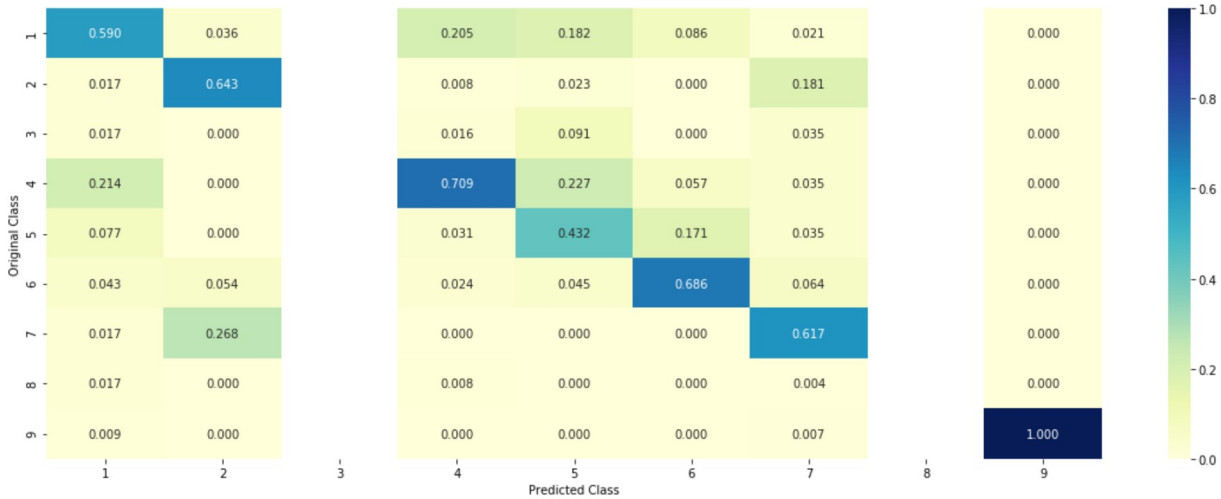
4.7.2 testing the model with the best hyper parameters

In [92]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=
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_onehot
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding))
```

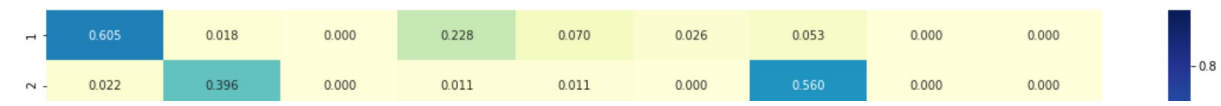
```
Log loss (train) on the stacking classifier : 0.5331398617874226
Log loss (CV) on the stacking classifier : 1.1470486777095854
Log loss (test) on the stacking classifier : 1.1546404170148894
Number of missclassified point : 0.3744360902255639
-------------------- Confusion matrix --------------------
```




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



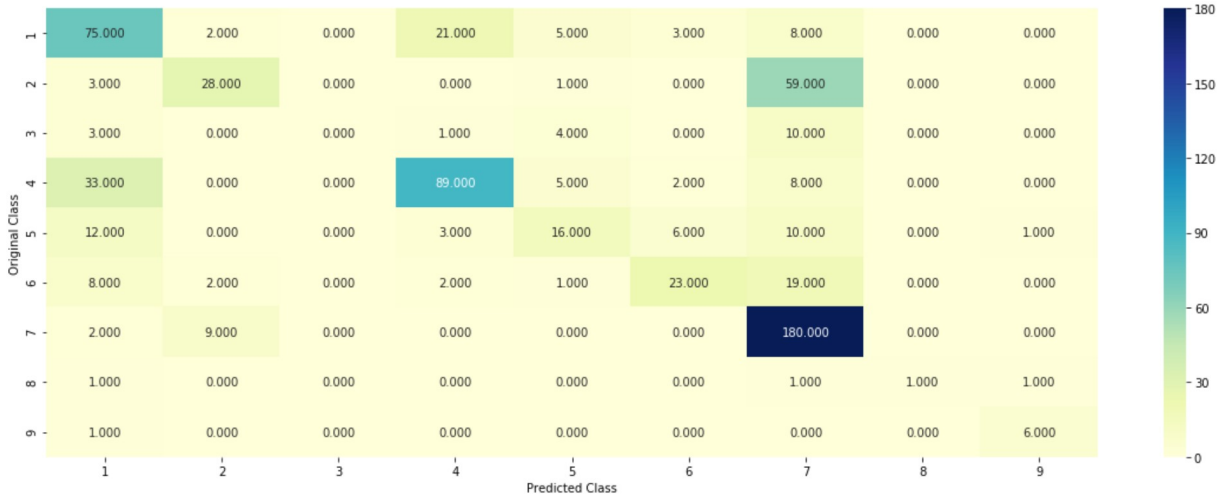

