TFIDF_1.0

August 2, 2018

Personalized cancer diagnosis

1. Business Problem

1.1. Description
Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/
Data: Memorial Sloan Kettering Cancer Center (MSKCC)
Download training_variants.zip and training_text.zip from Kaggle.
Context:
Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/discussion/35336#198462
Problem statement :
Classify the given genetic variations/mutations based on evidence from text-based clinical literature.
1.2. Source/Useful Links
Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

```
<li> Apply All the models with tf-idf features (Replace CountVectorizer with tfidfVectorizer an
<li> Instead of using all the words in the dataset, use only the top 1000 words based of tf-id
<li>Apply Logistic regression with CountVectorizer Features, including both unigrams and bigra
<li> Try any of the feature engineering techniques discussed in the course to reduce the CV an
```

1.4. Assignment

```
<li> Apply All the models with tf-idf features (Replace CountVectorizer with tfidfVectorizer an
<li> Instead of using all the words in the dataset, use only the top 1000 words based of tf-id
<li>Apply Logistic regression with CountVectorizer Features, including both unigrams and bigra
<li> Try any of the feature engineering techniques discussed in the course to reduce the CV an
```

2. Machine Learning Problem Formulation

2.1. Data
2.1.1. Data Overview

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

```
<li>
training_variants (ID , Gene, Variations, Class)
</li>
<li>
training_text (ID, Text)
</li>
```

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 cla
```

2.2.2. Performance Metric

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

Metric(s): * Multi class log-loss * Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

```
In [91]: 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
```

```python
from sklearn.svm import SVC
from sklearn.cross_validation import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier
import nltk
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [92]:
```python
data = pd.read_csv('training_variants')
print('Number of data points : ', data.shape[0])
print('Number of features : ', data.shape[1])
print('Features : ', data.columns.values)
data.head()
```

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

Out[92]:

	ID	Gene	Variation	Class
0	0	FAM58A	Truncating Mutations	1
1	1	CBL	W802*	2
2	2	CBL	Q249E	2
3	3	CBL	N454D	3
4	4	CBL	L399V	4

```
training_variants is a comma separated file containing the description of the genetic mutations
Fields are
<ul>
    <li><b>ID : </b>the id of the row used to link the mutation to the clinical evidence</li>
    <li><b>Gene : </b>the gene where this genetic mutation is located </li>
    <li><b>Variation : </b>the aminoacid change for this mutations </li>
    <li><b>Class :</b> 1-9 the class this genetic mutation has been classified on</li>
</ul>
```

3.1.2. Reading Text Data

In [93]:
```
# note the seprator in this file
data_text =pd.read_csv("training_text",sep="\|\|",engine="python",names=["ID","TEXT"]
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[93]:
```
   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 [94]:
```
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))
#sno = nltk.stem.SnowballStemmer('english') #initialising the snowball stemmer

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]', ' ', str(total_text))
        # replace multiple spaces with single space
        total_text = re.sub('\s+',' ', total_text)
        # converting all the chars into lower-case.
        total_text = total_text.lower()
        for word in total_text.split():
        # if the word is a not a stop word then retain that word from the data
            #word=(sno.stem(word.lower())).encode('utf8')
            #    print(word)
            if not word in stop_words:
                string += word + " "

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

In [100]:
```
#text processing stage. this step takes a lot of ti
from os import path
start_time = time.clock()
if os.path.isfile("result.pickle"):
  print("file already present")
  result=pd.read_pickle("result.pickle")
```

```
        else:
          for index, row in data_text.iterrows():
            nlp_preprocessing(row['TEXT'], index, 'TEXT')
          print('Time took for preprocessing the text :',time.clock() - start_time, "seconds"
          #merging both gene_variations and text data based on ID
          result = pd.merge(data, data_text,on='ID', how='left')
          result.head()
          result.to_pickle("result.pickle")
```

```
file already present
```

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 [101]: y_true = result['Class'].values
          result.Gene      = result.Gene.str.replace('\s+', '_')
          result.Variation = result.Variation.str.replace('\s+', '_')

          # split the data into test and train by maintaining same distribution of output vara
          X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true
          # split the train data into train and cross validation by maintaining same distribut
          train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train
```

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

```
In [102]: 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])
          train_df['TEXT'].shape
```

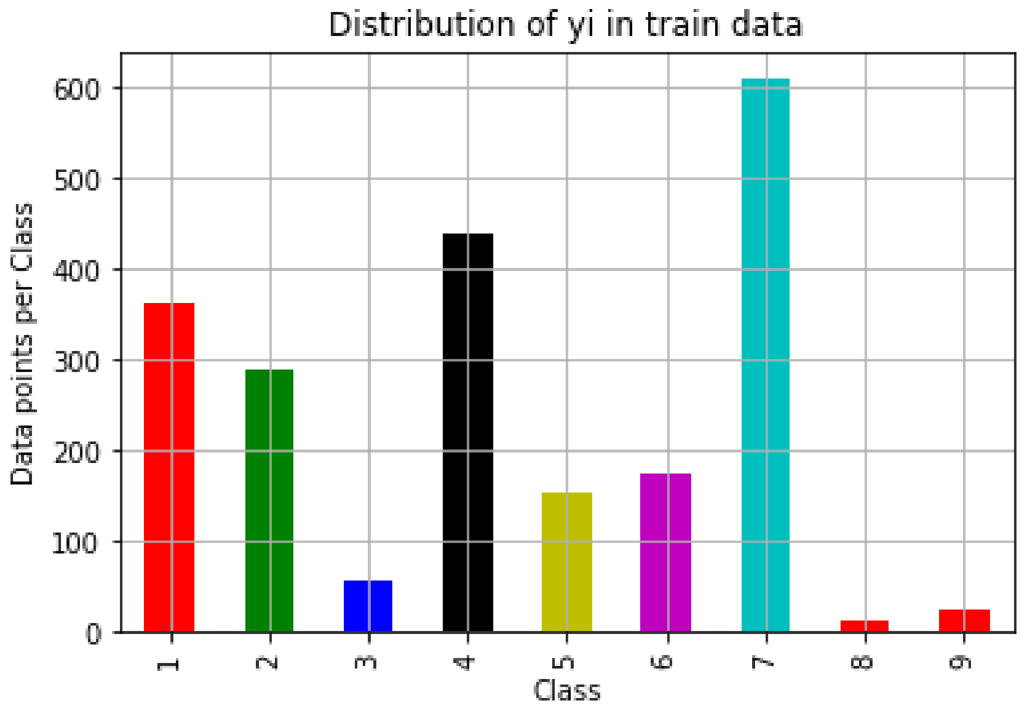
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

```
Out[102]: (2124,)
```

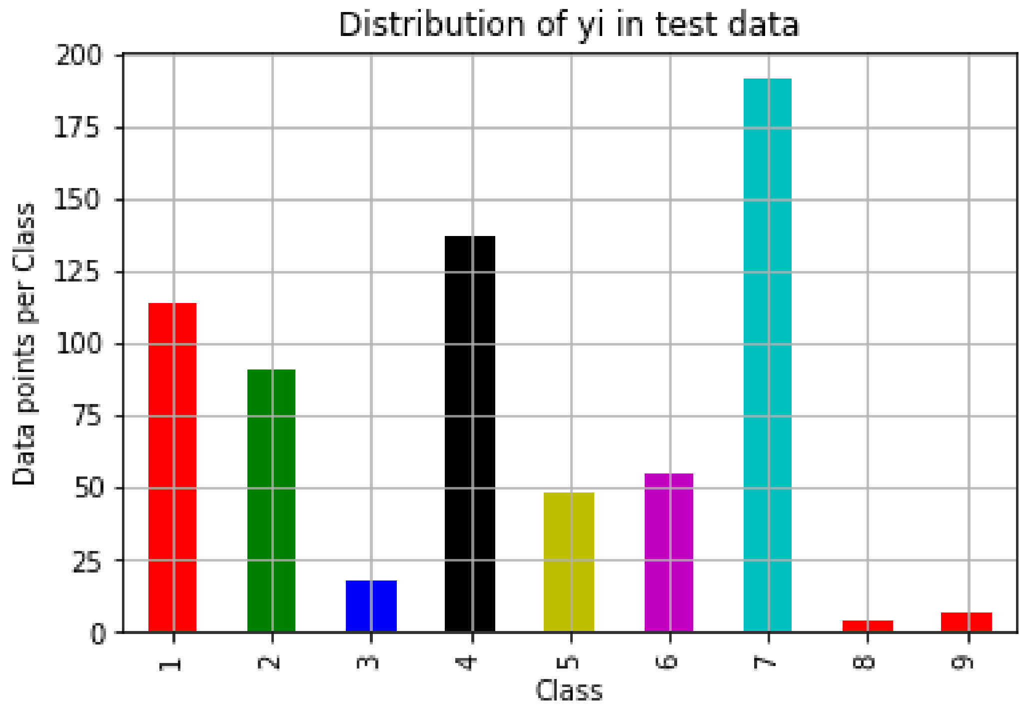
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

```
In [103]: # it returns a dict, keys as class labels and values as the number of data points in
          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 = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
          train_class_distribution.plot(kind='bar', color=my_colors)
          plt.xlabel('Class')
```

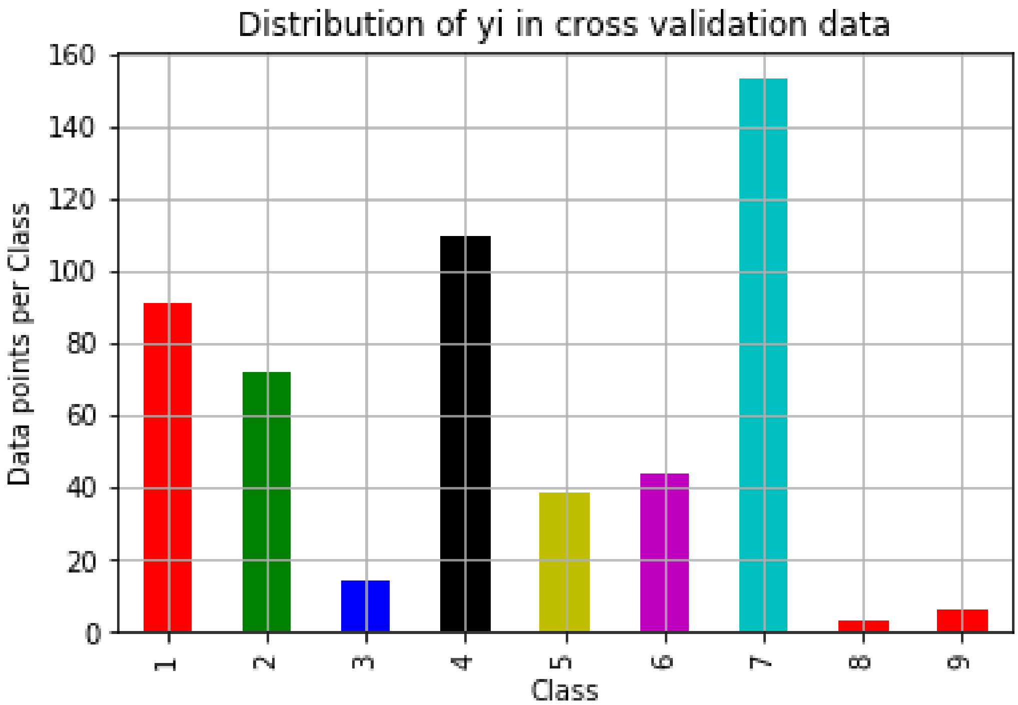
```
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()

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


print('-'*80)
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
test_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

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

print('-'*80)
my_colors = ['r', 'g', 'b', 'k', 'y', 'm', 'c']
cv_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

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




```
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.

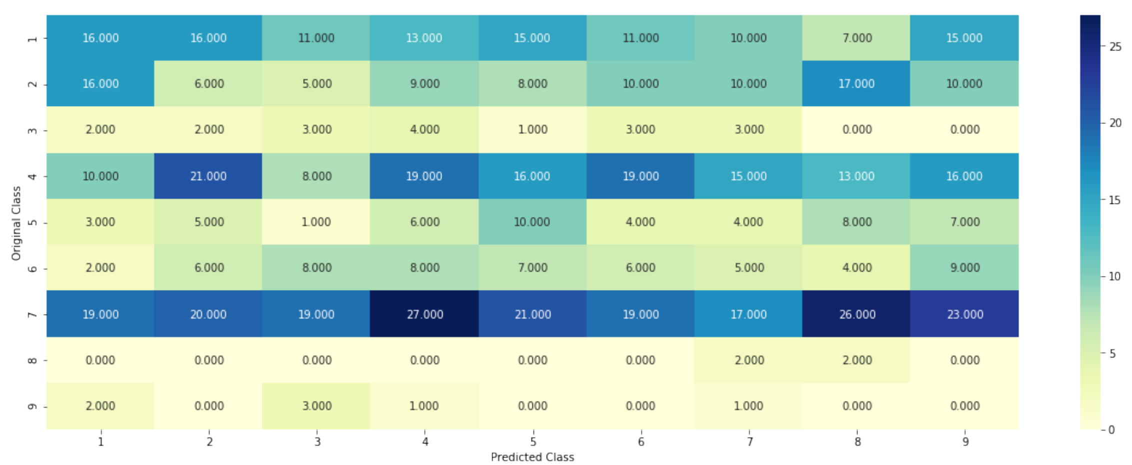
```
In [104]: # This function plots the confusion matrices given y_i, y_i_hat.
          def plot_confusion_matrix(test_y, predict_y):
              C = confusion_matrix(test_y, predict_y)
              # C = 9,9 matrix, each cell (i,j) represents number of points of class i are pre

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

              # C = [[1, 2],
```

```
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows i
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                            [2/3, 4/7]]

    # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
    #                            [3/7, 4/7]]
    # sum of row elements = 1

    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows i
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]

    labels = [1,2,3,4,5,6,7,8,9]
    # representing A in heatmap format
    print("-"*20, "Confusion matrix", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklal
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    print("-"*20, "Precision matrix (Columm Sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklal
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    # representing B in heatmap format
    print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(A, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklal
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

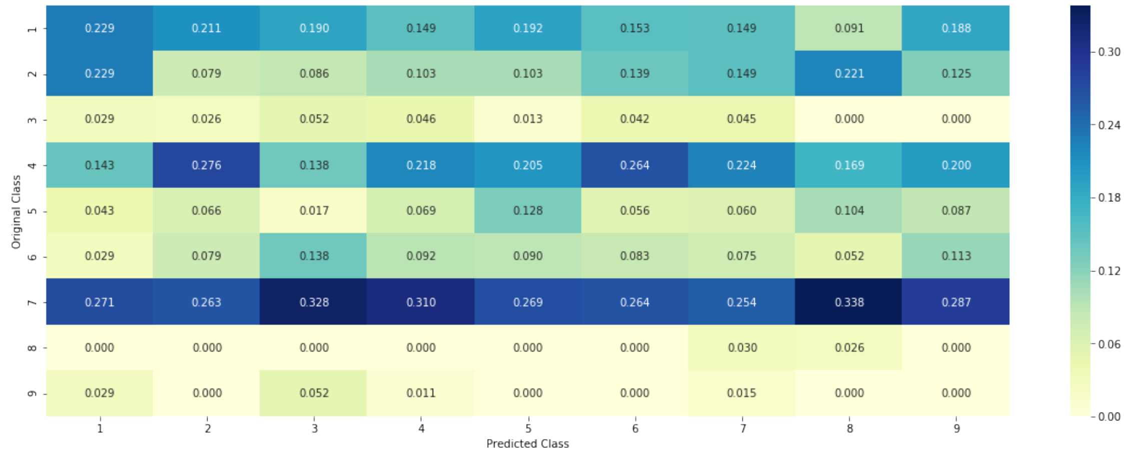
```
In [105]: # we need to generate 9 numbers and the sum of numbers should be 1
          # one solution is to genarate 9 numbers and divide each of the numbers by their sum
          # ref: https://stackoverflow.com/a/18662466/4084039
```

