# Personalized cancer diagnosis

## 1. Business Problem

### 1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

***Context:***

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

***Problem statement :***

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

### 1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

### 1.3. Real-world/Business objectives and constraints.

- No low-latency requirement.
- Interpretability is important.

- Errors can be very costly.
- Probability of a data-point belonging to each class is needed.

# 2. Machine Learning Problem Formulation

## 2.1. Data

### 2.1.1. Data Overview

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

### 2.1.2. Example Data Point

***training_variants***

---

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

***training_text***

---

ID,Text
0||Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity revealed. Previous work has shown that CDK10 silencing increases ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The precise mechanisms by which CDK10 modulates ETS2 activity, and more generally the functions of CDK10, remain elusive. Here we demonstrate that CDK10 is a cyclin-dependent kinase by identifying cyclin M as an activating cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR

syndrome, a human developmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malformations. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and in conferring tamoxifen resistance to breast cancer cells. CDK10/cyclin M phosphorylates ETS2 in vitro, and in cells it positively controls ETS2 degradation by the proteasome. ETS2 protein levels are increased in cells derived from a STAR patient, and this increase is attributable to decreased cyclin M levels. Altogether, our results reveal an additional regulatory mechanism for ETS2, which plays key roles in cancer and development. They also shed light on the molecular mechanisms underlying STAR syndrome.Cyclin-dependent kinases (CDKs) play a pivotal role in the control of a number of fundamental cellular processes (1). The human genome contains 21 genes encoding proteins that can be considered as members of the CDK family owing to their sequence similarity with bona fide CDKs, those known to be activated by cyclins (2). Although discovered almost 20 y ago (3, 4), CDK10 remains one of the two CDKs without an identified cyclin partner. This knowledge gap has largely impeded the exploration of its biological functions. CDK10 can act as a positive cell cycle regulator in some cells (5, 6) or as a tumor suppressor in others (7, 8). CDK10 interacts with the ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity through an unknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK pathway, and induces resistance of MCF7 cells to tamoxifen (6). ...

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

## 2.2.1. Type of Machine Learning Problem

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

## 2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

## 2.2.3. Machine Learing Objectives and Constraints

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

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

## 2.3. Train, CV and Test Datasets

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

# 3. Exploratory Data Analysis

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

from mlxtend.classifier import StackingClassifier

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

```
/usr/local/lib/python3.6/dist-packages/sklearn/externals/six.py:31: Deprecation
Warning: The module is deprecated in version 0.21 and will be removed in versio
n 0.23 since we've dropped support for Python 2.7. Please rely on the official
version of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", DeprecationWarning)
```

# 3.1. Reading Data

## 3.1.1. Reading Gene and Variation Data

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code (https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code)

Enter your authorization code:
··········
Mounted at /content/drive
```

In [3]:
```
cd drive/My\ Drive/
```

```
/content/drive/My Drive
```

In [4]:
```
data = pd.read_csv('training/training_variants')
print('Number of data points : ', data.shape[0])
print('Number of features : ', data.shape[1])
print('Features : ', data.columns.values)
data.head()
```

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

Out[4]:

| | 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 [5]:
```
# note the seprator in this file
data_text =pd.read_csv("training/training_text",sep="\|\|",engine="python",names=
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[5]:

| | 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 [6]:
```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Out[6]:
```
True
```

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 [8]:
```
# 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, "secon
```

```
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 : 258.643321 seconds
```

In [13]:
```
# Merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[13]:

| | 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 [10]:
```
result[result.isnull().any(axis=1)]
```

Out[10]:

| | 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['Variati
```

In [12]:
```
result[result['ID']==1109]
```

Out[12]:

| | ID | Gene | Variation | Class | TEXT |
|---|---|---|---|---|---|
| **1109** | 1109 | FANCA | S1088F | 1 | FANCA S1088F |

## 3.1.4. Test, Train and Cross Validation Split

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

In [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
```

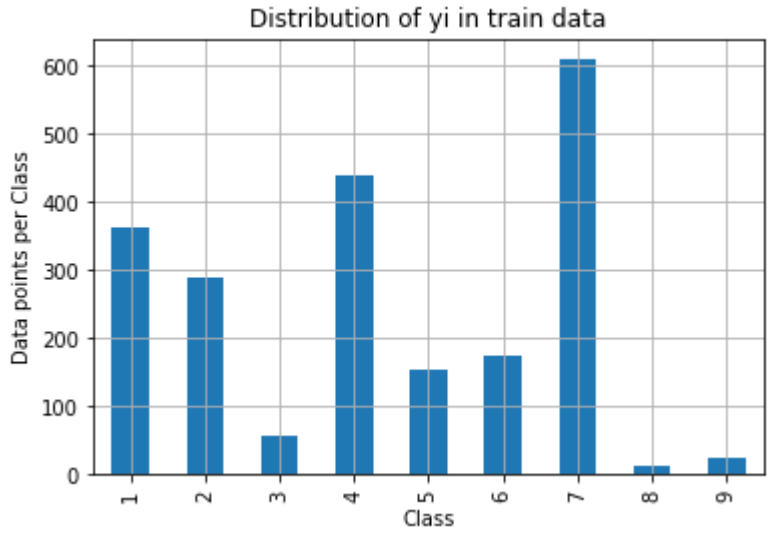
We split the data into train, test and cross validation data sets, preserving the ratio of class distribution in the original data set

In [15]:
```
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])
```

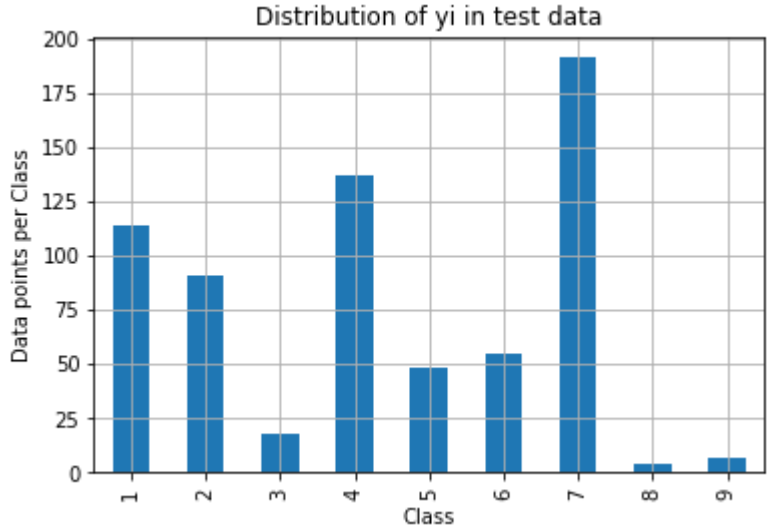
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

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

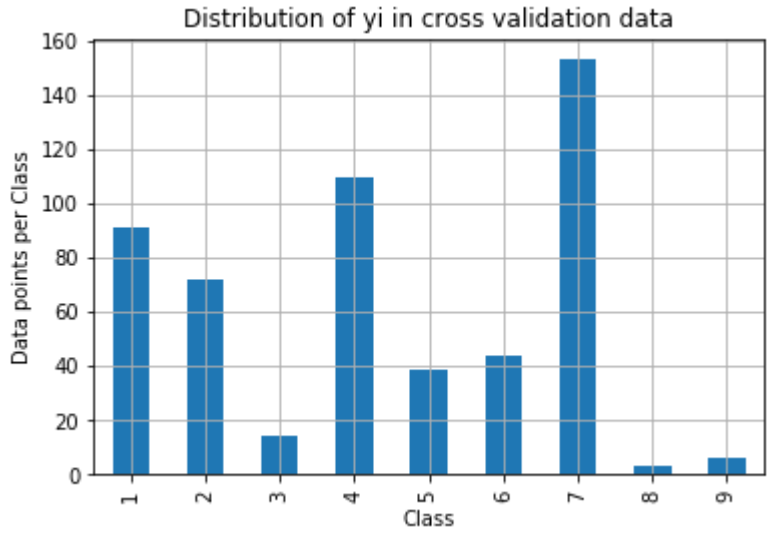
In [16]:
```
# it returns a dict, keys as class labels and values as the number of data points
train_class_distribution = train_df['Class'].value_counts().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

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 %)
```

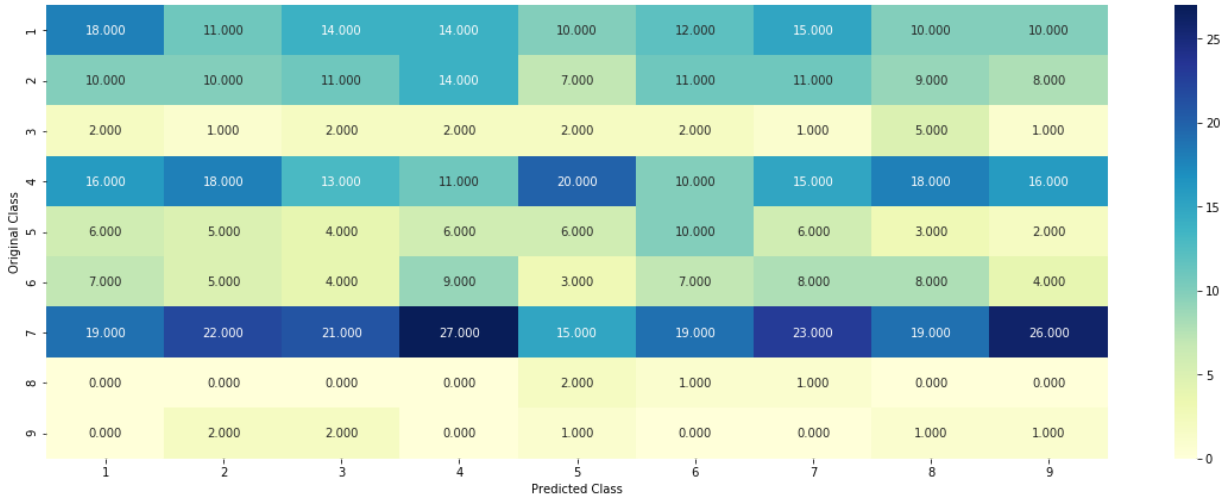
## 3.2 Prediction using a 'Random' Model

In a 'Random' Model, we generate the '9' 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

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

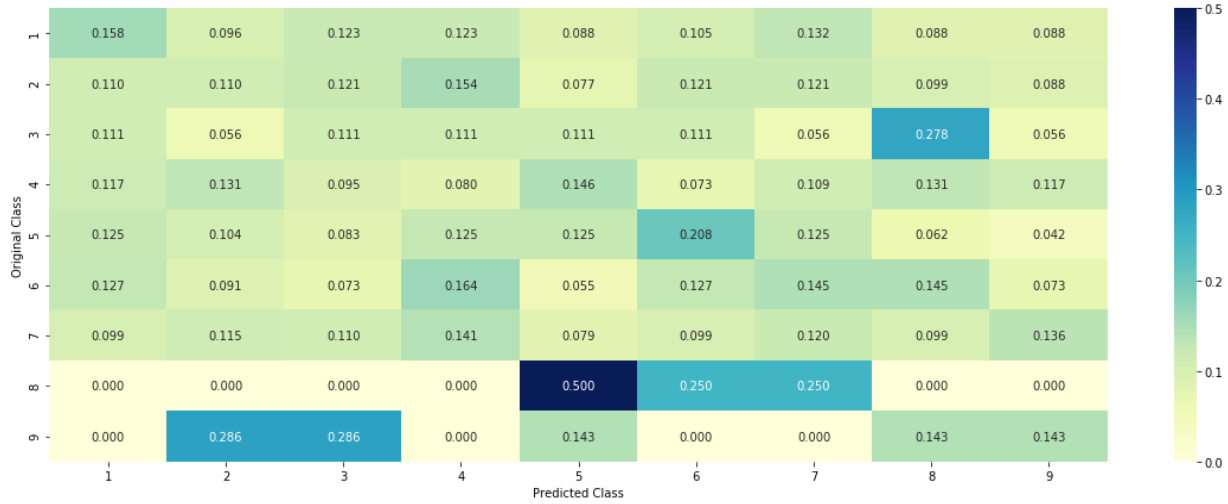
In [18]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their s
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# 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_pr

# 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 2.4656644599289783
Log loss on Test Data using Random Model 2.46456672366406
-------------------- 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']=='BR
            #         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]==

            # 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, 0.068181818181818
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.06122448979591
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.0681818
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.06060606060606
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.06918238993710
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.07284768211920529
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.07333333333333
    #      ...
    #     }
    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 featur
    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?

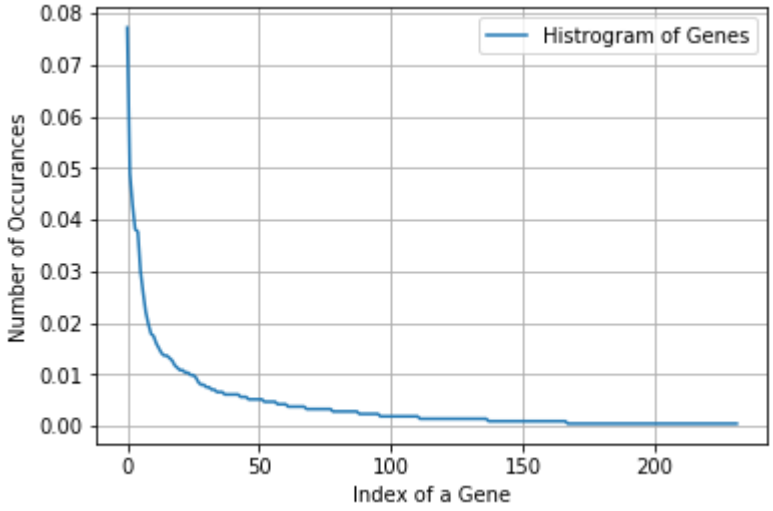
In [20]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 232
BRCA1     164
TP53      104
EGFR       91
BRCA2      81
PTEN       80
KIT        64
BRAF       55
ALK        47
ERBB2      42
PIK3CA     38
Name: Gene, dtype: int64
```

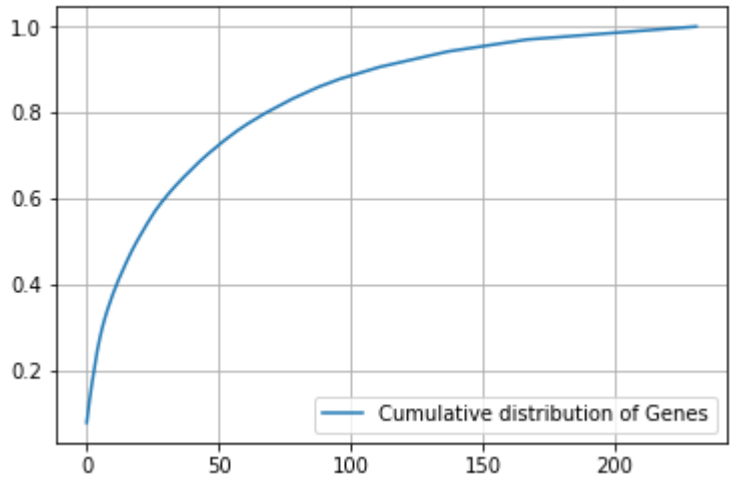
In [21]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in
```

```
Ans: There are 232 different categories of genes in the train data, and they ar
e distibuted as follows
```

In [22]:
```
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 [23]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```




## **Q3.** How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

In [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_d
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
```

In [25]:
```
print("train_gene_feature_responseCoding is converted feature using respone codi
```

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

In [28]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot encodi
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding met
hod. The shape of gene feature: (2124, 231)
```

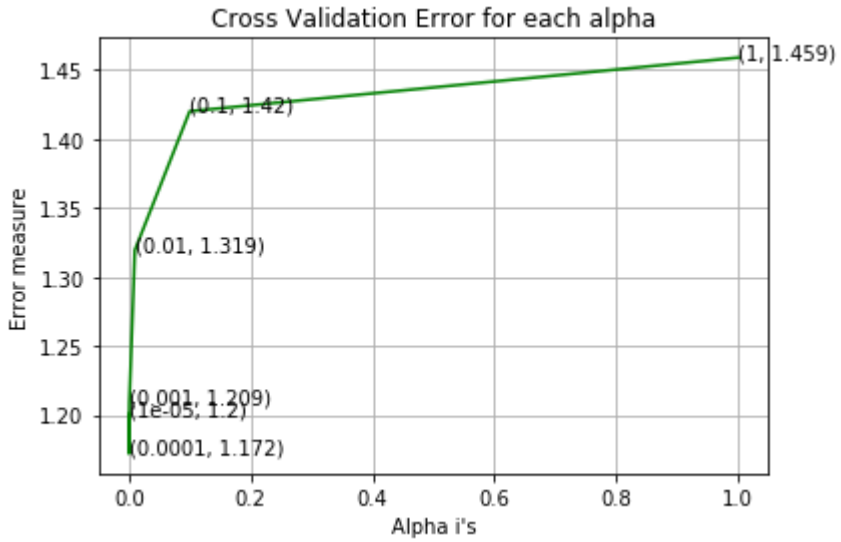
## **Q4.** How good is this gene feature in predicting y_i?

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 [29]:
```
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 Gra
# 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:",
```

```
For values of alpha =  1e-05 The log loss is: 1.200111712853571
For values of alpha =  0.0001 The log loss is: 1.1719965005843778
```

```
For values of alpha =  0.001 The log loss is: 1.2088338956349436
For values of alpha =  0.01 The log loss is: 1.3193471871349913
For values of alpha =  0.1 The log loss is: 1.4199536095481953
For values of alpha =  1 The log loss is: 1.4588250952455677
```




```
For values of best alpha =  0.0001 The train log loss is: 0.9984059775252763
For values of best alpha =  0.0001 The cross validation log loss is: 1.17199650
05843778
For values of best alpha =  0.0001 The test log loss is: 1.1943256866929843
```

## **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 [30]:
```
print("Q6. How many data points in Test and CV datasets are covered by the ", un

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],":" ,(
```

```
Q6. How many data points in Test and CV datasets are covered by the  232  genes
in train dataset?
Ans
1. In test data 645 out of 665 : 96.99248120300751
2. In cross validation data 517 out of  532 : 97.18045112781954
```

### 3.2.2 Univariate Analysis on Variation Feature

**Q7.** Variation, What type of feature is it ?

**Ans.** Variation is a categorical variable

**Q8.** How many categories are there?

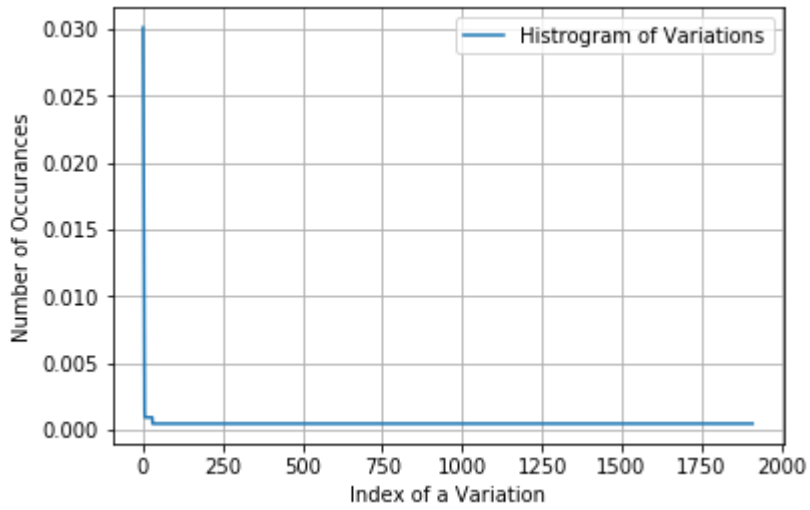
In [32]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1911
Truncating_Mutations    64
Deletion                51
Amplification           47
Fusions                 25
Overexpression           5
T58I                     3
Q61L                     2
F384L                    2
K117N                    2
C618R                    2
Name: Variation, dtype: int64
```

