OBJECTIVE :-

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 top 1000 words based of tf-idf values

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 [83]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
import nltk
nltk.download('stopwords')
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 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
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

3.1. Reading Data

In [2]:
```
!pip install kaggle
from google.colab import files
files.upload()
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.6/dist-packages
(1.5.2)
Requirement already satisfied: urllib3<1.23.0,>=1.15 in /usr/local/lib/python3.
6/dist-packages (from kaggle) (1.22)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.6/dist-packa
ges (from kaggle) (1.11.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.6/dist-package
s (from kaggle) (2018.11.29)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.6/dist
-packages (from kaggle) (2.5.3)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packag
es (from kaggle) (2.18.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-packages
(from kaggle) (4.28.1)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.6/dist-
packages (from kaggle) (2.0.1)
Requirement already satisfied: idna<2.7,>=2.5 in /usr/local/lib/python3.6/dist-
packages (from requests->kaggle) (2.6)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.
6/dist-packages (from requests->kaggle) (3.0.4)
Requirement already satisfied: Unidecode>=0.04.16 in /usr/local/lib/python3.6/d
ist-packages (from python-slugify->kaggle) (1.0.23)
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
```

Out[2]:
```
{'kaggle.json': b'{"username":"pankajkarki","key":"6df9de76323f35cb7449790489c9
d534"}'}
```

In [4]:
```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/

# This permissions change avoids a warning on Kaggle tool startup.
!chmod 600 ~/.kaggle/kaggle.json

!kaggle competitions download -c msk-redefining-cancer-treatment

!ls
```

```
Downloading training_variants.zip to /content
  0% 0.00/24.2k [00:00<?, ?B/s]
100% 24.2k/24.2k [00:00<00:00, 8.68MB/s]
Downloading test_variants.zip to /content
  0% 0.00/47.5k [00:00<?, ?B/s]
100% 47.5k/47.5k [00:00<00:00, 15.0MB/s]
Downloading training_text.zip to /content
 94% 57.0M/61.0M [00:01<00:00, 45.5MB/s]
100% 61.0M/61.0M [00:01<00:00, 57.0MB/s]
Downloading test_text.zip to /content
 90% 89.0M/99.0M [00:00<00:00, 84.6MB/s]
100% 99.0M/99.0M [00:01<00:00, 103MB/s]
Downloading stage2_sample_submission.csv.7z to /content
  0% 0.00/765 [00:00<?, ?B/s]
100% 765/765 [00:00<00:00, 791kB/s]
Downloading stage2_test_variants.csv.7z to /content
  0% 0.00/7.25k [00:00<?, ?B/s]
100% 7.25k/7.25k [00:00<00:00, 7.08MB/s]
Downloading stage2_test_text.csv.7z to /content
 56% 5.00M/8.88M [00:00<00:00, 26.7MB/s]
100% 8.88M/8.88M [00:00<00:00, 35.1MB/s]
Downloading stage1_solution_filtered.csv.7z to /content
  0% 0.00/1.28k [00:00<?, ?B/s]
100% 1.28k/1.28k [00:00<00:00, 1.20MB/s]
Downloading stage_2_private_solution.csv.7z to /content
  0% 0.00/592 [00:00<?, ?B/s]
100% 592/592 [00:00<00:00, 565kB/s]
kaggle.json                     stage2_test_variants.csv.7z
sample_data                     test_text.zip
stage1_solution_filtered.csv.7z  test_variants.zip
stage_2_private_solution.csv.7z  training_text.zip
stage2_sample_submission.csv.7z  training_variants.zip
stage2_test_text.csv.7z
```

In [5]:
```
!unzip test_text.zip
!unzip test_variants.zip
!unzip training_text.zip
!unzip training_variants.zip
```

```
Archive:  test_text.zip
  inflating: test_text
Archive:  test_variants.zip
  inflating: test_variants
Archive:  training_text.zip
  inflating: training_text
Archive:  training_variants.zip
  inflating: training_variants
```

3.1.1. Reading Gene and Variation Data

In [74]:
```
data = pd.read_csv('training_variants')
print('Number of data points : ', data.shape[0])
print('Number of features : ', data.shape[1])
print('Features : ', data.columns.values)
data.head()
```

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

Out[74]:

	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 [75]:
```
# note the seprator in this file
data_text =pd.read_csv("training_text",sep="\|\|",engine="python",names=["ID","TE
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[75]:

	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 [0]:
```
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))


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

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

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

In [77]:
```
#text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_time, "second
```

```
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 : 383.6303250000001 seconds
```

In [78]:
```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[78]:

	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 [79]:
```
result[result.isnull().any(axis=1)]
```

Out[79]:

	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 [0]:
```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variatio
```

In [81]:
```
result[result['ID']==1277]
```

Out[81]:

	ID	Gene	Variation	Class	TEXT
1277	1277	ARID5B	Truncating Mutations	1	ARID5B Truncating Mutations

In [82]:
```
result.shape
```

Out[82]:
```
(3321, 5)
```

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 [0]:
```
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 v
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_t
# split the train data into train and cross validation by maintaining same distri
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_tr
```

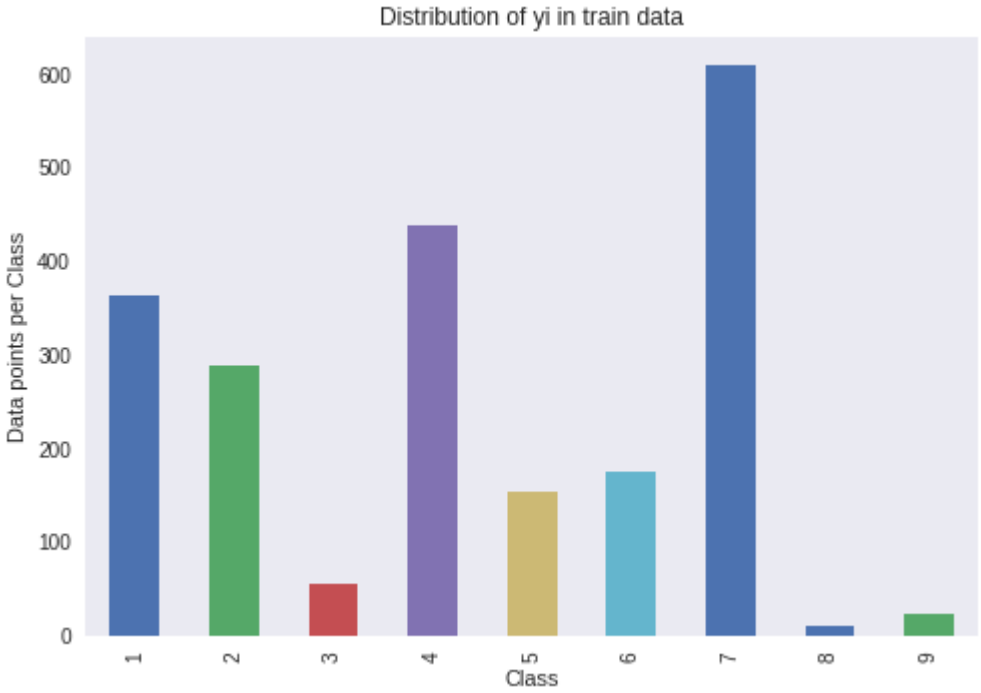
We split the data into train, test and cross validation data sets, preserving the ratio of class distribution in the original data set

In [85]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

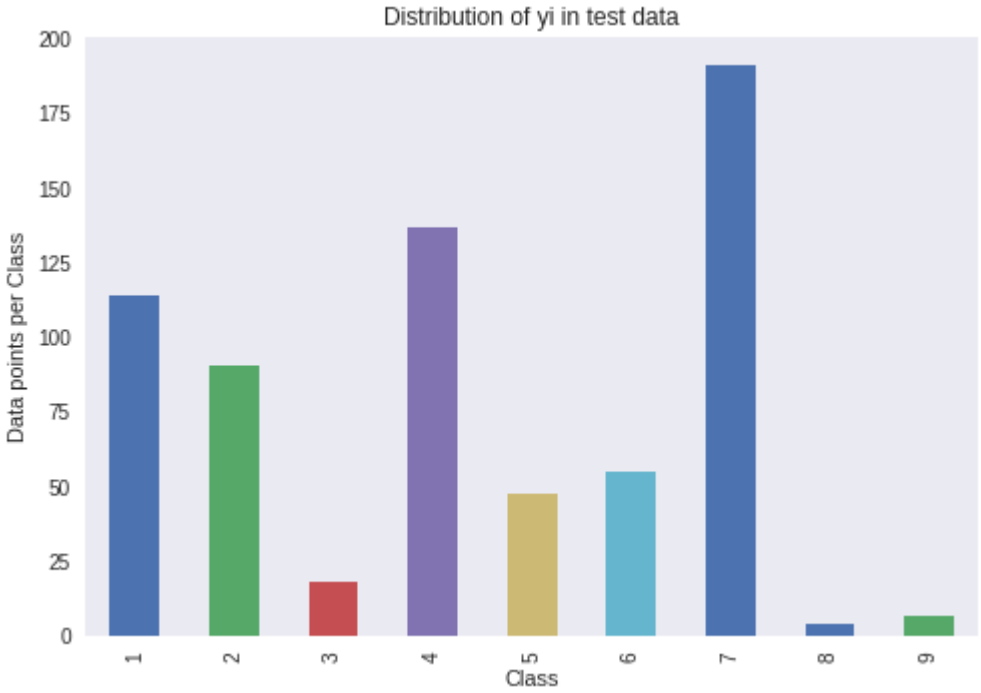
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

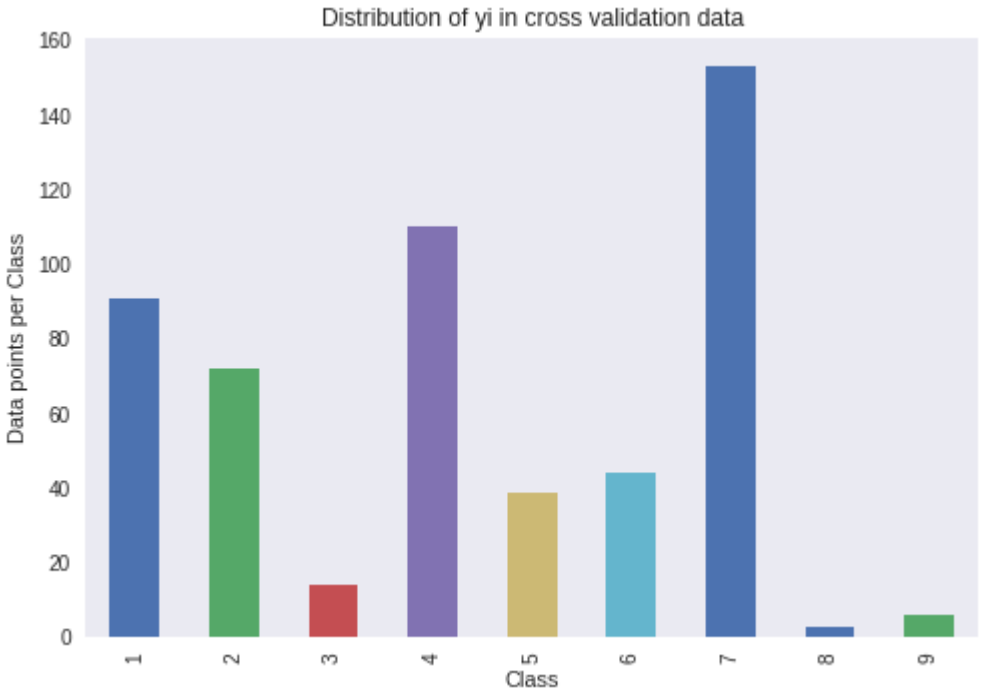
In [86]:
```
# it returns a dict, keys as class labels and values as the number of data points
train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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
# -(train_class_distribution.values): the minus sign will give us in decreasing o
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.val


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
# -(train_class_distribution.values): the minus sign will give us in decreasing o
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.valu

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
# -(train_class_distribution.values): the minus sign will give us in decreasing o
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
```




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



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




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

```
Number of data points in class 9 : 6 ( 1.128 %)
Number of data points in class 8 : 3 ( 0.564 %)
```

3.2 Prediction using a 'Random' Model

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

In [0]:
```
# 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 p

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

    # 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
    # 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 c
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows
    # 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, ytic
    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, ytic
    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, ytic
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

In [88]:
```
# 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 s
# 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_pre


# 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

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

```
Log loss on Cross Validation Data using Random Model 2.525787386472515
Log loss on Test Data using Random Model 2.4737129884864246
-------------------- Confusion matrix --------------------
```


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

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

3.3 Univariate Analysis

In [0]:
```
# 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 tra
# build a vector (1*9) , the first element = (number of times it occured in class
# gv_dict is like a look up table, for every gene it store a (1*9) representation
# 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 eac
    gv_dict = dict()

    # denominator will contain the number of time that particular feature occured
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to
        # vec is 9 diamensional vector
        vec = []
```

```
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=='BRCA
            #         ID   Gene             Variation  Class
            # 2470  2470  BRCA1                S1715C      1
            # 2486  2486  BRCA1                S1841R      1
            # 2614  2614  BRCA1                   M1R      1
            # 2432  2432  BRCA1                L1657P      1
            # 2567  2567  BRCA1                T1685A      1
            # 2583  2583  BRCA1                E1660G      1
            # 2634  2634  BRCA1                W1718L      1
            # cls_cnt.shape[0] will return the number of rows

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

            # cls_cnt.shape[0](numerator) will contain the number of time that pa
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.0681818181818181
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.0612244897959183
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.06818181
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.07284768211920529
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.0733333333333333
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

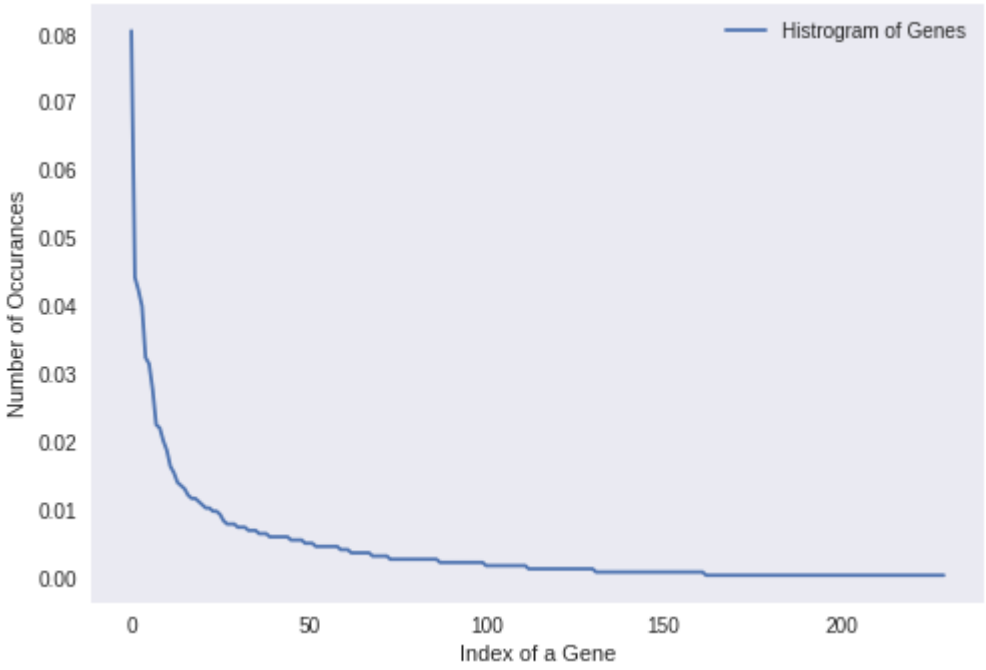
In [90]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 230
BRCA1     171
TP53       94
EGFR       90
BRCA2      85
PTEN       69
KIT        67
BRAF       59
PDGFRA     48
ERBB2      47
PIK3CA     43
Name: Gene, dtype: int64
```

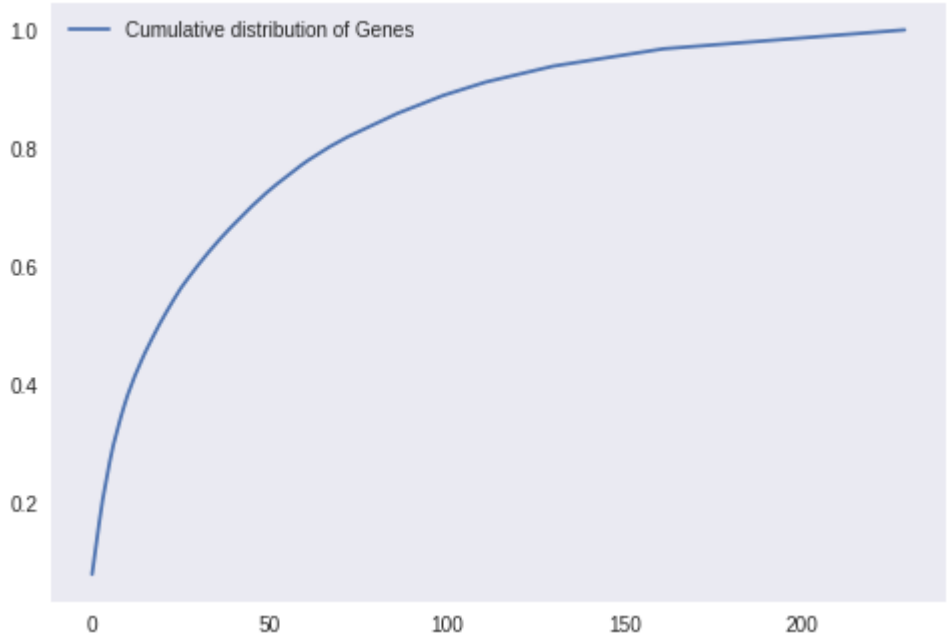
In [91]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in
```

```
Ans: There are 230 different categories of genes in the train data, and they ar
e distibuted as follows
```