```
-------------------- Recall matrix (Row sum=1) --------------------
```



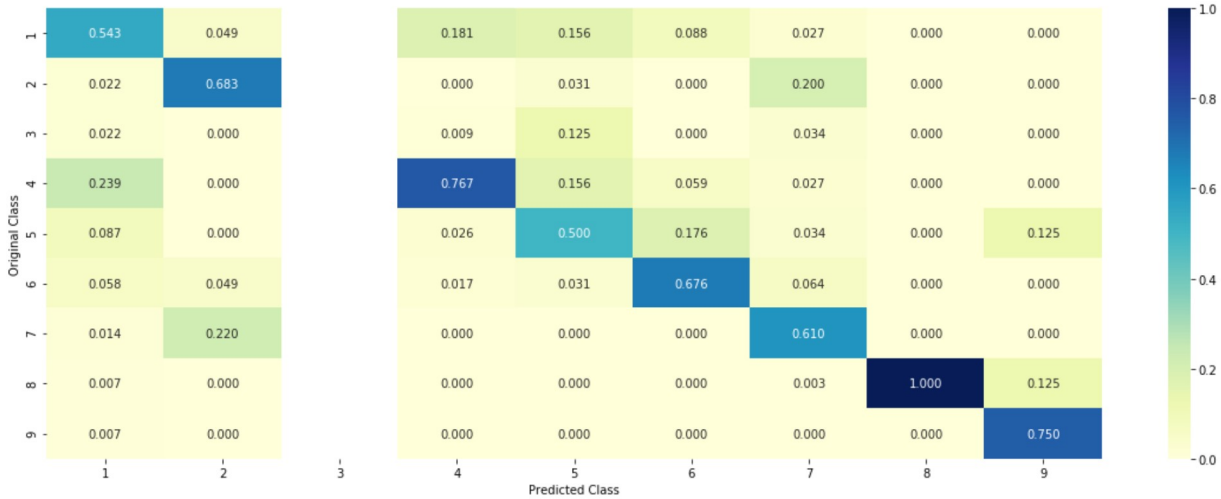
4.7.3 Maximum Voting classifier

In [93]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassi
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_cl
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_pro
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehot
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

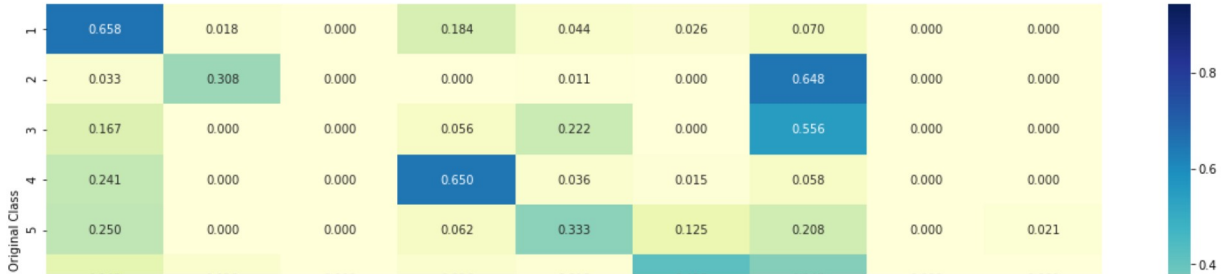
```
Log loss (train) on the VotingClassifier : 0.8308942878615584
Log loss (CV) on the VotingClassifier : 1.1990632287860528
Log loss (test) on the VotingClassifier : 1.20226988269786
Number of missclassified point : 0.37142857142857144
-------------------- Confusion matrix --------------------
```




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




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



5. Assignments

1. Apply All the models with tf-idf features (Replace CountVectorizer with tfidfVectorizer and run the same cells)
2. Instead of using all the words in the dataset, use only the top 1000 words based of tf-idf values
3. Apply Logistic regression with CountVectorizer Features, including both unigrams and bigrams
4. Try any of the feature engineering techniques discussed in the course to reduce the CV and test log-loss to a value less than 1.0

Logistic regression with CountVectorizer Features, including both unigrams and bigrams

In [105]:
```
train_variation=train_df['Variation'].values
cv_variation=cv_df['Variation'].values
test_variation=test_df['Variation'].values

train_gene=train_df['Gene'].values
cv_gene=cv_df['Gene'].values
test_gene=test_df['Gene'].values

train_text=train_df['TEXT'].values
cv_text=cv_df['TEXT'].values
```

In [106]:
```
from sklearn.feature_extraction.text import CountVectorizer
model=CountVectorizer(min_df=5,ngram_range=(1, 2))
```

In [107]:
```
train_variation=model.fit_transform(train_variation)
test_variation=model.transform(test_variation)
cv_variation=model.transform(cv_variation)

train_gene=model.fit_transform(train_gene)
test_gene=model.transform(test_gene)
cv_gene=model.transform(cv_gene)

train_text=model.fit_transform(train_text)
test_text=model.transform(test_text)
cv_text=model.transform(cv_text)
```

In [108]:
```
train_variation=normalize(train_variation,axis=0)
test_variation=normalize(test_variation,axis=0)
cv_variation=normalize(cv_variation,axis=0)

