

# PersonalizedCancerDiagnosis

June 22, 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.

## 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 contains 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 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 class classification

### 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 Learning 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 probabilities => Metric is Log-loss.
- No Latency constraints.

### 2.3. Train, CV and Test Datasets

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

## 3. Exploratory Data Analysis

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

from sklearn import model_selection
from sklearn.linear_model import LogisticRegression

/glob/intel-python/versions/2018u2/intelpython3/lib/python3.6/site-packages/sklearn/cross_vali
  "This module will be removed in 0.20.", DeprecationWarning)

```

### 3.1. Reading Data

#### 3.1.1. Reading Gene and Variation Data

```

In [2]: 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/training\_variants is a comma separated file containing the description of the genetic 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 separator 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  Cyclin-dependent kinases (CDKs) regulate a var...
          1  Abstract Background Non-small cell lung canc...
          2  Abstract Background Non-small cell lung canc...
          3  Recent evidence has demonstrated that acquired...
          4  Oncogenic mutations in the monomeric Casitas B...
```

### 3.1.3. Preprocessing of text

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


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

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

        data_text[column][index] = string

In [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 : 131.46 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 lineage...
```

### 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 variable
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true, random_state=42)
# split the train data into train and cross validation by maintaining same distribution of output variable
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, random_state=42)
```

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])
```

```

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

#### 3.1.4.2. Distribution of y\_i's in Train, Test and Cross Validation datasets

```

In [9]: # it returns a dict, keys as class labels and values as the number of data points in that class
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 = list('rgbkymc')
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]

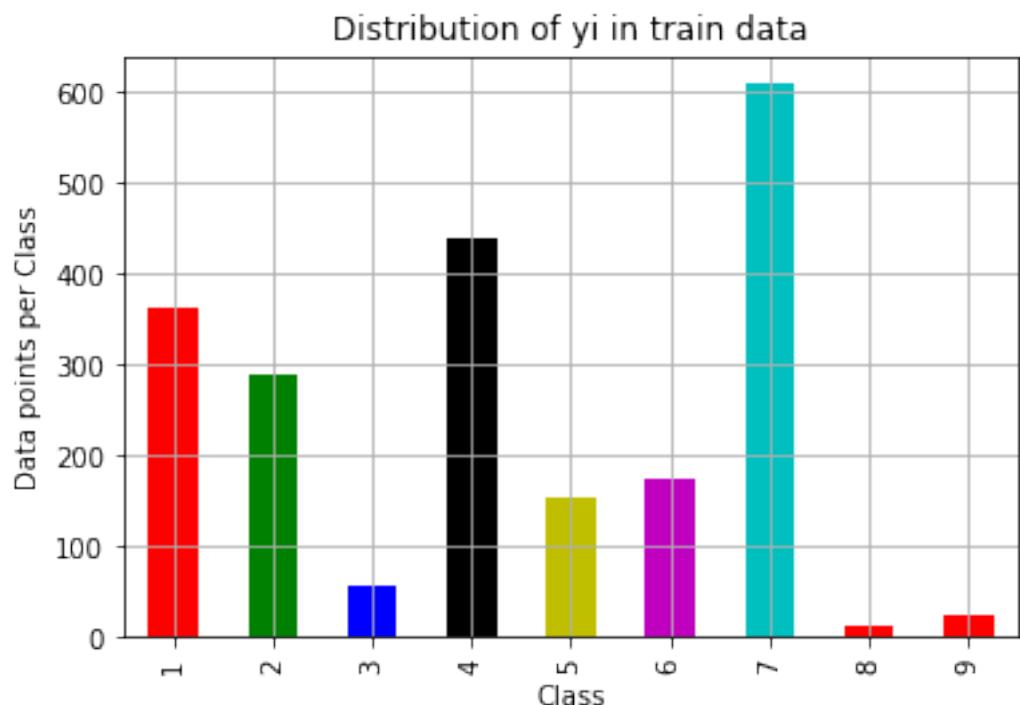
print('*'*80)
my_colors = list('rgbkymc')
test_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

# 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 = list('rgbkymc')
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()

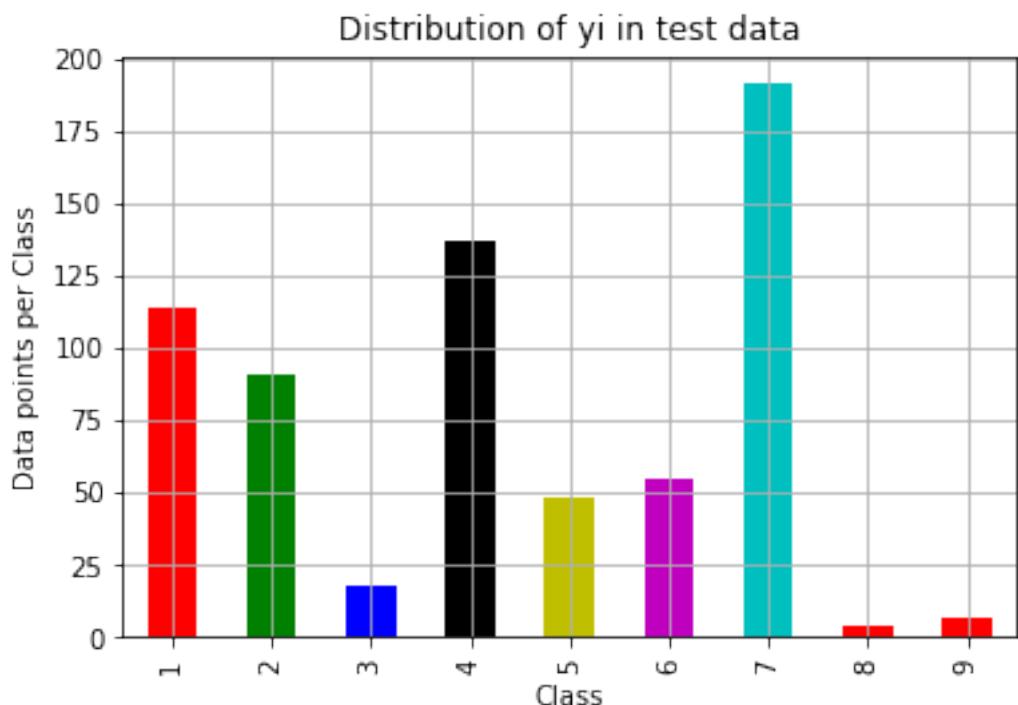
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.values[i],

```



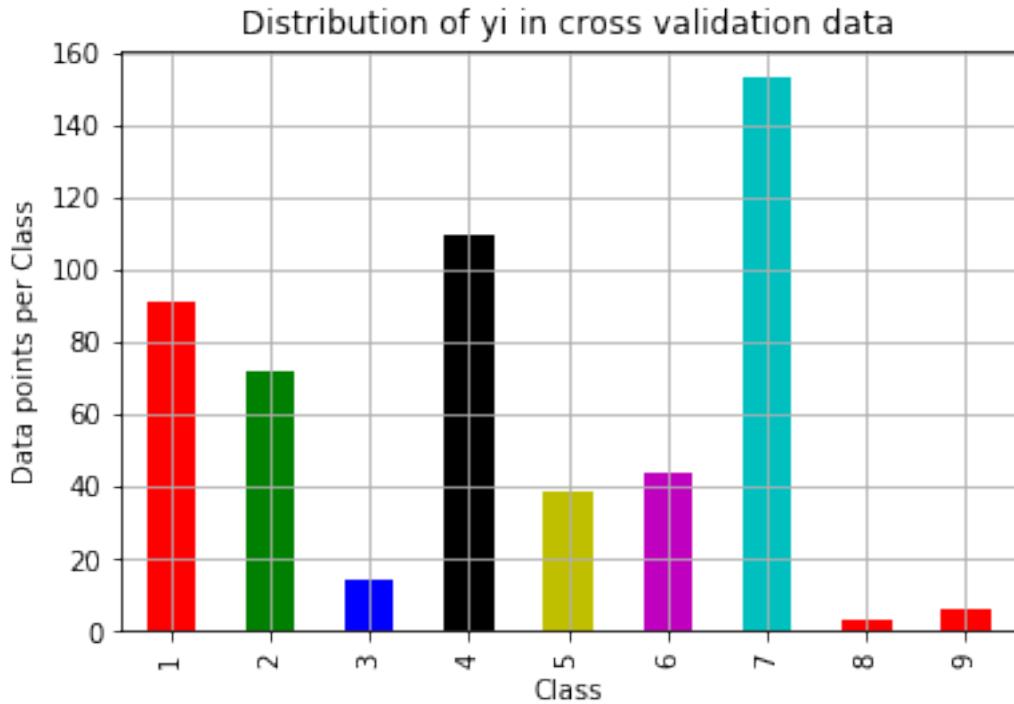
Number of data points in class 7 : 609 ( 28.672 %)  
Number of data points in class 4 : 439 ( 20.669 %)  
Number of data points in class 1 : 363 ( 17.09 %)  
Number of data points in class 2 : 289 ( 13.606 %)  
Number of data points in class 6 : 176 ( 8.286 %)  
Number of data points in class 5 : 155 ( 7.298 %)  
Number of data points in class 3 : 57 ( 2.684 %)  
Number of data points in class 9 : 24 ( 1.1300000000000001 %)  
Number of data points in class 8 : 12 ( 0.5650000000000001 %)

---



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 ( 13.684000000000001 %)  
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 ( 1.1280000000000001 %)
Number of data points in class 8 : 3 ( 0.5640000000000001 %)