In [92]:
```
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 [93]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

Ans.there are two ways we can featurize this variable check out this video: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/ (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 [0]:
```
#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_
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_df
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
```

In [95]:
```
print("train_gene_feature_responseCoding is converted feature using respone codin
```

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

In [0]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [97]:
```
train_df['Gene'].head()
```

Out[97]:
```
948     PDGFRB
2542     BRCA1
1274    PIK3R2
1762      IDH1
702      ASXL1
Name: Gene, dtype: object
```

In [98]:
```
gene_vectorizer.get_feature_names()
```

Out[98]:
```
['abl1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl1',
 'asxl2',
 'atm',
 'atrx',
 'aurka',
 'aurkb',
```

In [99]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot encodin
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding met
hod. The shape of gene feature: (2124, 229)
```

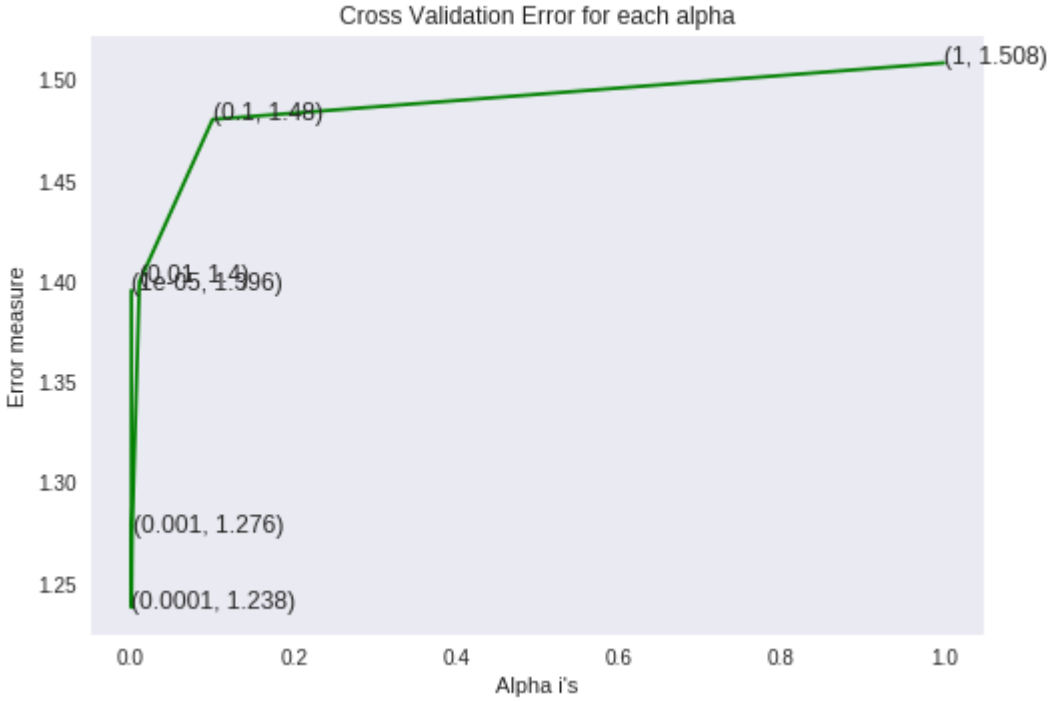
Q4. How good is this gene feature in predicting y_i?

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 [100]:
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/gener
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rat
# 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 Grad
# 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=
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_

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_sta
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:",
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log
predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",l
```

```
For values of alpha =  1e-05 The log loss is: 1.3955823789857584
For values of alpha =  0.0001 The log loss is: 1.2382800290182474
For values of alpha =  0.001 The log loss is: 1.2760896976836704
```

```
For values of alpha =  0.01 The log loss is: 1.3998523450899127
For values of alpha =  0.1 The log loss is: 1.4804157601145391
For values of alpha =  1 The log loss is: 1.5084428145532818
```




```
For values of best alpha =  0.0001 The train log loss is: 1.0177409158655868
For values of best alpha =  0.0001 The cross validation log loss is: 1.23828002
90182474
For values of best alpha =  0.0001 The test log loss is: 1.2169023513782375
```

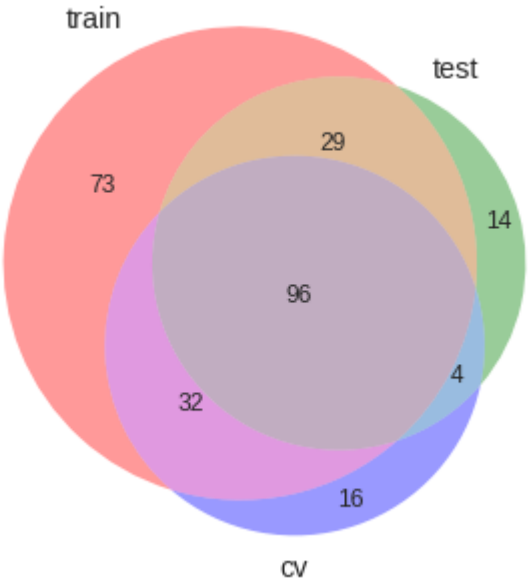
Q5. Is the Gene feature stable across all the data sets (Test, Train, Cross validation)?

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

In [101]:
```
print("Q6. How many data points in Test and CV datasets are covered by the ", uni

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

```
Q6. How many data points in Test and CV datasets are covered by the  230  genes
in train dataset?
Ans
1. In test data 638 out of 665 : 95.93984962406014
2. In cross validation data 507 out of  532 : 95.30075187969925
```

In [102]:
```
import matplotlib.pyplot as plt
from matplotlib_venn import venn3

# Make the venn diagram
venn3(subsets = ([set(train_df['Gene'].values),set(test_df['Gene'].values),set(cv
plt.show()
```



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?

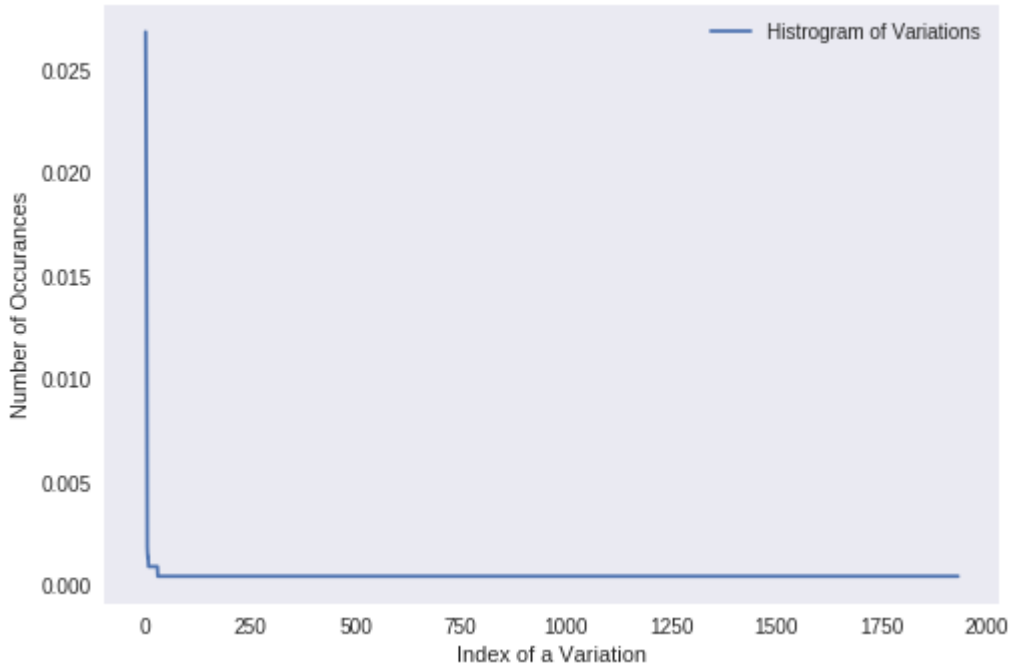
In [103]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1932
Truncating_Mutations    57
Deletion                49
Amplification           38
Fusions                 23
Overexpression           4
G12V                     3
Q61L                     3
F384L                    2
G12D                     2
E542K                    2
Name: Variation, dtype: int64
```

In [104]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categories of vari
```

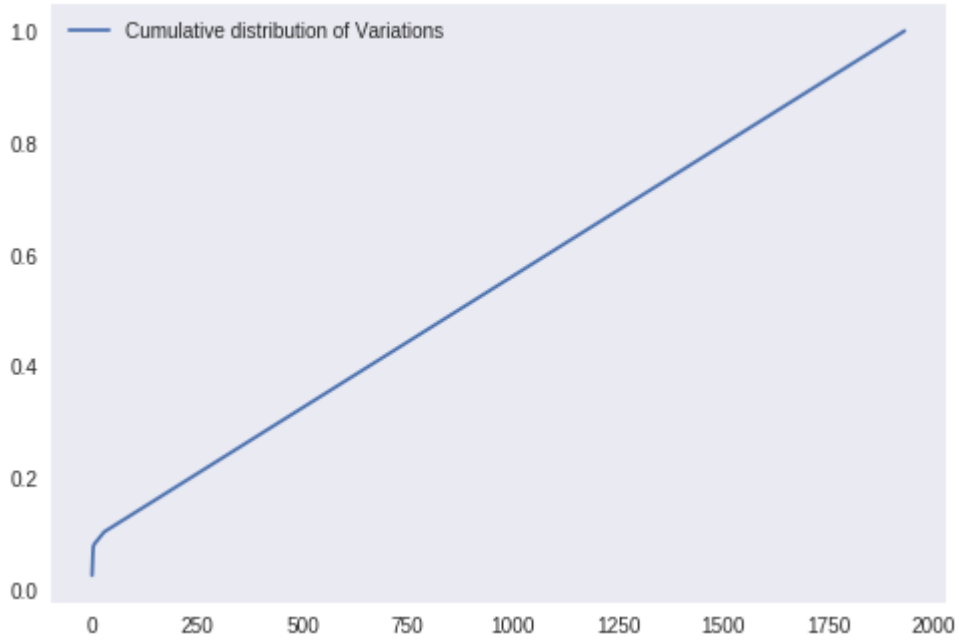
```
Ans: There are 1932 different categories of variations in the train data, and t
hey are distibuted as follows
```

In [105]:
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



In [106]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.02683616 0.04990584 0.06779661 ... 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 [0]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variatio
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation",
```

In [108]:
```
print("train_variation_feature_responseCoding is a converted feature using the re
```

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

In [0]:
```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_d
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Var
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variati
```

In [110]:
```
print("train_variation_feature_onehotEncoded is converted feature using the onne-
```

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot e
ncoding method. The shape of Variation feature: (2124, 1959)
```

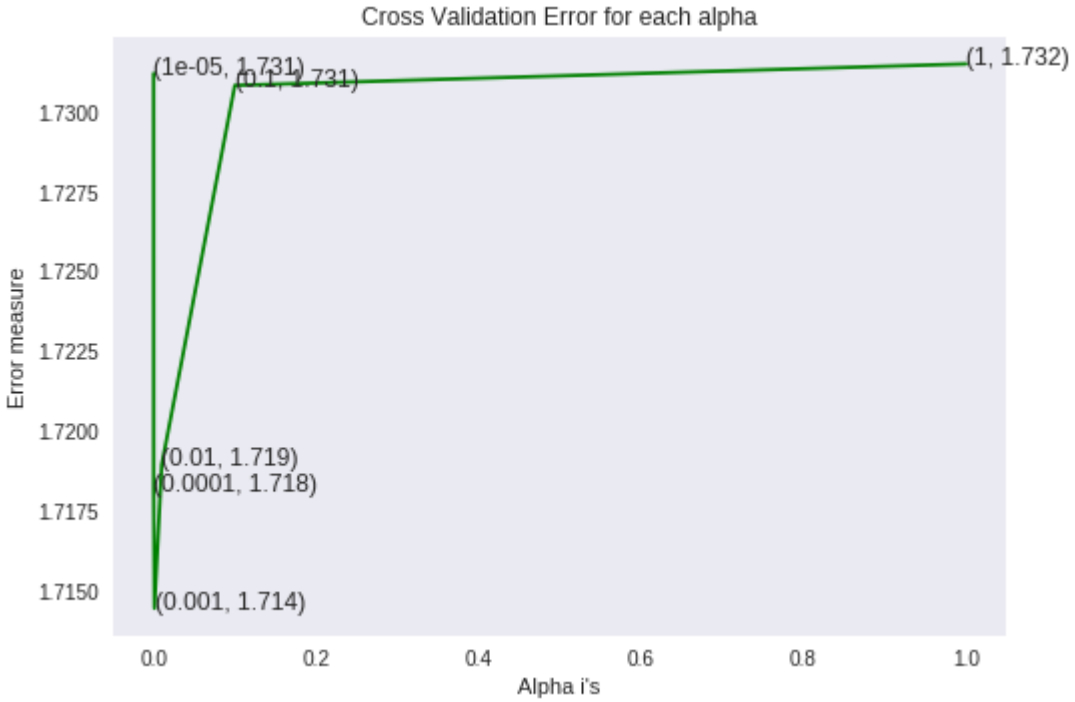
Q10. How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

In [111]:
```
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/genera
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rat
# 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 Grad
# 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=
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_

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_sta
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:",
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log
predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",l
```

```
For values of alpha =  1e-05 The log loss is: 1.7312116440022174
```

```
For values of alpha =  0.0001 The log loss is: 1.718141658385793
For values of alpha =  0.001 The log loss is: 1.7144478512494536
For values of alpha =  0.01 The log loss is: 1.718951719697346
For values of alpha =  0.1 The log loss is: 1.730836621524426
For values of alpha =  1 The log loss is: 1.7315095464548302
```




```
For values of best alpha =  0.001 The train log loss is: 1.08188520833864
For values of best alpha =  0.001 The cross validation log loss is: 1.714447851
2494536
For values of best alpha =  0.001 The test log loss is: 1.714884587747109
```

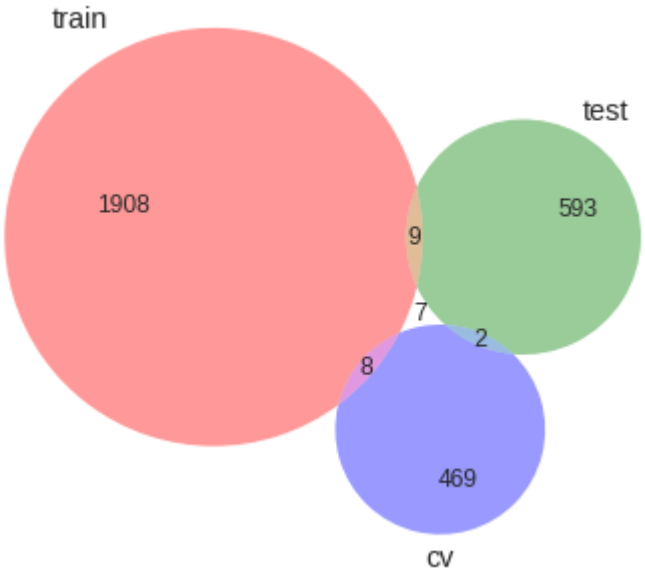
Q11. Is the Variation feature stable across all the data sets (Test, Train, Cross validation)?