```python
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,9))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,9)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Cross Validation Data using Random Model",log_loss(y_cv,cv_predic


# 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, ep

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

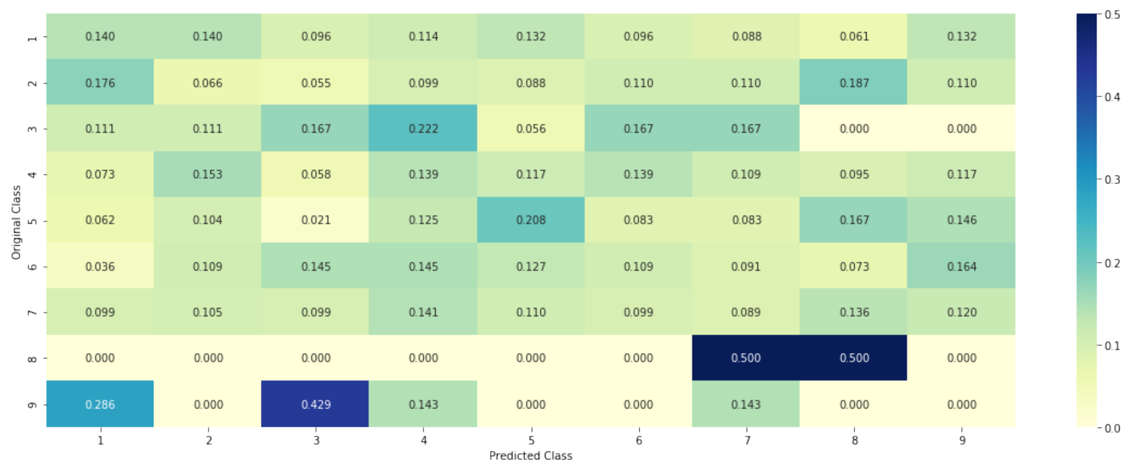
```
Log loss on Cross Validation Data using Random Model 2.49003195514
Log loss on Test Data using Random Model 2.46748575083
-------------------- Confusion matrix --------------------
```




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



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



3.3 Univariate Analysis

```
In [106]: aa=pd.DataFrame({'type':['Random model'],'hyperparameter':['NA'],'log loss CV':[log_l
                               'log loss Test':[log_loss(y_test,test_predicted_y, eps=1e-15)] })

          # code for response coding with Laplace smoothing.
          # alpha : used for laplace smoothing
          # feature: ['gene', 'variation']
          # df: ['train_df', 'test_df', 'cv_df']
          # algorithm
          # ----------
          # Consider all unique values and the number of occurances of given feature in train d
          # build a vector (1*9) , the first element = (number of times it occured in class1 +
```

```
# gv_dict is like a look up table, for every gene it store a (1*9) representation of
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

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

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

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

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

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

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

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177,
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366,
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.06818181818
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295,
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334,
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

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

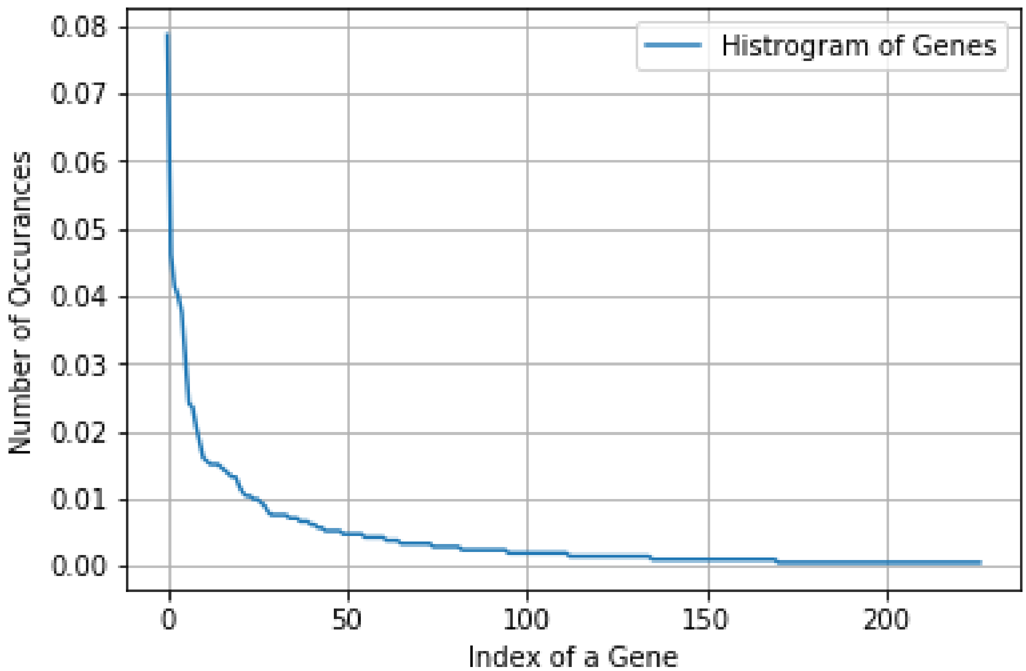
```
In [107]: unique_genes = train_df['Gene'].value_counts()
          print('Number of Unique Genes :', unique_genes.shape[0])
          # the top 10 genes that occured most
          print(unique_genes.head(10))
```

```
Number of Unique Genes : 227
BRCA1     167
TP53       98
PTEN       88
EGFR       85
BRCA2      80
BRAF       65
ERBB2      51
KIT        50
ALK        44
PDGFRA     39
Name: Gene, dtype: int64
```

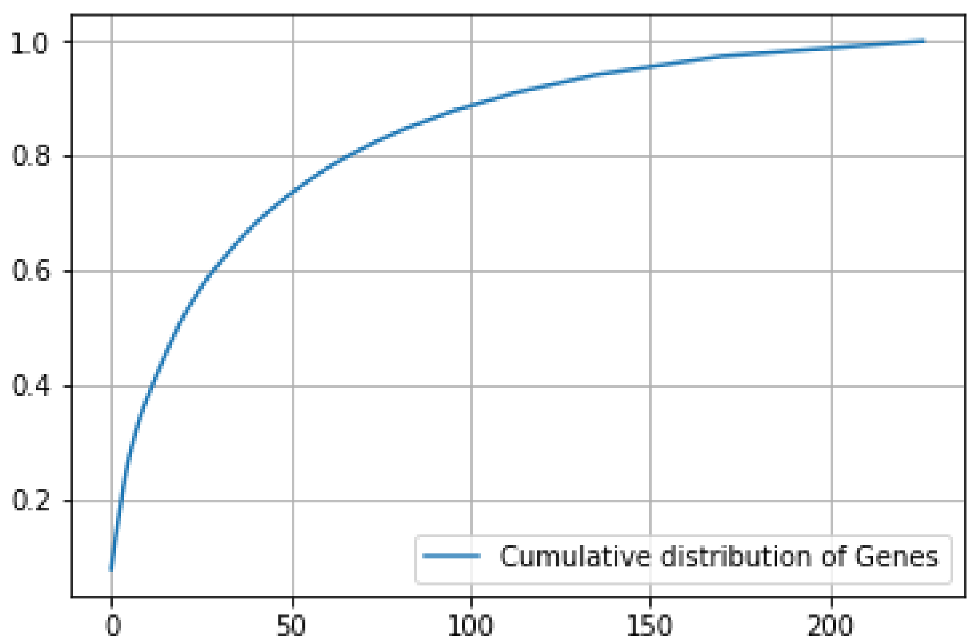
```
In [108]: print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the
```

```
Ans: There are 227 different categories of genes in the train data, and they are distibuted as
```

```
In [109]: 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 [110]: 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/

One hot Encoding

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 [111]:
```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", train_df)
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
```

In [112]:
```
print("train_gene_feature_responseCoding is converted feature using respone coding me
```

```
train_gene_feature_responseCoding is converted feature using respone coding method. The shape o
```

In [113]:
```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [114]:
```
train_df['Gene'].head()
```

Out[114]:
```
2145    KEAP1
1399    FGFR3
2909      NF2
1524      ALK
1424    FGFR3
Name: Gene, dtype: object
```

In [115]:
```
gene_vectorizer.get_feature_names()[0:5]
```

Out[115]:
```
['abl1', 'acvr1', 'ago2', 'akt1', 'akt2']
```

In [116]:
```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding me
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape o
```

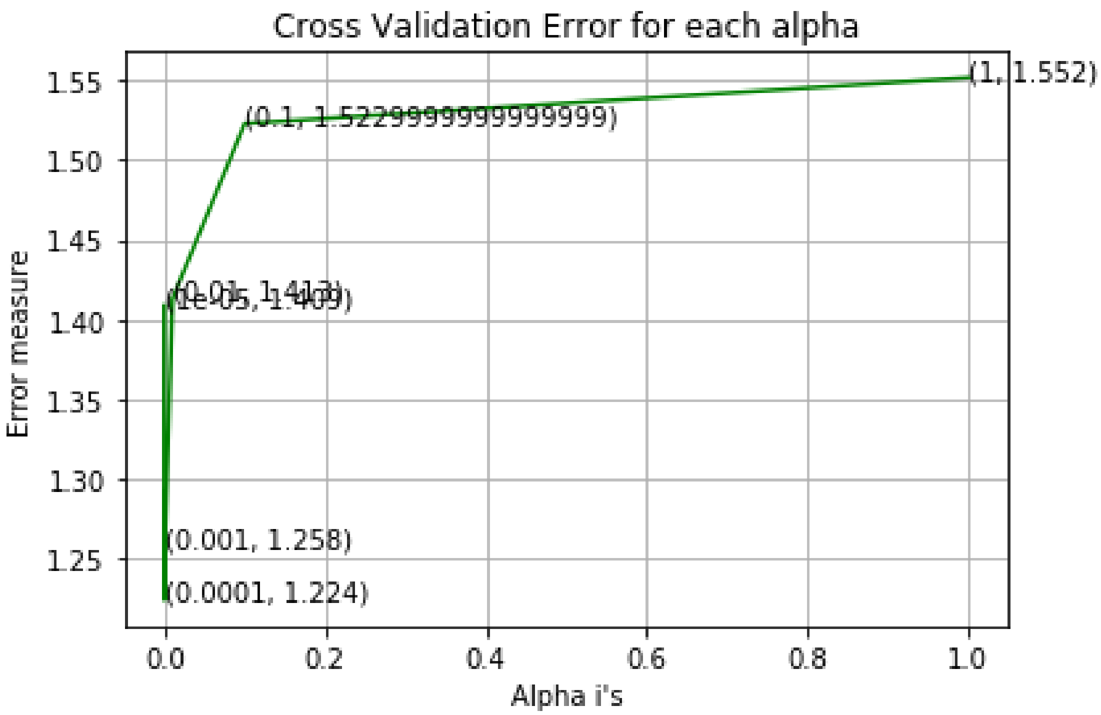
Q4. How good is this gene feature in predicting y_i?

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 [117]:
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generate
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=T
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=o
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_gene_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, 

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


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

```
    sig_clf.fit(train_gene_feature_onehotCoding, y_train)

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

```
For values of alpha =  1e-05 The log loss is: 1.40869181558
For values of alpha =  0.0001 The log loss is: 1.22390477683
For values of alpha =  0.001 The log loss is: 1.25829114541
For values of alpha =  0.01 The log loss is: 1.41280103967
For values of alpha =  0.1 The log loss is: 1.52285303622
For values of alpha =  1 The log loss is: 1.55164563141
```




```
For values of best alpha =  0.0001 The train log loss is: 1.0479972557
For values of best alpha =  0.0001 The cross validation log loss is: 1.22390477683
For values of best alpha =  0.0001 The test log loss is: 1.22042075712
```

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

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

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

```
Q6. How many data points in Test and CV datasets are covered by the  227  genes in train datase
Ans
1. In test data 638 out of 665 : 95.93984962406014
2. In cross validation data 511 out of  532 : 96.05263157894737
```

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?

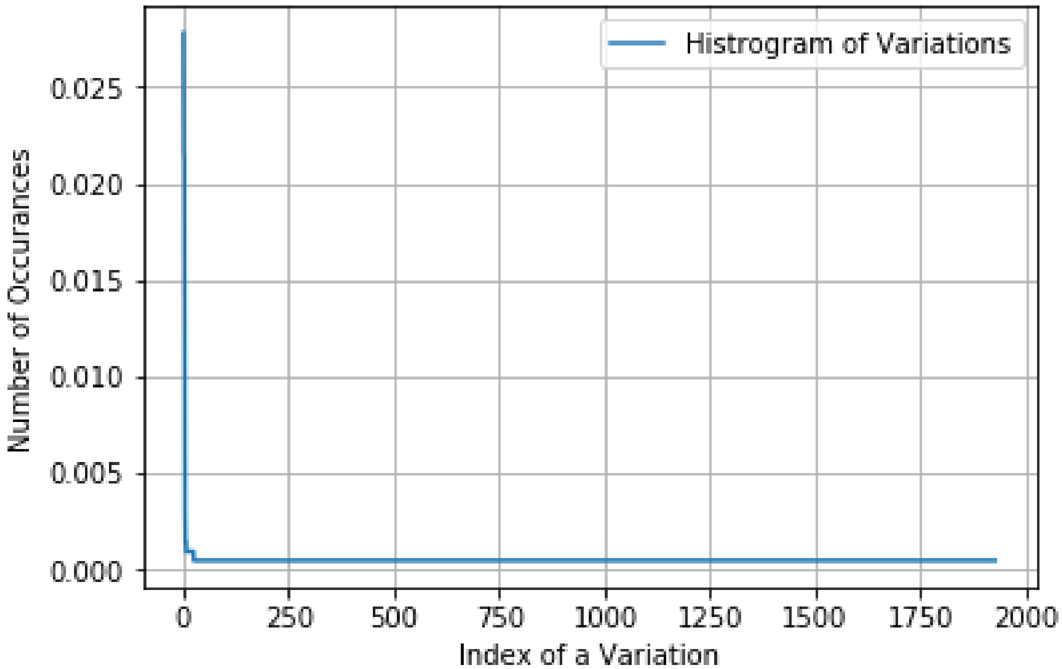
```
In [119]: unique_variations = train_df['Variation'].value_counts()
          print('Number of Unique Variations :', unique_variations.shape[0])
          # the top 10 variations that occured most
          print(unique_variations.head(10))
```

```
Number of Unique Variations : 1927
Truncating_Mutations    59
Deletion                46
Amplification           45
Fusions                 24
Overexpression           6
G12V                     3
T58I                     3
T73I                     2
E330K                    2
A146V                    2
Name: Variation, dtype: int64
```

```
In [120]: print("Ans: There are", unique_variations.shape[0] ,"different categories of variatio
```

