Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

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

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [ ]:
```
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
from sklearn.feature_extraction.text import TfidfVectorizer
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

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

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

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

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

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

3.1.2. Reading Text Data

In [3]:
```
# note the seprator in this file
data_text =pd.read_csv("training_text",sep="\|\|",engine="python",name
print('Number of data points : ', data_text.shape[0])
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

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

Out[3]:

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

3.1.3. Preprocessing of text

In [4]:
```
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))


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

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

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

In [5]:
```
#text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_ti
```

```
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 : 225.53517599999998 seconds
```

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

Out[6]:

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

In [7]:
```
result[result.isnull().any(axis=1)]
```

Out[7]:

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

In [8]:
```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+resul
```

In [9]:
```
result[result['ID']==1109]
```

Out[9]:

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

Feature Engineering of given data

In [10]:
```
# number of words in each column
result["no_of_words"] = result["TEXT"].apply(lambda x: len(x.split()))
result.head()
```

Out[10]:

	ID	Gene	Variation	Class	TEXT	no_of_words
0	0	FAM58A	Truncating Mutations	1	cyclin dependent kinases cdks regulate variety...	4370
1	1	CBL	W802*	2	abstract background non small cell lung cancer...	4139
2	2	CBL	Q249E	2	abstract background non small cell lung cancer...	4139
3	3	CBL	N454D	3	recent evidence demonstrated acquired uniparen...	3841
4	4	CBL	L399V	4	oncogenic mutations monomeric casitas b lineag...	4254

In [11]:
```
# number of characters in the column
result['no_of_characters'] = result['TEXT'].apply(lambda x: len(str(x)
result.head()
```

Out[11]:

	ID	Gene	Variation	Class	TEXT	no_of_words	no_of_characters
0	0	FAM58A	Truncating Mutations	1	cyclin dependent kinases cdks regulate variety...	4370	30836
1	1	CBL	W802*	2	abstract background non small cell lung cancer...	4139	27844
2	2	CBL	Q249E	2	abstract background non small cell lung cancer...	4139	27844
3	3	CBL	N454D	3	recent evidence demonstrated acquired uniparen...	3841	28093
4	4	CBL	L399V	4	oncogenic mutations monomeric casitas b lineag...	4254	31649

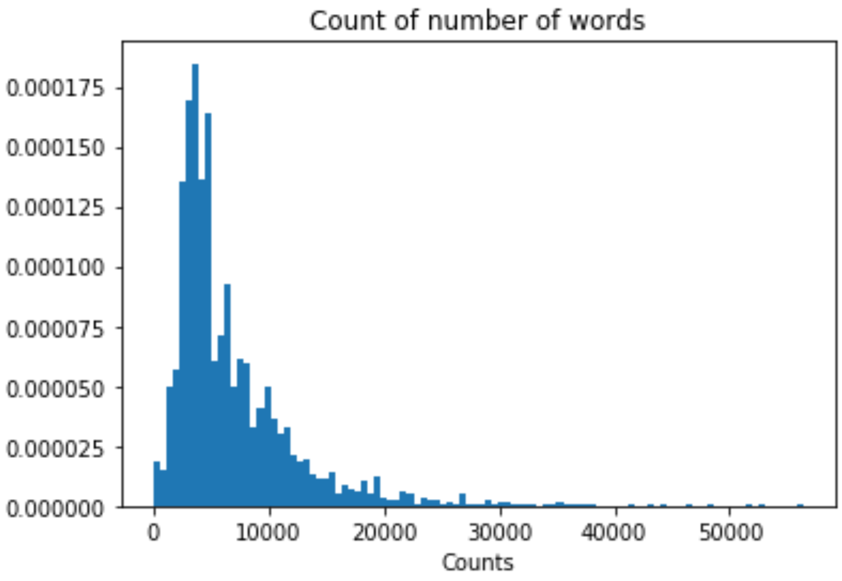
In [12]:
```
# combining the Gene column and Variation column
result['gene_variation'] = result['Gene'] + " " + result["Variation"]
result.head()
```

Out[12]:

	ID	Gene	Variation	Class	TEXT	no_of_words	no_of_characters	gene_variatio
0	0	FAM58A	Truncating Mutations	1	cyclin dependent kinases cdks regulate variety...	4370	30836	FAM58. Truncatin Mutation
1	1	CBL	W802*	2	abstract background non small cell lung cancer...	4139	27844	CBL W802
2	2	CBL	Q249E	2	abstract background non small cell lung cancer...	4139	27844	CBL Q249
3	3	CBL	N454D	3	recent evidence demonstrated acquired uniparen...	3841	28093	CBL N454I
4	4	CBL	L399V	4	oncogenic mutations monomeric casitas b lineag...	4254	31649	CBL L399

In [13]:
```
plt.hist(result["no_of_words"], normed=True, bins=100)
plt.xlabel('Counts');
plt.title("Count of number of words")
```

Out[13]:
```
Text(0.5,1,'Count of number of words')
```

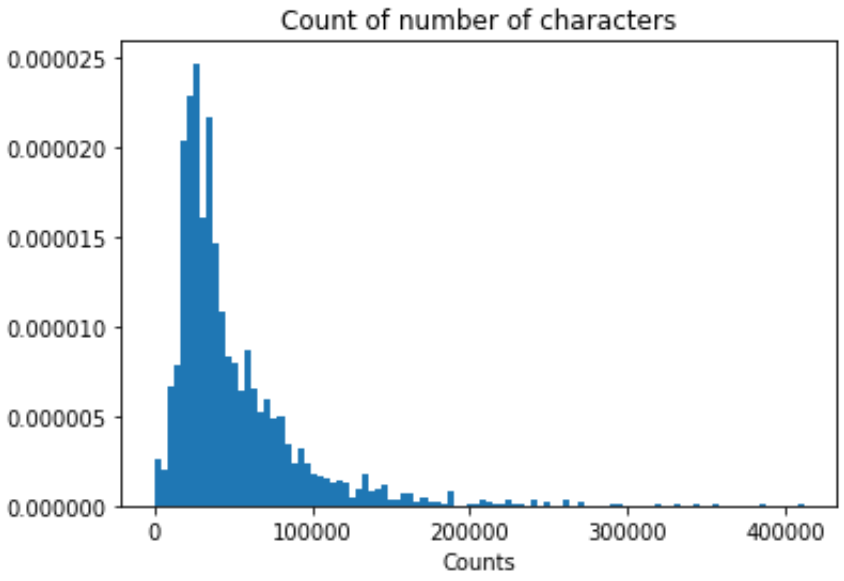


In [14]:
```
# we can also build a column of if the no. of words are greater than 5
result["word_count_of_5k"] = result["no_of_words"].apply(lambda x: 1 i
print("Head result",result.head(5))
print("****************************")
print("\nTail result",result.tail())
```

```
Head result    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

                                                TEXT  no_of_words  \
0  cyclin dependent kinases cdks regulate variety...         4370
1  abstract background non small cell lung cancer...         4139
2  abstract background non small cell lung cancer...         4139
3  recent evidence demonstrated acquired uniparen...         3841
4  oncogenic mutations monomeric casitas b lineag...         4254

   no_of_characters              gene_variation  word_count_of_5k
0             30836  FAM58A Truncating Mutations                0
1             27844                   CBL W802*                0
2             27844                   CBL Q249E                0
3             28093                   CBL N454D                0
4             31649                   CBL L399V                0
****************************

Tail result         ID   Gene Variation  Class  \
3316  3316  RUNX1     D171N      4
3317  3317  RUNX1     A122*      1
3318  3318  RUNX1   Fusions      1
3319  3319  RUNX1      R80C      4
3320  3320  RUNX1      K83E      4

                                                   TEXT  no_of_words
\
3316  introduction myelodysplastic syndromes mds het...         8153
3317  introduction myelodysplastic syndromes mds het...         4495
3318  runt related transcription factor 1 gene runx1...         4593
3319  runx1 aml1 gene frequent target chromosomal tr...         3465
3320  frequent mutations associated leukemia recurre...         7013

      no_of_characters gene_variation  word_count_of_5k
3316             57217    RUNX1 D171N                 1
3317             30800    RUNX1 A122*                 0
3318             28080  RUNX1 Fusions                 0
3319             25404     RUNX1 R80C                 0
3320             52582     RUNX1 K83E                 1
```

In [15]:
```
plt.hist(result["no_of_characters"], normed=True, bins=100)
plt.xlabel('Counts');
plt.title("Count of number of characters")
```

Out[15]:
```
Text(0.5,1,'Count of number of characters')
```



In [16]:
```
# we can also build a column of if the no. of characters are greater t
result["character_count_of_50k"] = result["no_of_characters"].apply(la
print("Head result",result.head(5))
print("****************************")
print("\nTail result",result.tail())
```

```
Head result    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

                                                TEXT  no_of_words  \
0  cyclin dependent kinases cdks regulate variety...         4370
1  abstract background non small cell lung cancer...         4139
2  abstract background non small cell lung cancer...         4139
3  recent evidence demonstrated acquired uniparen...         3841
4  oncogenic mutations monomeric casitas b lineag...         4254

   no_of_characters               gene_variation  word_count_of_5k
\
0             30836  FAM58A Truncating Mutations                 0
1             27844                    CBL W802*                 0
2             27844                    CBL Q249E                 0
3             28093                    CBL N454D                 0
4             31649                    CBL L399V                 0

   character_count_of_50k
0                       0
```

```
1                        0
2                        0
3                        0
4                        0
*****************************

Tail result        ID   Gene Variation  Class  \
3316  3316  RUNX1     D171N      4
3317  3317  RUNX1     A122*      1
3318  3318  RUNX1   Fusions      1
3319  3319  RUNX1      R80C      4
3320  3320  RUNX1      K83E      4

                                                    TEXT  no_of_words
\
3316  introduction myelodysplastic syndromes mds het...         8153
3317  introduction myelodysplastic syndromes mds het...         4495
3318  runt related transcription factor 1 gene runx1...         4593
3319  runx1 aml1 gene frequent target chromosomal tr...         3465
3320  frequent mutations associated leukemia recurre...         7013

      no_of_characters gene_variation  word_count_of_5k  \
3316             57217    RUNX1 D171N                 1
3317             30800    RUNX1 A122*                 0
3318             28080  RUNX1 Fusions                 0
3319             25404     RUNX1 R80C                 0
3320             52582     RUNX1 K83E                 1

      character_count_of_50k
3316                       1
3317                       0
3318                       0
3319                       0
3320                       1
```

In [17]:
```
result.columns.values
```

Out[17]:
```
array(['ID', 'Gene', 'Variation', 'Class', 'TEXT', 'no_of_words',
       'no_of_characters', 'gene_variation', 'word_count_of_5k',
       'character_count_of_50k'], dtype=object)
```

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 [18]:
```
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 o
X_train, test_df, y_train, y_test = train_test_split(result, y_true, st
# split the train data into train and cross validation by maintaining s
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, st
```

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

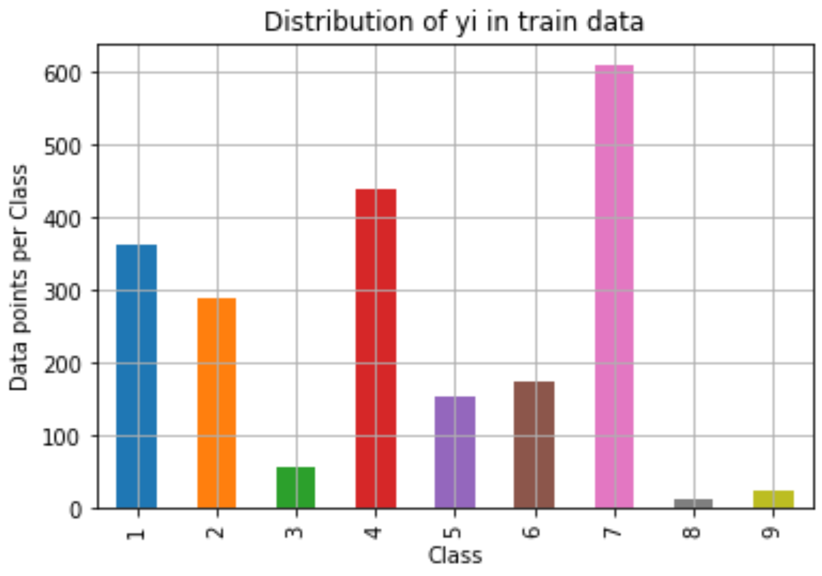
In [19]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0
```

```
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
```

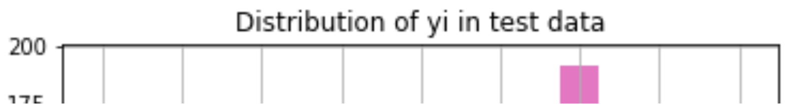
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

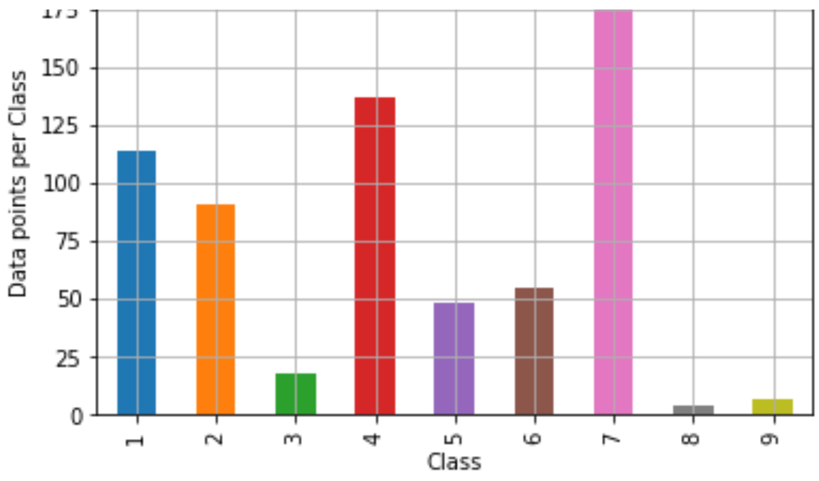
In [20]:
```
# it returns a dict, keys as class labels and values as the number of d
train_class_distribution = train_df['Class'].value_counts().sortlevel(
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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

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


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/nu
# -(train_class_distribution.values): the minus sign will give us in d
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distri

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/nu
# -(train_class_distribution.values): the minus sign will give us in d
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribu
```

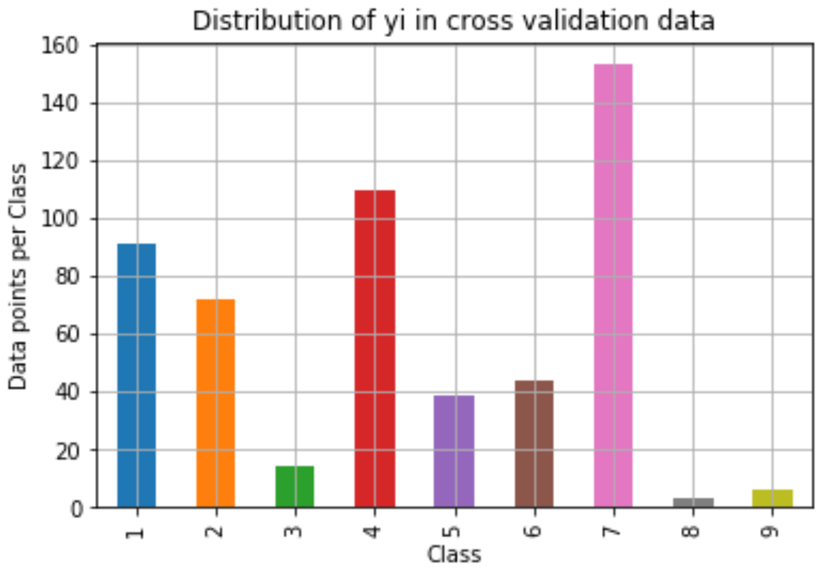


```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298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





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



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

3.2 Prediction using a 'Random' Model

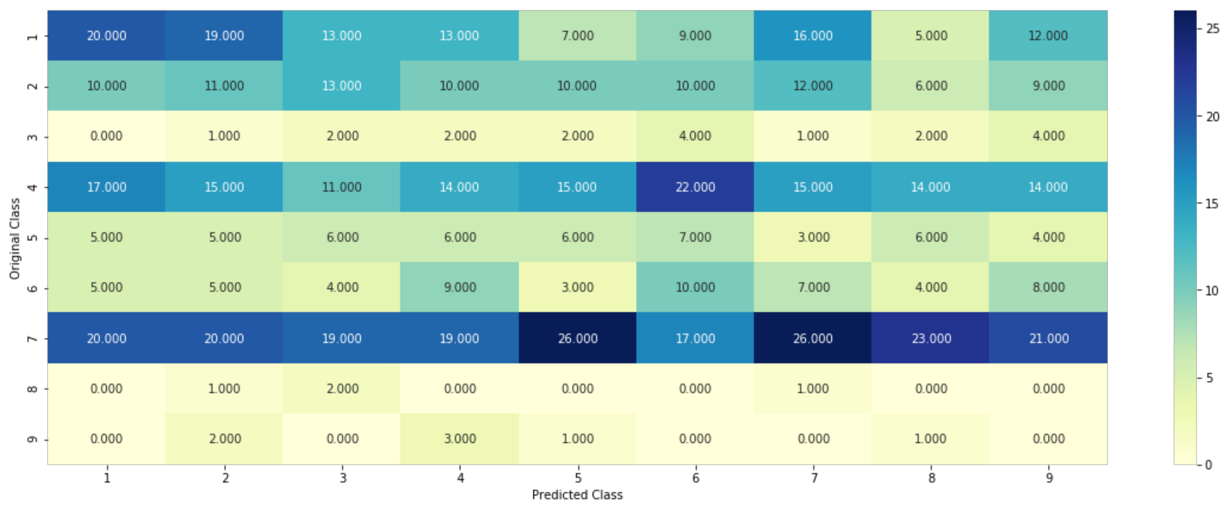
In a 'Random' Model, we generate the NINE class probabilites randomly such that they sum to 1.

In [21]:
```
# 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 c

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

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresp
    # 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 elemen
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresp
    # 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=la
    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=la
    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=la
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

In [22]:

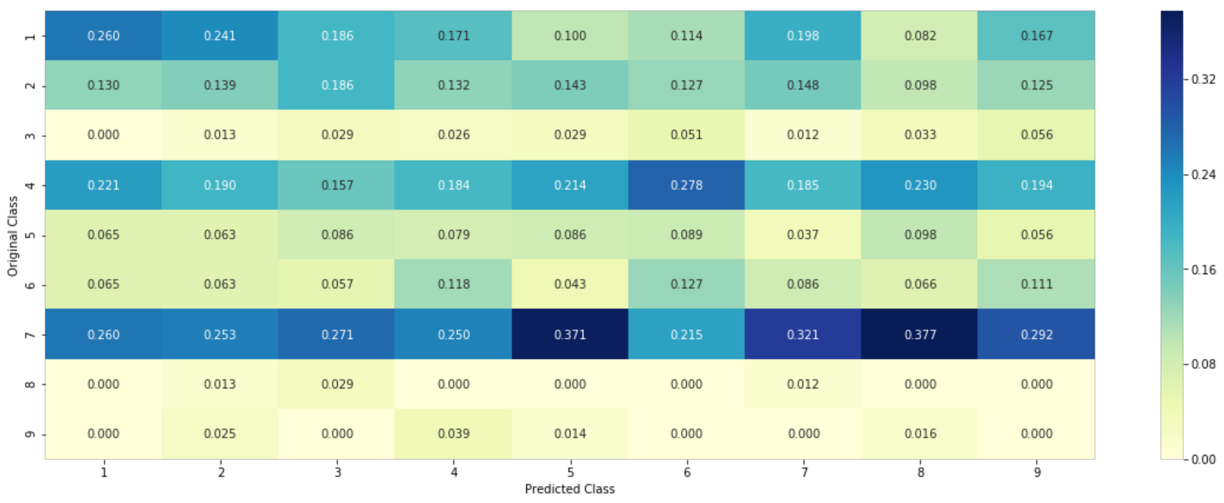
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# 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


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

```
Log loss on Cross Validation Data using Random Model 2.4640965314393
575
Log loss on Test Data using Random Model 2.446975397398525
-------------------- Confusion matrix --------------------
```

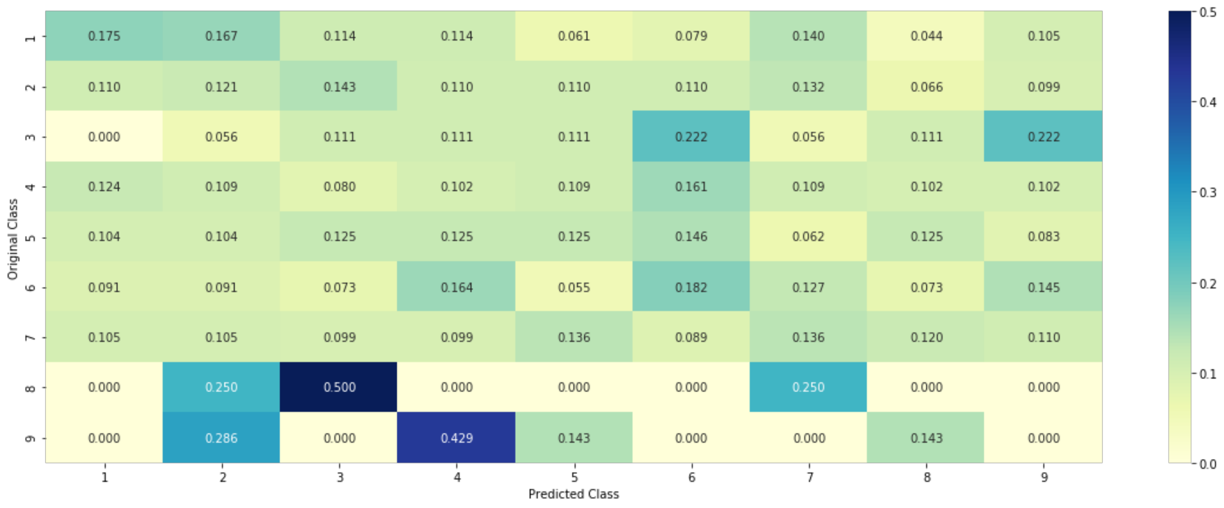


```
-------------------- Precision matrix (Columm Sum=1) ---------------
```

```
-----
```



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



3.3 Univariate Analysis

In [23]:
```
# 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 fea
# build a vector (1*9) , the first element = (number of times it occur
# gv_dict is like a look up table, for every gene it store a (1*9) rep
# 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_
# 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 ar
    gv_dict = dict()

    # denominator will contain the number of time that particular feat
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation 
        # vec is 9 diamensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['G
            #         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[

            # cls_cnt.shape[0](numerator) will contain the number of t
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 9

        # we are adding the gene/variation to the dict as key and vec
        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, 0.06818
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.06122
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625,
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.0606
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.0691
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.07333
    #      ...
    #     }
    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
    gv_fea = []
    # for every feature values in the given data frame we will check i
    # 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_f
    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?

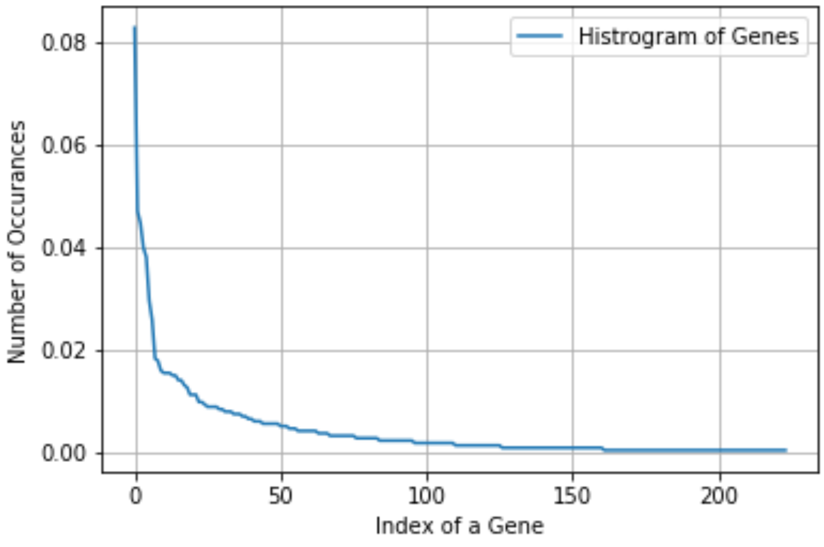
In [24]:
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 224
BRCA1     176
TP53      100
EGFR       95
BRCA2      85
PTEN       81
KIT        63
BRAF       55
ERBB2      39
ALK        38
FGFR2      34
Name: Gene, dtype: int64
```

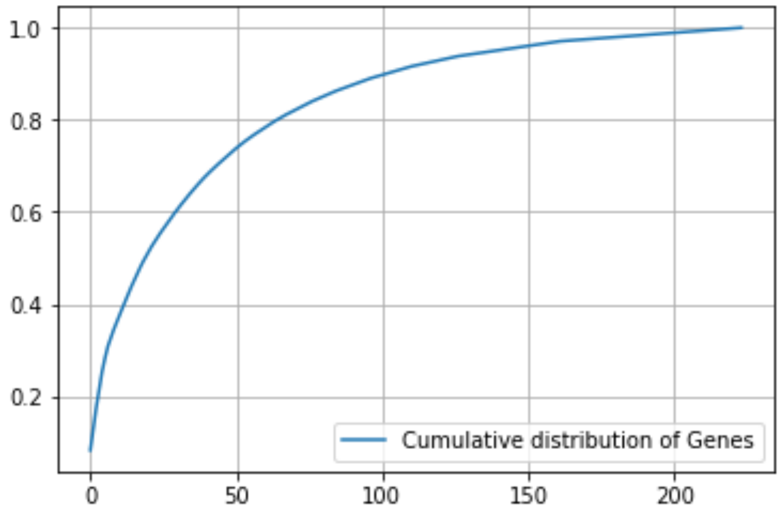
In [25]:
```
print("Ans: There are", unique_genes.shape[0] ,"different categories o
```

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

In [26]:
```
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 [27]:
```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

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

1. One hot Encoding
2. Response coding

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

In [28]:
```
#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, "Ge
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene"
```

In [29]:
```
print("train_gene_feature_responseCoding is converted feature using re
```

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

In [30]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer(max_features=220)
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Ge
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene']
```

In [31]:
```
train_gene_feature_onehotCoding.shape
```

Out[31]:
```
(2124, 220)
```

In [32]:
```
train_df['Gene'].head()
```

Out[32]:
```
12        CBL
192      EGFR
2558    BRCA1
1062    EWSR1
2395      NF1
Name: Gene, dtype: object
```

In [33]:
```
gene_vectorizer.get_feature_names()
```

Out[33]:
```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1b',
 'arid2',
 'atm',
 'atrx',
 'aurka',
 'axl',
 'b2m',
 'bap1',
 'bard1',
```

In [34]:
```
# creating a pandas dataframe of the vectorized features
df_gene_train = pd.DataFrame(train_gene_feature_onehotCoding.toarray()
df_gene_test = pd.DataFrame(test_gene_feature_onehotCoding.toarray(), 
df_gene_cv = pd.DataFrame(cv_gene_feature_onehotCoding.toarray(), colu
```

In [35]:
```
print("train_gene_feature_onehotCoding is converted feature using one-
```

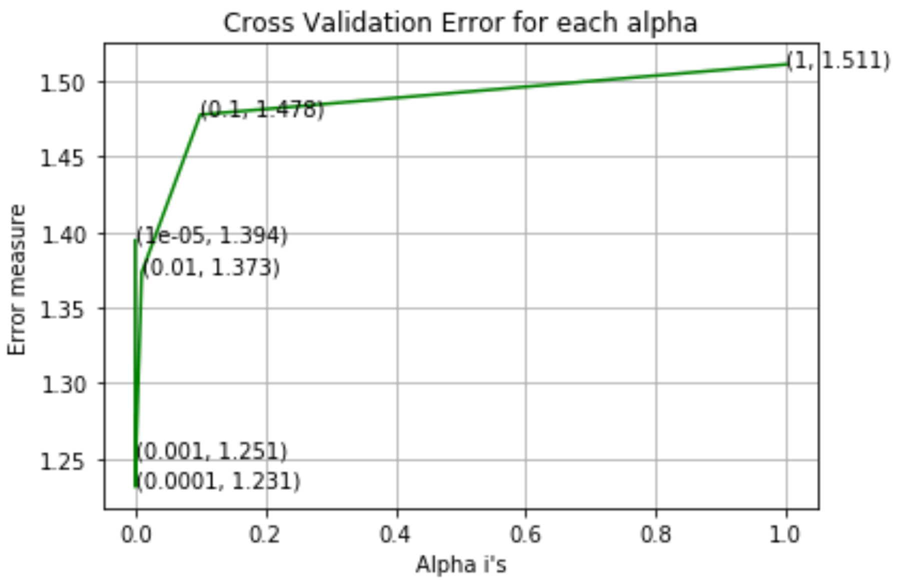
```
train_gene_feature_onehotCoding is converted feature using one-hot e
ncoding method. The shape of gene feature: (2124, 220)
```

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 [36]:
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifie

# read more about SGDClassifier() at http://scikit-learn.org/stable/mo
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, l
# 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 Stoc
# 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_stat
    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.cla
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_c

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_arr
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',
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
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross val
predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log
```

```
For values of alpha =  1e-05 The log loss is: 1.3944084716286247
For values of alpha =  0.0001 The log loss is: 1.231383844235684
For values of alpha =  0.001 The log loss is: 1.2512813436315235
For values of alpha =  0.01 The log loss is: 1.3731077811425252
For values of alpha =  0.1 The log loss is: 1.4776878210761122
For values of alpha =  1 The log loss is: 1.510967003941564
```



```
For values of best alpha =  0.0001 The train log loss is: 1.04615541
41926563
For values of best alpha =  0.0001 The cross validation log loss is:
1.231383844235684
For values of best alpha =  0.0001 The test log loss is: 1.209865659
1038013
```

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 [37]:
```
print("Q6. How many data points in Test and CV datasets are covered by

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

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

```
Q6. How many data points in Test and CV datasets are covered by the
224  genes in train dataset?
Ans
1. In test data 645 out of 665 : 96.99248120300751
2. In cross validation data 507 out of  532 : 95.30075187969925
```

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?

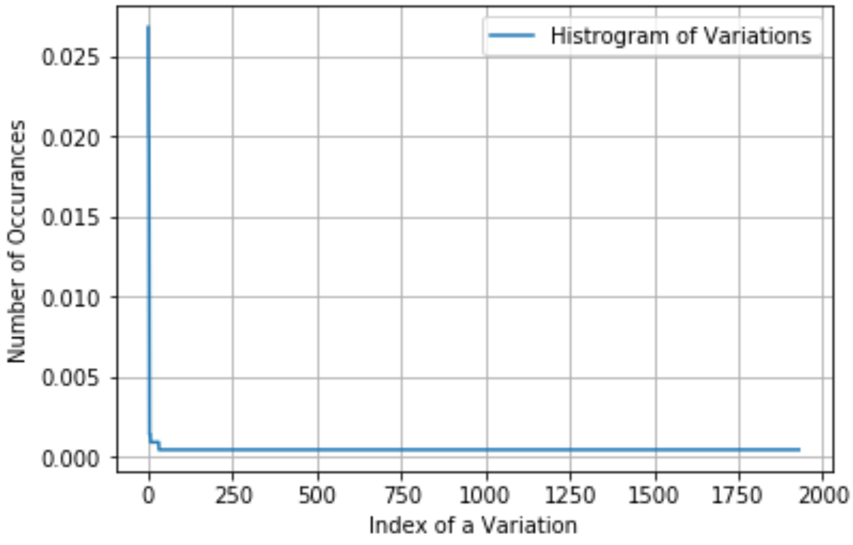
In [38]:
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1930
Truncating_Mutations    57
Deletion                43
Amplification           42
Fusions                 25
T58I                     3
Overexpression           3
G12V                     3
R170W                    2
Q61L                     2
Y64A                     2
Name: Variation, dtype: int64
```

In [39]:
```
print("Ans: There are", unique_variations.shape[0] ,"different categor
```

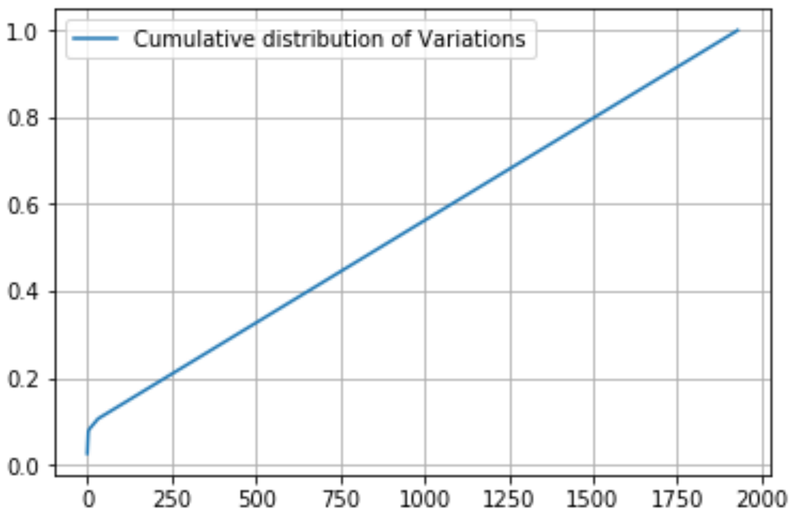
```
Ans: There are 1930 different categories of variations in the train
data, and they are distibuted as follows
```

In [40]:
```
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 [41]:
```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.02683616 0.04708098 0.06685499 ... 0.99905838 0.99952919 1.
]
```



Q9. How to featurize this Variation feature ?

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

1. One hot Encoding
2. Response coding

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

In [42]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "
```

In [43]:
```
print("train_variation_feature_responseCoding is a converted feature us
```

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

In [44]:
```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer(max_features=1700)
train_variation_feature_onehotCoding = variation_vectorizer.fit_transfo
test_variation_feature_onehotCoding = variation_vectorizer.transform(te
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_c
```

In [45]:
```
df_var_train = pd.DataFrame(train_variation_feature_onehotCoding.toarra
df_var_test = pd.DataFrame(test_variation_feature_onehotCoding.toarray
df_var_cv = pd.DataFrame(cv_variation_feature_onehotCoding.toarray(), c
```

In [46]:
```
print("train_variation_feature_onehotEncoded is converted feature usin
```

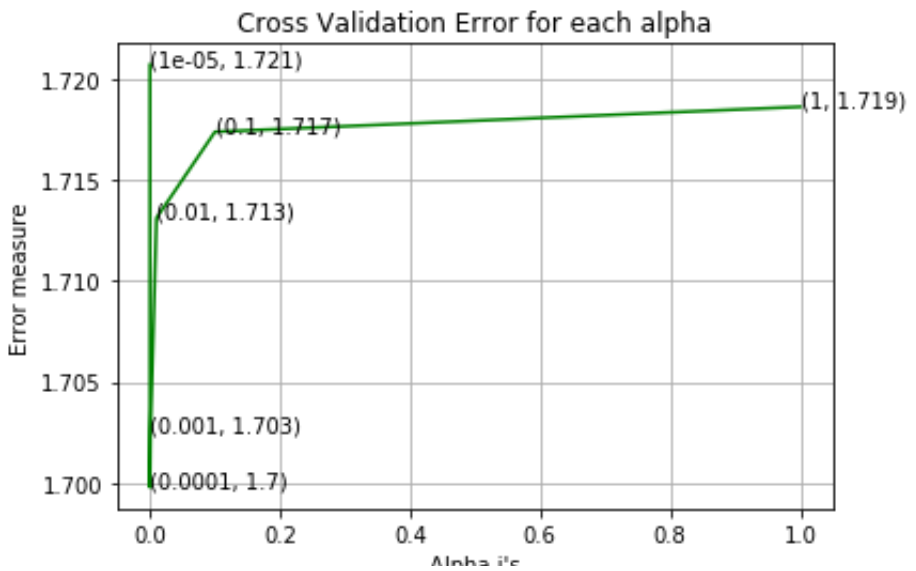
```
train_variation_feature_onehotEncoded is converted feature using the
onne-hot encoding method. The shape of Variation feature: (2124, 170
0)
```

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

Let's build a model just like the earlier!