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

In [112]:
```
print("Q12. How many data points are covered by total ", unique_variations.shape[
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shap
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(c
```

```
Q12. How many data points are covered by total  1932  genes in test and cross v
alidation data sets?
Ans
1. In test data 68 out of 665 : 10.225563909774436
2. In cross validation data 61 out of  532 : 11.466165413533833
```

In [113]:
```
import matplotlib.pyplot as plt
from matplotlib_venn import venn3

# Make the venn diagram
venn3(subsets = ([set(train_df['Variation'].values),set(test_df['Variation'].valu
plt.show()
```



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 [0]:
```
# cls_text is a data frame
# for every row in data fram consider the 'TEXT'
# split the words by space
# make a dict with those words
# increment its count whenever we see that word

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

In [0]:
```
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.
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row
            row_index += 1
    return text_feature_responseCoding
```

In [116]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in t
text_vectorizer = TfidfVectorizer(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
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 time
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))


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

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

In [0]:
```
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = train_df[train_df['Class']==i]
    # build a word dict based on the words in that class
    dict_list.append(extract_dictionary_paddle(cls_text))
    # append it to dict_list

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


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

In [0]:
```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
test_text_feature_responseCoding  = get_text_responsecoding(test_df)
cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)
```

In [0]:
```
# 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_te
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_featur
```

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

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

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

In [0]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , re
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

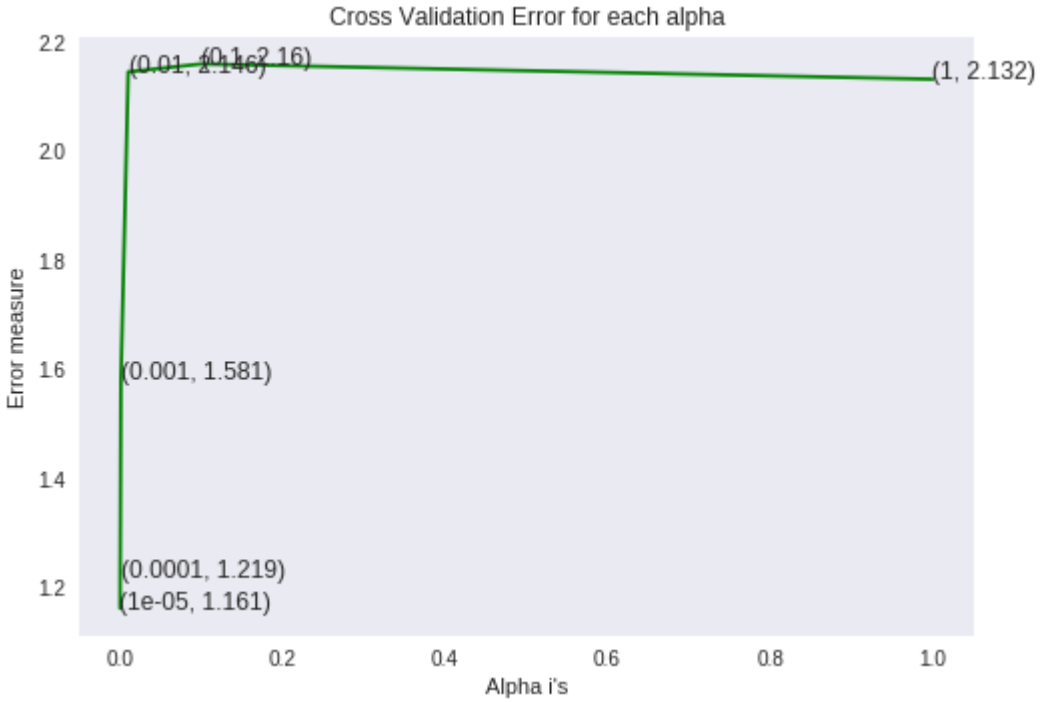
In [122]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({252.26605166742783: 1, 176.45667237161715: 1, 140.04574799033637: 1,
130.67866917925568: 1, 125.1685537213252: 1, 118.04530545020928: 1, 117.47662
824248445: 1, 116.35246915999613: 1, 113.33714315535984: 1, 111.1256616417569
5: 1, 107.09738517116413: 1, 89.7364784656502: 1, 87.84045759682655: 1, 79.35
304449166624: 1, 79.28747838885722: 1, 77.64168091203743: 1, 77.4594310511125
6: 1, 77.0369247176263: 1, 76.91860359523676: 1, 76.1145424043782: 1, 74.2116
4246683001: 1, 72.06184737759995: 1, 71.54364665136097: 1, 70.66753854592757:
1, 68.2496520099553: 1, 65.13186638333399: 1, 65.04207978286996: 1, 64.385655
84005319: 1, 63.733654327196966: 1, 63.51447504762927: 1, 63.353842685238895:
1, 63.29541334427019: 1, 62.425500624499165: 1, 60.59209171364802: 1, 59.2785
59871991746: 1, 57.34022221077641: 1, 56.22253218398862: 1, 55.82289228919340
5: 1, 55.18142557118034: 1, 52.56074047971346: 1, 50.690924076135474: 1, 50.2
0944570776891: 1, 49.31963718863089: 1, 48.84267899557314: 1, 48.592365767425
4: 1, 47.37002950658376: 1, 47.11042132833309: 1, 46.395056224444915: 1, 44.8
835727845218: 1, 44.28356070001744: 1, 44.023900728127096: 1, 43.930486947653
94: 1, 43.55145973579665: 1, 43.35174592466596: 1, 43.25719581559103: 1, 43.2
05879717303226: 1, 42.75701065241467: 1, 42.702253623445: 1, 42.5245842193738
7: 1, 42.24455019775725: 1, 41.7320139919214: 1, 41.55300818175305: 1, 41.292
230551239655: 1, 41.08004939283904: 1, 40.249137435739364: 1, 40.138494942040
```

In [123]:
```
# Train a Logistic regression+Calibration model using text features whicha re on-
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/gener
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rat
# 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 Grad
# 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=
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_

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_sta
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:",
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log
predict_y = sig_clf.predict_proba(test_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",l
```

```
For values of alpha =  1e-05 The log loss is: 1.1610240234532136
```

```
For values of alpha =  0.0001 The log loss is: 1.219436516066769
For values of alpha =  0.001 The log loss is: 1.5806349185566204
For values of alpha =  0.01 The log loss is: 2.145589550928892
For values of alpha =  0.1 The log loss is: 2.1601723945645444
For values of alpha =  1 The log loss is: 2.1321230506006605
```




```
For values of best alpha =  1e-05 The train log loss is: 0.7436051228288942
For values of best alpha =  1e-05 The cross validation log loss is: 1.161024023
4532136
For values of best alpha =  1e-05 The test log loss is: 1.1826009304463512
```

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

Ans. Yes, it seems like!

In [0]:
```
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))
    return len1,len2
```

In [125]:
```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train dat
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in tr
```

```
94.8 % of word of test data appeared in train data
94.1 % of word of Cross Validation appeared in train data
```

In [126]:
```
import matplotlib.pyplot as plt
from matplotlib_venn import venn3

# Make the venn diagram
venn3(subsets = ([set(train_df['TEXT'].values),set(test_df['TEXT'].values),set(cv
plt.show()
```

4. Machine Learning Models

In [0]:
```
#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
    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))
    plot_confusion_matrix(test_y, pred_y)
```

In [0]:
```
def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

In [0]:
```
# 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(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 [{}]".form
        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 [{}]"
        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 [{}]".form

    print("Out of the top ",no_features," features ", word_present, "are present
```

Stacking the three types of features

In [0]:
```
# 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_varia
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variatio
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feat

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

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

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


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,trai
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_v
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variati

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_fea
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_featur
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_resp
```

In [131]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_on
print("(number of data points * number of features) in test data = ", test_x_oneh
print("(number of data points * number of features) in cross validation data =",
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 3188)
(number of data points * number of features) in test data =  (665, 3188)
(number of data points * number of features) in cross validation data = (532, 3
188)
```

In [132]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_re
print("(number of data points * number of features) in test data = ", test_x_resp
print("(number of data points * number of features) in cross validation data =",
```

```
 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, 2
7)
```

4.1. Base Line Model

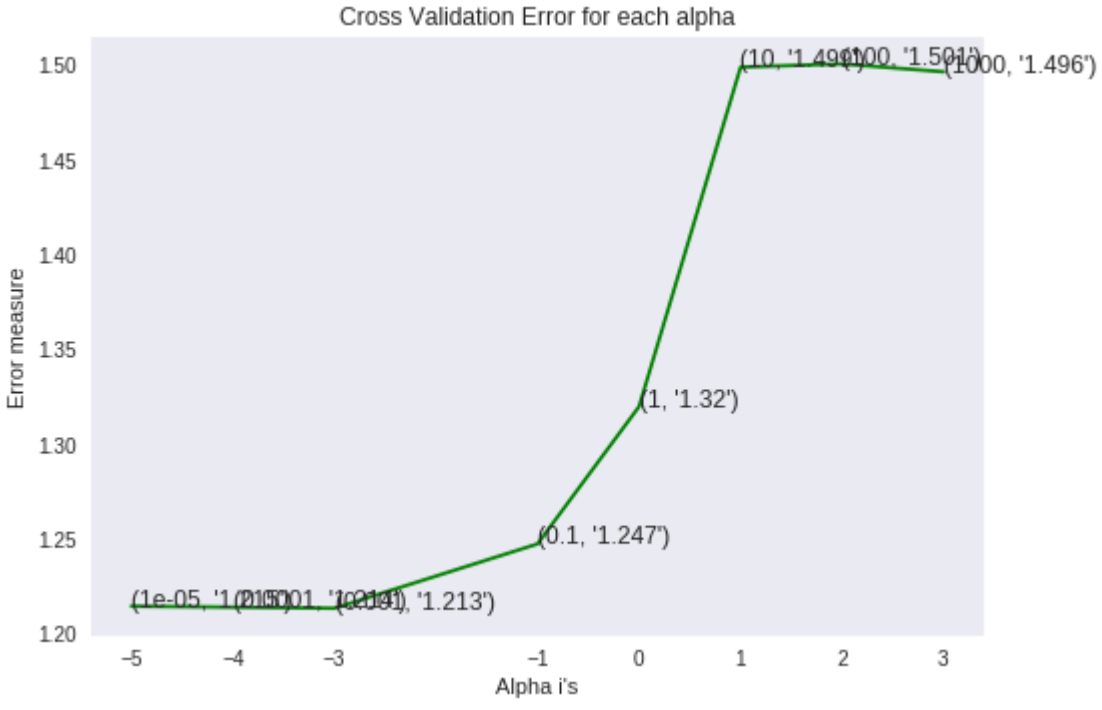
4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

In [170]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/st
# -------------------------
# 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/les
# -----------------------


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


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_,
    # to avoid rounding error while multiplying probabilites we use log-probabili
    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:",
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",l


# Variables that will be used in the end to make comparison table of all models
nb_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding), labels=
nb_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=clf.class
nb_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding), labels=clf
```

```
for alpha = 1e-05
Log Loss : 1.2145724823920876
for alpha = 0.0001
Log Loss : 1.2140266417697827
for alpha = 0.001
Log Loss : 1.2134678589675338
for alpha = 0.1
Log Loss : 1.2473940685716034
for alpha = 1
Log Loss : 1.3198475179149676
for alpha = 10
Log Loss : 1.4988357011607183
for alpha = 100
Log Loss : 1.5005269906402512
for alpha = 1000
Log Loss : 1.4963665637451866
```



```
For values of best alpha =  0.001 The train log loss is: 0.5054619475695253
For values of best alpha =  0.001 The cross validation log loss is: 1.213467858
9675338
For values of best alpha =  0.001 The test log loss is: 1.222175891271175
```

4.1.1.2. Testing the model with best hyper paramters

In [171]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/st
# -------------------------
# 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/les
# -----------------------


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

#Variables that will be used in the end to make comparison table of models
nb_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)- cv_y))/
```

```
Log Loss : 1.2134678589675338
Number of missclassified point : 0.39285714285714285
-------------------- Confusion matrix --------------------
```

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

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

4.1.1.3. Feature Importance, Correctly classified point

In [172]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_one
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['
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0592 0.0643 0.0088 0.0833 0.0329 0.0301 0.71
56 0.0029 0.0029]]
Actual Class : 2
--------------------------------------------------
14 Text feature [activation] present in test data point [True]
16 Text feature [activated] present in test data point [True]
19 Text feature [inhibitor] present in test data point [True]
20 Text feature [downstream] present in test data point [True]
21 Text feature [kinase] present in test data point [True]
22 Text feature [cells] present in test data point [True]
23 Text feature [activating] present in test data point [True]
24 Text feature [expressing] present in test data point [True]
25 Text feature [contrast] present in test data point [True]
26 Text feature [signaling] present in test data point [True]
27 Text feature [presence] present in test data point [True]
28 Text feature [also] present in test data point [True]
29 Text feature [sensitive] present in test data point [True]
30 Text feature [addition] present in test data point [True]
31 Text feature [growth] present in test data point [True]
32 Text feature [independent] present in test data point [True]
33 Text feature [treatment] present in test data point [True]
34 Text feature [well] present in test data point [True]
35 Text feature [phosphorylation] present in test data point [True]
36 Text feature [10] present in test data point [True]
37 Text feature [shown] present in test data point [True]
38 Text feature [inhibition] present in test data point [True]
39 Text feature [factor] present in test data point [True]
40 Text feature [treated] present in test data point [True]
41 Text feature [however] present in test data point [True]
42 Text feature [similar] present in test data point [True]
43 Text feature [previously] present in test data point [True]
44 Text feature [mutations] present in test data point [True]
45 Text feature [compared] present in test data point [True]
46 Text feature [higher] present in test data point [True]
47 Text feature [oncogenic] present in test data point [True]
48 Text feature [constitutive] present in test data point [True]
49 Text feature [activate] present in test data point [True]
50 Text feature [may] present in test data point [True]
51 Text feature [inhibitors] present in test data point [True]
52 Text feature [increased] present in test data point [True]
53 Text feature [found] present in test data point [True]
54 Text feature [cell] present in test data point [True]
55 Text feature [recently] present in test data point [True]
56 Text feature [mechanism] present in test data point [True]
60 Text feature [potential] present in test data point [True]
```

```
61 Text feature [suggest] present in test data point [True]
62 Text feature [absence] present in test data point [True]
63 Text feature [showed] present in test data point [True]
64 Text feature [tyrosine] present in test data point [True]
65 Text feature [constitutively] present in test data point [True]
66 Text feature [enhanced] present in test data point [True]
67 Text feature [receptor] present in test data point [True]
68 Text feature [24] present in test data point [True]
69 Text feature [total] present in test data point [True]
70 Text feature [pathways] present in test data point [True]
71 Text feature [increase] present in test data point [True]
72 Text feature [identified] present in test data point [True]
73 Text feature [without] present in test data point [True]
74 Text feature [fig] present in test data point [True]
75 Text feature [mutation] present in test data point [True]
76 Text feature [therapeutic] present in test data point [True]
77 Text feature [proliferation] present in test data point [True]
78 Text feature [although] present in test data point [True]
79 Text feature [3b] present in test data point [True]
80 Text feature [results] present in test data point [True]
81 Text feature [interestingly] present in test data point [True]
82 Text feature [mutant] present in test data point [True]
83 Text feature [observed] present in test data point [True]
84 Text feature [reported] present in test data point [True]
85 Text feature [two] present in test data point [True]
86 Text feature [12] present in test data point [True]
87 Text feature [different] present in test data point [True]
88 Text feature [phospho] present in test data point [True]
89 Text feature [described] present in test data point [True]
90 Text feature [20] present in test data point [True]
91 Text feature [culture] present in test data point [True]
92 Text feature [respectively] present in test data point [True]
93 Text feature [point] present in test data point [True]
94 Text feature [consistent] present in test data point [True]
95 Text feature [18] present in test data point [True]
97 Text feature [antibodies] present in test data point [True]
98 Text feature [discussion] present in test data point [True]
99 Text feature [including] present in test data point [True]
Out of the top  100  features  79 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [173]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_one
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['
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0599 0.1236 0.0093 0.0683 0.0331 0.032  0.66
79 0.0031 0.0028]]
Actual Class : 7
--------------------------------------------------
14 Text feature [activation] present in test data point [True]
16 Text feature [activated] present in test data point [True]
19 Text feature [inhibitor] present in test data point [True]
20 Text feature [downstream] present in test data point [True]
21 Text feature [kinase] present in test data point [True]
22 Text feature [cells] present in test data point [True]
23 Text feature [activating] present in test data point [True]
24 Text feature [expressing] present in test data point [True]
25 Text feature [contrast] present in test data point [True]
26 Text feature [signaling] present in test data point [True]
27 Text feature [presence] present in test data point [True]
28 Text feature [also] present in test data point [True]
29 Text feature [sensitive] present in test data point [True]
30 Text feature [addition] present in test data point [True]
31 Text feature [growth] present in test data point [True]
32 Text feature [independent] present in test data point [True]
33 Text feature [treatment] present in test data point [True]
34 Text feature [well] present in test data point [True]
35 Text feature [phosphorylation] present in test data point [True]
36 Text feature [10] present in test data point [True]
37 Text feature [shown] present in test data point [True]
38 Text feature [inhibition] present in test data point [True]
39 Text feature [factor] present in test data point [True]
40 Text feature [treated] present in test data point [True]
41 Text feature [however] present in test data point [True]
42 Text feature [similar] present in test data point [True]
43 Text feature [previously] present in test data point [True]
44 Text feature [mutations] present in test data point [True]
45 Text feature [compared] present in test data point [True]
46 Text feature [higher] present in test data point [True]
47 Text feature [oncogenic] present in test data point [True]
50 Text feature [may] present in test data point [True]
51 Text feature [inhibitors] present in test data point [True]
52 Text feature [increased] present in test data point [True]
53 Text feature [found] present in test data point [True]
54 Text feature [cell] present in test data point [True]
55 Text feature [recently] present in test data point [True]
56 Text feature [mechanism] present in test data point [True]
60 Text feature [potential] present in test data point [True]
61 Text feature [suggest] present in test data point [True]
62 Text feature [absence] present in test data point [True]
```