```

### 3.2 Prediction using a 'Random' Model

In a 'Random' Model, we generate the NINE class probabilities 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 column
    # C = [[1, 2],

```

```

#      [3, 4]
# C.T = [[1, 3],
#         [2, 4]]
# C.sum(axis = 1)  axis=0 corresponds 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 corresponds 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, yticklabels=labels)
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, yticklabels=labels)
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, yticklabels=labels)
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_predicted_y))

# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y, eps=0.01))

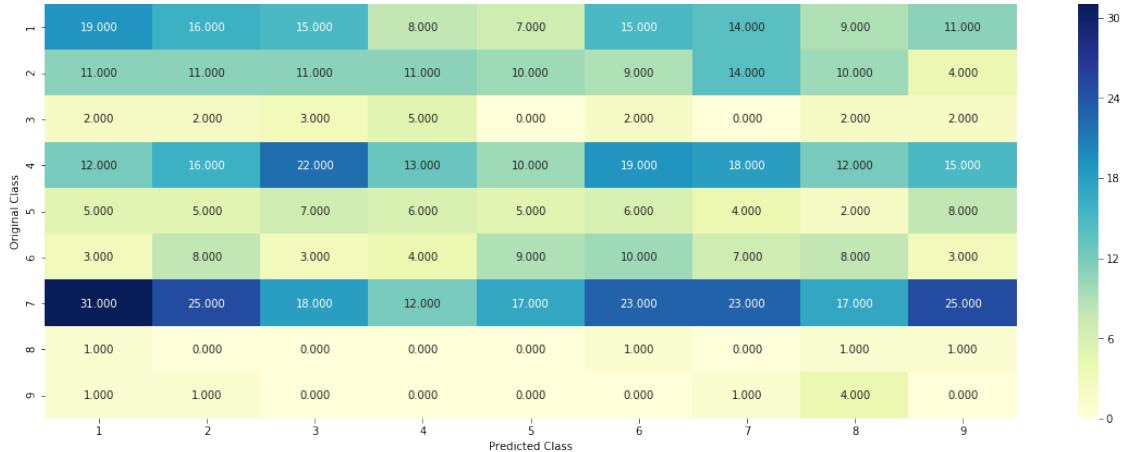
predicted_y =np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y+1)

```

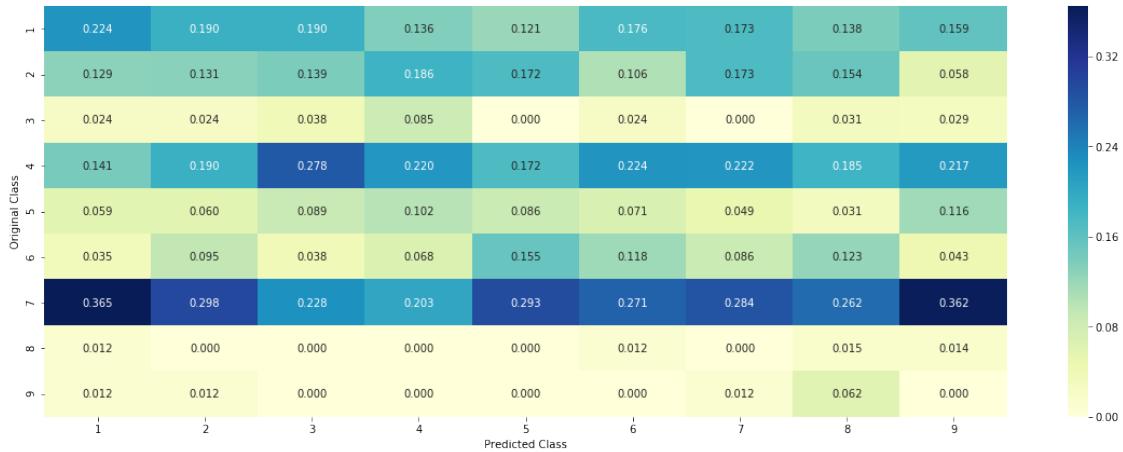
Log loss on Cross Validation Data using Random Model 2.4589935283876656

Log loss on Test Data using Random Model 2.408751654795735

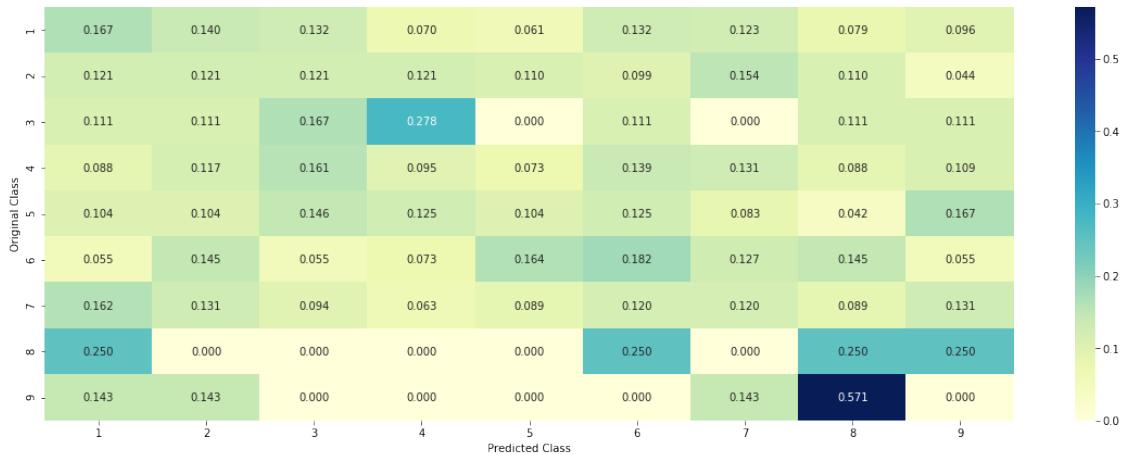
----- Confusion matrix -----



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



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



### 3.3 Univariate Analysis

```
In [12]: # 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 occurrences of given feature in train df
# build a vector (1*9) , the first element = (number of times it occurred 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] to 'gv_fea'
# return 'gv_fea'
# -------

# get_gv_fea_dict: Get Gene variation 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 gene
    gv_dict = dict()

    # denominator will contain the number of time that particular feature occurred in train
    for i, denominator in value_count.items():
        # vec will contain ( $p(y_i=1/G_i)$ ) probability of gene/variation belongs to perturb
        # vec is 9 dimensional 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': [0.2007575757575757, 0.03787878787878788, 0.0681818181818177,
    # 'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366,
    # 'EGFR': [0.0568181818181816, 0.21590909090909091, 0.0625, 0.06818181818181816,
    # 'BRCA2': [0.1333333333333333, 0.060606060606060608, 0.060606060606060608,
    # 'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917,
    # 'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.07333333333333334,
    # 'BRAF': [0.066666666666666666, 0.17999999999999999, 0.07333333333333334,
    # ...
    #     ]
    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()

    # gvfea: Gene_variation feature, it will contain the feature for each feature va
    gvfea = []
    # 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 gvfea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gvfea.append(gv_dict[row[feature]])
        else:
            gvfea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
            gvfea.append([-1,-1,-1,-1,-1,-1,-1,-1])
    return gvfea

```

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

$(\text{numerator} + 10 * \text{alpha}) / (\text{denominator} + 90 * \text{alpha})$

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 occurred most
    print(unique_genes.head(10))
```