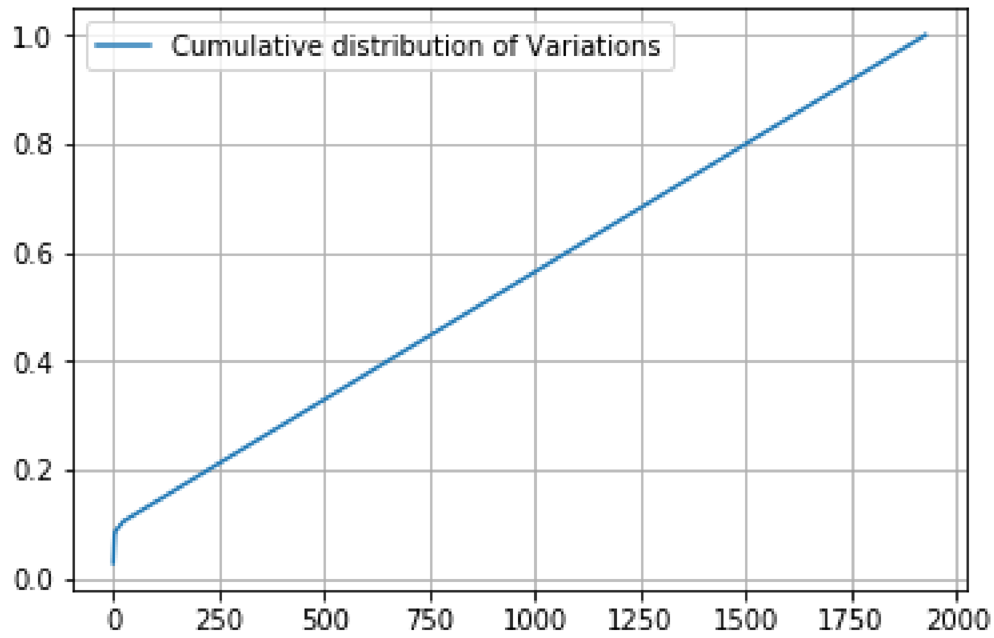
```
Ans: There are 1927 different categories of variations in the train data, and they are distibut
```

```
In [121]: 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 [122]: c = np.cumsum(h)
          print(c)
          plt.plot(c,label='Cumulative distribution of Variations')
          plt.grid()
          plt.legend()
          plt.show()
```

```
[ 0.02777778  0.04943503  0.07062147 ...,  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/

One hot Encoding

Response coding

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

In [123]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation",
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation",
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_
```

In [124]:
```
print("train_variation_feature_responseCoding is a converted feature using the respo
```

```
train_variation_feature_responseCoding is a converted feature using the response coding method
```

In [125]:
```
# one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variati
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation']
```

In [126]:
```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot
```

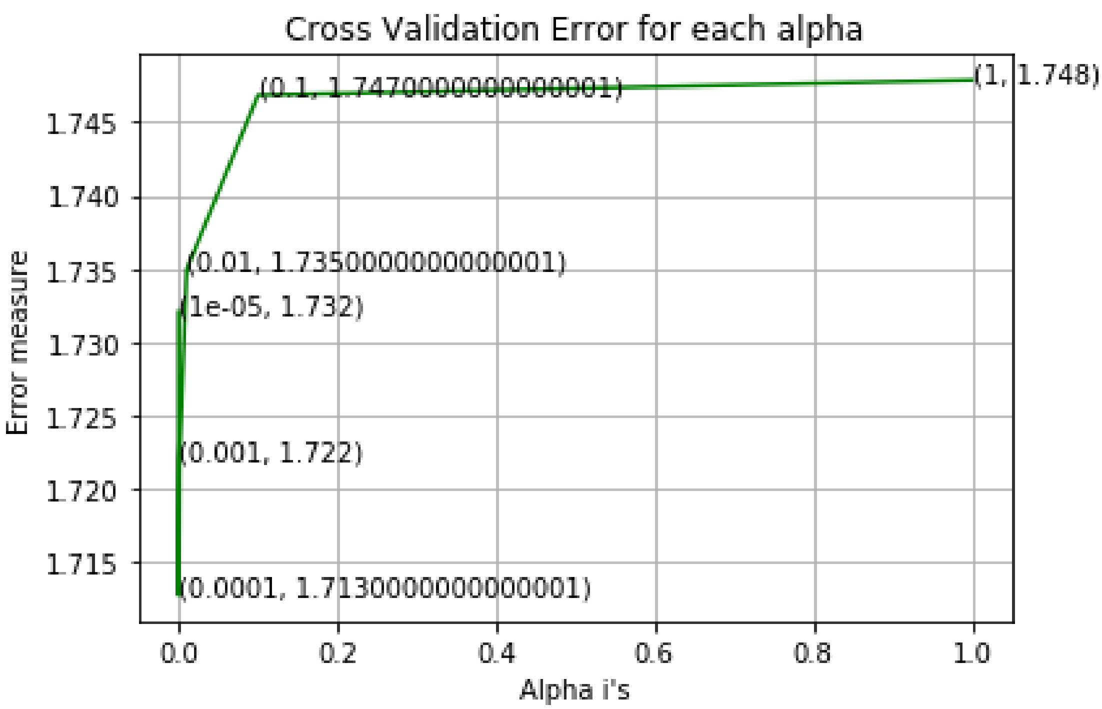
```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method.
```

Q10. How good is this Variation feature in predicting y_i?
Let's build a model just like the earlier!

In [127]:
```
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generate
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=T
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=o
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_variation_feature_onehotCoding, y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_variation_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)

    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, l

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

```
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=4
clf.fit(train_variation_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_variation_feature_onehotCoding, y_train)

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

```
For values of alpha =  1e-05 The log loss is: 1.73207265006
For values of alpha =  0.0001 The log loss is: 1.71274508902
For values of alpha =  0.001 The log loss is: 1.72200732949
For values of alpha =  0.01 The log loss is: 1.73492937759
For values of alpha =  0.1 The log loss is: 1.74685436209
For values of alpha =  1 The log loss is: 1.74788391719
```




```
For values of best alpha =  0.0001 The train log loss is: 0.768069507202
For values of best alpha =  0.0001 The cross validation log loss is: 1.71274508902
```

```
For values of best alpha =  0.0001 The test log loss is: 1.7026439441
```

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

```
Q12. How many data points are covered by total  1927  genes in test and cross validation data s
Ans
1. In test data 70 out of 665 : 10.526315789473683
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 [129]: # 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 [130]: import math
          #https://stackoverflow.com/a/1602964
          def get_text_responsecoding(df):
              text_feature_responseCoding = np.zeros((df.shape[0],9))
              for i in range(0,9):
                  row_index = 0
                  for index, row in df.iterrows():
                      sum_prob = 0
                      for word in row['TEXT'].split():
                          sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_dict.get
```

```
                text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['T
                row_index += 1
        return text_feature_responseCoding
```

In [131]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in trai
text_vectorizer_onehotCoding = CountVectorizer(min_df=3)
train_text_feature_onehotCoding = text_vectorizer_onehotCoding.fit_transform(train_d

#SMUK
# getting all the feature names (words)
train_text_features_1= text_vectorizer_onehotCoding.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*n
train_text_fea_counts_1 = train_text_feature_onehotCoding.sum(axis=0).A1

# zip(list(text_features),text_fea_counts) will zip a word with its number of times
text_fea_dict_1 = dict(zip(list(train_text_features_1),train_text_fea_counts_1))

print("Total number of unique words in train data BOW: shape", len(train_text_feature

# building a CountVectorizer with all the words that occured minimum 3 times in trai
text_vectorizer_ngram = CountVectorizer(min_df=3,ngram_range=(1,4))
train_text_feature_ngram = text_vectorizer_ngram.fit_transform(train_df['TEXT'])

# getting all the feature names (words)
train_text_features_2= text_vectorizer_ngram.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*n
train_text_fea_counts_2 = train_text_feature_ngram.sum(axis=0).A1

# zip(list(text_features),text_fea_counts) will zip a word with its number of times
text_fea_dict_2 = dict(zip(list(train_text_features_2),train_text_fea_counts_2))

print("Total number of unique words in train data ngram: shape", len(train_text_featu


# building a TFIDFVectorizer with all the words that occured minimum 3 times in trai
text_vectorizer_tfidf = TfidfVectorizer(min_df=3)
train_text_feature_tfidf = text_vectorizer_tfidf.fit_transform(train_df['TEXT'])

# getting all the feature names (words)
train_text_features_3= text_vectorizer_tfidf.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*n
train_text_fea_counts_3 = train_text_feature_tfidf.sum(axis=0).A1

# zip(list(text_features),text_fea_counts) will zip a word with its number of times
text_fea_dict_3 = dict(zip(list(train_text_features_3),train_text_fea_counts_3))
```

```
print("Total number of unique words in train data tfidf: shape", len(train_text_featu


# building a TFIDFVectorizer with all the words that occured minimum 3 times in trai
text_vectorizer_tfidf1000 = TfidfVectorizer(min_df=3)
train_text_feature_tfidf = text_vectorizer_tfidf1000.fit_transform(train_df['TEXT'])

#Take top 1000 words start here
indices = np.argsort(text_vectorizer_tfidf1000.idf_)[::-1]
features = text_vectorizer_tfidf1000.get_feature_names()
top_features = [features[i] for i in indices[:2000]]
#add the other feature in stopwords
bottom_features=[features[i] for i in indices[2000:]]
print(top_features[0:10])
#print feature and tfidf score
idf = text_vectorizer_tfidf1000.idf_
#print(dict(zip(text_vectorizer.get_feature_names(), idf)))
text_vectorizer_tfidf1000 = TfidfVectorizer(min_df=3,stop_words=bottom_features)
train_text_feature_tfidf1000 = text_vectorizer_tfidf1000.fit_transform(train_df['TEX

# getting all the feature names (words)
train_text_features_4 = text_vectorizer_tfidf1000.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*n
train_text_fea_counts_4 = train_text_feature_tfidf1000.sum(axis=0).A1

# zip(list(text_features),text_fea_counts) will zip a word with its number of times
text_fea_dict_4 = dict(zip(list(train_text_features_4),train_text_fea_counts_4))

print("Total number of unique words in train data tfidf1000: shape", len(train_text_
```

```
Total number of unique words in train data BOW: shape 53349 (2124, 53349) (2124,)
Total number of unique words in train data ngram: shape 3010578 (2124, 3010578)
Total number of unique words in train data tfidf: shape 53349 (2124, 53349)
['rebbeck', 'evp', 'etv', 'eufa1341', 'eukaryotes3', 'euphorbiae', 'mohler', 'eurohear', 'subg
Total number of unique words in train data tfidf1000: shape 2000 (2124, 2000)
```

```
In [132]: 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)

#train_text_features SMUK 1:bow,2:ngram,3:tfidf 4:tfidf1000
confuse_array_1 = []
for i in train_text_features_1:
    ratios = []
    max_val = -1
    for j in range(0,9):
        ratios.append((dict_list[j][i]+10 )/(total_dict[i]+90))
    confuse_array_1.append(ratios)
confuse_array_1 = np.array(confuse_array_1)

confuse_array_2 = []
for i in train_text_features_2:
    ratios = []
    max_val = -1
    for j in range(0,9):
        ratios.append((dict_list[j][i]+10 )/(total_dict[i]+90))
    confuse_array_2.append(ratios)
confuse_array_2 = np.array(confuse_array_2)

confuse_array_3 = []
for i in train_text_features_3:
    ratios = []
    max_val = -1
    for j in range(0,9):
        ratios.append((dict_list[j][i]+10 )/(total_dict[i]+90))
    confuse_array_3.append(ratios)
confuse_array_3 = np.array(confuse_array_3)

confuse_array_4 = []
for i in train_text_features_4:
    ratios = []
    max_val = -1
    for j in range(0,9):
        ratios.append((dict_list[j][i]+10 )/(total_dict[i]+90))
    confuse_array_4.append(ratios)
confuse_array_4 = np.array(confuse_array_4)
```

In [133]:
```
#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 [134]:
```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.T/train_text_
```

```
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_fea
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_re
```

```
In [135]: # don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

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

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


train_text_feature_ngramg = normalize(train_text_feature_ngram, axis=0)
test_text_feature_ngram = text_vectorizer_ngram.transform(test_df['TEXT'])
test_text_feature_ngram = normalize(test_text_feature_ngram, axis=0)
cv_text_feature_ngram = text_vectorizer_ngram.transform(cv_df['TEXT'])
cv_text_feature_ngram = normalize(cv_text_feature_ngram, axis=0)


train_text_feature_tfidf = normalize(train_text_feature_tfidf, axis=0)
test_text_feature_tfidf = text_vectorizer_tfidf.transform(test_df['TEXT'])
test_text_feature_tfidf = normalize(test_text_feature_tfidf, axis=0)
cv_text_feature_tfidf = text_vectorizer_tfidf.transform(cv_df['TEXT'])
cv_text_feature_tfidf = normalize(cv_text_feature_tfidf, axis=0)

train_text_feature_tfidf1000 = normalize(train_text_feature_tfidf1000, axis=0)
test_text_feature_tfidf1000 = text_vectorizer_tfidf1000.transform(test_df['TEXT'])
test_text_feature_tfidf1000 = normalize(test_text_feature_tfidf1000, axis=0)
cv_text_feature_tfidf1000 = text_vectorizer_tfidf1000.transform(cv_df['TEXT'])
cv_text_feature_tfidf1000 = normalize(cv_text_feature_tfidf1000, axis=0)
```

```
In [136]: #https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict_1 = dict(sorted(text_fea_dict_1.items(), key=lambda x: x[1] , r
sorted_text_occur_1 = np.array(list(sorted_text_fea_dict_1.values()))

sorted_text_fea_dict_2 = dict(sorted(text_fea_dict_2.items(), key=lambda x: x[1] , r
sorted_text_occur_2 = np.array(list(sorted_text_fea_dict_2.values()))

sorted_text_fea_dict_3 = dict(sorted(text_fea_dict_3.items(), key=lambda x: x[1] , r
sorted_text_occur_3 = np.array(list(sorted_text_fea_dict_3.values()))

sorted_text_fea_dict_4 = dict(sorted(text_fea_dict_4.items(), key=lambda x: x[1] , r
sorted_text_occur_4 = np.array(list(sorted_text_fea_dict_4.values()))
```

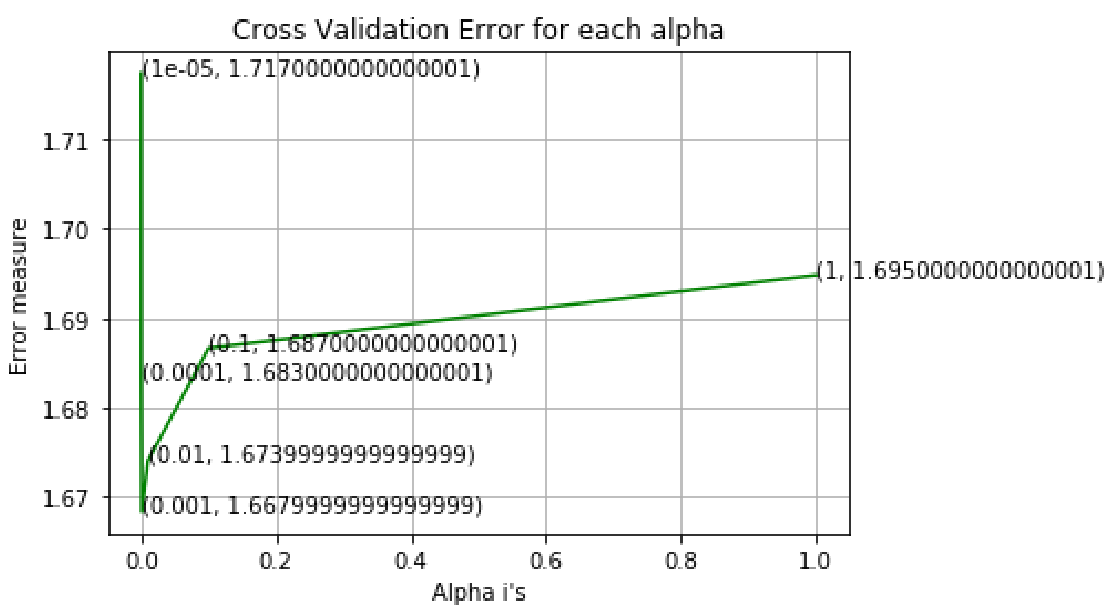
```
In [137]: # Number of words for a given frequency.
          print(Counter(sorted_text_occur_1[0:10]))
          print(Counter(sorted_text_occur_2[0:10]))
          print(Counter(sorted_text_occur_3[0:10]))
          print(Counter(sorted_text_occur_4[0:10]))
          print(len(train_df['TEXT']))
```

```
Counter({4: 2, 433: 1, 197: 1, 6: 1, 9: 1, 10: 1, 13: 1, 2158: 1, 1263: 1})
Counter({3: 5, 16: 1, 4: 1, 5: 1, 8: 1, 9: 1})
Counter({0.07397031698369963: 1, 0.035920396892100172: 1, 2.4179885527512641: 1, 3.99529951865
Counter({0.08303727786663323: 1, 0.32794866168699111: 1, 0.49897847740610007: 1, 0.11914522061
2124
```

```
In [138]: # Train a Logistic regression+Calibration model using text features whicha re tfidf
          alpha = [10 ** x for x in range(-5, 1)]

          # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generate
          # ------------------------------
          # default parameters
          # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=T
          # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=o
          # class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


          cv_log_error_array=[]
          for i in alpha:
              clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
              clf.fit(train_text_feature_tfidf1000, y_train)

              sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
              sig_clf.fit(train_text_feature_tfidf1000, y_train)
              predict_y = sig_clf.predict_proba(cv_text_feature_tfidf1000)
              cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-
              print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, l

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

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=4
clf.fit(train_text_feature_tfidf1000, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_text_feature_tfidf1000, y_train)

predict_y = sig_clf.predict_proba(train_text_feature_tfidf1000)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_text_feature_tfidf1000)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log los
predict_y = sig_clf.predict_proba(test_text_feature_tfidf1000)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_l
```

```
For values of alpha =  1e-05 The log loss is: 1.71740434506
For values of alpha =  0.0001 The log loss is: 1.68341832777
For values of alpha =  0.001 The log loss is: 1.66836480869
For values of alpha =  0.01 The log loss is: 1.6740567879
For values of alpha =  0.1 The log loss is: 1.68664485678
For values of alpha =  1 The log loss is: 1.69482899879
```



```
For values of best alpha =  0.001 The train log loss is: 1.49295849378
For values of best alpha =  0.001 The cross validation log loss is: 1.66836480869
For values of best alpha =  0.001 The test log loss is: 1.69609464973
```