```
63 Text feature [showed] present in test data point [True]
64 Text feature [tyrosine] present in test data point [True]
66 Text feature [enhanced] present in test data point [True]
67 Text feature [receptor] present in test data point [True]
68 Text feature [24] present in test data point [True]
69 Text feature [total] present in test data point [True]
70 Text feature [pathways] present in test data point [True]
71 Text feature [increase] present in test data point [True]
72 Text feature [identified] present in test data point [True]
73 Text feature [without] present in test data point [True]
74 Text feature [fig] present in test data point [True]
75 Text feature [mutation] present in test data point [True]
76 Text feature [therapeutic] present in test data point [True]
77 Text feature [proliferation] present in test data point [True]
78 Text feature [although] present in test data point [True]
79 Text feature [3b] present in test data point [True]
80 Text feature [results] present in test data point [True]
81 Text feature [interestingly] present in test data point [True]
82 Text feature [mutant] present in test data point [True]
83 Text feature [observed] present in test data point [True]
84 Text feature [reported] present in test data point [True]
85 Text feature [two] present in test data point [True]
86 Text feature [12] present in test data point [True]
87 Text feature [different] present in test data point [True]
89 Text feature [described] present in test data point [True]
90 Text feature [20] present in test data point [True]
91 Text feature [culture] present in test data point [True]
92 Text feature [respectively] present in test data point [True]
93 Text feature [point] present in test data point [True]
94 Text feature [consistent] present in test data point [True]
95 Text feature [18] present in test data point [True]
96 Text feature [survival] present in test data point [True]
97 Text feature [antibodies] present in test data point [True]
98 Text feature [discussion] present in test data point [True]
99 Text feature [including] present in test data point [True]
Out of the top  100  features  76 are present in query point
```

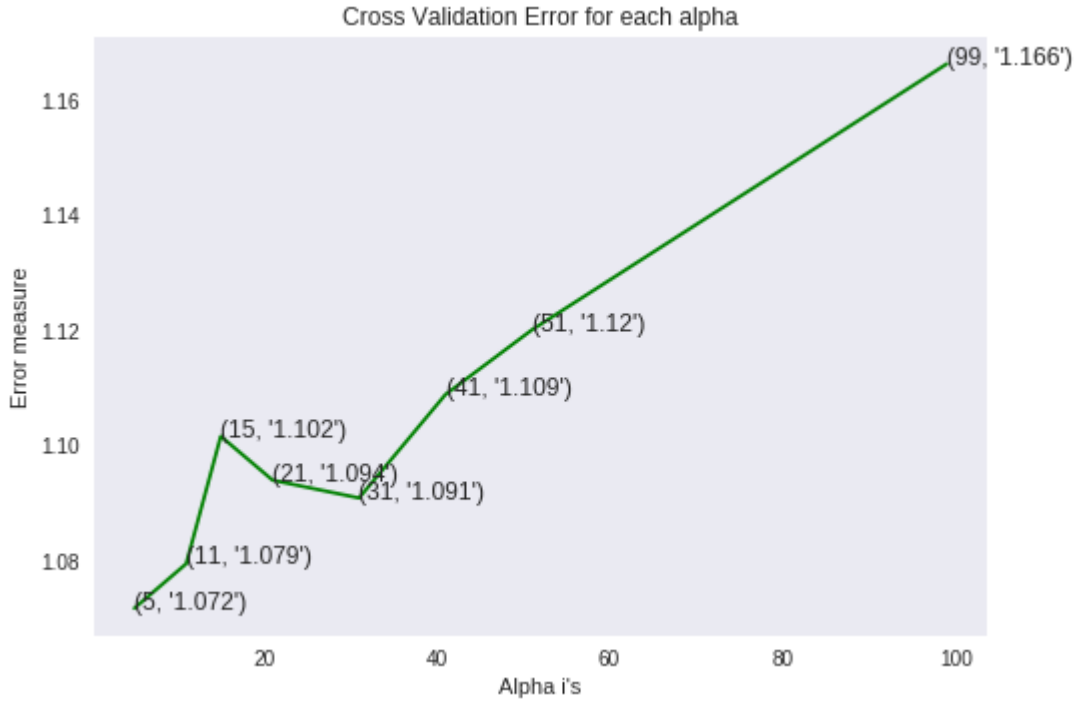
4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [174]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modu
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_s
# 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/les
#-------------------------------------


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

# Variables that will be used in the end to make comparison table of all models
knn_train = log_loss(y_train, sig_clf.predict_proba(train_x_responseCoding), labe
knn_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_responseCoding), labels=clf.cl
knn_test = log_loss(y_test, sig_clf.predict_proba(test_x_responseCoding), labels=
```

```
for alpha = 5
Log Loss : 1.0717436548852675
for alpha = 11
Log Loss : 1.0794733330523927
for alpha = 15
Log Loss : 1.1016076675004276
for alpha = 21
Log Loss : 1.093977530290967
for alpha = 31
Log Loss : 1.0908905980351284
for alpha = 41
Log Loss : 1.1088581453752637
for alpha = 51
Log Loss : 1.1200756181538045
for alpha = 99
Log Loss : 1.16625338161877
```



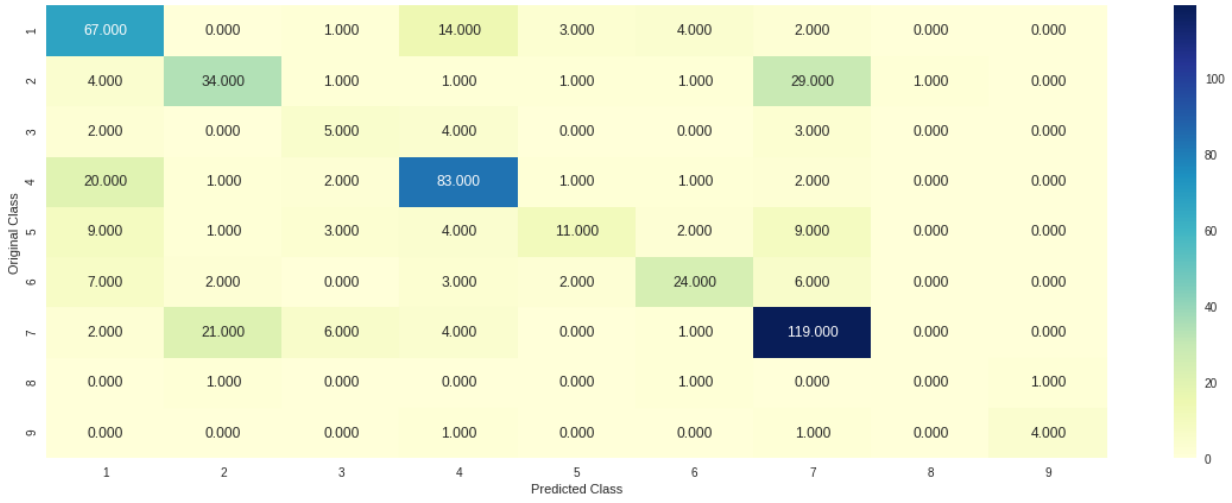

```
For values of best alpha =  5 The train log loss is: 0.47911637068502594
For values of best alpha =  5 The cross validation log loss is: 1.0717436548852
```

```
675
For values of best alpha =  5 The test log loss is: 1.0411793983349593
```

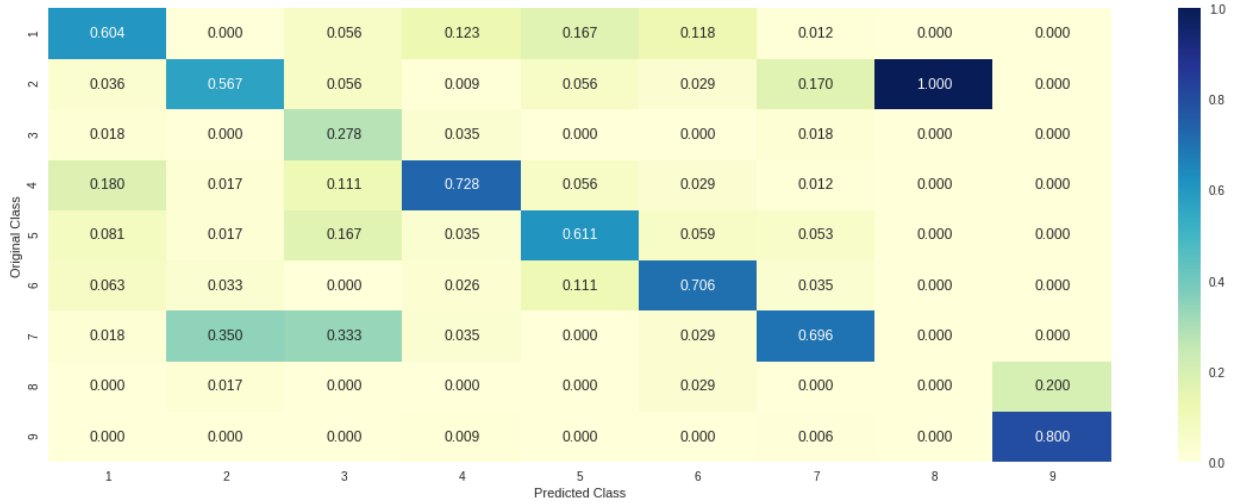
4.2.2. Testing the model with best hyper paramters

In [175]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modu
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_s
# 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/les
#-------------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseC

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

# Variables that will be used in the end to make comparison table of models
knn_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_responseCoding)- cv_y
```

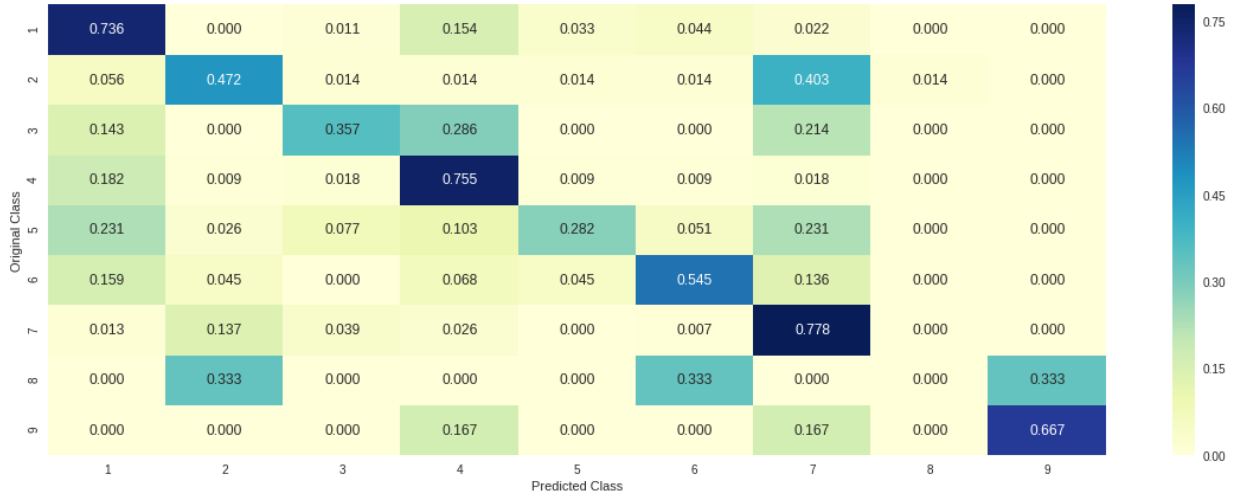
```
Log loss : 1.0717436548852675
Number of mis-classified points : 0.34774436090225563
-------------------- Confusion matrix --------------------
```




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




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



4.2.3.Sample Query point -1

In [176]:
```
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)
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 4
Actual Class : 2
The  5  nearest neighbours of the test points belongs to classes [7 7 7 7 7]
Fequency of nearest points : Counter({7: 5})
```

4.2.4. Sample Query Point-2

In [177]:
```
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
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)
print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of t
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

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

4.3. Logistic Regression

4.3.1. With Class balancing

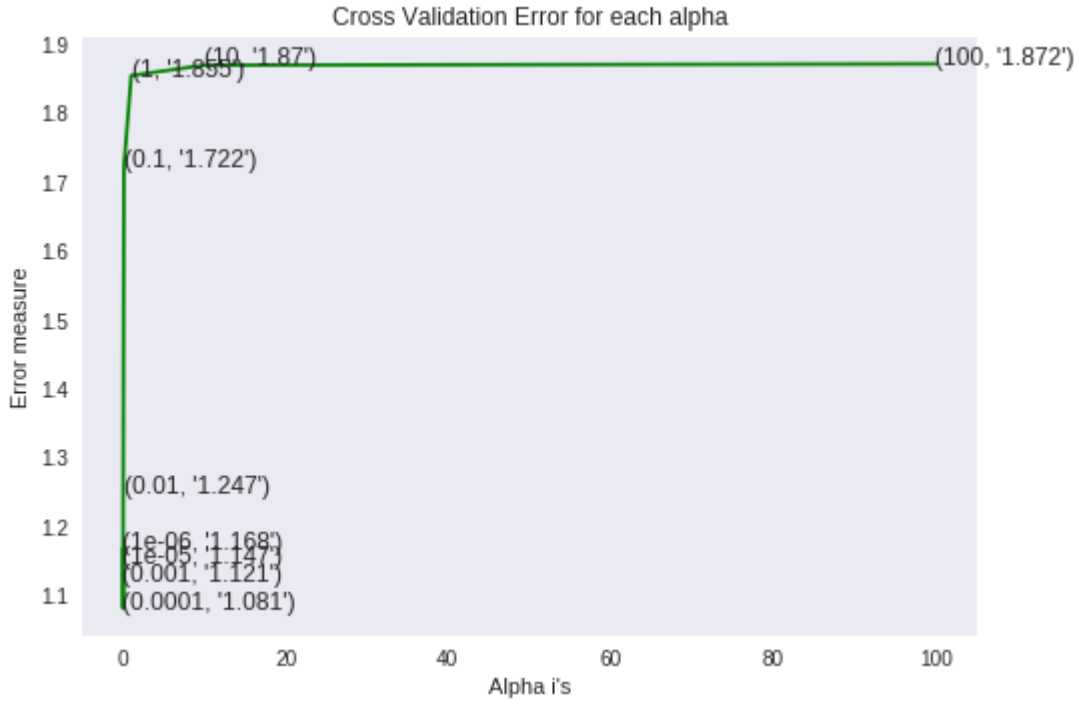
4.3.1.1. Hyper paramter tuning

In [178]:

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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/m
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid
#
# 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
    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_,
    # to avoid rounding error while multiplying probabilites we use log-probabili
    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
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:",
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",l

# Variables that will be used in the end to make comparison table of all models
lr_balance_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding),
lr_balance_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=c
lr_balance_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding), la
```