train_gene=normalize(train_gene,axis=0)
test_gene=normalize(test_gene,axis=0)
cv_gene=normalize(cv_gene,axis=0)

train_text=normalize(train_text,axis=0)
test_text=normalize(test_text,axis=0)
cv_text=normalize(cv_text,axis=0)
```

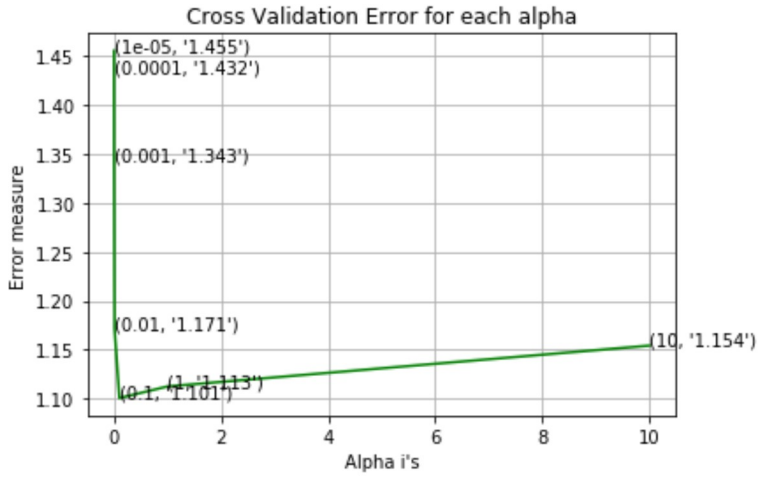
In [109]:
```
train_y = np.array(list(train_df['Class']))
test_y = np.array(list(test_df['Class']))
cv_y = np.array(list(cv_df['Class']))
```

In [110]:
```
train_data=hstack([train_variation,train_gene,train_text]).tocsr()
cv_data=hstack([cv_variation,cv_gene,cv_text]).tocsr()
```

In [115]:
```
cv_scores=[]
alpha=[10 ** x for x in range(-5, 2)]
for k in alpha:
    print("for alpha =", k)
    lr = LogisticRegression(C=k,class_weight='balanced',n_jobs=-1)
    clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
    clf.fit(train_data,y_train)
    cv_score=clf.predict_proba(cv_data)
    print(log_loss(y_cv, cv_score))
    cv_scores.append(log_loss(y_cv, cv_score))
```

```
for alpha = 1e-05
1.4554598137229253
for alpha = 0.0001
1.4323490680720417
for alpha = 0.001
1.3429176569178756
for alpha = 0.01
1.1714751146578346
for alpha = 0.1
1.1007569824258836
for alpha = 1
1.1126316057196264
for alpha = 10
1.1542875876892638
```

In [120]:
```
fig, ax = plt.subplots()
ax.plot(alpha, cv_scores,c='g')
for i, txt in enumerate(np.round(cv_scores,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_scores[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
```

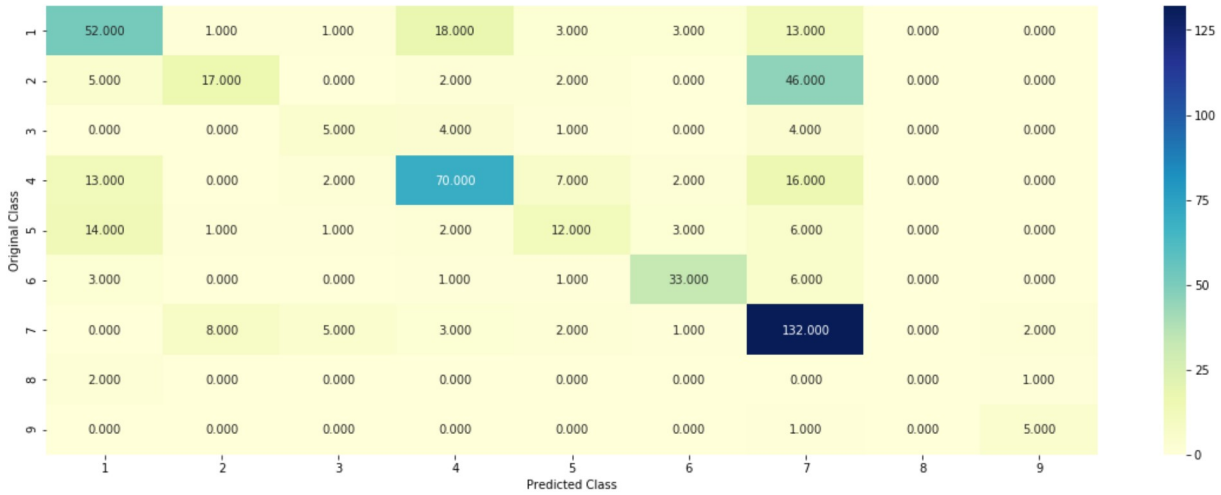


In [121]:
```
lr = LogisticRegression( C=0.1,class_weight='balanced',n_jobs=-1)
clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
clf.fit(train_data,y_train)
test_op=clf.predict_proba(test_data)
```