Number of Unique Genes : 232

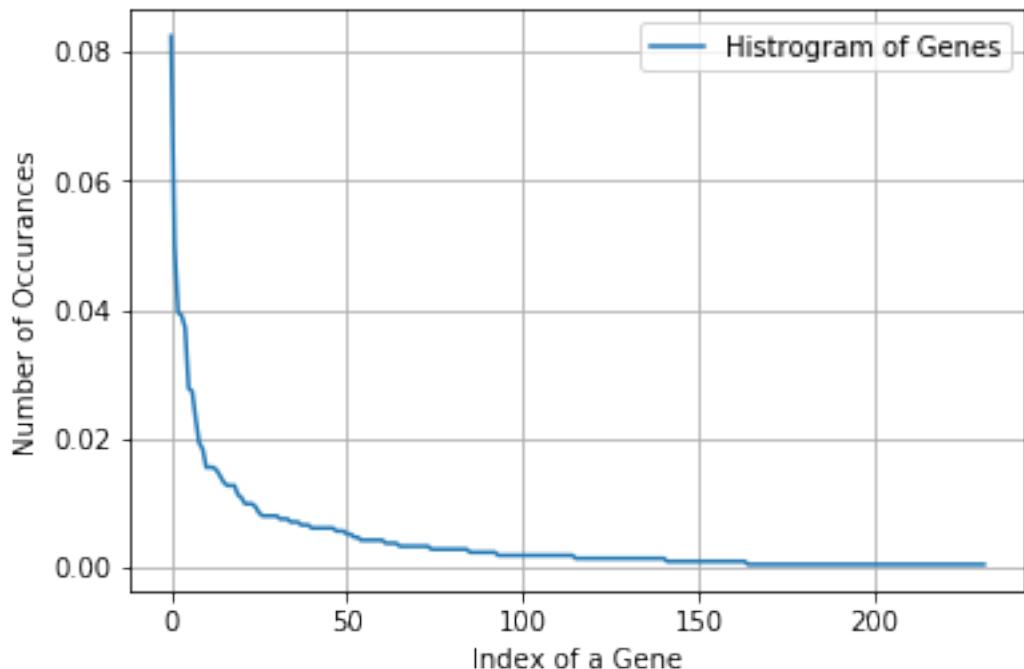
BRCA1	175
TP53	106
EGFR	84
PTEN	83
BRCA2	79
BRAF	59
KIT	58
ALK	49
PDGFRA	41
ERBB2	39

Name: Gene, dtype: int64

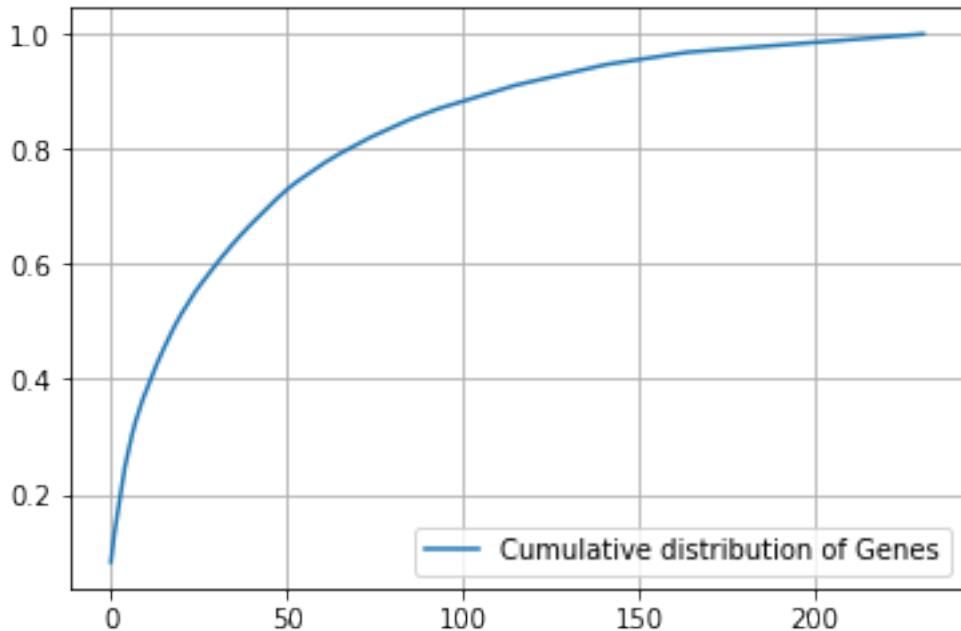
```
In [14]: print("Ans: There are", unique_genes.shape[0] , "different categories of genes in the train data")
```

Ans: There are 232 different categories of genes in the train data, and they are distributed as follows:

```
In [15]: s = sum(unique_genes.values);
    h = unique_genes.values/s;
    plt.plot(h, label="Histogram of Genes")
    plt.xlabel('Index of a Gene')
    plt.ylabel('Number of Occurrences')
    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 response coding method")  
train_gene_feature_responseCoding is converted feature using response coding method. The shape of
```

```
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]: 2545    BRCA1  
2164    PTEN  
1619    VHL  
1548    ALK  
1897    MTOR  
Name: Gene, dtype: object
```

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

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

'akt3',  
'alk',  
'apc',  
'ar',  
'araf',  
'arid1a',  
'arid1b',  
'arid2',  
'arid5b',  
'atm',  
'atr',  
'atrx',  
'aurka',  
'aurkb',  
'axin1',  
'axl',  
'b2m',  
'bap1',  
'bard1',  
'bcl10',  
'bcl2l11',  
'bcor',  
'braf',  
'brca1',  
'brca2',  
'brd4',  
'brip1',  
'btk',  
'card11',  
'carm1',  
'casp8',  
'cbl',  
'ccnd1',  
'ccnd2',  
'ccnd3',  
'ccne1',  
'cdh1',  
'cdk12',  
'cdk4',  
'cdk6',  
'cdk8',  
'cdkn1a',  
'cdkn1b',  
'cdkn2a',  
'cdkn2b',  
'cdkn2c',  
'chek2',  
'cic',

'crebbp',  
'ctcf',  
'ctnnb1',  
'ddr2',  
'dicer1',  
'dnmt3a',  
'dnmt3b',  
'egfr',  
'eif1ax',  
'elf3',  
'ep300',  
'epas1',  
'epcam',  
'erbb2',  
'erbb3',  
'erbb4',  
'ercc2',  
'ercc3',  
'ercc4',  
'erg',  
'errfi1',  
'esr1',  
'etv1',  
'etv6',  
'ewsr1',  
'ezh2',  
'fanca',  
'fancc',  
'fat1',  
'fbxw7',  
'fgf19',  
'fgfr1',  
'fgfr2',  
'fgfr3',  
'fgfr4',  
'flt1',  
'flt3',  
'foxa1',  
'foxl2',  
'foxp1',  
'fubp1',  
'gata3',  
'gli1',  
'gna11',  
'gnaq',  
'gnas',  
'h3f3a',  
'hla',

'hnf1a',  
'hras',  
'idh1',  
'idh2',  
'igf1r',  
'ikbke',  
'ikzf1',  
'jak1',  
'jak2',  
'jun',  
'kdm5a',  
'kdm5c',  
'kdm6a',  
'kdr',  
'keap1',  
'kit',  
'kmt2a',  
'kmt2c',  
'kmt2d',  
'knstrn',  
'kras',  
'lats1',  
'lats2',  
'map2k1',  
'map2k2',  
'map2k4',  
'map3k1',  
'mapk1',  
'mdm2',  
'med12',  
'mef2b',  
'men1',  
'met',  
'mlh1',  
'mpl',  
'msh2',  
'msh6',  
'mtor',  
'myc',  
'mycn',  
'myd88',  
'ncor1',  
'nf1',  
'nf2',  
'nfe212',  
'nkbia',  
'nkx2',  
'notch1',

```
'notch2',
'npm1',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'nup93',
'pak1',
'pax8',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pik3r3',
'pim1',
'pms2',
'pole',
'ppp2r1a',
'ppp6c',
'prdm1',
'ptch1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rab35',
'rac1',
'rad21',
'rad50',
'rad51b',
'rad51d',
'rad541',
'raf1',
'rasa1',
'rb1',
'rbm10',
'ret',
'rheb',
'rhoa',
'riktor',
'rit1',
'ros1',
'rras2',
```

```
'runx1',
'rxra',
'rybp',
'sdhb',
'sdhc',
'setd2',
'sf3b1',
'smad2',
'smad3',
'smad4',
'smarca4',
'smo',
'sos1',
'sox9',
'spop',
'src',
'srsf2',
'stat3',
'stk11',
'tert',
'tet1',
'tet2',
'tgfbr1',
'tgfbr2',
'tmprss2',
'tp53',
'tsc1',
'tsc2',
'u2af1',
'vegfa',
'vhl',
'whsc1',
'xpo1',
'xrcc2',
'yap1']
```

```
In [22]: print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method")
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of
```

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=True,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=optimal,
# 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 Gradient Descent
# 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, labels=clf.classes_, eps=1e-15))

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)