```
for alpha = 1e-06
Log Loss : 1.1682900960928022
for alpha = 1e-05
Log Loss : 1.147311644382708
for alpha = 0.0001
Log Loss : 1.0809344524198765
for alpha = 0.001
Log Loss : 1.120974975895979
for alpha = 0.01
Log Loss : 1.2474936305575592
for alpha = 0.1
Log Loss : 1.7215078581313388
for alpha = 1
Log Loss : 1.8546341991510482
for alpha = 10
Log Loss : 1.8696473111545933
for alpha = 100
Log Loss : 1.8715082657258804
```



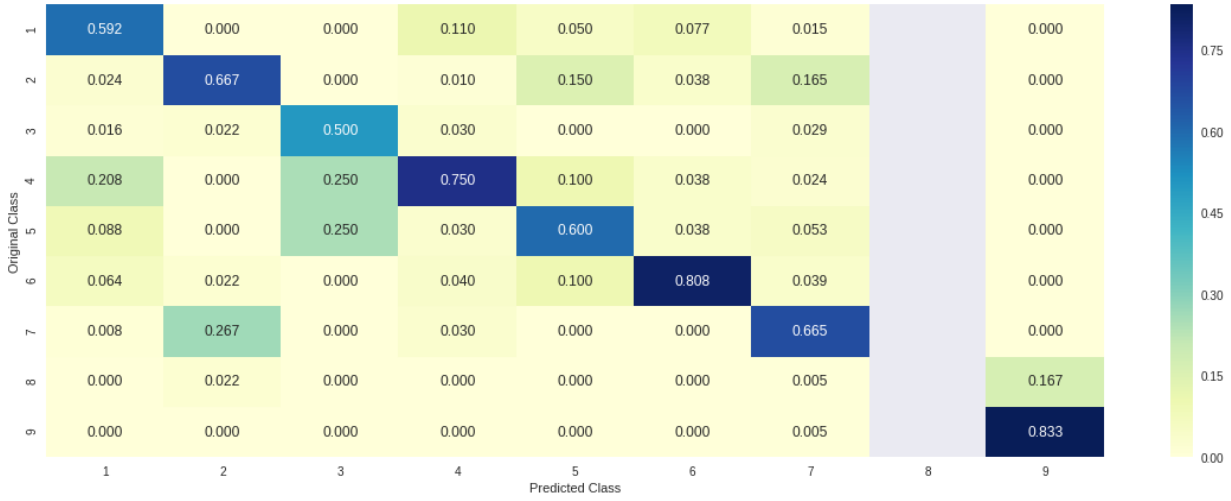
```
For values of best alpha =  0.0001 The train log loss is: 0.4256278197062046
For values of best alpha =  0.0001 The cross validation log loss is: 1.08093445
24198765
For values of best alpha =  0.0001 The test log loss is: 1.0601944839119568
```

4.3.1.2. Testing the model with best hyper paramters

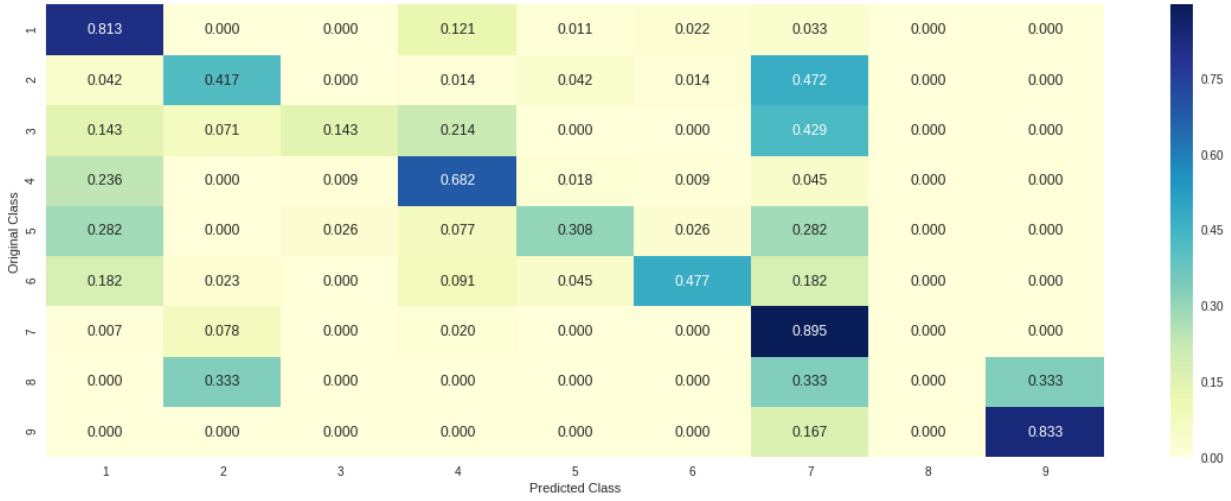
In [179]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/genera
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rat
# 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 Grad
# predict(X)    Predict class labels for samples in X.

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

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

# Variables that will be used in the end to make comparison table of models
lr_balance_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)-
```

```
Log loss : 1.0809344524198765
Number of mis-classified points : 0.3308270676691729
-------------------- Confusion matrix --------------------
```


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



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



4.3.1.3. Feature Importance

In [0]:
```
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_n
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0]," class
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or No
```

4.3.1.3.1. Correctly Classified point

In [181]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_one
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['
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.038  0.0887 0.004  0.1044 0.0492 0.0115 0.69
59 0.0051 0.0033]]
Actual Class : 2
--------------------------------------------------
13 Text feature [activation] present in test data point [True]
21 Text feature [enhanced] present in test data point [True]
22 Text feature [ligand] present in test data point [True]
25 Text feature [activate] present in test data point [True]
28 Text feature [activated] present in test data point [True]
30 Text feature [combination] present in test data point [True]
32 Text feature [downstream] present in test data point [True]
35 Text feature [constitutive] present in test data point [True]
58 Text feature [activating] present in test data point [True]
66 Text feature [lung] present in test data point [True]
67 Text feature [oncogene] present in test data point [True]
68 Text feature [pathways] present in test data point [True]
72 Text feature [inhibitor] present in test data point [True]
75 Text feature [oncogenic] present in test data point [True]
89 Text feature [transformed] present in test data point [True]
93 Text feature [signaling] present in test data point [True]
99 Text feature [3t3] present in test data point [True]
104 Text feature [factor] present in test data point [True]
106 Text feature [codon] present in test data point [True]
136 Text feature [approximately] present in test data point [True]
137 Text feature [phospho] present in test data point [True]
154 Text feature [presence] present in test data point [True]
155 Text feature [3b] present in test data point [True]
159 Text feature [transforming] present in test data point [True]
162 Text feature [derived] present in test data point [True]
173 Text feature [promote] present in test data point [True]
189 Text feature [sensitive] present in test data point [True]
196 Text feature [mechanism] present in test data point [True]
197 Text feature [fold] present in test data point [True]
229 Text feature [demonstrated] present in test data point [True]
234 Text feature [carcinoma] present in test data point [True]
239 Text feature [mechanisms] present in test data point [True]
240 Text feature [medium] present in test data point [True]
245 Text feature [culture] present in test data point [True]
248 Text feature [days] present in test data point [True]
263 Text feature [2a] present in test data point [True]
265 Text feature [leukemia] present in test data point [True]
279 Text feature [contrast] present in test data point [True]
```

```
284 Text feature [extracellular] present in test data point [True]
286 Text feature [per] present in test data point [True]
303 Text feature [volume] present in test data point [True]
315 Text feature [effective] present in test data point [True]
320 Text feature [membrane] present in test data point [True]
322 Text feature [phosphorylated] present in test data point [True]
331 Text feature [24] present in test data point [True]
358 Text feature [bp] present in test data point [True]
362 Text feature [raf] present in test data point [True]
372 Text feature [her2] present in test data point [True]
380 Text feature [inhibited] present in test data point [True]
388 Text feature [fig] present in test data point [True]
396 Text feature [expressing] present in test data point [True]
400 Text feature [ph] present in test data point [True]
407 Text feature [addition] present in test data point [True]
408 Text feature [constitutively] present in test data point [True]
433 Text feature [tyrosine] present in test data point [True]
450 Text feature [regulated] present in test data point [True]
452 Text feature [express] present in test data point [True]
461 Text feature [inhibition] present in test data point [True]
463 Text feature [gefitinib] present in test data point [True]
469 Text feature [interestingly] present in test data point [True]
470 Text feature [molecule] present in test data point [True]
476 Text feature [tumors] present in test data point [True]
489 Text feature [occur] present in test data point [True]
493 Text feature [interface] present in test data point [True]
496 Text feature [concentrations] present in test data point [True]
Out of the top  500  features  65 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [182]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_one
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['
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0155 0.2535 0.0008 0.0373 0.0047 0.026  0.65
79 0.0031 0.0012]]
Actual Class : 7
--------------------------------------------------
13 Text feature [activation] present in test data point [True]
21 Text feature [enhanced] present in test data point [True]
28 Text feature [activated] present in test data point [True]
30 Text feature [combination] present in test data point [True]
32 Text feature [downstream] present in test data point [True]
58 Text feature [activating] present in test data point [True]
60 Text feature [s3] present in test data point [True]
66 Text feature [lung] present in test data point [True]
67 Text feature [oncogene] present in test data point [True]
68 Text feature [pathways] present in test data point [True]
72 Text feature [inhibitor] present in test data point [True]
73 Text feature [overexpression] present in test data point [True]
75 Text feature [oncogenic] present in test data point [True]
77 Text feature [positive] present in test data point [True]
89 Text feature [transformed] present in test data point [True]
93 Text feature [signaling] present in test data point [True]
104 Text feature [factor] present in test data point [True]
115 Text feature [del] present in test data point [True]
136 Text feature [approximately] present in test data point [True]
154 Text feature [presence] present in test data point [True]
155 Text feature [3b] present in test data point [True]
159 Text feature [transforming] present in test data point [True]
161 Text feature [regions] present in test data point [True]
162 Text feature [derived] present in test data point [True]
169 Text feature [factors] present in test data point [True]
173 Text feature [promote] present in test data point [True]
189 Text feature [sensitive] present in test data point [True]
195 Text feature [akt] present in test data point [True]
196 Text feature [mechanism] present in test data point [True]
229 Text feature [demonstrated] present in test data point [True]
234 Text feature [carcinoma] present in test data point [True]
240 Text feature [medium] present in test data point [True]
245 Text feature [culture] present in test data point [True]
248 Text feature [days] present in test data point [True]
258 Text feature [colony] present in test data point [True]
263 Text feature [2a] present in test data point [True]
278 Text feature [cancers] present in test data point [True]
279 Text feature [contrast] present in test data point [True]
284 Text feature [extracellular] present in test data point [True]
286 Text feature [per] present in test data point [True]
303 Text feature [volume] present in test data point [True]
```

```
315 Text feature [effective] present in test data point [True]
321 Text feature [fragment] present in test data point [True]
322 Text feature [phosphorylated] present in test data point [True]
327 Text feature [upon] present in test data point [True]
331 Text feature [24] present in test data point [True]
335 Text feature [distinct] present in test data point [True]
380 Text feature [inhibited] present in test data point [True]
388 Text feature [fig] present in test data point [True]
396 Text feature [expressing] present in test data point [True]
406 Text feature [transformation] present in test data point [True]
407 Text feature [addition] present in test data point [True]
415 Text feature [epithelial] present in test data point [True]
433 Text feature [tyrosine] present in test data point [True]
436 Text feature [driven] present in test data point [True]
443 Text feature [colonies] present in test data point [True]
450 Text feature [regulated] present in test data point [True]
452 Text feature [express] present in test data point [True]
456 Text feature [evaluated] present in test data point [True]
461 Text feature [inhibition] present in test data point [True]
469 Text feature [interestingly] present in test data point [True]
470 Text feature [molecule] present in test data point [True]
476 Text feature [tumors] present in test data point [True]
489 Text feature [occur] present in test data point [True]
491 Text feature [additional] present in test data point [True]
496 Text feature [concentrations] present in test data point [True]
Out of the top  500  features  66 are present in query point
```

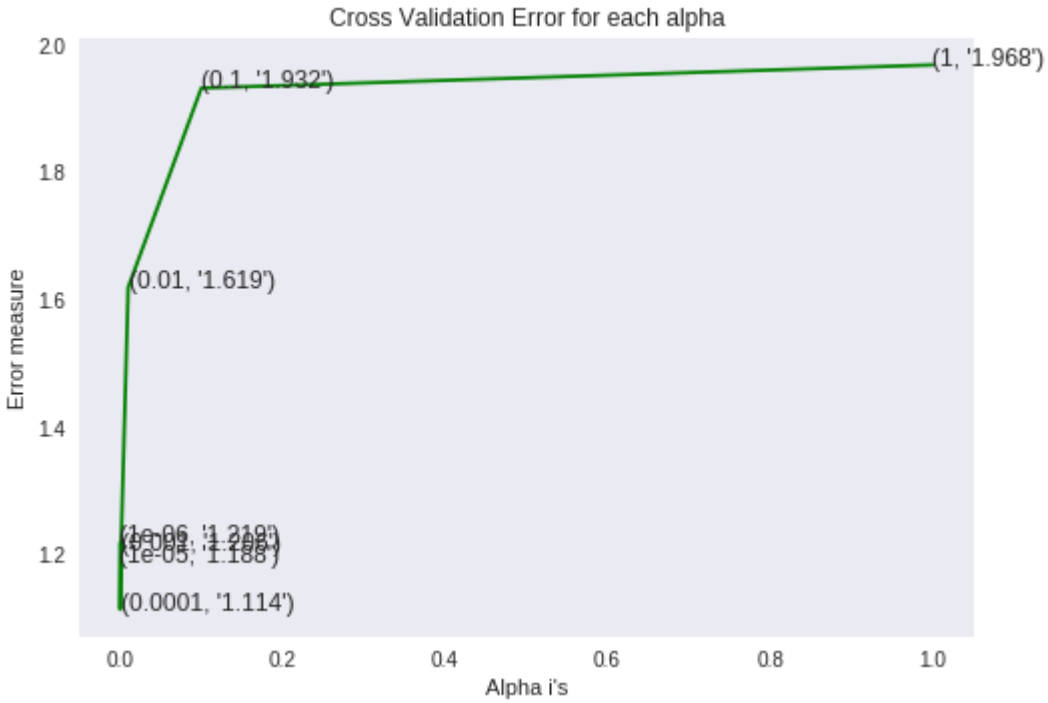
4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

In [183]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/genera
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rat
# 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 Grad
# predict(X)    Predict class labels for samples in X.

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



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

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

# Variables that will be used in the end to make comparison table of all models
lr_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding), labels=
lr_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=clf.class
lr_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding), labels=clf
```

```
for alpha = 1e-06
Log Loss : 1.2185280264738914
for alpha = 1e-05
Log Loss : 1.1877879745882793
for alpha = 0.0001
Log Loss : 1.1142853250344336
for alpha = 0.001
Log Loss : 1.2064854620747771
for alpha = 0.01
Log Loss : 1.6193278189806706
for alpha = 0.1
Log Loss : 1.9318969278194662
for alpha = 1
Log Loss : 1.9682196745558729
```

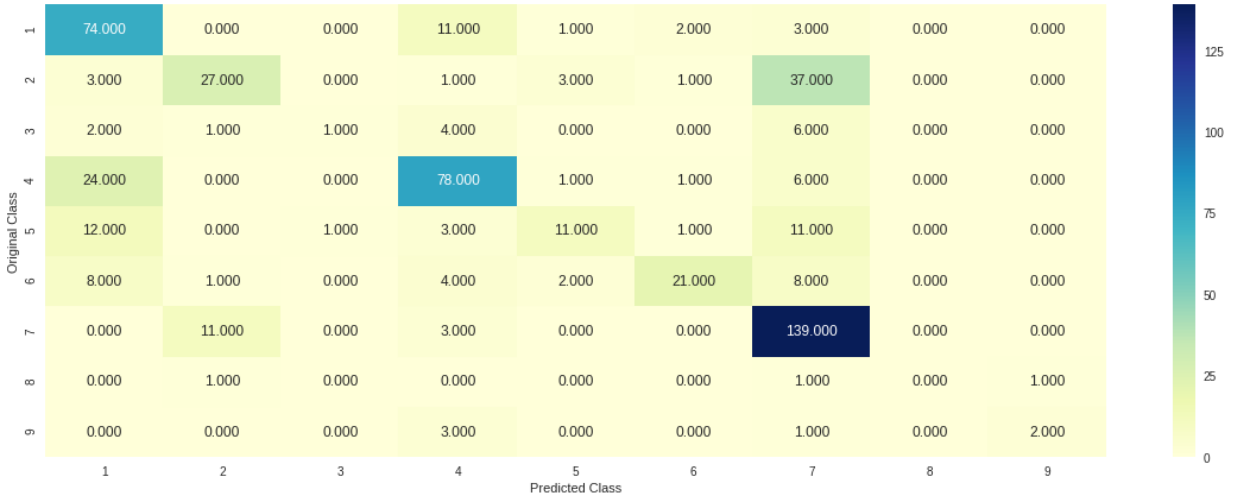



```
For values of best alpha =  0.0001 The train log loss is: 0.41691877939454725
For values of best alpha =  0.0001 The cross validation log loss is: 1.11428532
50344336
For values of best alpha =  0.0001 The test log loss is: 1.0812278554321473
```

4.3.2.2. Testing model with best hyper parameters

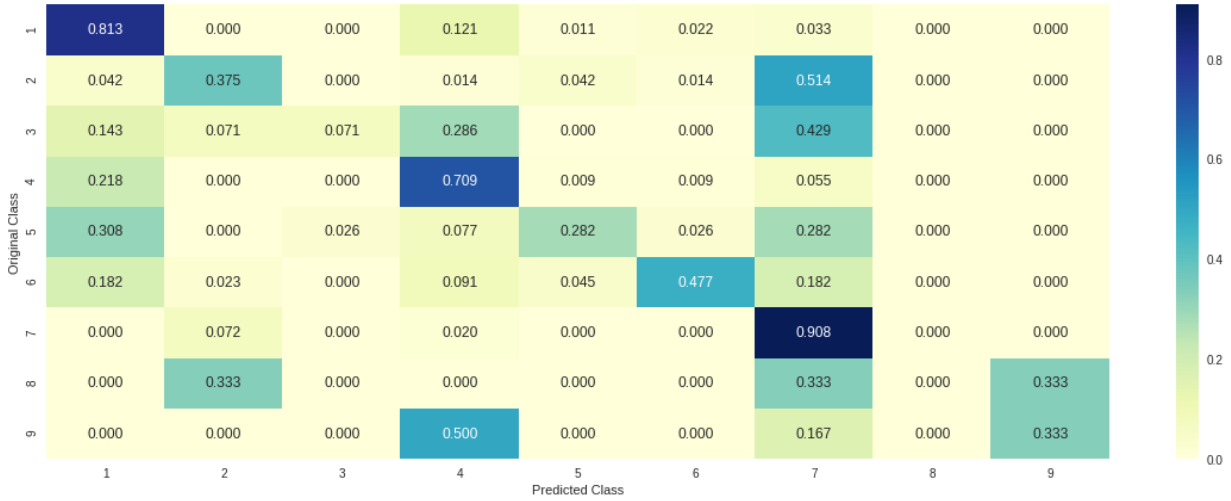
In [184]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/genera
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rat
# 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 Grad
# predict(X)    Predict class labels for samples in X.