```
Log Loss value =  1.1311679799913772
```

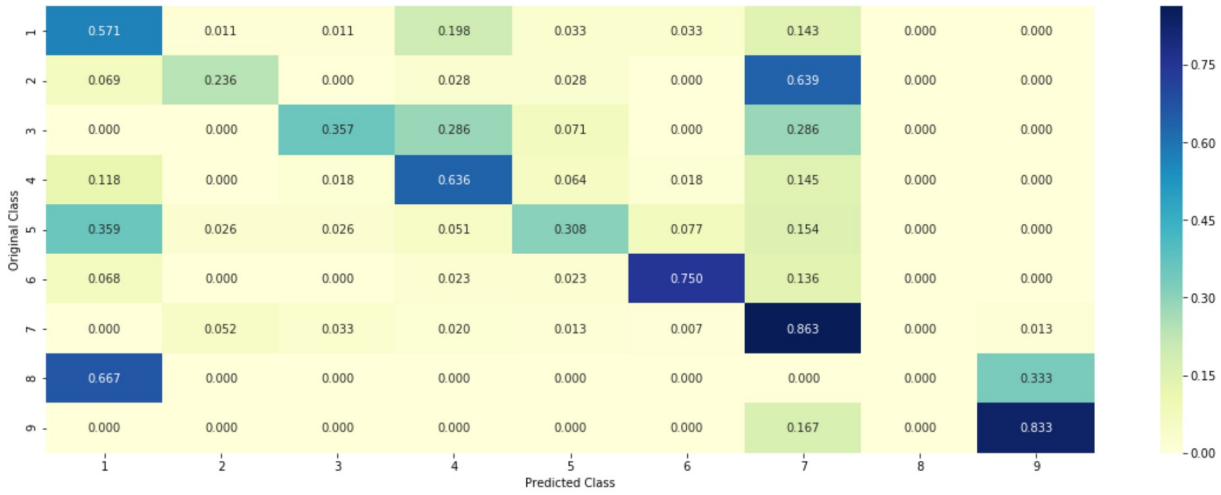
In [122]:
```
lr = LogisticRegression(C=0.1,class_weight='balanced',n_jobs=-1)
clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
```

```
Log loss : 1.127630807140615
Number of mis-classified points : 0.38721804511278196
-------------------- Confusion matrix --------------------
```




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


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



feature engineering techniques to reduce the CV and test log-loss to a value less than 1.0

In [99]:
```
result = pd.merge(data, data_text,on='ID', how='left')
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
y_true = result['Class'].values
result.Gene = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

x_train, x_test, y_train, y_test = train_test_split(result, y_true, stratify=y_true, t
```

In [100]:
```
def get_gv_fea_dict1(alpha, feature, df):
    value_count = x_train[feature].value_counts()
    gv_dict = dict()
    for i, denominator in value_count.items():
        vec = []
        for k in range(1,10):
            cls_cnt = x_train.loc[(x_train['Class']==k) & (x_train[feature]==i)]
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))
        gv_dict[i]=vec
    return gv_dict


def get_gv_feature1(alpha, feature, df):
    gv_dict = get_gv_fea_dict1(alpha, feature, df)
    value_count = x_train[feature].value_counts()
    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])
```

In [101]:
```
alpha = 1

# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature1(alpha, "Gene", x_train))

# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature1(alpha, "Gene", x_test))

# cross validation gene feature
```

In [102]:
```
tfidf = TfidfVectorizer(min_df=3,max_features=1000)
train_gene_feature_onehotCoding = tfidf.fit_transform(x_train['Gene'])
test_gene_feature_onehotCoding = tfidf.transform(x_test['Gene'])
```

In [103]:
```
alpha = 1

# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature1(alpha, "Variation",

# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature1(alpha, "Variation",

# cross validation gene feature
```

In [104]:
```
vtfidf = TfidfVectorizer(min_df=3,max_features=1000 )
train_variation_feature_onehotCoding = vtfidf.fit_transform(x_train['Variation'])
test_variation_feature_onehotCoding = vtfidf.transform(x_test['Variation'])
```

In [105]:
```
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


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(
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TE
            row_index += 1
```

In [106]:
```
text_vectorizer = TfidfVectorizer(min_df=3,max_features=1000 )
train_text_feature_onehotCoding = text_vectorizer.fit_transform(x_train['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*nu
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 i
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))
```

```
Total number of unique words in train data : 125624
```

In [107]:
```
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = x_train[x_train['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(x_train)


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)
```

In [108]:
```
train_text_feature_responseCoding  = get_text_responsecoding(x_train)
test_text_feature_responseCoding  = get_text_responsecoding(x_test)
cv_text_feature_responseCoding  = get_text_responsecoding(x_cv)

