TFIDF

July 19, 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 bigram
<li> Try any of the feature engineering techniques discussed in the course to reduce the CV and
```

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 bigram
<li> Try any of the feature engineering techniques discussed in the course to reduce the CV and
```

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

```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
```

```
C:\Users\suman\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning:
  "This module will be removed in 0.20.", DeprecationWarning)
```

3.1. Reading Data
3.1.1. Reading Gene and Variation Data

In [2]:
```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[2]:

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

```
training_variants is a comma separated file containing the description of the genetic mutation
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 [3]: # note the seprator in this file
        data_text =pd.read_csv("training_text",sep="\|\|",engine="python",names=["ID","TEXT"],s
        print('Number of data points : ', data_text.shape[0])
        print('Number of features : ', data_text.shape[1])
        print('Features : ', data_text.columns.values)
        data_text.head()
```

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

Out[3]:

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

3.1.3. Preprocessing of text

```
In [4]: # loading stop words from nltk library
        stop_words = set(stopwords.words('english'))
        #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 [5]: #text processing stage.
        start_time = time.clock()
```

```
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")
```

```
Time took for preprocessing the text : 239.53121542838545 seconds
```

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

Out[6]:
```
   ID    Gene             Variation  Class  \
0   0  FAM58A  Truncating Mutations      1
1   1     CBL                 W802*      2
2   2     CBL                 Q249E      2
3   3     CBL                 N454D      3
4   4     CBL                 L399V      4

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

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 [7]:
```
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

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

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

In [8]:
```
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
```

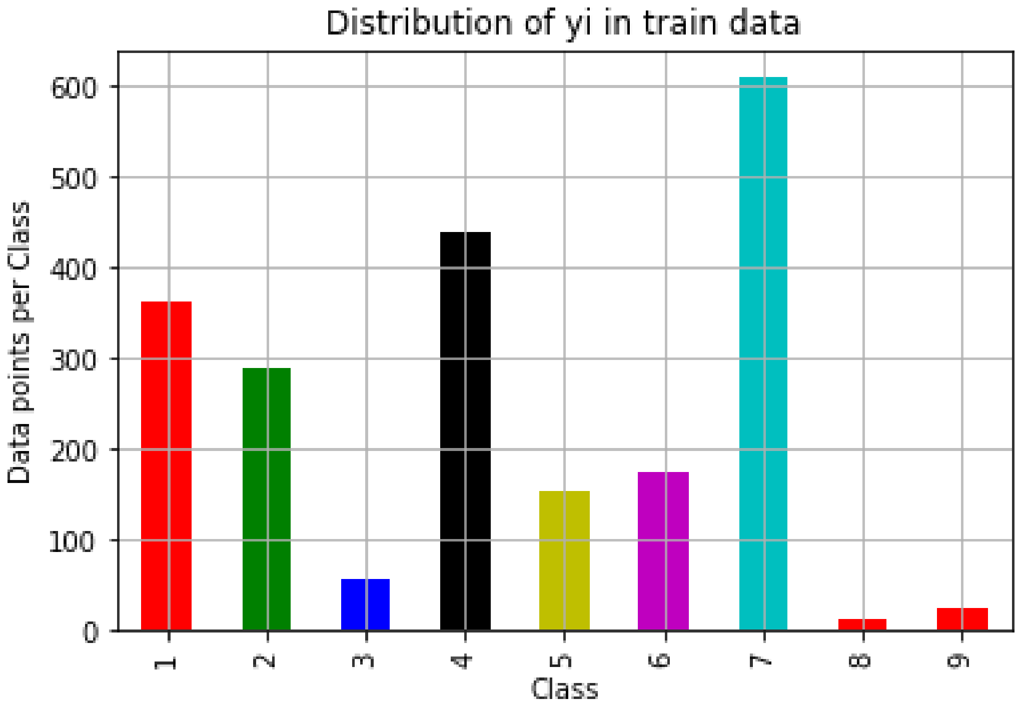
```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

Out[8]: (2124,)

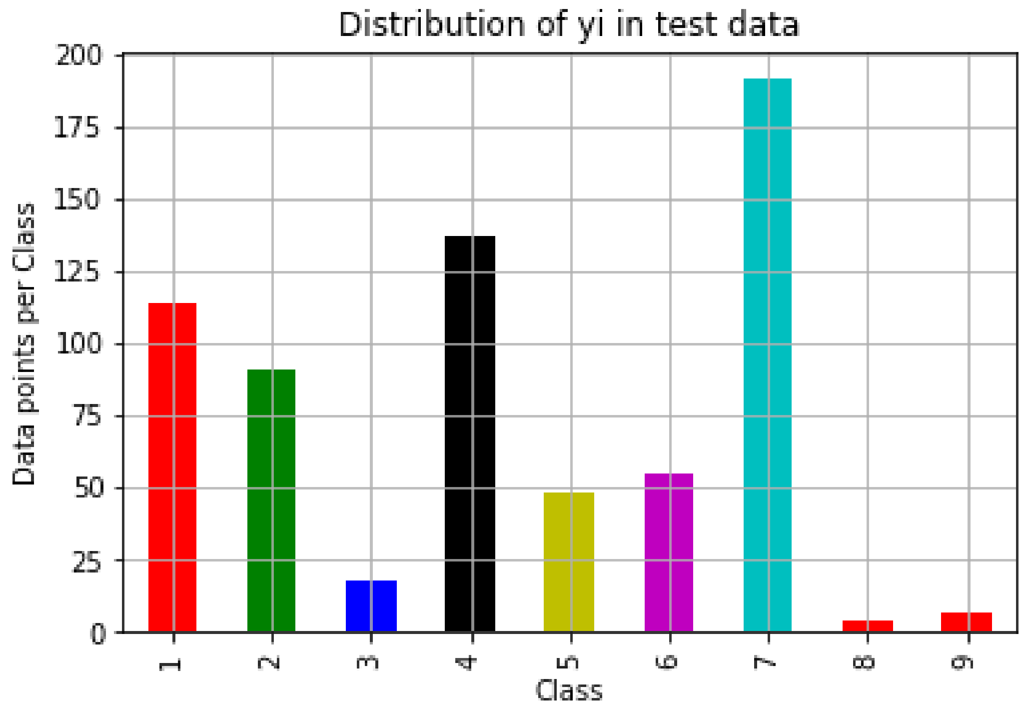
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

```
In [9]: # it returns a dict, keys as class labels and values as the number of data points in t
        train_class_distribution = train_df['Class'].value_counts().sortlevel()
        test_class_distribution = test_df['Class'].value_counts().sortlevel()
        cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

        my_colors = ['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.html
        # -(train_class_distribution.values): the minus sign will give us in decreasing order
        sorted_yi = np.argsort(-train_class_distribution.values)
        for i in sorted_yi:
            print('Number of data points in class', i+1, ':',train_class_distribution.values[i]


        print('-'*80)
        my_colors = ['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.html
        # -(train_class_distribution.values): the minus sign will give us in decreasing order
        sorted_yi = np.argsort(-test_class_distribution.values)
        for i in sorted_yi:
            print('Number of data points in class', i+1, ':',test_class_distribution.values[i]

        print('-'*80)
        my_colors = ['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()
```

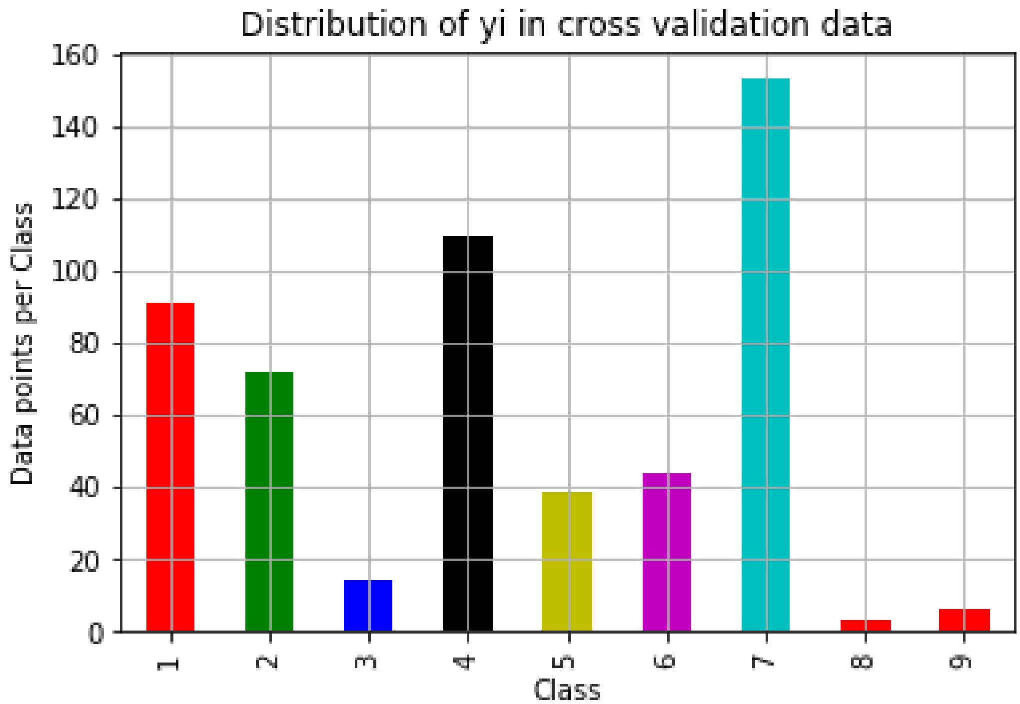
```
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1,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




```
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.

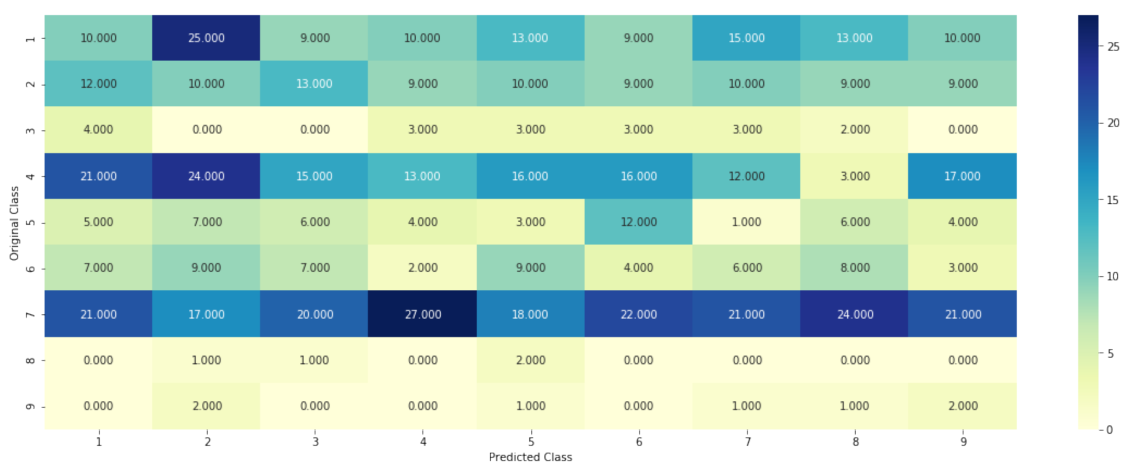
```
In [10]: # This function plots the confusion matrices given y_i, y_i_hat.
         def plot_confusion_matrix(test_y, predict_y):
             C = confusion_matrix(test_y, predict_y)
             # C = 9,9 matrix, each cell (i,j) represents number of points of class i are pred

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

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

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

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

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

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

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

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

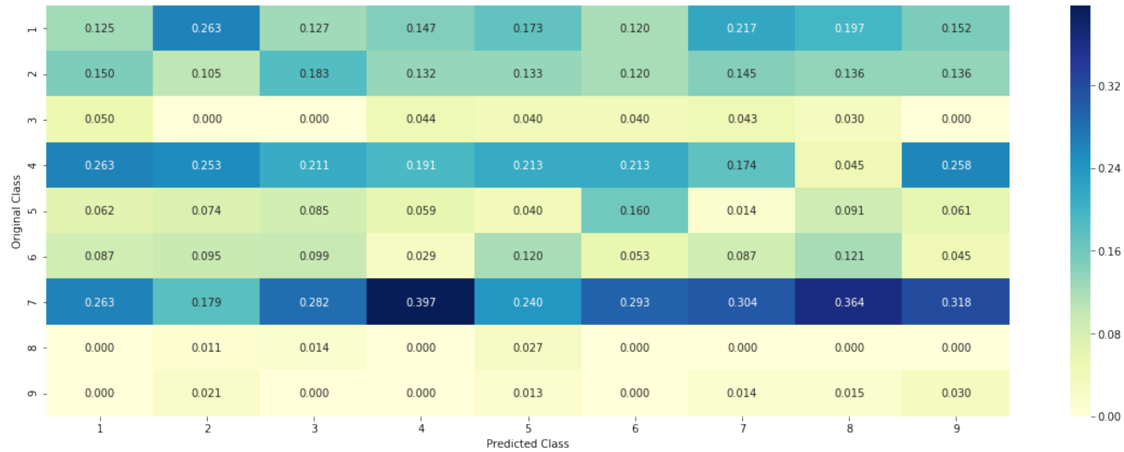
In [11]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
```