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

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_sta
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCodin

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

# Variables that will be used in the end to make comparison table of models
lr_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)- cv_y))/
```

```
Log loss : 1.1142853250344336
Number of mis-classified points : 0.33646616541353386
-------------------- Confusion matrix --------------------
```




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

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



4.3.2.3. Feature Importance, Correctly Classified point

In [185]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_sta
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_one
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['
```

```
Predicted Class : 7
Predicted Class Probabilities: [[3.880e-02 7.880e-02 2.200e-03 1.117e-01 4.450e
-02 1.010e-02 7.108e-01
  2.500e-03 6.000e-04]]
Actual Class : 2
--------------------------------------------------
26 Text feature [enhanced] present in test data point [True]
32 Text feature [activate] present in test data point [True]
33 Text feature [activation] present in test data point [True]
38 Text feature [ligand] present in test data point [True]
50 Text feature [activated] present in test data point [True]
51 Text feature [combination] present in test data point [True]
55 Text feature [downstream] present in test data point [True]
58 Text feature [constitutive] present in test data point [True]
67 Text feature [activating] present in test data point [True]
75 Text feature [inhibitor] present in test data point [True]
89 Text feature [lung] present in test data point [True]
99 Text feature [pathways] present in test data point [True]
118 Text feature [signaling] present in test data point [True]
120 Text feature [approximately] present in test data point [True]
145 Text feature [codon] present in test data point [True]
150 Text feature [oncogene] present in test data point [True]
151 Text feature [derived] present in test data point [True]
169 Text feature [factor] present in test data point [True]
177 Text feature [oncogenic] present in test data point [True]
179 Text feature [sensitive] present in test data point [True]
188 Text feature [3b] present in test data point [True]
209 Text feature [presence] present in test data point [True]
218 Text feature [transformed] present in test data point [True]
249 Text feature [phosphorylated] present in test data point [True]
254 Text feature [medium] present in test data point [True]
265 Text feature [fold] present in test data point [True]
274 Text feature [3t3] present in test data point [True]
276 Text feature [carcinoma] present in test data point [True]
284 Text feature [mechanism] present in test data point [True]
289 Text feature [phospho] present in test data point [True]
293 Text feature [2a] present in test data point [True]
295 Text feature [contrast] present in test data point [True]
302 Text feature [demonstrated] present in test data point [True]
308 Text feature [per] present in test data point [True]
309 Text feature [days] present in test data point [True]
316 Text feature [inhibited] present in test data point [True]
321 Text feature [mechanisms] present in test data point [True]
323 Text feature [promote] present in test data point [True]
```

```
337 Text feature [24] present in test data point [True]
341 Text feature [transforming] present in test data point [True]
349 Text feature [culture] present in test data point [True]
351 Text feature [ph] present in test data point [True]
358 Text feature [bp] present in test data point [True]
362 Text feature [extracellular] present in test data point [True]
370 Text feature [leukemia] present in test data point [True]
379 Text feature [raf] present in test data point [True]
388 Text feature [membrane] present in test data point [True]
390 Text feature [her2] present in test data point [True]
392 Text feature [fig] present in test data point [True]
419 Text feature [molecule] present in test data point [True]
422 Text feature [interface] present in test data point [True]
424 Text feature [volume] present in test data point [True]
425 Text feature [addition] present in test data point [True]
443 Text feature [high] present in test data point [True]
446 Text feature [effective] present in test data point [True]
467 Text feature [endogenous] present in test data point [True]
474 Text feature [expressing] present in test data point [True]
475 Text feature [lead] present in test data point [True]
483 Text feature [gefitinib] present in test data point [True]
485 Text feature [concentrations] present in test data point [True]
489 Text feature [constitutively] present in test data point [True]
494 Text feature [phosphorylation] present in test data point [True]
495 Text feature [examined] present in test data point [True]
Out of the top  500  features  63 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [186]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_one
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['
```

```
Predicted Class : 7
Predicted Class Probabilities: [[1.410e-02 2.330e-01 4.000e-04 3.420e-02 3.90
0e-03 2.200e-02 6.898e-01
  2.300e-03 2.000e-04]]
Actual Class : 7
--------------------------------------------------
26 Text feature [enhanced] present in test data point [True]
33 Text feature [activation] present in test data point [True]
50 Text feature [activated] present in test data point [True]
51 Text feature [combination] present in test data point [True]
55 Text feature [downstream] present in test data point [True]
65 Text feature [s3] present in test data point [True]
67 Text feature [activating] present in test data point [True]
70 Text feature [positive] present in test data point [True]
75 Text feature [inhibitor] present in test data point [True]
89 Text feature [lung] present in test data point [True]
92 Text feature [overexpression] present in test data point [True]
99 Text feature [pathways] present in test data point [True]
118 Text feature [signaling] present in test data point [True]
```

4.4. Linear Support Vector Machines

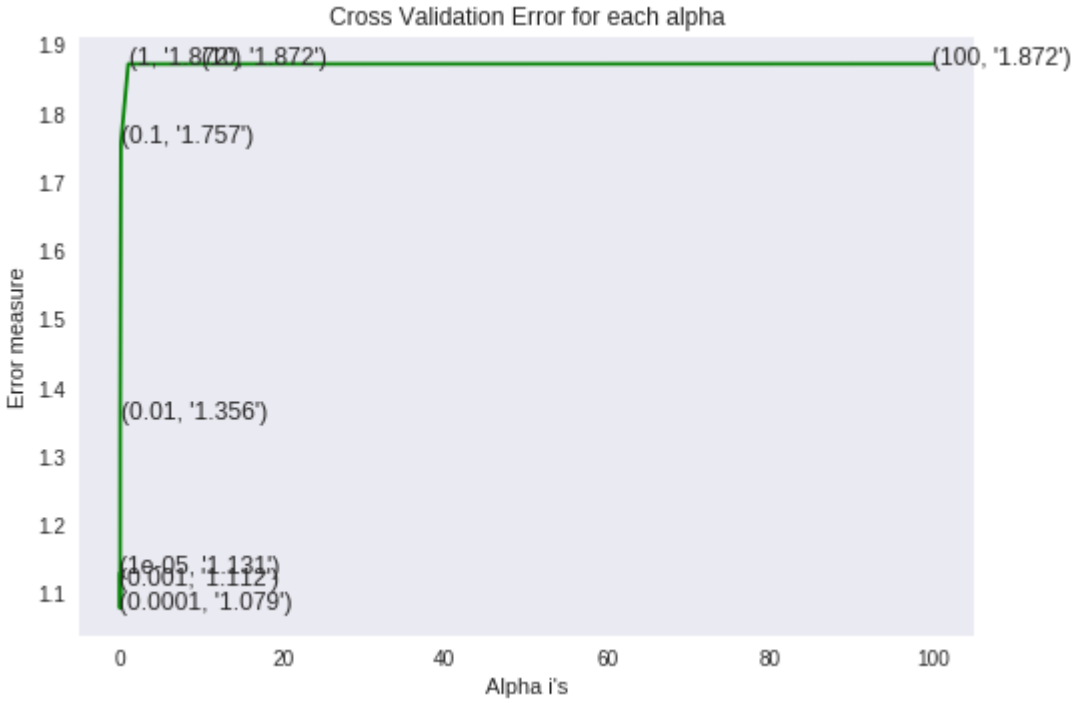
4.4.1. Hyper paramter tuning

In [187]:

```
# read more about support vector machines with linear kernals here http://scikit-

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

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



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


# Variables that will be used in the end to make comparison table of all models
svm_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding), labels
svm_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=clf.clas
svm_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding), labels=cl
```

```
for C = 1e-05
Log Loss : 1.1305600866281276
for C = 0.0001
Log Loss : 1.0793042134510777
for C = 0.001
Log Loss : 1.1123827181732295
for C = 0.01
Log Loss : 1.3562188772397261
for C = 0.1
Log Loss : 1.7573557625542624
for C = 1
Log Loss : 1.872087044601528
for C = 10
Log Loss : 1.872110679129784
for C = 100
Log Loss : 1.872216555878159
```



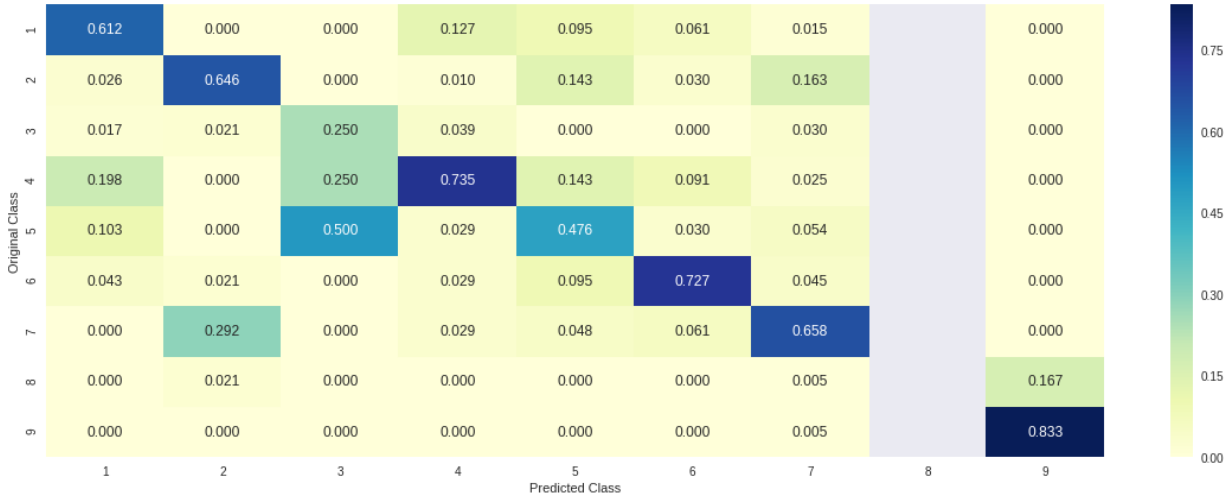
```
For values of best alpha =  0.0001 The train log loss is: 0.46268152220160697
For values of best alpha =  0.0001 The cross validation log loss is: 1.07930421
34510777
For values of best alpha =  0.0001 The test log loss is: 1.0989796436884167
```