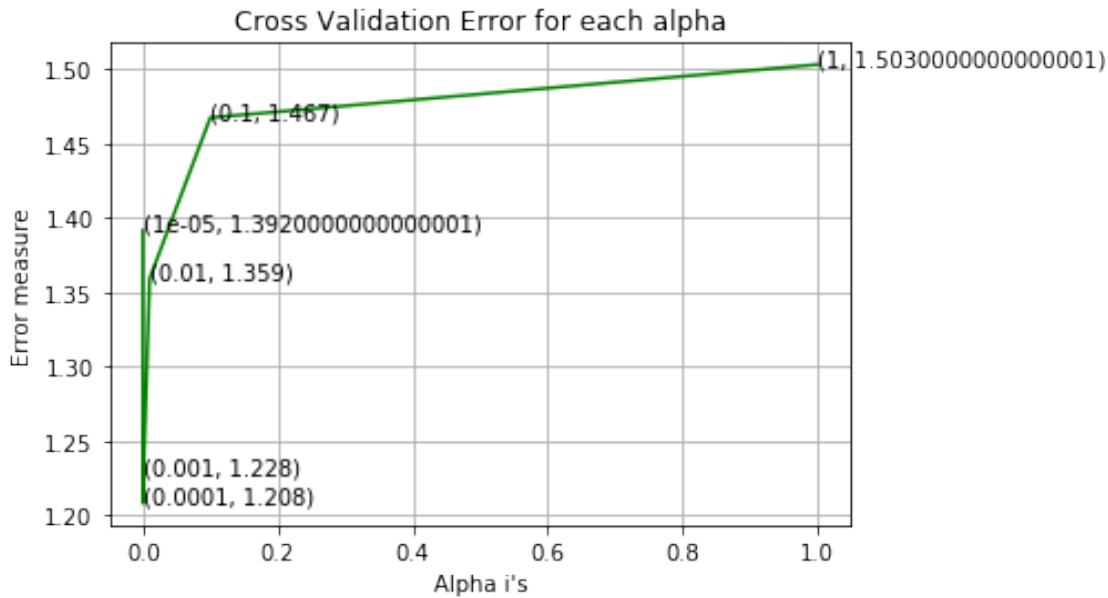
predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha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

```

```

For values of alpha = 1e-05 The log loss is: 1.391601657026069
For values of alpha = 0.0001 The log loss is: 1.208257521682012
For values of alpha = 0.001 The log loss is: 1.227676465298504
For values of alpha = 0.01 The log loss is: 1.3591445037227299
For values of alpha = 0.1 The log loss is: 1.4673045184099351
For values of alpha = 1 The log loss is: 1.5029652919991914

```



```

For values of best alpha = 0.0001 The train log loss is: 1.0398342608237008
For values of best alpha = 0.0001 The cross validation log loss is: 1.208257521682012
For values of best alpha = 0.0001 The test log loss is: 1.216176033523782

```

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_genes)

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_coverage/test_df.shape[0]*100))
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0], ":" ,(cv_coverage/cv_df.shape[0]*100))

```

Q6. How many data points in Test and CV datasets are covered by the 232 genes in train dataset  
Ans

1. In test data 647 out of 665 : 97.29323308270676
2. In cross validation data 514 out of 532 : 96.61654135338345

### 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 occurred most
    print(unique_variations.head(10))
```

Number of Unique Variations : 1912

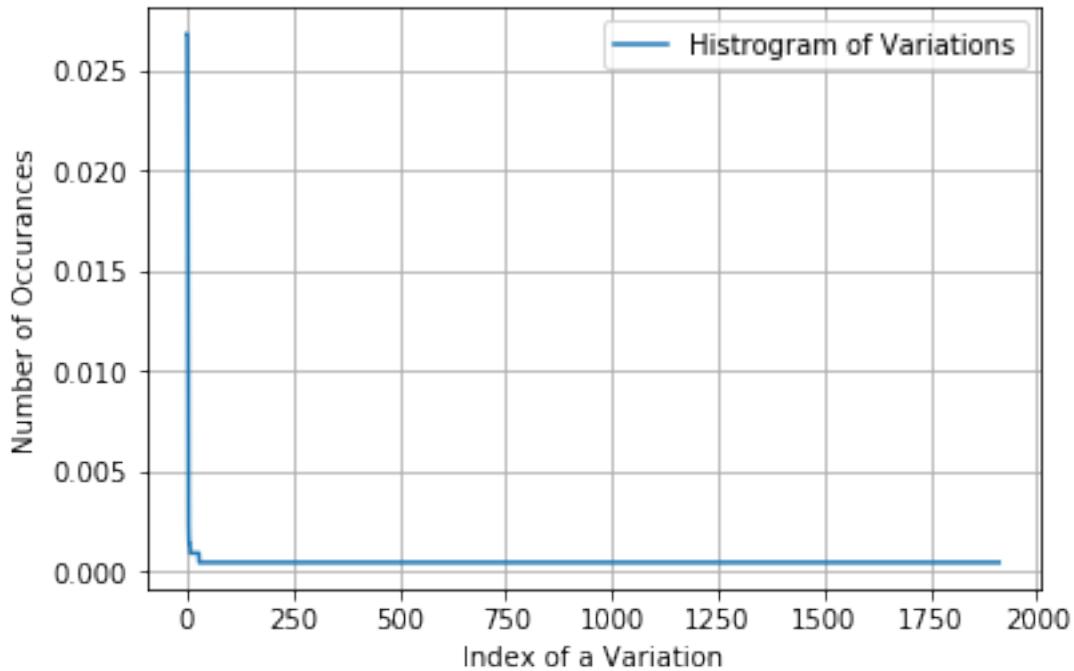
Amplification	57
Truncating_Mutations	57
Deletion	47
Fusions	24
Overexpression	5
E17K	3
Q61R	3
Q61L	3
P130S	2
T58I	2

Name: Variation, dtype: int64

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

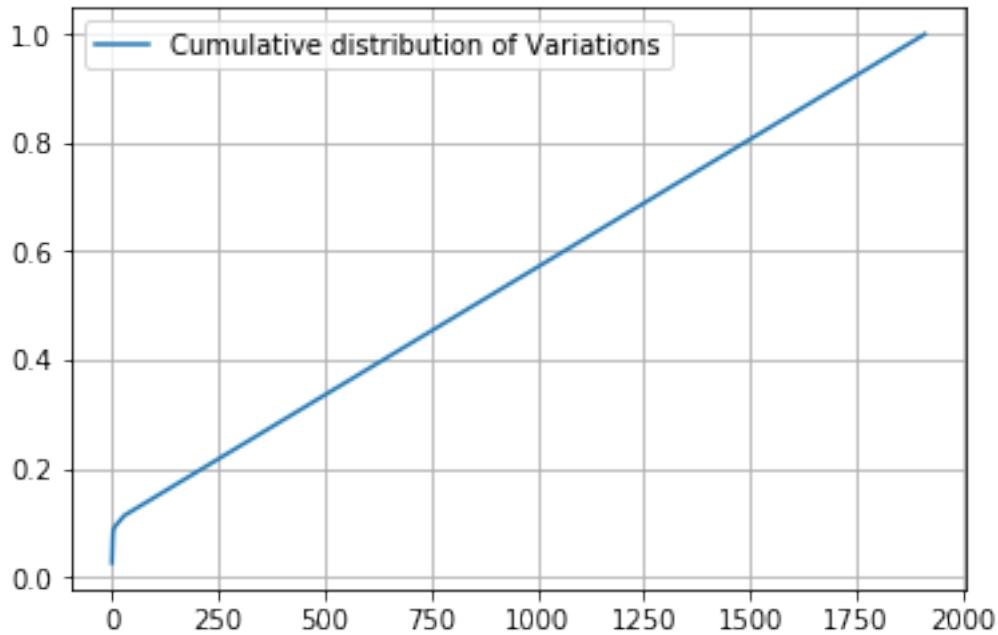
Ans: There are 1912 different categories of variations in the train data, and they are distributed as follows:

```
In [27]: s = sum(unique_variations.values);
    h = unique_variations.values/s;
    plt.plot(h, label="Histogram of Variations")
    plt.xlabel('Index of a Variation')
    plt.ylabel('Number of Occurrences')
    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()

[0.02683616 0.05367232 0.07580038 ... 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", "train"))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", "test"))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", "cv"))
```

```
In [30]: print("train_variation_feature_responseCoding is a converted feature using the response coding method")
```

```
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['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

```
In [32]: print("train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method.\n\ntrain_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/sklearn.linear_model.SGDClassifier.html#sklearn-linear-model-SGDClassifier
# -----
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=optimal,
# 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 Gradient Descent (SGD) algorithm.
# 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, labels=clf.classes_, eps=1e-15))

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()
```