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

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


# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y, eps

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

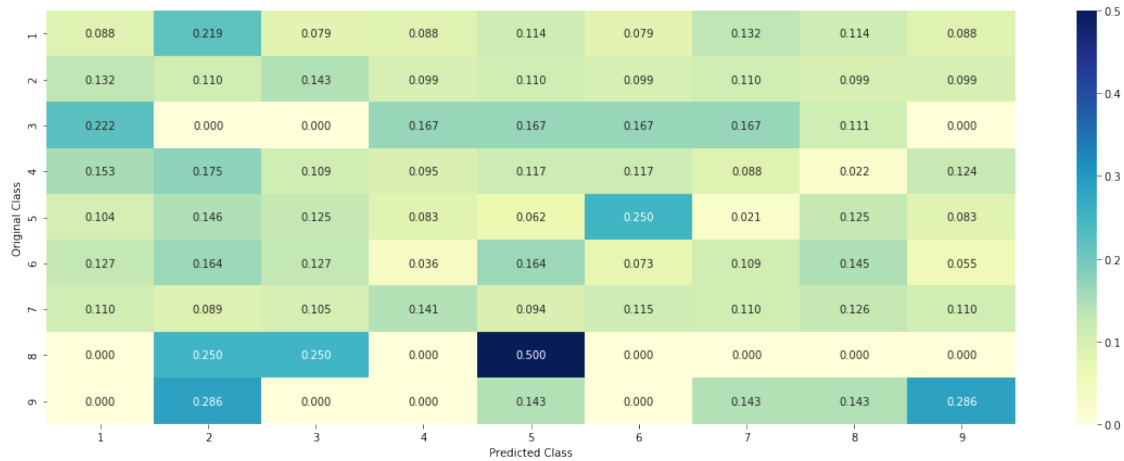
```
Log loss on Cross Validation Data using Random Model 2.44342284749
Log loss on Test Data using Random Model 2.5416295459
-------------------- Confusion matrix --------------------
```




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




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



3.3 Univariate Analysis

```
In [12]: aa=pd.DataFrame({'type':['Random model'],'hyperparameter':['NA'],'log loss CV':[log_lo
                           '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 da
         # build a vector (1*9) , the first element = (number of times it occured in class1 + 1
```

```
# gv_dict is like a look up table, for every gene it store a (1*9) representation of
# 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 ge
    gv_dict = dict()

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

```
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=='BRCA1')]
            #          ID   Gene             Variation  Class
            # 2470  2470  BRCA1                S1715C      1
            # 2486  2486  BRCA1                S1841R      1
            # 2614  2614  BRCA1                   M1R      1
            # 2432  2432  BRCA1                L1657P      1
            # 2567  2567  BRCA1                T1685A      1
            # 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 partic
            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, 0.068181818181
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0
    #      '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 va
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there i
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_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?

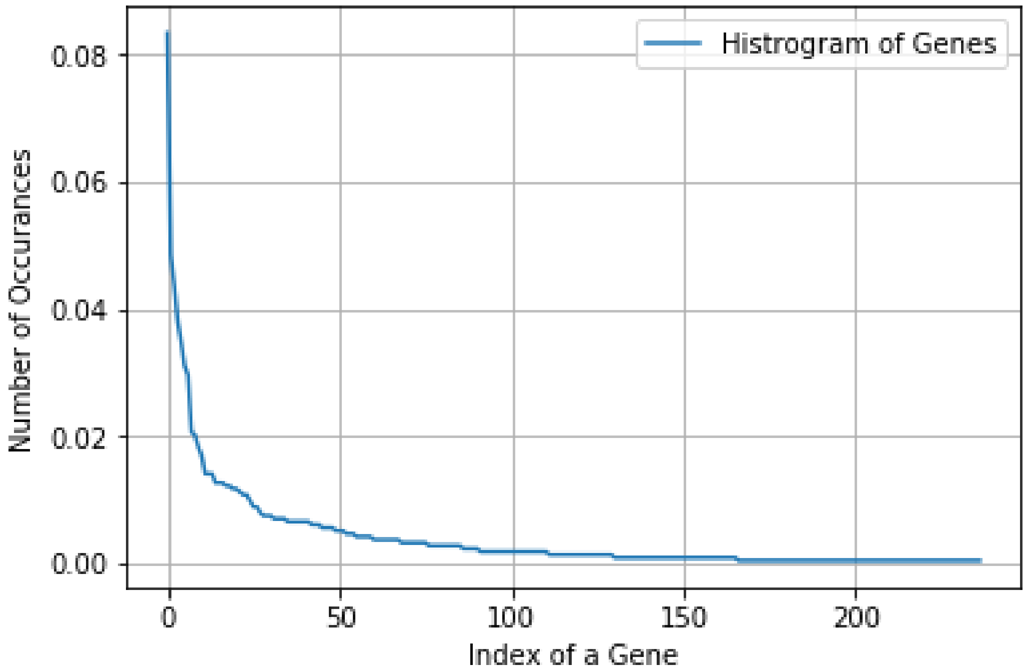
```
In [13]: unique_genes = train_df['Gene'].value_counts()
         print('Number of Unique Genes :', unique_genes.shape[0])
         # the top 10 genes that occured most
         print(unique_genes.head(10))
```

```
Number of Unique Genes : 237
BRCA1     177
TP53      103
EGFR       93
PTEN       81
BRCA2      74
BRAF       66
KIT        63
ERBB2      44
ALK        43
PDGFRA     39
Name: Gene, dtype: int64
```

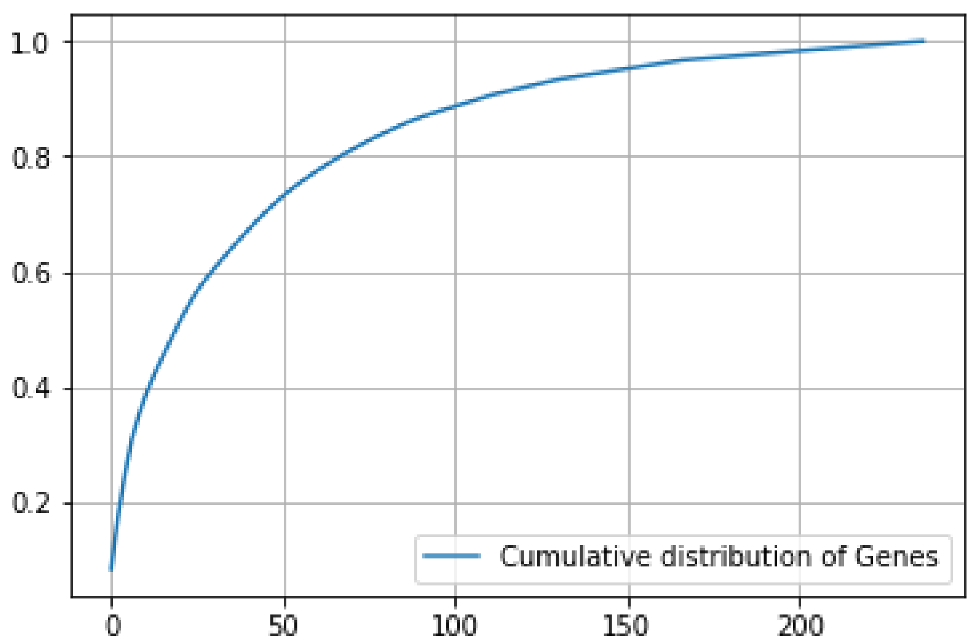
```
In [14]: print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the 
```

```
Ans: There are 237 different categories of genes in the train data, and they are distibuted as
```

```
In [15]: 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 [16]:
```
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 [17]: #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 [18]: 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
```

```
In [19]: # 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 [20]: train_df['Gene'].head()
```

```
Out[20]: 1041     TSC2
         752     ERBB2
         1447     SPOP
         2657    BRCA1
         1333     MLH1
         Name: Gene, dtype: object
```

```
In [21]: gene_vectorizer.get_feature_names()[0:5]
```

```
Out[21]: ['abl1', 'acvr1', 'ago2', 'akt1', 'akt2']
```

```
In [22]: 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
```

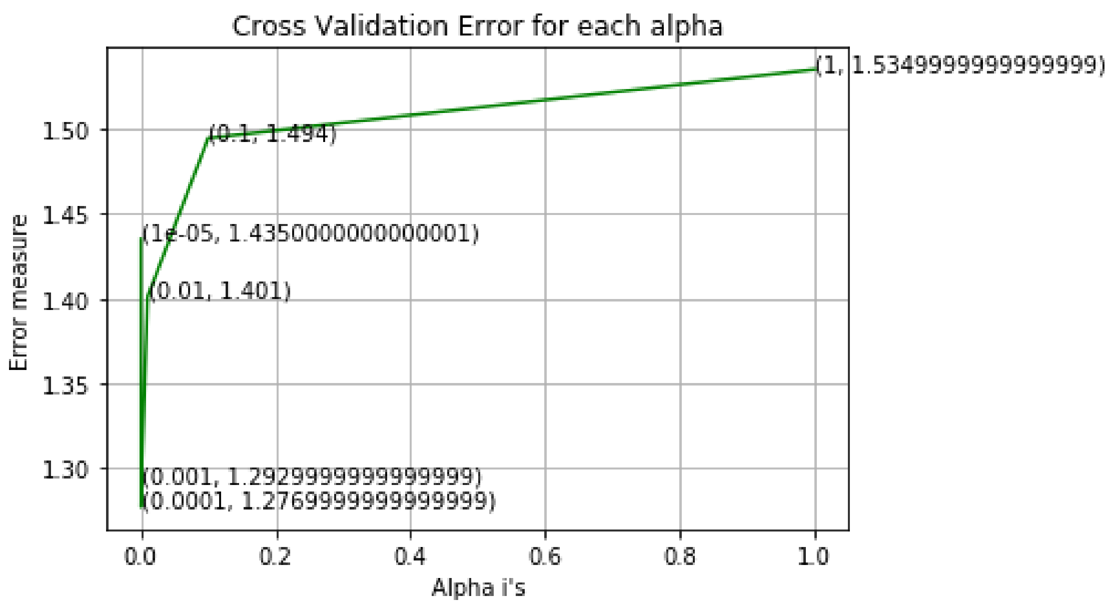
Q4. How good is this gene feature in predicting y_i?

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 [23]:
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
# 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 Gr
# predict(X)        Predict class labels for samples in X.

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


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_gene_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, la

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


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42
clf.fit(train_gene_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```
        sig_clf.fit(train_gene_feature_onehotCoding, y_train)

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

```
For values of alpha =  1e-05 The log loss is: 1.43496390031
For values of alpha =  0.0001 The log loss is: 1.27730325535
For values of alpha =  0.001 The log loss is: 1.29253800146
For values of alpha =  0.01 The log loss is: 1.40097114873
For values of alpha =  0.1 The log loss is: 1.49442604829
For values of alpha =  1 The log loss is: 1.53485198572
```




```
For values of best alpha =  0.0001 The train log loss is: 1.04105673336
For values of best alpha =  0.0001 The cross validation log loss is: 1.27730325535
For values of best alpha =  0.0001 The test log loss is: 1.2014005268
```

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

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

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

print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_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  237  genes in train datase
Ans
1. In test data 649 out of 665 : 97.59398496240601
2. In cross validation data 517 out of  532 : 97.18045112781954
```

3.2.2 Univariate Analysis on Variation Feature
Q7. Variation, What type of feature is it ?
Ans. Variation is a categorical variable
Q8. How many categories are there?

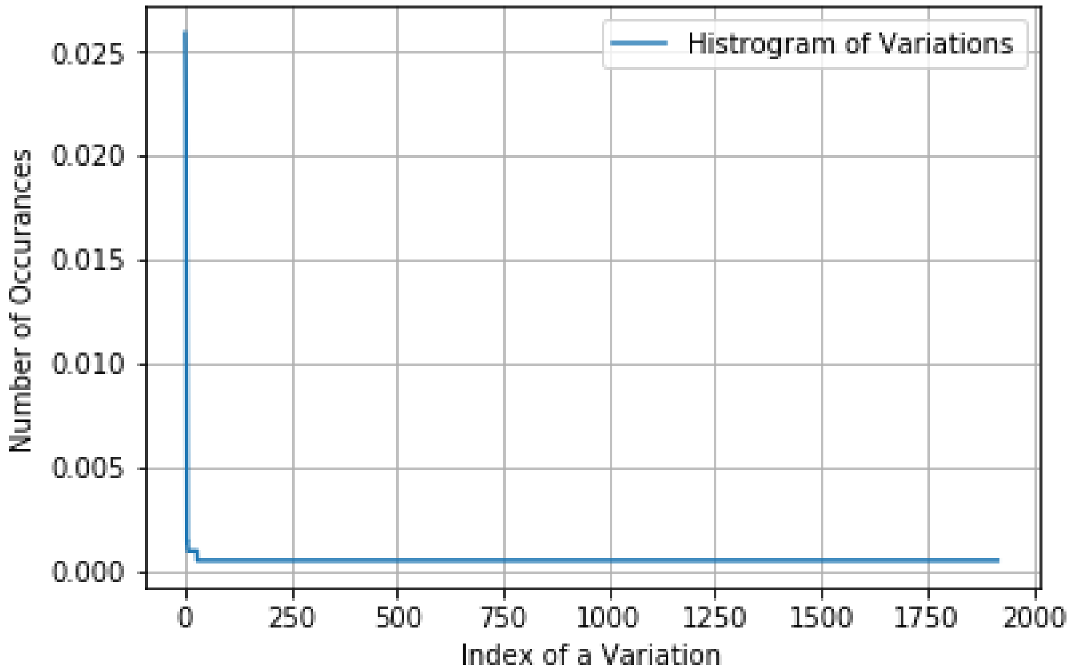
```
In [25]: unique_variations = train_df['Variation'].value_counts()
         print('Number of Unique Variations :', unique_variations.shape[0])
         # the top 10 variations that occured most
         print(unique_variations.head(10))
```

```
Number of Unique Variations : 1917
Deletion                55
Truncating_Mutations    50
Amplification           47
Fusions                 28
Overexpression           6
G12V                     3
E17K                     3
R841K                    2
A146V                    2
G13D                     2
Name: Variation, dtype: int64
```

```
In [26]: print("Ans: There are", unique_variations.shape[0] ,"different categories of variatio
```