In [47]:
```
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
# 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 Stoc
# 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
    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_onehotCodin

    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.cla
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_c

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_arr
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',
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
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross val
predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log
```

```
For values of alpha =  1e-05 The log loss is: 1.7207189013959114
For values of alpha =  0.0001 The log loss is: 1.6997689084309093
For values of alpha =  0.001 The log loss is: 1.7026016503896444
For values of alpha =  0.01 The log loss is: 1.7131165722610504
For values of alpha =  0.1 The log loss is: 1.7173932048592393
For values of alpha =  1 The log loss is: 1.7186310786492505
```



```
For values of best alpha =  0.0001 The train log loss is: 0.86508446
86325048
For values of best alpha =  0.0001 The cross validation log loss is:
1.6997689084309093
For values of best alpha =  0.0001 The test log loss is: 1.703251480
2236773
```

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 [48]:
```
print("Q12. How many data points are covered by total ", unique_variat
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Var
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0],
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape
```

```
Q12. How many data points are covered by total  1930  genes in test
and cross validation data sets?
Ans
1. In test data 69 out of 665 : 10.37593984962406
2. In cross validation data 55 out of  532 : 10.338345864661653
```

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 [49]:
```
# cls_text is a data frame
# for every row in data fram consider the 'TEXT'
# split the words by space
# make a dict with those words
# increment its count whenever we see that word

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

In [50]:
```
import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(
            text_feature_responseCoding[row_index][i] = math.exp(sum_pr
            row_index += 1
    return text_feature_responseCoding
```

In [51]:
```
# building a CountVectorizer with all the words that occured minimum 3
text_vectorizer = TfidfVectorizer(min_df=5,ngram_range=(1,2),max_featu
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_
# 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
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 nu
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_coun


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

```
Total number of unique words in train data : 20000
```

In [52]:
```
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 [53]:
```
#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 [54]:
```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding
test_text_feature_responseCoding = (test_text_feature_responseCoding.T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_
```

In [55]:
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotC

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

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

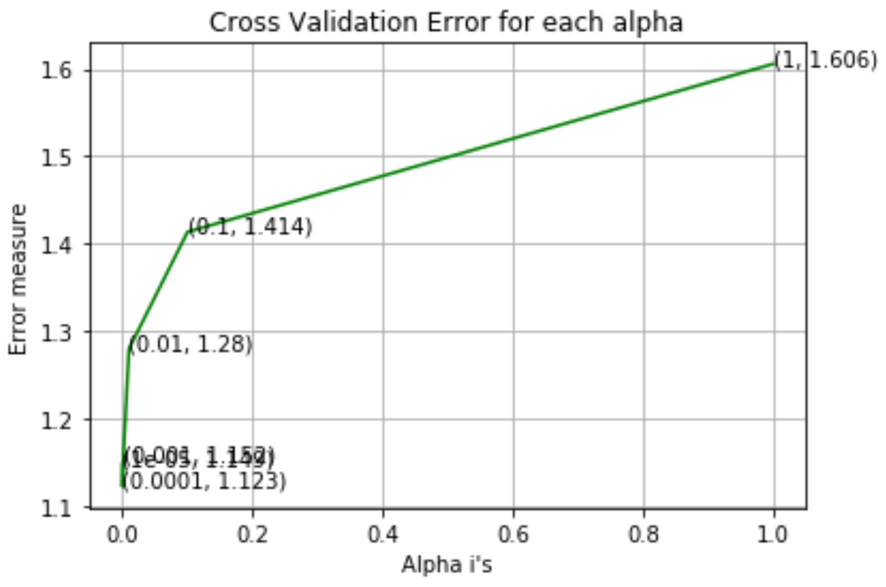
In [56]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

In [57]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({0.5928400952859082: 39, 0.7544202940499756: 32, 0.666516722
1822955: 26, 0.6388896632493938: 20, 0.8886889629097272: 16, 1.06481
61054156562: 15, 0.9085949753176206: 14, 1.0389694259244064: 10, 3.1
937793216260233: 8, 0.851852884332525: 8, 0.7415078640131119: 7, 0.6
975390741666331: 7, 0.889260142928862: 6, 0.6641917020380229: 6, 0.5
572043451611401: 6, 2.589247760039847: 5, 2.1296322108313124: 5, 1.4
907425475819174: 5, 1.0836875989439445: 5, 0.9421724920023471: 5, 0.
6171172984548712: 5, 0.5800558825457663: 5, 4.487153349100253: 4, 2.
422919934180324: 4, 1.621224820902286: 4, 1.514324958862703: 4, 1.50
88405880999511: 4, 1.333033444364591: 4, 1.2982056046452644: 4, 1.21
1459967090162: 4, 1.1108612036371595: 4, 0.753737993601878: 4, 0.694
994699185837: 4, 0.6686452141933684: 4, 0.5255716752962893: 4, 0.522
4245169756063: 4, 0.4785805665793877: 4, 2.7685218740807045: 3, 2.63
46823125623207: 3, 2.342595431914442: 3, 1.8174878465033697: 3, 1.69
59104856042255: 3, 1.555205685092022: 3, 1.2777793264987876: 3, 1.13
16304410749625: 3, 1.0294102356167: 3, 0.91333191791486: 3, 0.817018
3652277363: 3, 0.7378770862301951: 3, 0.7273698269507515: 3, 0.72167
13210917822: 3, 0.6980113222458281: 3, 0.6491028023226322: 3, 0.6123
100131839002: 3, 0.5988436571197083: 3, 0.5559957593486697: 3, 0.532
[illegible]
```

In [58]:
```
# Train a Logistic regression+Calibration model using text features wh
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/mo
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, l
# 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 Stoc
# 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_stat
    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.cla
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_c
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_arra
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',
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
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross val
predict_y = sig_clf.predict_proba(test_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log
```

```
For values of alpha =  1e-05 The log loss is: 1.149056113623601
For values of alpha =  0.0001 The log loss is: 1.122895305061825
For values of alpha =  0.001 The log loss is: 1.152449660058874
For values of alpha =  0.01 The log loss is: 1.2802353839118785
For values of alpha =  0.1 The log loss is: 1.4137492341435582
For values of alpha =  1 The log loss is: 1.605535246074835
```



```
For values of best alpha =  0.0001 The train log loss is: 0.86219088
58537646
For values of best alpha =  0.0001 The cross validation log loss is:
1.122895305061825
For values of best alpha =  0.0001 The test log loss is: 1.131701569
```

```
2145014
```

In [59]:
```
df_text_train = pd.DataFrame(train_text_feature_onehotCoding.toarray()
df_text_test = pd.DataFrame(test_text_feature_onehotCoding.toarray(), 
df_text_cv = pd.DataFrame(cv_text_feature_onehotCoding.toarray(), colu
```

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

Ans. Yes, it seems like!

In [60]:
```
def get_intersec_text(df):
    df_text_vec = TfidfVectorizer(min_df=3)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

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

In [61]:
```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared i
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation app
```

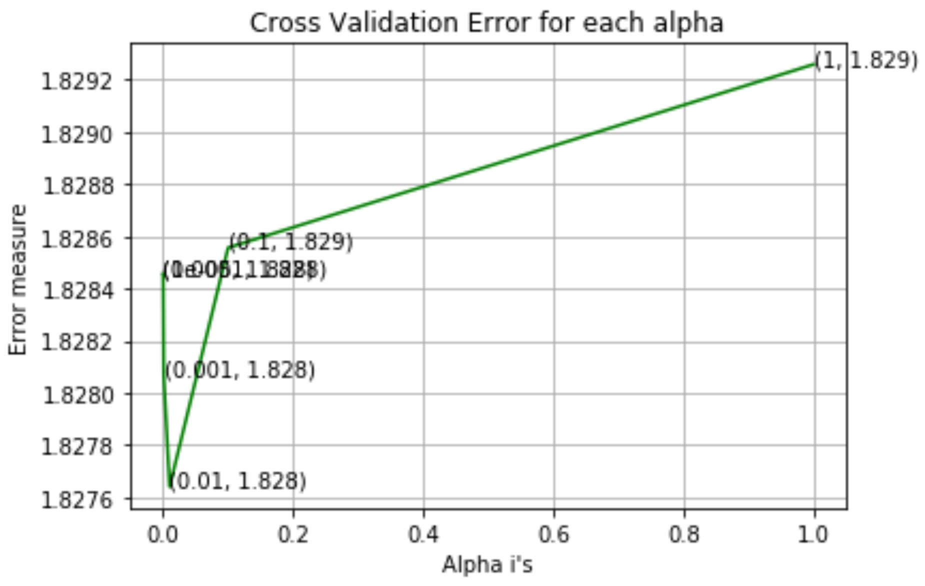
```
30.223 % of word of test data appeared in train data
35.099 % of word of Cross Validation appeared in train data
```

Univariate analysis of no_of_words

In [62]:
```
alpha = [10 ** x for x in range(-5, 1)]


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_stat
    clf.fit(train_df["no_of_words"].values.reshape(-1,1), y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_df.no_of_words.values.reshape(-1,1), y_train)
    predict_y = sig_clf.predict_proba(cv_df.no_of_words.values.reshape

    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.cla
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_c

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_arra
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',
clf.fit(train_df.no_of_words.values.reshape(-1,1), y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_df.no_of_words.values.reshape(-1,1), y_train)

predict_y = sig_clf.predict_proba(train_df.no_of_words.values.reshape(
print('For values of best alpha = ', alpha[best_alpha], "The train log
predict_y = sig_clf.predict_proba(cv_df.no_of_words.values.reshape(-1,
print('For values of best alpha = ', alpha[best_alpha], "The cross val
predict_y = sig_clf.predict_proba(test_df.no_of_words.values.reshape(-
print('For values of best alpha = ', alpha[best_alpha], "The test log
```

```
For values of alpha =  1e-05 The log loss is: 1.828456718086271
For values of alpha =  0.0001 The log loss is: 1.8284567180762197
For values of alpha =  0.001 The log loss is: 1.8280692399388123
For values of alpha =  0.01 The log loss is: 1.8276398372891882
For values of alpha =  0.1 The log loss is: 1.8285563158071414
For values of alpha =  1 The log loss is: 1.8292590645707076
```

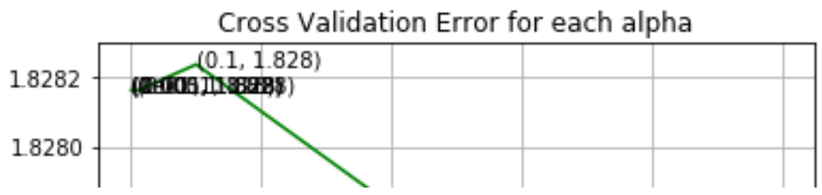


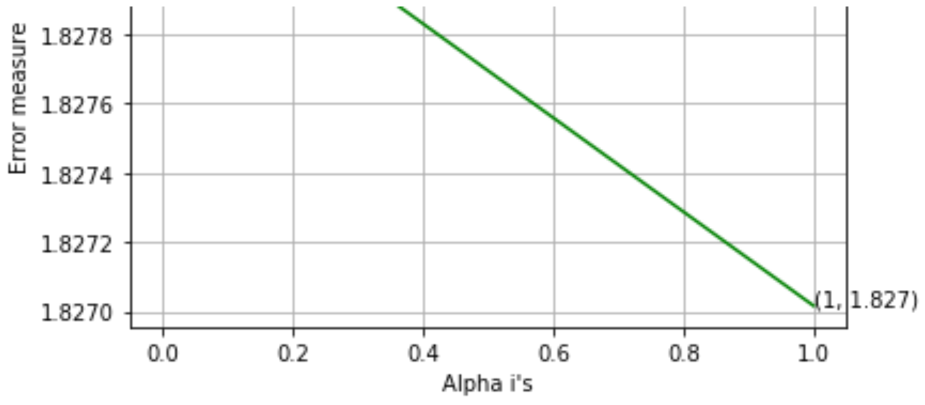
```
For values of best alpha =  0.01 The train log loss is: 1.8123028601
858309
For values of best alpha =  0.01 The cross validation log loss is: 1
.8276398372891882
For values of best alpha =  0.01 The test log loss is: 1.81123765184
82927
```

Univariate analysis of no_of_characters

In [63]:
```
alpha = [10 ** x for x in range(-5, 1)]


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_stat
    clf.fit(train_df["no_of_characters"].values.reshape(-1,1), y_train

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_df.no_of_characters.values.reshape(-1,1), y_trai
    predict_y = sig_clf.predict_proba(cv_df.no_of_characters.values.re

    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.cla
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_c

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_arr
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',
clf.fit(train_df.no_of_characters.values.reshape(-1,1), y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_df.no_of_characters.values.reshape(-1,1), y_train)

predict_y = sig_clf.predict_proba(train_df.no_of_characters.values.res
print('For values of best alpha = ', alpha[best_alpha], "The train log
predict_y = sig_clf.predict_proba(cv_df.no_of_characters.values.reshap
print('For values of best alpha = ', alpha[best_alpha], "The cross val
predict_y = sig_clf.predict_proba(test_df.no_of_characters.values.resh
print('For values of best alpha = ', alpha[best_alpha], "The test log
```

```
For values of alpha =  1e-05 The log loss is: 1.828162256019388
For values of alpha =  0.0001 The log loss is: 1.8281622560192532
For values of alpha =  0.001 The log loss is: 1.8281622559915802
For values of alpha =  0.01 The log loss is: 1.8281622560558215
For values of alpha =  0.1 The log loss is: 1.828236977767708
For values of alpha =  1 The log loss is: 1.8270168837018603
```





```
For values of best alpha =  1 The train log loss is: 1.8089394348503
06
For values of best alpha =  1 The cross validation log loss is: 1.82
70168837018603
For values of best alpha =  1 The test log loss is: 1.80826891851075
92
```

Univariate analysis of gene_variation

In [64]:
```
gene_var = train_df['gene_variation'].value_counts()
print('Count of Gene and Variation which are unique :', gene_var.shape
# the top 10 variations that occured most
print("Head",gene_var.head(10))
print("Tail",gene_var.head(10))
```

```
Count of Gene and Variation which are unique : 2124
Head SMAD3 S425C    1
TET2 R1896M    1
MTOR D2512G    1
CTNNB1 G34E    1
KIT D816E      1
TSC2 R611W     1
RASA1 Y472H    1
ALK D1349H     1
ALK S1206R     1
IDH2 R172K     1
Name: gene_variation, dtype: int64
Tail SMAD3 S425C    1
TET2 R1896M    1
MTOR D2512G    1
CTNNB1 G34E    1
KIT D816E      1
TSC2 R611W     1
RASA1 Y472H    1
ALK D1349H     1
ALK S1206R     1
IDH2 R172K     1
Name: gene_variation, dtype: int64
```

In [65]:
```
# Featurizing the Gene_and_Variation Feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_and_variation_feature_responseCoding = np.array(get_gv_featu
# test gene feature
test_gene_and_variation_feature_responseCoding = np.array(get_gv_featu
# cross validation gene feature
cv_gene_and_variation_feature_responseCoding = np.array(get_gv_feature
```

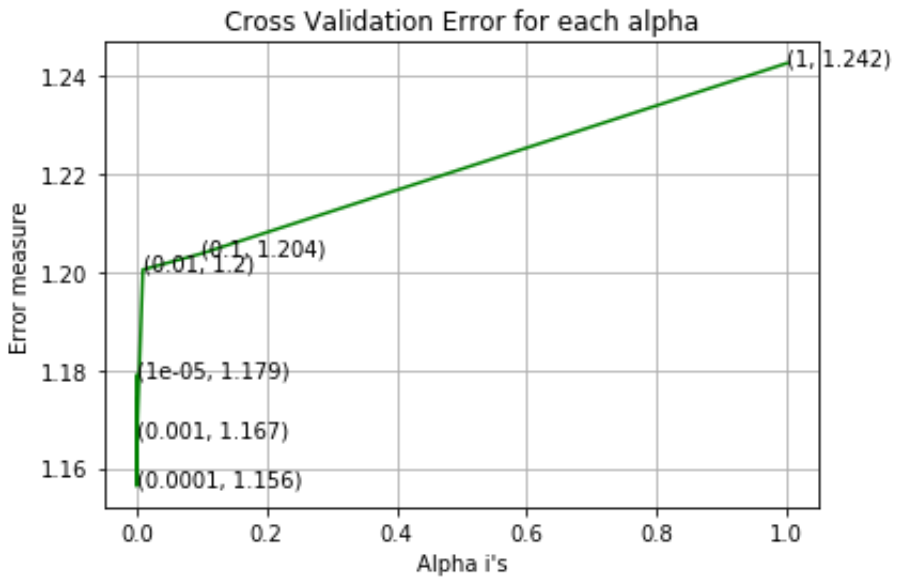
In [66]:
```
# one-hot encoding of gene_and_variation feature.
gene_variation_vectorizer = TfidfVectorizer()
train_gene_and_variation_feature_onehotCoding = gene_variation_vectori
test_gene_and_variation_feature_onehotCoding = gene_variation_vectoriz
cv_gene_and_variation_feature_onehotCoding = gene_variation_vectorizer
```

In [67]:
```
df_geneandvar_train = pd.DataFrame(train_gene_and_variation_feature_one
df_geneandvar_test = pd.DataFrame(test_gene_and_variation_feature_oneh
df_geneandvar_cv = pd.DataFrame(cv_gene_and_variation_feature_onehotCo
```

In [68]:
```
alpha = [10 ** x for x in range(-5, 1)]


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state
    clf.fit(df_geneandvar_train, y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(df_geneandvar_train, y_train)
    predict_y = sig_clf.predict_proba(df_geneandvar_cv)

    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.clas
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv

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_arra
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',
clf.fit(df_geneandvar_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(df_geneandvar_train, y_train)
```

```
predict_y = sig_clf.predict_proba(df_geneandvar_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log
predict_y = sig_clf.predict_proba(df_geneandvar_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross val
predict_y = sig_clf.predict_proba(df_geneandvar_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log
```

```
For values of alpha =  1e-05 The log loss is: 1.1788850909646191
For values of alpha =  0.0001 The log loss is: 1.1564038096282738
For values of alpha =  0.001 The log loss is: 1.1666389694783292
For values of alpha =  0.01 The log loss is: 1.2004413194107715
For values of alpha =  0.1 The log loss is: 1.2037014208637031
For values of alpha =  1 The log loss is: 1.2424739088579682
```

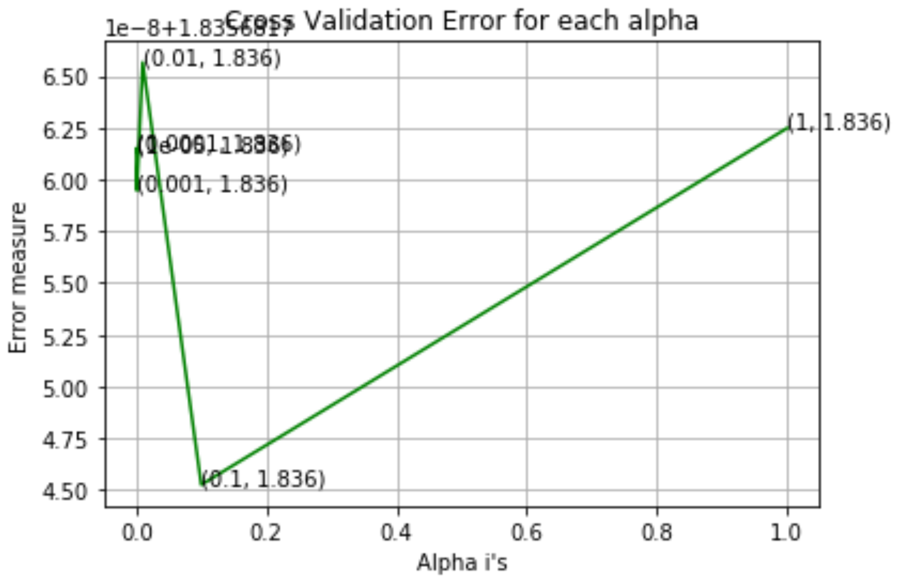


```
For values of best alpha =  0.0001 The train log loss is: 0.50735322
79878828
For values of best alpha =  0.0001 The cross validation log loss is:
1.1564038096282738
For values of best alpha =  0.0001 The test log loss is: 1.113590880
0719008
```