# 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_f
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_feat
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_re
```

In [109]:
```
test_text_feature_onehotCoding = text_vectorizer.transform(x_test['TEXT'])
```

In [110]:
```
gene_variation = []

for gene in data['Gene'].values:
    gene_variation.append(gene)

for variation in data['Variation'].values:
```

In [111]:
```
tfidfVectorizer = TfidfVectorizer(min_df=3 ,max_features=1000)
text2 = tfidfVectorizer.fit_transform(gene_variation)
gene_variation_features = tfidfVectorizer.get_feature_names()

train_text = tfidfVectorizer.transform(x_train['TEXT'])
test_text = tfidfVectorizer.transform(x_test['TEXT'])
```

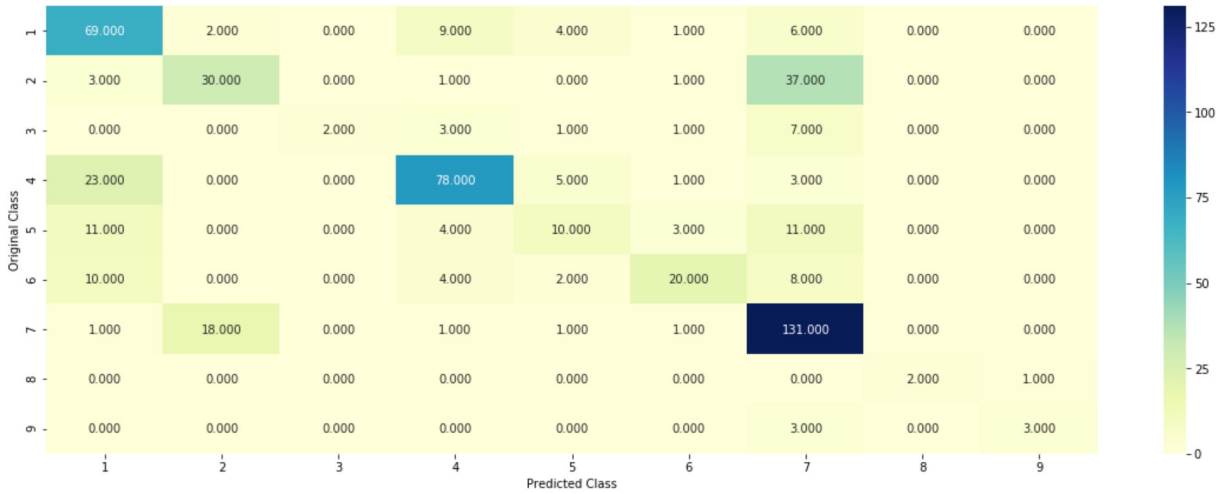
In [112]:
```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_fe
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_

# Adding the train_text feature
train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text))
train_x_onehotCoding = hstack((train_x_onehotCoding, train_text_feature_onehotCoding)
train_y = np.array(list(x_train['Class']))

# Adding the test_text feature
test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text))
test_x_onehotCoding = hstack((test_x_onehotCoding, test_text_feature_onehotCoding)).t
test_y = np.array(list(x_test['Class']))

# Adding the cv_text feature
cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text))
cv_x_onehotCoding = hstack((cv_x_onehotCoding, cv_text_feature_onehotCoding)).tocsr()
cv_y = np.array(list(x_cv['Class']))


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_va
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_varia
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_f

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_re
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_response
```

In [120]:
```
alpha = [10 ** x for x in range(-7, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log')
    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=
    # to avoid rounding error while multiplying probabilites we use log-probability e
    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', l
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