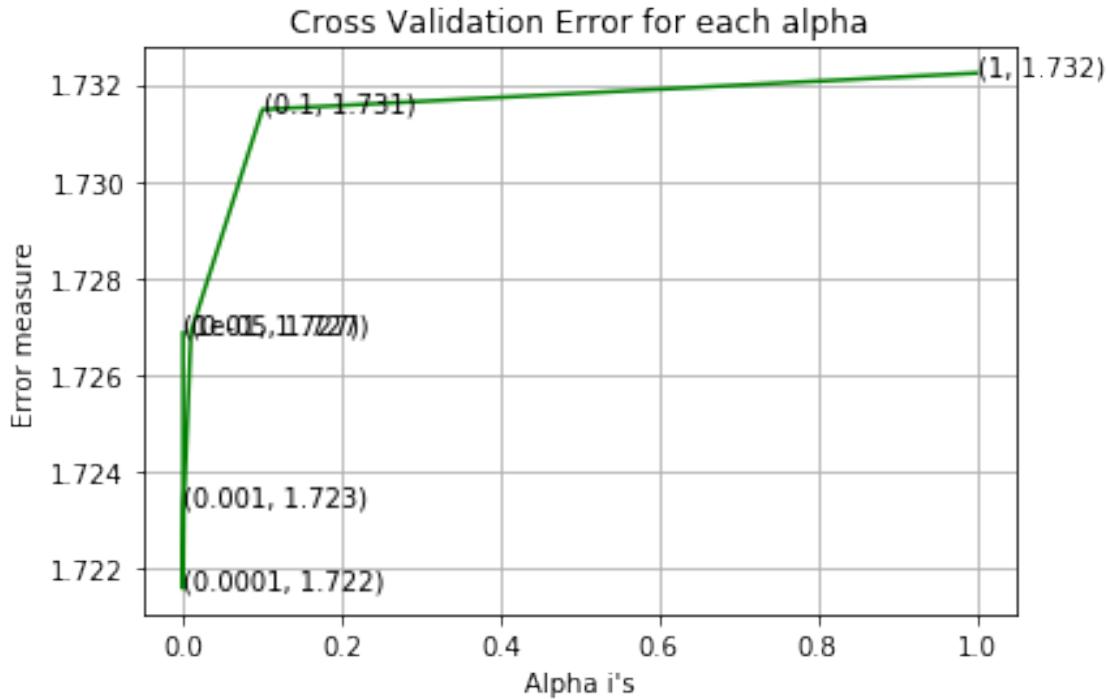
```

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_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_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(cv_y, predict_y))
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_loss(test_y, predict_y))

For values of alpha = 1e-05 The log loss is: 1.7268712251341676
For values of alpha = 0.0001 The log loss is: 1.7215993457036758
For values of alpha = 0.001 The log loss is: 1.723312823593727
For values of alpha = 0.01 The log loss is: 1.7268944599921006
For values of alpha = 0.1 The log loss is: 1.731479643555196
For values of alpha = 1 The log loss is: 1.7322297998051366

```



```

For values of best alpha = 0.0001 The train log loss is: 0.7771012818133722
For values of best alpha = 0.0001 The cross validation log loss is: 1.7215993457036758

```

```
For values of best alpha = 0.0001 The test log loss is: 1.7013910860284391
```

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'])))].shape[0]  
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_coverage/test_df.shape[0]))  
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0], ":" ,(cv_coverage/cv_df.shape[0]))
```

Q12. How many data points are covered by total 1912 genes in test and cross validation data ?

Ans

1. In test data 58 out of 665 : 8.721804511278195
2. In cross validation data 50 out of 532 : 9.398496240601503

### 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 predicting 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(word,0)+10)))  
            text_feature_responseCoding[row_index][i] = sum_prob  
        row_index += 1
```

```

        text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEXT']))
        row_index += 1
    return text_feature_responseCoding

```

```

In [37]: # building a CountVectorizer with all the words that occurred minimum 3 times in train
text_vectorizer = CountVectorizer(min_df=3)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of words)
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it occurred
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))

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

```

Total number of unique words in train data : 53678

```

In [38]: dict_list = []
# dict_list =[] contains 9 dictionaries 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 build on whole training text data
total_dict = extract_dictionary_paddle(train_df)

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

```

```

In [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)

```



```

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

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_text_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))

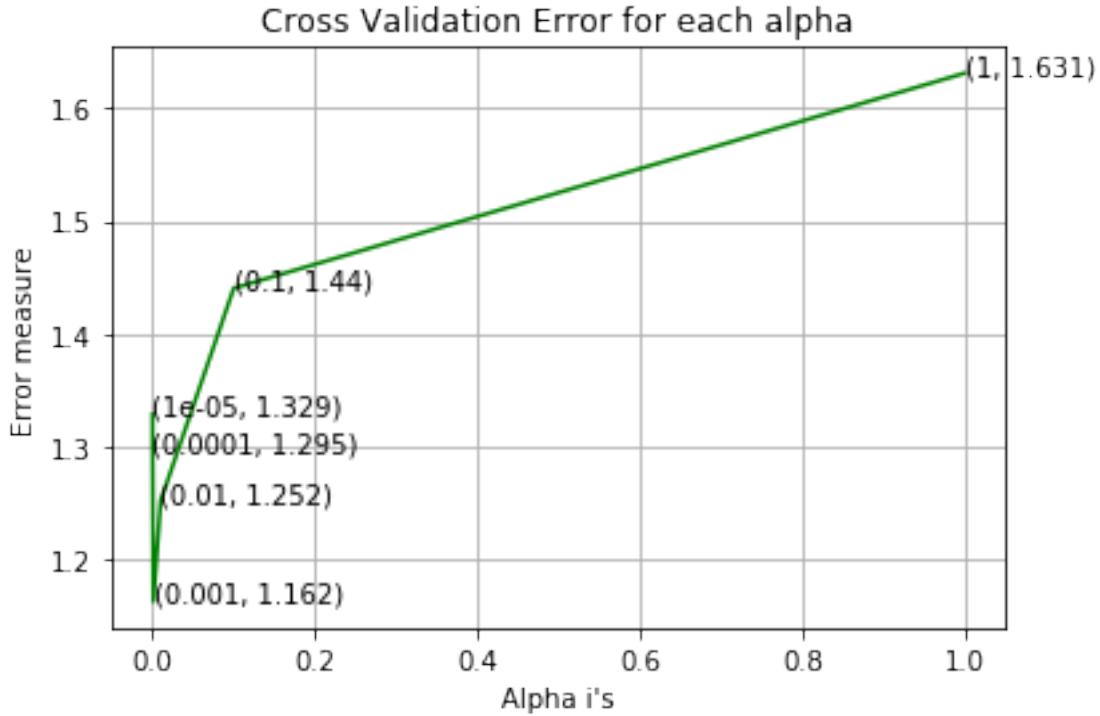
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error 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_text_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_text_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(test_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y))

For values of alpha =  1e-05 The log loss is: 1.3285660324157205
For values of alpha =  0.0001 The log loss is: 1.2953735194322111
For values of alpha =  0.001 The log loss is: 1.162480241128727
For values of alpha =  0.01 The log loss is: 1.2518603772630041
For values of alpha =  0.1 The log loss is: 1.4403977247408821
For values of alpha =  1 The log loss is: 1.6311556400265939

```



For values of best alpha = 0.001 The train log loss is: 0.7672970313577607

For values of best alpha = 0.001 The cross validation log loss is: 1.162480241128727

For values of best alpha = 0.001 The test log loss is: 1.1752238561382986

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):
    df_text_vec = CountVectorizer(min_df=3)
    df_textfea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

    df_textfea_counts = df_textfea.sum(axis=0).A1
    df_textfea_dict = dict(zip(list(df_text_features),df_textfea_counts))
    len1 = len(set(df_text_features))
    len2 = len(set(train_text_features) & set(df_text_features))
    return len1,len2

In [46]: len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train data")
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in train data")
```

```
95.371 % of word of test data appeared in train data  
96.68900000000001 % of word of Cross Validation appeared in train data
```

#### 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 will 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))/test_y)  
    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\_imfeature\_names(indices, text, gene, var, no\_features):  
 gene\_count\_vec = CountVectorizer()  
 var\_count\_vec = CountVectorizer()  
 text\_count\_vec = CountVectorizer(min\_df=3)  
  
 gene\_vec = gene\_count\_vec.fit(train\_df['Gene'])  
 var\_vec = var\_count\_vec.fit(train\_df['Variation'])  
 text\_vec = text\_count\_vec.fit(train\_df['TEXT'])  
  
 fea1\_len = len(gene\_vec.get\_feature\_names())  
 fea2\_len = len(var\_count\_vec.get\_feature\_names())  
  
 word\_present = 0  
 for i,v in enumerate(indices):  
 if (v < fea1\_len):  
 word = gene\_vec.get\_feature\_names()[v]

```

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

```

Stacking the three types of features

In [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_onehotCod
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']))


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_f

```

```
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_responseC
```

```
In [51]: print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCo
print("(number of data points * number of features) in test data = ", test_x_onehotCo
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, 55853)
(number of data points * number of features) in test data = (665, 55853)
(number of data points * number of features) in cross validation data = (532, 55853)
```

```
In [52]: print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_respon
print("(number of data points * number of features) in test data = ", test_x_respons
print("(number of data points * number of features) in cross validation data =", cv_x_
```

```
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)
```

## 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/modules
# -----
# 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/
# -----



alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(predict_y,train_y))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(predict_y,cv_y))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)