Q. Is the Text feature stable across all the data sets (Test, Train, Cross validation)?
Ans. Yes, it seems like!

In [139]:
```python
def get_intersec_text(df,type=1):
    df_text_vec = CountVectorizer(min_df=3)
    if type==2:
        df_text_vec = CountVectorizer(min_df=3,ngram_range=(1,4))
    if type==3:
        df_text_vec = TfidfVectorizer(min_df=3)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

    df_text_fea_counts = df_text_fea.sum(axis=0).A1
    df_text_fea_dict = dict(zip(list(df_text_features),df_text_fea_counts))
    len1 = len(set(df_text_features))
    len2 = len(set(train_text_features_1) & set(df_text_features))
    if type==2:
        len2 = len(set(train_text_features_2) & set(df_text_features))
    if type==3:
        len2 = len(set(train_text_features_2) & set(df_text_features))

    return len1,len2
```

In [140]:
```python
len1,len2 = get_intersec_text(test_df,1)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train data fo
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in train

len1,len2 = get_intersec_text(test_df,2)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train data fo
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in train

len1,len2 = get_intersec_text(test_df,3)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train data fo
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in train
```

```
97.473 % of word of test data appeared in train data for bow
97.346 % of word of Cross Validation appeared in train data for bow
94.134 % of word of test data appeared in train data for ngram
97.346 % of word of Cross Validation appeared in train data for ngram
97.473 % of word of test data appeared in train data for tfidf
```

```
97.346 % of word of Cross Validation appeared in train data for tfidf
```

4. Machine Learning Models

```
In [141]: #Data preparation for ML models.

          #Misc. functionns for ML models


          def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y, clf):
              clf.fit(train_x, train_y)
              sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
              sig_clf.fit(train_x, train_y)
              pred_y = sig_clf.predict(test_x)

              # for calculating log_loss we willl provide the array of probabilities belongs t
              print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
              # calculating the number of data points that are misclassified
              print("Number of mis-classified points :", np.count_nonzero((pred_y- test_y))/tes
              plot_confusion_matrix(test_y, pred_y)
```

```
In [142]: 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 [143]: # this function will be used just for naive bayes
          # for the given indices, we will print the name of the features
          # and we will check whether the feature present in the test point text or not
          def get_impfeature_names(indices, text, gene, var, no_features):
              gene_count_vec = CountVectorizer()
              var_count_vec = CountVectorizer()
              text_count_vec = CountVectorizer(min_df=3)

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

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

              word_present = 0
              for i,v in enumerate(indices):
                  if (v < fea1_len):
                      word = gene_vec.get_feature_names()[v]
```

```
                yes_no = True if word == gene else False
                if yes_no:
                    word_present += 1
                    print(i, "Gene feature [{}] present in test data point [{}]".format(w
            elif (v < fea1_len+fea2_len):
                word = var_vec.get_feature_names()[v-(fea1_len)]
                yes_no = True if word == var else False
                if yes_no:
                    word_present += 1
                    print(i, "variation feature [{}] present in test data point [{}]".fo
            else:
                word = text_vec.get_feature_names()[v-(fea1_len+fea2_len)]
                yes_no = True if word in text.split() else False
                if yes_no:
                    word_present += 1
                    print(i, "Text feature [{}] present in test data point [{}]".format(w

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

Stacking the three types of features

In [144]:
```
# merging gene, variance and text features

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

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_fe
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_

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

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

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

#apply ngram on text and onehotCoding in gene and variation
train_x_ngram = hstack((train_gene_var_onehotCoding, train_text_feature_ngram)).tocsr
test_x_ngram = hstack((test_gene_var_onehotCoding, test_text_feature_ngram)).tocsr()
cv_x_ngram = hstack((cv_gene_var_onehotCoding, cv_text_feature_ngram)).tocsr()
```

```
#apply tfidf on text and onehotCoding in gene and variation
train_x_tfidf = hstack((train_gene_var_onehotCoding, train_text_feature_tfidf)).tocs
test_x_tfidf = hstack((test_gene_var_onehotCoding, test_text_feature_tfidf)).tocsr()
cv_x_tfidf = hstack((cv_gene_var_onehotCoding, cv_text_feature_tfidf)).tocsr()

#apply tfidf(top1000 words) on text and onehotCoding in gene and variation
train_x_tfidf1000 = hstack((train_gene_var_onehotCoding, train_text_feature_tfidf100
test_x_tfidf1000 = hstack((test_gene_var_onehotCoding, test_text_feature_tfidf1000))
cv_x_tfidf1000 = hstack((cv_gene_var_onehotCoding, cv_text_feature_tfidf1000)).tocsr


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_va
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_vari
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_featur
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_r
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_respons
```

In [145]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_oneho
print("(number of data points * number of features) in test data = ", test_x_onehotC
print("(number of data points * number of features) in cross validation data =", cv_

print("ngram features :")
print("(number of data points * number of features) in train data = ", train_x_ngram
print("(number of data points * number of features) in test data = ", test_x_ngram.s
print("(number of data points * number of features) in cross validation data =", cv_

print("tfidf features :")
print("(number of data points * number of features) in train data = ", train_x_tfidf
print("(number of data points * number of features) in test data = ", test_x_tfidf.s
print("(number of data points * number of features) in cross validation data =", cv_

print("tfidf to 1000 words features :")
print("(number of data points * number of features) in train data = ", train_x_tfidf
print("(number of data points * number of features) in test data = ", test_x_tfidf10
print("(number of data points * number of features) in cross validation data =", cv_

print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_respo
print("(number of data points * number of features) in test data = ", test_x_respons
print("(number of data points * number of features) in cross validation data =", cv_
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 55528)
(number of data points * number of features) in test data =  (665, 55528)
```

```
(number of data points * number of features) in cross validation data = (532, 55528)
ngram features :
(number of data points * number of features) in train data =  (2124, 3012757)
(number of data points * number of features) in test data =  (665, 3012757)
(number of data points * number of features) in cross validation data = (532, 3012757)
tfidf features :
(number of data points * number of features) in train data =  (2124, 55528)
(number of data points * number of features) in test data =  (665, 55528)
(number of data points * number of features) in cross validation data = (532, 55528)
tfidf to 1000 words features :
(number of data points * number of features) in train data =  (2124, 4179)
(number of data points * number of features) in test data =  (665, 4179)
(number of data points * number of features) in cross validation data = (532, 4179)
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 27)
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data = (532, 27)
```

In [146]:
```
#Try feature engineering technique to use log of train_gene_var_onehotCoding
print(train_gene_var_onehotCoding.shape)
#first make same variable as without feature transformation
train_gene_var_feature=train_gene_var_onehotCoding
test_gene_var_feature=test_gene_var_onehotCoding
cv_gene_var_feature=cv_gene_var_onehotCoding

train_gene_var_feature.data=np.log(train_gene_var_onehotCoding.data+1)
test_gene_var_feature.data=np.log(test_gene_var_onehotCoding.data+1)
cv_gene_var_feature.data=np.log(cv_gene_var_onehotCoding.data+1)
print(train_gene_var_onehotCoding.shape)
print(train_gene_var_onehotCoding.data)

#apply ngram on text and onehotCoding+log transform in gene and variation
train_x_feature = hstack((train_gene_var_feature, train_text_feature_tfidf1000)).tocs
test_x_feature = hstack((test_gene_var_feature, test_text_feature_tfidf1000)).tocsr(
cv_x_feature = hstack((cv_gene_var_feature, cv_text_feature_tfidf1000)).tocsr()

print(" After log transformation on gene and variation features :")
print("(number of data points * number of features) in train data = ", train_x_featu
print("(number of data points * number of features) in test data = ", test_x_feature
print("(number of data points * number of features) in cross validation data =", cv_
```

```
(2124, 2179)
(2124, 2179)
[ 0.69314718  0.69314718  0.69314718 ...,  0.69314718  0.69314718
  0.69314718]
 After log transformation on gene and variation features :
(number of data points * number of features) in train data =  (2124, 4179)
```

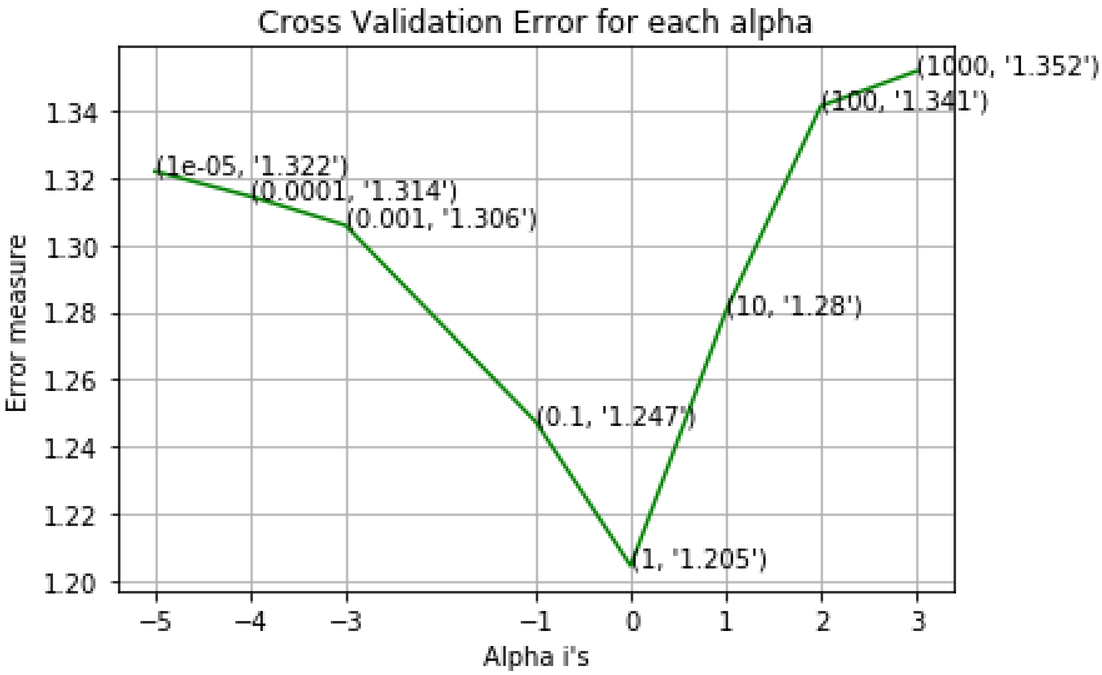
```
(number of data points * number of features) in test data =  (665, 4179)
(number of data points * number of features) in cross validation data = (532, 4179)
```

4.1. Base Line Model
4.1.1. Naive Bayes
4.1.1.1. Hyper parameter tuning

```
In [147]: # find more about Multinomial Naive base function here http://scikit-learn.org/stabl
          # -------------------------
          # default paramters
          # sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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

          #any dataset can be applied here like bow,tfidf,featurized,response coding
          #train_x_onehotCoding/train_x_ngram/train_x_tfidf/train_x_tfidf1000/train_x_feature(

          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_tfidf1000, train_y)
              sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
              sig_clf.fit(train_x_tfidf1000, train_y)
              sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf1000)
```

```
            cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=
            # to avoid rounding error while multiplying probabilites we use log-probability e
            print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


        best_alpha = np.argmin(cv_log_error_array)
        clf = MultinomialNB(alpha=alpha[best_alpha])
        clf.fit(train_x_tfidf1000, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_tfidf1000, train_y)


        predict_y = sig_clf.predict_proba(train_x_tfidf1000)
        print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
        predict_y = sig_clf.predict_proba(cv_x_tfidf1000)
        print('For values of best alpha = ', alpha[best_alpha], "The cross validation log los
        predict_y = sig_clf.predict_proba(test_x_tfidf1000)
        print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_l
```

```
for alpha = 1e-05
Log Loss : 1.32187296869
for alpha = 0.0001
Log Loss : 1.31445763862
for alpha = 0.001
Log Loss : 1.30581859155
for alpha = 0.1
Log Loss : 1.24746251061
for alpha = 1
Log Loss : 1.20455400661
for alpha = 10
Log Loss : 1.2801630638
for alpha = 100
Log Loss : 1.341346931
for alpha = 1000
Log Loss : 1.35170792813
```



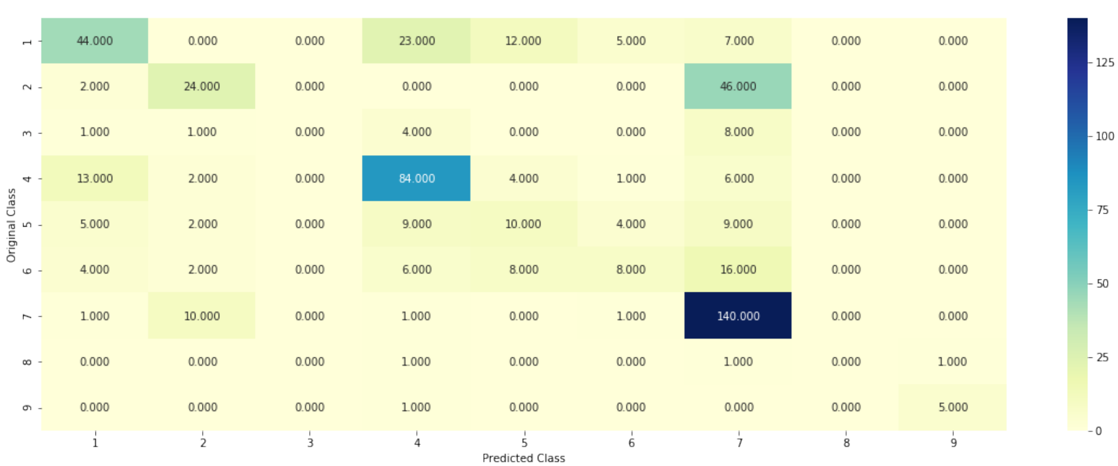

```
For values of best alpha =  1 The train log loss is: 0.75648434171
For values of best alpha =  1 The cross validation log loss is: 1.20455400661
For values of best alpha =  1 The test log loss is: 1.19806412359
```