Univariate analysis of word_count_of_5k

In [69]:
```
alpha = [10 ** x for x in range(-5, 1)]


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state
    clf.fit(train_df["word_count_of_5k"].values.reshape(-1,1), y_train

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_df.word_count_of_5k.values.reshape(-1,1), y_trai
    predict_y = sig_clf.predict_proba(cv_df.word_count_of_5k.values.res
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.cla
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv

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_arra
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',
clf.fit(train_df.word_count_of_5k.values.reshape(-1,1), y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_df.word_count_of_5k.values.reshape(-1,1), y_train)

predict_y = sig_clf.predict_proba(train_df.word_count_of_5k.values.res
print('For values of best alpha = ', alpha[best_alpha], "The train log
predict_y = sig_clf.predict_proba(cv_df.word_count_of_5k.values.reshap
print('For values of best alpha = ', alpha[best_alpha], "The cross val
predict_y = sig_clf.predict_proba(test_df.word_count_of_5k.values.resh
print('For values of best alpha = ', alpha[best_alpha], "The test log
```

```
For values of alpha =  1e-05 The log loss is: 1.8356817613570136
For values of alpha =  0.0001 The log loss is: 1.8356817615055787
For values of alpha =  0.001 The log loss is: 1.8356817594697372
For values of alpha =  0.01 The log loss is: 1.8356817656528432
For values of alpha =  0.1 The log loss is: 1.8356817452536163
For values of alpha =  1 The log loss is: 1.8356817624766348
```



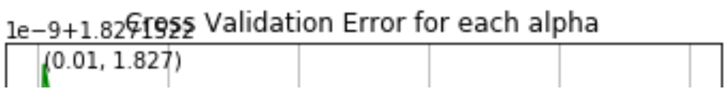
```
For values of best alpha =  0.1 The train log loss is: 1.81826136004
60846
For values of best alpha =  0.1 The cross validation log loss is: 1.
8356817452536163
```

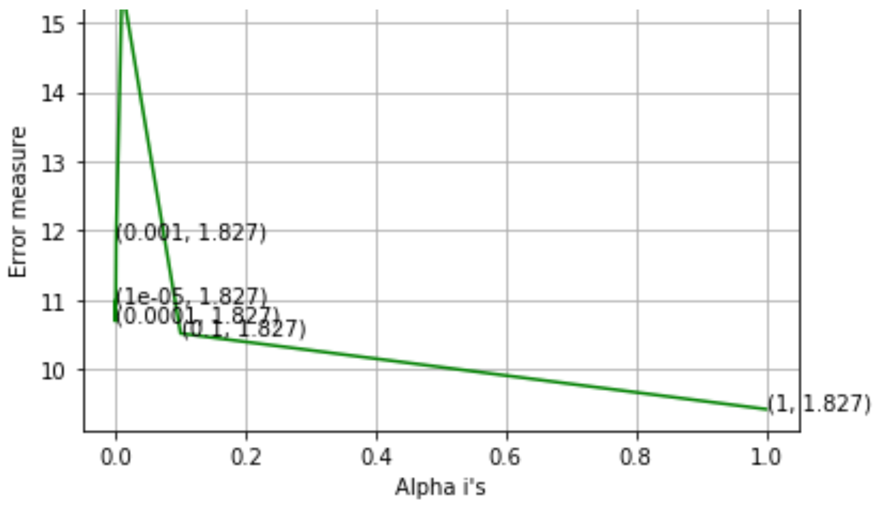
```
For values of best alpha =  0.1 The test log loss is: 1.812748250682
8
```

Univariate analysis of character_count_of_50k

In [70]:
```
alpha = [10 ** x for x in range(-5, 1)]


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state
    clf.fit(train_df["character_count_of_50k"].values.reshape(-1,1), y_

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_df.character_count_of_50k.values.reshape(-1,1), y
    predict_y = sig_clf.predict_proba(cv_df.character_count_of_50k.valu

    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.clas
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv

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_arra
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',
clf.fit(train_df.character_count_of_50k.values.reshape(-1,1), y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_df.character_count_of_50k.values.reshape(-1,1), y_tra

predict_y = sig_clf.predict_proba(train_df.character_count_of_50k.value
print('For values of best alpha = ', alpha[best_alpha], "The train log
predict_y = sig_clf.predict_proba(cv_df.character_count_of_50k.values.r
print('For values of best alpha = ', alpha[best_alpha], "The cross val
predict_y = sig_clf.predict_proba(test_df.character_count_of_50k.values
print('For values of best alpha = ', alpha[best_alpha], "The test log l
```

```
For values of alpha =  1e-05 The log loss is: 1.8271522109805236
For values of alpha =  0.0001 The log loss is: 1.8271522106894227
For values of alpha =  0.001 The log loss is: 1.8271522119019332
For values of alpha =  0.01 The log loss is: 1.8271522155237467
For values of alpha =  0.1 The log loss is: 1.8271522105135294
For values of alpha =  1 The log loss is: 1.8271522094197878
```





```
For values of best alpha =  1 The train log loss is: 1.8105178137952
094
For values of best alpha =  1 The cross validation log loss is: 1.82
71522094197878
For values of best alpha =  1 The test log loss is: 1.80827531152302
26
```

4. Machine Learning Models

In [71]:
```
#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.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 probabili
    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_
    plot_confusion_matrix(test_y, pred_y)
```

In [72]:
```
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 [110]:
```
# 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
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3,max_features=1000)

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

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

    word_present = 0
    for i,v in enumerate(indices):
        if (v < fea1_len):
            word = gene_vec.get_feature_names()[v]
            yes_no = True if word == gene else False
            if yes_no:
                word_present += 1
                print(i, "Gene feature [{}] present in test data point
        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
        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

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

Stacking the three types of features

In [74]:
```
#target variables
train_y = train_df['Class'].values
test_y = test_df['Class'].values
cv_y = cv_df['Class'].values

# concatenating all the vectorized dataframes
df_gene_var_train = pd.concat([df_gene_train, df_var_train], axis=1)
df_gene_var_test = pd.concat([df_gene_test, df_var_test], axis=1)
df_gene_var_cv = pd.concat([df_gene_cv, df_var_cv], axis=1)

df_gene_and_var_train = pd.concat([df_gene_var_train, df_geneandvar_tr
df_gene_and_var_test = pd.concat([df_gene_var_test, df_geneandvar_test
df_gene_and_var_cv = pd.concat([df_gene_var_cv, df_geneandvar_cv], axi

df_train = pd.concat([df_gene_and_var_train, df_text_train], axis=1)
df_test = pd.concat([df_gene_and_var_test, df_text_test], axis=1)
df_cv = pd.concat([df_gene_and_var_cv, df_text_cv], axis=1)
```

In [75]:
```
# scaling the text_count feature
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
train_df["no_of_words"] = scaler.fit_transform(train_df["no_of_words"]
test_df["no_of_words"] = scaler.fit_transform(test_df["no_of_words"].va
cv_df["no_of_words"] = scaler.fit_transform(cv_df["no_of_words"].values

train_df["no_of_characters"] = scaler.fit_transform(train_df["no_of_cha
test_df["no_of_characters"] = scaler.fit_transform(test_df["no_of_chara
cv_df["no_of_characters"] = scaler.fit_transform(cv_df["no_of_characte

train_df["word_count_of_5k"] = scaler.fit_transform(train_df["word_cou
test_df["word_count_of_5k"] = scaler.fit_transform(test_df["word_count_
cv_df["word_count_of_5k"] = scaler.fit_transform(cv_df["word_count_of_

train_df["character_count_of_50k"] = scaler.fit_transform(train_df["cha
test_df["character_count_of_50k"] = scaler.fit_transform(test_df["chara
cv_df["character_count_of_50k"] = scaler.fit_transform(cv_df["characte



df_train["no_of_words"] = train_df.no_of_words.values
df_train["no_of_characters"] = train_df.no_of_characters.values
df_train["word_count_of_5k"] = train_df.word_count_of_5k.values
df_train["character_count_of_50k"] = train_df.character_count_of_50k.va

df_test["no_of_words"] = test_df.no_of_words.values
df_test["no_of_characters"] = test_df.no_of_characters.values
df_test["word_count_of_5k"] = test_df.word_count_of_5k.values
df_test["character_count_of_50k"] = test_df.character_count_of_50k.valu


df_cv["no_of_words"] = cv_df.no_of_words.values
df_cv["no_of_characters"] = cv_df.no_of_characters.values
df_cv["word_count_of_5k"] = cv_df.word_count_of_5k.values
df_cv["character_count_of_50k"] = cv_df.character_count_of_50k.values
```

In [76]:
```
train_gene_var_responseCoding = np.hstack((train_gene_feature_response(
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCo
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding

train_geneandvar_responseCoding = np.hstack((train_gene_var_responseCo
test_geneandvar_responseCoding = np.hstack((test_gene_var_responseCodi
cv_geneandvar_responseCoding = np.hstack((cv_gene_var_responseCoding,c

train_x_responseCoding = np.hstack((train_geneandvar_responseCoding, t
test_x_responseCoding = np.hstack((test_geneandvar_responseCoding, tes
cv_x_responseCoding = np.hstack((cv_geneandvar_responseCoding, cv_text_

train_no_of_words = np.column_stack((train_x_responseCoding, train_df.
test_no_of_words = np.column_stack((test_x_responseCoding, test_df.no_
cv_no_of_words = np.column_stack((cv_x_responseCoding, cv_df.no_of_wor

train_no_of_characters = np.column_stack((train_no_of_words, train_df.
test_no_of_characters = np.column_stack((test_no_of_words, test_df.no_
cv_no_of_characters = np.column_stack((cv_no_of_words, cv_df.no_of_cha

train_word_count_of_5k = np.column_stack((train_no_of_characters, trai
test_word_count_of_5k = np.column_stack((test_no_of_characters, test_d
cv_word_count_of_5k = np.column_stack((cv_no_of_characters, cv_df.word_

train_x_response = np.column_stack((train_word_count_of_5k, train_df.c
test_x_response = np.column_stack((test_word_count_of_5k, test_df.char
cv_x_response = np.column_stack((cv_word_count_of_5k, cv_df.character_



train_x_onehotCoding = df_train
test_x_onehotCoding = df_test
cv_x_onehotCoding = df_cv

train_x_responseCoding = pd.DataFrame(train_x_response)
test_x_responseCoding = pd.DataFrame(test_x_response)
cv_x_responseCoding = pd.DataFrame(cv_x_response)
```

In [77]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ",
print("(number of data points * number of features) in test data = ", 
print("(number of data points * number of features) in cross validatio
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124,
24071)
(number of data points * number of features) in test data =  (665, 2
4071)
(number of data points * number of features) in cross validation dat
a = (532, 24071)
```

In [78]:
```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ",
print("(number of data points * number of features) in test data = ",
print("(number of data points * number of features) in cross validatio
```

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124,
40)
(number of data points * number of features) in test data =  (665, 4
0)
(number of data points * number of features) in cross validation dat
a = (532, 40)
```

4.1. Base Line Model

4.1.1. Naive Bayes

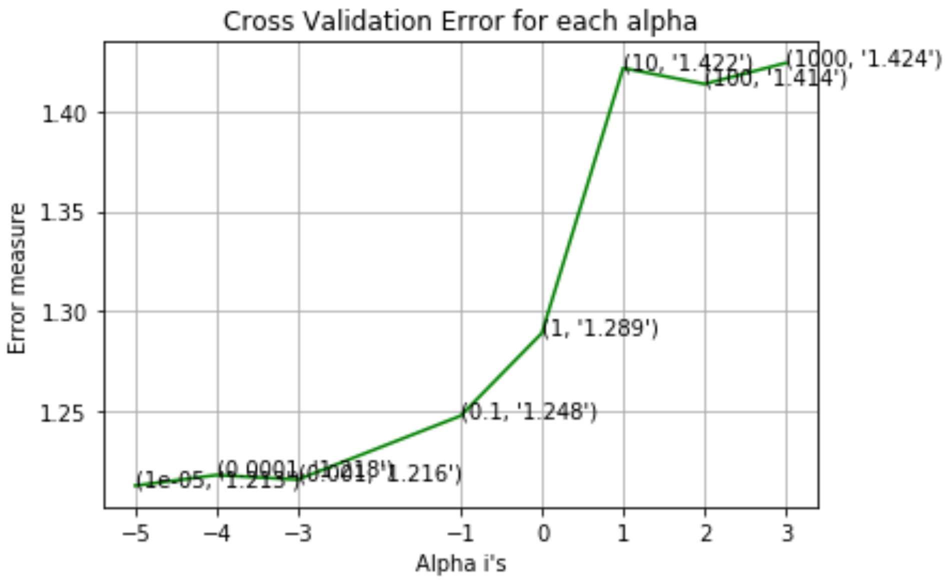
4.1.1.1. Hyper parameter tuning

In [79]:
```
# find more about Multinomial Naive base function here http://scikit-l
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_p

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


# find more about CalibratedClassifierCV here at http://scikit-learn.o
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, meth
#
# 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
# ------------------------


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
    # to avoid rounding error while multiplying probabilites we use lo
    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_
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
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross val
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log
```

```
for alpha = 1e-05
Log Loss : 1.2128577780836407
for alpha = 0.0001
Log Loss : 1.21817052556479
for alpha = 0.001
Log Loss : 1.2158982215448673
for alpha = 0.1
Log Loss : 1.2477365646690524
for alpha = 1
Log Loss : 1.2890814072732695
for alpha = 10
Log Loss : 1.4216474071762748
for alpha = 100
Log Loss : 1.4136778012386413
```

```
for alpha = 1000
Log Loss : 1.424206335067884
```

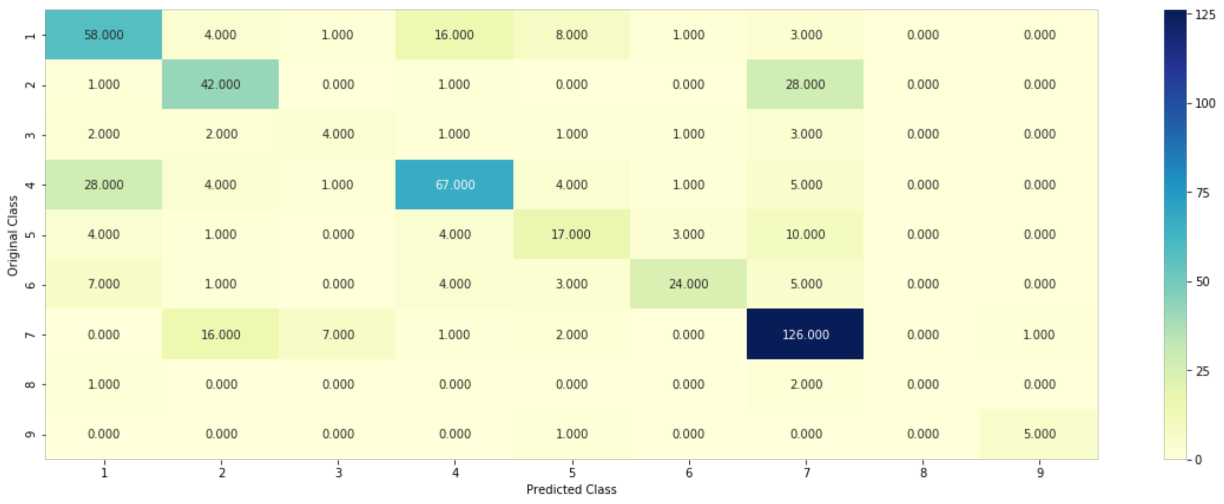


```
For values of best alpha =  1e-05 The train log loss is: 0.794534970
4848299
For values of best alpha =  1e-05 The cross validation log loss is:
1.2128577780836407
For values of best alpha =  1e-05 The test log loss is: 1.2321949417
796758
```

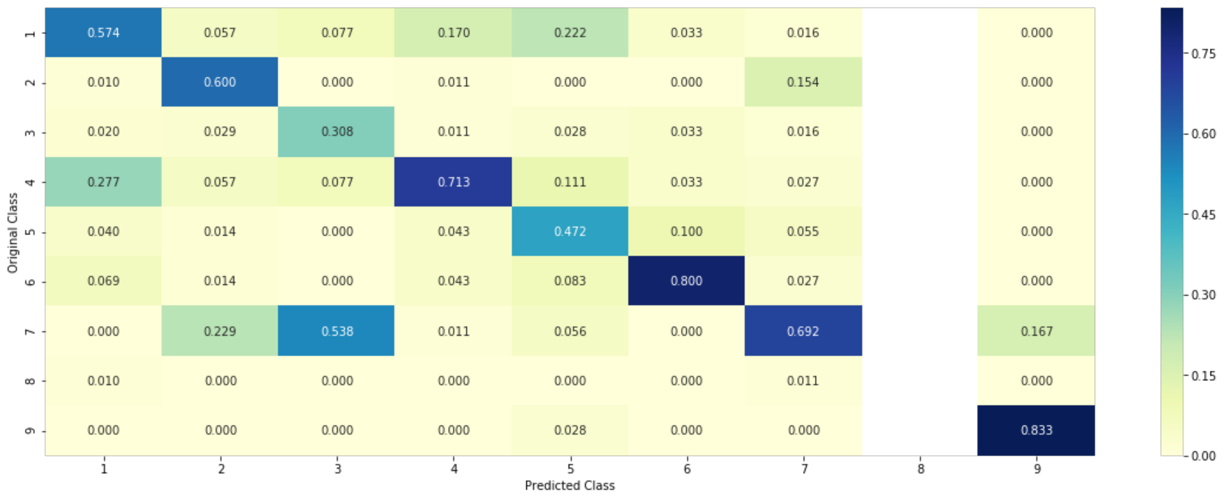
4.1.1.2. Testing the model with best hyper paramters

In [80]:
```
# find more about Multinomial Naive base function here http://scikit-l
# --------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_p

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


# find more about CalibratedClassifierCV here at http://scikit-learn.o
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, meth
#
# 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
# 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-pr
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.pr
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding))
```

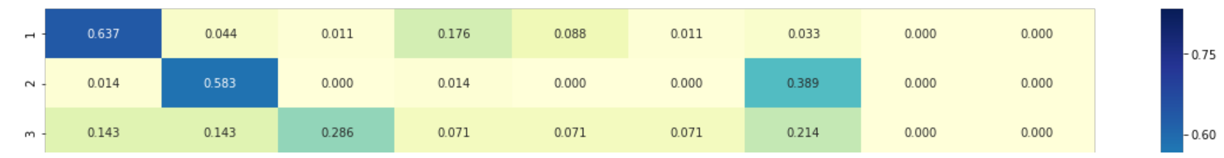
```
Log Loss : 1.2128577780836407
Number of missclassified point : 0.35526315789473684
-------------------- Confusion matrix --------------------
```