In [33]:
```
print("Ans: There are", unique_variations.shape[0] ,
      "different categories of variations in the train data, and they are distibu
```

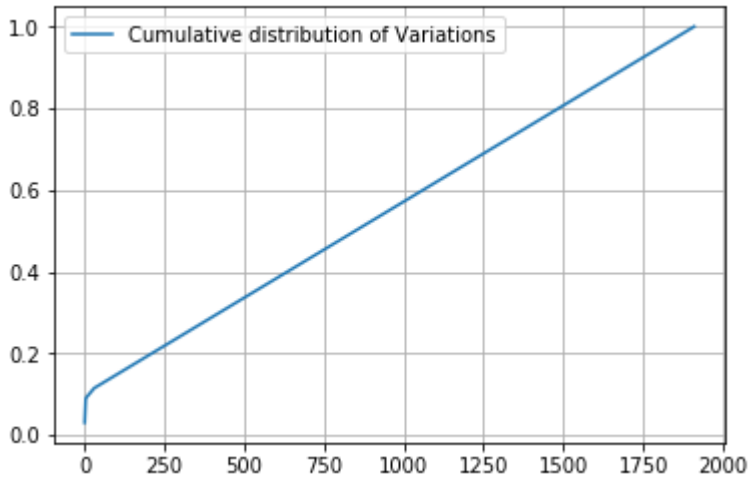
```
Ans: There are 1911 different categories of variations in the train data, and t
hey are distibuted as follows
```

In [34]:
```
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 [35]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.03013183 0.05414313 0.07627119 ... 0.99905838 0.99952919 1.        ]
```




## **Q9.** How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

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

In [37]:
```
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 = CountVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Va
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variat
```

In [39]:
```
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, 1934)
```

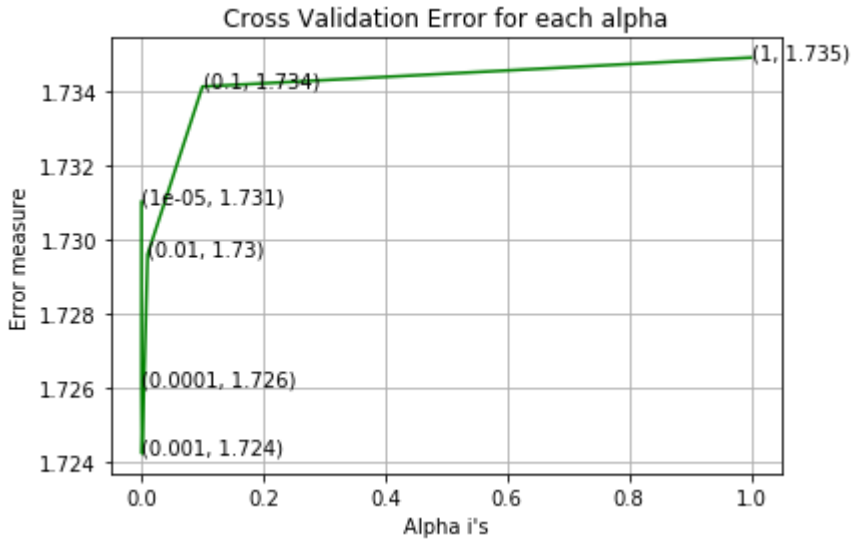
## **Q10.** How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

In [40]:
```
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 Gra
# 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:",
```

```
For values of alpha =  1e-05 The log loss is: 1.731030818324406
```

```
For values of alpha =  0.0001 The log loss is: 1.7261073625601082
For values of alpha =  0.001 The log loss is: 1.724231306907978
For values of alpha =  0.01 The log loss is: 1.729608329827887
For values of alpha =  0.1 The log loss is: 1.7341390479904928
For values of alpha =  1 The log loss is: 1.7349209643158414
```




```
For values of best alpha =  0.001 The train log loss is: 1.04314135900955
For values of best alpha =  0.001 The cross validation log loss is: 1.724231306
907978
For values of best alpha =  0.001 The test log loss is: 1.7376946858265268
```

**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 [42]:
```
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']))))].sha
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],":" ,(
```

```
Q12. How many data points are covered by total  1911  genes in test and cross v
alidation data sets?
Ans
1. In test data 61 out of 665 : 9.172932330827068
2. In cross validation data 47 out of  532 : 8.834586466165414
```

### 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 str(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 str(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(str
            row_index += 1
    return text_feature_responseCoding
```

In [0]:
```
def top_features(X, feature_names, top_n=1000):
    D_ = X.toarray()
    D_[D_ < 0.1] = 0
    tfidf_ = np.mean(D_, axis=0)
    top_values = np.argsort(tfidf_)[::-1][:top_n]
    top_features = [(feature_names[i], tfidf_[i]) for i in top_values]
    df = pd.DataFrame(top_features)
    df.columns = ['feature', 'tfidf']
    return df
```

In [0]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in t
text_vectorizer = TfidfVectorizer(min_df=3)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT']

# getting top 1000 feature names (words)
train_text_features = top_features(train_text_feature_onehotCoding,text_vectorize
```

In [54]:
```
# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1
train_text_fea_counts
```

Out[54]:
```
array([8.07708176, 8.7038545 , 0.03644943, ..., 0.05182414, 0.03734922,
       0.05893848])
```

In [55]:
```
# zip(list(text_features),text_fea_counts) will zip a word with its number of ti
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 [64]:
```
train_text_feature_responseCoding.shape
```

Out[64]:
```
(2124, 9)
```

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_t
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_featu
```

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

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

# we use the same vectorizer that was trained on train data
cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'].astype('U
# 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] , r
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

In [69]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({0.02092756925753886: 29, 0.05681876907915845: 16, 0.03323433075541987:
9, 0.027881103137322893: 9, 0.30318923920890434: 7, 0.01978757262757599: 7, 0.0
14697174967263527: 7, 0.22547853257422737: 6, 0.019733848228254896: 6, 0.066468
66151083974: 4, 0.010447320695484985: 4, 0.07122082778844357: 3, 0.031652714588
7315: 3, 0.02865839248207269: 3, 0.02581376679517794: 3, 0.022060361393053264:
3, 0.020154921262204285: 3, 0.01527744176020004: 3, 0.014520240441861182: 3, 0.
010879811275748642: 3, 0.14983304950933246: 2, 0.0997029922662596: 2, 0.0713835
5752594606: 2, 0.06430910921901416: 2, 0.058132131715844: 2, 0.058042699097884
5: 2, 0.05576220627464579: 2, 0.047978962541363177: 2, 0.04356072132558354: 2,
0.040125074778875294: 2, 0.03749145965057497: 2, 0.03729369118955814: 2, 0.0336
6052480000138: 2, 0.0260570990966432: 2, 0.024404044050942954: 2, 0.02363271005
170159: 2, 0.023040818061998158: 2, 0.02240878539108351: 2, 0.02211138389915684
5: 2, 0.02152890019946404: 2, 0.018089079338907545: 2, 0.01584035308086458: 2,
0.012349849614293341: 2, 0.010581631769988675: 2, 0.007990594702733432: 2, 0.00
6440110481197183: 2, 0.00592517806787597: 2, 32.56493845149944: 1, 18.696378759
703844: 1, 17.897771808301748: 1, 14.721375843285887: 1, 12.780923906126567: 1,
8.703854497667791: 1, 8.07708175733795: 1, 6.413379529493093: 1, 5.410160222706
21: 1, 4.9528000866394155: 1, 4.9326744190279825: 1, 3.7563362280748906: 1, 3.7
095599538795585: 1, 3.6814550690062524: 1, 3.5507835615007406: 1, 3.52257812981
11856: 1, 3.5068776218955287: 1, 3.3700042686002947: 1, 3.347190513439555: 1,
3.255794528522321: 1, 2.9900109636898953: 1, 2.550227866382949: 1, 2.5115689100
881573: 1, 2.4802638583165: 1, 2.4344001741123615: 1, 2.3970622991200607: 1, 2.
337930477190061: 1, 2.327422451720292: 1, 2.0359528575800403: 1, 2.014181609723
5226: 1, 2.0041669189295646: 1, 1.922096264391664: 1, 1.8706261266100475: 1, 1.
7781581440492251: 1, 1.7437795744422757: 1, 1.7289675715910875: 1, 1.6031258033
435527: 1, 1.554960211821221: 1, 1.4611577897677936: 1, 1.3732395139913594: 1,
1.3287778198152067: 1, 1.3184040873594594: 1, 1.2951004843490055: 1, 1.29269587
5999052: 1, 1.228591197817425: 1, 1.1843207536124911: 1, 1.1657489849266132: 1,
1.1582439672194884: 1, 1.0369062095580373: 1, 1.014390284356823: 1, 1.011228527
7366229: 1, 0.9988283960417164: 1, 0.993597419241028: 1, 0.9382732235594159: 1,
0.9348244888608884: 1, 0.9230329053059608: 1, 0.9056019986006146: 1, 0.88108849
98012589: 1, 0.8761642403411726: 1, 0.8565818318311471: 1, 0.8564408058083164:
1, 0.8159845921732578: 1, 0.791135829630542: 1, 0.7695466927588206: 1, 0.761543
4830376102: 1, 0.7434589395395239: 1, 0.7249114161295634: 1, 0.716279348594697
1: 1, 0.7104116956449038: 1, 0.6965710884831778: 1, 0.6610732369057742: 1, 0.64
36203317913454: 1, 0.6354349363515752: 1, 0.6168805456344261: 1, 0.614827879230
6024: 1, 0.6065815323961143: 1, 0.599573930741384: 1, 0.59833220577763: 1, 0.59
4571527134534: 1, 0.5874556743001969: 1, 0.5393697158621633: 1, 0.5370950588390
706: 1, 0.498569857806335: 1, 0.48034000815414335: 1, 0.47682330832846187: 1,
0.47595511874514884: 1, 0.47375123961725646: 1, 0.4733199133747597: 1, 0.472633
24732410816: 1, 0.45393879739786275: 1, 0.44714988489604934: 1, 0.4188556875016
223: 1, 0.41787375472749644: 1, 0.41165873237780093: 1, 0.4106800695454297: 1,
0.41008984580016494: 1, 0.40764453544134055: 1, 0.40416682198507864: 1, 0.40302
1047956442: 1, 0.39987941945595357: 1, 0.39882837840881613: 1, 0.38313709352274
383: 1, 0.3828272528683193: 1, 0.38193387036918836: 1, 0.38142684147984574: 1,
0.3788626115129242: 1, 0.3724771183299282: 1, 0.3696377317886478: 1, 0.36846304
63800166: 1, 0.36634672166153426: 1, 0.36622908344423155: 1, 0.365432484268196
4: 1, 0.36435267979333974: 1, 0.3630663918159289: 1, 0.354706669452431: 1, 0.35
349089918055127: 1, 0.3506434075884525: 1, 0.345178891405181: 1, 0.341573014845
1192: 1, 0.3412158108498965: 1, 0.33975191638733665: 1, 0.3387422882563616: 1,
0.33733683861615965: 1, 0.3367038500580500 4: 1, 0.335255011620289: 1, 0.3340583
155087081: 1, 0.3272286422564516: 1, 0.325653523842549: 1, 0.32527439218553345:
1, 0.32477772538188476: 1, 0.3241601317500778: 1, 0.3228265072581931: 1, 0.3220
```