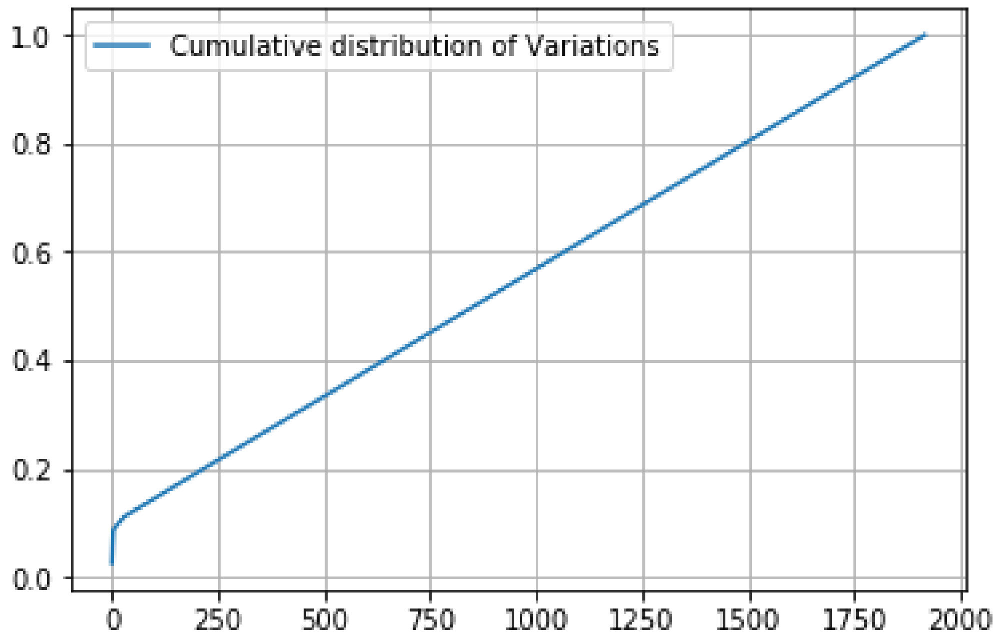
```
Ans: There are 1917 different categories of variations in the train data, and they are distibu
```

```
In [27]: 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 [28]: c = np.cumsum(h)
         print(c)
         plt.plot(c,label='Cumulative distribution of Variations')
         plt.grid()
         plt.legend()
         plt.show()
```

```
[ 0.02589454  0.04943503  0.07156309 ...,  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 [29]: # alpha is used for laplace smoothing
         alpha = 1
         # train gene feature
         train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", t
         # test gene feature
         test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", te
         # cross validation gene feature
         cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_c
```

```
In [30]: print("train_variation_feature_responseCoding is a converted feature using the respons

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

```
In [31]: # one-hot encoding of variation feature.
         variation_vectorizer = CountVectorizer()
         train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Va
         test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variatio
         cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation']]
```

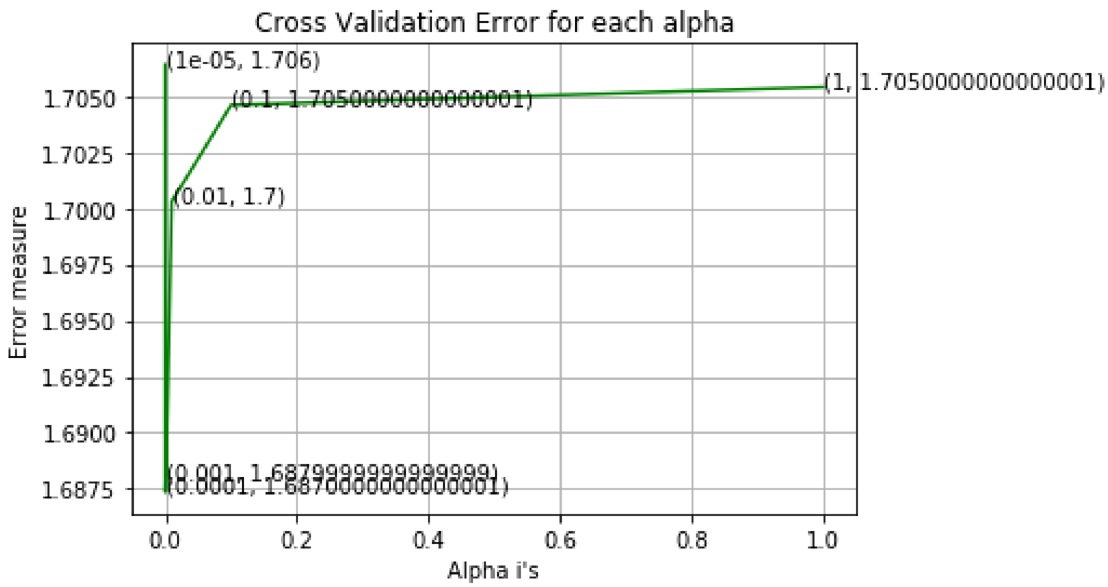
```
In [32]: print("train_variation_feature_onehotEncoded is converted feature using the onne-hot e

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

         # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
         # ------------------------------
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
         # 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 Gra
         # predict(X)        Predict class labels for samples in X.

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


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

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

             cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15
             print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, la

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

```python
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42
clf.fit(train_variation_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_variation_feature_onehotCoding, y_train)

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

```
For values of alpha =  1e-05 The log loss is: 1.70640152602
For values of alpha =  0.0001 The log loss is: 1.68735596421
For values of alpha =  0.001 The log loss is: 1.68800070903
For values of alpha =  0.01 The log loss is: 1.70030155105
For values of alpha =  0.1 The log loss is: 1.70460586526
For values of alpha =  1 The log loss is: 1.70540928944
```




```
For values of best alpha =  0.0001 The train log loss is: 0.711648604856
For values of best alpha =  0.0001 The cross validation log loss is: 1.68735596421
For values of best alpha =  0.0001 The test log loss is: 1.74442057301
```

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 [34]: print("Q12. How many data points are covered by total ", unique_variations.shape[0], "
         test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].sha
         cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
         print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_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  1917  genes in test and cross validation data s
Ans
1. In test data 58 out of 665 : 8.721804511278195
2. In cross validation data 54 out of  532 : 10.150375939849624
```

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 [35]: # 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 [36]: import math
         #https://stackoverflow.com/a/1602964
         def get_text_responsecoding(df):
             text_feature_responseCoding = np.zeros((df.shape[0],9))
             for i in range(0,9):
                 row_index = 0
                 for index, row in df.iterrows():
                     sum_prob = 0
                     for word in row['TEXT'].split():
                         sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_dict.get(w
                     text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEX
                     row_index += 1
             return text_feature_responseCoding
```

```
In [37]: # building a CountVectorizer with all the words that occured minimum 3 times in train
         text_vectorizer_onehotCoding = CountVectorizer(min_df=3)
```

```
train_text_feature_onehotCoding = text_vectorizer_onehotCoding.fit_transform(train_df

#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*nu
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 i
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 train
text_vectorizer_ngram = CountVectorizer(min_df=3,ngram_range=(1,2))
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*nu
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 i
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 train
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*nu
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 i
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 train
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[:1000]]
#add the other feature in stopwords
bottom_features=[features[i] for i in indices[1000:]]
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['TEXT

# 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*nu
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 i
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_f
```

```
Total number of unique words in train data BOW: shape 53424 (2124, 53424) (2124,)
Total number of unique words in train data ngram: shape 782559 (2124, 782559)
Total number of unique words in train data tfidf: shape 53424 (2124, 53424)
['canavanine', 'homlogy', 'statue', 'stats9', '5109', '51004', 'statistician', 'phush', '1790d
Total number of unique words in train data tfidf1000: shape 1000 (2124, 1000)
```

In [38]:
```
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 [39]:
```
#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 [40]:
```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.T/train_text_fe
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_featu
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_res
```

In [41]:
```
# 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['TEXT
# don't forget to normalize every feature
test_text_feature_onehotCoding = normalize(test_text_feature_onehotCoding, axis=0)

# we use the same vectorizer that was trained on train data
cv_text_feature_onehotCoding = text_vectorizer_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 [42]:
```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict_1 = dict(sorted(text_fea_dict_1.items(), key=lambda x: x[1] , re
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] , re
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] , re
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] , re
sorted_text_occur_4 = np.array(list(sorted_text_fea_dict_4.values()))
```

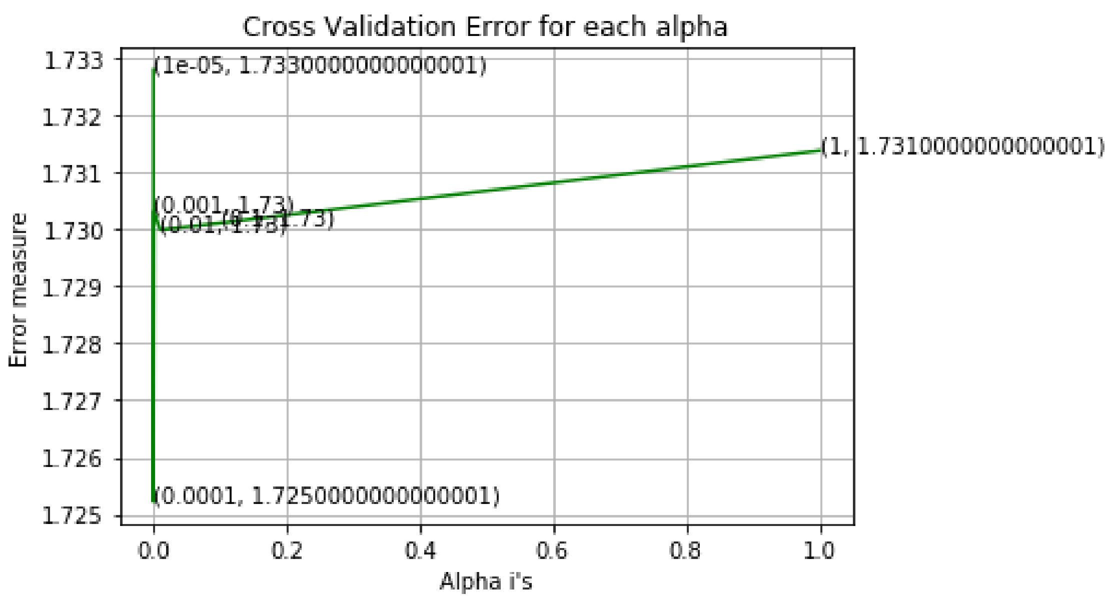
In [43]:
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur_1[0:10]))
print(Counter(sorted_text_occur_2[0:10]))
print(Counter(sorted_text_occur_3[0:10]))
print(Counter(sorted_text_occur_4[0:10]))
print(len(train_df['TEXT']))
```

```
Counter({17: 1, 3: 1, 4: 1, 7: 1, 8: 1, 20: 1, 411: 1, 12: 1, 29: 1, 31: 1})
Counter({3: 3, 4: 2, 9: 2, 17: 1, 5: 1, 14: 1})
Counter({0.08206138200566386: 1, 0.024264537047104122: 1, 0.071951868269086575: 1, 0.51968534
Counter({0.59925566690654186: 1, 1.1729020209318675: 1, 0.49651707481871821: 1, 1.496988974743
2124
```

In [44]:
```
# Train a Logistic regression+Calibration model using text features whicha re tfidf e
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
# 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 Gr
# 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-15
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, la

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

```python
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_l
predict_y = sig_clf.predict_proba(cv_text_feature_tfidf1000)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
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_lo
```

```
For values of alpha =  1e-05 The log loss is: 1.73279715355
For values of alpha =  0.0001 The log loss is: 1.72521903097
For values of alpha =  0.001 The log loss is: 1.73031758235
For values of alpha =  0.01 The log loss is: 1.72998876353
For values of alpha =  0.1 The log loss is: 1.73010627635
For values of alpha =  1 The log loss is: 1.73137522498
```




```
For values of best alpha =  0.0001 The train log loss is: 1.50909009785
For values of best alpha =  0.0001 The cross validation log loss is: 1.72521903097
For values of best alpha =  0.0001 The test log loss is: 1.72519932409
```

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

```
In [45]: 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=(1,2))
             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 [46]: 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
```

```
97.178 % of word of test data appeared in train data for bow
97.924 % of word of Cross Validation appeared in train data for bow
93.956 % of word of test data appeared in train data for ngram
97.924 % of word of Cross Validation appeared in train data for ngram
97.178 % of word of test data appeared in train data for tfidf
97.924 % of word of Cross Validation appeared in train data for tfidf
```

4. Machine Learning Models

```
In [47]: #Data preparation for ML models.

         #Misc. functionns for ML models
```

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

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

In [48]:
```
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 [49]:
```
# 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(wo
        elif (v < fea1_len+fea2_len):
            word = var_vec.get_feature_names()[v-(fea1_len)]
            yes_no = True if word == var else False
            if yes_no:
```

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

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

Stacking the three types of features

```
In [50]: # merging gene, variance and text features

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

         train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_
         test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_fea
         cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_o

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

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

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

         #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)).tocsr
         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_tfidf1000)
```

```
test_x_tfidf1000 = hstack((test_gene_var_onehotCoding, test_text_feature_tfidf1000)).t
cv_x_tfidf1000 = hstack((cv_gene_var_onehotCoding, cv_text_feature_tfidf1000)).tocsr()