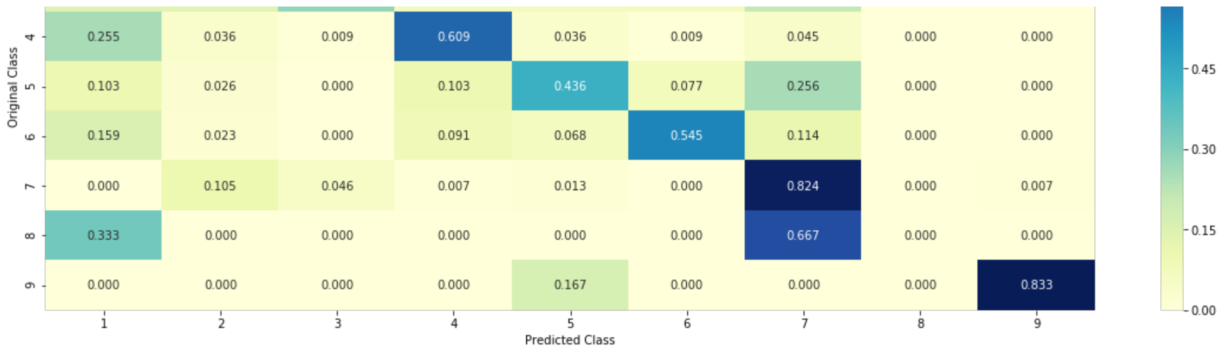


```
-------------------- Precision matrix (Columm Sum=1) ---------------
-----
```



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





4.1.1.3. Feature Importance, Correctly classified point

In [81]:
```
test_point_index = 12
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding.iloc[test_point_in
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0789 0.2437 0.0122 0.1006 0.0426
0.0374 0.4723 0.006  0.0062]]
Actual Class : 7
--------------------------------------------------
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [82]:
```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding.iloc[test_point_in
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

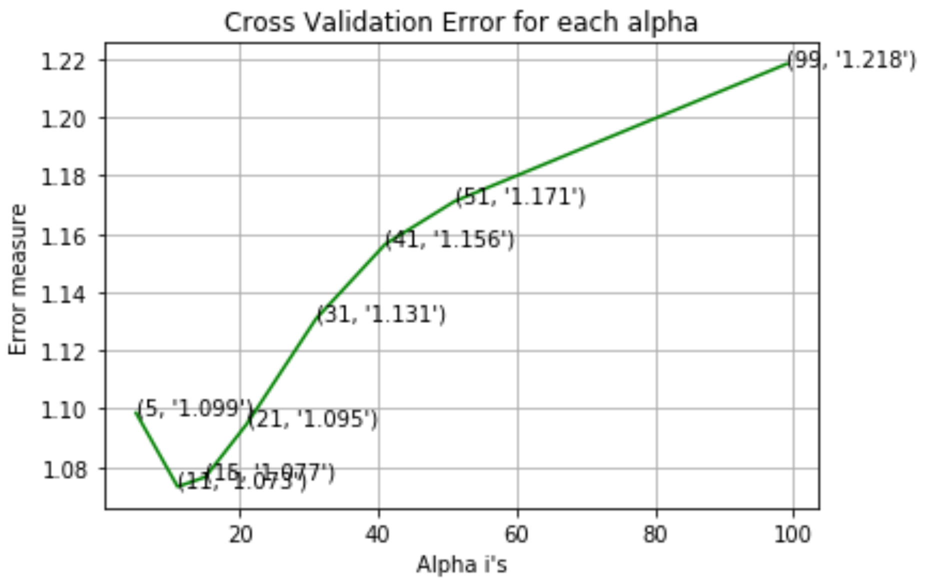
```
Predicted Class : 2
Predicted Class Probabilities: [[0.09   0.567  0.0135 0.1138 0.048
0.0422 0.1118 0.0068 0.007 ]]
Actual Class : 1
--------------------------------------------------
```

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

In [111]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='au
# 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 v
# 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
#--------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.o
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, meth
#
# 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
    # to avoid rounding error while multiplying probabilites we use lo
    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
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross val
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log
```

```
for alpha = 5
Log Loss : 1.0985511326771829
for alpha = 11
Log Loss : 1.0732328472930994
for alpha = 15
Log Loss : 1.076528634036376
for alpha = 21
Log Loss : 1.0947602662954434
for alpha = 31
Log Loss : 1.1308694422401118
for alpha = 41
Log Loss : 1.156478238520508
for alpha = 51
Log Loss : 1.171171764652043
for alpha = 99
Log Loss : 1.2183618899099915
```



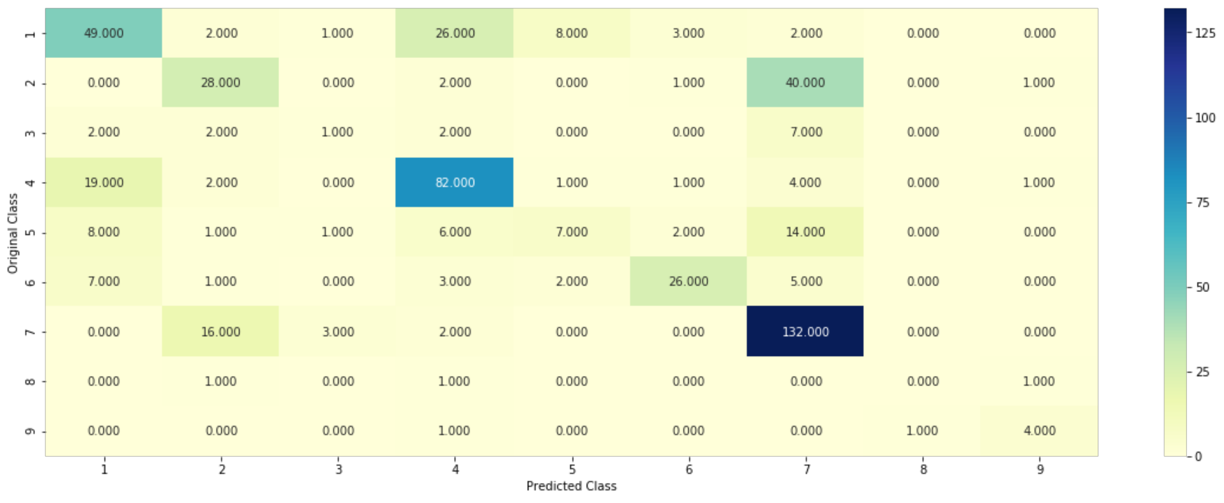
```
For values of best alpha =  11 The train log loss is: 0.757102919734
9849
```

```
For values of best alpha =  11 The cross validation log loss is: 1.0
732328472930994
For values of best alpha =  11 The test log loss is: 1.0983942729743
246
```

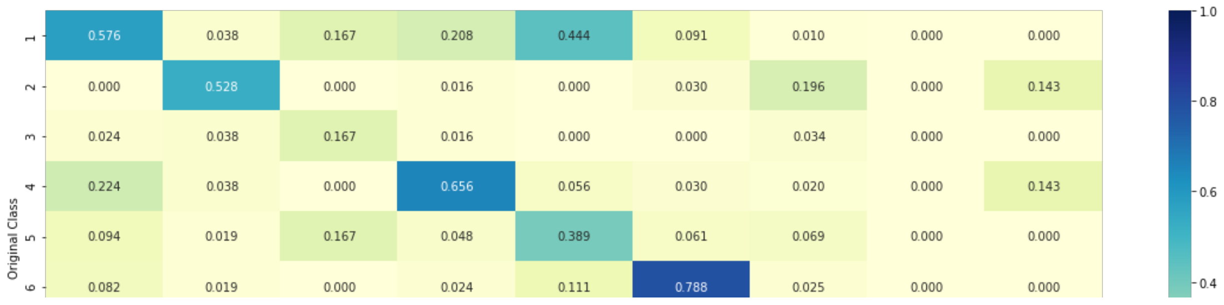
4.2.2. Testing the model with best hyper paramters

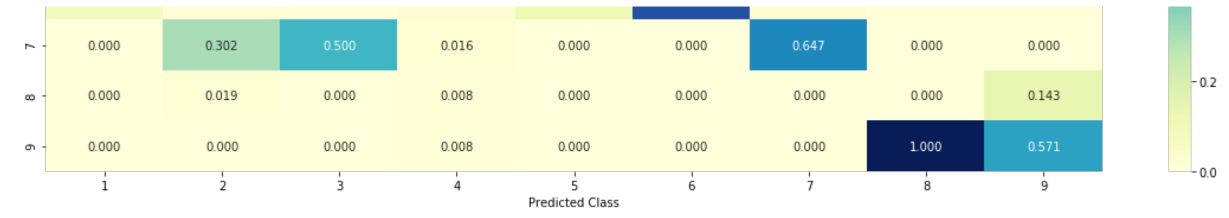
In [112]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='au
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target va
# 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
#-------------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_
```

```
Log loss : 1.0732328472930994
Number of mis-classified points : 0.3815789473684211
-------------------- Confusion matrix --------------------
```

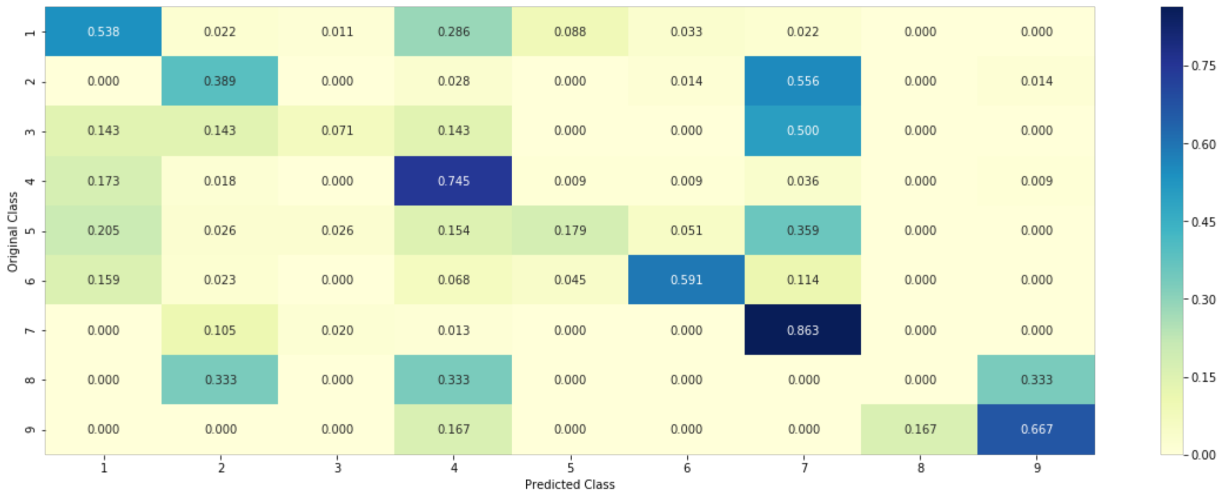


```
-------------------- Precision matrix (Columm Sum=1) ---------------
-----
```





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



	1	2	3	4	5	6	7	8	9
1	0.538	0.022	0.011	0.286	0.088	0.033	0.022	0.000	0.000
2	0.000	0.389	0.000	0.028	0.000	0.014	0.556	0.000	0.014
3	0.143	0.143	0.071	0.143	0.000	0.000	0.500	0.000	0.000
4	0.173	0.018	0.000	0.745	0.009	0.009	0.036	0.000	0.009
5	0.205	0.026	0.026	0.154	0.179	0.051	0.359	0.000	0.000
6	0.159	0.023	0.000	0.068	0.045	0.591	0.114	0.000	0.000
7	0.000	0.105	0.020	0.013	0.000	0.000	0.863	0.000	0.000
8	0.000	0.333	0.000	0.333	0.000	0.000	0.000	0.000	0.333
9	0.000	0.000	0.000	0.167	0.000	0.000	0.000	0.167	0.667

Partial matrix above (rows 7–9):

	1	2	3	4	5	6	7	8	9
7	0.000	0.302	0.500	0.016	0.000	0.000	0.647	0.000	0.000
8	0.000	0.019	0.000	0.008	0.000	0.000	0.000	0.000	0.143
9	0.000	0.000	0.000	0.008	0.000	0.000	0.000	1.000	0.571

4.3. Logistic Regression

4.3.1. With Class balancing

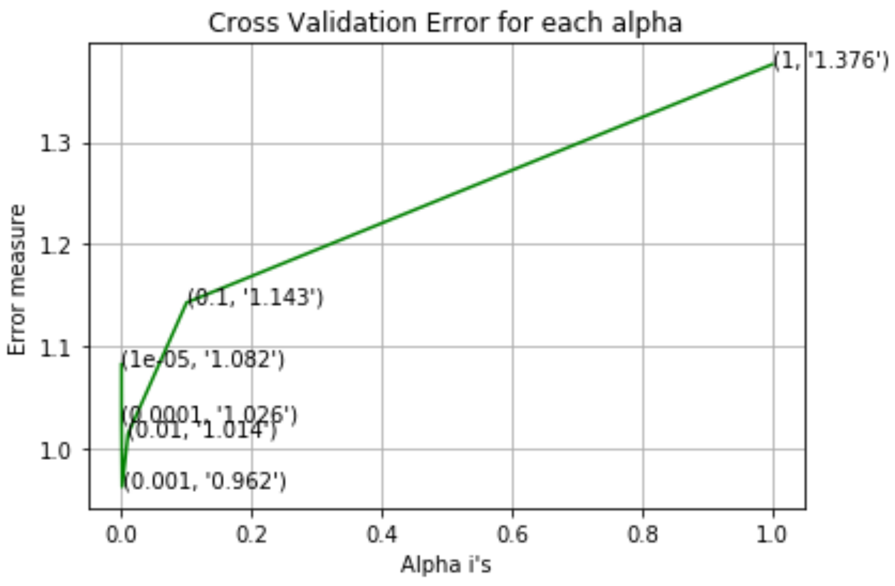
4.3.1.1. Hyper paramter tuning

In [116]:

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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.o
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, meth
#
# 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, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2'
    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
    # to avoid rounding error while multiplying probabilites we use lo
    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], 
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
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross val
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log 
```

```
for alpha = 1e-05
Log Loss : 1.0820257983988437
for alpha = 0.0001
Log Loss : 1.0264629789850206
for alpha = 0.001
Log Loss : 0.9618451656833001
```

```
for alpha = 0.01
Log Loss : 1.0136912392422004
for alpha = 0.1
Log Loss : 1.1425637431283917
for alpha = 1
Log Loss : 1.3763226423522668
```



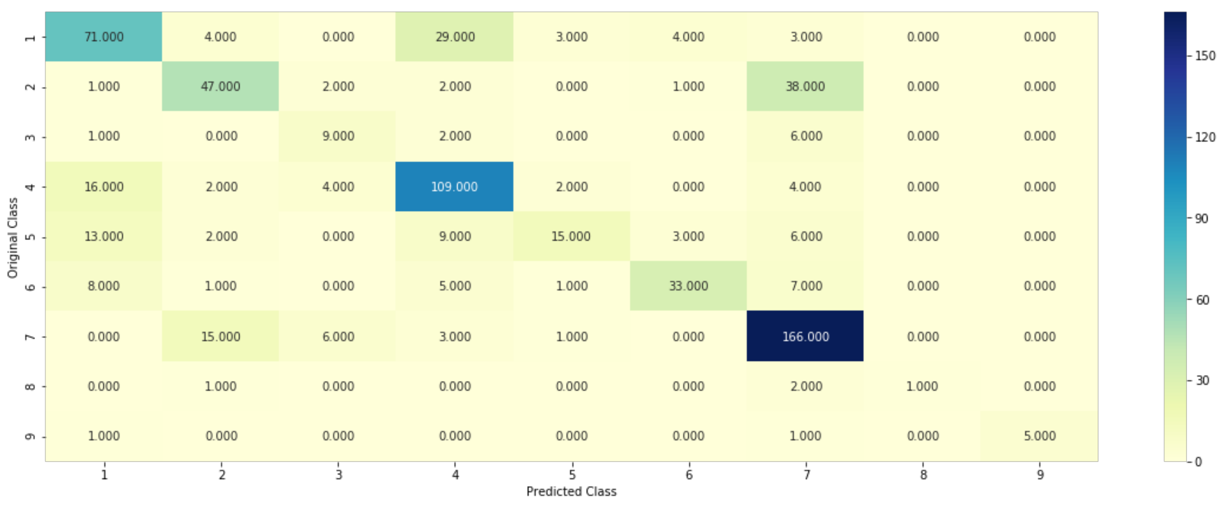
```
For values of best alpha =  0.001 The train log loss is: 0.525758897
0311815
For values of best alpha =  0.001 The cross validation log loss is:
0.9618451656833001
For values of best alpha =  0.001 The test log loss is: 0.9183512118
693308
```

4.3.1.2. Testing the model with best hyper paramters

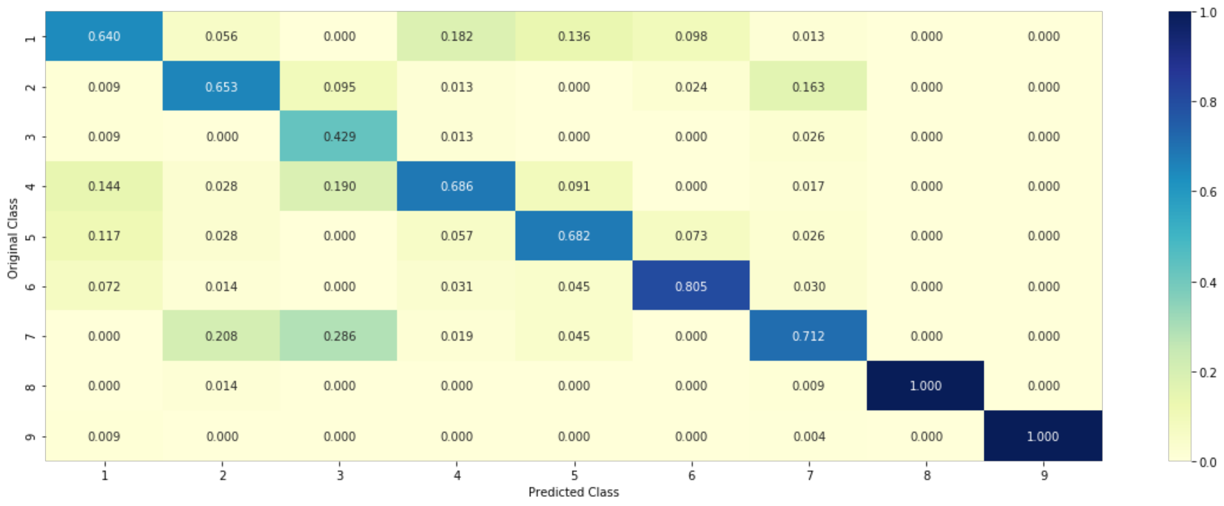
In [117]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mo
# -------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, l
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