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

predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ',
      alpha[best_alpha], "The test log loss is:",
```

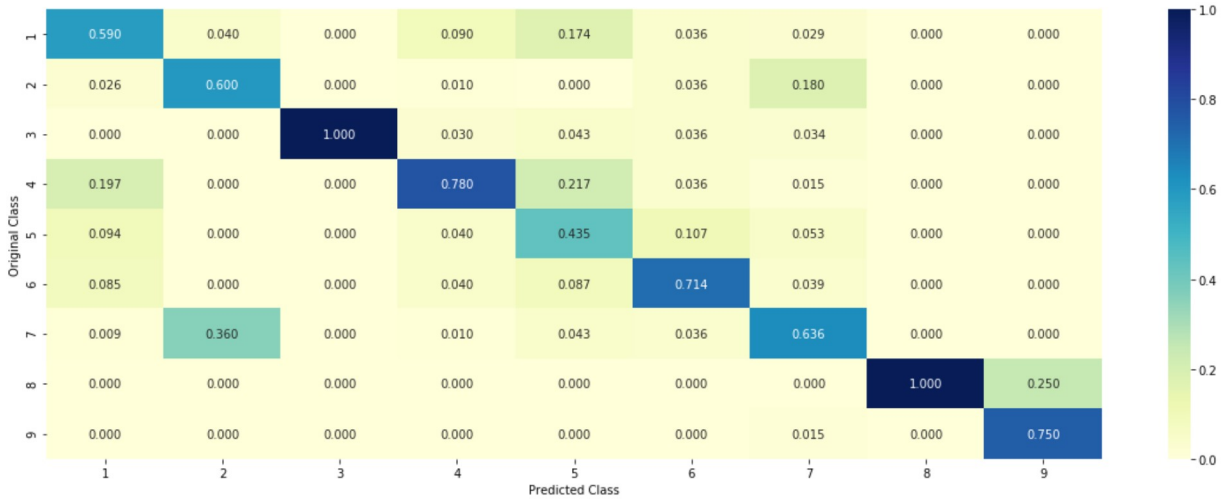
```
for alpha = 1e-07
Log Loss : 1.155468933430785
for alpha = 1e-06
Log Loss : 1.1553191236949283
for alpha = 1e-05
Log Loss : 1.0914343986011419
for alpha = 0.0001
Log Loss : 1.0068795296626258
for alpha = 0.001
Log Loss : 1.0656412412804437
for alpha = 0.01
Log Loss : 1.279583918958128
for alpha = 0.1
Log Loss : 1.6281249080711433
for alpha = 1
Log Loss : 1.7220081457810754
for alpha = 10
Log Loss : 1.7317582374885776
```

In [119]:
```
clf = SGDClassifier(class_weight='balanced',alpha=0.0001, penalty='l2', loss='log',)
```

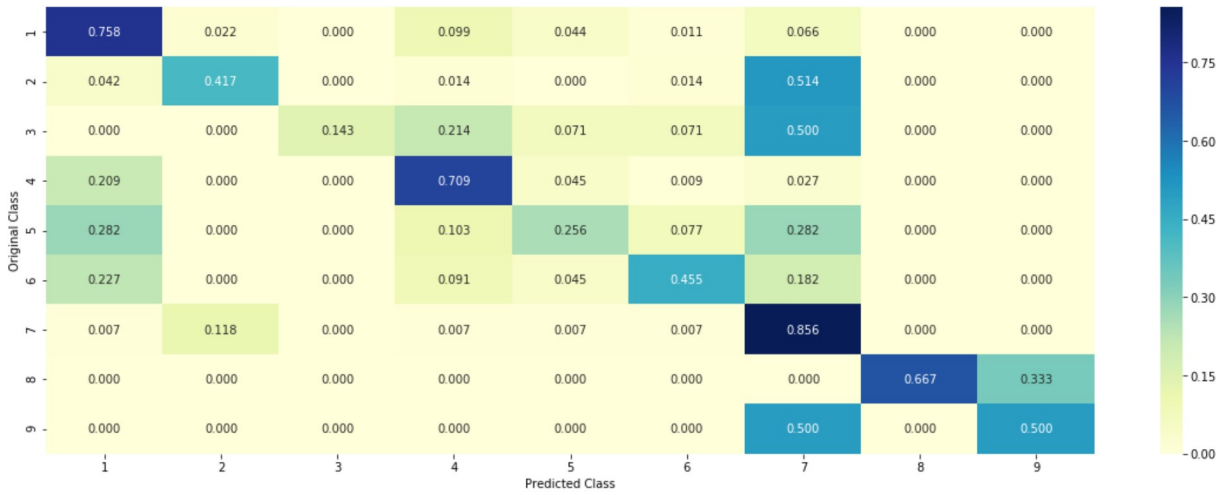
```
Log loss : 0.9992276711851135
Number of mis-classified points : 0.35150375939849626
-------------------- Confusion matrix --------------------
```




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




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



Steps followed

1.took the data and split into train,cv and test in 80:20 ratio