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

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

In [51]:
```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehot(
print("(number of data points * number of features) in test data = ", test_x_onehotCod
print("(number of data points * number of features) in cross validation data =", cv_x_

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

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

print("tfidf to 1000 words features :")
print("(number of data points * number of features) in train data = ", train_x_tfidf1(
print("(number of data points * number of features) in test data = ", test_x_tfidf100(
print("(number of data points * number of features) in cross validation data =", cv_x_

print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_respons
print("(number of data points * number of features) in test data = ", test_x_response(
print("(number of data points * number of features) in cross validation data =", cv_x_
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 55610)
(number of data points * number of features) in test data =  (665, 55610)
(number of data points * number of features) in cross validation data = (532, 55610)
ngram features :
(number of data points * number of features) in train data =  (2124, 784745)
(number of data points * number of features) in test data =  (665, 784745)
(number of data points * number of features) in cross validation data = (532, 784745)
tfidf features :
(number of data points * number of features) in train data =  (2124, 55610)
(number of data points * number of features) in test data =  (665, 55610)
```

```
(number of data points * number of features) in cross validation data = (532, 55610)
tfidf to 1000 words features :
(number of data points * number of features) in train data =  (2124, 3186)
(number of data points * number of features) in test data =  (665, 3186)
(number of data points * number of features) in cross validation data = (532, 3186)
 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 [52]: #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_featur
         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_x
```

```
(2124, 2186)
(2124, 2186)
[ 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, 3186)
(number of data points * number of features) in test data =  (665, 3186)
(number of data points * number of features) in cross validation data = (532, 3186)
```

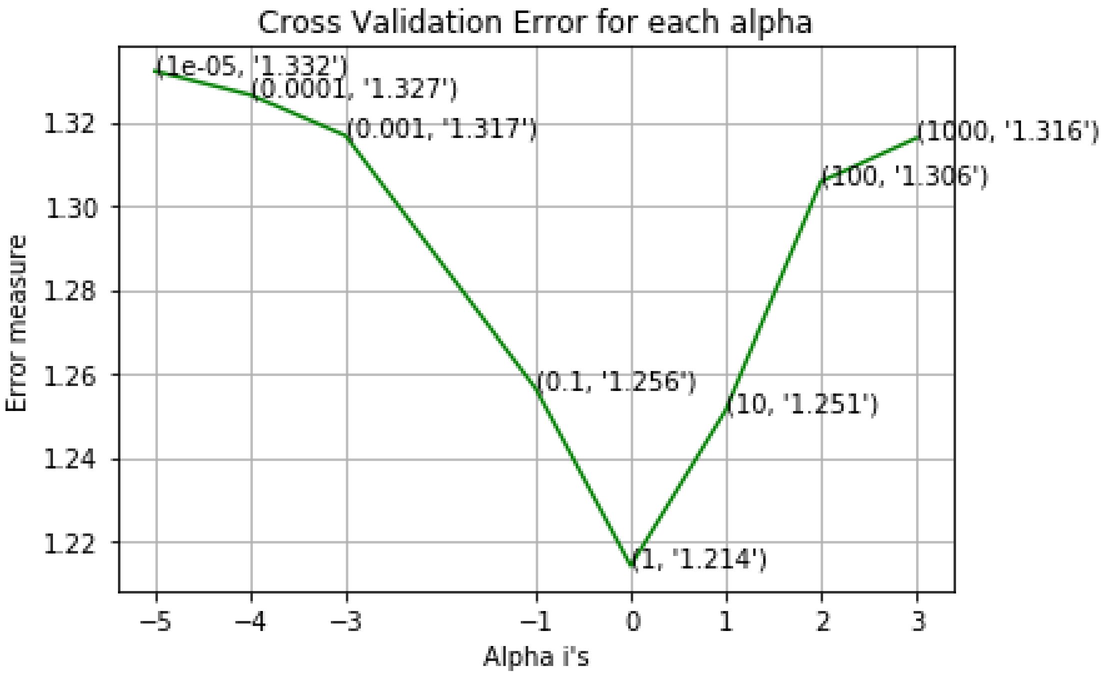
4.1. Base Line Model
4.1.1. Naive Bayes
4.1.1.1. Hyper parameter tuning

```
In [53]: # find more about Multinomial Naive base function here http://scikit-learn.org/stable
         # -------------------------
```

```
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


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

#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(f

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

```
for alpha = 1e-05
Log Loss : 1.33215336771
for alpha = 0.0001
Log Loss : 1.32652913169
for alpha = 0.001
Log Loss : 1.31676173746
for alpha = 0.1
Log Loss : 1.25645181835
for alpha = 1
Log Loss : 1.21412090082
for alpha = 10
Log Loss : 1.25135688416
for alpha = 100
Log Loss : 1.30580472951
for alpha = 1000
Log Loss : 1.31617424418
```



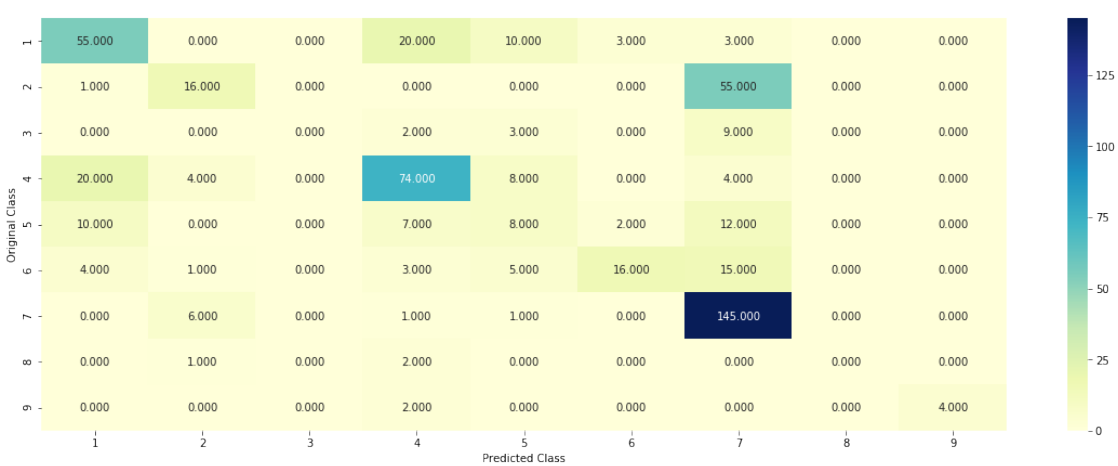

```
For values of best alpha =  1 The train log loss is: 0.727784694113
For values of best alpha =  1 The cross validation log loss is: 1.21412090082
For values of best alpha =  1 The test log loss is: 1.21258922438
```

4.1.1.2. Testing the model with best hyper paramters

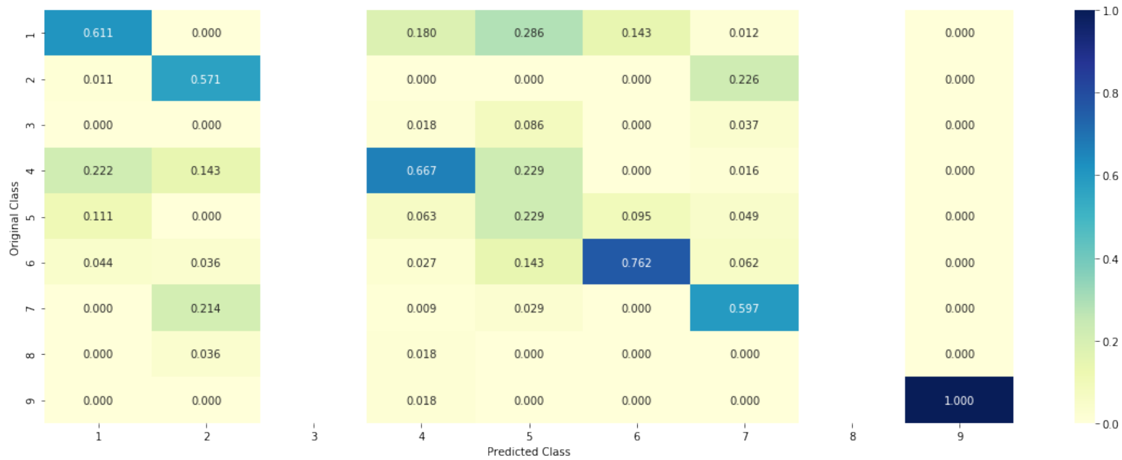
In [64]:
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modul
# ----------------------------
# 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(f

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 estim
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_tfid
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_loss
                    'log loss Test':[log_loss(test_y, sig_clf.predict_proba(test_x_tfi
aa=aa.append(bb)
```

```
Log Loss : 1.21412090082
Number of missclassified point : 0.40225563909774437
-------------------- Confusion matrix --------------------
```



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



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

alpha :1

4.1.1.3. Feature Importance, Correctly classified point

```
In [65]: 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_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: [[ 0.0319  0.0497  0.0245  0.0313  0.0286  0.029   0.7949  0.00
Actual Class : 7
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

```
In [66]: 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_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 : 2
Predicted Class Probabilities: [[ 0.0541  0.4344  0.0243  0.0531  0.047   0.0517  0.3221  0.00
Actual Class : 7
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

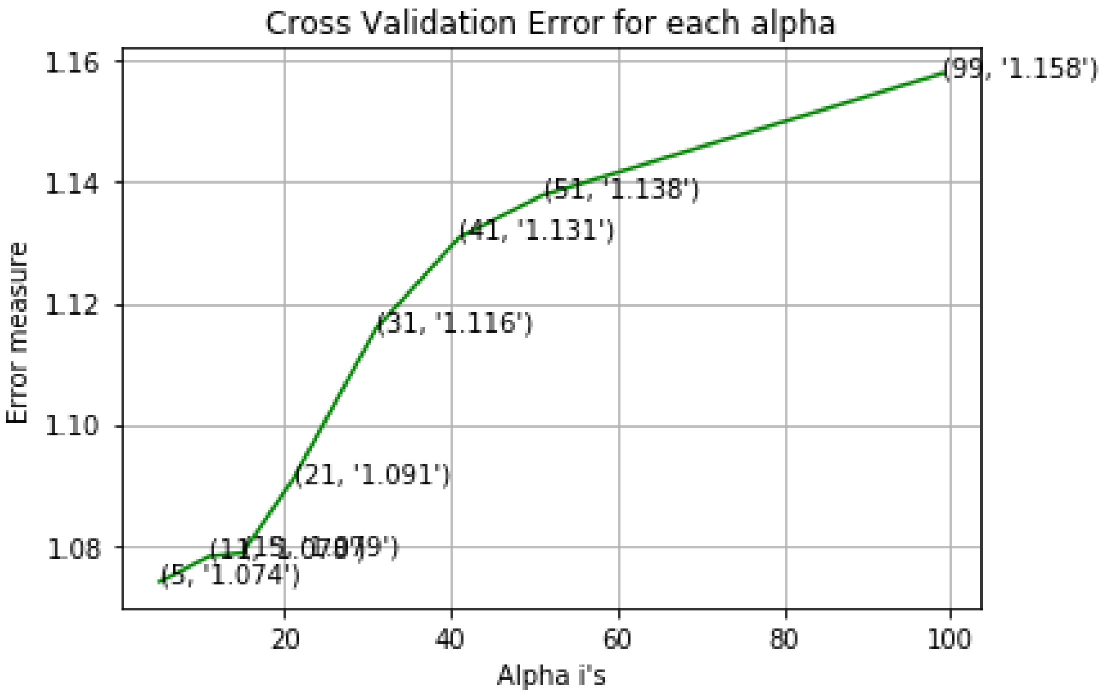
4.2. K Nearest Neighbour Classification
4.2.1. Hyper parameter tuning

```
In [68]: # 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/lessons
        #--------------------------------------
```

```
# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modul
# ----------------------------
# 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 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 = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

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

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

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

```
for alpha = 5
Log Loss : 1.07396828121
for alpha = 11
Log Loss : 1.0782165235
for alpha = 15
Log Loss : 1.07878858094
for alpha = 21
Log Loss : 1.09084818902
for alpha = 31
Log Loss : 1.1158020833
for alpha = 41
Log Loss : 1.13085658127
for alpha = 51
Log Loss : 1.13783753968
for alpha = 99
Log Loss : 1.15795238243
```



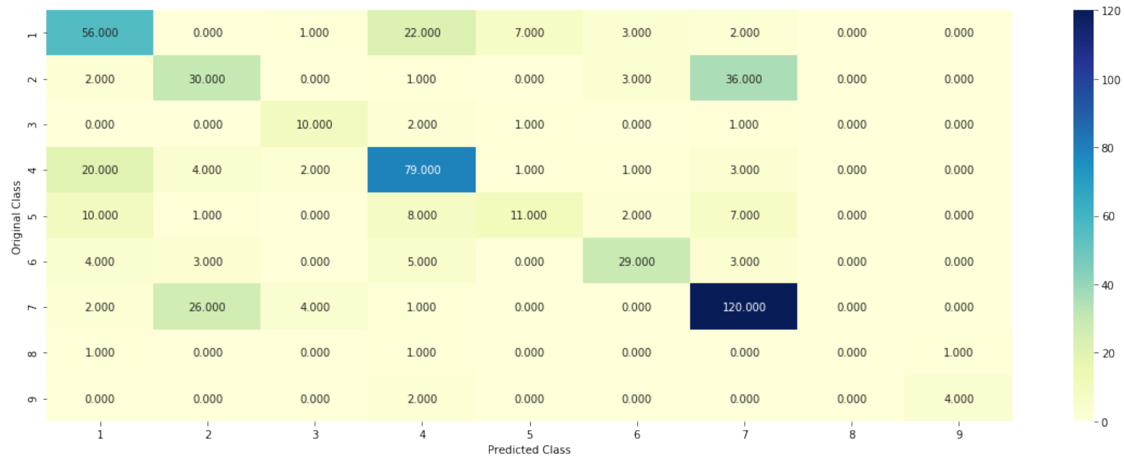