4.1.1.2. Testing the model with best hyper paramters

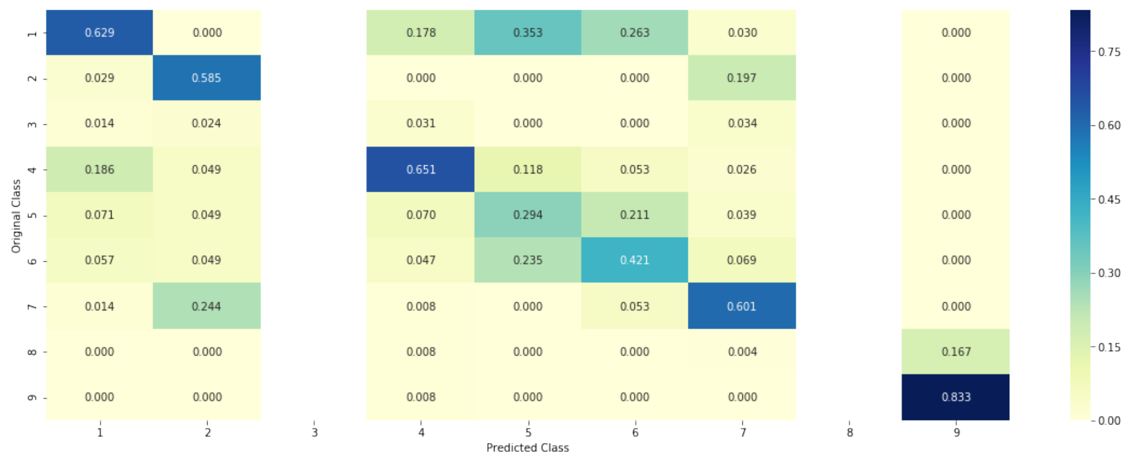
In [148]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stabl
# --------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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

# -----------------------


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

```
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])        Fit the calibrated model
# get_params([deep])        Get parameters for this estimator.
# predict(X)        Predict the target of new samples.
# predict_proba(X)        Posterior probabilities of classification
# ----------------------------

#any dataset can be applied here like bow,tfidf,featurized,response coding
#train_x_onehotCoding/train_x_ngram/train_x_tfidf/train_x_tfidf1000/train_x_feature(

clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_tfidf1000, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf1000, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf1000)
# to avoid rounding error while multiplying probabilites we use log-probability esti
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_tfic
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_tfidf1000.toarray()))
#print(str(2),alpha[best_alpha])
xx='alpha :'+str(alpha[best_alpha])
print(xx)
bb=pd.DataFrame({'type':['naive bayes'],'hyperparameter':[xx],'log loss CV':[log_los
                    'log loss Test':[log_loss(test_y, sig_clf.predict_proba(test_x_tf
aa=aa.append(bb)
```

```
Log Loss : 1.20455400661
Number of missclassified point : 0.40789473684210525
-------------------- Confusion matrix --------------------
```



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




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


```
alpha :1
```

4.1.1.3. Feature Importance, Correctly classified point

```
In [149]: test_point_index = 1
          no_feature = 100
          predicted_cls = sig_clf.predict(test_x_tfidf1000[test_point_index])
          print("Predicted Class :", predicted_cls[0])
```

```
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf1
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gen
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.0391  0.1306  0.0191  0.0503  0.0314  0.032   0.6872  0.00
Actual Class : 2
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

```
In [150]: test_point_index = 100
          no_feature = 100
          predicted_cls = sig_clf.predict(test_x_tfidf1000[test_point_index])
          print("Predicted Class :", predicted_cls[0])
          print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf1
          print("Actual Class :", test_y[test_point_index])
          indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
          print("-"*50)
          get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gen
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.0509  0.1456  0.0216  0.089   0.038   0.0409  0.6021  0.00
Actual Class : 7
--------------------------------------------------
96 Text feature [166] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

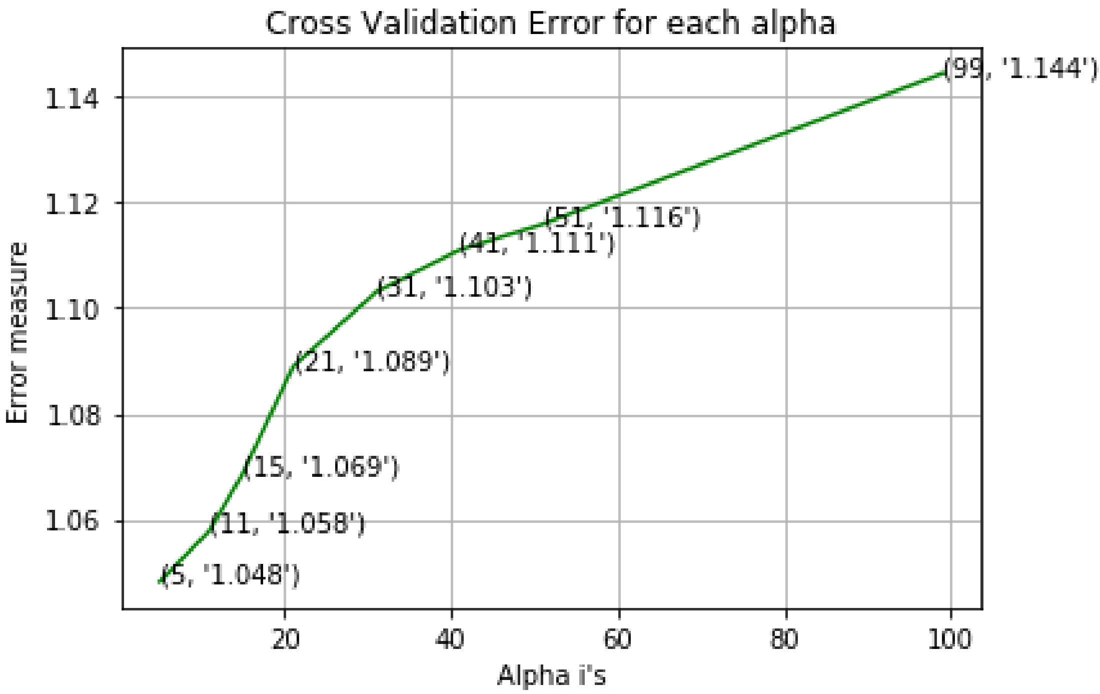
4.2. K Nearest Neighbour Classification
4.2.1. Hyper parameter tuning

```
In [151]: # find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules
          # -------------------------
          # default parameter
          # KNeighborsClassifier(n_neighbors=5, weights=uniform, algorithm=auto, leaf_size=30,
          # metric=minkowski, metric_params=None, n_jobs=1, **kwargs)

          # methods of
          # fit(X, y) : Fit the model using X as training data and y as target values
          # predict(X):Predict the class labels for the provided data
          # predict_proba(X):Return probability estimates for the test data X.
          #--------------------------------------
          # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lesson
          #--------------------------------------
```

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


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

```python
        print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
        predict_y = sig_clf.predict_proba(cv_x_responseCoding)
        print('For values of best alpha = ', alpha[best_alpha], "The cross validation log los
        predict_y = sig_clf.predict_proba(test_x_responseCoding)
        print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_l

        xx='k :'+str(alpha[best_alpha])
        bb=pd.DataFrame({'type':['knn'],'hyperparameter':[xx],'log loss CV':[log_loss(y_cv, s
                         'log loss Test':[log_loss(y_test, sig_clf.predict_proba(test_x_res
        aa=aa.append(bb)
```

```
for alpha = 5
Log Loss : 1.04818416132
for alpha = 11
Log Loss : 1.05789777417
for alpha = 15
Log Loss : 1.06875573315
for alpha = 21
Log Loss : 1.08885521802
for alpha = 31
Log Loss : 1.10304198087
for alpha = 41
Log Loss : 1.11097483162
for alpha = 51
Log Loss : 1.11586058868
for alpha = 99
Log Loss : 1.14436972809
```



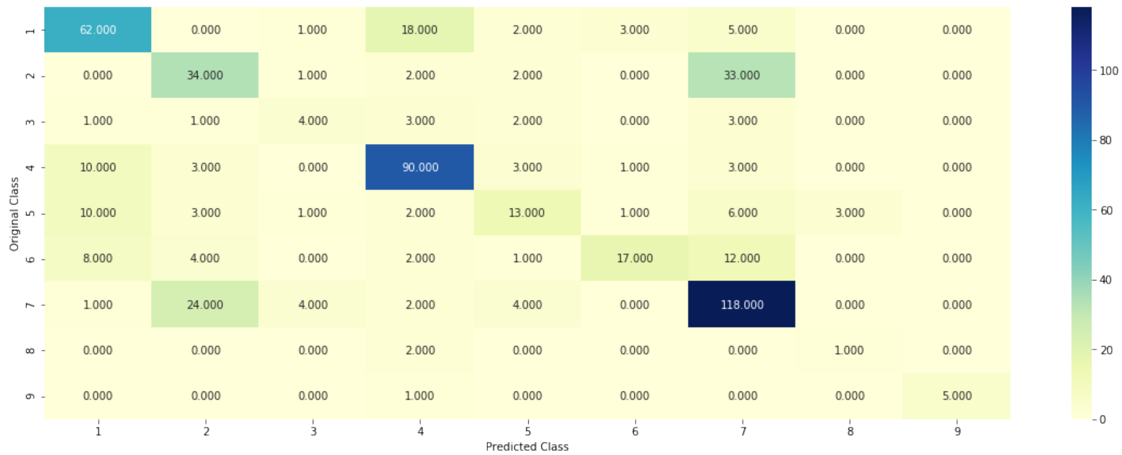

```
For values of best alpha =  5 The train log loss is: 0.481213924402
For values of best alpha =  5 The cross validation log loss is: 1.04818416132
For values of best alpha =  5 The test log loss is: 1.06327266269
```

4.2.2. Testing the model with best hyper paramters

```
In [152]: # find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/
          # -------------------------
          # default parameter
          # KNeighborsClassifier(n_neighbors=5, weights=uniform, algorithm=auto, leaf_size=30,
          # metric=minkowski, metric_params=None, n_jobs=1, **kwargs)

          # methods of
          # fit(X, y) : Fit the model using X as training data and y as target values
          # predict(X):Predict the class labels for the provided data
          # predict_proba(X):Return probability estimates for the test data X.
          #-------------------------------------
          # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lesson
          #-------------------------------------
          clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
          predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCodi
```

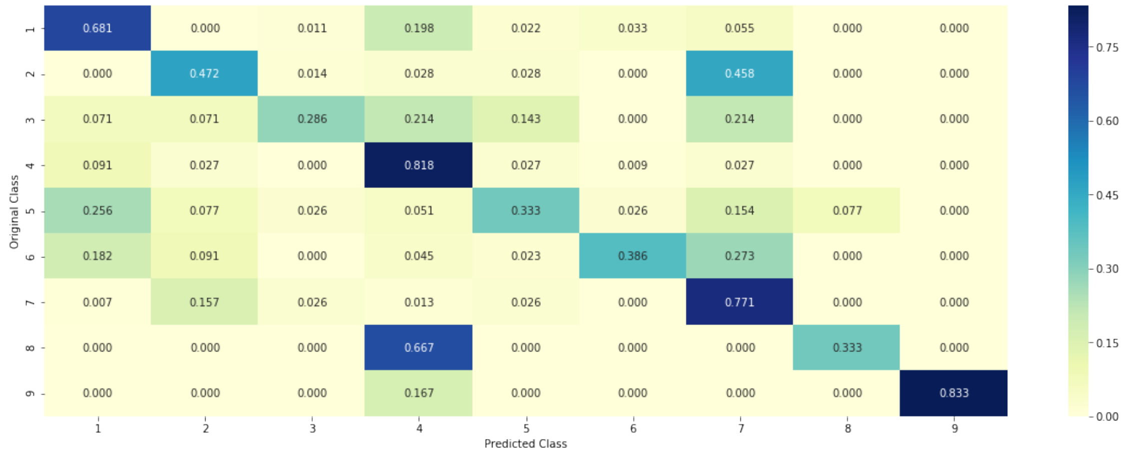
```
Log loss : 1.04818416132
Number of mis-classified points : 0.3533834586466165
```

-------------------- Confusion matrix --------------------



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

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



4.2.3.Sample Query point -1

```
In [153]: clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
          clf.fit(train_x_responseCoding, train_y)
          sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
          sig_clf.fit(train_x_responseCoding, train_y)

          test_point_index = 1
          predicted_cls = sig_clf.predict(test_x_responseCoding[0].reshape(1,-1))
          print("Predicted Class :", predicted_cls[0])
          print("Actual Class :", test_y[test_point_index])
          neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), al
          print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to cla
          print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 2
The  5  nearest neighbours of the test points belongs to classes [2 2 2 7 2]
Fequency of nearest points : Counter({2: 4, 7: 1})
```

4.2.4. Sample Query Point-2

```
In [154]: clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
          clf.fit(train_x_responseCoding, train_y)
          sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
          sig_clf.fit(train_x_responseCoding, train_y)

          test_point_index = 100

          predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1)
          print("Predicted Class :", predicted_cls[0])
          print("Actual Class :", test_y[test_point_index])
```

```
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), al
print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of the t
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

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

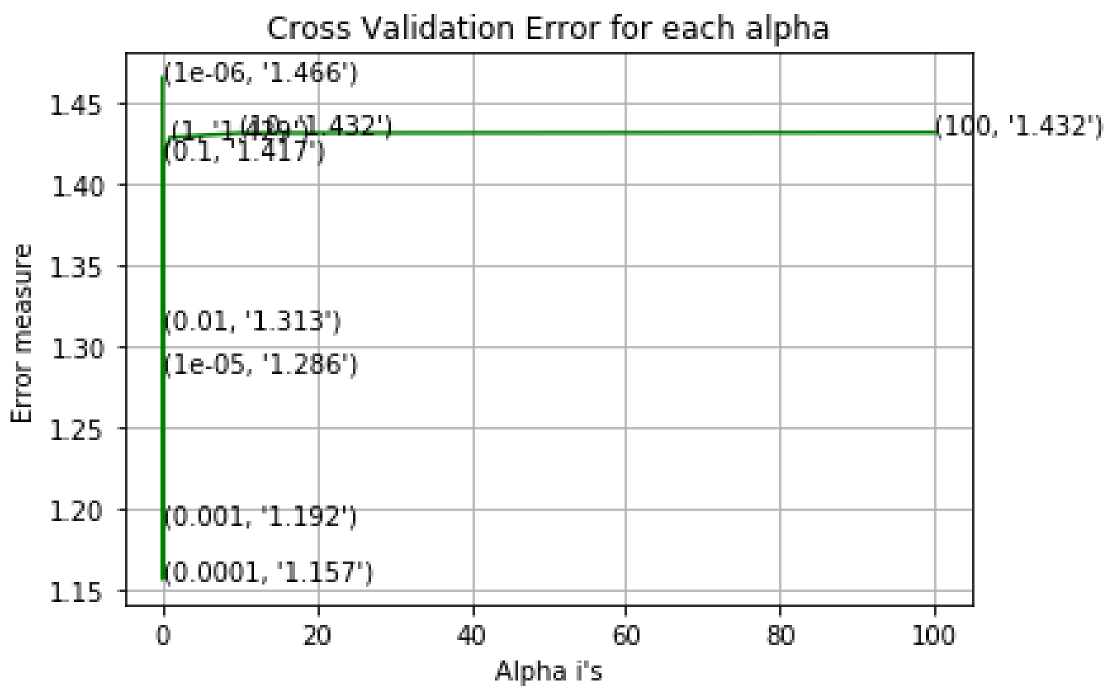
4.3. Logistic Regression
4.3.1. With Class balancing
4.3.1.1. Hyper paramter tuning

In [155]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generate
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=T
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modu
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv=
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])        Fit the calibrated model
# get_params([deep])        Get parameters for this estimator.
# predict(X)        Predict the target of new samples.
# predict_proba(X)        Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------
#any dataset can be applied here like bow,tfidf,featurized,response coding
#train_x_onehotCoding/train_x_ngram/train_x_tfidf/train_x_tfidf1000/train_x_feature(


alpha = [10 ** x for x in range(-6, 3)]
```

```
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', r
    clf.fit(train_x_feature, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_feature, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_feature)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=
    # to avoid rounding error while multiplying probabilites we use log-probability e
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', l
clf.fit(train_x_feature, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_feature, train_y)

predict_y = sig_clf.predict_proba(train_x_feature)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_feature)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log los
predict_y = sig_clf.predict_proba(test_x_feature)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_l

xx='C :'+str(alpha[best_alpha])
bb=pd.DataFrame({'type':['logistic'],'hyperparameter':[xx],'log loss CV':[log_loss(y_
                  'log loss Test':[log_loss(y_test, sig_clf.predict_proba(test_x_fea
aa=aa.append(bb)
```

```
for alpha = 1e-06
Log Loss : 1.46579210949
for alpha = 1e-05
Log Loss : 1.28562332957
for alpha = 0.0001
Log Loss : 1.15705605143
for alpha = 0.001
```

```
Log Loss : 1.19198303174
for alpha = 0.01
Log Loss : 1.31264856627
for alpha = 0.1
Log Loss : 1.41721084724
for alpha = 1
Log Loss : 1.42900133271
for alpha = 10
Log Loss : 1.43177672884
for alpha = 100
Log Loss : 1.43216404682
```