2.then we have declared functions to compute response coding for gene,variation and test with alpha=1

3.then for one hot encoding we have used min_df, max_features

4.after that computation we have used hstack to combine all features

5.then we have used logistic regression model with hyper parameter ranging from 10^-7 to 10^3

6.we got best alpha at 0.0001

s.no	models	train	cv	test	mis-classification
1	naive bayes	0.52	1.16	1.18	37%
2	k-NN	0.69	1.05	1.05	36%
3	logistic regression with class balancing	0.44	1.01	1.02	34%
4	logistic regression without class balancing	0.44	1.051	1.04	34%
5	linear svm	0.47	1.032	1.065	35%
6	Random forest	0.85	1.185	1.196	39%
7	stacking models	0.5	1.147	1.154	37%
8	maximum vote classifier	0.83	1.199	1.202	37%
9	logistic regression with unigram & bigram	1.127	1.1007	1.131	38%
10	feature engineering	0.44	0.99	0.96	35%

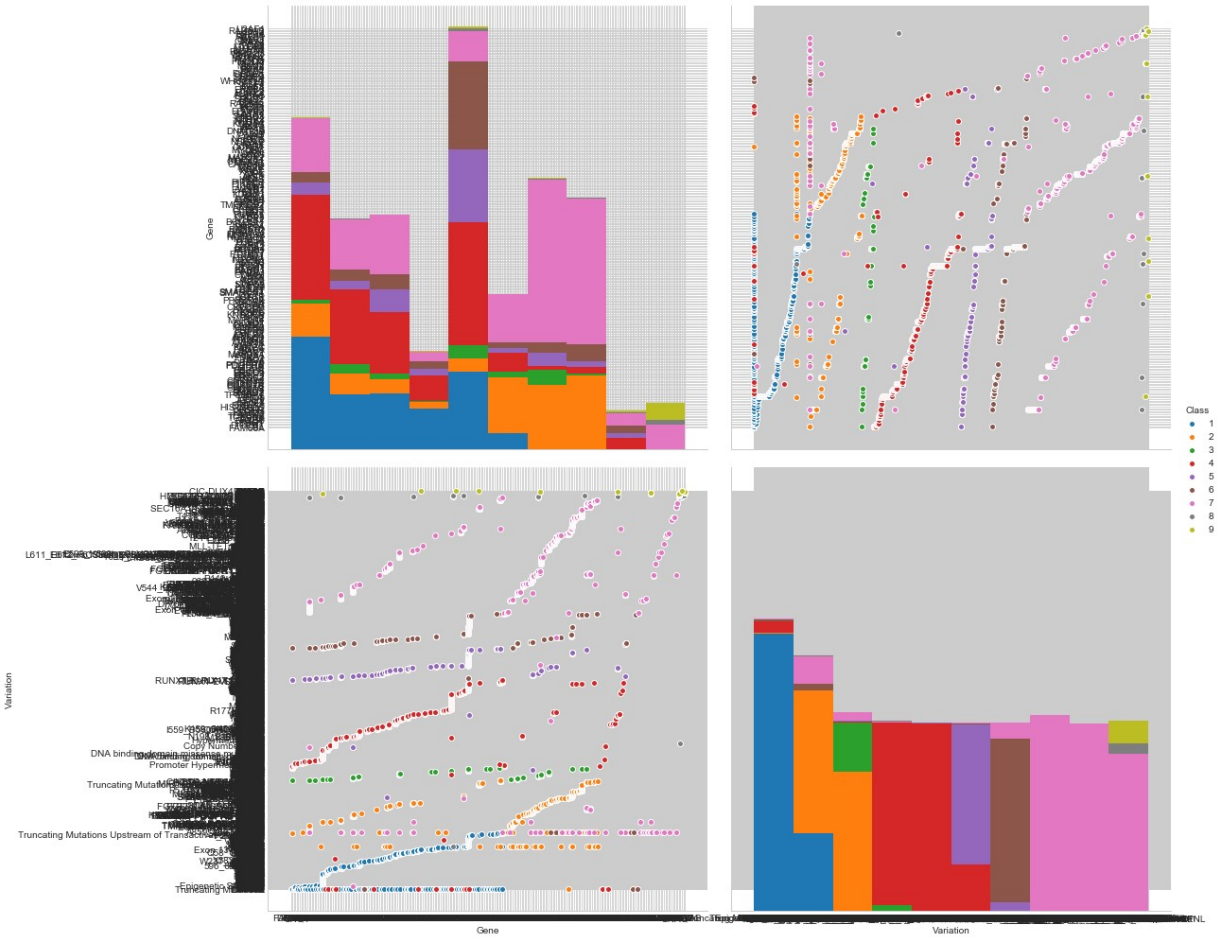
Conclusions

From the above table we can conclude that the best model is logistic regression with class balancing in which we are having less train error and less mis classification, compared to other model.

But when we have done some feature engineering we are able to reduce the log loss less than 1 but our mis classification error is one point higher.