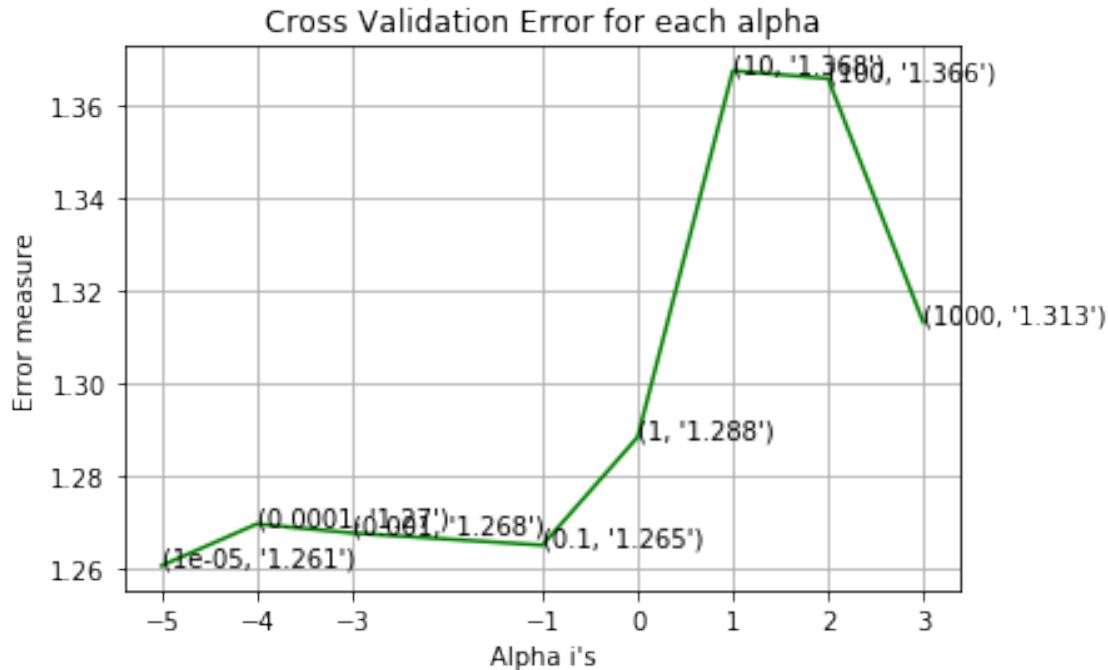
```

```

print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_10)

for alpha = 1e-05
Log Loss : 1.2606074918336592
for alpha = 0.0001
Log Loss : 1.2696090517382133
for alpha = 0.001
Log Loss : 1.2676585637457518
for alpha = 0.1
Log Loss : 1.2650263091506022
for alpha = 1
Log Loss : 1.2883884387808155
for alpha = 10
Log Loss : 1.3676091828992478
for alpha = 100
Log Loss : 1.3659261746319138
for alpha = 1000
Log Loss : 1.3132606535431315

```



For values of best alpha = 1e-05 The train log loss is: 0.8542605119654857  
 For values of best alpha = 1e-05 The cross validation log loss is: 1.2606074918336592  
 For values of best alpha = 1e-05 The test log loss is: 1.2994197242643335

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

```
In [60]: # 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/
# -----

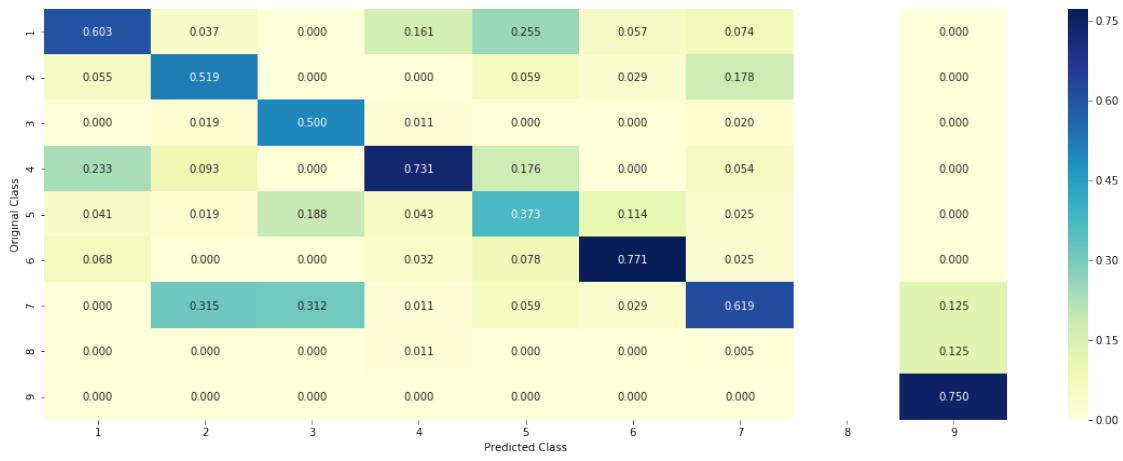

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/
# -----
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=5)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])      Fit the calibrated model
# get_params([deep])            Get parameters for this estimator.
# predict(X)          Predict the target of new samples.
# predict_proba(X)        Posterior probabilities of classification
# -----


clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilitites we use log-probability estimation
print("Log Loss :", log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotCoding) != cv_y)))
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

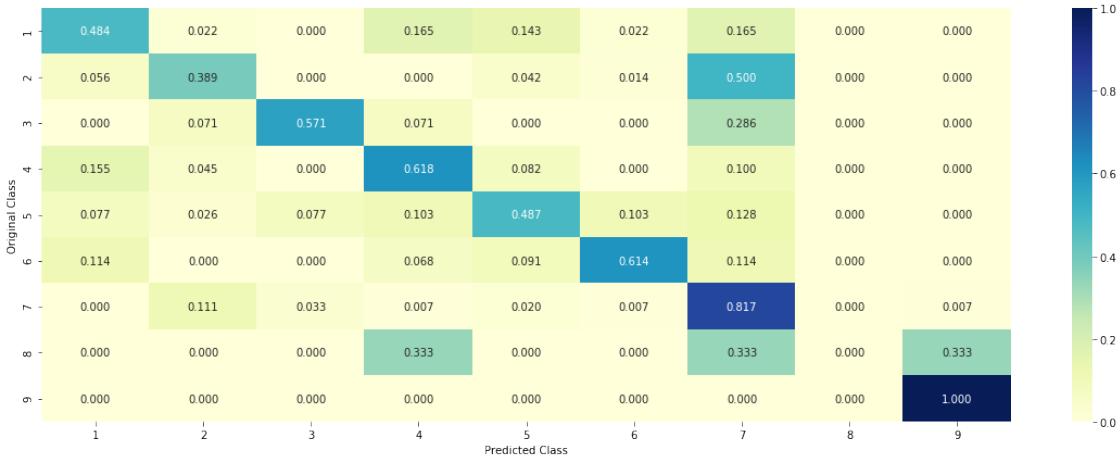
Log Loss : 1.2606074918336592  
Number of missclassified point : 0.3890977443609023  
----- Confusion matrix -----



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



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



#### 4.1.1.3. Feature Importance, Correctly classified point

```
In [61]: test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding)[test_point_index]))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imptfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'])
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0947 0.0922 0.0119 0.1126 0.0395 0.0401 0.5982 0.0052 0.0052 0.0052]
Actual Class : 2
-----
17 Text feature [kinase] present in test data point [True]
18 Text feature [presence] present in test data point [True]
20 Text feature [well] present in test data point [True]
24 Text feature [independent] present in test data point [True]
25 Text feature [inhibitor] present in test data point [True]
26 Text feature [downstream] present in test data point [True]
27 Text feature [cell] present in test data point [True]
29 Text feature [contrast] present in test data point [True]
30 Text feature [potential] present in test data point [True]
31 Text feature [previously] present in test data point [True]
32 Text feature [compared] present in test data point [True]
33 Text feature [recently] present in test data point [True]
34 Text feature [growth] present in test data point [True]
35 Text feature [higher] present in test data point [True]
37 Text feature [factor] present in test data point [True]
38 Text feature [cells] present in test data point [True]
```

```
39 Text feature [also] present in test data point [True]
40 Text feature [shown] present in test data point [True]
41 Text feature [10] present in test data point [True]
42 Text feature [obtained] present in test data point [True]
43 Text feature [however] present in test data point [True]
44 Text feature [found] present in test data point [True]
45 Text feature [activation] present in test data point [True]
46 Text feature [suggest] present in test data point [True]
47 Text feature [similar] present in test data point [True]
48 Text feature [treated] present in test data point [True]
49 Text feature [addition] present in test data point [True]
50 Text feature [may] present in test data point [True]
51 Text feature [described] present in test data point [True]
52 Text feature [mutations] present in test data point [True]
53 Text feature [studies] present in test data point [True]
54 Text feature [inhibition] present in test data point [True]
55 Text feature [proliferation] present in test data point [True]
57 Text feature [observed] present in test data point [True]
58 Text feature [new] present in test data point [True]
60 Text feature [using] present in test data point [True]
62 Text feature [different] present in test data point [True]
63 Text feature [followed] present in test data point [True]
64 Text feature [including] present in test data point [True]
66 Text feature [inhibitors] present in test data point [True]
72 Text feature [mutation] present in test data point [True]
74 Text feature [report] present in test data point [True]
75 Text feature [enhanced] present in test data point [True]
76 Text feature [increased] present in test data point [True]
78 Text feature [molecular] present in test data point [True]
79 Text feature [figure] present in test data point [True]
80 Text feature [interestingly] present in test data point [True]
82 Text feature [approximately] present in test data point [True]
83 Text feature [reported] present in test data point [True]
84 Text feature [identified] present in test data point [True]
86 Text feature [either] present in test data point [True]
87 Text feature [phosphorylation] present in test data point [True]
88 Text feature [small] present in test data point [True]
90 Text feature [recent] present in test data point [True]
91 Text feature [consistent] present in test data point [True]
92 Text feature [signaling] present in test data point [True]
95 Text feature [suggests] present in test data point [True]
96 Text feature [increase] present in test data point [True]
99 Text feature [revealed] present in test data point [True]
Out of the top 100 features 59 are present in query point
```