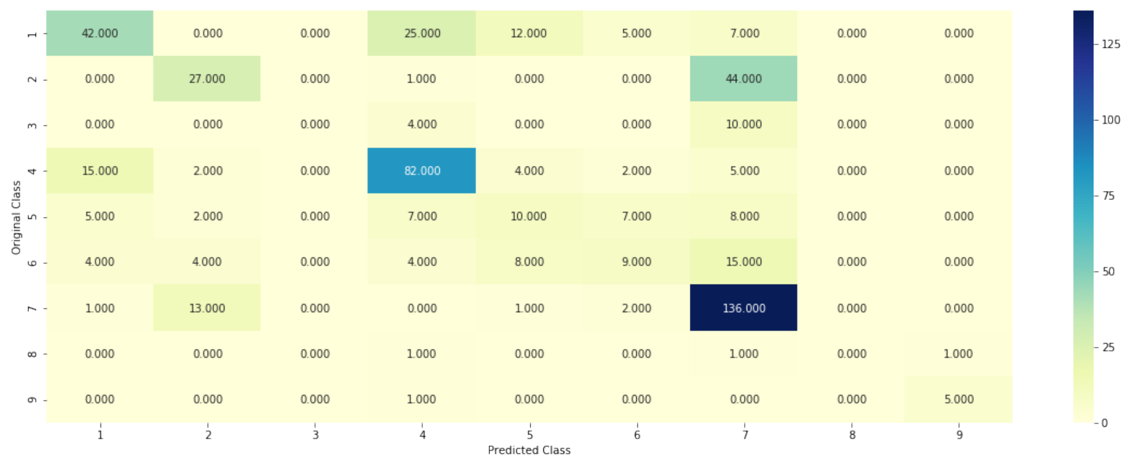

```
For values of best alpha =  0.0001 The train log loss is: 0.580073852145
For values of best alpha =  0.0001 The cross validation log loss is: 1.15705605143
For values of best alpha =  0.0001 The test log loss is: 1.14102909951
```

4.3.1.2. Testing the model with best hyper paramters

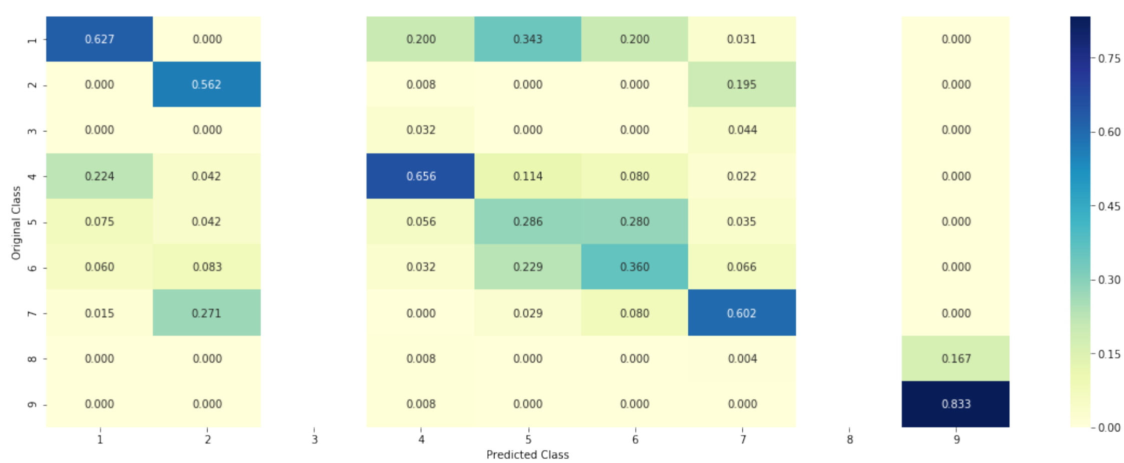
In [156]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generate
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=T
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=o
```

```
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lesson
#------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', l
predict_and_plot_confusion_matrix(train_x_feature, train_y, cv_x_feature, cv_y, clf)
```

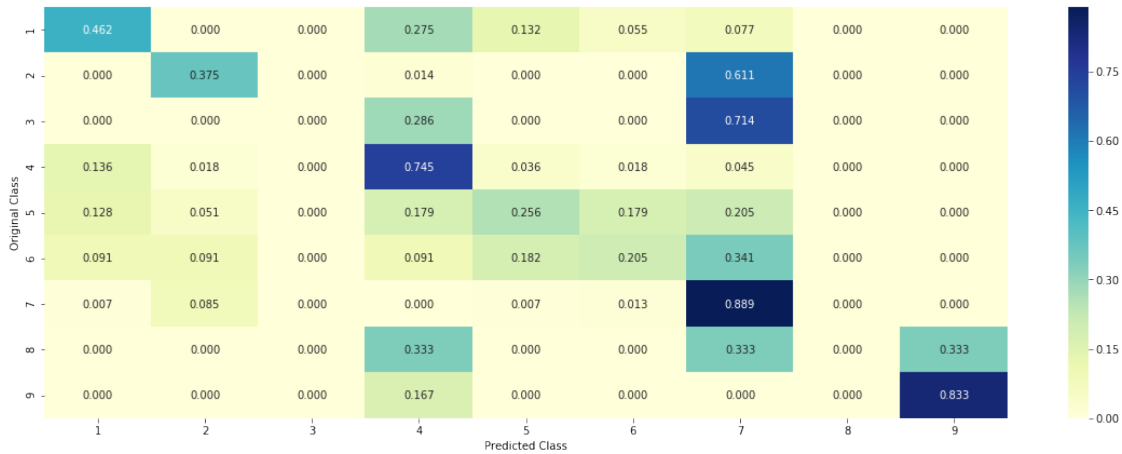
```
Log loss : 1.15705605143
Number of mis-classified points : 0.41541353383458646
-------------------- Confusion matrix --------------------
```



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



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



4.3.1.3. Feature Importance

```
In [157]: 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_feature.shape[1]:
                      tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
                  elif i< 18:
                      tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
                  if ((i > 17) & (i not in removed_ind)) :
                      word = train_text_features[i]
                      yes_no = True if word in text.split() else False
                      if yes_no:
                          word_present += 1
                      tabulte_list.append([incresingorder_ind,train_text_features[i], yes_no])
                  incresingorder_ind += 1
              print(word_present, "most importent features are present in our query point")
              print("-"*50)
              print("The features that are most importent of the ",predicted_cls[0]," class:")
              print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or Not']
```

4.3.1.3.1. Correctly Classified point

```
In [158]: # from tabulate import tabulate
          clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
```

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

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.0227  0.2563  0.0166  0.0463  0.0368  0.0147  0.5937  0.005
Actual Class : 2
--------------------------------------------------
239 Text feature [156027120] present in test data point [True]
281 Text feature [19ex] present in test data point [True]
357 Text feature [12] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

```
In [159]: test_point_index = 100
          no_feature = 500
          predicted_cls = sig_clf.predict(test_x_feature[test_point_index])
          print("Predicted Class :", predicted_cls[0])
          print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_feature
          print("Actual Class :", test_y[test_point_index])
          indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
          print("-"*50)
          get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene
```

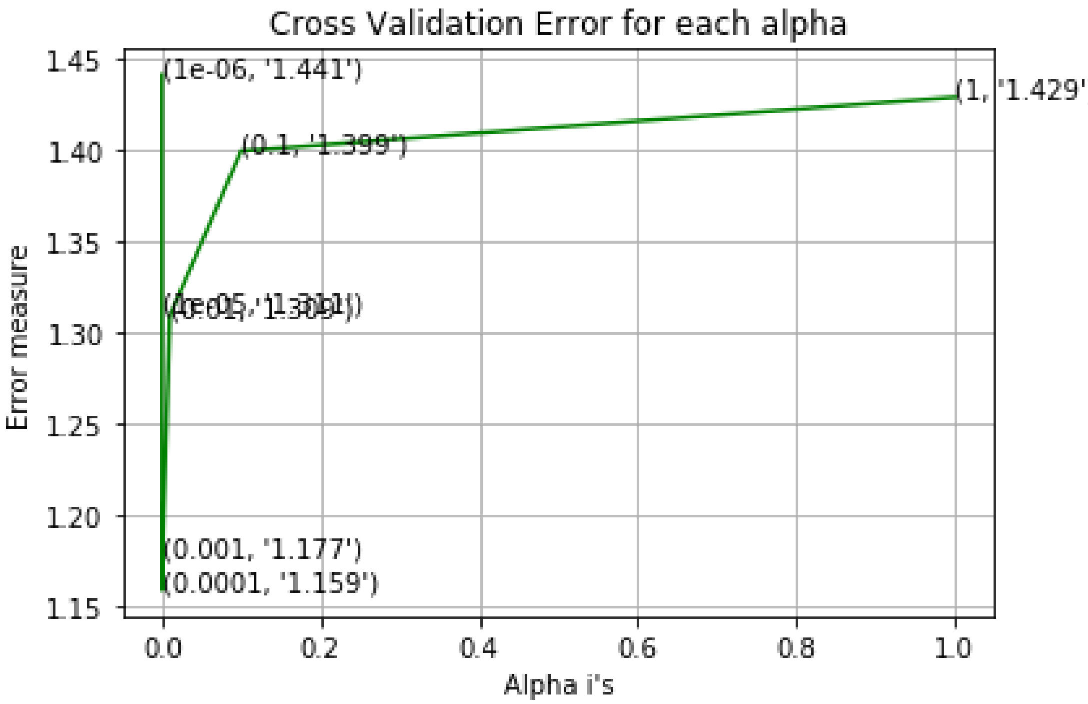
```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.0247  0.1112  0.0112  0.0939  0.04    0.0586  0.6476  0.004
Actual Class : 7
--------------------------------------------------
108 Text feature [153] present in test data point [True]
119 Text feature [188] present in test data point [True]
176 Text feature [106] present in test data point [True]
182 Text feature [171] present in test data point [True]
259 Text feature [194] present in test data point [True]
271 Text feature [158] present in test data point [True]
357 Text feature [12] present in test data point [True]
378 Text feature [190] present in test data point [True]
Out of the top  500  features  8 are present in query point
```

4.3.2. Without Class balancing
4.3.2.1. Hyper paramter tuning

In [160]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generate
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=T
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=o
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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



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

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

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

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=4
clf.fit(train_x_feature, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_feature, train_y)

predict_y = sig_clf.predict_proba(train_x_feature)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_feature)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log los
predict_y = sig_clf.predict_proba(test_x_feature)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_l

xx='C :'+str(alpha[best_alpha])
bb=pd.DataFrame({'type':['logistic no load balance'],'hyperparameter':[xx],'log loss
                   'log loss Test':[log_loss(y_test, sig_clf.predict_proba(test_x_fea
aa=aa.append(bb)
```

```
for alpha = 1e-06
Log Loss : 1.44111742502
for alpha = 1e-05
Log Loss : 1.31140079307
for alpha = 0.0001
Log Loss : 1.15855303809
for alpha = 0.001
Log Loss : 1.17708086349
for alpha = 0.01
Log Loss : 1.30892341532
for alpha = 0.1
Log Loss : 1.39931914119
for alpha = 1
Log Loss : 1.42873699809
```

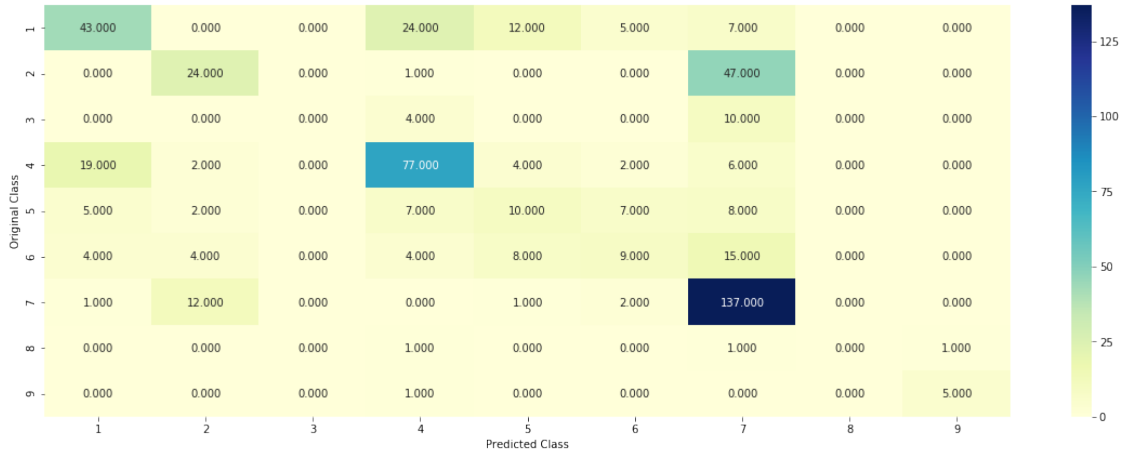



```
For values of best alpha =  0.0001 The train log loss is: 0.565136036128
For values of best alpha =  0.0001 The cross validation log loss is: 1.15855303809
For values of best alpha =  0.0001 The test log loss is: 1.14336186273
```

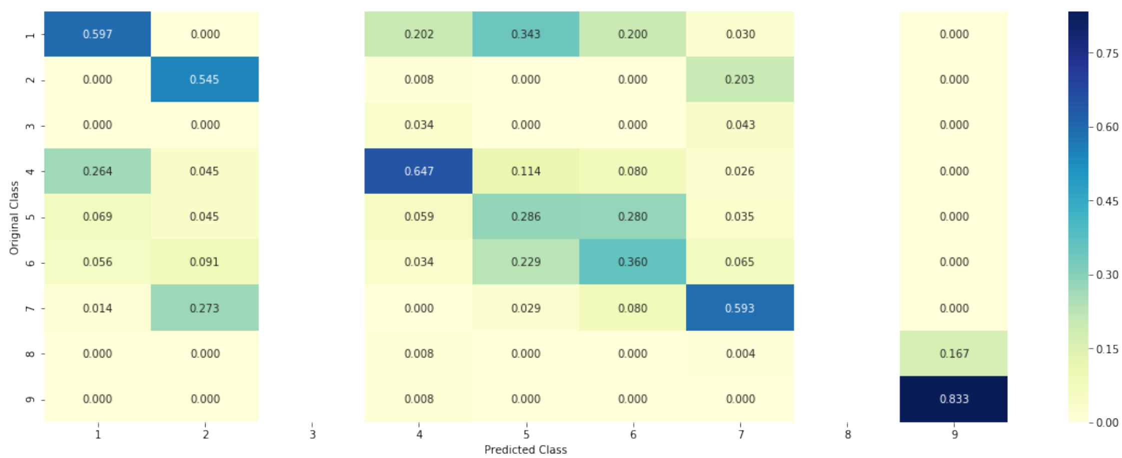
4.3.2.2. Testing model with best hyper parameters