```
For values of best alpha =  5 The train log loss is: 0.484686008081
For values of best alpha =  5 The cross validation log loss is: 1.07396828121
For values of best alpha =  5 The test log loss is: 1.11081789433
```

4.2.2. Testing the model with best hyper paramters

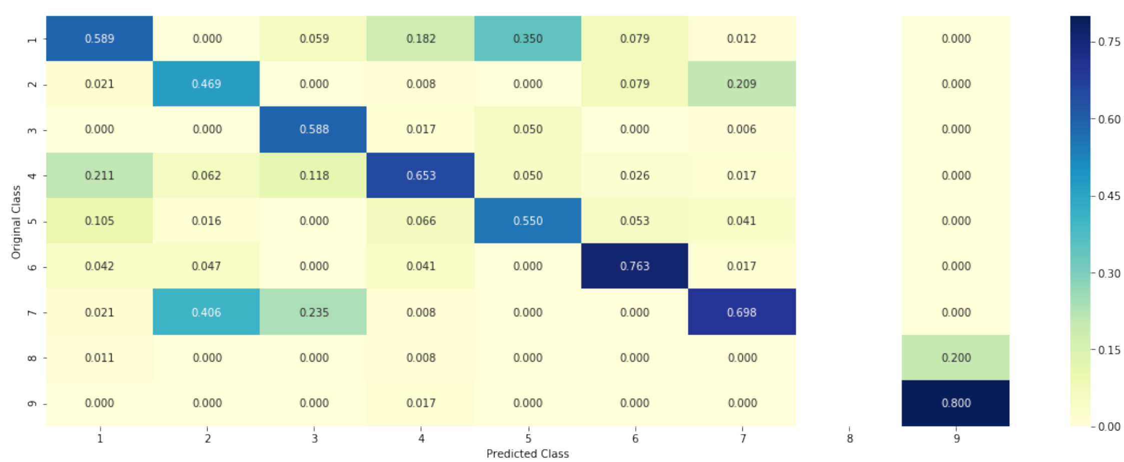
In [69]:
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights=uniform, algorithm=auto, leaf_size=30, 
# metric=minkowski, metric_params=None, n_jobs=1, **kwargs)

# 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/lessons
#-------------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCodin
```

```
Log loss : 1.07396828121
Number of mis-classified points : 0.36278195488721804
```

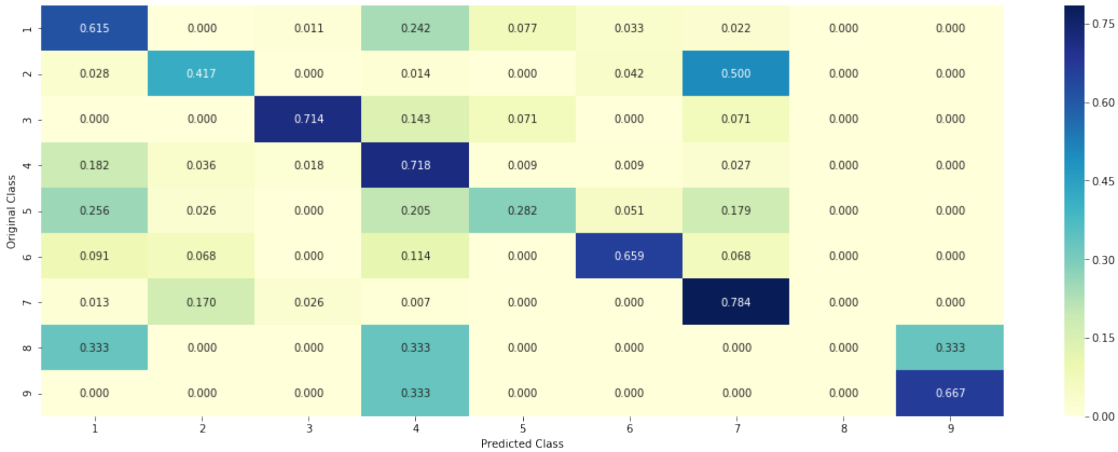
------------------- Confusion matrix --------------------



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



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



4.2.3.Sample Query point -1

```
In [70]: 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 clas
         print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 1
Actual Class : 7
The  5  nearest neighbours of the test points belongs to classes [7 7 7 3 3]
Fequency of nearest points : Counter({7: 3, 3: 2})
```

4.2.4. Sample Query Point-2

```
In [71]: clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
         clf.fit(train_x_responseCoding, train_y)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_x_responseCoding, train_y)

         test_point_index = 100

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

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

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

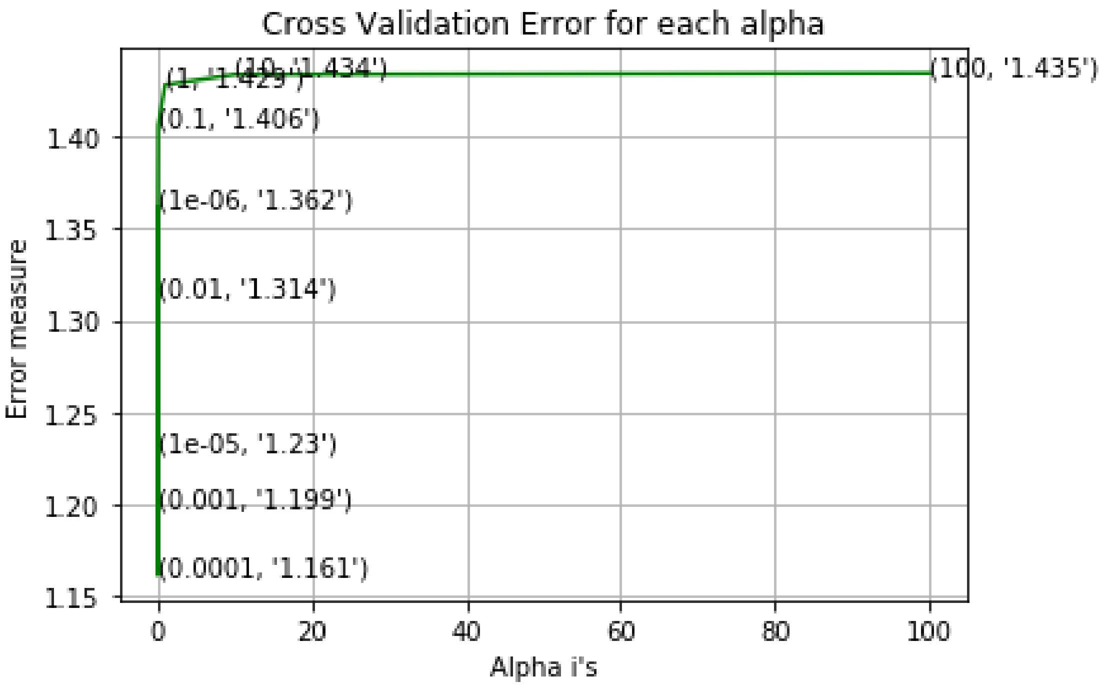
4.3. Logistic Regression
4.3.1. With Class balancing
4.3.1.1. Hyper paramter tuning

In [72]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
# 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 Gr
# 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/modul
# ----------------------------
# 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(f


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

```
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', ra
    clf.fit(train_x_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', lo
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_l
predict_y = sig_clf.predict_proba(cv_x_feature)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
predict_y = sig_clf.predict_proba(test_x_feature)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo

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

```
for alpha = 1e-06
Log Loss : 1.36202690608
for alpha = 1e-05
Log Loss : 1.22983091293
for alpha = 0.0001
Log Loss : 1.16131603047
for alpha = 0.001
```

```
Log Loss : 1.19926606089
for alpha = 0.01
Log Loss : 1.3142286555
for alpha = 0.1
Log Loss : 1.40630033735
for alpha = 1
Log Loss : 1.42852537391
for alpha = 10
Log Loss : 1.43392328692
for alpha = 100
Log Loss : 1.43465566179
```



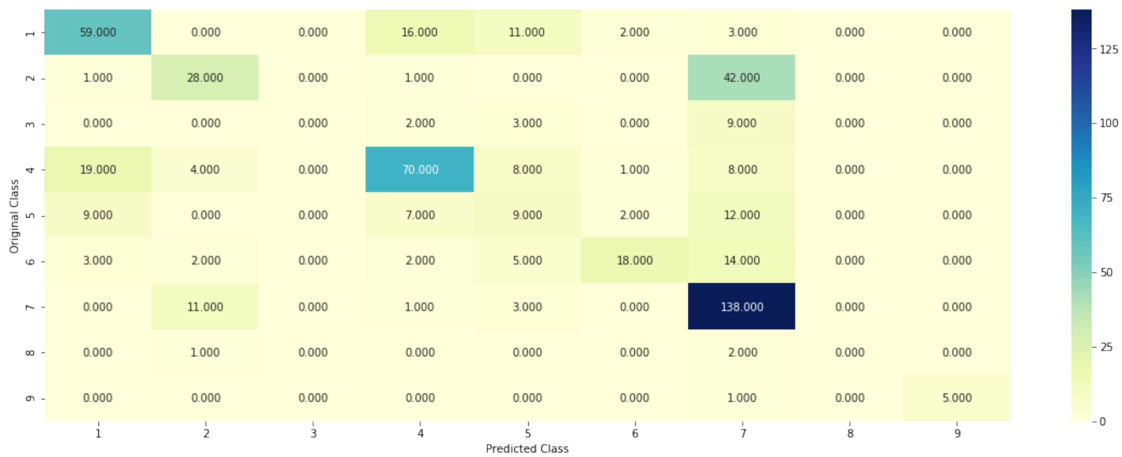

```
For values of best alpha =  0.0001 The train log loss is: 0.551163605053
For values of best alpha =  0.0001 The cross validation log loss is: 1.16131603047
For values of best alpha =  0.0001 The test log loss is: 1.140643371
```

4.3.1.2. Testing the model with best hyper paramters

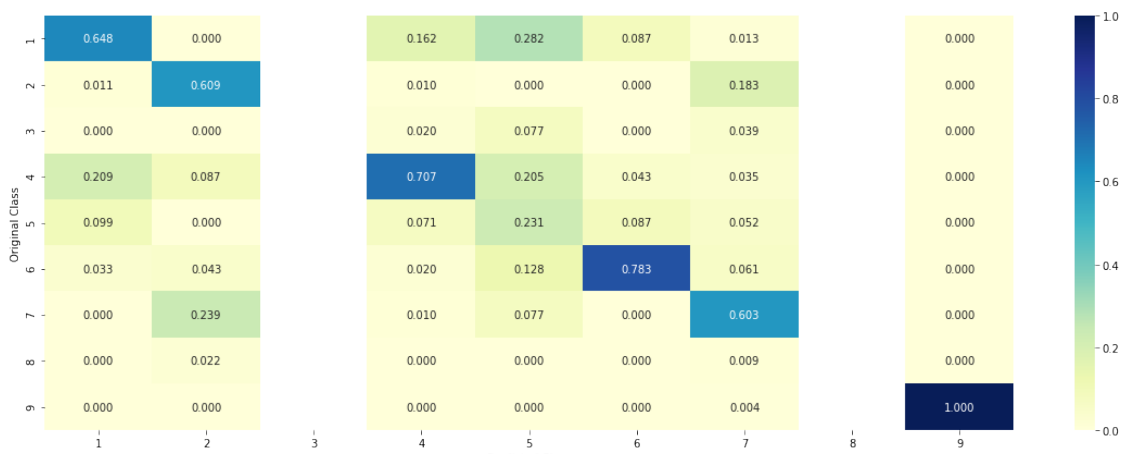
In [73]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None,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

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

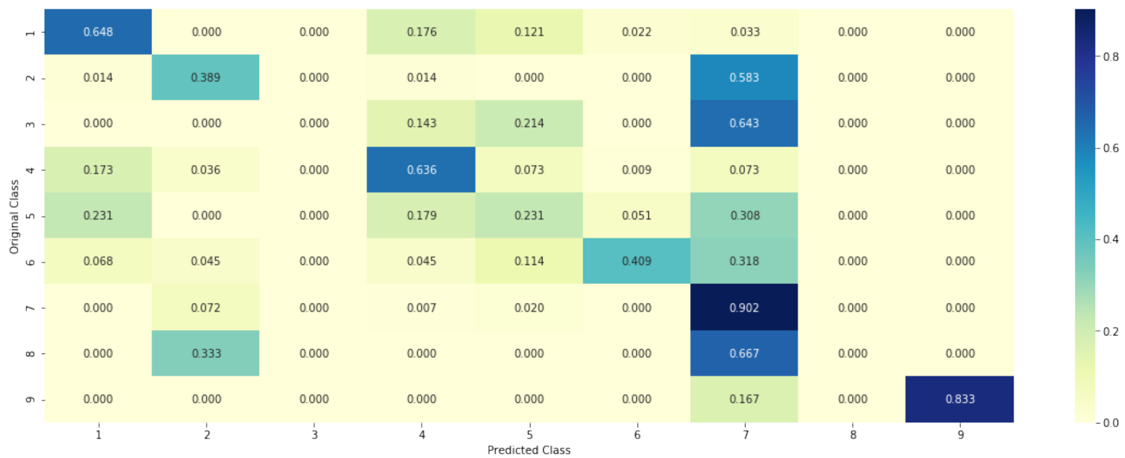
```
Log loss : 1.16131603047
Number of mis-classified points : 0.38533834586466165
-------------------- Confusion matrix --------------------
```




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



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



4.3.1.3. Feature Importance

```
In [74]: 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 [75]: # from tabulate import tabulate
         clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', lo
```

```
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: [[ 0.018   0.1514  0.0347  0.0273  0.0387  0.0155  0.7037  0.004
Actual Class : 7
--------------------------------------------------
449 Text feature [12] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