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

In [62]: `test_point_index = 100`

```

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

```

Predicted Class : 1  
Predicted Class Probabilities: [[0.5846 0.094 0.0122 0.1147 0.0406 0.0408 0.1021 0.0053 0.0053 0.0053]  
Actual Class : 1

---

11 Text feature [dna] present in test data point [True]  
12 Text feature [affect] present in test data point [True]  
13 Text feature [type] present in test data point [True]  
14 Text feature [one] present in test data point [True]  
15 Text feature [protein] present in test data point [True]  
16 Text feature [wild] present in test data point [True]  
17 Text feature [function] present in test data point [True]  
18 Text feature [two] present in test data point [True]  
19 Text feature [binding] present in test data point [True]  
20 Text feature [sequence] present in test data point [True]  
21 Text feature [four] present in test data point [True]  
23 Text feature [amino] present in test data point [True]  
27 Text feature [reduced] present in test data point [True]  
28 Text feature [containing] present in test data point [True]  
32 Text feature [region] present in test data point [True]  
34 Text feature [possible] present in test data point [True]  
35 Text feature [form] present in test data point [True]  
36 Text feature [determined] present in test data point [True]  
37 Text feature [loss] present in test data point [True]  
38 Text feature [large] present in test data point [True]  
39 Text feature [data] present in test data point [True]  
40 Text feature [least] present in test data point [True]  
41 Text feature [analysis] present in test data point [True]  
42 Text feature [three] present in test data point [True]  
43 Text feature [specific] present in test data point [True]  
44 Text feature [used] present in test data point [True]  
48 Text feature [ability] present in test data point [True]  
49 Text feature [identified] present in test data point [True]  
52 Text feature [using] present in test data point [True]  
53 Text feature [results] present in test data point [True]  
54 Text feature [expected] present in test data point [True]  
55 Text feature [therefore] present in test data point [True]  
56 Text feature [surface] present in test data point [True]  
58 Text feature [nonsense] present in test data point [True]  
59 Text feature [changes] present in test data point [True]

```

61 Text feature [previous] present in test data point [True]
62 Text feature [indicate] present in test data point [True]
63 Text feature [indicated] present in test data point [True]
64 Text feature [likely] present in test data point [True]
65 Text feature [functions] present in test data point [True]
66 Text feature [located] present in test data point [True]
67 Text feature [result] present in test data point [True]
68 Text feature [important] present in test data point [True]
69 Text feature [conserved] present in test data point [True]
70 Text feature [genes] present in test data point [True]
71 Text feature [contains] present in test data point [True]
72 Text feature [structure] present in test data point [True]
74 Text feature [discussion] present in test data point [True]
81 Text feature [gene] present in test data point [True]
82 Text feature [full] present in test data point [True]
84 Text feature [proteins] present in test data point [True]
86 Text feature [similar] present in test data point [True]
87 Text feature [also] present in test data point [True]
89 Text feature [control] present in test data point [True]
91 Text feature [within] present in test data point [True]
93 Text feature [whether] present in test data point [True]
95 Text feature [significant] present in test data point [True]
96 Text feature [essential] present in test data point [True]
98 Text feature [addition] present in test data point [True]
Out of the top 100 features 59 are present in query point

```

## 4.2. K Nearest Neighbour Classification

### 4.2.1. Hyper parameter tuning

```

In [63]: # 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=softmax, cv=5)
#
# 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=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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

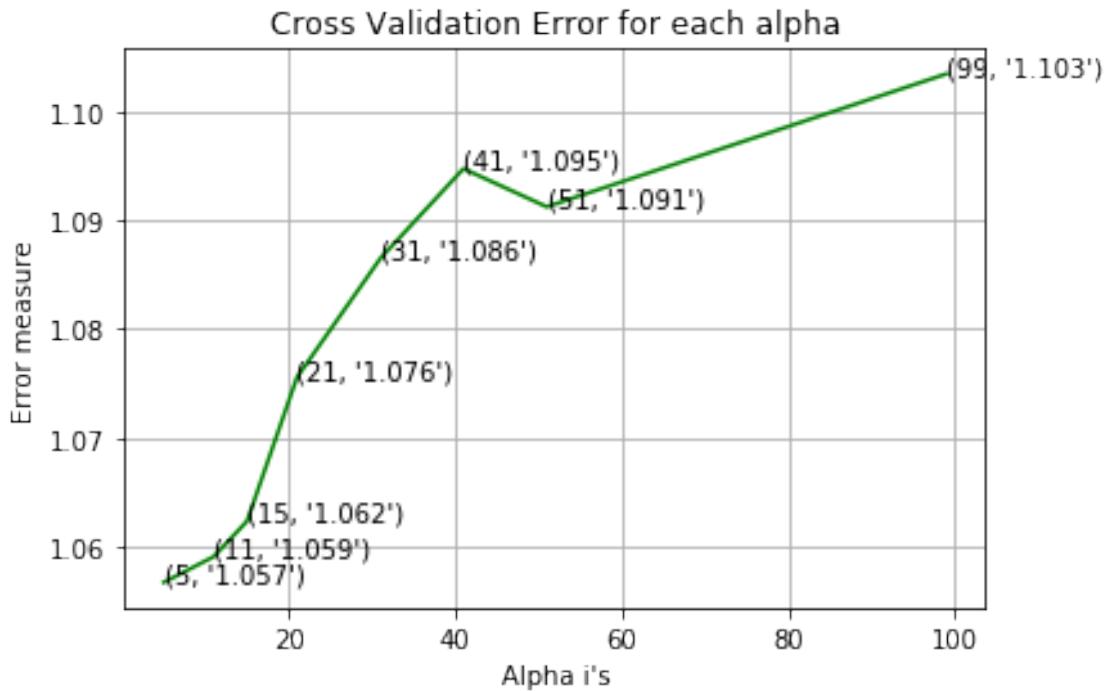
predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_
predict_y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

```

```

for alpha = 5
Log Loss : 1.0567050721490958
for alpha = 11
Log Loss : 1.0590652583653688
for alpha = 15
Log Loss : 1.0623316913964689
for alpha = 21
Log Loss : 1.0755511561710451
for alpha = 31
Log Loss : 1.0864695370873942
for alpha = 41
Log Loss : 1.0947332280130773
for alpha = 51
Log Loss : 1.0912299467166293
for alpha = 99
Log Loss : 1.103444416152279

```



For values of best alpha = 5 The train log loss is: 0.49434789589556366  
 For values of best alpha = 5 The cross validation log loss is: 1.0567050721490958  
 For values of best alpha = 5 The test log loss is: 1.0936065617839725