```
794259144067: 1, 0.3190342814175288: 1, 0.31474879425068514: 1, 0.3146662680313
7504: 1, 0.3124765969535257: 1, 0.31215382631362026: 1, 0.31156288833590756: 1,
0.31110086422766214: 1, 0.3100511925855666: 1, 0.3089528338712016: 1, 0.3056692
3728925754: 1, 0.3018994987442658: 1, 0.29813363064374093: 1, 0.295942684190016
57: 1, 0.29339623680029436: 1, 0.28647127818663043: 1, 0.28424590276734263: 1,
0.28205581605001395: 1, 0.28018563801084934: 1, 0.27978571232325716: 1, 0.27429
639303082815: 1, 0.27010925012520604: 1, 0.2701046009642913: 1, 0.2690461423658
9736: 1, 0.2674456914083372: 1, 0.2661443132431684: 1, 0.2632672577666817: 1,
0.2625581680604977: 1, 0.26126653916092624: 1, 0.25918726587449614: 1, 0.251760
74629858136: 1, 0.24543284825157918: 1, 0.24476923702511497: 1, 0.2428332107520
253: 1, 0.24256875016756865: 1, 0.2418121297277197: 1, 0.2377276799275092: 1,
0.23692770066459332: 1, 0.23669806994101036: 1, 0.23660647414177763: 1, 0.23408
910855869444: 1, 0.23400682845346063: 1, 0.23374966634736363: 1, 0.233299531386
59482: 1, 0.23260837739473844: 1, 0.23086692848778806: 1, 0.23042806442903863:
1, 0.229171536567271: 1, 0.22661198292293985: 1, 0.22642257678979283: 1, 0.2262
9534520506747: 1, 0.22604150445596283: 1, 0.22460517932048835: 1, 0.22383668695
59041: 1, 0.221452109013266: 1, 0.21885289540071307: 1, 0.21749321781555025: 1,
0.21707233051080385: 1, 0.21550766338180974: 1, 0.21439028599717413: 1, 0.21363
366237638615: 1, 0.21186622776326725: 1, 0.20973027445081271: 1, 0.209603732858
4011: 1, 0.20775414980244095: 1, 0.20754771831440882: 1, 0.20638120397672088:
1, 0.20590825239550015: 1, 0.2046178294658139: 1, 0.20371972719846868: 1, 0.199
74776067160074: 1, 0.19938202275215883: 1, 0.19842842860064014: 1, 0.1973846662
8517706: 1, 0.19575291959302957: 1, 0.19512612671692312: 1, 0.1949340654146114
7: 1, 0.19156495541965926: 1, 0.19015890977176989: 1, 0.18885185584218794: 1,
0.18571365678766813: 1, 0.18532516175801964: 1, 0.1836656180412497: 1, 0.183611
71512051785: 1, 0.18055411414003522: 1, 0.17917236261683656: 1, 0.1778544188303
064: 1, 0.1765674578926294: 1, 0.1746127905553978: 1, 0.17240506998834498: 1,
0.17122433981289556: 1, 0.17111273407369496: 1, 0.17098311594906665: 1, 0.17047
424718824736: 1, 0.16728661882393736: 1, 0.16698727969688792: 1, 0.166696791834
11432: 1, 0.1663805140347526: 1, 0.16277592770433202: 1, 0.15866891190623836:
1, 0.1586571145907574: 1, 0.15843129762819366: 1, 0.15842194191951167: 1, 0.155
55948979932116: 1, 0.15414098204293247: 1, 0.1541049387463455: 1, 0.15264257205
25777: 1, 0.1492427805368489: 1, 0.14898006730465954: 1, 0.14858270100894025:
1, 0.14722601357288284: 1, 0.14692171180021094: 1, 0.14619676076114343: 1, 0.14
603655784553296: 1, 0.1453541453607725: 1, 0.14489092189435127: 1, 0.1430280545
03709: 1, 0.1427909490605252: 1, 0.14244802901726264: 1, 0.1414906876852805: 1,
0.14139677579687887: 1, 0.1396542941169127: 1, 0.13919096479529505: 1, 0.138447
88349497072: 1, 0.13831931945010095: 1, 0.13826697853620962: 1, 0.1364976275562
1236: 1, 0.13636920358372354: 1, 0.13533550831828228: 1, 0.13477045670517468:
1, 0.1346138656249388: 1, 0.13447441997272053: 1, 0.13430638169631112: 1, 0.133
33973609944458: 1, 0.1311620016341301: 1, 0.1310794719168834: 1, 0.130079663554
35222: 1, 0.1300550409662742: 1, 0.12986141975747448: 1, 0.1280672719613749: 1,
0.12766644245865472: 1, 0.1275477842074608: 1, 0.1271026503425065: 1, 0.1269971
294371338: 1, 0.12697221118385282: 1, 0.12487506682828269: 1, 0.124597048371915
61: 1, 0.12363479229595996: 1, 0.12353924191357336: 1, 0.12266067735096599: 1,
0.12111424916789953: 1, 0.12079160749732804: 1, 0.12013587897437851: 1, 0.11951
927263676446: 1, 0.11921623855181208: 1, 0.11866354027250897: 1, 0.117690656927
22035: 1, 0.11766573793689013: 1, 0.11699132086865544: 1, 0.11657062686879932:
1, 0.11615775146589682: 1, 0.1160543362728227: 1, 0.11573205128193532: 1, 0.115
15906134049753: 1, 0.11515610317938445: 1, 0.11465677369672009: 1, 0.1146226746
7489673: 1, 0.11424954974834278: 1, 0.11419003798971286: 1, 0.1136299560006783
9: 1, 0.11329112140142872: 1, 0.1126279477631634: 1, 0.11246978860626973: 1, 0.
11221296856850567: 1, 0.11148423524126985: 1, 0.11139969367468369: 1, 0.1108792
13825502: 1, 0.11080341999873183: 1, 0.11052102290754795: 1, 0.1100203996626973
5: 1, 0.10958306412165217: 1, 0.10957397066170795: 1, 0.10915537606571651: 1,
0.10869285897875952: 1, 0.1077910286820545: 1, 0.1074903565782662: 1, 0.1068164
2251764265: 1, 0.10638937185177919: 1, 0.10507340195017782: 1, 0.10504231286212
```

685: 1, 0.10425451553329129: 1, 0.10329113859047721: 1, 0.10320530684526456: 1,
0.1025960264426856: 1, 0.101602863640826: 1, 0.10150002897580827: 1, 0.10127769
685087575: 1, 0.10088251874343339: 1, 0.10088018793631434: 1, 0.100517637511746
32: 1, 0.09958934848342119: 1, 0.09930871906192618: 1, 0.09841628739687502: 1,
0.09834746021779982: 1, 0.09806016849903523: 1, 0.09699986571059303: 1, 0.09664
087870350972: 1, 0.09635825838333832: 1, 0.09602128991464859: 1, 0.095902815760
5569: 1, 0.09585588771341205: 1, 0.0957203668885998: 1, 0.0950771326508579: 1,
0.0944370793535648: 1, 0.09438238035238239: 1, 0.09347082968836952: 1, 0.093277
73500816562: 1, 0.09287950564038663: 1, 0.09220312972248954: 1, 0.0911028825555
4239: 1, 0.09072900469484668: 1, 0.09059785486386195: 1, 0.09053911706289301:
1, 0.09029948110708706: 1, 0.09027052475295945: 1, 0.09011934461911099: 1, 0.08
996306642347122: 1, 0.08990172556644214: 1, 0.08930374680552036: 1, 0.088907043
91705959: 1, 0.08887505603302542: 1, 0.0878424018055827: 1, 0.0869793729323133
8: 1, 0.08697002967433859: 1, 0.08681065840652767: 1, 0.08658126542806578: 1,
0.08571366508550693: 1, 0.08560610592642576: 1, 0.08549762446622389: 1, 0.08546
074595334043: 1, 0.08528429481456795: 1, 0.0840148362294437: 1, 0.0832484074645
843: 1, 0.08284099117376575: 1, 0.08243406320680396: 1, 0.08203919821974882: 1,
0.08190675389857555: 1, 0.08179016200095737: 1, 0.08104366162079718: 1, 0.08026
624620989722: 1, 0.0802365628432816: 1, 0.07993328689979431: 1, 0.0796628079172
3884: 1, 0.07962154440738856: 1, 0.07895098478242454: 1, 0.0786046585993378: 1,
0.07850403923840886: 1, 0.07812600249925264: 1, 0.07792704419789445: 1, 0.07783
972325195285: 1, 0.07739996231130758: 1, 0.07710353975398183: 1, 0.076924815250
60614: 1, 0.07649493385209971: 1, 0.07646609306367372: 1, 0.07608356472489054:
1, 0.07579909940915913: 1, 0.07557064374443105: 1, 0.07507020400960675: 1, 0.07
377217889826305: 1, 0.0736849584999768: 1, 0.07367478635281255: 1, 0.0732589089
6924733: 1, 0.07320820452320322: 1, 0.07296537606194438: 1, 0.0729590411352821:
1, 0.07293529962619606: 1, 0.07234203357857848: 1, 0.0715331720336738: 1, 0.071
40517563061066: 1, 0.07127412185584688: 1, 0.07126858508233055: 1, 0.0711993048
6916028: 1, 0.0711887039334828: 1, 0.07112853060463295: 1, 0.07038534857140083:
1, 0.07015981020320007: 1, 0.06994344287800754: 1, 0.06982719027168498: 1, 0.06
947251243598401: 1, 0.06855423551222016: 1, 0.06823251671894237: 1, 0.068083585
63705015: 1, 0.06795344068459915: 1, 0.06792339698039414: 1, 0.0678766081882892
1: 1, 0.06778230158083175: 1, 0.06777418887338527: 1, 0.06738522835258734: 1,
0.06648510926813388: 1, 0.06635966766784057: 1, 0.06632995810597553: 1, 0.06594
88080848972: 1, 0.06578485774187776: 1, 0.06550249150446658: 1, 0.0654825459977
8096: 1, 0.06532604520497454: 1, 0.06518323580215706: 1, 0.0650898756368086: 1,
0.0650754763806346: 1, 0.06463499969059464: 1, 0.06444409642153677: 1, 0.064377
41437617364: 1, 0.06336122683680716: 1, 0.063305429177463: 1, 0.063034552853816
4: 1, 0.0626042292287869: 1, 0.06255303492928614: 1, 0.06253234041978822: 1, 0.
06239760276831396: 1, 0.062162567505087925: 1, 0.061858303190245094: 1, 0.06167
1280609126425: 1, 0.061363480146443605: 1, 0.06135814124277102: 1, 0.0612224749
9409882: 1, 0.061142769688023554: 1, 0.061139489587394166: 1, 0.060892673046969
12: 1, 0.06003060491756187: 1, 0.059907610985474084: 1, 0.05990227766957074: 1,
0.05975425652439829: 1, 0.05940178437386514: 1, 0.05936271788272797: 1, 0.05908
7043361274585: 1, 0.0587969868856933: 1, 0.058744959880079176: 1, 0.05856623455
4216626: 1, 0.058364101877994595: 1, 0.05747461129836276: 1, 0.0571813620398730
77: 1, 0.05710953098296454: 1, 0.05689098404417975: 1, 0.05674868128348327: 1,
0.056619715346581945: 1, 0.056344538720203924: 1, 0.056067467673323004: 1, 0.05
6053636478744506: 1, 0.05599393147639249: 1, 0.05599272035746747: 1, 0.05584706
18495921: 1, 0.05579649940013397: 1, 0.05532432172575332: 1, 0.0550137359518401
84: 1, 0.054750005670025574: 1, 0.05472274463462549: 1, 0.05460959460083671: 1,
0.05428587797790197: 1, 0.05424910287102801: 1, 0.054052280221415554: 1, 0.0540
37508924245276: 1, 0.05402225742208406: 1, 0.05387136334327038: 1, 0.0534124228
475321: 1, 0.0529491791340096: 1, 0.05278016049303411: 1, 0.052685745742144734:
1, 0.052487756238435615: 1, 0.052457603028700506: 1, 0.05242187603487786: 1, 0.
051950132690053476: 1, 0.051888820043895896: 1, 0.05163323674480221: 1, 0.05128
062758128429: 1, 0.05125807382371826: 1, 0.05118524024238139: 1, 0.051175157595

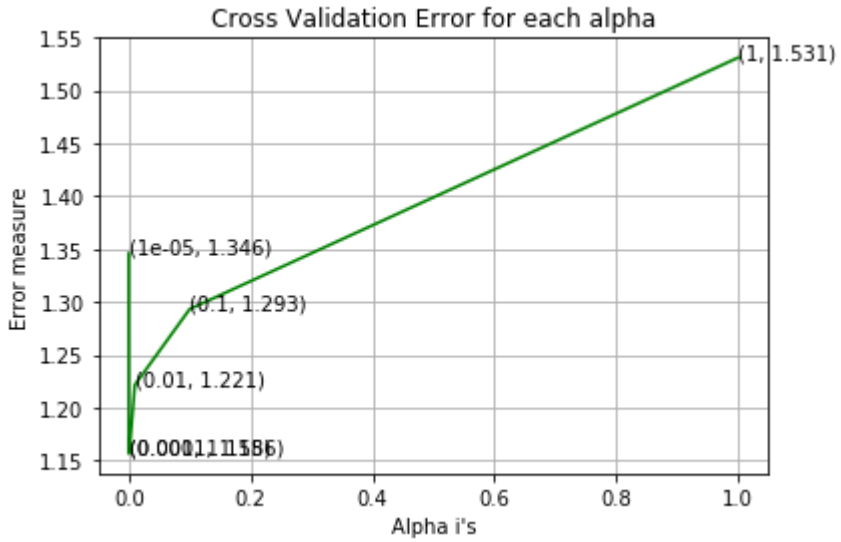
27369: 1, 0.05099226206412839: 1, 0.050532950217480846: 1, 0.05049444287963189
6: 1, 0.05037026303426785: 1, 0.05011137638068361: 1, 0.050090574725044704: 1,
0.049727616866280064: 1, 0.049637228950377185: 1, 0.049330931826234825: 1, 0.04
880808810188591: 1, 0.048527312162814: 1, 0.04839222687332448: 1, 0.04814827584
447664: 1, 0.04810031211695612: 1, 0.04723613169841162: 1, 0.04715076184395092:
1, 0.04668538922008989: 1, 0.04638708391059253: 1, 0.046370072430625436: 1, 0.0
4602338154448422: 1, 0.04600169012008115: 1, 0.045933689055288454: 1, 0.0458367
7345912117: 1, 0.04583232528060012: 1, 0.04554846721215702: 1, 0.04547688166143
067: 1, 0.04536366671009939: 1, 0.04512267995575041: 1, 0.044887618937652536:
1, 0.04452805406041436: 1, 0.04440863070496533: 1, 0.044246977857775: 1, 0.044
22276779831369: 1, 0.04420982238558023: 1, 0.04385544947477649: 1, 0.0438429337
76254256: 1, 0.04376881991805324: 1, 0.04307282789686459: 1, 0.0429430835770215
1: 1, 0.042859635766935966: 1, 0.042679448255590405: 1, 0.04256324730722678: 1,
0.042173244053704546: 1, 0.04210293771830512: 1, 0.04186727930108745: 1, 0.0418
2955593994886: 1, 0.04173825994552654: 1, 0.041278506732316374: 1, 0.0409549269
3977491: 1, 0.04060360324288295: 1, 0.040586743735122624: 1, 0.0404339837961718
64: 1, 0.04036877980278815: 1, 0.04020417710657428: 1, 0.040110460393460806: 1,
0.03963921832057639: 1, 0.03957514525515198: 1, 0.03956429356172243: 1, 0.03954
492799877071: 1, 0.03945877168347467: 1, 0.03922445345600466: 1, 0.039018847323
01467: 1, 0.03868411101310815: 1, 0.0381091265199807: 1, 0.038090811782468946:
1, 0.03804583661259847: 1, 0.03803826236584501: 1, 0.037690811545059044: 1, 0.0
3765001702525064: 1, 0.03762431011203438: 1, 0.037323486366299674: 1, 0.0369154
0633653157: 1, 0.036857208687309566: 1, 0.03663987211701162: 1, 0.0364581567267
3774: 1, 0.036449434067637426: 1, 0.03641795695057058: 1, 0.0362993884427328:
1, 0.036217655989859016: 1, 0.036011358321290245: 1, 0.03583403146693428: 1, 0.
03563548456896472: 1, 0.03560611190097075: 1, 0.03521771232009177: 1, 0.0347771
36651193294: 1, 0.03477699558430293: 1, 0.034730904399218325: 1, 0.034674712157
191666: 1, 0.03465519558373954: 1, 0.03440863395183821: 1, 0.0343509439173001O
6: 1, 0.03433790908339811: 1, 0.034278394997450536: 1, 0.03396810626832517: 1,
0.03359103524280988: 1, 0.03324155668366957: 1, 0.033210894538906674: 1, 0.0330
87499899635525: 1, 0.03291656073840764: 1, 0.032851566111616395: 1, 0.032770464
30443335: 1, 0.032642838208788175: 1, 0.03252780580183565: 1, 0.032423430579676
66: 1, 0.032167257531175227: 1, 0.031983010249676: 1, 0.03171159170667678: 1,
0.03164967875036264: 1, 0.03148494096574951: 1, 0.03144742879634749: 1, 0.03119
9400287994155: 1, 0.03114667082464237: 1, 0.03107581083347263: 1, 0.03105393651
3534025: 1, 0.03105026622200515: 1, 0.03086775328092939: 1, 0.03078122196852097
6: 1, 0.03040753010878319: 1, 0.03036871739939646: 1, 0.03033526777166989: 1,
0.03027357232287131: 1, 0.03002795830656984: 1, 0.0297994794638281: 1, 0.029601
70595086858: 1, 0.02954364325360276: 1, 0.02943360219875594: 1, 0.0293463692952
52226: 1, 0.029346117742784117: 1, 0.02927526883062173: 1, 0.02923827842140165
6: 1, 0.02896370946458059: 1, 0.02894352966972965: 1, 0.028563359983332856: 1,
0.028455369117764417: 1, 0.028448525889663116: 1, 0.0283481966072183: 1, 0.0281
5922382219481: 1, 0.028010026262719374: 1, 0.027851609858528435: 1, 0.027652860
72723028: 1, 0.027494482985977786: 1, 0.027309700001806332: 1, 0.02702988203840
5978: 1, 0.02694645681135419: 1, 0.026657794681824155: 1, 0.026585330536747828:
1, 0.026460641733257417: 1, 0.026228801514350253: 1, 0.026191152159912093: 1,
0.026152275213435234: 1, 0.026011819576281056: 1, 0.02595777279162012: 1, 0.025
849315225258185: 1, 0.02583862734476794: 1, 0.0257206052076291З: 1, 0.025228282
977478665: 1, 0.02520762575144097: 1, 0.025114684911877565: 1, 0.02510143997573
7175: 1, 0.025034563498518198: 1, 0.02498961827598837: 1, 0.024924450587991663:
1, 0.02490873155572403: 1, 0.02488625335894109: 1, 0.02461892355279316: 1, 0.02
458650394634208: 1, 0.02458182789460434: 1, 0.0242787193368579: 1, 0.0242773246
82277026: 1, 0.024198719599871087: 1, 0.02406074390189837: 1, 0.024027175794875
698: 1, 0.023913112843074576: 1, 0.023810873981764753: 1, 0.023764525854948572:
1, 0.023709958462897952: 1, 0.02370975390574179: 1, 0.023234778178481234: 1, 0.
023201130291476243: 1, 0.02305144614888024: 1, 0.0229878934078201: 1, 0.0229124
20403447677: 1, 0.022889392081656246: 1, 0.022872544082224194: 1, 0.02284271349

8048058: 1, 0.022838487724931848: 1, 0.022723980092746447: 1, 0.022616467903155857: 1, 0.022559290441014193: 1, 0.022421226599121616: 1, 0.02225327759318856: 1, 0.022221688568270732: 1, 0.02215108206134105: 1, 0.022073526724175577: 1, 0.021953037476188465: 1, 0.02186969822138065: 1, 0.021759622551497285: 1, 0.021560832316022883: 1, 0.021479465836657705: 1, 0.02142261547367326: 1, 0.021396618459137092: 1, 0.021246585605532345: 1, 0.021107236045059704: 1, 0.021086622026852273: 1, 0.021037423630774106: 1, 0.02089464139096997: 1, 0.02066402664721385: 1, 0.020476688474643887: 1, 0.02046028006460119: 1, 0.020401288027610653: 1, 0.020166709205009924: 1, 0.020163188274937665: 1, 0.02013683022345541: 1, 0.020135053118008797: 1, 0.020081805261021164: 1, 0.019897804677300804: 1, 0.019666562666043: 1, 0.01961023291367143: 1, 0.01960855328673544: 1, 0.019318453347615925: 1, 0.01910571532231357: 1, 0.019006793508323374: 1, 0.018975897798227483: 1, 0.018954357810640005: 1, 0.01888591437953166 3: 1, 0.018650077950710595: 1, 0.018639055144849117: 1, 0.018369797273885848: 1, 0.018253714087536274: 1, 0.018106462340525258: 1, 0.017934063891647055: 1, 0.017834560556251957: 1, 0.017772789634939 25: 1, 0.017754712924170316: 1, 0.017553805811663235: 1, 0.017463729149181 24: 1, 0.017384072504006996: 1, 0.01735064147548708: 1, 0.017237391380398183: 1, 0.01720946912192624: 1, 0.017155179322379935: 1, 0.017044021256171216: 1, 0.017008157780060318: 1, 0.016977465523764425: 1, 0.016944031680687387: 1, 0.016904168394739627: 1, 0.016758393398588946: 1, 0.016606173785775256: 1, 0.016550074318187676: 1, 0.01650273920272203: 1, 0.016336222626137098: 1, 0.016314922828693533: 1, 0.01615261235392226: 1, 0.015983398361046845: 1, 0.015887635173253842: 1, 0.01570722426737869: 1, 0.015465219566213398: 1, 0.015351206953373612: 1, 0.015306254353023997: 1, 0.015076082028119206: 1, 0.015031754042184901: 1, 0.014688233504283877: 1, 0.014494811155130538: 1, 0.014242198949457528: 1, 0.014089280928456066: 1, 0.013902149957988873: 1, 0.013868046019896097: 1, 0.013678096473423057: 1, 0.013646026074859926: 1, 0.013389370022005104: 1, 0.013369587080658572: 1, 0.01334843646879573: 1, 0.013077803828888197: 1, 0.01301450756741035: 1, 0.012927933106109176: 1, 0.011918959525929137: 1, 0.011854979231448976: 1, 0.011465762606489988: 1, 0.011371758432393596: 1, 0.011148014035948554: 1, 0.011022845077226649: 1, 0.01082741765515425: 1, 0.010819118702177555: 1, 0.010700074603507066: 1, 0.010565110269986314: 1, 0.010547207715593145: 1, 0.010544642952807 13: 1, 0.010543311013426137: 1, 0.0104977800346775: 1, 0.010343411981489864: 1, 0.009618396394266862: 1, 0.009479980985246815: 1, 0.009449100107118866: 1, 0.009289308762795363: 1, 0.009196365065923237: 1, 0.009079317608507442: 1, 0.009079047441160497: 1, 0.008789659184933464: 1, 0.008686928388793754: 1, 0.007891963383611368: 1, 0.007791819904023787: 1, 0.007439630466213544: 1, 0.007437716012316955: 1, 0.0071342162155350036: 1, 0.00696072376630592: 1})

In [70]:
```
# 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:",
```

```
For values of alpha =  1e-05 The log loss is: 1.3457729458880652
```

```
For values of alpha =  0.0001 The log loss is: 1.1564188004898084
For values of alpha =  0.001 The log loss is: 1.1576643831045526
For values of alpha =  0.01 The log loss is: 1.2213483868514812
For values of alpha =  0.1 The log loss is: 1.2930920342498144
For values of alpha =  1 The log loss is: 1.5308778288471732
```




```
For values of best alpha =  0.0001 The train log loss is: 0.657821453269681
For values of best alpha =  0.0001 The cross validation log loss is: 1.15641880
04898084
For values of best alpha =  0.0001 The test log loss is: 1.1610924080622236
```

**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(min_df=3)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])

    df_text_features = top_features(df_text_fea,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 [74]:
```
len1,len2 = get_intersec_text(test_df.astype('U'))
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train dat
len1,len2 = get_intersec_text(cv_df.astype('U'))
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in tr
```

```
61.9 % of word of test data appeared in train data
59.9 % of word of Cross Validation appeared in train data
```

# 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 belong
    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 = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3)

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

    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 [{}]".for
        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 [{}]".for

    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,tra
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_
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 [79]:
```
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_onel
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, 55977)
(number of data points * number of features) in test data =  (665, 55977)
(number of data points * number of features) in cross validation data = (532, 5
5977)
```

In [80]:
```
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

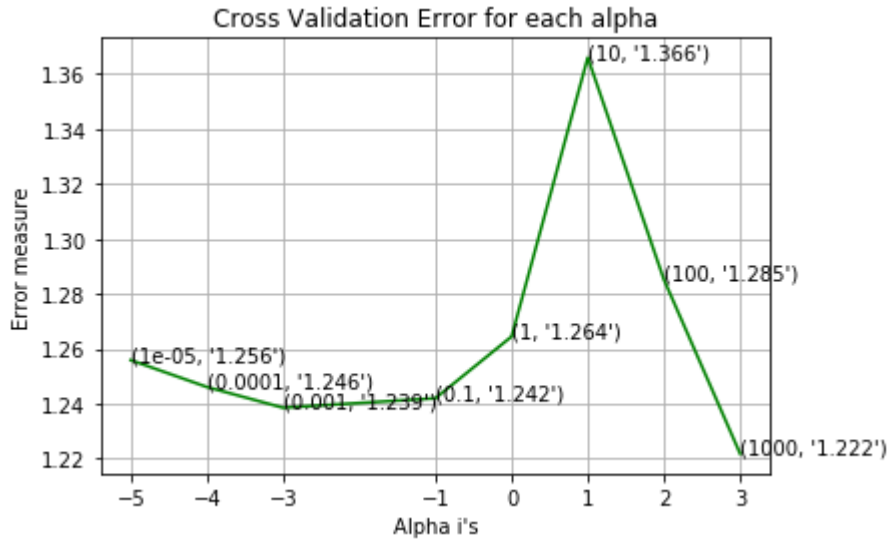
## 4.1.1. Naive Bayes

#### 4.1.1.1. Hyper parameter tuning

In [81]:
```
# 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-probabil
    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:",
```

```
for alpha = 1e-05
Log Loss : 1.2558660576507639
for alpha = 0.0001
Log Loss : 1.246078525669944
for alpha = 0.001
Log Loss : 1.2385245681274566
for alpha = 0.1
Log Loss : 1.2419638148552121
for alpha = 1
Log Loss : 1.2644566174088891
for alpha = 10
Log Loss : 1.3660114894381243
for alpha = 100
Log Loss : 1.2851685303532099
for alpha = 1000
Log Loss : 1.2218090115085845
```



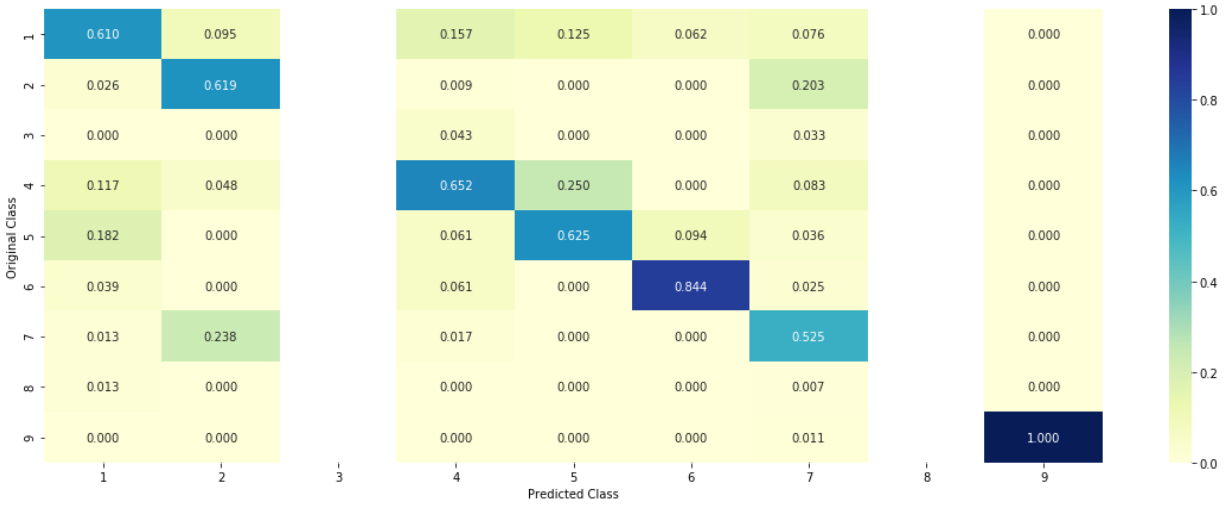

```
For values of best alpha =  1000 The train log loss is: 0.9030795578142151
For values of best alpha =  1000 The cross validation log loss is: 1.2218090115
085845
For values of best alpha =  1000 The test log loss is: 1.23040790946074
```