```
In [76]: 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 : 2
Predicted Class Probabilities: [[ 0.0421  0.4864  0.0187  0.0433  0.0566  0.061   0.2784  0.005
Actual Class : 7
--------------------------------------------------
157 Text feature [10q] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

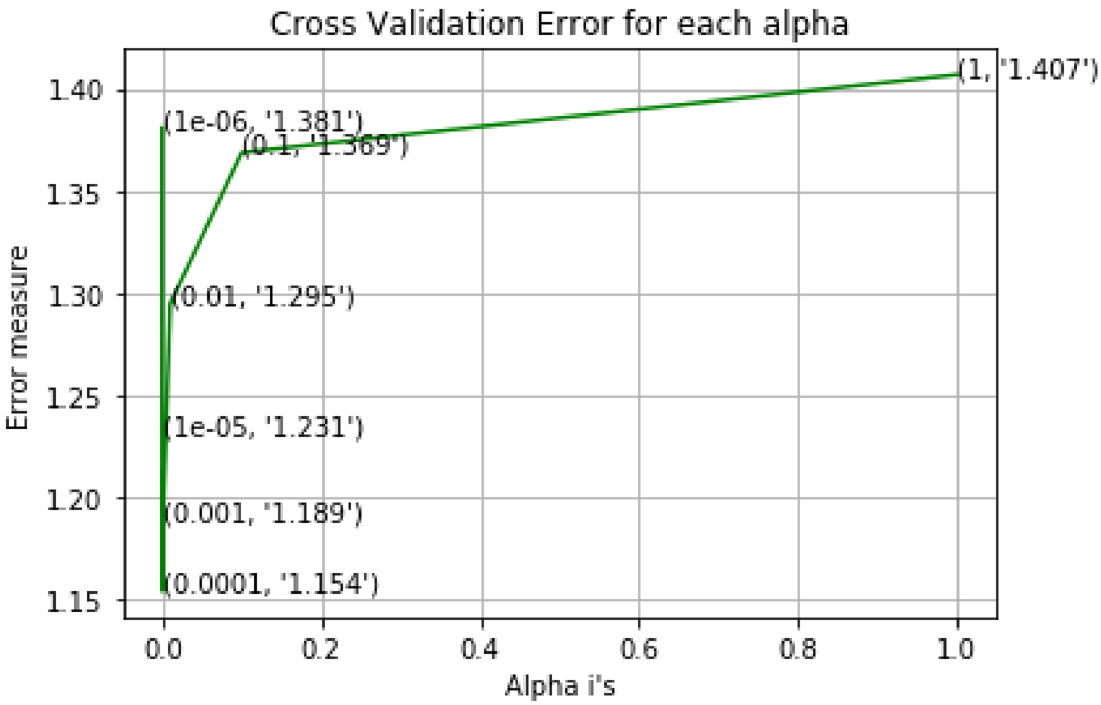
4.3.2. Without Class balancing
4.3.2.1. Hyper paramter tuning

```
In [78]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
         # ------------------------------
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
         # 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)
```

```python
# some of methods
# fit(X, y[, coef_init, intercept_init, ])        Fit linear model with Stochastic Gr
# 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/modul
# ----------------------------
# 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_l
    predict_y = sig_clf.predict_proba(cv_x_feature)
    print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
    predict_y = sig_clf.predict_proba(test_x_feature)
    print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo

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

```
for alpha = 1e-06
Log Loss : 1.38087609088
for alpha = 1e-05
Log Loss : 1.23132712867
for alpha = 0.0001
Log Loss : 1.15374445919
for alpha = 0.001
Log Loss : 1.1892318873
for alpha = 0.01
Log Loss : 1.29461406025
for alpha = 0.1
Log Loss : 1.36877894356
for alpha = 1
Log Loss : 1.40693732554
```

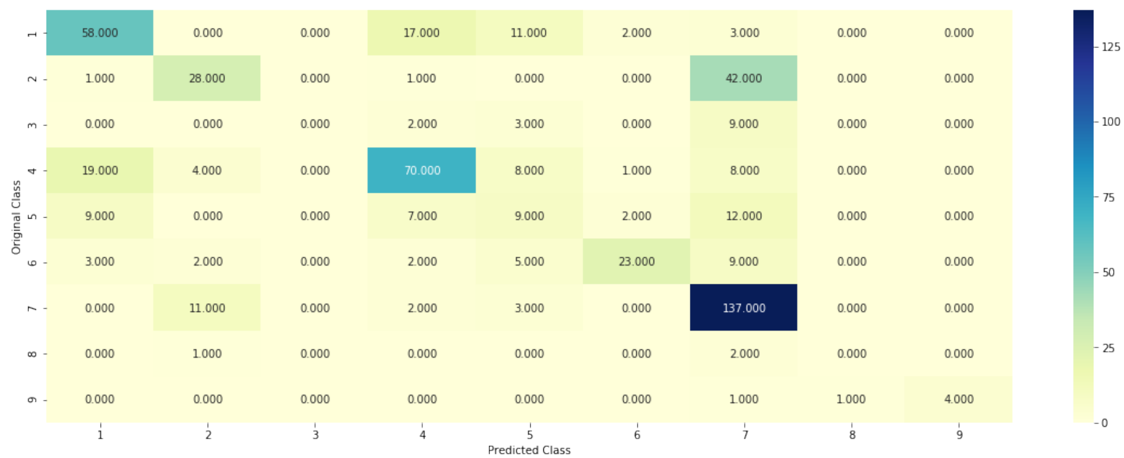



```
For values of best alpha =  0.0001 The train log loss is: 0.528582769322
For values of best alpha =  0.0001 The cross validation log loss is: 1.15374445919
For values of best alpha =  0.0001 The test log loss is: 1.1405008001
```

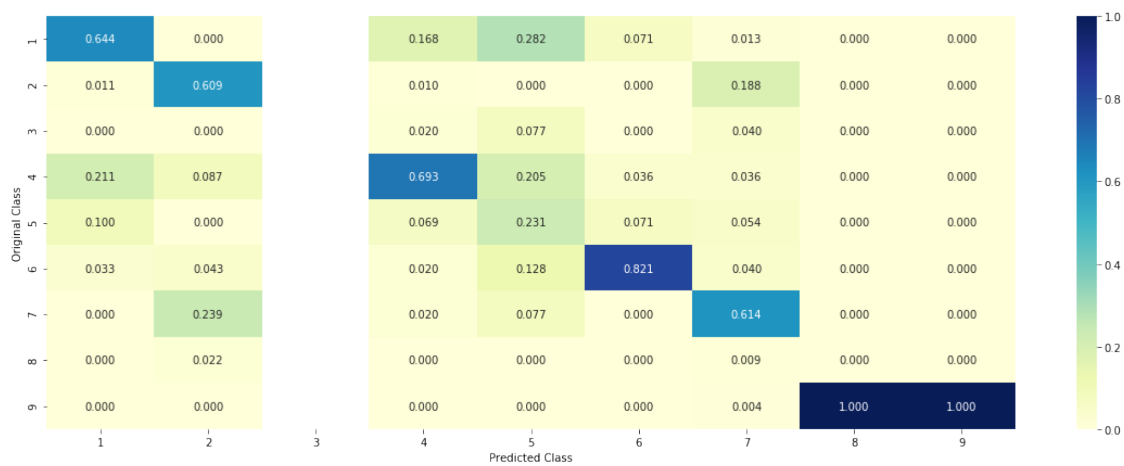
4.3.2.2. Testing model with best hyper parameters

In [79]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
# 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 Gr
# predict(X)        Predict class labels for samples in X.

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

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42
predict_and_plot_confusion_matrix(train_x_feature, train_y, cv_x_feature, cv_y, clf)
```

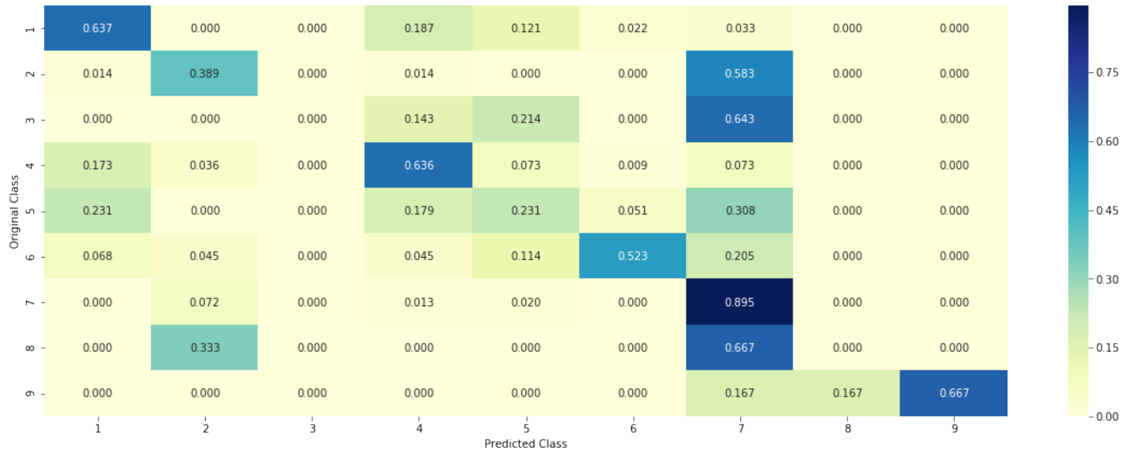
```
Log loss : 1.15374445919
Number of mis-classified points : 0.3815789473684211
-------------------- Confusion matrix --------------------
```



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



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



4.3.2.3. Feature Importance, Correctly Classified point

```
In [80]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42
         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: [[ 0.0175  0.1644  0.0509  0.0301  0.0386  0.0156  0.6761  0.003
Actual Class : 7
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

```
In [81]: 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 : 2
Predicted Class Probabilities: [[ 0.0396  0.4777  0.0178  0.0396  0.055   0.0582  0.3006  0.005
Actual Class : 7
--------------------------------------------------
140 Text feature [10q] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

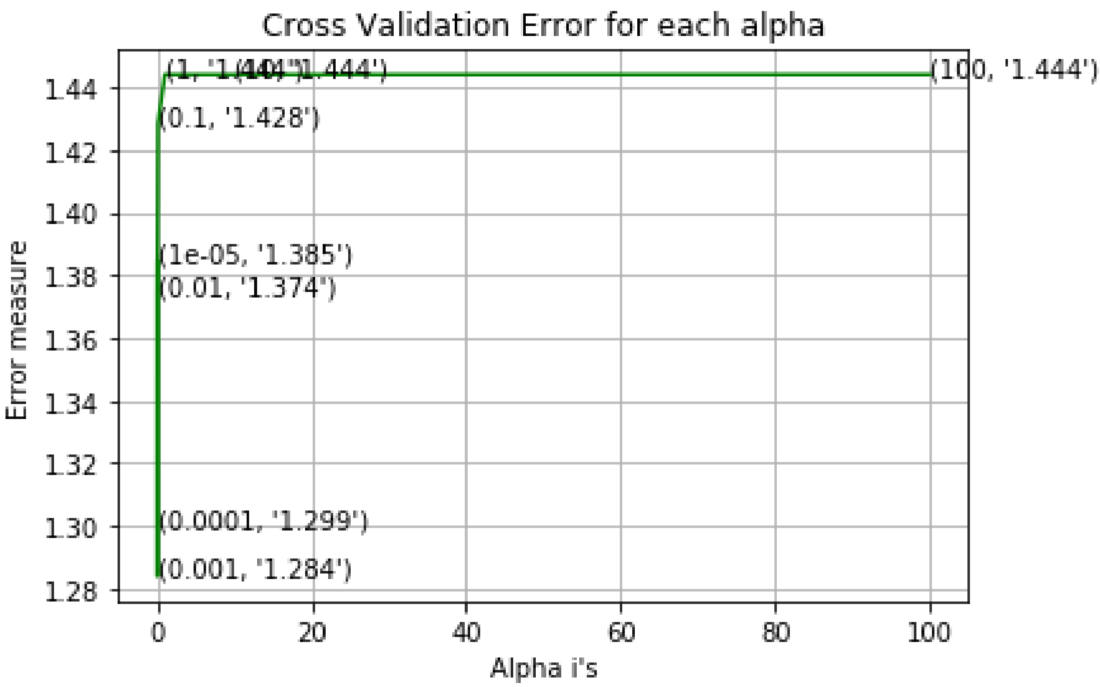
4.4. Linear Support Vector Machines
4.4.1. Hyper paramter tuning

In [82]:
```
# read more about support vector machines with linear kernals here http://scikit-lear

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

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



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modul
# ----------------------------
# 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', lo
    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_l
    predict_y = sig_clf.predict_proba(cv_x_tfidf1000)
    print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
    predict_y = sig_clf.predict_proba(test_x_tfidf1000)
    print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo

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

```
for C = 1e-05
Log Loss : 1.3845216042
for C = 0.0001
Log Loss : 1.29923895475
for C = 0.001
Log Loss : 1.28404894103
for C = 0.01
Log Loss : 1.37399924272
for C = 0.1
Log Loss : 1.4284326375
for C = 1
Log Loss : 1.44389713903
```