4.4.2. Testing model with best hyper parameters

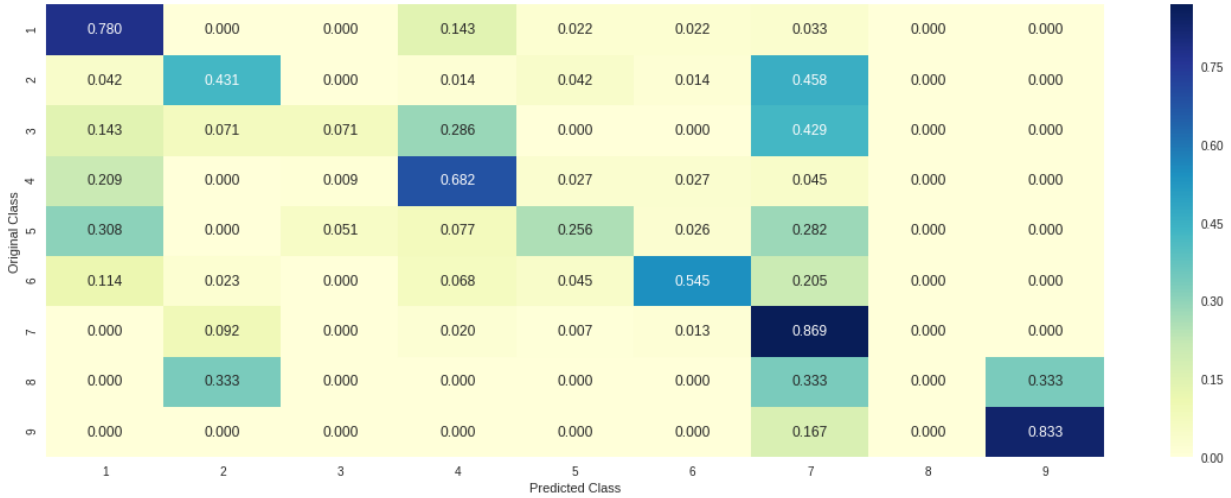
In [188]:
```
# read more about support vector machines with linear kernals here http://scikit-

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

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


# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='b
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_s
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding

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

# Variables that will be used in the end to make comparison table of models
svm_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)- cv_y))
```

```
Log loss : 1.0793042134510777
Number of mis-classified points : 0.34210526315789475
-------------------- 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 [189]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_s
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_one
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['
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1366 0.0653 0.009  0.1091 0.0642 0.0393 0.56
78 0.0035 0.0051]]
Actual Class : 2
--------------------------------------------------
36 Text feature [activate] present in test data point [True]
37 Text feature [downstream] present in test data point [True]
42 Text feature [activation] present in test data point [True]
46 Text feature [enhanced] present in test data point [True]
49 Text feature [inhibitor] present in test data point [True]
214 Text feature [phosphorylated] present in test data point [True]
216 Text feature [codon] present in test data point [True]
217 Text feature [combination] present in test data point [True]
218 Text feature [ligand] present in test data point [True]
221 Text feature [approximately] present in test data point [True]
224 Text feature [activated] present in test data point [True]
226 Text feature [3t3] present in test data point [True]
230 Text feature [derived] present in test data point [True]
231 Text feature [3b] present in test data point [True]
233 Text feature [bp] present in test data point [True]
234 Text feature [activating] present in test data point [True]
236 Text feature [constitutive] present in test data point [True]
237 Text feature [pathways] present in test data point [True]
240 Text feature [ph] present in test data point [True]
246 Text feature [signaling] present in test data point [True]
247 Text feature [lung] present in test data point [True]
249 Text feature [oncogene] present in test data point [True]
250 Text feature [phospho] present in test data point [True]
251 Text feature [mechanisms] present in test data point [True]
253 Text feature [membrane] present in test data point [True]
254 Text feature [demonstrated] present in test data point [True]
255 Text feature [transformed] present in test data point [True]
259 Text feature [2a] present in test data point [True]
272 Text feature [contrast] present in test data point [True]
273 Text feature [her2] present in test data point [True]
280 Text feature [lead] present in test data point [True]
282 Text feature [sensitive] present in test data point [True]
283 Text feature [interface] present in test data point [True]
284 Text feature [24] present in test data point [True]
288 Text feature [position] present in test data point [True]
293 Text feature [75] present in test data point [True]
294 Text feature [fig] present in test data point [True]
296 Text feature [fold] present in test data point [True]
```

```
298 Text feature [medium] present in test data point [True]
299 Text feature [presence] present in test data point [True]
301 Text feature [addition] present in test data point [True]
304 Text feature [effective] present in test data point [True]
305 Text feature [provided] present in test data point [True]
307 Text feature [factor] present in test data point [True]
308 Text feature [mechanism] present in test data point [True]
309 Text feature [culture] present in test data point [True]
312 Text feature [high] present in test data point [True]
314 Text feature [per] present in test data point [True]
315 Text feature [interestingly] present in test data point [True]
316 Text feature [days] present in test data point [True]
319 Text feature [structures] present in test data point [True]
320 Text feature [expressing] present in test data point [True]
321 Text feature [inhibited] present in test data point [True]
322 Text feature [use] present in test data point [True]
323 Text feature [molecule] present in test data point [True]
324 Text feature [gefitinib] present in test data point [True]
325 Text feature [61] present in test data point [True]
326 Text feature [recently] present in test data point [True]
327 Text feature [recurrent] present in test data point [True]
329 Text feature [available] present in test data point [True]
332 Text feature [extracellular] present in test data point [True]
333 Text feature [made] present in test data point [True]
334 Text feature [leukemia] present in test data point [True]
Out of the top  500  features  63 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [190]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_one
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['
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.029  0.4447 0.0024 0.0368 0.0269 0.0234 0.42
92 0.0028 0.0048]]
Actual Class : 7
--------------------------------------------------
89 Text feature [rearrangements] present in test data point [True]
179 Text feature [initial] present in test data point [True]
180 Text feature [rate] present in test data point [True]
188 Text feature [study] present in test data point [True]
189 Text feature [amplified] present in test data point [True]
192 Text feature [median] present in test data point [True]
195 Text feature [events] present in test data point [True]
204 Text feature [invitrogen] present in test data point [True]
205 Text feature [genes] present in test data point [True]
206 Text feature [identification] present in test data point [True]
210 Text feature [plates] present in test data point [True]
211 Text feature [none] present in test data point [True]
215 Text feature [apoptosis] present in test data point [True]
216 Text feature [american] present in test data point [True]
217 Text feature [90] present in test data point [True]
218 Text feature [products] present in test data point [True]
219 Text feature [copy] present in test data point [True]
220 Text feature [response] present in test data point [True]
221 Text feature [indeed] present in test data point [True]
222 Text feature [inhibit] present in test data point [True]
223 Text feature [rates] present in test data point [True]
225 Text feature [molecular] present in test data point [True]
227 Text feature [patients] present in test data point [True]
229 Text feature [major] present in test data point [True]
230 Text feature [transduced] present in test data point [True]
231 Text feature [experimental] present in test data point [True]
232 Text feature [cellular] present in test data point [True]
237 Text feature [clear] present in test data point [True]
238 Text feature [significance] present in test data point [True]
240 Text feature [due] present in test data point [True]
326 Text feature [stimulated] present in test data point [True]
327 Text feature [secondary] present in test data point [True]
328 Text feature [gene] present in test data point [True]
329 Text feature [genome] present in test data point [True]
330 Text feature [overexpression] present in test data point [True]
331 Text feature [across] present in test data point [True]
332 Text feature [complete] present in test data point [True]
333 Text feature [confer] present in test data point [True]
334 Text feature [finding] present in test data point [True]
337 Text feature [another] present in test data point [True]
338 Text feature [established] present in test data point [True]
```

```
340 Text feature [long] present in test data point [True]
341 Text feature [small] present in test data point [True]
342 Text feature [potent] present in test data point [True]
343 Text feature [serum] present in test data point [True]
344 Text feature [novel] present in test data point [True]
345 Text feature [greater] present in test data point [True]
347 Text feature [rather] present in test data point [True]
350 Text feature [12] present in test data point [True]
351 Text feature [presented] present in test data point [True]
352 Text feature [samples] present in test data point [True]
353 Text feature [target] present in test data point [True]
354 Text feature [selection] present in test data point [True]
355 Text feature [number] present in test data point [True]
364 Text feature [syndrome] present in test data point [True]
365 Text feature [pcr] present in test data point [True]
366 Text feature [pocket] present in test data point [True]
368 Text feature [pattern] present in test data point [True]
369 Text feature [tissues] present in test data point [True]
370 Text feature [many] present in test data point [True]
371 Text feature [kinases] present in test data point [True]
372 Text feature [multiple] present in test data point [True]
373 Text feature [clones] present in test data point [True]
375 Text feature [defects] present in test data point [True]
376 Text feature [partial] present in test data point [True]
378 Text feature [normalized] present in test data point [True]
379 Text feature [part] present in test data point [True]
380 Text feature [resistant] present in test data point [True]
384 Text feature [known] present in test data point [True]
385 Text feature [important] present in test data point [True]
386 Text feature [involved] present in test data point [True]
387 Text feature [37] present in test data point [True]
388 Text feature [selected] present in test data point [True]
389 Text feature [exhibited] present in test data point [True]
390 Text feature [led] present in test data point [True]
391 Text feature [normal] present in test data point [True]
392 Text feature [point] present in test data point [True]
396 Text feature [types] present in test data point [True]
397 Text feature [ratio] present in test data point [True]
399 Text feature [revealed] present in test data point [True]
400 Text feature [even] present in test data point [True]
403 Text feature [induce] present in test data point [True]
404 Text feature [100] present in test data point [True]
405 Text feature [conditions] present in test data point [True]
406 Text feature [indicates] present in test data point [True]
407 Text feature [included] present in test data point [True]
409 Text feature [www] present in test data point [True]
410 Text feature [driver] present in test data point [True]
412 Text feature [differentiation] present in test data point [True]
414 Text feature [set] present in test data point [True]
415 Text feature [72] present in test data point [True]
416 Text feature [using] present in test data point [True]
417 Text feature [characterized] present in test data point [True]
418 Text feature [indicating] present in test data point [True]
419 Text feature [recently] present in test data point [True]
425 Text feature [defined] present in test data point [True]
426 Text feature [eight] present in test data point [True]
429 Text feature [d1] present in test data point [True]
```

```
430 Text feature [different] present in test data point [True]
431 Text feature [model] present in test data point [True]
432 Text feature [primary] present in test data point [True]
433 Text feature [finally] present in test data point [True]
434 Text feature [least] present in test data point [True]
435 Text feature [respectively] present in test data point [True]
436 Text feature [various] present in test data point [True]
438 Text feature [highly] present in test data point [True]
439 Text feature [basis] present in test data point [True]
441 Text feature [inhibitor] present in test data point [True]
442 Text feature [70] present in test data point [True]
444 Text feature [oncogenic] present in test data point [True]
447 Text feature [specimens] present in test data point [True]
450 Text feature [enzyme] present in test data point [True]
451 Text feature [studies] present in test data point [True]
453 Text feature [measured] present in test data point [True]
455 Text feature [unknown] present in test data point [True]
456 Text feature [isolated] present in test data point [True]
457 Text feature [based] present in test data point [True]
459 Text feature [active] present in test data point [True]
464 Text feature [therapeutic] present in test data point [True]
465 Text feature [resistance] present in test data point [True]
468 Text feature [hypothesis] present in test data point [True]
469 Text feature [cultured] present in test data point [True]
470 Text feature [used] present in test data point [True]
472 Text feature [control] present in test data point [True]
473 Text feature [18] present in test data point [True]
474 Text feature [cohort] present in test data point [True]
475 Text feature [constructs] present in test data point [True]
477 Text feature [proliferation] present in test data point [True]
478 Text feature [treatment] present in test data point [True]
481 Text feature [harboring] present in test data point [True]
482 Text feature [moreover] present in test data point [True]
483 Text feature [time] present in test data point [True]
484 Text feature [affected] present in test data point [True]
485 Text feature [require] present in test data point [True]
487 Text feature [17] present in test data point [True]
488 Text feature [could] present in test data point [True]
489 Text feature [15] present in test data point [True]
490 Text feature [oncogene] present in test data point [True]
491 Text feature [functionally] present in test data point [True]
493 Text feature [mutational] present in test data point [True]
495 Text feature [targets] present in test data point [True]
496 Text feature [testing] present in test data point [True]
497 Text feature [essential] present in test data point [True]
499 Text feature [95] present in test data point [True]
Out of the top  500  features  144 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [191]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_stat
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training
# 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/les
# --------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/m
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid
#
# 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=
        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.classe
        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_l
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='gi
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross val
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log


#Variables that will be used in the end to make comparison table of all models
rf_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding), labels=
rf_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=clf.class
rf_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding), labels=clf
```

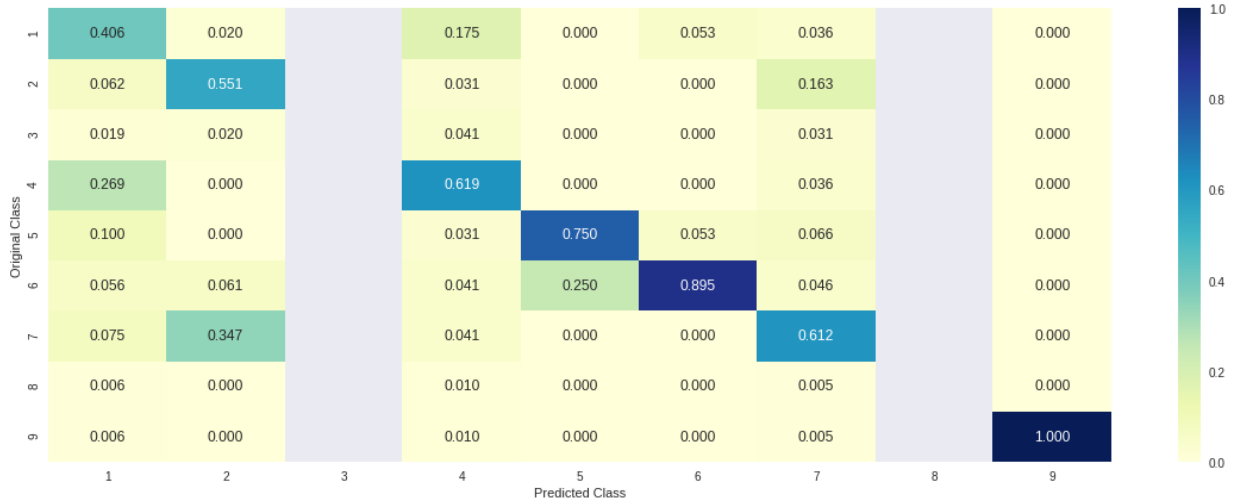
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.257309219630261
for n_estimators = 100 and max depth =  10
Log Loss : 1.2755761815370217
for n_estimators = 200 and max depth =  5
Log Loss : 1.2470926767610415
for n_estimators = 200 and max depth =  10
Log Loss : 1.2615047122976504
for n_estimators = 500 and max depth =  5
Log Loss : 1.234662719305649
for n_estimators = 500 and max depth =  10
Log Loss : 1.2580000774542115
for n_estimators = 1000 and max depth =  5
Log Loss : 1.231668328695989
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2516336485184894
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2320198655209524
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2529726039296833
For values of best estimator =  1000 The train log loss is: 0.8455951994272335
For values of best estimator =  1000 The cross validation log loss is: 1.231668
328695989
For values of best estimator =  1000 The test log loss is: 1.222234497892795
```

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

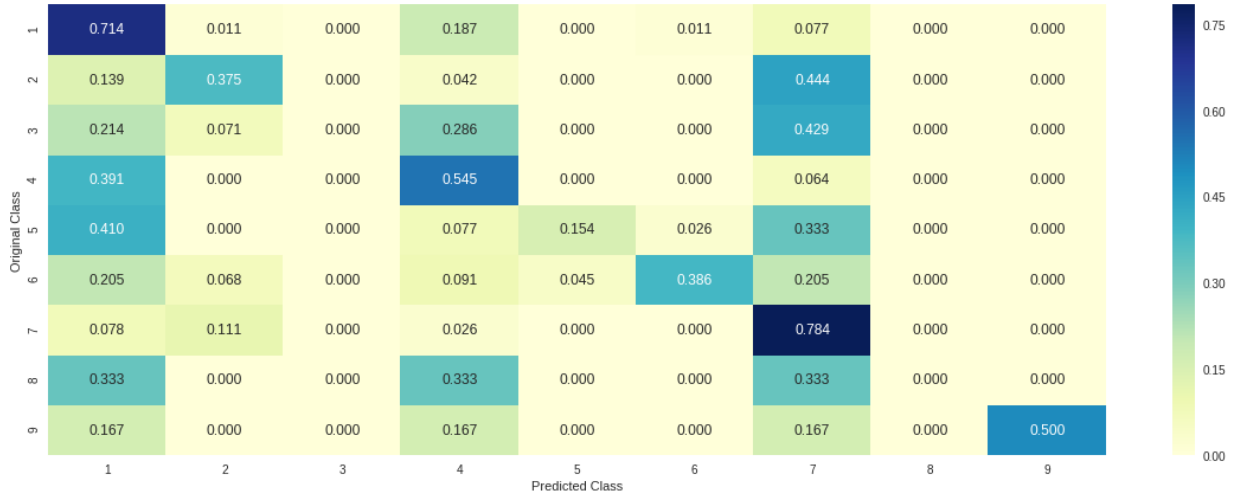
In [192]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_stat
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training
# 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/les
# --------------------------------

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


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

# Variables that will be used in the end to make comparison table of models
rf_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)- cv_y))/
```

```
Log loss : 1.231668328695989
Number of mis-classified points : 0.4398496240601504
-------------------- Confusion matrix --------------------
```


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




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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [193]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gi
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_one
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]
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0297 0.1542 0.0172 0.0262 0.039  0.0326 0.69
23 0.0068 0.0021]]
Actual Class : 2
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [activation] present in test data point [True]
3 Text feature [activated] present in test data point [True]
5 Text feature [tyrosine] present in test data point [True]
6 Text feature [inhibitors] present in test data point [True]
8 Text feature [inhibitor] present in test data point [True]
9 Text feature [missense] present in test data point [True]
10 Text feature [phosphorylation] present in test data point [True]
11 Text feature [treatment] present in test data point [True]
12 Text feature [loss] present in test data point [True]
13 Text feature [constitutive] present in test data point [True]
14 Text feature [oncogenic] present in test data point [True]
16 Text feature [erk] present in test data point [True]
17 Text feature [growth] present in test data point [True]
18 Text feature [protein] present in test data point [True]
21 Text feature [therapeutic] present in test data point [True]
23 Text feature [signaling] present in test data point [True]
24 Text feature [functional] present in test data point [True]
27 Text feature [constitutively] present in test data point [True]
29 Text feature [drug] present in test data point [True]
30 Text feature [treated] present in test data point [True]
31 Text feature [variants] present in test data point [True]
32 Text feature [receptor] present in test data point [True]
38 Text feature [proteins] present in test data point [True]
39 Text feature [cells] present in test data point [True]
43 Text feature [cell] present in test data point [True]
45 Text feature [57] present in test data point [True]
47 Text feature [extracellular] present in test data point [True]
50 Text feature [kinases] present in test data point [True]
51 Text feature [activate] present in test data point [True]
52 Text feature [expression] present in test data point [True]
53 Text feature [inhibition] present in test data point [True]
54 Text feature [ic50] present in test data point [True]
57 Text feature [sensitivity] present in test data point [True]
```

```
59 Text feature [oncogene] present in test data point [True]
60 Text feature [efficacy] present in test data point [True]
61 Text feature [phospho] present in test data point [True]
63 Text feature [transforming] present in test data point [True]
66 Text feature [patients] present in test data point [True]
67 Text feature [variant] present in test data point [True]
69 Text feature [stimulation] present in test data point [True]
72 Text feature [downstream] present in test data point [True]
74 Text feature [dna] present in test data point [True]
75 Text feature [ring] present in test data point [True]
79 Text feature [resistance] present in test data point [True]
81 Text feature [conserved] present in test data point [True]
82 Text feature [3t3] present in test data point [True]
83 Text feature [proportion] present in test data point [True]
84 Text feature [activity] present in test data point [True]
85 Text feature [response] present in test data point [True]
89 Text feature [catalytic] present in test data point [True]
90 Text feature [assays] present in test data point [True]
91 Text feature [clinical] present in test data point [True]
92 Text feature [proliferation] present in test data point [True]
94 Text feature [control] present in test data point [True]
95 Text feature [ligand] present in test data point [True]
96 Text feature [tumors] present in test data point [True]
99 Text feature [expected] present in test data point [True]
Out of the top  100  features  59 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [194]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_one
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]
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1937 0.1677 0.0186 0.1634 0.06   0.0458 0.32
22 0.0129 0.0156]]
Actuall Class : 7
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [activation] present in test data point [True]
3 Text feature [activated] present in test data point [True]
4 Text feature [suppressor] present in test data point [True]
5 Text feature [tyrosine] present in test data point [True]
6 Text feature [inhibitors] present in test data point [True]
7 Text feature [function] present in test data point [True]
8 Text feature [inhibitor] present in test data point [True]
10 Text feature [phosphorylation] present in test data point [True]
11 Text feature [treatment] present in test data point [True]
12 Text feature [loss] present in test data point [True]
14 Text feature [oncogenic] present in test data point [True]
17 Text feature [growth] present in test data point [True]
18 Text feature [protein] present in test data point [True]
21 Text feature [therapeutic] present in test data point [True]
22 Text feature [therapy] present in test data point [True]
23 Text feature [signaling] present in test data point [True]
24 Text feature [functional] present in test data point [True]
29 Text feature [drug] present in test data point [True]
30 Text feature [treated] present in test data point [True]
31 Text feature [variants] present in test data point [True]
32 Text feature [receptor] present in test data point [True]
35 Text feature [yeast] present in test data point [True]
37 Text feature [akt] present in test data point [True]
38 Text feature [proteins] present in test data point [True]
39 Text feature [cells] present in test data point [True]
42 Text feature [classified] present in test data point [True]
43 Text feature [cell] present in test data point [True]
45 Text feature [57] present in test data point [True]
47 Text feature [extracellular] present in test data point [True]
49 Text feature [trials] present in test data point [True]
50 Text feature [kinases] present in test data point [True]
52 Text feature [expression] present in test data point [True]
53 Text feature [inhibition] present in test data point [True]
54 Text feature [ic50] present in test data point [True]
55 Text feature [repair] present in test data point [True]
57 Text feature [sensitivity] present in test data point [True]
59 Text feature [oncogene] present in test data point [True]
60 Text feature [efficacy] present in test data point [True]
63 Text feature [transforming] present in test data point [True]
```