#### 4.1.1.2. Testing the model with best hyper paramters

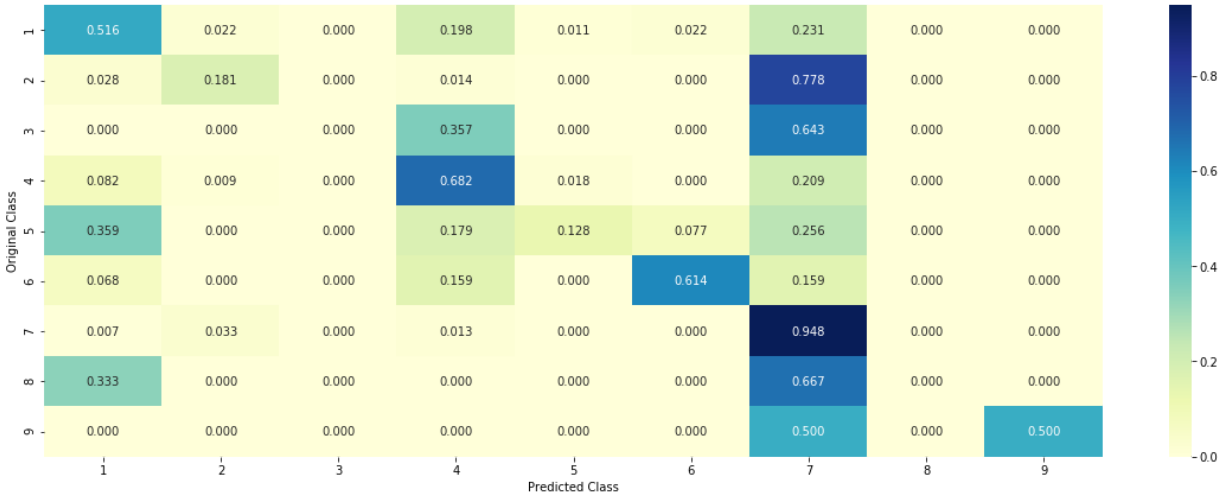
In [82]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/s
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# 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/
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoi
#
# 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 e
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()))
```

```
Log Loss : 1.2218090115085845
Number of missclassified point : 0.40789473684210525
-------------------- Confusion matrix --------------------
```

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




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




### 4.1.1.3. Feature Importance, Correctly classified point

In [86]:
```
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'].astype('U').iloc[test_point_inde
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3314 0.0903 0.0253 0.2039 0.0596 0.0505 0.22
68 0.0059 0.0064]]
Actual Class : 1
--------------------------------------------------
12 Text feature [protein] present in test data point [True]
13 Text feature [type] present in test data point [True]
14 Text feature [one] present in test data point [True]
15 Text feature [dna] present in test data point [True]
16 Text feature [wild] present in test data point [True]
17 Text feature [two] present in test data point [True]
18 Text feature [results] present in test data point [True]
19 Text feature [containing] present in test data point [True]
20 Text feature [function] present in test data point [True]
21 Text feature [region] present in test data point [True]
22 Text feature [therefore] present in test data point [True]
23 Text feature [also] present in test data point [True]
24 Text feature [loss] present in test data point [True]
25 Text feature [functions] present in test data point [True]
26 Text feature [binding] present in test data point [True]
27 Text feature [affect] present in test data point [True]
28 Text feature [possible] present in test data point [True]
29 Text feature [using] present in test data point [True]
30 Text feature [control] present in test data point [True]
31 Text feature [indicate] present in test data point [True]
32 Text feature [role] present in test data point [True]
34 Text feature [determined] present in test data point [True]
37 Text feature [either] present in test data point [True]
40 Text feature [three] present in test data point [True]
41 Text feature [indicated] present in test data point [True]
42 Text feature [specific] present in test data point [True]
43 Text feature [human] present in test data point [True]
44 Text feature [similar] present in test data point [True]
45 Text feature [however] present in test data point [True]
46 Text feature [may] present in test data point [True]
47 Text feature [reduced] present in test data point [True]
48 Text feature [essential] present in test data point [True]
49 Text feature [expression] present in test data point [True]
50 Text feature [analysis] present in test data point [True]
51 Text feature [amino] present in test data point [True]
52 Text feature [form] present in test data point [True]
53 Text feature [gene] present in test data point [True]
54 Text feature [respectively] present in test data point [True]
57 Text feature [observed] present in test data point [True]
61 Text feature [directly] present in test data point [True]
```

```
64 Text feature [result] present in test data point [True]
65 Text feature [structure] present in test data point [True]
66 Text feature [ability] present in test data point [True]
67 Text feature [previous] present in test data point [True]
68 Text feature [important] present in test data point [True]
69 Text feature [previously] present in test data point [True]
70 Text feature [significant] present in test data point [True]
71 Text feature [contains] present in test data point [True]
73 Text feature [conserved] present in test data point [True]
76 Text feature [addition] present in test data point [True]
77 Text feature [within] present in test data point [True]
79 Text feature [four] present in test data point [True]
83 Text feature [suggest] present in test data point [True]
85 Text feature [cancer] present in test data point [True]
86 Text feature [described] present in test data point [True]
87 Text feature [proteins] present in test data point [True]
89 Text feature [highly] present in test data point [True]
91 Text feature [well] present in test data point [True]
92 Text feature [deletion] present in test data point [True]
95 Text feature [including] present in test data point [True]
97 Text feature [10] present in test data point [True]
98 Text feature [based] present in test data point [True]
Out of the top  100  features  62 are present in query point
```

#### 4.1.1.4. Feature Importance, Incorrectly classified point

In [87]:
```
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'].astype('U').iloc[test_point_inde
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3487 0.0883 0.0028 0.3124 0.0384 0.0299 0.17
82 0.0008 0.0004]]
Actual Class : 4
--------------------------------------------------
12 Text feature [protein] present in test data point [True]
13 Text feature [type] present in test data point [True]
14 Text feature [one] present in test data point [True]
15 Text feature [dna] present in test data point [True]
16 Text feature [wild] present in test data point [True]
17 Text feature [two] present in test data point [True]
18 Text feature [results] present in test data point [True]
19 Text feature [containing] present in test data point [True]
20 Text feature [function] present in test data point [True]
21 Text feature [region] present in test data point [True]
22 Text feature [therefore] present in test data point [True]
23 Text feature [also] present in test data point [True]
24 Text feature [loss] present in test data point [True]
26 Text feature [binding] present in test data point [True]
27 Text feature [affect] present in test data point [True]
28 Text feature [possible] present in test data point [True]
29 Text feature [using] present in test data point [True]
30 Text feature [control] present in test data point [True]
31 Text feature [indicate] present in test data point [True]
32 Text feature [role] present in test data point [True]
33 Text feature [table] present in test data point [True]
34 Text feature [determined] present in test data point [True]
35 Text feature [involved] present in test data point [True]
36 Text feature [effect] present in test data point [True]
37 Text feature [either] present in test data point [True]
38 Text feature [present] present in test data point [True]
39 Text feature [shown] present in test data point [True]
40 Text feature [three] present in test data point [True]
41 Text feature [indicated] present in test data point [True]
42 Text feature [specific] present in test data point [True]
43 Text feature [human] present in test data point [True]
44 Text feature [similar] present in test data point [True]
45 Text feature [however] present in test data point [True]
46 Text feature [may] present in test data point [True]
47 Text feature [reduced] present in test data point [True]
48 Text feature [essential] present in test data point [True]
49 Text feature [expression] present in test data point [True]
50 Text feature [analysis] present in test data point [True]
51 Text feature [amino] present in test data point [True]
52 Text feature [form] present in test data point [True]
```

```
53 Text feature [gene] present in test data point [True]
54 Text feature [respectively] present in test data point [True]
55 Text feature [critical] present in test data point [True]
56 Text feature [indicating] present in test data point [True]
57 Text feature [observed] present in test data point [True]
59 Text feature [terminal] present in test data point [True]
60 Text feature [sequences] present in test data point [True]
61 Text feature [directly] present in test data point [True]
64 Text feature [result] present in test data point [True]
65 Text feature [structure] present in test data point [True]
66 Text feature [ability] present in test data point [True]
67 Text feature [previous] present in test data point [True]
68 Text feature [important] present in test data point [True]
69 Text feature [previously] present in test data point [True]
70 Text feature [significant] present in test data point [True]
72 Text feature [several] present in test data point [True]
73 Text feature [conserved] present in test data point [True]
74 Text feature [complex] present in test data point [True]
75 Text feature [fig] present in test data point [True]
76 Text feature [addition] present in test data point [True]
77 Text feature [within] present in test data point [True]
79 Text feature [four] present in test data point [True]
80 Text feature [25] present in test data point [True]
81 Text feature [specifically] present in test data point [True]
82 Text feature [whereas] present in test data point [True]
83 Text feature [suggest] present in test data point [True]
84 Text feature [remaining] present in test data point [True]
85 Text feature [cancer] present in test data point [True]
86 Text feature [described] present in test data point [True]
87 Text feature [proteins] present in test data point [True]
88 Text feature [remains] present in test data point [True]
89 Text feature [highly] present in test data point [True]
90 Text feature [another] present in test data point [True]
91 Text feature [well] present in test data point [True]
92 Text feature [deletion] present in test data point [True]
93 Text feature [corresponding] present in test data point [True]
94 Text feature [plays] present in test data point [True]
95 Text feature [including] present in test data point [True]
96 Text feature [obtained] present in test data point [True]
97 Text feature [10] present in test data point [True]
98 Text feature [based] present in test data point [True]
99 Text feature [associated] present in test data point [True]
Out of the top  100  features  82 are present in query point
```

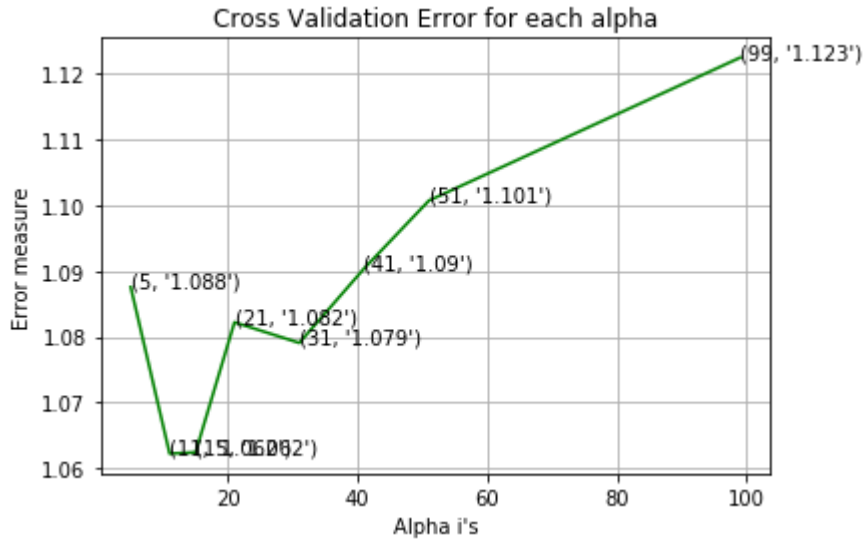
# 4.2. K Nearest Neighbour Classification

## 4.2.1. Hyper parameter tuning

In [88]:
```
# find more about KNeighborsClassifier()
# here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighb
# ------------------------
# 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-probabil
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
```

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

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

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

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

```
for alpha = 5
Log Loss : 1.087579696072036
for alpha = 11
Log Loss : 1.0622310072296028
for alpha = 15
Log Loss : 1.0624265911600341
for alpha = 21
Log Loss : 1.082222996134962
for alpha = 31
Log Loss : 1.079090098367995
for alpha = 41
Log Loss : 1.0904136555188224
for alpha = 51
Log Loss : 1.1007499890177799
for alpha = 99
Log Loss : 1.1225084479990217
```

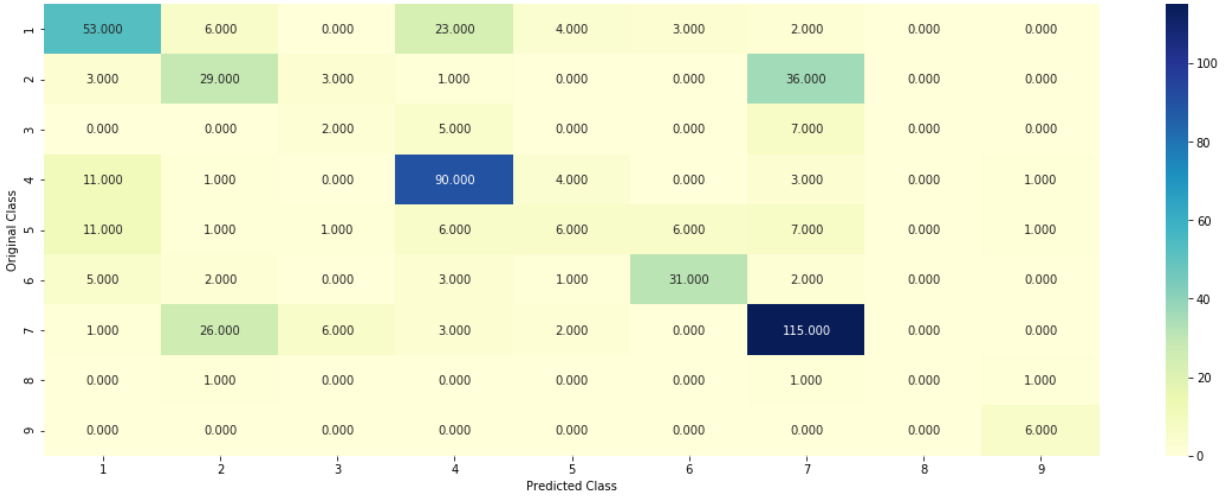



```
For values of best alpha =  11 The train log loss is: 0.6159276040983569
For values of best alpha =  11 The cross validation log loss is: 1.062231007229
6028
For values of best alpha =  11 The test log loss is: 1.1086313683996372
```

## 4.2.2. Testing the model with best hyper paramters

In [89]:
```
# find more about KNeighborsClassifier()
# here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighb
# ------------------------
# 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
```

```
Log loss : 1.0622310072296028
Number of mis-classified points : 0.37593984962406013
-------------------- Confusion matrix --------------------
```




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


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


### 4.2.3.Sample Query point -1

In [90]:
```
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 : 6
Actual Class : 1
The  11  nearest neighbours of the test points belongs to classes [1 1 4 4 1 4
1 1 4 1 1]
Fequency of nearest points : Counter({1: 7, 4: 4})
```

### 4.2.4. Sample Query Point-2

In [91]:
```
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(
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 : 4
Actual Class : 4
the k value for knn is 11 and the nearest neighbours of the test points belongs
to classes [4 4 1 4 4 4 4 5 1 4 4]
Fequency of nearest points : Counter({4: 8, 1: 2, 5: 1})
```

# 4.3. Logistic Regression

## 4.3.1. With Class balancing

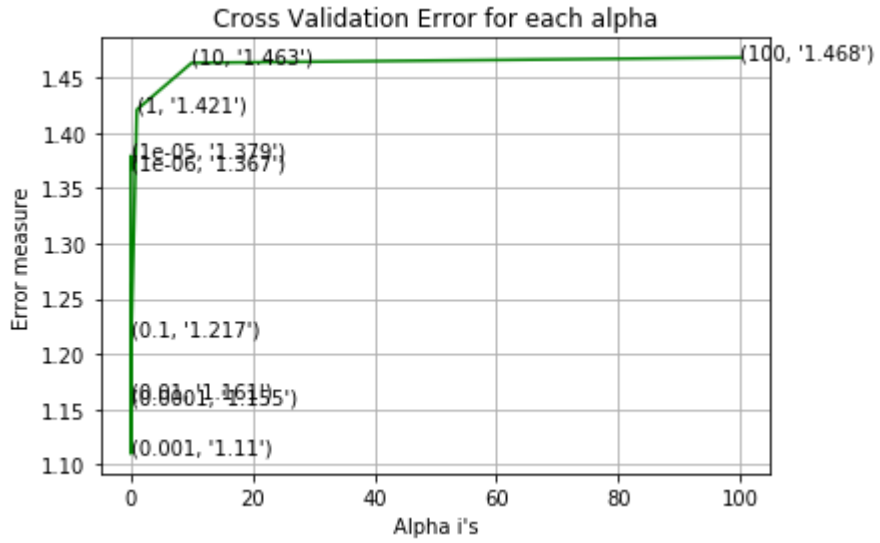
### 4.3.1.1. Hyper paramter tuning

In [92]:

```
# 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: 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-probabil
    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='l
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

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

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

```
for alpha = 1e-06
Log Loss : 1.3672545698815266
for alpha = 1e-05
Log Loss : 1.3791547530050154
for alpha = 0.0001
Log Loss : 1.1552541103769731
for alpha = 0.001
Log Loss : 1.1100130974786606
for alpha = 0.01
Log Loss : 1.1613793401142918
for alpha = 0.1
Log Loss : 1.216985867479718
for alpha = 1
Log Loss : 1.4207526851512473
for alpha = 10
Log Loss : 1.463117013899427
for alpha = 100
Log Loss : 1.4678845558748719
```

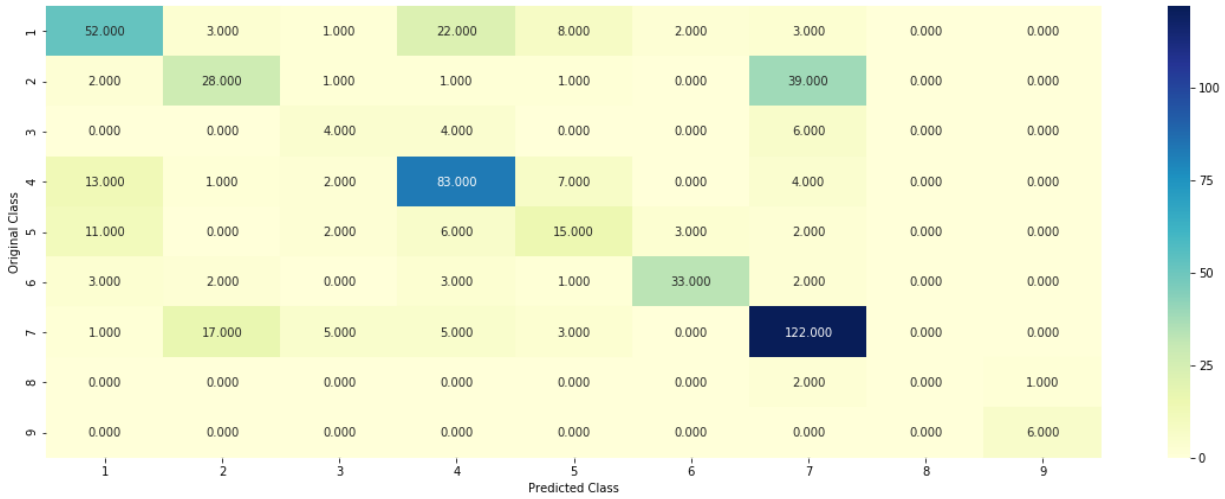



```
For values of best alpha =  0.001 The train log loss is: 0.5034468506603218
For values of best alpha =  0.001 The cross validation log loss is: 1.110013097
4786606
For values of best alpha =  0.001 The test log loss is: 1.0947340897456572
```