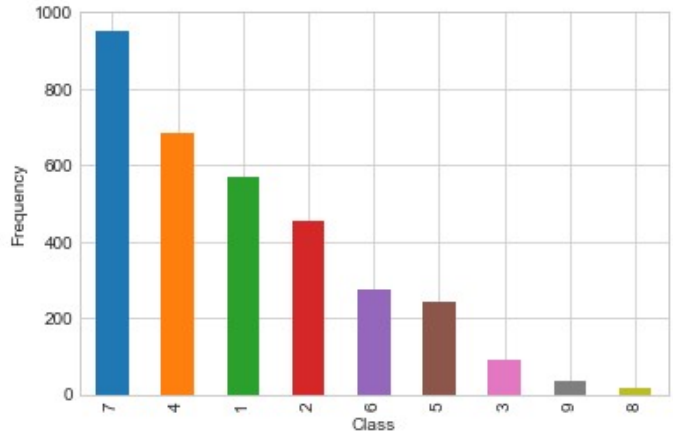
EDA Additional

In [11]:
```
sns.set_style("whitegrid")
sns.pairplot(data,hue='Class',vars=['Gene','Variation'],size=7)
```



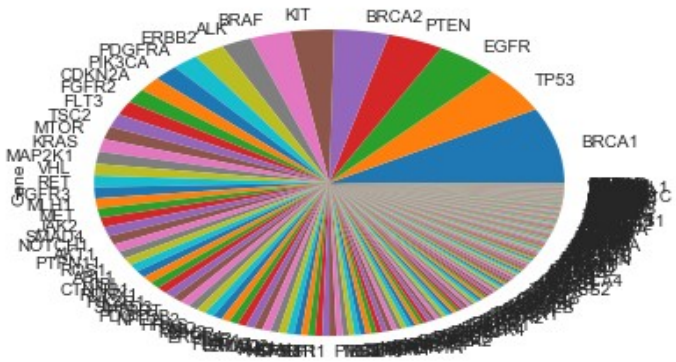
Observation: From the above pair plots we can't infer much about genes but we can see that class '7' is having more frequency

In [43]:
```
plt.figure()
ax = result['Class'].value_counts().plot(kind='bar')
ax.set_xlabel('Class')
ax.set_ylabel('Frequency')
```



Observation: we can observe that class 7&4 are high and 8&9 are very less

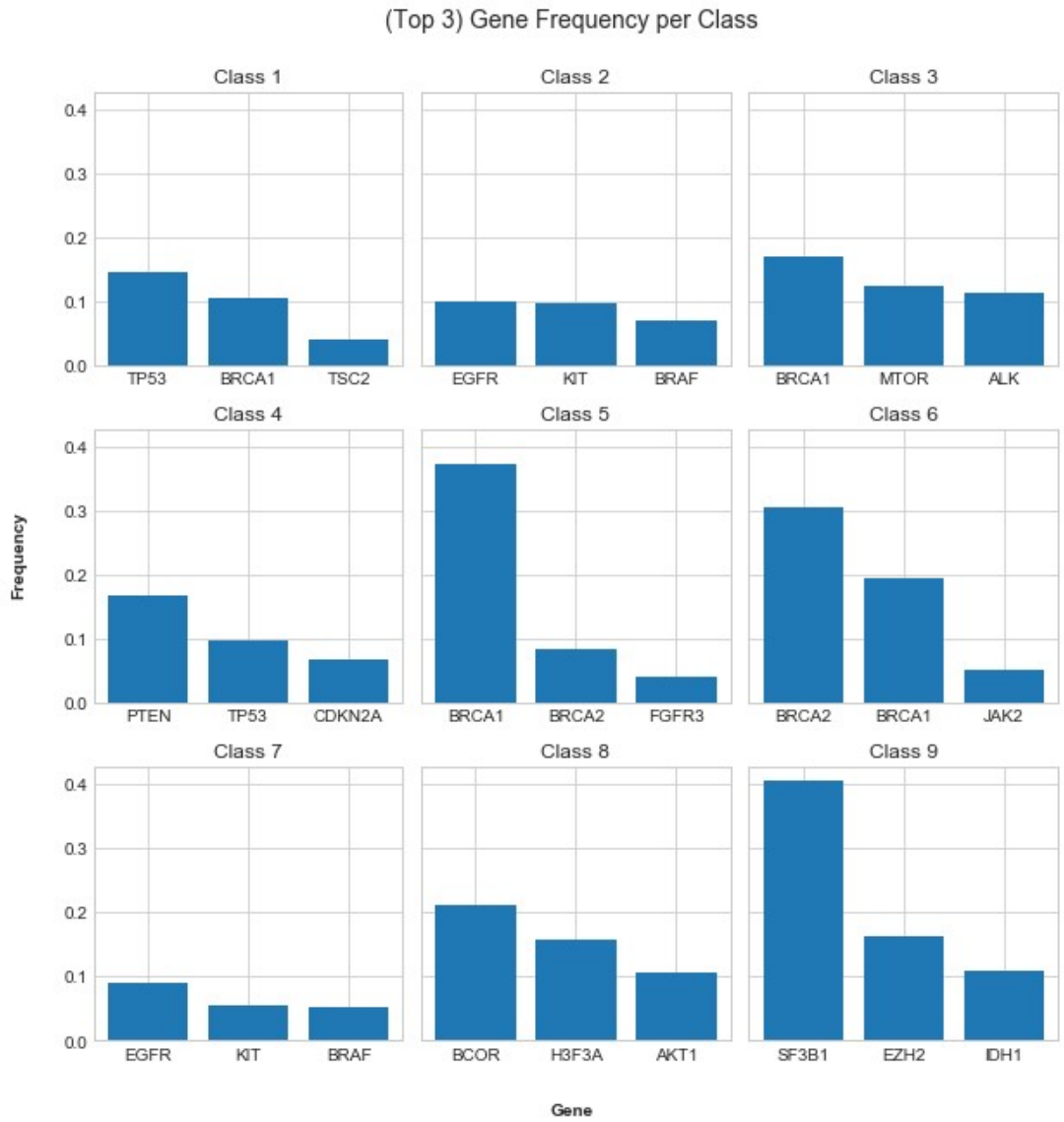
In [44]:
```
plt.figure()
ax = result['Gene'].value_counts().plot(kind='pie')
```



observation: from the pie chart we can see that gene BRCA1 has more frequency followed hy TP53,EGFR,PTEN,BRCA2..etc

In [49]:

```
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(3), val[:3],
           tick_label=label[:3])
    ax.set_title("Class {}".format(g[0]))

fig.text(0.5, 0.97, '(Top 3) Gene Frequency per Class', ha='center', fontsize=14)
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])
```



Observation: we can see that top 3 gene in each class