```
66 Text feature [patients] present in test data point [True]
67 Text feature [variant] present in test data point [True]
68 Text feature [functions] present in test data point [True]
69 Text feature [stimulation] present in test data point [True]
72 Text feature [downstream] present in test data point [True]
73 Text feature [inactivation] present in test data point [True]
74 Text feature [dna] present in test data point [True]
78 Text feature [information] present in test data point [True]
79 Text feature [resistance] present in test data point [True]
80 Text feature [nuclear] present in test data point [True]
81 Text feature [conserved] present in test data point [True]
83 Text feature [proportion] present in test data point [True]
84 Text feature [activity] present in test data point [True]
85 Text feature [response] present in test data point [True]
87 Text feature [null] present in test data point [True]
89 Text feature [catalytic] present in test data point [True]
90 Text feature [assays] present in test data point [True]
91 Text feature [clinical] present in test data point [True]
92 Text feature [proliferation] present in test data point [True]
94 Text feature [control] present in test data point [True]
96 Text feature [tumors] present in test data point [True]
Out of the top  100  features  62 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [195]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_stat
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training
# 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/les
# --------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/m
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid
#
# 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=
        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.classe
        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_l
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='gi
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log los
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross validat
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log loss

# Variables that will be used in the end to make comparison table of all models
rf_response_train = log_loss(y_train, sig_clf.predict_proba(train_x_responseCodin
rf_response_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_responseCoding), label
rf_response_test = log_loss(y_test, sig_clf.predict_proba(test_x_responseCoding),
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.137586239511197
for n_estimators = 10 and max depth =  3
Log Loss : 1.823802533574521
for n_estimators = 10 and max depth =  5
Log Loss : 1.404374049469815
for n_estimators = 10 and max depth =  10
Log Loss : 1.9723313902855153
for n_estimators = 50 and max depth =  2
Log Loss : 1.7924652705324478
for n_estimators = 50 and max depth =  3
Log Loss : 1.5081063075123846
for n_estimators = 50 and max depth =  5
Log Loss : 1.3961437507423022
for n_estimators = 50 and max depth =  10
Log Loss : 1.6994038419610424
for n_estimators = 100 and max depth =  2
Log Loss : 1.6473377693955675
for n_estimators = 100 and max depth =  3
Log Loss : 1.524876101494743
for n_estimators = 100 and max depth =  5
Log Loss : 1.3301555857717446
for n_estimators = 100 and max depth =  10
Log Loss : 1.6406405229811214
for n_estimators = 200 and max depth =  2
Log Loss : 1.7424639434716385
for n_estimators = 200 and max depth =  3
Log Loss : 1.5493829978793061
for n_estimators = 200 and max depth =  5
Log Loss : 1.3957988451814929
for n_estimators = 200 and max depth =  10
Log Loss : 1.6713572701609143
for n_estimators = 500 and max depth =  2
```

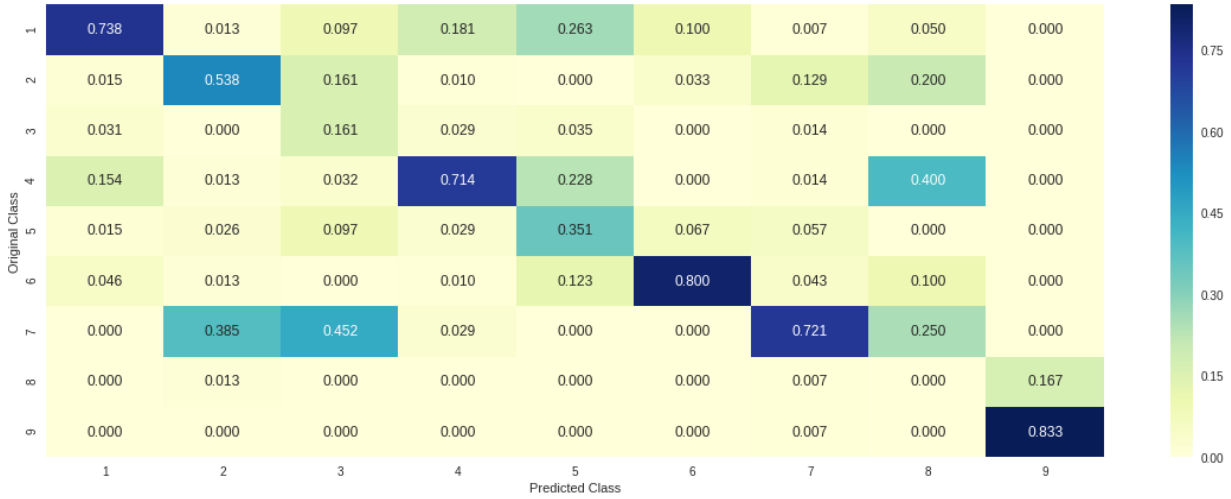
```
Log Loss : 1.777986793226693
for n_estimators = 500 and max depth =  3
Log Loss : 1.6201072768882128
for n_estimators = 500 and max depth =  5
Log Loss : 1.4153623423563004
for n_estimators = 500 and max depth =  10
Log Loss : 1.6880027335542096
for n_estimators = 1000 and max depth =  2
Log Loss : 1.748029631112248
for n_estimators = 1000 and max depth =  3
Log Loss : 1.627213177218563
for n_estimators = 1000 and max depth =  5
Log Loss : 1.4162000622183617
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6392265846047482
For values of best alpha =  100 The train log loss is: 0.05804385125011432
For values of best alpha =  100 The cross validation log loss is: 1.33015558577
17446
For values of best alpha =  100 The test log loss is: 1.2915127144161835
```

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

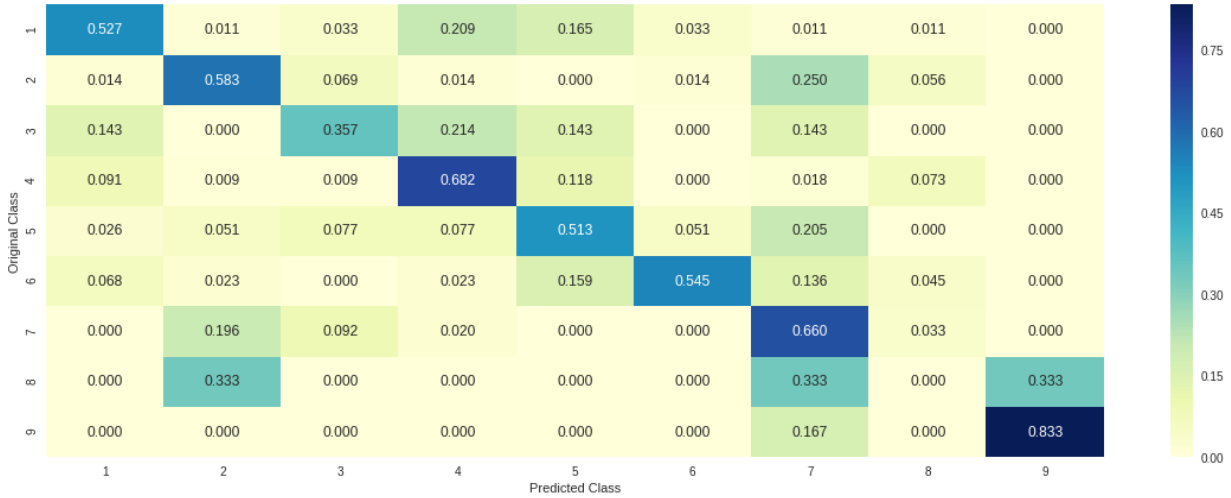
In [196]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_stat
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training
# 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/les
# --------------------------------

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

clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)
# Variables that will be used in the end to make comparison table of models
rf_response_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_responseCodin
```

```
Log loss : 1.3301555857717446
Number of mis-classified points : 0.39849624060150374
-------------------- Confusion matrix --------------------
```


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



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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [197]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gi
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)


test_point_index = 1
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_res
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: [[0.0229 0.1934 0.1705 0.0214 0.0422 0.0482 0.45
69 0.0332 0.0112]]
Actual Class : 2
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text 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
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [198]:
```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_res
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: [[0.0041 0.2513 0.0036 0.0058 0.0022 0.0123 0.71
23 0.0041 0.0042]]
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Text 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
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Variation is important feature
Gene is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [202]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/genera
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_inte
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rat
# 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 Grad
# predict(X)    Predict class labels for samples in X.

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


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

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


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

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training
# 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/les
# --------------------------------

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

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

```
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


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

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

```
Logistic Regression :  Log Loss: 1.11
Support vector machines : Log Loss: 1.87
Naive Bayes : Log Loss: 1.21
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.177
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.031
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.507
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.210
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.456
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.938
```

4.7.2 testing the model with the best hyper parameters

In [204]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classi
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_error1 = log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
print("Log loss (CV) on the stacking classifier :",log_error)

log_error2 = 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_o
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding))

# Variables that will be used in the end to make comparison table of all models
stack_train = log_error
stack_cv = log_error1
stack_test = log_error2
stack_misclassified = (np.count_nonzero((sclf.predict(test_x_onehotCoding)- test_
```

```
Log loss (train) on the stacking classifier : 0.522495785746969
Log loss (CV) on the stacking classifier : 0.522495785746969
Log loss (test) on the stacking classifier : 0.522495785746969
Number of missclassified point : 0.37293233082706767
-------------------- Confusion matrix --------------------
```

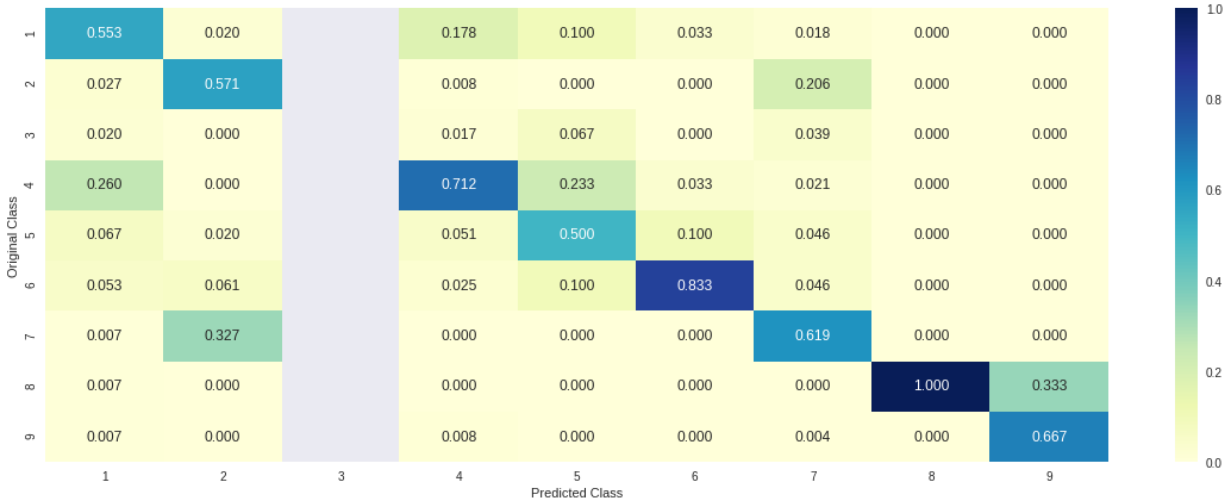

```
-------------------- Precision matrix (Columm Sum=1) --------------------
```

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

In [205]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingCl
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', s
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predic
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_prob
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_o
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))

# Variables that will be used in the end to make comparison table of all models
mvc_train = log_loss(train_y, vclf.predict_proba(train_x_onehotCoding))
mvc_cv = log_loss(cv_y, vclf.predict_proba(cv_x_onehotCoding))
mvc_test = log_loss(test_y, vclf.predict_proba(test_x_onehotCoding))
mvc_misclassified = (np.count_nonzero((vclf.predict(test_x_onehotCoding)- test_y)
```

```
Log loss (train) on the VotingClassifier : 0.8136985249131796
Log loss (CV) on the VotingClassifier : 1.231841420211862
Log loss (test) on the VotingClassifier : 1.2256897228805255
Number of missclassified point : 0.3774436090225564
-------------------- Confusion matrix --------------------
```


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




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


Original Class
Predicted Class
1 0.728 0.009 0.000 0.184 0.026 0.009 0.044 0.000 0.000
2 0.044 0.308 0.000 0.011 0.000 0.000 0.637 0.000 0.000
3 0.167 0.000 0.000 0.111 0.111 0.000 0.611 0.000 0.000
4 0.285 0.000 0.000 0.613 0.051 0.007 0.044 0.000 0.000
5 0.208 0.021 0.000 0.125 0.312 0.062 0.271 0.000 0.000
6 0.145 0.055 0.000 0.055 0.055 0.455 0.236 0.000 0.000
7 0.005 0.084 0.000 0.000 0.000 0.000 0.911 0.000 0.000
8 0.250 0.000 0.000 0.000 0.000 0.000 0.000 0.250 0.500
9 0.143 0.000 0.000 0.143 0.000 0.000 0.143 0.000 0.571
1 2 3 4 5 6 7 8 9
0.8
0.6
0.4
0.2
0.0

In [206]:
```
# Creating table using PrettyTable library
from prettytable import PrettyTable

# Names of models
names =['Naive Bayes','K-Nearest Neighbour','LR With Class Balancing',\
        'LR Without Class Balancing','Linear SVM',\
        'RF With One hot Encoding','RF With Response Coding',\
        'Stacking Classifier','Maximum Voting Classifier']

# Training loss
train_loss = [nb_train,knn_train,lr_balance_train,lr_train,svm_train,rf_train,rf_

# Cross Validation loss
cv_loss = [nb_cv,knn_cv,lr_balance_cv,lr_cv,svm_cv,rf_cv,rf_response_cv,stack_cv,

# Test loss
test_loss = [nb_test,knn_test,lr_balance_test,lr_test,svm_test,rf_test,rf_respons

# Percentage Misclassified points
misclassified = [nb_misclassified,knn_misclassified,lr_balance_misclassified,lr_m
                 rf_misclassified,rf_response_misclassified,stack_misclassified,m

numbering = [1,2,3,4,5,6,7,8,9]

# Initializing prettytable
ptable = PrettyTable()

# Adding columns
ptable.add_column("S.NO.",numbering)
ptable.add_column("MODEL",names)
ptable.add_column("Train_loss",train_loss)
ptable.add_column("CV_loss",cv_loss)
ptable.add_column("Test_loss",test_loss)
ptable.add_column("Misclassified(%)",misclassified)

# Printing the Table
print(ptable)
```

```
+-------+---------------------------+--------------------+------------------
-+--------------------+--------------------+
| S.NO. |           MODEL           |      Train_loss    |      CV_loss
|     Test_loss      |  Misclassified(%)  |
+-------+---------------------------+--------------------+------------------
-+--------------------+--------------------+
|   1   |        Naive Bayes        |  0.5054619475695253 | 1.2134678589675338
| 1.222175891271175  | 39.285714285714285 |
|   2   |    K-Nearest Neighbour    | 0.47911637068502594 | 1.0717436548852675
| 1.0411793983349593 | 34.774436090225564 |
|   3   |  LR With Class Balancing  |  0.4256278197062046 | 1.0809344524198765
| 1.0601944839119568 | 33.08270676691729  |
|   4   | LR Without Class Balancing | 0.41691877939454725 | 1.1142853250344336
| 1.0812278554321473 | 33.64661654135339  |
|   5   |         Linear SVM        | 0.46268152220160697 | 1.0793042134510777
| 1.0989796436884167 | 34.21052631578947  |
|   6   |  RF With One hot Encoding  |  0.8455951994272335 | 1.231668328695989
| 1.222234497892795  | 43.984962406015036 |
```

```
|   7   |  RF With Response Coding   | 0.05804385125011431 | 1.3301555857717446
| 1.2915127144161835 | 39.849624060150376 |
|   8   |    Stacking Classifier     |  0.522495785746969  | 1.210428252413819
| 1.2036080986324276 | 37.29323308270677  |
|   9   | Maximum Voting Classifier  |  0.8136985249131796 | 1.231841420211862
| 1.2256897228805255 | 37.74436090225564  |
+-------+----------------------------+---------------------+------------------
-+--------------------+--------------------+
```