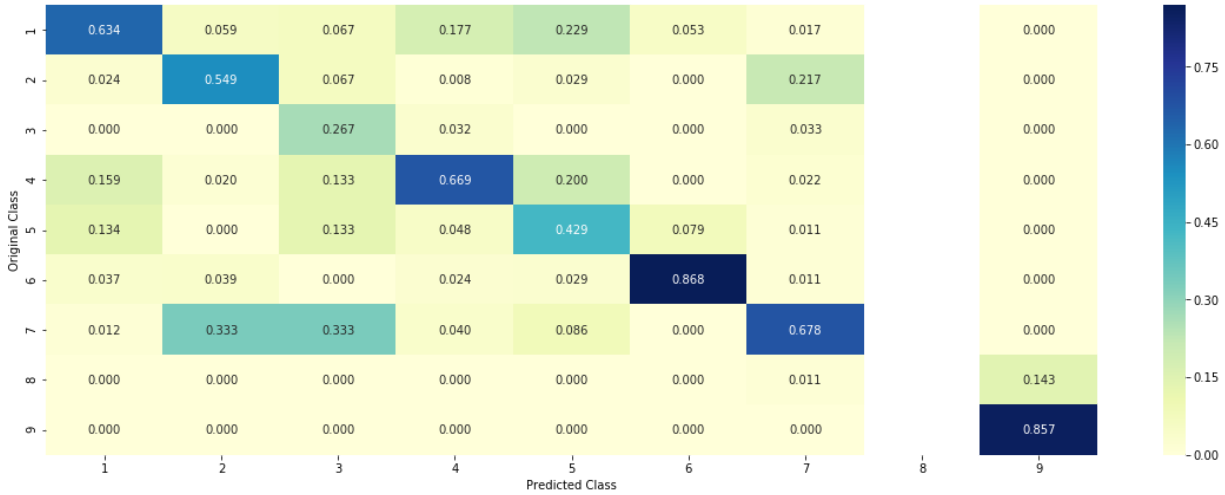
#### 4.3.1.2. Testing the model with best hyper paramters

In [93]:
```
# 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: https://www.appliedaicourse.com/course/applied-ai-course-online/les
#-------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l1
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCodi
```

```
Log loss : 1.1100130974786606
Number of mis-classified points : 0.35526315789473684
-------------------- 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_
        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]," clas
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or N
```

##### *4.3.1.3.1. Correctly Classified point*

In [95]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l
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_on
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'].astype('U').iloc[test_point_ind
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.7383 0.0509 0.0119 0.0798 0.0293 0.0192 0.06
31 0.0042 0.0033]]
Actual Class : 1
--------------------------------------------------
92 Text feature [sk18] present in test data point [True]
94 Text feature [neutravidin] present in test data point [True]
96 Text feature [pcp2] present in test data point [True]
105 Text feature [homophilic] present in test data point [True]
123 Text feature [mam] present in test data point [True]
127 Text feature [sulfo] present in test data point [True]
135 Text feature [besco] present in test data point [True]
145 Text feature [baculoviral] present in test data point [True]
235 Text feature [adhesion] present in test data point [True]
265 Text feature [ptp] present in test data point [True]
395 Text feature [sf9] present in test data point [True]
429 Text feature [5ml] present in test data point [True]
476 Text feature [aggregation] present in test data point [True]
487 Text feature [frameshifts] present in test data point [True]
489 Text feature [kalnay] present in test data point [True]
Out of the top  500  features  15 are present in query point
```

#### *4.3.1.3.2. Incorrectly Classified point*

In [96]:
```
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'].astype('U').iloc[test_point_inde
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.4327 0.063  0.0027 0.4655 0.0205 0.0065 0.00
66 0.0017 0.0008]]
Actual Class : 4
--------------------------------------------------
108 Text feature [summarised] present in test data point [True]
123 Text feature [atpase] present in test data point [True]
166 Text feature [homologues] present in test data point [True]
203 Text feature [mucosa] present in test data point [True]
233 Text feature [biallelic] present in test data point [True]
245 Text feature [germline] present in test data point [True]
289 Text feature [evidences] present in test data point [True]
291 Text feature [come] present in test data point [True]
334 Text feature [kindreds] present in test data point [True]
346 Text feature [nonsense] present in test data point [True]
353 Text feature [pseudogenes] present in test data point [True]
366 Text feature [microscopy] present in test data point [True]
404 Text feature [noncarriers] present in test data point [True]
467 Text feature [pten] present in test data point [True]
470 Text feature [ovary] present in test data point [True]
489 Text feature [carriers] present in test data point [True]
Out of the top  500  features  16 are present in query point
```

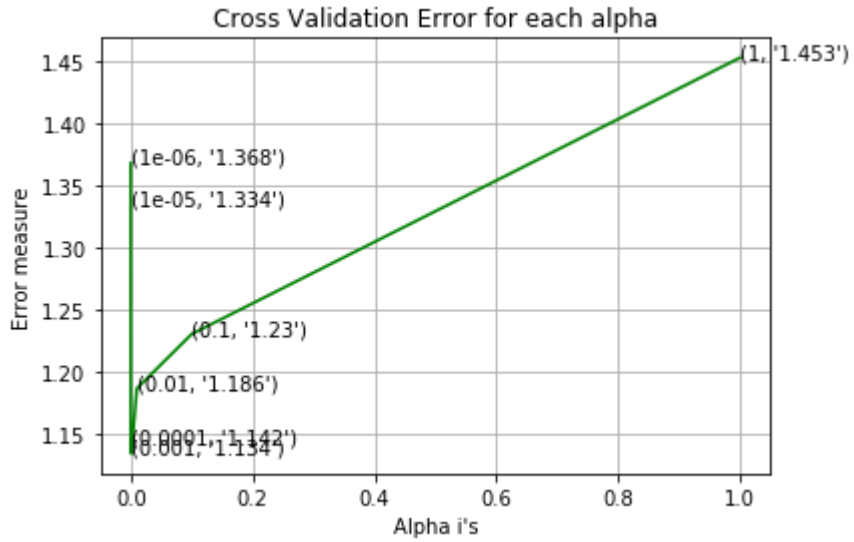
## 4.3.2. Without Class balancing

#### 4.3.2.1. Hyper paramter tuning

In [97]:
```
# 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: 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:",
      log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))

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

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

```
for alpha = 1e-06
Log Loss : 1.367740281134464
for alpha = 1e-05
Log Loss : 1.3341270577237252
for alpha = 0.0001
Log Loss : 1.1415782872609666
for alpha = 0.001
Log Loss : 1.1337655364236958
for alpha = 0.01
Log Loss : 1.1862477228856494
for alpha = 0.1
Log Loss : 1.2300824960763768
for alpha = 1
Log Loss : 1.4527576477606614
```

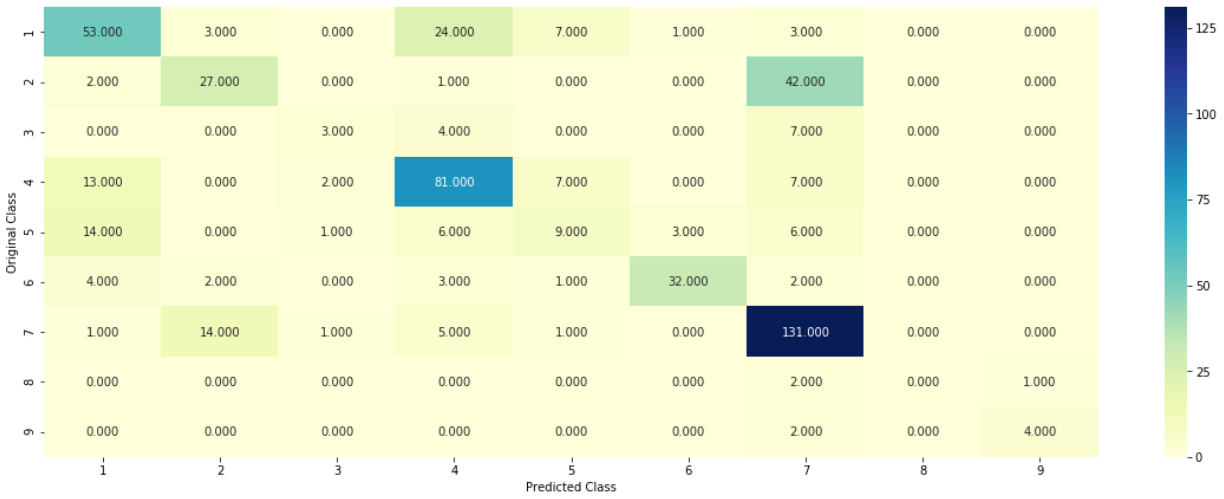


```
For values of best alpha =  0.001 The train log loss is: 0.5015480262595232
For values of best alpha =  0.001 The cross validation log loss is: 1.133765536
4236958
For values of best alpha =  0.001 The test log loss is: 1.1152097469480025
```

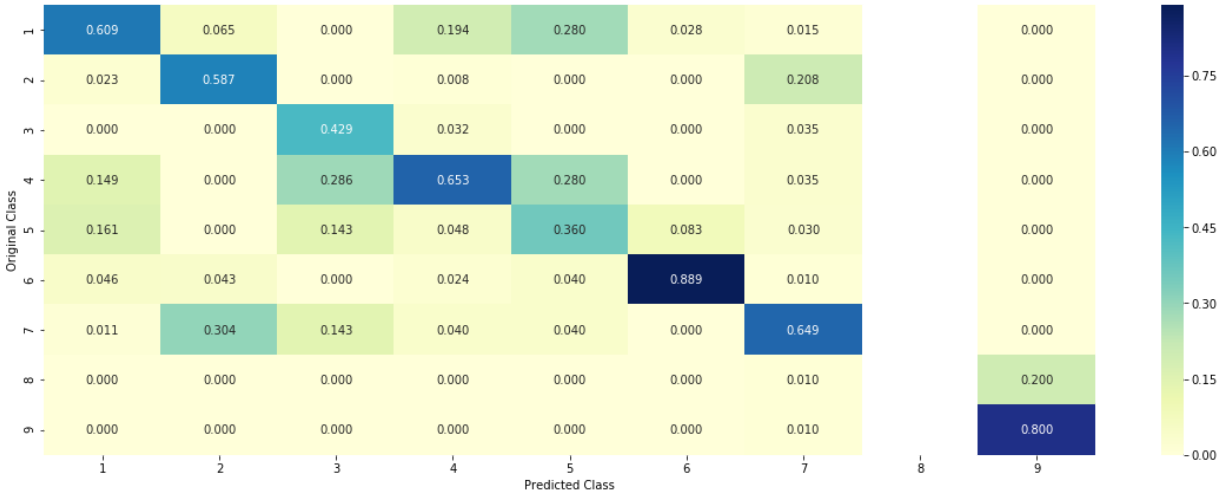
#### 4.3.2.2. Testing model with best hyper parameters

In [98]:
```
# 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:
#-------------------------------

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

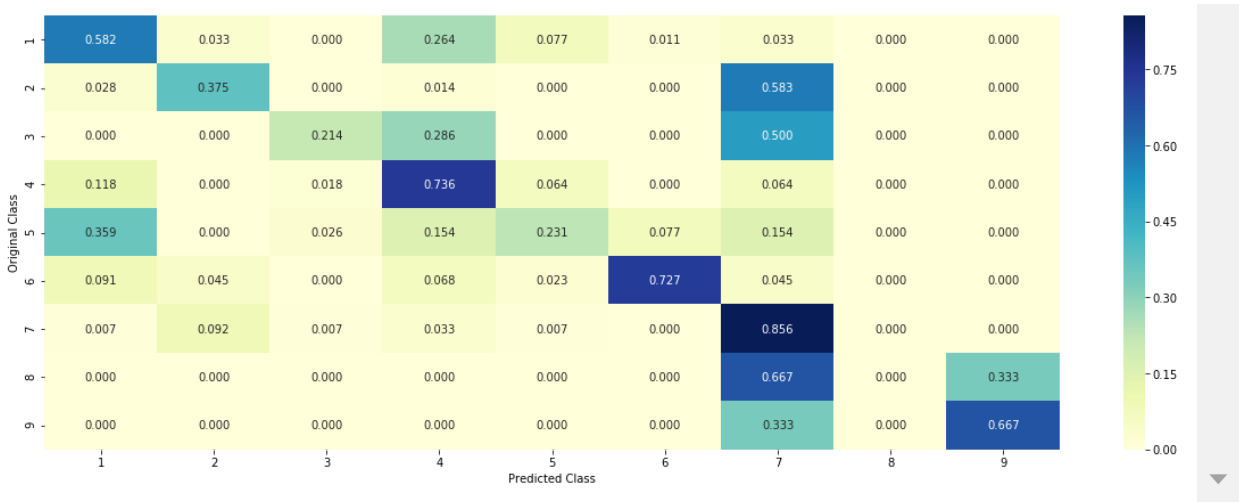
```
Log loss : 1.1337655364236958
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```




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




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




### 4.3.2.3. Feature Importance, Correctly Classified point

In [99]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l
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_on
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'].astype('U').iloc[test_point_ind
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.7434 0.0508 0.0118 0.0771 0.029  0.0191 0.06
24 0.0032 0.0033]]
Actual Class : 1
--------------------------------------------------
92 Text feature [sk18] present in test data point [True]
94 Text feature [neutravidin] present in test data point [True]
96 Text feature [pcp2] present in test data point [True]
105 Text feature [homophilic] present in test data point [True]
123 Text feature [mam] present in test data point [True]
127 Text feature [sulfo] present in test data point [True]
135 Text feature [besco] present in test data point [True]
145 Text feature [baculoviral] present in test data point [True]
235 Text feature [adhesion] present in test data point [True]
265 Text feature [ptp] present in test data point [True]
395 Text feature [sf9] present in test data point [True]
429 Text feature [5ml] present in test data point [True]
476 Text feature [aggregation] present in test data point [True]
487 Text feature [frameshifts] present in test data point [True]
489 Text feature [kalnay] present in test data point [True]
Out of the top  500  features  15 are present in query point
```

### 4.3.2.4. Feature Importance, Inorrectly Classified point

In [100]:
```
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'].astype('U').iloc[test_point_inde
```

```
Predicted Class : 4
Predicted Class Probabilities: [[4.329e-01 5.620e-02 9.000e-04 4.800e-01 1.34
0e-02 4.900e-03 1.160e-02
  0.000e+00 1.000e-04]]
Actual Class : 4
--------------------------------------------------
108 Text feature [summarised] present in test data point [True]
123 Text feature [atpase] present in test data point [True]
166 Text feature [homologues] present in test data point [True]
203 Text feature [mucosa] present in test data point [True]
233 Text feature [biallelic] present in test data point [True]
245 Text feature [germline] present in test data point [True]
289 Text feature [evidences] present in test data point [True]
291 Text feature [come] present in test data point [True]
334 Text feature [kindreds] present in test data point [True]
346 Text feature [nonsense] present in test data point [True]
353 Text feature [pseudogenes] present in test data point [True]
366 Text feature [microscopy] present in test data point [True]
404 Text feature [noncarriers] present in test data point [True]
```

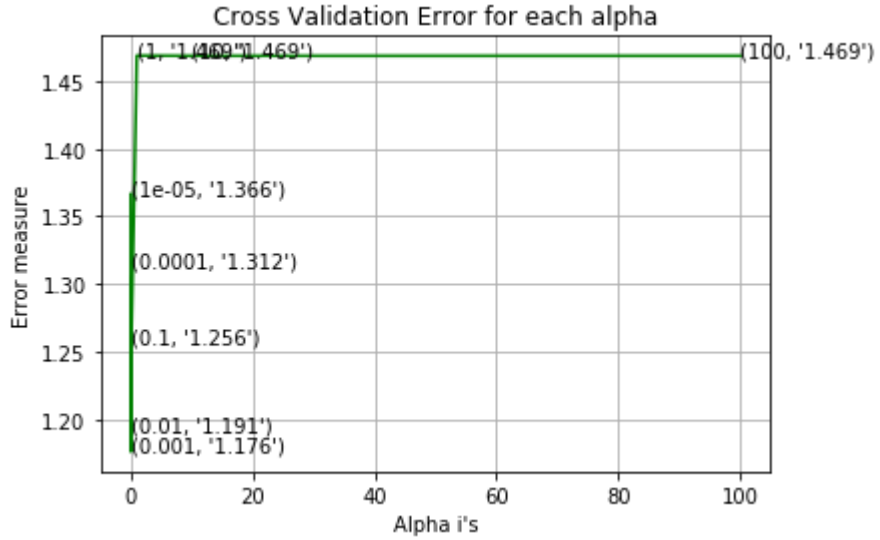
# 4.4. Linear Support Vector Machines

## 4.4.1. Hyper paramter tuning

In [101]:
```
# 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='l
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

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

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

```
for C = 1e-05
Log Loss : 1.3664752857082438
for C = 0.0001
Log Loss : 1.312467741471717
for C = 0.001
Log Loss : 1.1760276797804703
for C = 0.01
Log Loss : 1.1909377343682135
for C = 0.1
Log Loss : 1.2560237426647782
for C = 1
Log Loss : 1.4689072959852896
for C = 10
Log Loss : 1.4687474901889155
for C = 100
Log Loss : 1.4687474705925854
```



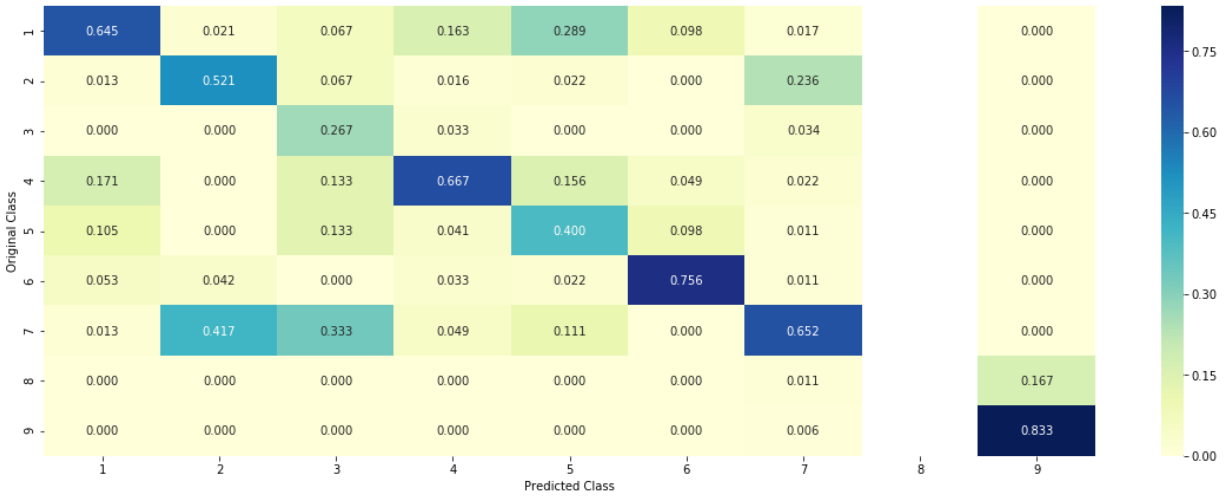
```
For values of best alpha =  0.001 The train log loss is: 0.5573683850002489
For values of best alpha =  0.001 The cross validation log loss is: 1.176027679
7804703
For values of best alpha =  0.001 The test log loss is: 1.1278615933978533
```