```
for C = 10
Log Loss : 1.4438971562
for C = 100
Log Loss : 1.44389492569
```



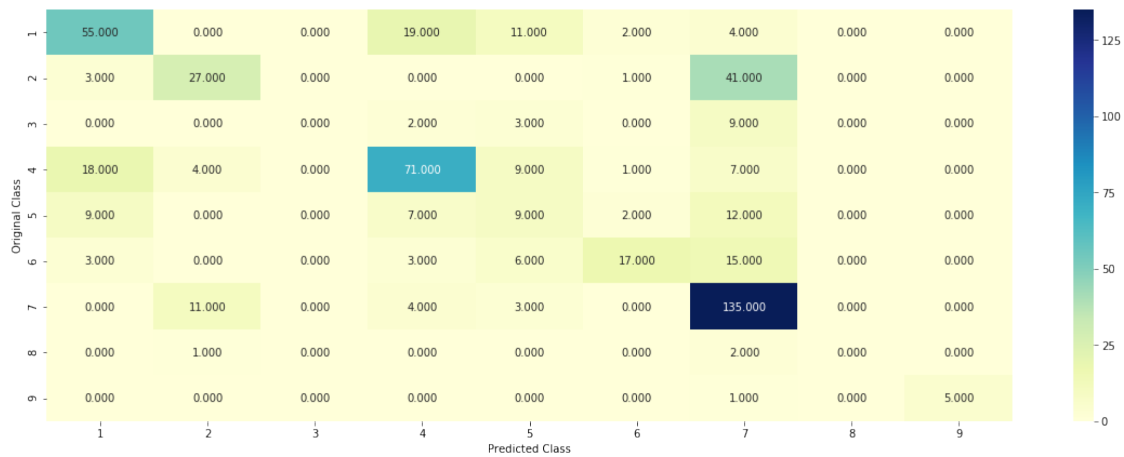

```
For values of best alpha =  0.001 The train log loss is: 0.661156819791
For values of best alpha =  0.001 The cross validation log loss is: 1.28404894103
For values of best alpha =  0.001 The test log loss is: 1.27592718571
```

4.4.2. Testing model with best hyper parameters

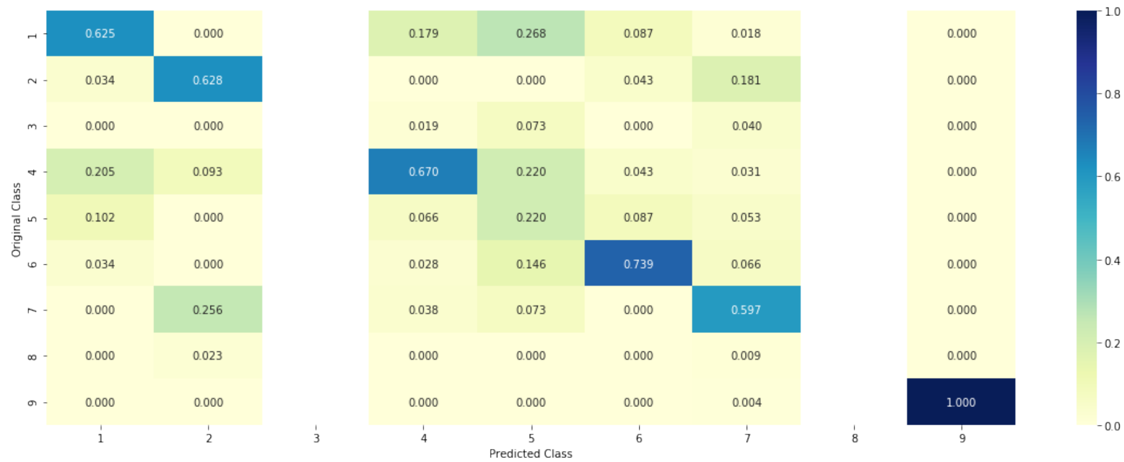
In [83]:
```
# read more about support vector machines with linear kernals here http://scikit-lear

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

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

```
# --------------------------------


# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='balan
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, clf
```

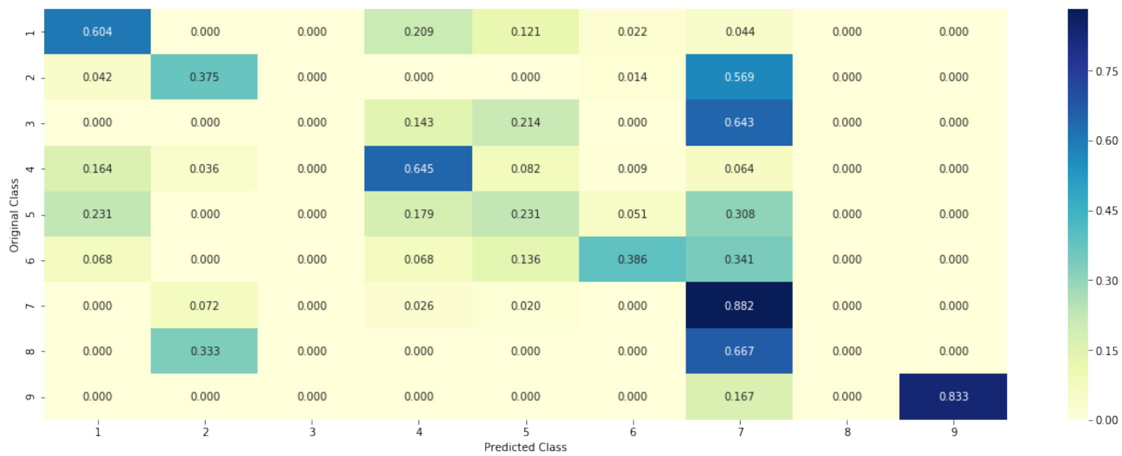
```
Log loss : 1.28404894103
Number of mis-classified points : 0.40037593984962405
-------------------- Confusion matrix --------------------
```




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



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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

```
In [84]: 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_tfidf100
         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: [[ 0.0664  0.1279  0.0142  0.0914  0.0339  0.037   0.6208  0.003
Actual Class : 7
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

4.3.3.2. For Incorrectly classified point

```
In [85]: 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_tfidf100
```

```
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 : 2
Predicted Class Probabilities: [[ 0.099   0.4161  0.0259  0.1071  0.0611  0.0598  0.2192  0.005
Actual Class : 7
--------------------------------------------------
286 Text feature [10q] present in test data point [True]
441 Text feature [0449] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.5 Random Forest Classifier
4.5.1. Hyper paramter tuning (With One hot Encoding)

In [90]:
```
# --------------------------------
# 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_nodes
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=No
# class_weight=None)

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

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

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


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

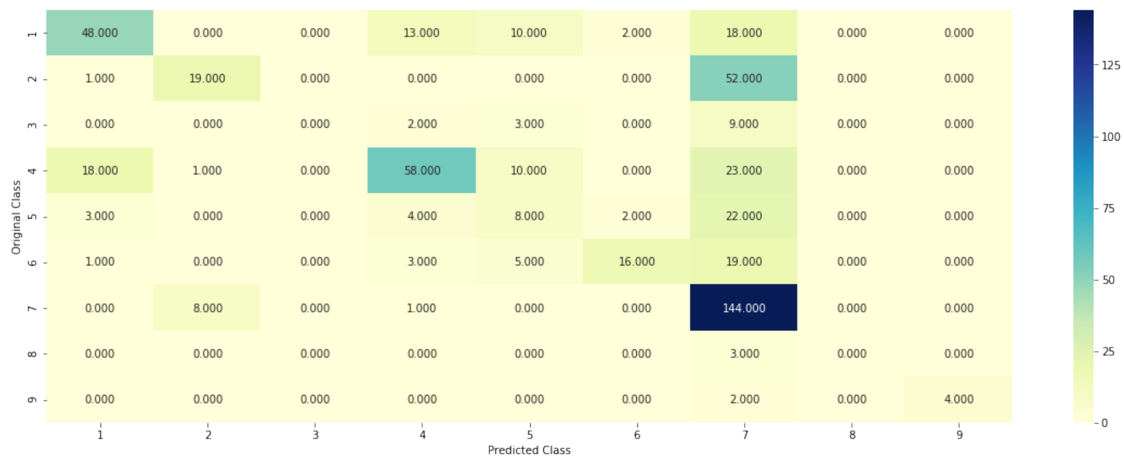
```
#-------------------------------------
# video link:
#-------------------------------------

alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, ra
        clf.fit(train_x_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_, e
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

'''fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)), (features[i],cv_log_e
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 loss
predict_y = sig_clf.predict_proba(cv_x_tfidf1000)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validati
predict_y = sig_clf.predict_proba(test_x_tfidf1000)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss

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, sig
                   'log loss Test':[log_loss(y_test, sig_clf.predict_proba(test_x_tfic
aa=aa.append(bb)
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.33341692818
for n_estimators = 100 and max depth =  10
Log Loss : 1.30461264477
for n_estimators = 200 and max depth =  5
Log Loss : 1.31365656882
for n_estimators = 200 and max depth =  10
Log Loss : 1.30160399367
for n_estimators = 500 and max depth =  5
Log Loss : 1.29811077325
for n_estimators = 500 and max depth =  10
Log Loss : 1.28744854611
for n_estimators = 1000 and max depth =  5
Log Loss : 1.29590513624
for n_estimators = 1000 and max depth =  10
Log Loss : 1.28681278263
for n_estimators = 2000 and max depth =  5
Log Loss : 1.29396233262
for n_estimators = 2000 and max depth =  10
Log Loss : 1.28599841279
For values of best estimator =  2000 The train log loss is: 1.00758447601
For values of best estimator =  2000 The cross validation log loss is: 1.28599841279
For values of best estimator =  2000 The test log loss is: 1.27108765453
```

4.5.2. Testing model with best hyper parameters (TFIDF top 1000 words)

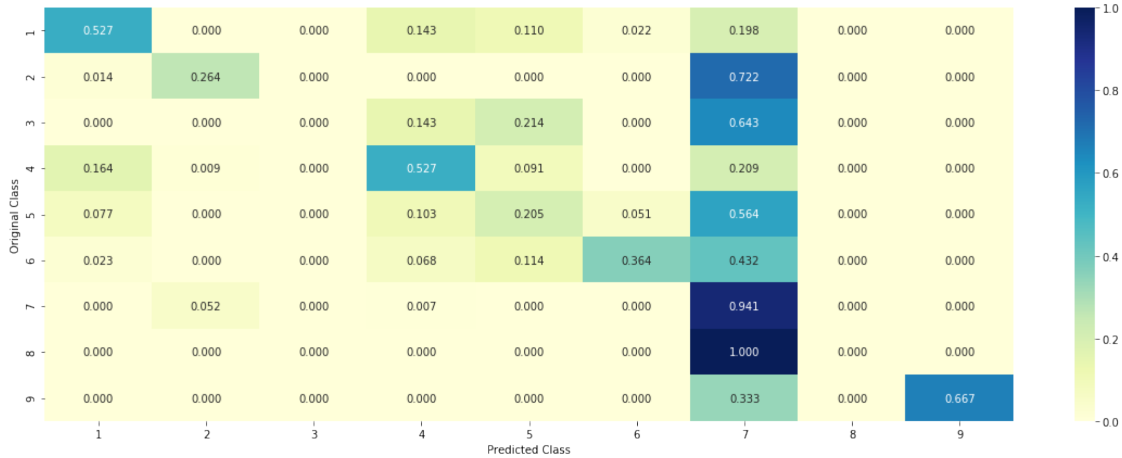
In [91]:
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth=N
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_nodes=
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=Non
# class_weight=None)

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

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

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

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

```
Log loss : 1.28599841279
Number of mis-classified points : 0.4417293233082707
-------------------- 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 [92]:
```
# 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_tfidf100
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],tes
```

```
Predicted Class : 7
Predicted Class Probabilities: [[ 0.0431  0.0482  0.0669  0.0444  0.034   0.0322  0.7221  0.00
Actual Class : 7
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

4.5.3.2. Inorrectly Classified point

In [93]:
```
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_tfidf10
```

```
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],tes
```

```
Predicted Class : 2
Predicted Class Probabilities: [[ 0.1159  0.3143  0.0248  0.1315  0.0564  0.0657  0.2772  0.006
Actuall Class : 7
--------------------------------------------------
87 Text feature [029] 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 [94]: # --------------------------------
         # 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_nodes
         # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=No
         # class_weight=None)

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

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

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


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

```
#--------------------------------------

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

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

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log loss is
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross validation l
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_alpha%
bb=pd.DataFrame({'type':['RF response coding'],'hyperparameter':[xx],'log loss CV':[lo
                   'log loss Test':[log_loss(y_test, sig_clf.predict_proba(test_x_resp
aa=aa.append(bb)
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.12215177523
```

```
for n_estimators = 10 and max depth =  3
Log Loss : 1.9075218561
for n_estimators = 10 and max depth =  5
Log Loss : 1.4572732625
for n_estimators = 10 and max depth =  10
Log Loss : 1.75588041664
for n_estimators = 50 and max depth =  2
Log Loss : 1.7896740243
for n_estimators = 50 and max depth =  3
Log Loss : 1.50652012325
for n_estimators = 50 and max depth =  5
Log Loss : 1.40582094844
for n_estimators = 50 and max depth =  10
Log Loss : 1.5905191437
for n_estimators = 100 and max depth =  2
Log Loss : 1.65420128083
for n_estimators = 100 and max depth =  3
Log Loss : 1.5758314393
for n_estimators = 100 and max depth =  5
Log Loss : 1.40965556238
for n_estimators = 100 and max depth =  10
Log Loss : 1.64743672281
for n_estimators = 200 and max depth =  2
Log Loss : 1.66577907462
for n_estimators = 200 and max depth =  3
Log Loss : 1.56831984598
for n_estimators = 200 and max depth =  5
Log Loss : 1.45068916511
for n_estimators = 200 and max depth =  10
Log Loss : 1.61743457878
for n_estimators = 500 and max depth =  2
Log Loss : 1.70861432952
for n_estimators = 500 and max depth =  3
Log Loss : 1.60922287307
for n_estimators = 500 and max depth =  5
Log Loss : 1.45043523113
for n_estimators = 500 and max depth =  10
Log Loss : 1.67424922721
for n_estimators = 1000 and max depth =  2
Log Loss : 1.67409076911
for n_estimators = 1000 and max depth =  3
Log Loss : 1.60005232969
for n_estimators = 1000 and max depth =  5
Log Loss : 1.42341748452
for n_estimators = 1000 and max depth =  10
Log Loss : 1.65673085324
For values of best alpha =  50 The train log loss is: 0.0578810059565
For values of best alpha =  50 The cross validation log loss is: 1.40582097518
```

```
For values of best alpha =  50 The test log loss is: 1.4396231931
```