In [161]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generate
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=T
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=o
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=
predict_and_plot_confusion_matrix(train_x_feature, train_y, cv_x_feature, cv_y, clf)
```

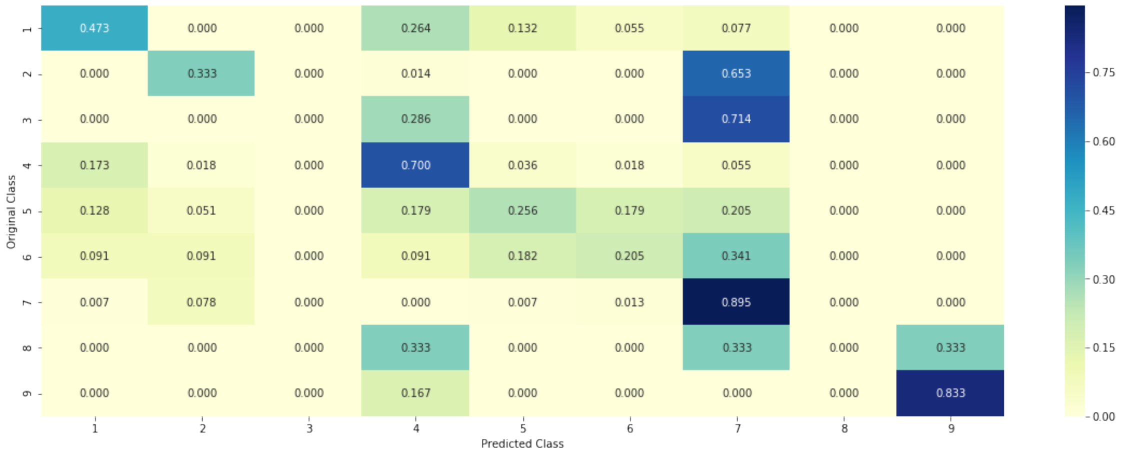
```
Log loss : 1.15855303809
Number of mis-classified points : 0.4266917293233083
-------------------- Confusion matrix --------------------
```




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




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



4.3.2.3. Feature Importance, Correctly Classified point

```
In [162]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=4
          clf.fit(train_x_feature,train_y)
          test_point_index = 1
          no_feature = 500
          predicted_cls = sig_clf.predict(test_x_feature[test_point_index])
          print("Predicted Class :", predicted_cls[0])
          print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_feature
          print("Actual Class :", test_y[test_point_index])
          indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
          print("-"*50)
          get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.0247  0.2556  0.0108  0.0548  0.0357  0.0148  0.5911  0.00
Actual Class : 2
--------------------------------------------------
201 Text feature [19ex] present in test data point [True]
302 Text feature [156027120] present in test data point [True]
372 Text feature [12] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

```
In [163]: test_point_index = 100
          no_feature = 500
          predicted_cls = sig_clf.predict(test_x_feature[test_point_index])
          print("Predicted Class :", predicted_cls[0])
          print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_feature
          print("Actual Class :", test_y[test_point_index])
          indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
```

```
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene
```

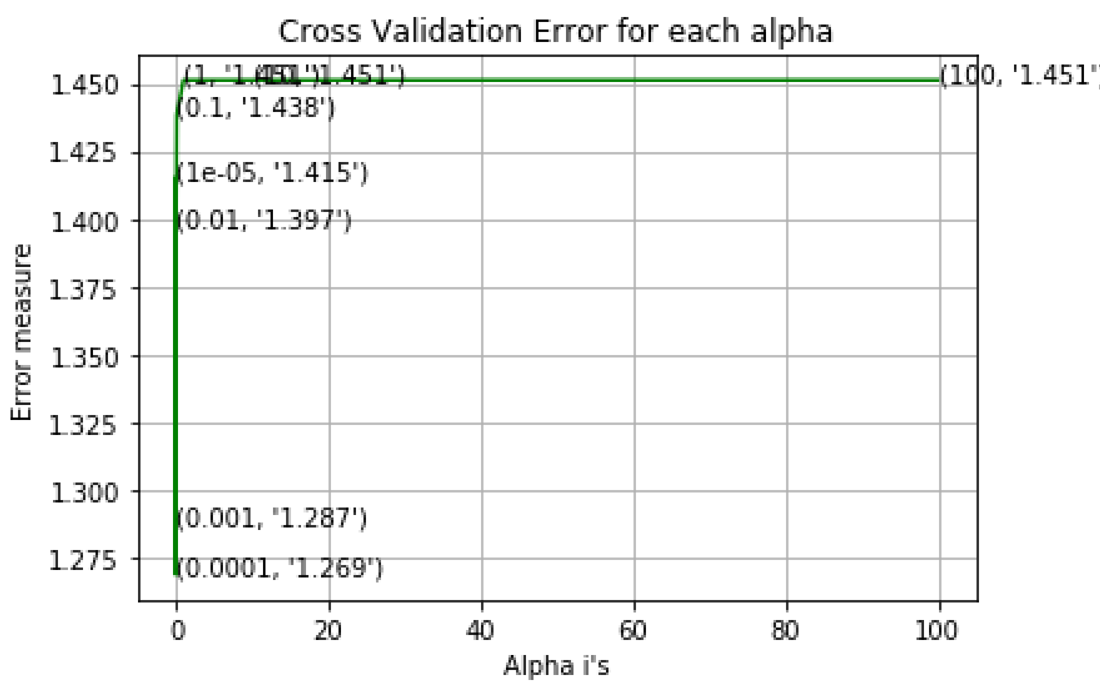
```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.0252  0.1056  0.0078  0.0908  0.0363  0.0568  0.6667  0.004
Actual Class : 7
--------------------------------------------------
164 Text feature [153] present in test data point [True]
198 Text feature [188] present in test data point [True]
235 Text feature [171] present in test data point [True]
282 Text feature [106] present in test data point [True]
337 Text feature [194] present in test data point [True]
372 Text feature [12] present in test data point [True]
415 Text feature [143] present in test data point [True]
447 Text feature [190] present in test data point [True]
450 Text feature [158] present in test data point [True]
499 Text feature [117] present in test data point [True]
Out of the top  500  features  10 are present in query point
```

4.4. Linear Support Vector Machines
4.4.1. Hyper paramter tuning

In [164]:
```
# read more about support vector machines with linear kernals here http://scikit-lea

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

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



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

```
# predict_proba(X)        Posterior probabilities of classification
#-------------------------------------
# video link:
#-------------------------------------

alpha = [10 ** x for x in range(-5, 3)]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
#     clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='hinge
    clf.fit(train_x_tfidf1000, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf1000, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf1000)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', l
clf.fit(train_x_tfidf1000, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf1000, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidf1000)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_tfidf1000)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log los
predict_y = sig_clf.predict_proba(test_x_tfidf1000)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_l

xx='C :'+str(alpha[best_alpha])
bb=pd.DataFrame({'type':['SVM linear'],'hyperparameter':[xx],'log loss CV':[log_loss
                    'log loss Test':[log_loss(y_test, sig_clf.predict_proba(test_x_tf
aa=aa.append(bb)
```

```
for C = 1e-05
Log Loss : 1.41511139614
for C = 0.0001
Log Loss : 1.26855117324
for C = 0.001
Log Loss : 1.28683839082
for C = 0.01
Log Loss : 1.39717196429
for C = 0.1
Log Loss : 1.43849148817
for C = 1
Log Loss : 1.45123222901
for C = 10
Log Loss : 1.45123222456
for C = 100
Log Loss : 1.45123220507
```

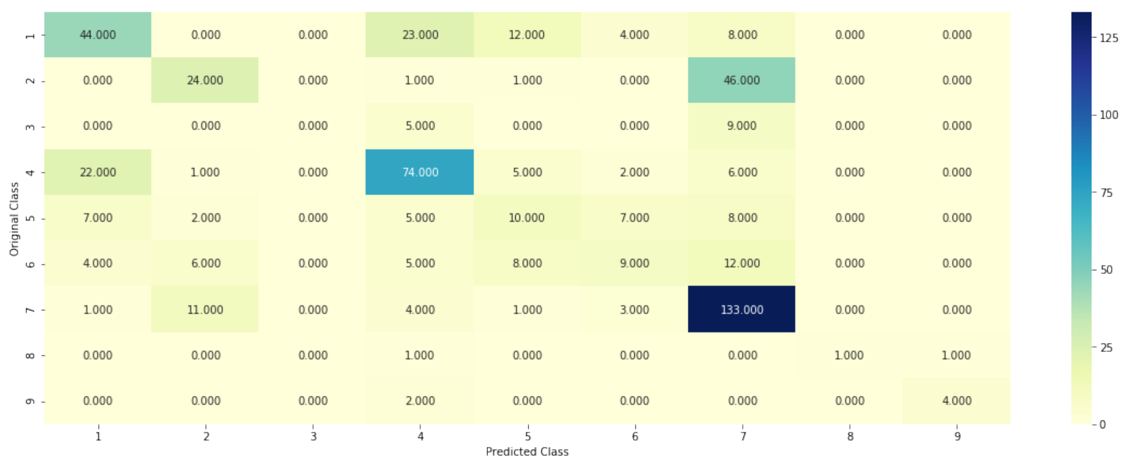



```
For values of best alpha =  0.0001 The train log loss is: 0.626700243572
For values of best alpha =  0.0001 The cross validation log loss is: 1.26855117324
For values of best alpha =  0.0001 The test log loss is: 1.26051059103
```

4.4.2. Testing model with best hyper parameters

```
In [165]: # read more about support vector machines with linear kernals here http://scikit-lea

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

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


          # clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='bala
          clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state
          predict_and_plot_confusion_matrix(train_x_tfidf1000, train_y,cv_x_tfidf1000,cv_y, cl
```

```
Log loss : 1.26855117324
Number of mis-classified points : 0.43796992481203006
-------------------- Confusion matrix --------------------
```




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


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

4.3.3. Feature Importance
4.3.3.1. For Correctly classified point

```
In [166]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state
          clf.fit(train_x_tfidf1000,train_y)
          test_point_index = 1
          # test_point_index = 100
          no_feature = 500
          predicted_cls = sig_clf.predict(test_x_tfidf1000[test_point_index])
          print("Predicted Class :", predicted_cls[0])
          print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf1
          print("Actual Class :", test_y[test_point_index])
          indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
```

```
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.061   0.2232  0.0204  0.1177  0.0612  0.0237  0.4792  0.006
Actual Class : 2
--------------------------------------------------
212 Text feature [19ex] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

4.3.3.2. For Incorrectly classified point

In [167]:
```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf1000[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf10
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.0627  0.1091  0.0214  0.0461  0.0629  0.143   0.5401  0.005
Actual Class : 7
--------------------------------------------------
191 Text feature [171] present in test data point [True]
245 Text feature [188] present in test data point [True]
248 Text feature [153] present in test data point [True]
279 Text feature [158] present in test data point [True]
294 Text feature [194] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

In [168]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_node
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=N
# class_weight=None)

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

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

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


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

alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, r
        clf.fit(train_x_tfidf1000, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_tfidf1000, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf1000)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_,
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

```
    '''

    best_alpha = np.argmin(cv_log_error_array)
    clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini'
    clf.fit(train_x_tfidf1000, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf1000, train_y)

    predict_y = sig_clf.predict_proba(train_x_tfidf1000)
    print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log los
    predict_y = sig_clf.predict_proba(cv_x_tfidf1000)
    print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross valida
    predict_y = sig_clf.predict_proba(test_x_tfidf1000)
    print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log los

    xx='n_estimator :'+str(alpha[int(best_alpha/2)])+'depth'+str(max_depth[int(best_alpha
    bb=pd.DataFrame({'type':['RF'],'hyperparameter':[xx],'log loss CV':[log_loss(y_cv, s
                       'log loss Test':[log_loss(y_test, sig_clf.predict_proba(test_x_tf
    aa=aa.append(bb)
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.34262640827
for n_estimators = 100 and max depth =  10
Log Loss : 1.30118562315
for n_estimators = 200 and max depth =  5
Log Loss : 1.32269298474
for n_estimators = 200 and max depth =  10
Log Loss : 1.30127235048
for n_estimators = 500 and max depth =  5
Log Loss : 1.32261626535
for n_estimators = 500 and max depth =  10
Log Loss : 1.30033213525
for n_estimators = 1000 and max depth =  5
Log Loss : 1.31630339687
for n_estimators = 1000 and max depth =  10
Log Loss : 1.29852154437
for n_estimators = 2000 and max depth =  5
Log Loss : 1.312879509
for n_estimators = 2000 and max depth =  10
Log Loss : 1.29761115069
For values of best estimator =  2000 The train log loss is: 1.01952432791
For values of best estimator =  2000 The cross validation log loss is: 1.29761115069
For values of best estimator =  2000 The test log loss is: 1.26251208978
```

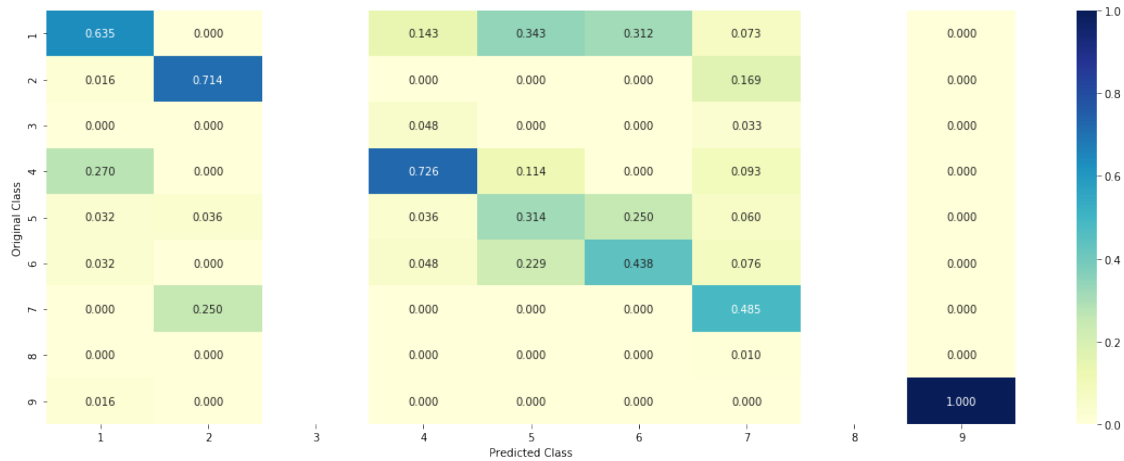
4.5.2. Testing model with best hyper parameters (TFIDF top 2000 words)

```
In [169]: # --------------------------------
          # default parameters
```

```
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_node
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=N
# class_weight=None)

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

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

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

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini'
predict_and_plot_confusion_matrix(train_x_tfidf1000, train_y,cv_x_tfidf1000,cv_y, cl
```

```
Log loss : 1.29761115069
Number of mis-classified points : 0.4548872180451128
-------------------- Confusion matrix --------------------
```


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




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

4.5.3. Feature Importance
4.5.3.1. Correctly Classified point

In [170]:
```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini'
clf.fit(train_x_tfidf1000, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf1000, train_y)

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidf1000[test_point_index])
print("Predicted Class :", predicted_cls[0])
```

```
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf1
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],te
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.0425  0.1932  0.0134  0.0539  0.0313  0.0286  0.6259  0.00
Actual Class : 2
--------------------------------------------------
99 Text feature [11] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [171]:
```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidf1000[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf1
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],te
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.1331  0.1546  0.0231  0.1957  0.0588  0.066   0.3534  0.00
Actuall Class : 7
--------------------------------------------------
99 Text feature [11] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

In [172]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_node
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=N
# class_weight=None)

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

# some of attributes of  RandomForestClassifier()
```