### 4.4.2. Testing model with best hyper parameters

In [102]:
```
# 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
```

```
Log loss : 1.1760276797804703
Number of mis-classified points : 0.37969924812030076
-------------------- 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 [103]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_
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_on
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 : 1
Predicted Class Probabilities: [[0.712  0.0613 0.0128 0.0749 0.038  0.0169 0.07
58 0.0044 0.0037]]
Actual Class : 1
--------------------------------------------------
138 Text feature [homophilic] present in test data point [True]
150 Text feature [neutravidin] present in test data point [True]
155 Text feature [sk18] present in test data point [True]
170 Text feature [pcp2] present in test data point [True]
179 Text feature [sulfo] present in test data point [True]
186 Text feature [baculoviral] present in test data point [True]
193 Text feature [mam] present in test data point [True]
236 Text feature [ptp] present in test data point [True]
238 Text feature [besco] present in test data point [True]
259 Text feature [pcp] present in test data point [True]
306 Text feature [sf9] present in test data point [True]
326 Text feature [adhesion] present in test data point [True]
469 Text feature [mediates] present in test data point [True]
Out of the top  500  features  13 are present in query point
```

#### 4.3.3.2. For Incorrectly classified point

In [104]:
```
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_on
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 : 4
Predicted Class Probabilities: [[0.3708 0.0841 0.0043 0.4781 0.0216 0.008  0.02
56 0.0036 0.0041]]
Actual Class : 4
--------------------------------------------------
225 Text feature [evidences] present in test data point [True]
227 Text feature [atpase] present in test data point [True]
264 Text feature [fancb] present in test data point [True]
271 Text feature [come] present in test data point [True]
281 Text feature [ovary] present in test data point [True]
293 Text feature [germline] present in test data point [True]
296 Text feature [summarised] present in test data point [True]
298 Text feature [homologues] present in test data point [True]
346 Text feature [carriers] present in test data point [True]
362 Text feature [thioguanine] present in test data point [True]
384 Text feature [kindreds] present in test data point [True]
433 Text feature [pseudogenes] present in test data point [True]
437 Text feature [onset] present in test data point [True]
455 Text feature [apparently] present in test data point [True]
Out of the top  500  features  14 are present in query point
```

# 4.5 Random Forest Classifier

## 4.5.1. Hyper paramter tuning (With One hot Encoding)

In [105]:
```
# ---------------------------------
# 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_lea
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_sta
# 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/
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoi
#
# 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='g
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

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

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

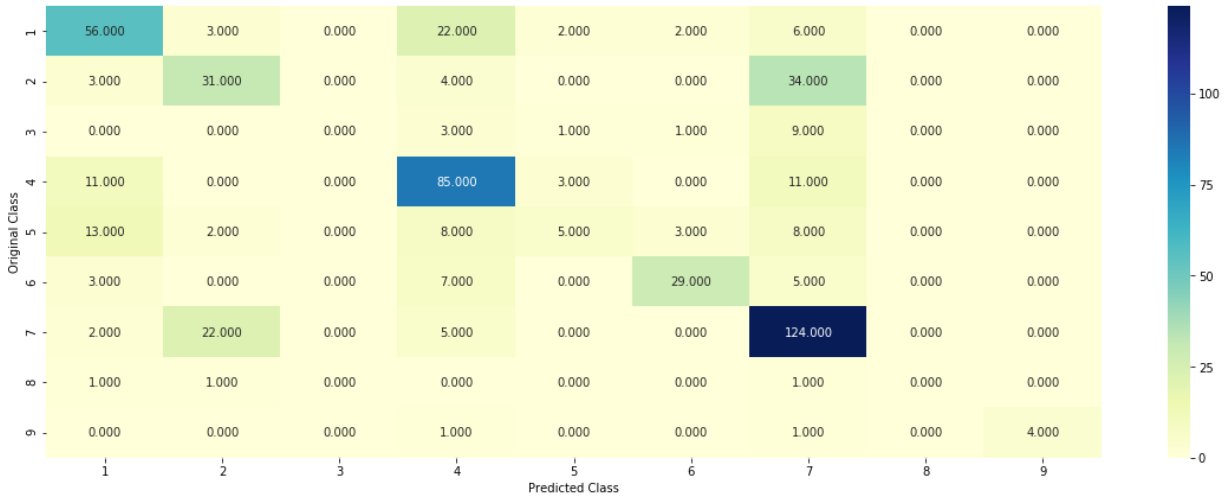
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2082582659318455
for n_estimators = 100 and max depth =  10
Log Loss : 1.163352929726615
for n_estimators = 200 and max depth =  5
Log Loss : 1.1954531015547913
for n_estimators = 200 and max depth =  10
Log Loss : 1.1490128742409007
for n_estimators = 500 and max depth =  5
Log Loss : 1.186724676376679
for n_estimators = 500 and max depth =  10
Log Loss : 1.1464929217290645
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1874202407867924
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1448771256475168
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1862517420617578
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1467562050394855
For values of best estimator =  1000 The train log loss is: 0.648753137956374
For values of best estimator =  1000 The cross validation log loss is: 1.144877
1256475168
For values of best estimator =  1000 The test log loss is: 1.1822686239249316
```

## 4.5.2. Testing model with best hyper parameters (One Hot

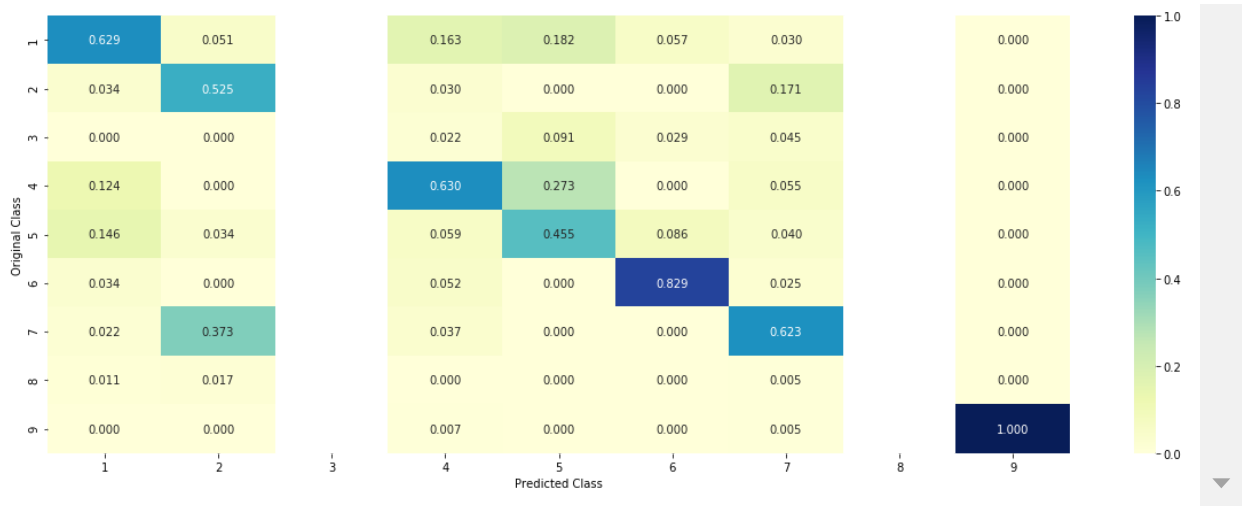
## Encoding)

In [106]:
```
# --------------------------------
# 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='g
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCodin
```

```
Log loss : 1.144877125647517
Number of mis-classified points : 0.37218045112781956
-------------------- 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 [107]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='g
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_on
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 : 1
Predicted Class Probabilities: [[0.3887 0.0625 0.0216 0.3094 0.0594 0.0497 0.09
23 0.0071 0.0093]]
Actual Class : 1
--------------------------------------------------
2 Text feature [tyrosine] present in test data point [True]
6 Text feature [suppressor] present in test data point [True]
7 Text feature [phosphorylation] present in test data point [True]
11 Text feature [loss] present in test data point [True]
13 Text feature [signaling] present in test data point [True]
16 Text feature [growth] present in test data point [True]
22 Text feature [function] present in test data point [True]
24 Text feature [protein] present in test data point [True]
25 Text feature [functional] present in test data point [True]
32 Text feature [ligand] present in test data point [True]
34 Text feature [nonsense] present in test data point [True]
36 Text feature [cells] present in test data point [True]
37 Text feature [missense] present in test data point [True]
40 Text feature [inhibited] present in test data point [True]
43 Text feature [receptor] present in test data point [True]
44 Text feature [proliferation] present in test data point [True]
51 Text feature [expressing] present in test data point [True]
52 Text feature [cell] present in test data point [True]
58 Text feature [functions] present in test data point [True]
62 Text feature [extracellular] present in test data point [True]
63 Text feature [defective] present in test data point [True]
67 Text feature [kinases] present in test data point [True]
76 Text feature [factor] present in test data point [True]
80 Text feature [il] present in test data point [True]
81 Text feature [predicted] present in test data point [True]
94 Text feature [phosphorylated] present in test data point [True]
95 Text feature [phosphatase] present in test data point [True]
Out of the top  100  features  27 are present in query point
```

#### 4.5.3.2. Inorrectly Classified point

In [108]:
```
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 : 1
Predicted Class Probabilities: [[0.4212 0.1198 0.0255 0.207  0.0884 0.0761 0.
0458 0.0077 0.0085]]
Actuall Class : 4
--------------------------------------------------
0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
3 Text feature [activation] present in test data point [True]
5 Text feature [inhibitor] present in test data point [True]
10 Text feature [constitutive] present in test data point [True]
11 Text feature [loss] present in test data point [True]
12 Text feature [treatment] present in test data point [True]
19 Text feature [therapy] present in test data point [True]
21 Text feature [pathogenic] present in test data point [True]
22 Text feature [function] present in test data point [True]
23 Text feature [frameshift] present in test data point [True]
24 Text feature [protein] present in test data point [True]
25 Text feature [functional] present in test data point [True]
26 Text feature [downstream] present in test data point [True]
27 Text feature [treated] present in test data point [True]
30 Text feature [trials] present in test data point [True]
33 Text feature [resistance] present in test data point [True]
34 Text feature [nonsense] present in test data point [True]
36 Text feature [cells] present in test data point [True]
37 Text feature [missense] present in test data point [True]
38 Text feature [stability] present in test data point [True]
39 Text feature [yeast] present in test data point [True]
44 Text feature [proliferation] present in test data point [True]
45 Text feature [patients] present in test data point [True]
46 Text feature [response] present in test data point [True]
47 Text feature [deleterious] present in test data point [True]
48 Text feature [clinical] present in test data point [True]
49 Text feature [resistant] present in test data point [True]
51 Text feature [expressing] present in test data point [True]
52 Text feature [cell] present in test data point [True]
57 Text feature [repair] present in test data point [True]
59 Text feature [classified] present in test data point [True]
61 Text feature [brca2] present in test data point [True]
63 Text feature [defective] present in test data point [True]
65 Text feature [sensitivity] present in test data point [True]
74 Text feature [neutral] present in test data point [True]
76 Text feature [factor] present in test data point [True]
77 Text feature [unstable] present in test data point [True]
78 Text feature [cosegregation] present in test data point [True]
79 Text feature [days] present in test data point [True]
```

```
81 Text feature [predicted] present in test data point [True]
82 Text feature [effective] present in test data point [True]
83 Text feature [likelihood] present in test data point [True]
84 Text feature [retained] present in test data point [True]
86 Text feature [splice] present in test data point [True]
87 Text feature [carriers] present in test data point [True]
88 Text feature [pathogenicity] present in test data point [True]
90 Text feature [truncating] present in test data point [True]
92 Text feature [amplification] present in test data point [True]
93 Text feature [months] present in test data point [True]
96 Text feature [ovarian] present in test data point [True]
97 Text feature [advanced] present in test data point [True]
98 Text feature [clinically] present in test data point [True]
Out of the top  100  features  53 are present in query point
```

### 4.5.3. Hyper paramter tuning (With Response Coding)

In [109]:
```
# --------------------------------
# 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='g
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

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

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

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

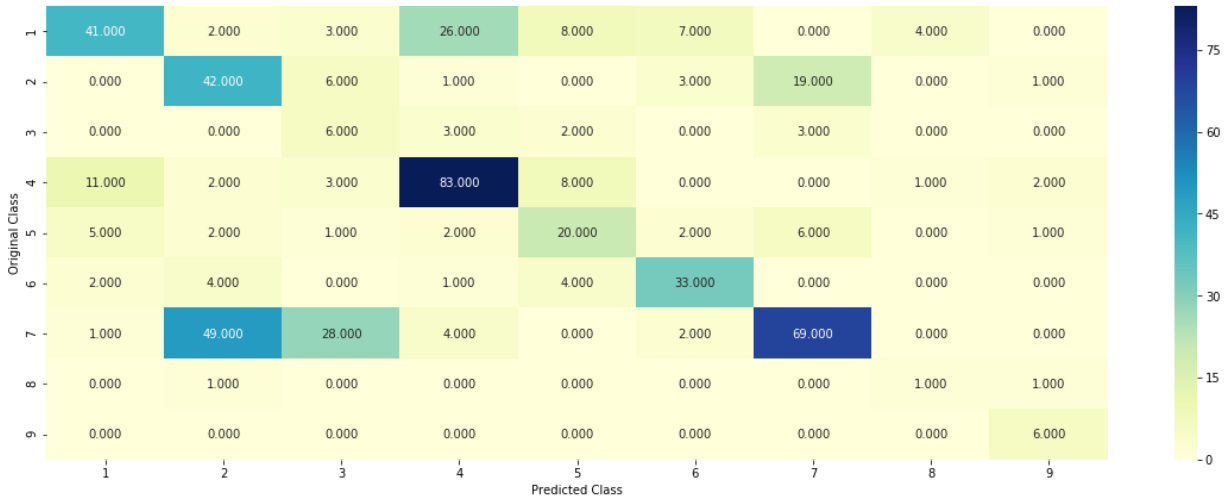
```
for n_estimators = 10 and max depth =  2
Log Loss : 2.2040794732044837
for n_estimators = 10 and max depth =  3
Log Loss : 1.7042758547134202
for n_estimators = 10 and max depth =  5
Log Loss : 1.6081884991089965
for n_estimators = 10 and max depth =  10
Log Loss : 1.8278085185747004
for n_estimators = 50 and max depth =  2
Log Loss : 1.6631833735022012
for n_estimators = 50 and max depth =  3
Log Loss : 1.3887797645889712
for n_estimators = 50 and max depth =  5
Log Loss : 1.329018743570734
for n_estimators = 50 and max depth =  10
Log Loss : 1.5251649134180507
for n_estimators = 100 and max depth =  2
Log Loss : 1.480860389019844
for n_estimators = 100 and max depth =  3
Log Loss : 1.395693250558745
for n_estimators = 100 and max depth =  5
Log Loss : 1.27501907205802
for n_estimators = 100 and max depth =  10
Log Loss : 1.5805756516162464
for n_estimators = 200 and max depth =  2
Log Loss : 1.5457801399179452
for n_estimators = 200 and max depth =  3
Log Loss : 1.4541329568781018
```

```
for n_estimators = 200 and max depth =  5
Log Loss : 1.3205345001129003
for n_estimators = 200 and max depth =  10
Log Loss : 1.6198640176936947
for n_estimators = 500 and max depth =  2
Log Loss : 1.5994244967904148
for n_estimators = 500 and max depth =  3
Log Loss : 1.485375112853613
for n_estimators = 500 and max depth =  5
Log Loss : 1.3626840295392366
for n_estimators = 500 and max depth =  10
Log Loss : 1.630750123818401
for n_estimators = 1000 and max depth =  2
Log Loss : 1.577767411785938
for n_estimators = 1000 and max depth =  3
Log Loss : 1.4939206157285194
for n_estimators = 1000 and max depth =  5
Log Loss : 1.3333086778373058
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6299769861163445
For values of best alpha =  100 The train log loss is: 0.05258383231318117
For values of best alpha =  100 The cross validation log loss is: 1.275019072
05802
For values of best alpha =  100 The test log loss is: 1.3657896871906017
```

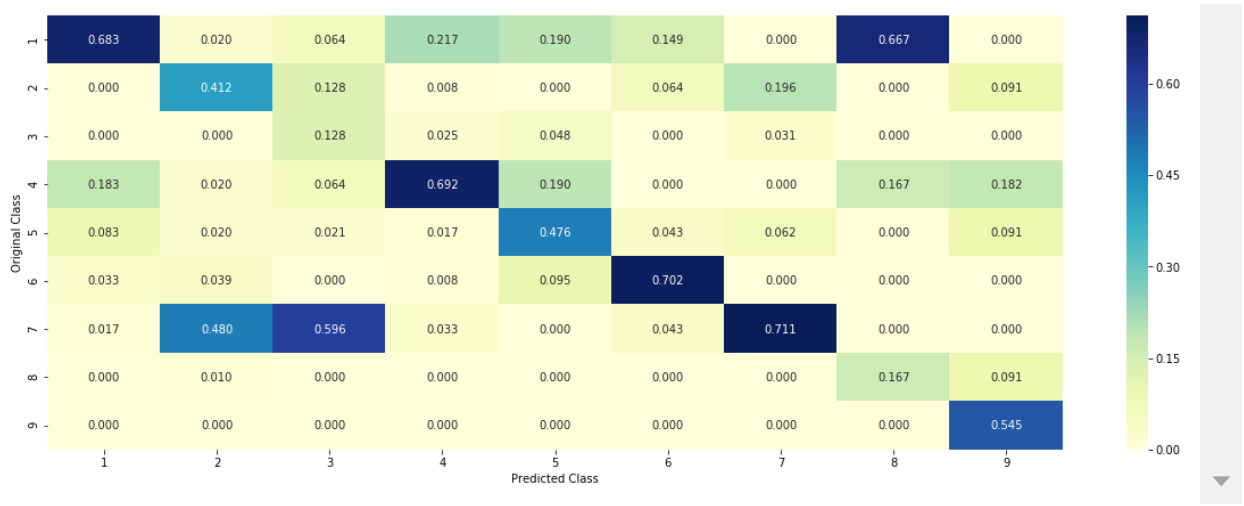
### 4.5.4. Testing model with best hyper parameters (Response Coding)

In [111]:
```
# --------------------------------
# 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
```

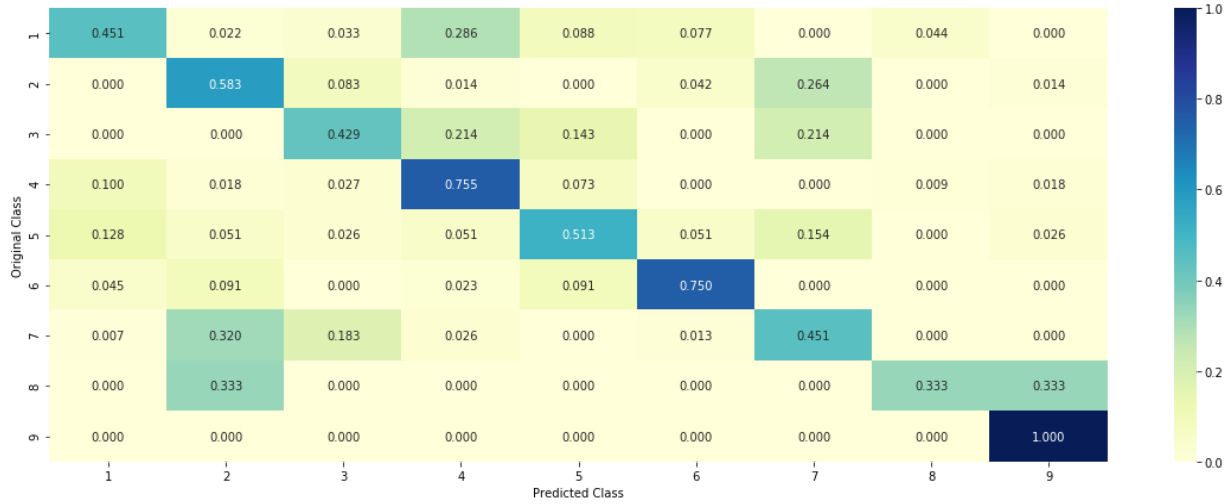
```
Log loss : 1.27501907205802
Number of mis-classified points : 0.4342105263157895
-------------------- Confusion matrix --------------------
```




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




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




## 4.5.5. Feature Importance

### 4.5.5.1. Correctly Classified point

In [112]:
```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='g
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(
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_re
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 : 1
Predicted Class Probabilities: [[0.414  0.0248 0.1609 0.2334 0.0481 0.0406 0.00
76 0.0263 0.0443]]
Actual Class : 1
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Gene 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
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