```
Log loss : 0.9183512118693308
Number of mis-classified points : 0.3142857142857143
```

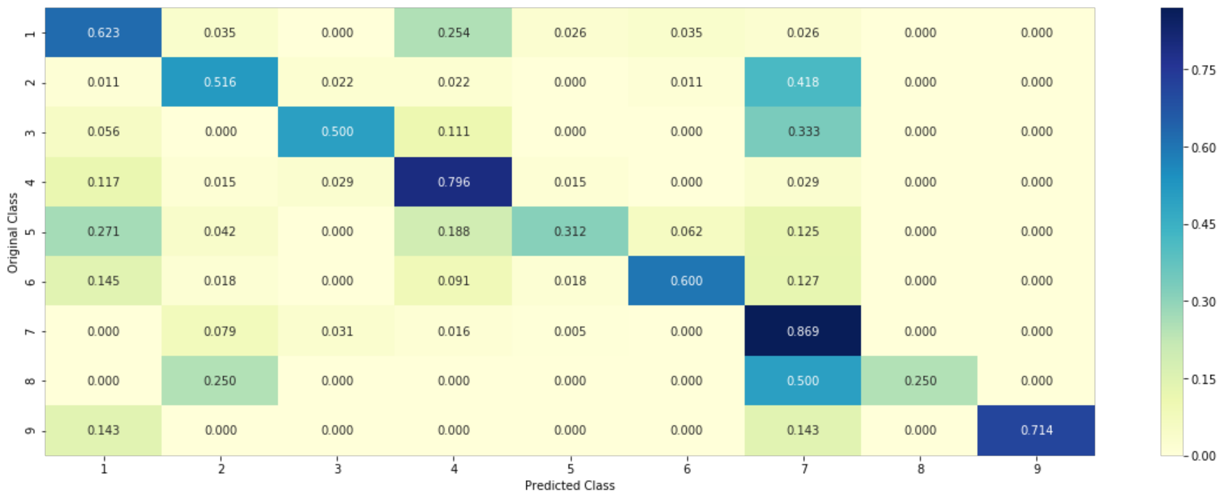
```
-------------------- Confusion matrix --------------------
```



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



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



4.3.1.3. Feature Importance

In [118]:
```
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_feature
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our que
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Pr
```

4.3.1.3.1. Correctly Classified point

In [119]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], r
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 50
predicted_cls = sig_clf.predict(test_x_onehotCoding.iloc[test_point_in
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.4303 0.1552 0.0084 0.1265 0.011
0.0039 0.0196 0.009  0.236 ]]
Actual Class : 1
--------------------------------------------------
```

4.3.1.3.2. correctly Classified point

In [120]:
```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], 
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 42
no_feature = 50
predicted_cls = sig_clf.predict(test_x_onehotCoding.iloc[test_point_in
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.7581 0.1618 0.0083 0.0175 0.0191
0.0159 0.0107 0.0061 0.0025]]
Actual Class : 1
--------------------------------------------------
```

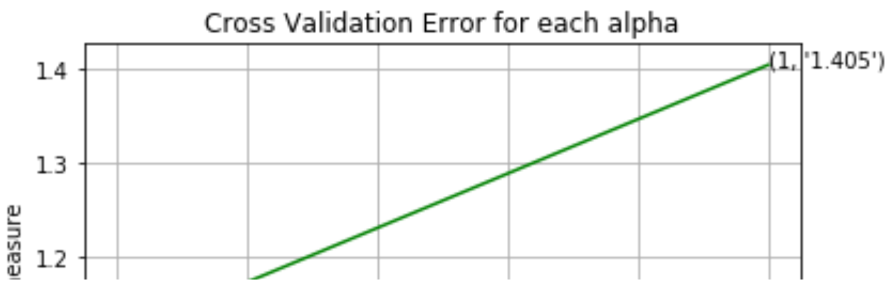
4.3.2. Without Class balancing

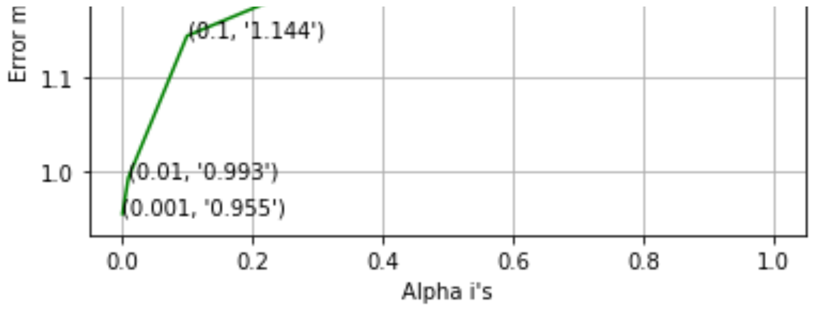
4.3.2.1. Hyper paramter tuning

In [121]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mo
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, l
# 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 Stoc
# predict(X)    Predict class labels for samples in X.

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


# find more about CalibratedClassifierCV here at http://scikit-learn.o
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, meth
#
# 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(-3, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state
    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
    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',
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
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross val
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log
```

```
for alpha = 0.001
Log Loss : 0.9549254257049363
for alpha = 0.01
Log Loss : 0.9933906777381515
for alpha = 0.1
Log Loss : 1.1443484005793763
for alpha = 1
Log Loss : 1.404725094137652
```



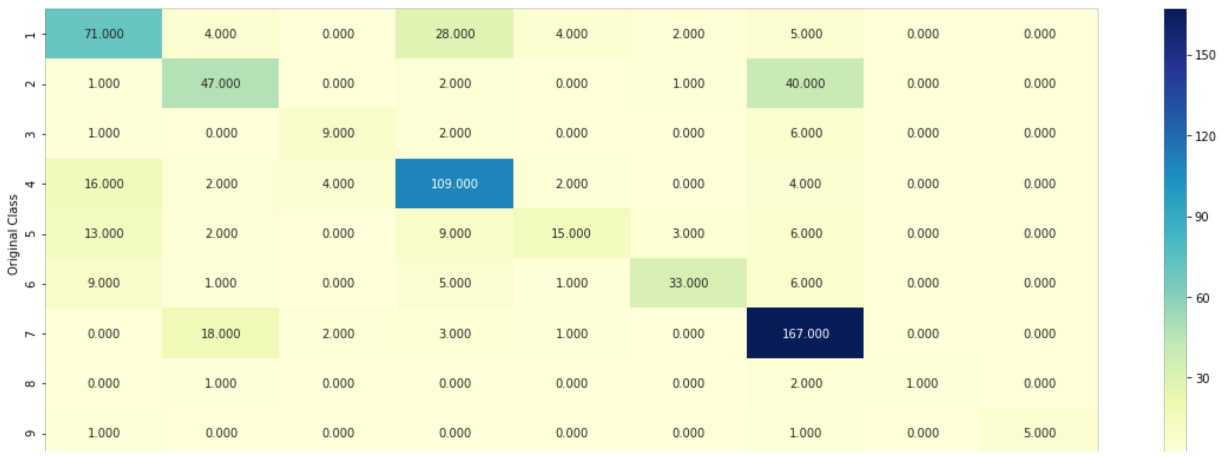


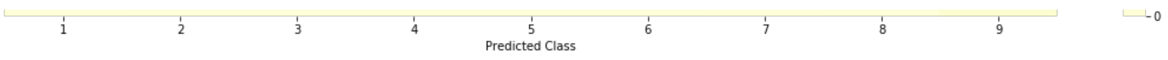
```
For values of best alpha =  0.001 The train log loss is: 0.516816751
0834673
For values of best alpha =  0.001 The cross validation log loss is:
0.9549254257049363
For values of best alpha =  0.001 The test log loss is: 0.9125467886
588063
```

4.3.2.2. Testing model with best hyper parameters

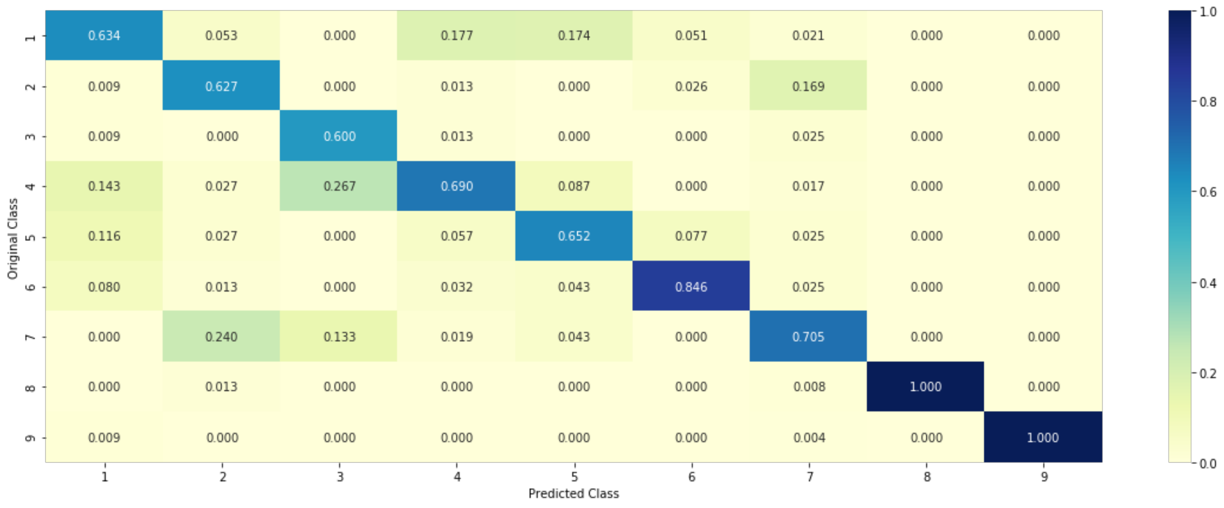
In [122]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mo
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, l
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

```
Log loss : 0.9125467886588063
Number of mis-classified points : 0.31278195488721805
-------------------- Confusion matrix --------------------
```

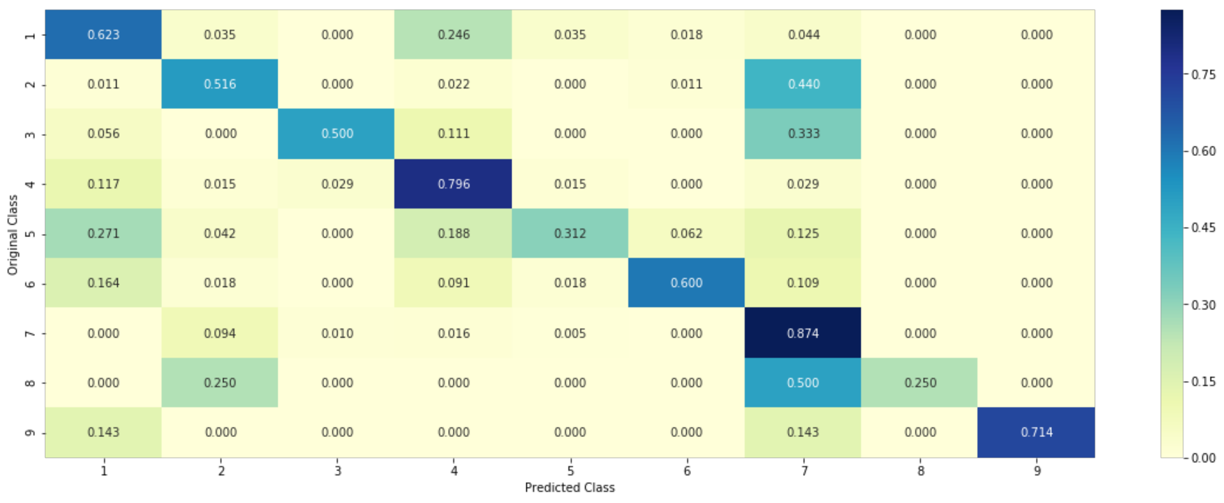




```
-------------------- Precision matrix (Columm Sum=1) ---------------
-----
```



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



4.3.2.3. Feature Importance, Correctly Classified point

In [123]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 42
no_feature = 50
predicted_cls = sig_clf.predict(test_x_onehotCoding.iloc[test_point_in
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.7494 0.175  0.0074 0.0172 0.02
0.0158 0.0089 0.0038 0.0024]]
Actual Class : 1
--------------------------------------------------
```

4.3.2.4. Feature Importance, Inorrectly Classified point

In [124]:
```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 21
no_feature = 50
predicted_cls = sig_clf.predict(test_x_onehotCoding.iloc[test_point_in
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.001  0.1837 0.0062 0.0092 0.0257
0.0216 0.7443 0.0041 0.0043]]
Actual Class : 2
--------------------------------------------------
```

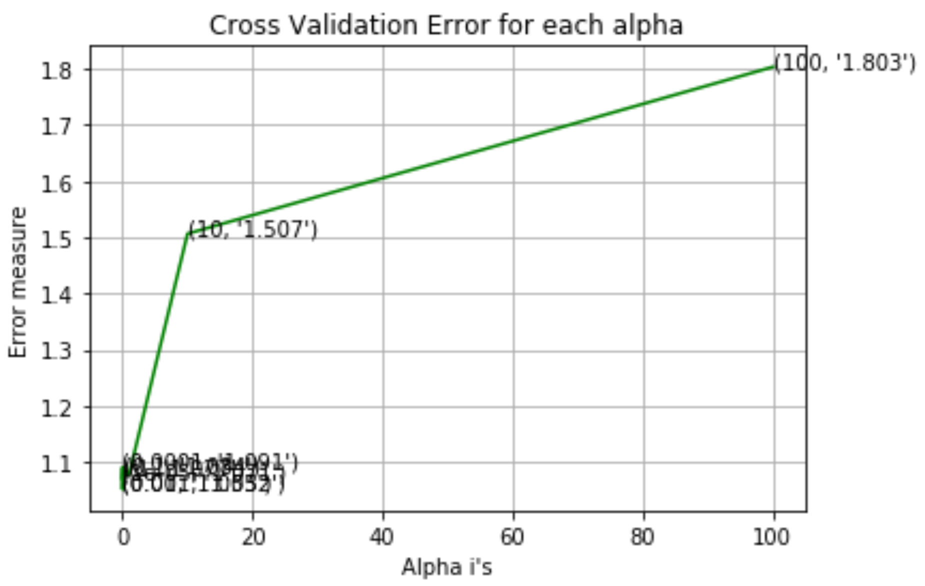
4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

In [125]:
```
# read more about support vector machines with linear kernals here htt

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

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



# find more about CalibratedClassifierCV here at http://scikit-learn.o
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, meth
#
# 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='ba
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2
    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
    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='balanc
clf = SGDClassifier(class_weight='balanced', 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
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross val
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log
```

```
for C = 1e-05
Log Loss : 1.0709184053649463
for C = 0.0001
Log Loss : 1.090737090452236
for C = 0.001
Log Loss : 1.0516495767011695
for C = 0.01
Log Loss : 1.0529856698264046
for C = 0.1
Log Loss : 1.0835262662532756
for C = 1
Log Loss : 1.077480200178823
for C = 10
Log Loss : 1.5065398393210485
for C = 100
Log Loss : 1.8034325767071846
```

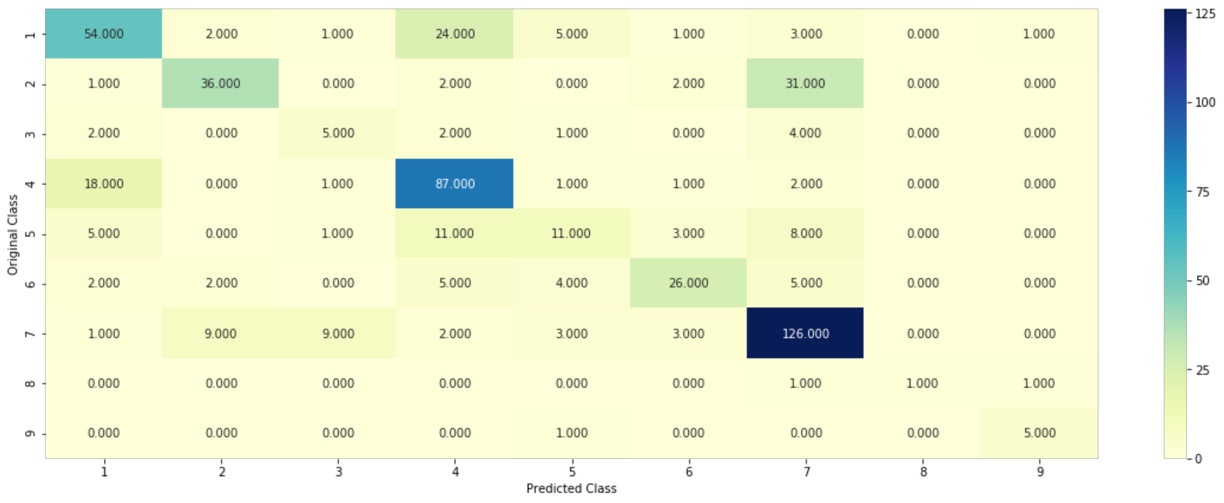


```
For values of best alpha =  0.001 The train log loss is: 0.572901793
1050538
For values of best alpha =  0.001 The cross validation log loss is:
1.0516495767011695
For values of best alpha =  0.001 The test log loss is: 0.9975590275
165219
```

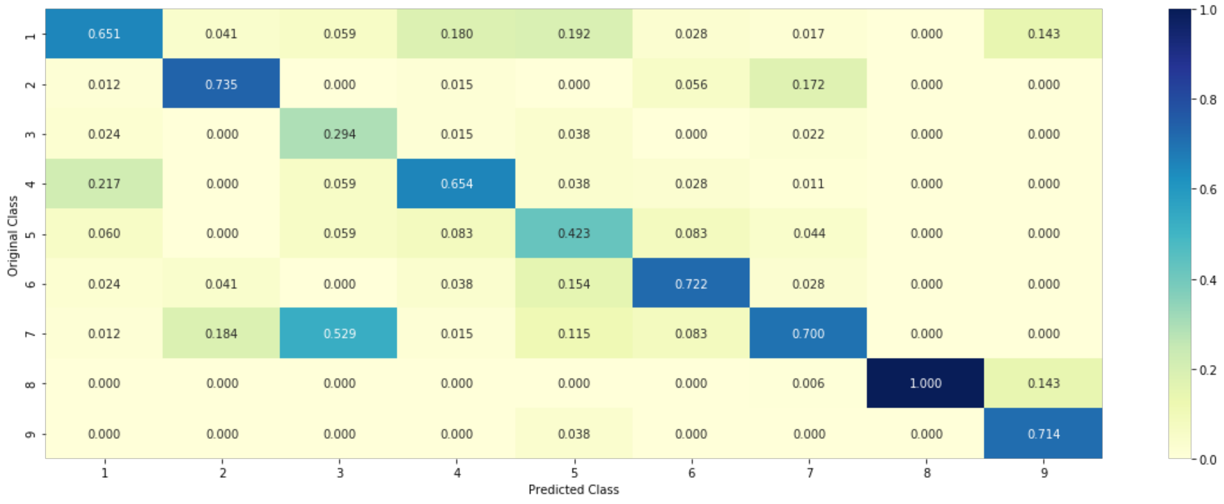
4.4.2. Testing model with best hyper parameters