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

```
In [95]: # --------------------------------
         # default parameters
         # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth=N
         # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_nodes=
         # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=Non
         # class_weight=None)

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

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

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

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

```
Log loss : 1.40582094844
Number of mis-classified points : 0.5037593984962406
-------------------- 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 [96]: 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
```

```python
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_response
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 2
Predicted Class Probabilities: [[ 0.0111  0.5733  0.1706  0.0114  0.0155  0.0207  0.1614  0.02
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

```
In [97]: test_point_index = 100
         predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_response
         print("Actual Class :", test_y[test_point_index])
         indices = np.argsort(-clf.feature_importances_)
         print("-"*50)
         for i in indices:
             if i<9:
                 print("Gene is important feature")
             elif i<18:
                 print("Variation is important feature")
             else:
                 print("Text is important feature")
```

```
Predicted Class : 2
Predicted Class Probabilities: [[ 0.0088  0.8081  0.0565  0.0091  0.0112  0.0187  0.0666  0.01
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [98]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
# -------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
# 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 Gr
# predict(X)        Predict class labels for samples in X.

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



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

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



# read more about support vector machines with linear kernals here http://scikit-lear
# ---------------------------------
# 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_nodes
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=No
# class_weight=None)

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

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

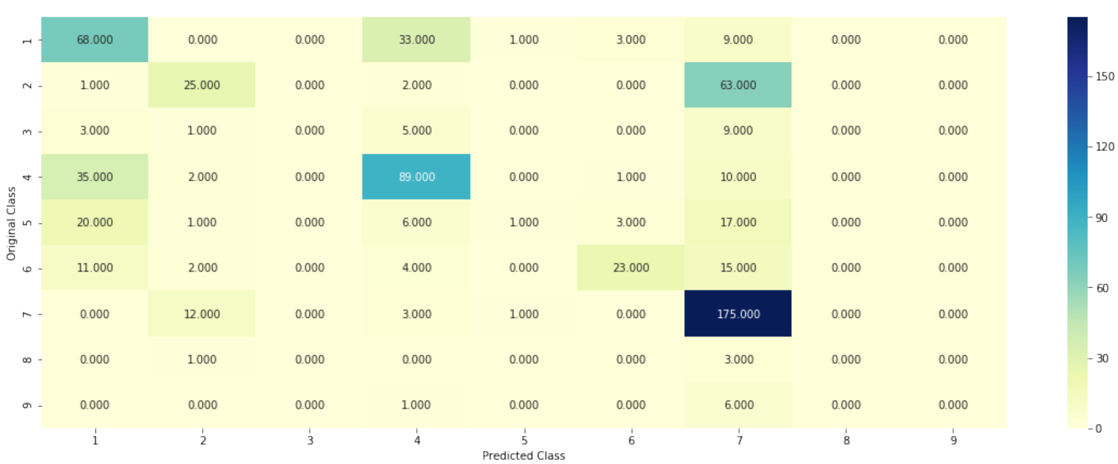
```python
# ------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/
# ------------------------------------


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

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', ran
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_prob
sig_clf2.fit(train_x_tfidf1000, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_p
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_tf
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_classi
    sclf.fit(train_x_tfidf1000, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_
    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: 1.18
Support vector machines : Log Loss: 1.44
Naive Bayes : Log Loss: 1.32
--------------------------------------------------
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: 1.773
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.323
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.253
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.493
```

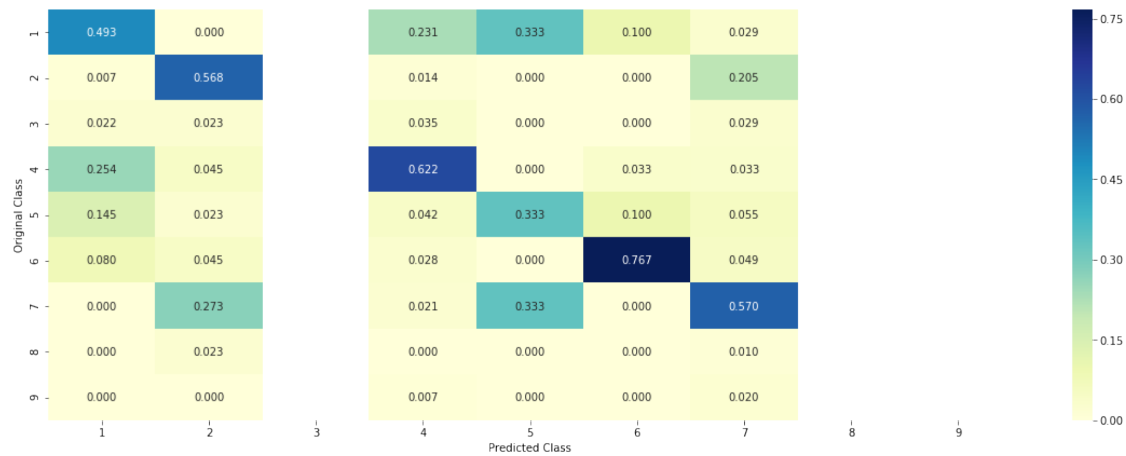
4.7.2 testing the model with the best hyper parameters

```
In [101]: 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)
```

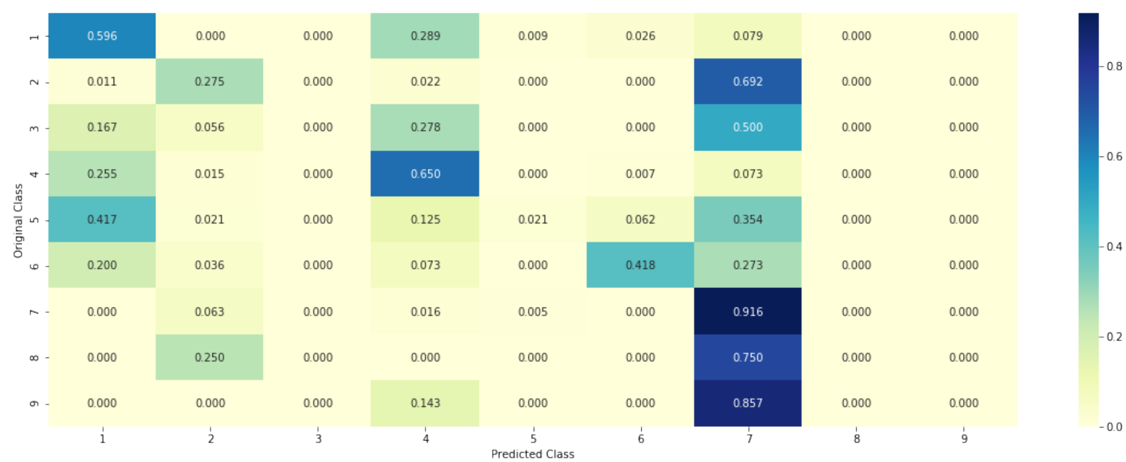
```
Log loss (train) on the stacking classifier : 0.97351296611
Log loss (CV) on the stacking classifier : 1.3232074977
Log loss (test) on the stacking classifier : 1.32496292862
Number of missclassified point : 0.4270676691729323
-------------------- Confusion matrix --------------------
```



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




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

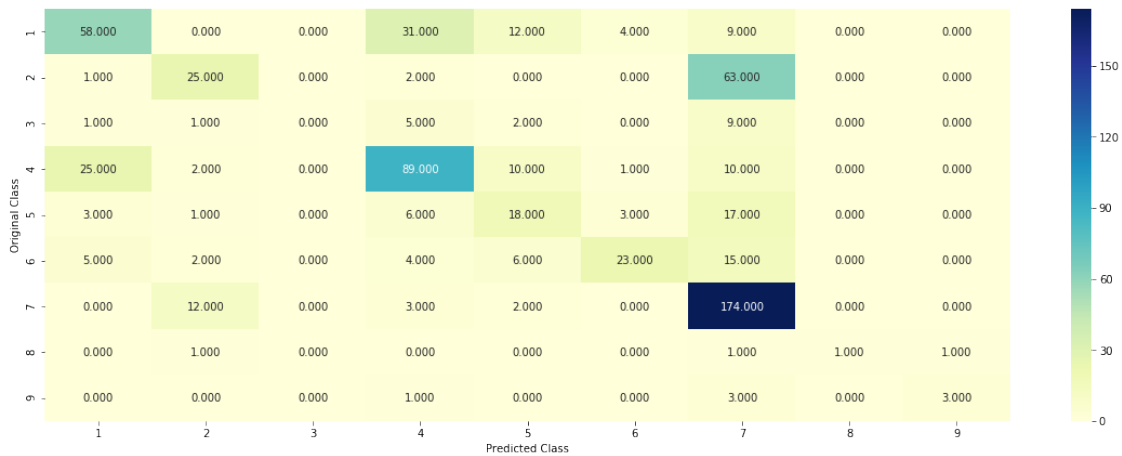


4.7.3 Maximum Voting classifier

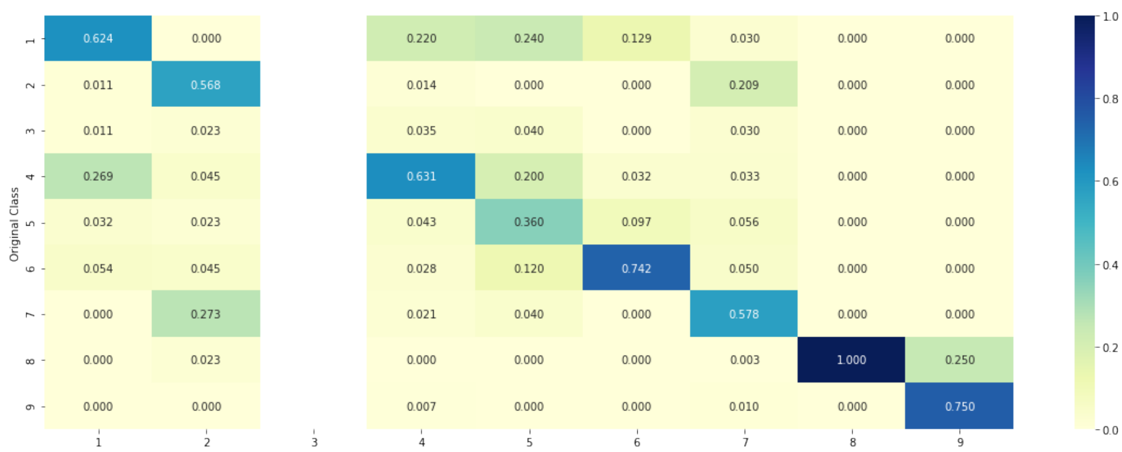
In [106]:
```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClass
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_
vclf.fit(train_x_tfidf1000, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_p
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(c
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_pro
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_tfid
```

```
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)
```

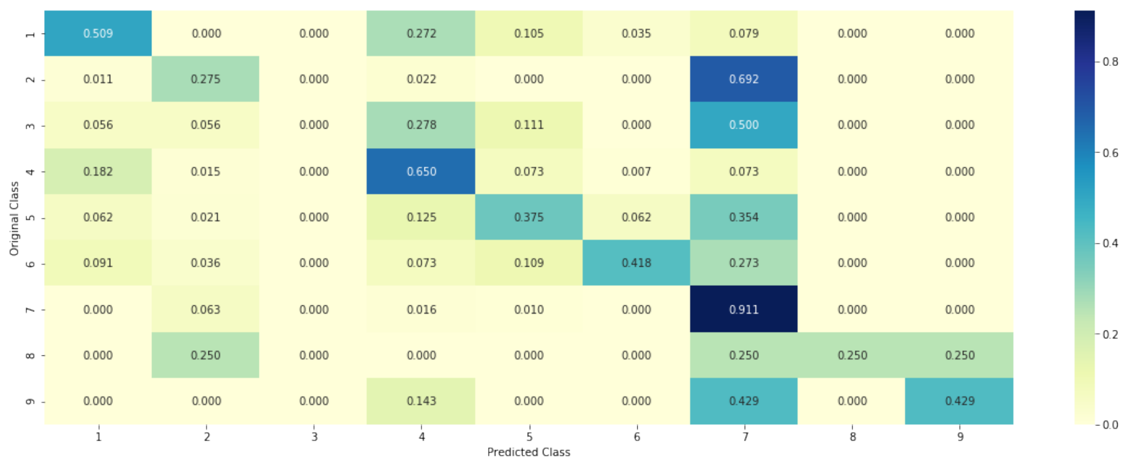
```
Log loss (train) on the VotingClassifier : 0.821002912704
Log loss (CV) on the VotingClassifier : 1.26163758132
Log loss (test) on the VotingClassifier : 1.24908604441
Number of missclassified point : 0.4120300751879699
-------------------- Confusion matrix --------------------
```




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



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



5. End

In [107]: aa

Out[107]:

```
                 hyperparameter  log loss CV  log loss Test  \
0                            NA     2.443423       2.541630
0                     alpha :1     1.214121       1.212589
0                         k :5     1.073968       1.110818
0                    C :0.0001     1.161316       1.140643
0                    C :0.0001     1.153744       1.140501
0                     C :0.001     1.284049       1.275927
0      n_estimator :50depth5     1.405821       1.439623
0  n_estimator :2000depth10     1.285998       1.271088
0      n_estimator :50depth5     1.405821       1.439623
0                            na     1.323207       1.324963
0                            na     1.261638       1.249086

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

```
0        RF response coding
0                     stack
0                max voting
```