#### 4.5.5.2. Incorrectly Classified point

In [113]:
```
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 : 4
Predicted Class Probabilities: [[0.2405 0.017  0.1459 0.2557 0.1658 0.1241 0.00
77 0.0157 0.0275]]
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Gene 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
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

# 4.7 Stack the models

## 4.7.1 testing with hyper parameter tuning

In [114]:
```
# 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: 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='balanc
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")
```

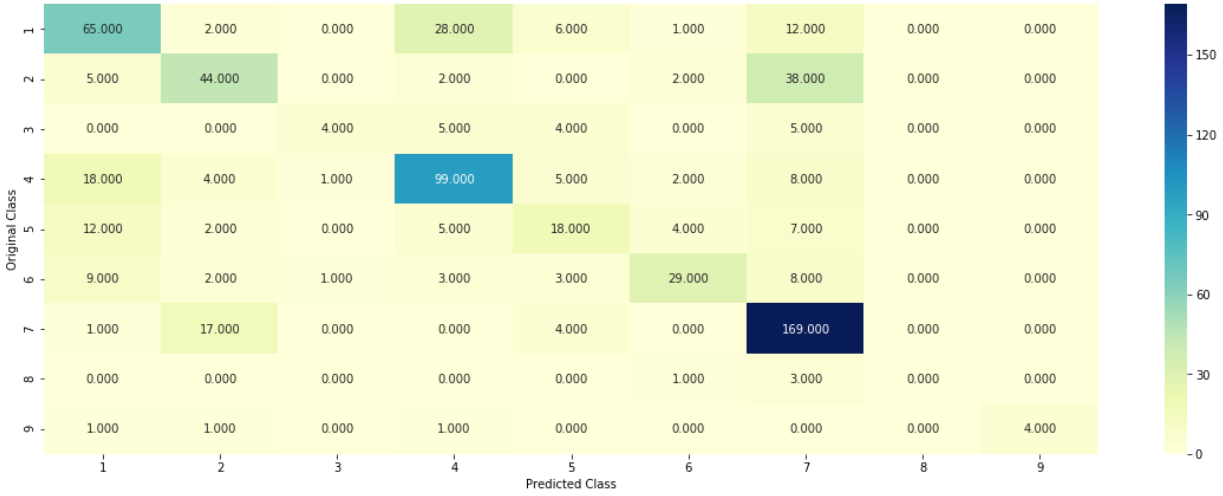
```
clf2 = SGDClassifier(alpha=0.001, penalty='l2', loss='hinge', class_weight='balan
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


clf3 = MultinomialNB(alpha=1000)
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: 1.16
Naive Bayes : Log Loss: 1.22
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.172
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.988
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.409
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.146
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.365
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.748
```

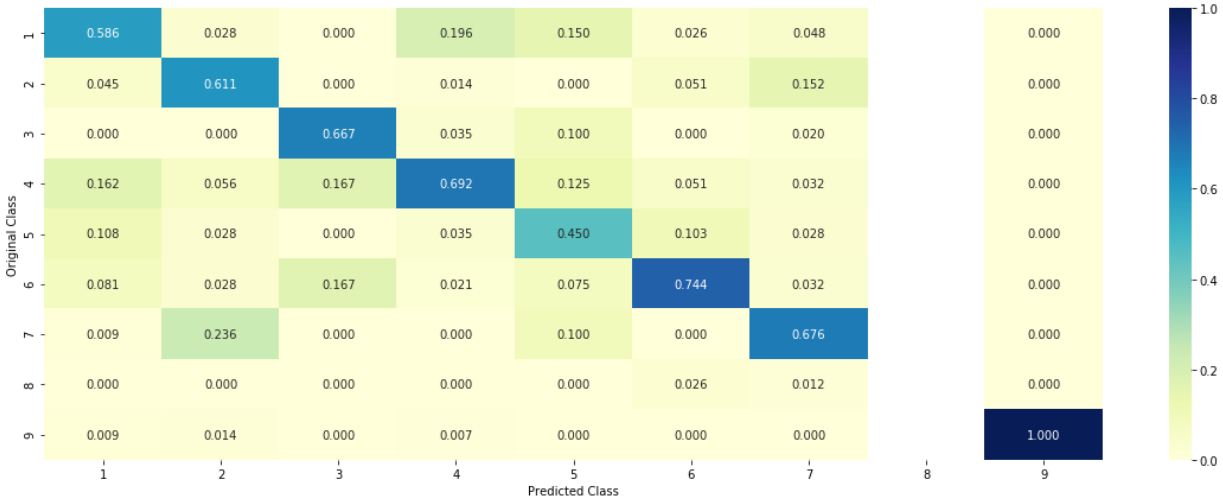
## 4.7.2 testing the model with the best hyper parameters

In [115]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_class
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

print("Number of missclassified point :", np.count_nonzero((sclf.predict(test_x_
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding)
```

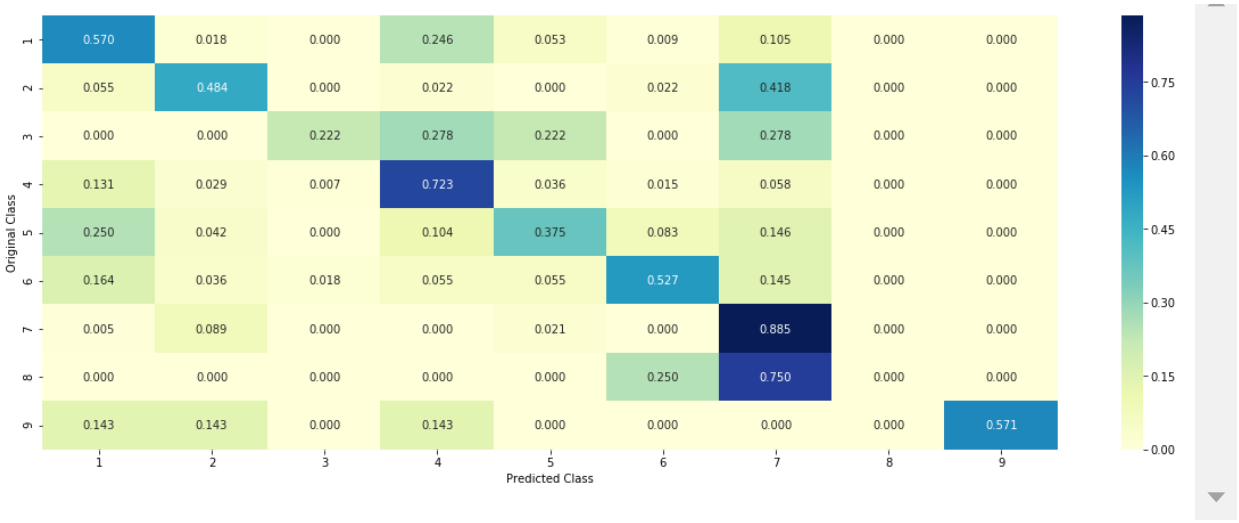
```
Log loss (train) on the stacking classifier : 0.5013991606563359
Log loss (CV) on the stacking classifier : 1.1462913881626369
Log loss (test) on the stacking classifier : 1.1296319358305498
Number of missclassified point : 0.35037593984962406
-------------------- Confusion matrix --------------------
```




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




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

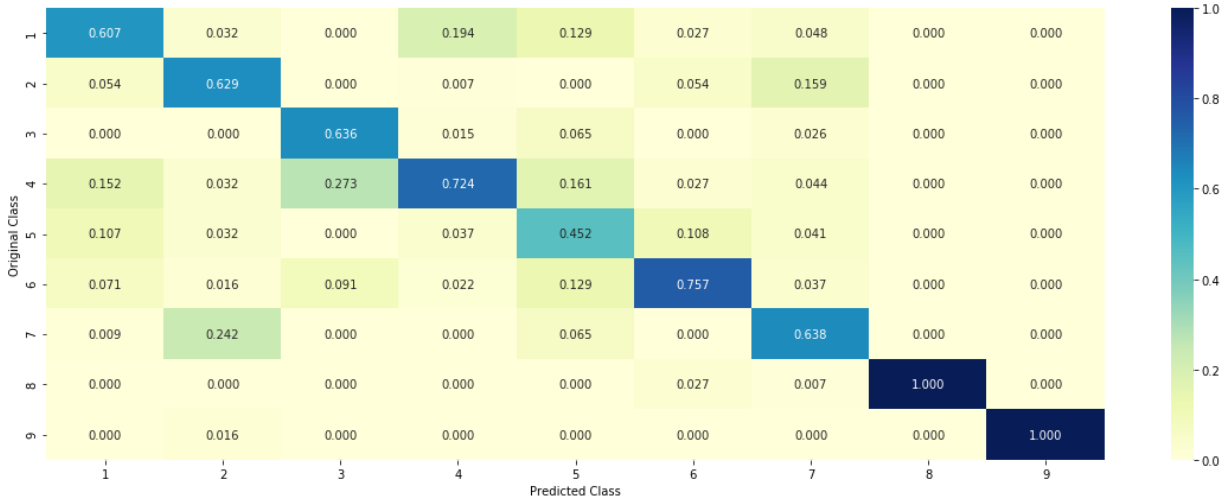



## 4.7.3 Maximum Voting classifier

In [116]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingC
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', 
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predi
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_pro
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_
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding)
```

```
Log loss (train) on the VotingClassifier : 0.6134101544066595
Log loss (CV) on the VotingClassifier : 1.0517451824328272
Log loss (test) on the VotingClassifier : 1.0695980642301117
Number of missclassified point : 0.34887218045112783
-------------------- Confusion matrix --------------------
```


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




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


Original Class
Predicted Class
1 2 3 4 5 6 7 8 9
0.596 0.018 0.000 0.228 0.035 0.009 0.114 0.000 0.000
0.066 0.429 0.000 0.011 0.000 0.022 0.473 0.000 0.000
0.000 0.000 0.389 0.111 0.111 0.000 0.389 0.000 0.000
0.124 0.015 0.022 0.708 0.036 0.007 0.088 0.000 0.000
0.250 0.042 0.000 0.104 0.292 0.083 0.229 0.000 0.000
0.145 0.018 0.018 0.055 0.073 0.509 0.182 0.000 0.000
0.005 0.079 0.000 0.000 0.010 0.000 0.906 0.000 0.000
0.000 0.000 0.000 0.000 0.000 0.250 0.500 0.250 0.000
0.000 0.143 0.000 0.000 0.000 0.000 0.000 0.000 0.857
0.0 0.2 0.4 0.6 0.8

## Summary

In [118]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.title = "****  Model Summary  **** [KPI USED: Log-loss]"
x.field_names=["Model Name","Train","CV","Test"," Misclassified Points"]
x.add_row(["Naive Bayes","0.92","1.21","1.19","41"])
x.add_row(["KNN","0.79","1.05","1.06","37"])
x.add_row(["Logistic Regression With Class balancing","0.54","1.06","1.01","34"]
x.add_row(["Logistic Regression Without Class balancing","0.54","1.07","1.2","34
x.add_row(["Linear SVM","0.79","1.03","1.06","37"])
x.add_row(["Random Forest Classifier With One hot Encoding","0.65","1.13","1.16"
x.add_row(["Random Forest Classifier With Response Coding","0.05","1.3","1.3","4
x.add_row(["Stack models:LR+NB+SVM","0.59","1.09","1.05","31"])
x.add_row(["Maximum Voting classifier","0.68","1.03","1.02","34"])
print(x)
print()
```

```
+-----------------------------------------------+-------+------+------+-------
---------------+
|                  Model Name                   | Train |  CV  | Test |  Miscl
assified Points |
+-----------------------------------------------+-------+------+------+-------
---------------+
|                 Naive Bayes                   |  0.92 | 1.21 | 1.19 |
41          |
|                     KNN                       |  0.79 | 1.05 | 1.06 |
37          |
|    Logistic Regression With Class balancing   |  0.54 | 1.06 | 1.01 |
34          |
|  Logistic Regression Without Class balancing  |  0.54 | 1.07 | 1.2  |
34          |
|                  Linear SVM                   |  0.79 | 1.03 | 1.06 |
37          |
| Random Forest Classifier With One hot Encoding |  0.65 | 1.13 | 1.16 |
36          |
| Random Forest Classifier With Response Coding  |  0.05 | 1.3  | 1.3  |
49          |
|            Stack models:LR+NB+SVM             |  0.59 | 1.09 | 1.05 |
31          |
|           Maximum Voting classifier           |  0.68 | 1.03 | 1.02 |
34          |
+-----------------------------------------------+-------+------+------+-------
---------------+
```

## Logistic Regression With Class Balancing

### Gene Feature

In [0]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer(ngram_range=(1, 2))
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'])

# don't forget to normalize every feature
train_gene_feature_onehotCoding = normalize(train_gene_feature_onehotCoding, axi
test_gene_feature_onehotCoding = normalize(test_gene_feature_onehotCoding, axis=
cv_gene_feature_onehotCoding = normalize(cv_gene_feature_onehotCoding, axis=0)
```

**Variation Feature**

In [0]:
```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer(ngram_range=(1, 2))
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Va
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variat

# don't forget to normalize every feature
train_variation_feature_onehotCoding = normalize(train_variation_feature_onehotC
test_variation_feature_onehotCoding = normalize(test_variation_feature_onehotCod
cv_variation_feature_onehotCoding = normalize(cv_variation_feature_onehotCoding,
```

**Text Feature**

In [122]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in t
text_vectorizer = CountVectorizer(min_df=3,ngram_range=(1, 2))
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT']

# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns
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 ti
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 : 796142
```

In [0]:
```
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axi

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

# we use the same vectorizer that was trained on train data
cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'].astype('U
cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding, axis=0)
```

**Stack above three features**

In [0]:
```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_varia
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variati
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_fea

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

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

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

In [127]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_o
print("(number of data points * number of features) in test data = ", test_x_onel
print("(number of data points * number of features) in cross validation data =",
```

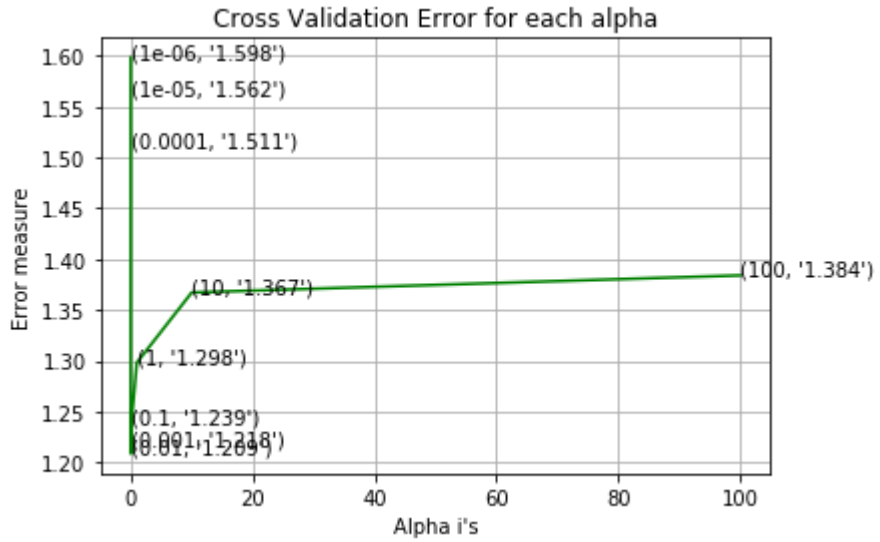
```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 798405)
(number of data points * number of features) in test data =  (665, 798405)
(number of data points * number of features) in cross validation data = (532, 7
98405)
```

**Applyting Logistic Regression**

In [128]:
```
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-probabil
    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:",
      log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))

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

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

```
for alpha = 1e-06
Log Loss : 1.5981860573710813
for alpha = 1e-05
Log Loss : 1.5618421675798824
for alpha = 0.0001
Log Loss : 1.5113041837660683
for alpha = 0.001
Log Loss : 1.2179303602275675
for alpha = 0.01
Log Loss : 1.208775708422102
```

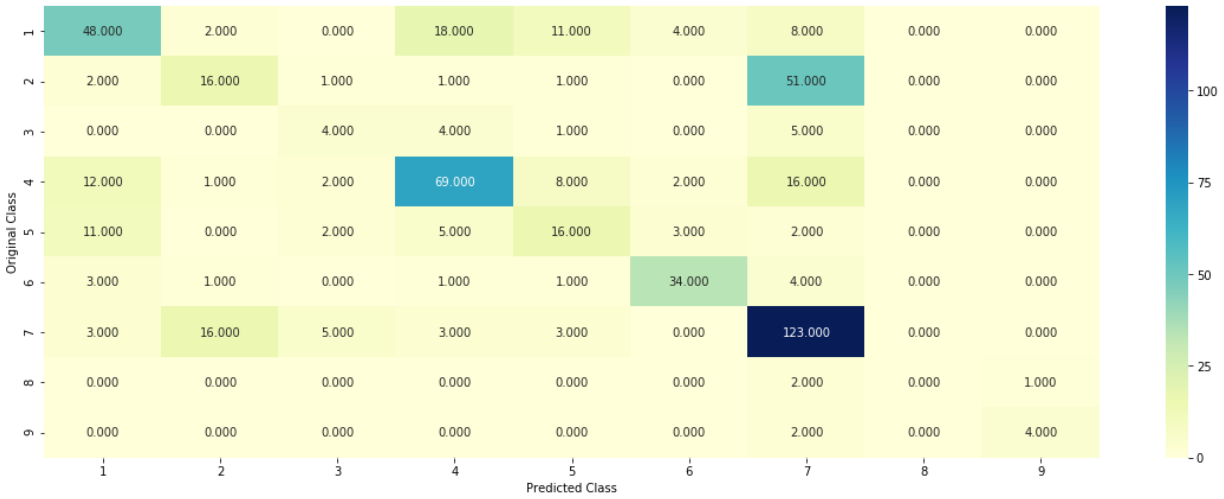
```
for alpha = 0.1
Log Loss : 1.238724352694891
for alpha = 1
Log Loss : 1.2981965106312383
for alpha = 10
Log Loss : 1.3671953606254361
for alpha = 100
Log Loss : 1.384057233940249
```