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

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


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

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

```python
best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini'
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

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

xx='n_estimator :'+str(alpha[int(best_alpha/4)])+'depth'+str(max_depth[int(best_alph
bb=pd.DataFrame({'type':['RF response coding'],'hyperparameter':[xx],'log loss CV':[
                  'log loss Test':[log_loss(y_test, sig_clf.predict_proba(test_x_re
aa=aa.append(bb)
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.1982673299
for n_estimators = 10 and max depth =  3
Log Loss : 1.78859808731
for n_estimators = 10 and max depth =  5
Log Loss : 1.72805740997
for n_estimators = 10 and max depth =  10
Log Loss : 1.80584027509
for n_estimators = 50 and max depth =  2
Log Loss : 1.7336730986
for n_estimators = 50 and max depth =  3
Log Loss : 1.53438654694
for n_estimators = 50 and max depth =  5
Log Loss : 1.44900419843
for n_estimators = 50 and max depth =  10
Log Loss : 1.68944921395
for n_estimators = 100 and max depth =  2
Log Loss : 1.59738894293
for n_estimators = 100 and max depth =  3
Log Loss : 1.52322173125
for n_estimators = 100 and max depth =  5
Log Loss : 1.34006003078
for n_estimators = 100 and max depth =  10
Log Loss : 1.6482305023
for n_estimators = 200 and max depth =  2
Log Loss : 1.61427326675
for n_estimators = 200 and max depth =  3
Log Loss : 1.53245696595
for n_estimators = 200 and max depth =  5
Log Loss : 1.38724956112
```

```
for n_estimators = 200 and max depth =  10
Log Loss : 1.67480593395
for n_estimators = 500 and max depth =  2
Log Loss : 1.6816121161
for n_estimators = 500 and max depth =  3
Log Loss : 1.55749156932
for n_estimators = 500 and max depth =  5
Log Loss : 1.41677638295
for n_estimators = 500 and max depth =  10
Log Loss : 1.72702155932
for n_estimators = 1000 and max depth =  2
Log Loss : 1.66510513847
for n_estimators = 1000 and max depth =  3
Log Loss : 1.57243950436
for n_estimators = 1000 and max depth =  5
Log Loss : 1.40388398679
for n_estimators = 1000 and max depth =  10
Log Loss : 1.72359874254
For values of best alpha =  100 The train log loss is: 0.0558923522127
For values of best alpha =  100 The cross validation log loss is: 1.34006003078
For values of best alpha =  100 The test log loss is: 1.34333370566
```

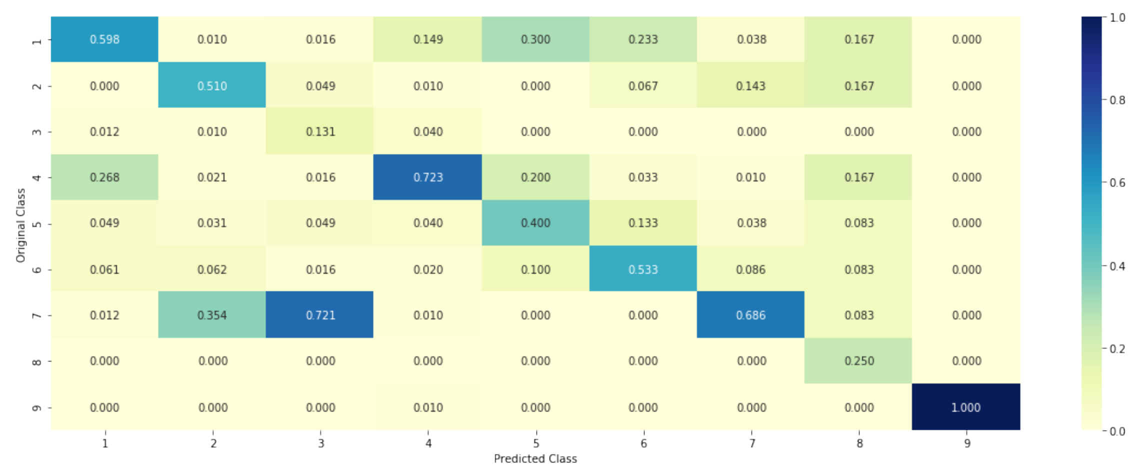
4.5.4. Testing model with best hyper parameters (Response Coding)

In [173]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_node
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=N
# class_weight=None)

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

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

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

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

```
Log loss : 1.34006003078
```

```
Number of mis-classified points : 0.45300751879699247
-------------------- Confusion matrix --------------------
```


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




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

4.5.5. Feature Importance
4.5.5.1. Correctly Classified point

```
In [174]: clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini'
          clf.fit(train_x_responseCoding, train_y)
          sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
          sig_clf.fit(train_x_responseCoding, train_y)


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

```
Predicted Class : 2
Predicted Class Probabilities: [[ 0.0125  0.3673  0.161   0.0156  0.0212  0.0739  0.2931  0.04
Actual Class : 2
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
```

```
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

In [175]:
```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_respons
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 2
Predicted Class Probabilities: [[ 0.0258  0.3959  0.1298  0.034   0.0359  0.0852  0.2142  0.06
Actual Class : 7
--------------------------------------------------
Variation is important feature
```

```
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models
4.7.1 testing with hyper parameter tuning

In [176]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generate
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=T
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=o
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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


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

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

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


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

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

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

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


clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced',
clf1.fit(train_x_tfidf1000, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', r
clf2.fit(train_x_tfidf1000, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


clf3 = MultinomialNB(alpha=0.001)
clf3.fit(train_x_tfidf1000, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")
```

```
sig_clf1.fit(train_x_tfidf1000, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_pro
sig_clf2.fit(train_x_tfidf1000, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_
sig_clf3.fit(train_x_tfidf1000, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_class
    sclf.fit(train_x_tfidf1000, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, lo
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_tfidf1000))
    if best_alpha > log_error:
        best_alpha = log_error
```

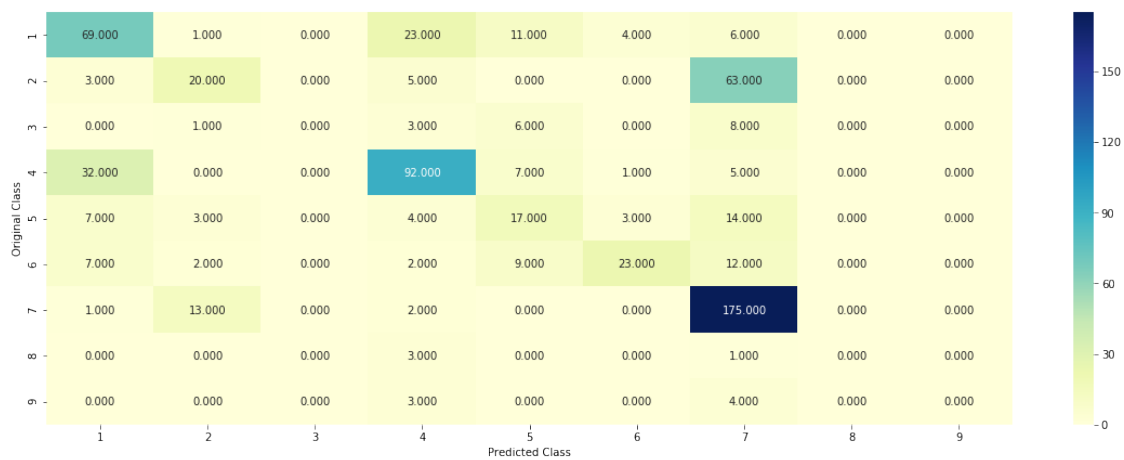
```
Logistic Regression :  Log Loss: 1.16
Support vector machines : Log Loss: 1.45
Naive Bayes : Log Loss: 1.31
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.186
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.104
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.777
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.316
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.200
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.370
```

4.7.2 testing the model with the best hyper parameters

In [177]:
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifie
sclf.fit(train_x_tfidf1000, train_y)

log_error = log_loss(train_y, sclf.predict_proba(train_x_tfidf1000))
print("Log loss (train) on the stacking classifier :",log_error)

log_error = log_loss(cv_y, sclf.predict_proba(cv_x_tfidf1000))
print("Log loss (CV) on the stacking classifier :",log_error)

log_error = log_loss(test_y, sclf.predict_proba(test_x_tfidf1000))
print("Log loss (test) on the stacking classifier :",log_error)

print("Number of missclassified point :", np.count_nonzero((sclf.predict(test_x_tfid
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_tfidf1000))
```

```
#xx='n_estimator :'+alpha[int(best_alpha/4)]+'depth'+max_depth[int(best_alpha%4)]

bb=pd.DataFrame({'type':['stack'],'hyperparameter':['na'],'log loss CV':[log_loss(cv_
                   'log loss Test':[log_loss(test_y, sclf.predict_proba(test_x_tfidf
aa=aa.append(bb)
```

```
Log loss (train) on the stacking classifier : 0.974471042293
Log loss (CV) on the stacking classifier : 1.31552588327
Log loss (test) on the stacking classifier : 1.30063852879
Number of missclassified point : 0.4045112781954887
-------------------- Confusion matrix --------------------
```



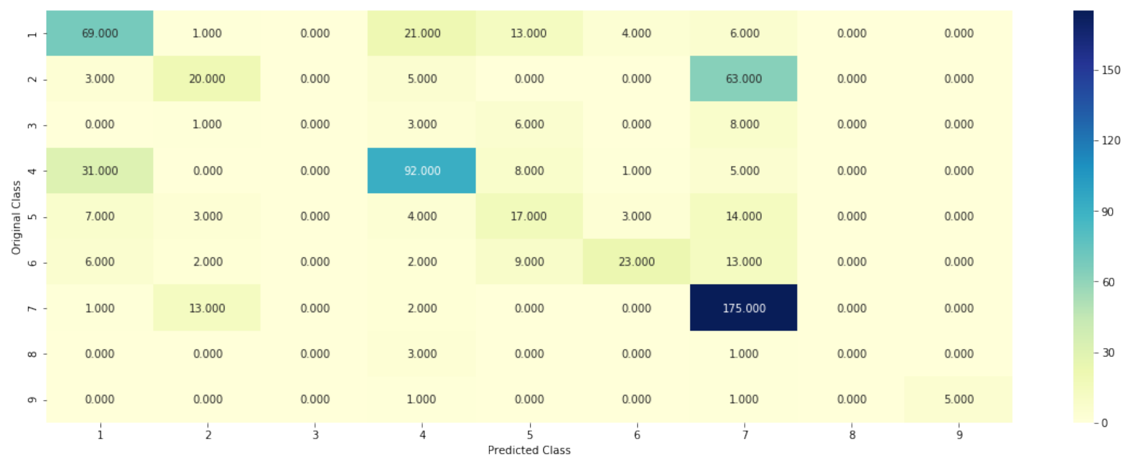

```
-------------------- Precision matrix (Columm Sum=1) --------------------
```

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

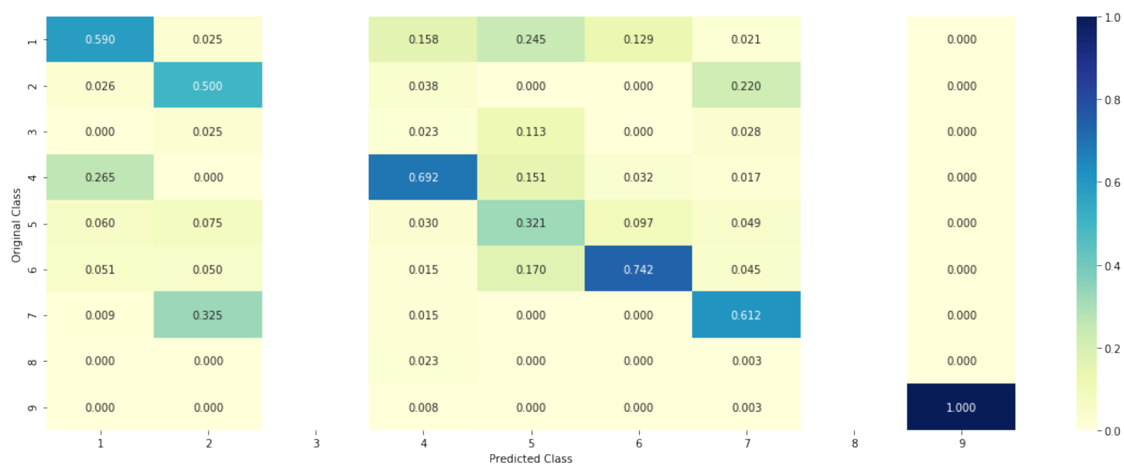
4.7.3 Maximum Voting classifier

In [178]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClass
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_c
vclf.fit(train_x_tfidf1000, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_pr
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_prob
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_tfidf
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_tfidf1000))

#xx='n_estimator :'+alpha[int(best_alpha/4)]+'depth'+max_depth[int(best_alpha%4)]

bb=pd.DataFrame({'type':['max voting'],'hyperparameter':['na'],'log loss CV':[log_los
                    'log loss Test':[log_loss(test_y, vclf.predict_proba(test_x_tfidf
aa=aa.append(bb)
```

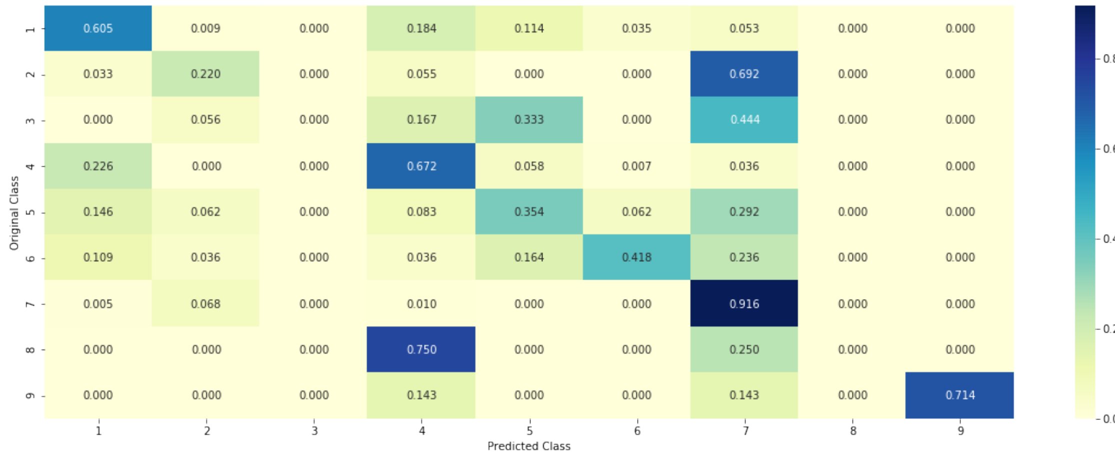
```
Log loss (train) on the VotingClassifier : 0.821187105449
Log loss (CV) on the VotingClassifier : 1.24438384412
Log loss (test) on the VotingClassifier : 1.23660488773
Number of missclassified point : 0.3969924812030075
-------------------- Confusion matrix --------------------
```



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



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



5. End

In [179]:
```
#All model logloss and hyperparemeter
aa
```

Out[179]:
```
               hyperparameter  log loss CV  log loss Test  \
0                          NA     2.490032       2.467486
0                    alpha :1     1.204554       1.198064
0                        k :5     1.048184       1.063273
0                 C :0.0001     1.157056       1.141029
0                 C :0.0001     1.158553       1.143362
0                 C :0.0001     1.268551       1.260511
0  n_estimator :2000depth10     1.297611       1.262512
0    n_estimator :100depth5     1.340060       1.343334
0                          na     1.315526       1.300639
0                          na     1.244384       1.236605

                         type
0                Random model
0                 naive bayes
0                         knn
0                    logistic
0  logistic no load balance
0                  SVM linear
0                          RF
0          RF response coding
0                       stack
0                  max voting
```