In [126]:
```
# read more about support vector machines with linear kernals here htt

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

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


# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, clas
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_o
```

```
Log loss : 1.0516495767011695
Number of mis-classified points : 0.34022556390977443
-------------------- Confusion matrix --------------------
```

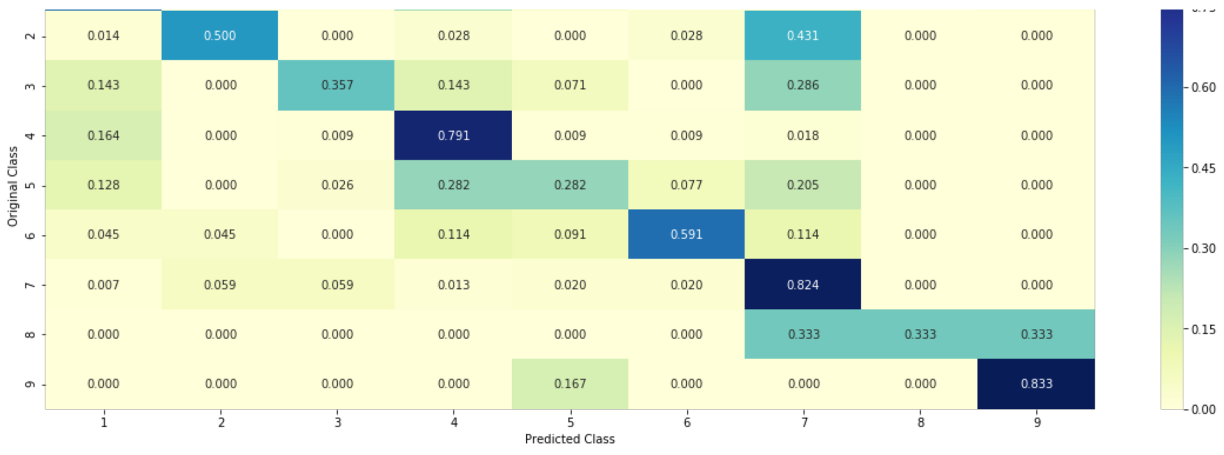


```
-------------------- Precision matrix (Columm Sum=1) ---------------
-----
```



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

0.593 0.022 0.011 0.264 0.055 0.011 0.033 0.000 0.011
1
0.75



Original Class	1	2	3	4	5	6	7	8	9
2	0.014	0.500	0.000	0.028	0.000	0.028	0.431	0.000	0.000
3	0.143	0.000	0.357	0.143	0.071	0.000	0.286	0.000	0.000
4	0.164	0.000	0.009	0.791	0.009	0.009	0.018	0.000	0.000
5	0.128	0.000	0.026	0.282	0.282	0.077	0.205	0.000	0.000
6	0.045	0.045	0.000	0.114	0.091	0.591	0.114	0.000	0.000
7	0.007	0.059	0.059	0.013	0.020	0.020	0.824	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.333	0.333	0.333
9	0.000	0.000	0.000	0.000	0.167	0.000	0.000	0.000	0.833

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [128]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, 
# class_weight=None)

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


# find more about CalibratedClassifierCV here at http://scikit-learn.o
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, meth
#
# 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',
        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=
        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]).ra
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)), (featu
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)], cr
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)], "Th
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "Th
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "Th
```

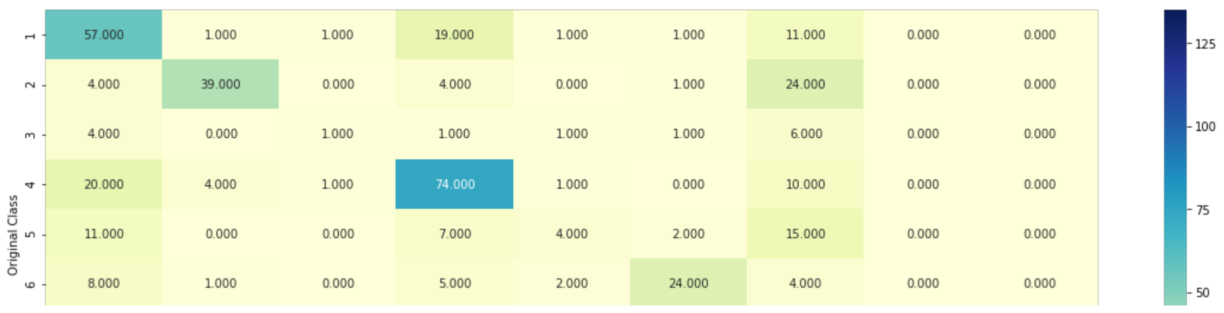
```
for n_estimators = 100 and max depth =  5
Log Loss : 1.221251856458788
for n_estimators = 100 and max depth =  10
Log Loss : 1.1523133259436458
for n_estimators = 200 and max depth =  5
Log Loss : 1.215983059461481
for n_estimators = 200 and max depth =  10
Log Loss : 1.1445768085235268
for n_estimators = 500 and max depth =  5
Log Loss : 1.2074140678407117
for n_estimators = 500 and max depth =  10
Log Loss : 1.1396000410342781
for n_estimators = 1000 and max depth =  5
```

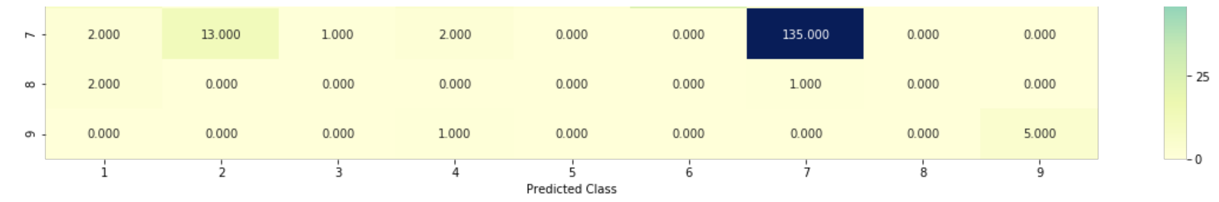
```
Log Loss : 1.1995699341288346
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1379069861482
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1956724884838927
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1327758191013155
For values of best estimator =  2000 The train log loss is: 0.629269
5980731973
For values of best estimator =  2000 The cross validation log loss i
s: 1.1327758191013153
For values of best estimator =  2000 The test log loss is: 1.0772846
773426699
```

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

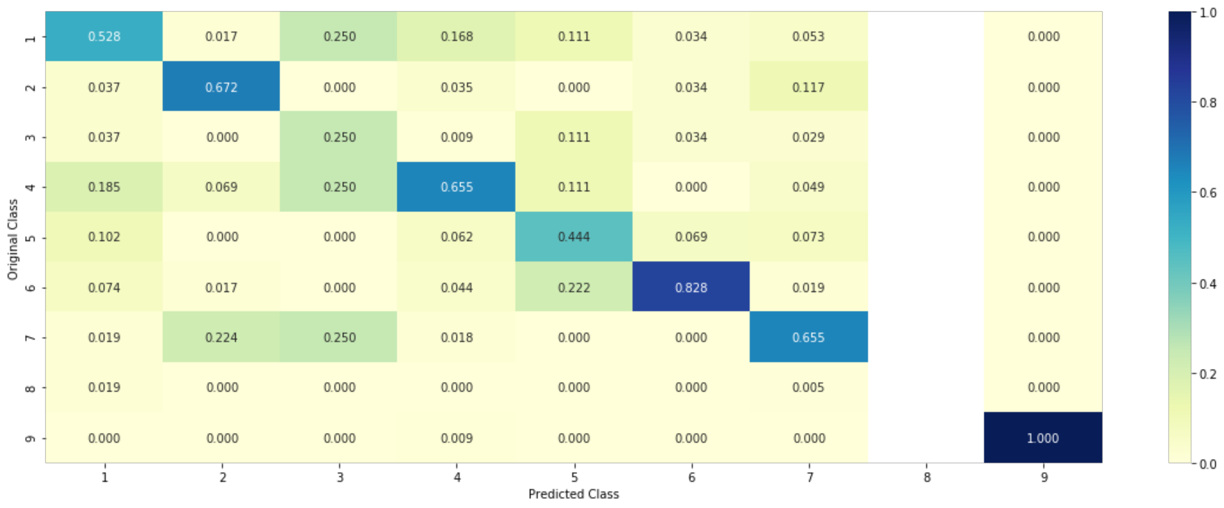
In [129]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, 
# class_weight=None)

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

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cr
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_o
```

```
Log loss : 1.1327758191013153
Number of mis-classified points : 0.36278195488721804
-------------------- Confusion matrix --------------------
```

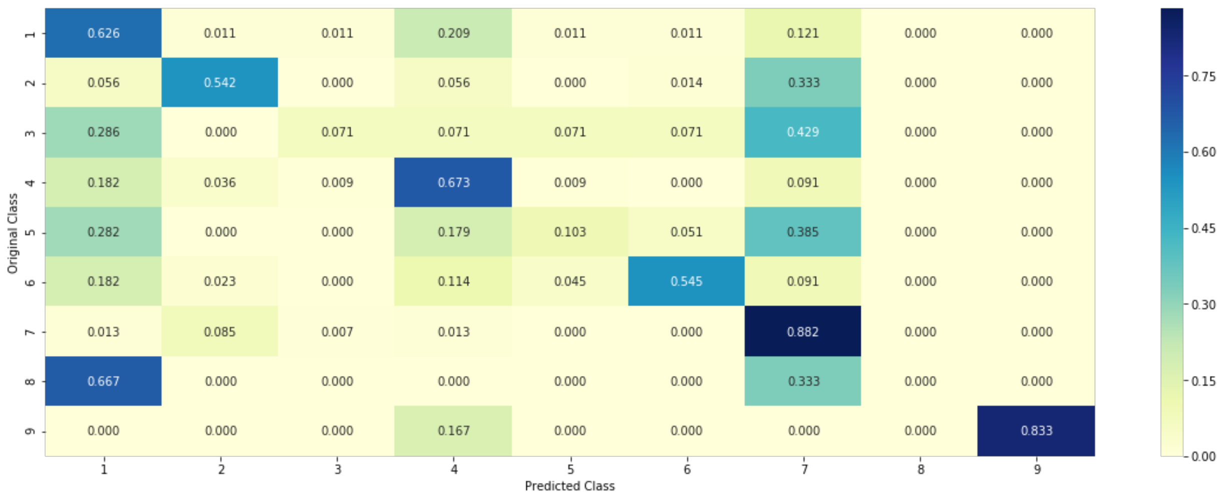




```
-------------------- Precision matrix (Columm Sum=1) --------------------
-----
```



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



4.5.3. Hyper paramter tuning (With Response Coding)

In [130]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, 
# class_weight=None)

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


# find more about CalibratedClassifierCV here at http://scikit-learn.o
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, meth
#
# 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',
        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
        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]).r
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)), (featu
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)], cr
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 tra
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cro
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The tes
```

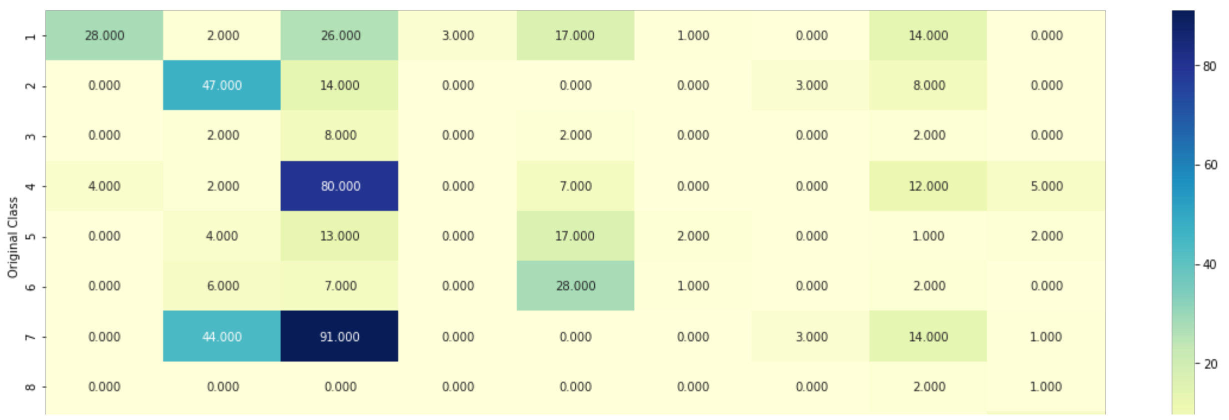
```
for n_estimators = 10 and max depth =  2
Log Loss : 2.985604279662273
for n_estimators = 10 and max depth =  3
Log Loss : 2.5805638925724144
for n_estimators = 10 and max depth =  5
Log Loss : 2.50874108489422
for n_estimators = 10 and max depth =  10
Log Loss : 2.500396140877808
for n_estimators = 50 and max depth =  2
Log Loss : 2.288155547931257
for n_estimators = 50 and max depth =  3
Log Loss : 2.325426838395101
for n_estimators = 50 and max depth =  5
Log Loss : 2.1372277705559157
for n_estimators = 50 and max depth =  10
Log Loss : 2.3333314077055536
for n_estimators = 100 and max depth =  2
Log Loss : 2.372609968962405
for n_estimators = 100 and max depth =  3
Log Loss : 2.557601374610918
for n_estimators = 100 and max depth =  5
Log Loss : 2.096004993001543
for n_estimators = 100 and max depth =  10
Log Loss : 2.346585959841944
for n_estimators = 200 and max depth =  2
Log Loss : 2.403972042361107
for n_estimators = 200 and max depth =  3
Log Loss : 2.5859570133876453
for n_estimators = 200 and max depth =  5
Log Loss : 2.251926017679841
for n_estimators = 200 and max depth =  10
Log Loss : 2.4353192227554064
for n_estimators = 500 and max depth =  2
Log Loss : 2.4422391006300255
for n_estimators = 500 and max depth =  3
Log Loss : 2.488401064217581
for n_estimators = 500 and max depth =  5
Log Loss : 2.2711479592792223
for n_estimators = 500 and max depth =  10
Log Loss : 2.4646423499305707
for n_estimators = 1000 and max depth =  2
Log Loss : 2.4038788188197193
for n_estimators = 1000 and max depth =  3
Log Loss : 2.34861746224742
```

```
for n_estimators = 1000 and max depth =  5
Log Loss : 2.216408979041495
for n_estimators = 1000 and max depth =  10
Log Loss : 2.4344455062889496
For values of best alpha =  100 The train log loss is: 0.02427030396
5366887
For values of best alpha =  100 The cross validation log loss is: 2.
096004993001543
For values of best alpha =  100 The test log loss is: 2.067159216343
359
```

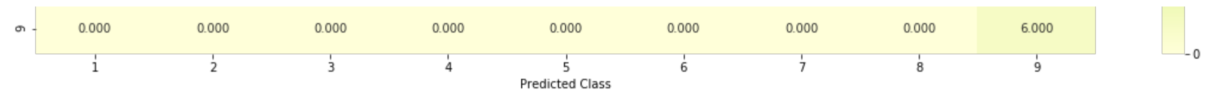
4.5.4. Testing model with best hyper parameters (Response Coding)

In [131]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, 
# class_weight=None)

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

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x
```

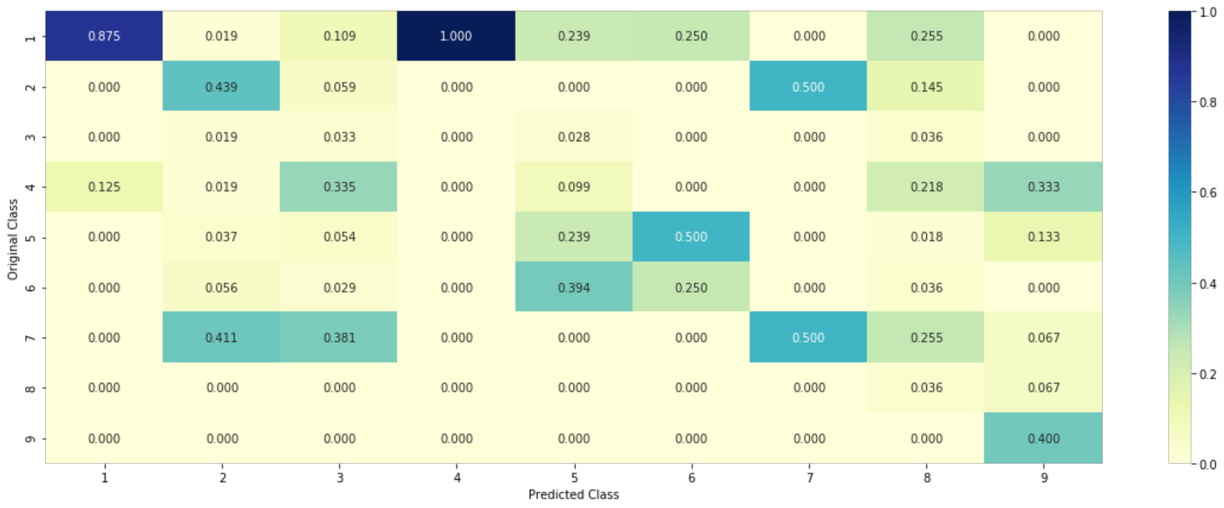
```
Log loss : 2.096004993001543
Number of mis-classified points : 0.7894736842105263
-------------------- Confusion matrix --------------------
```



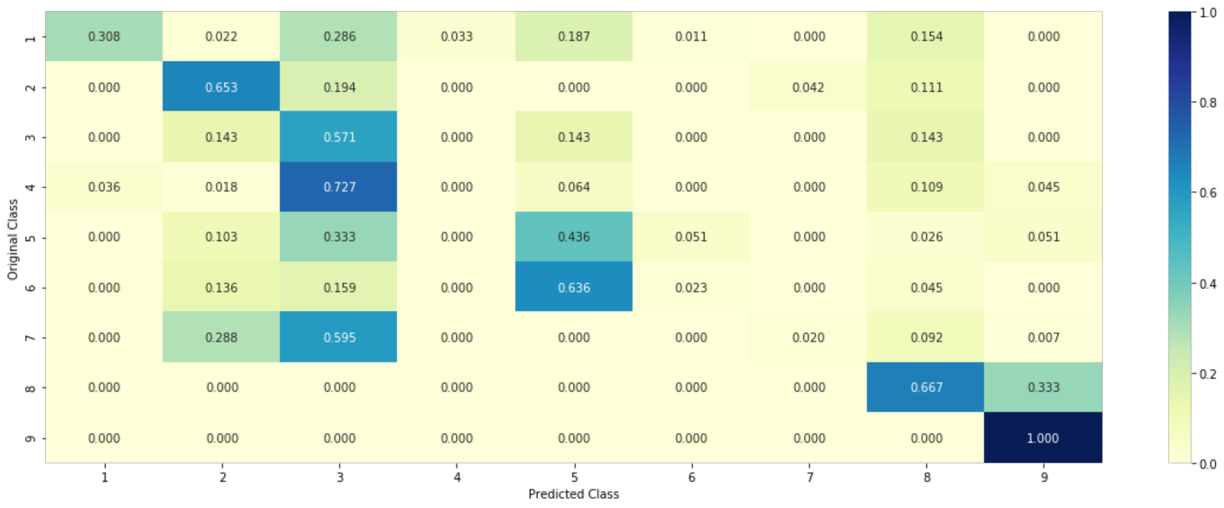
1	28.000	2.000	26.000	3.000	17.000	1.000	0.000	14.000	0.000
2	0.000	47.000	14.000	0.000	0.000	0.000	3.000	8.000	0.000
3	0.000	2.000	8.000	0.000	2.000	0.000	0.000	2.000	0.000
4	4.000	2.000	80.000	0.000	7.000	0.000	0.000	12.000	5.000
5	0.000	4.000	13.000	0.000	17.000	2.000	0.000	1.000	2.000
6	0.000	6.000	7.000	0.000	28.000	1.000	0.000	2.000	0.000
7	0.000	44.000	91.000	0.000	0.000	0.000	3.000	14.000	1.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.000	2.000	1.000



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



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



4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [133]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/mo
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, l
# 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 Stoc
# predict(X)    Predict class labels for samples in X.