```
For values of best alpha =  0.01 The train log loss is: 0.7084704067457185
For values of best alpha =  0.01 The cross validation log loss is: 1.2087757084
22102
For values of best alpha =  0.01 The test log loss is: 1.1918133446495311
```

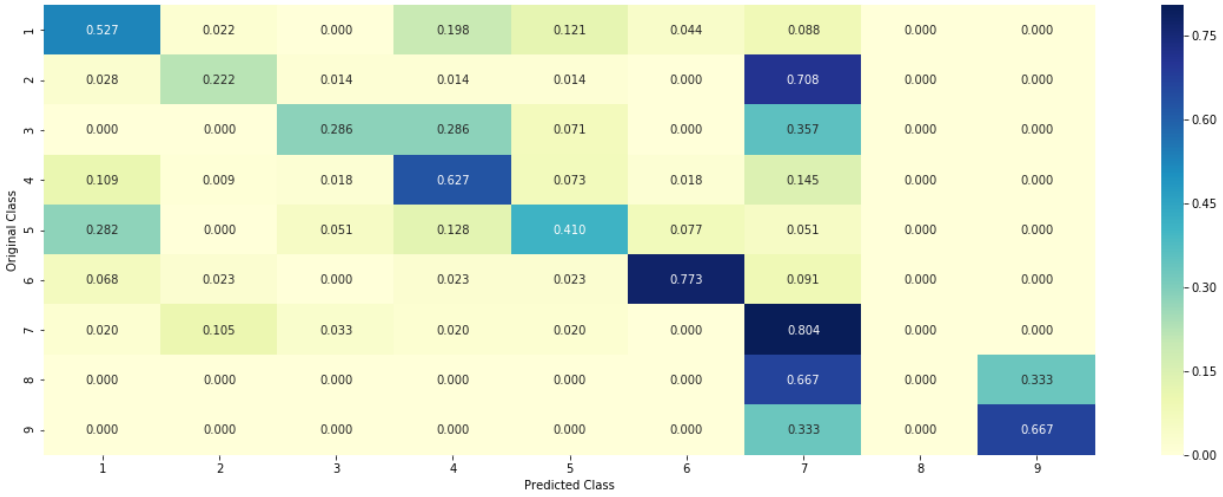
In [129]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCodi
```

```
Log loss : 1.208775708422102
Number of mis-classified points : 0.40977443609022557
-------------------- Confusion matrix --------------------
```




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

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




Even after applying the unigram and bigrams, log loss wasnt reducing much. Let's do some feature engineering and try

**Doing feature engineering.**

**Gene Feature**

In [0]:
```
result = pd.merge(data, data_text,on='ID', how='left')
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variatio
y_true = result['Class'].values
result.Gene = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

X_train, X_test, Y_train, Y_test = train_test_split(result, y_true, stratify=y_tr

X_train, X_cv, Y_train, Y_cv = train_test_split(X_train, Y_train, stratify=Y_trai
```

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

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

In [159]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in t
text_vectorizer = TfidfVectorizer()
train_text_feature_onehotCoding = text_vectorizer.fit_transform(X_train['TEXT'].a
train_text_features= text_vectorizer.get_feature_names()

train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

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 : 123389
```

In [0]:
```
test_text_feature_onehotCoding = text_vectorizer.transform(X_test['TEXT'].astype
cv_text_feature_onehotCoding = text_vectorizer.transform(X_cv['TEXT'].astype(str
```

In [0]:
```
gen_var = []
for i in data['Gene'].values:
    gen_var.append(i)
for j in data['Variation'].values:
    gen_var.append(j)
```

In [0]:
```
tf = TfidfVectorizer(max_features=1000)
t = tf.fit_transform(gen_var)
gene_variation_features = tf.get_feature_names()

train_text = tf.transform(X_train['TEXT'].astype(str))
test_text = tf.transform(X_test['TEXT'].astype(str))
cv_text = tf.transform(X_cv['TEXT'].astype(str))
```

In [0]:
```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_varia
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variati
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_fea

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

# Adding the test_text feature
test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text))
test_x_onehotCoding = hstack((test_x_onehotCoding, test_text_feature_onehotCoding
test_y = np.array(list(X_test['Class']))

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

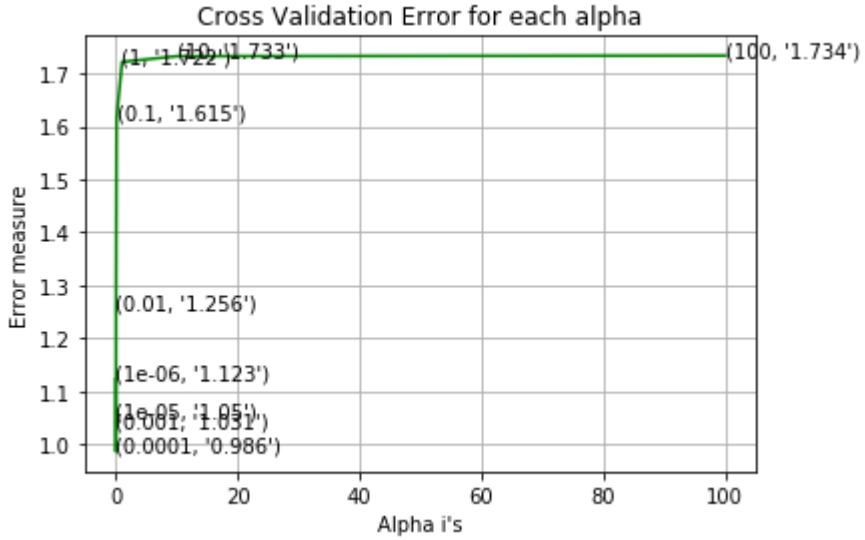
In [165]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_o
print("(number of data points * number of features) in test data = ", test_x_onel
print("(number of data points * number of features) in cross validation data =",
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 126585)
(number of data points * number of features) in test data =  (665, 126585)
(number of data points * number of features) in cross validation data = (532, 1
26585)
```

In [166]:
```
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-probabil
    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:",
      log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))

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

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

```
for alpha = 1e-06
Log Loss : 1.1226231138490614
for alpha = 1e-05
Log Loss : 1.0499177703965068
for alpha = 0.0001
Log Loss : 0.9855520134873146
for alpha = 0.001
Log Loss : 1.0310888752893532
for alpha = 0.01
```

```
Log Loss : 1.2563557351700816
for alpha = 0.1
Log Loss : 1.6154626892027597
for alpha = 1
Log Loss : 1.7222023992607902
for alpha = 10
Log Loss : 1.732956962258766
for alpha = 100
Log Loss : 1.7340974780495473
```

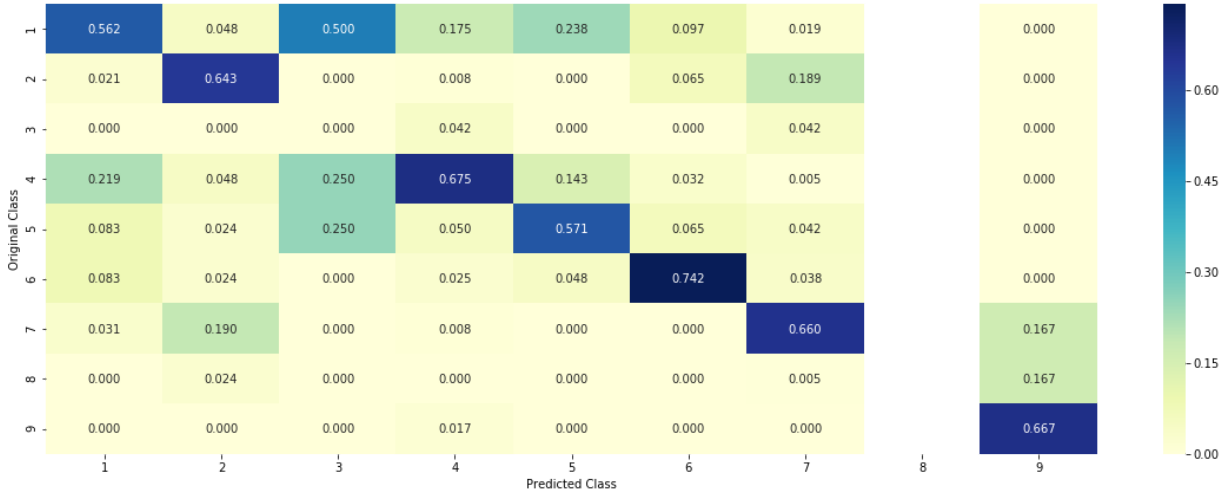



```
For values of best alpha =  0.0001 The train log loss is: 2.929842426071655
For values of best alpha =  0.0001 The cross validation log loss is: 2.64592611
9059635
For values of best alpha =  0.0001 The test log loss is: 2.8360572451484836
```

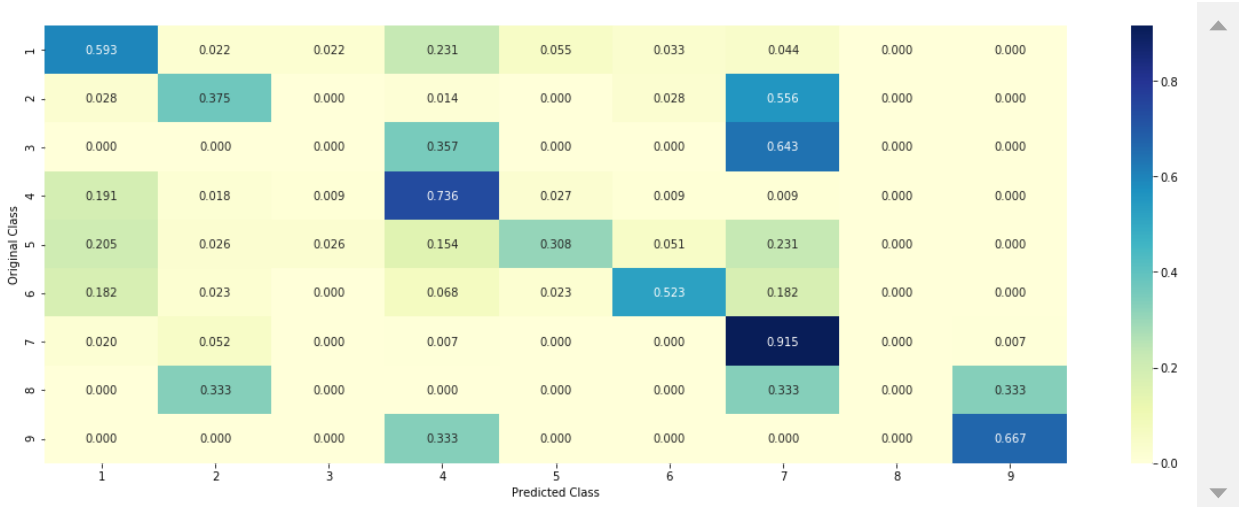
In [167]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCodi
```

```
Log loss : 0.9855520134873146
Number of mis-classified points : 0.35902255639097747
-------------------- Confusion matrix --------------------
```


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




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



In [0]:
```
### With 4 grams on text and applying logistic regression
```

**Gene Feature**

In [0]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer(ngram_range=(1,2))
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'])

# don't forget to normalize every feature
train_gene_feature_onehotCoding = normalize(train_gene_feature_onehotCoding, axi
test_gene_feature_onehotCoding = normalize(test_gene_feature_onehotCoding, axis=
cv_gene_feature_onehotCoding = normalize(cv_gene_feature_onehotCoding, axis=0)
```

**Variation Feature**

In [0]:
```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer(ngram_range=(1, 2))
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Va
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variat

# don't forget to normalize every feature
train_variation_feature_onehotCoding = normalize(train_variation_feature_onehotC
test_variation_feature_onehotCoding = normalize(test_variation_feature_onehotCod
cv_variation_feature_onehotCoding = normalize(cv_variation_feature_onehotCoding,
```

**Text Feature**

In [149]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in t
text_vectorizer = CountVectorizer(min_df=3,ngram_range=(3,4))
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
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 ti
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 : 2283223
```

In [0]:
```
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axi

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

# we use the same vectorizer that was trained on train data
cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'].astype('U
cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding, axis=0)
```

**Stack above three features**

In [0]:
```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_varia
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variati
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_fea

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

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

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

In [152]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_o
print("(number of data points * number of features) in test data = ", test_x_onel
print("(number of data points * number of features) in cross validation data =",
```

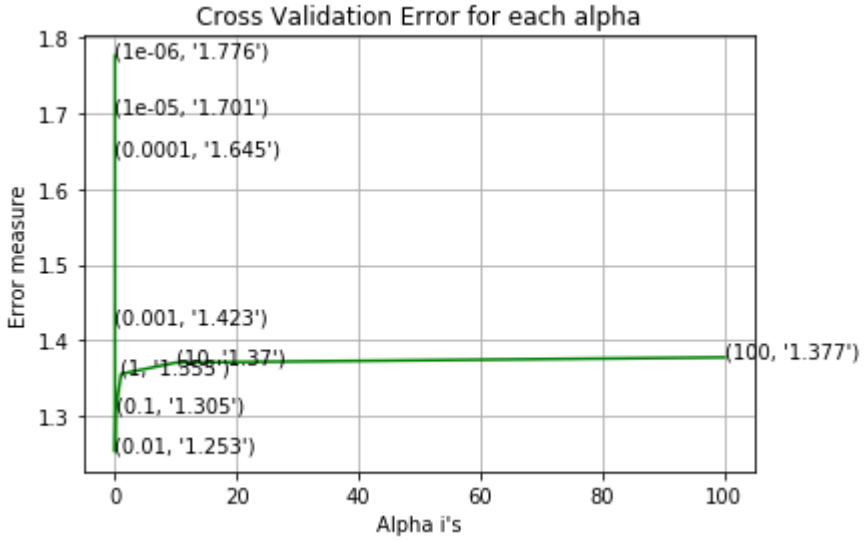
```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 2285486)
(number of data points * number of features) in test data =  (665, 2285486)
(number of data points * number of features) in cross validation data = (532, 2
285486)
```

**Applyting Logistic Regression**

In [153]:
```
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-probabil
    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:",
      log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))

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

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

```
for alpha = 1e-06
Log Loss : 1.7761953423905177
for alpha = 1e-05
Log Loss : 1.701377253023982
for alpha = 0.0001
Log Loss : 1.6448061911263334
for alpha = 0.001
Log Loss : 1.4228666245392652
for alpha = 0.01
Log Loss : 1.2527986414716488
```

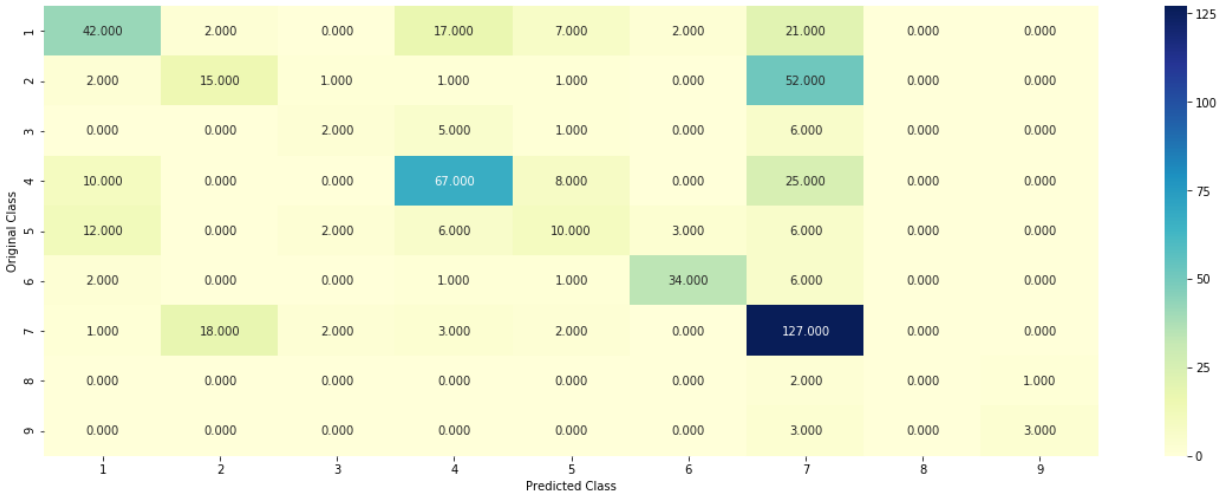
```
for alpha = 0.1
Log Loss : 1.305467958204696
for alpha = 1
Log Loss : 1.3554860967277031
for alpha = 10
Log Loss : 1.3704572398010795
for alpha = 100
Log Loss : 1.377362259743024
```



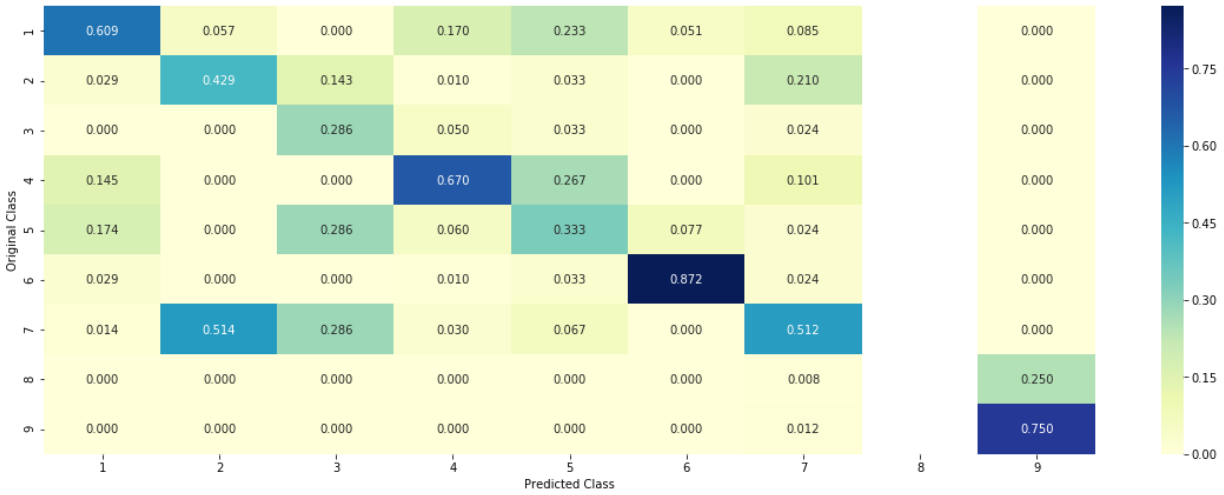

```
For values of best alpha =  0.01 The train log loss is: 0.8242059757923704
For values of best alpha =  0.01 The cross validation log loss is: 1.2527986414
716488
For values of best alpha =  0.01 The test log loss is: 1.2451852063989521
```

In [154]:
```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCodi
```

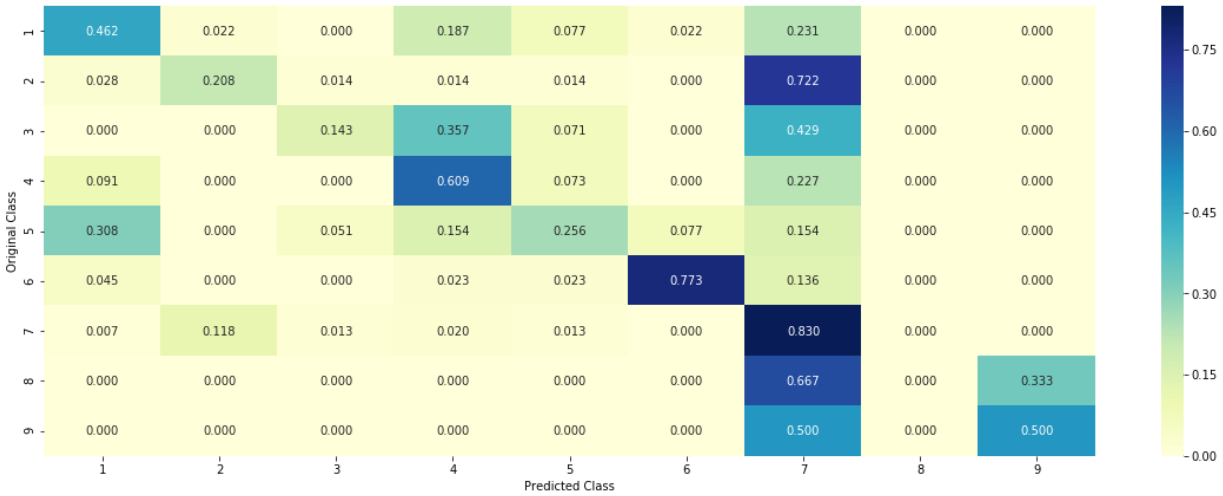
```
Log loss : 1.2527986414716488
Number of mis-classified points : 0.43609022556390975
-------------------- Confusion matrix --------------------
```




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



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




Finally for first feature engineering one, we got AUC less than 1 i.e, 0.98