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


# read more about support vector machines with linear kernals here htt
# --------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinkin
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decis

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


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

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


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

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


clf3 = MultinomialNB(alpha=0.001)
clf3.fit(train_x_onehotCoding, train_y)
```

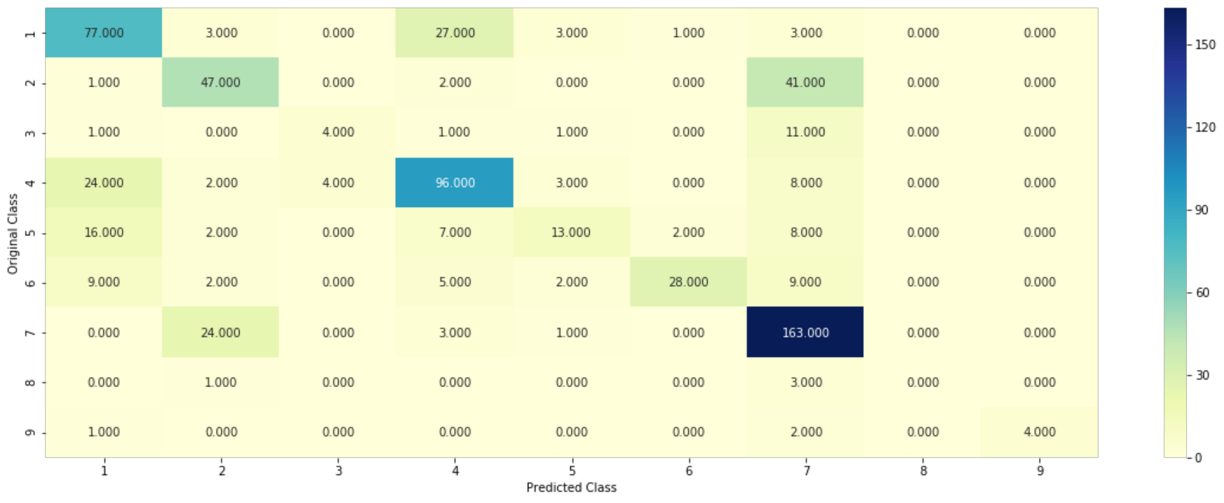
```
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_c
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, si
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predi
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_clf
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

```
Logistic Regression :  Log Loss: 0.95
Support vector machines : Log Loss: 1.09
Naive Bayes : Log Loss: 1.22
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.17
5
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.01
0
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.43
1
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.04
8
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.08
1
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.3
08
```

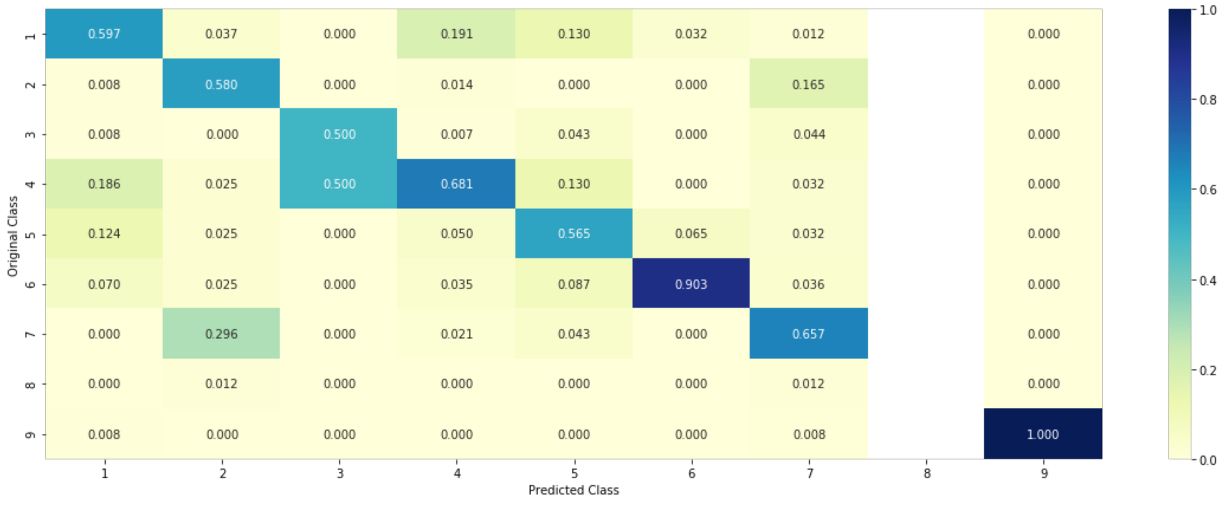
4.7.2 testing the model with the best hyper parameters

In [134]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], 
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.predi
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onel
```

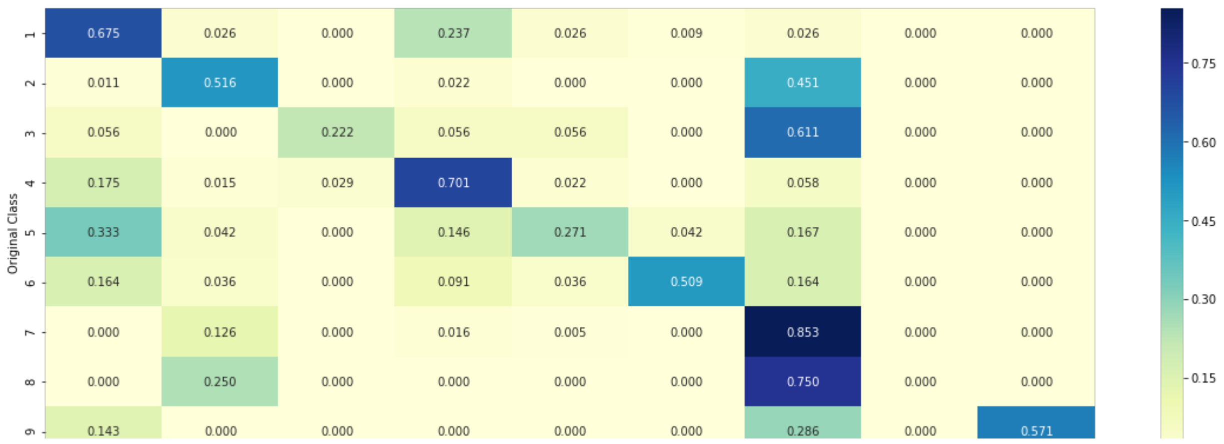
```
Log loss (train) on the stacking classifier : 0.6013716058609647
Log loss (CV) on the stacking classifier : 1.0475754062476912
Log loss (test) on the stacking classifier : 1.0445581939340607
Number of missclassified point : 0.35037593984962406
-------------------- Confusion matrix --------------------
```

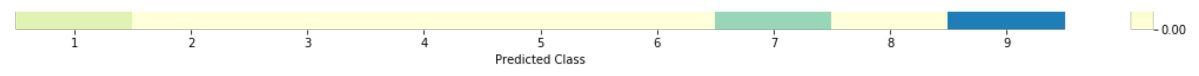


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```



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

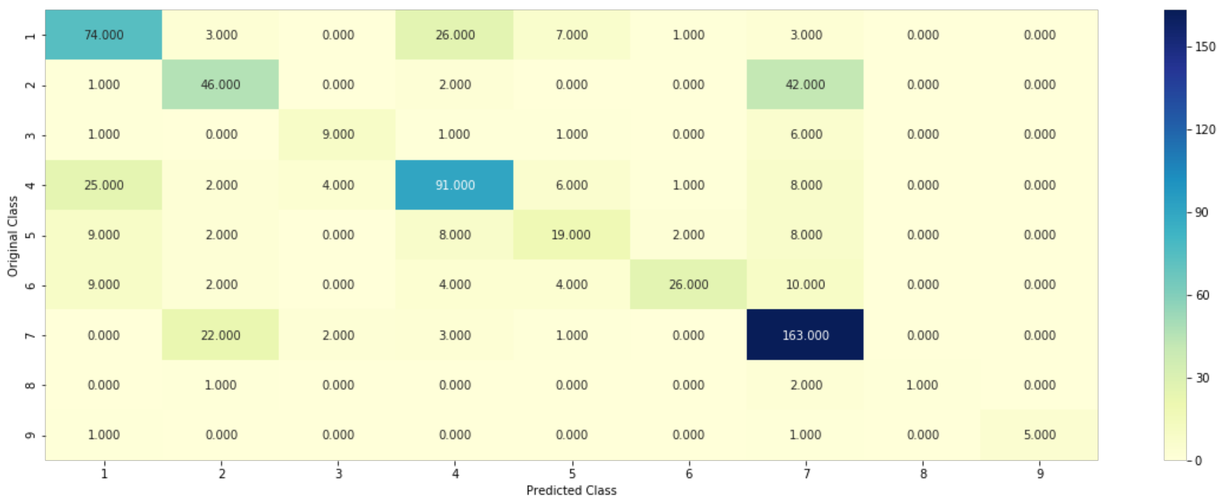




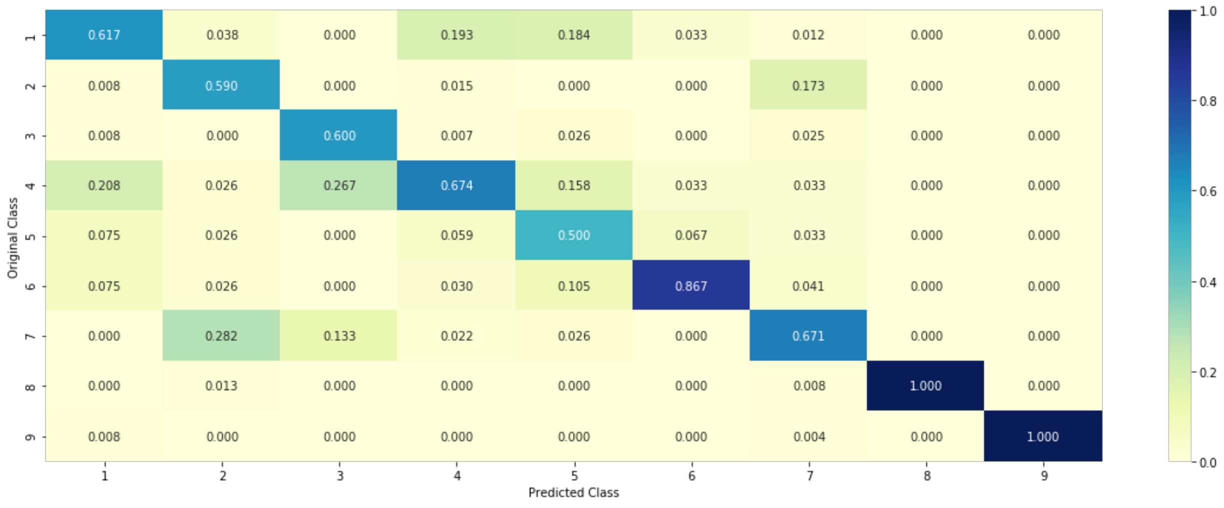
4.7.3 Maximum Voting classifier

In [135]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemb
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, 
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.p
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vc
print("Number of missclassified point :", np.count_nonzero((vclf.predi
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_one
```

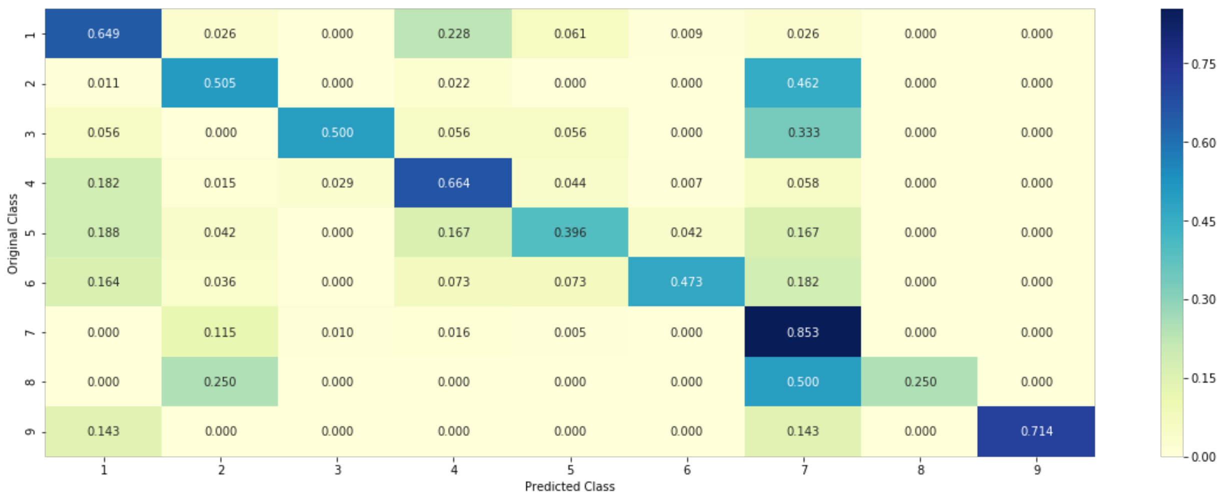
```
Log loss (train) on the VotingClassifier : 0.6997619720475359
Log loss (CV) on the VotingClassifier : 1.0295222706087999
Log loss (test) on the VotingClassifier : 1.0155378435687559
Number of missclassified point : 0.3473684210526316
-------------------- Confusion matrix --------------------
```



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



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



In [136]:
```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Model","Train_logloss","Cv_logloss","Test_logloss",

x.add_row(["Naive Bayes", 0.79453, 1.21285, 1.23219, 35.52])
x.add_row(["KNN", 0.75710, 1.07323, 1.09839, 38.15])
x.add_row(["LR (Balanced data)", 0.52575, 0.96184,  0.91835, 31.42])
x.add_row(["LR (Without Balanced data)",0.51681, 0.95492, 0.91254, 31.:
x.add_row(["Linear SVM", 0.57290, 1.05164, 0.99755, 34.02])
x.add_row(["Random Forest (one hot encoding)", 0.62926, 1.13277, 1.077:
x.add_row(["Random Forest (Response Coding)", 0.02427, 2.09600, 2.0671!
x.add_row(["Stacking model (LR, SVM, NB)",0.60137, 1.04757, 1.04455, 3!
x.add_row(["Maximum Voting Classifier", 0.69976,  1.02952, 1.01553, 34


print(x)
```

```
+----------------------------------+---------------+------------+---
-----------+----------------------+
|              Model               | Train_logloss | Cv_logloss | Te
st_logloss | Misclassified_error |
+----------------------------------+---------------+------------+---
-----------+----------------------+
|           Naive Bayes            |    0.79453    |  1.21285   |
1.23219    |        35.52         |
|               KNN                |     0.7571    |  1.07323   |
1.09839    |        38.15         |
|        LR (Balanced data)        |    0.52575    |  0.96184   |
0.91835    |        31.42         |
|    LR (Without Balanced data)    |    0.51681    |  0.95492   |
0.91254    |        31.27         |
|            Linear SVM            |     0.5729    |  1.05164   |
0.99755    |        34.02         |
| Random Forest (one hot encoding) |    0.62926    |  1.13277   |
1.07728    |        36.27         |
| Random Forest (Response Coding)  |    0.02427    |   2.096    |
2.06715    |        78.94         |
|   Stacking model (LR, SVM, NB)   |    0.60137    |  1.04757   |
1.04455    |        35.03         |
|    Maximum Voting Classifier     |    0.69976    |  1.02952   |
1.01553    |        34.73         |
+----------------------------------+---------------+------------+---
-----------+----------------------+
```

Steps followed

1) For reducing the loss in test data, Featurization for the text data used was TF-IDF Featurization. The least loss score for train, test, CV was using Logistic Regression slgorithm and class being balanced.

2) Instead of using all the features, the top 1000 features in TF-IDF features were selected. The least loss score for train, test, CV was using Logistic Regression slgorithm and class being balanced. KNN algorithm also performed well but the model is not interpretable and interpretability is required in this given problem.

3) As Logistic regression (both balanced and unbalanced class) was performing better than other algorithms. Uni-gram and Bi-gram in Bag of Words featurization was used. The least loss score for train, test, CV was using Logistic Regression slgorithm and class being balanced. Linear SVM and KNN algorithm also performed well when compared to other featurization. This also could not reduce the log loss less than 1.

4) To get log loss less than 1, A few feturization techniques were implemented.

1) Number of words in a given column.

2) Number of characters in a given column.

3) Combining of Gene feature and variation in a single column.

4) Number of words greater than 5000 in a given column. If it is greate than the value assigned is 1 else the value assigned is 0.

5) Number of characters greater than 50000 in a given column. If it is greater than the value is 1 else the values assigned uis 0.

The Log loss was less than 1 using Logistic regression with both balanced and imbalanced class.

5. Observation

To get log loss of less than 1.0, a few feature engineering was implemented :

1) Number of words in a given column

2) Number of characters in a given column

3) Combing gene and variation in a single column

4) Number of words greater than 5000 in a column, if true then 1 else 0

5) Number of chracters greater than 50000 in a column, if true then 1 else 0

The log loss is less than 1 for Logistic Regression with both balanced data and imbalanced data