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

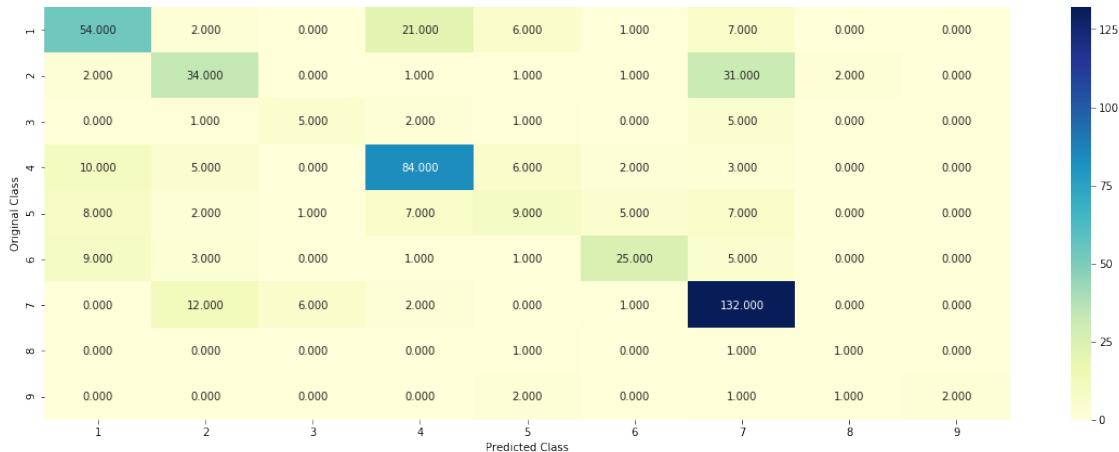
```
In [64]: # find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/
# -----
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights=uniform, algorithm=auto, leaf_size=30,
# metric=minkowski, metric_params=None, n_jobs=1, **kwargs)

# 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_responseCoding)
```

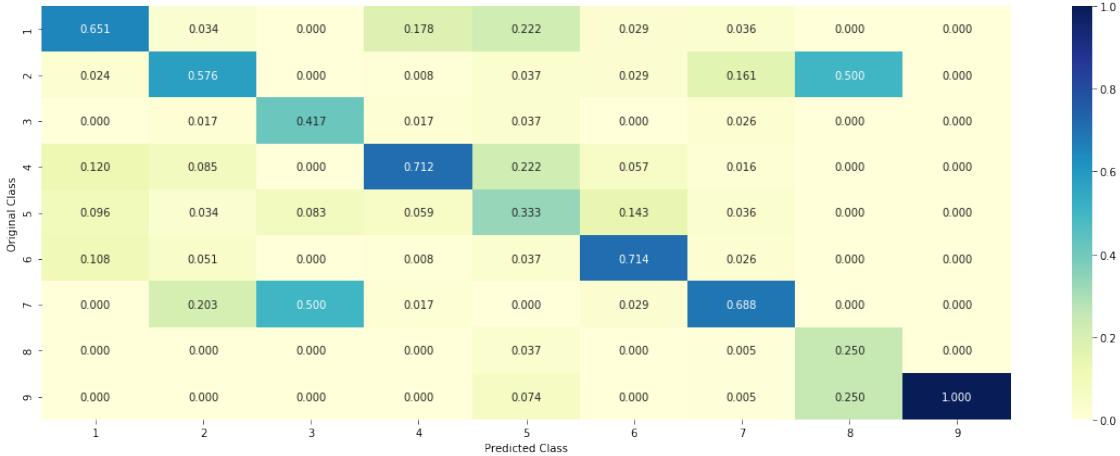
Log loss : 1.0567050721490958

Number of mis-classified points : 0.34962406015037595

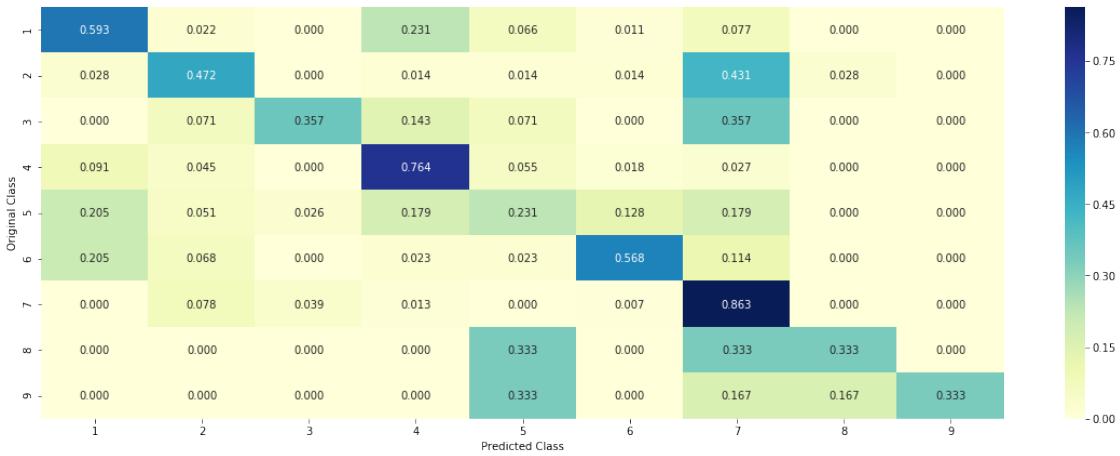
----- Confusion matrix -----



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



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



#### 4.2.3.Sample Query point -1

```
In [65]: 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), alpha[best_alpha])
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to class", neighbors[1][0])
print("Frequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```

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

```

#### 4.2.4. Sample Query Point-2

```

In [66]: 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 t
print("Frequency of nearest points :",Counter(train_y[neighbors[1][0]]))

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

```

### 4.3. Logistic Regression

#### 4.3.1. With Class balancing

##### 4.3.1.1. Hyper parameter tuning

```

In [67]: # 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=True,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=optimal,
# 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 Gradient Descent
# 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/modules/cv_validation.html
```

```

# -----
# default parameters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=softmax, cv=5)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])      Fit the calibrated model
# get_params([deep])            Get parameters for this estimator.
# predict(X)                  Predict the target of new samples.
# predict_proba(X)            Posterior probabilities of classification
#-----
# video link:
#-----

alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilités we use log-probability encoding
    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', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(predict_y,train_y))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(predict_y,cv_y))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)

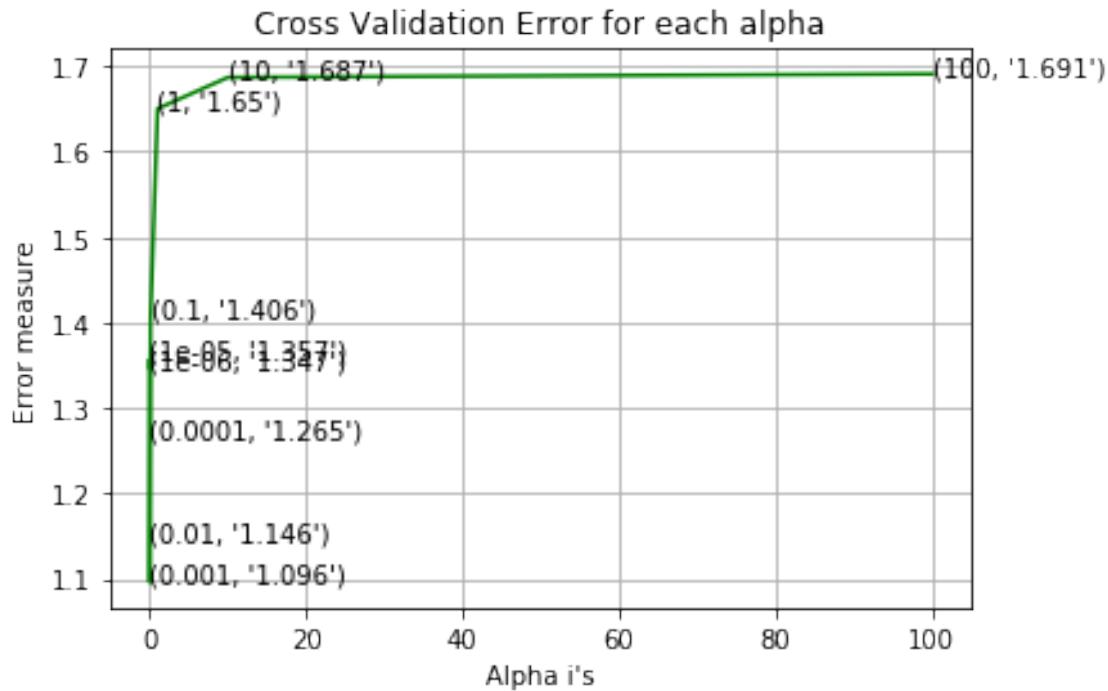
```

```

print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_10)

for alpha = 1e-06
Log Loss : 1.3470176427704843
for alpha = 1e-05
Log Loss : 1.3572677835783358
for alpha = 0.0001
Log Loss : 1.2647929304771672
for alpha = 0.001
Log Loss : 1.0959314791210701
for alpha = 0.01
Log Loss : 1.1456477165450882
for alpha = 0.1
Log Loss : 1.4063469458203461
for alpha = 1
Log Loss : 1.6501993896073293
for alpha = 10
Log Loss : 1.6868767776913007
for alpha = 100
Log Loss : 1.6908868695459032

```



For values of best alpha = 0.001 The train log loss is: 0.6325275085047739  
 For values of best alpha = 0.001 The cross validation log loss is: 1.0959314791210701  
 For values of best alpha = 0.001 The test log loss is: 1.0934430916012106

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

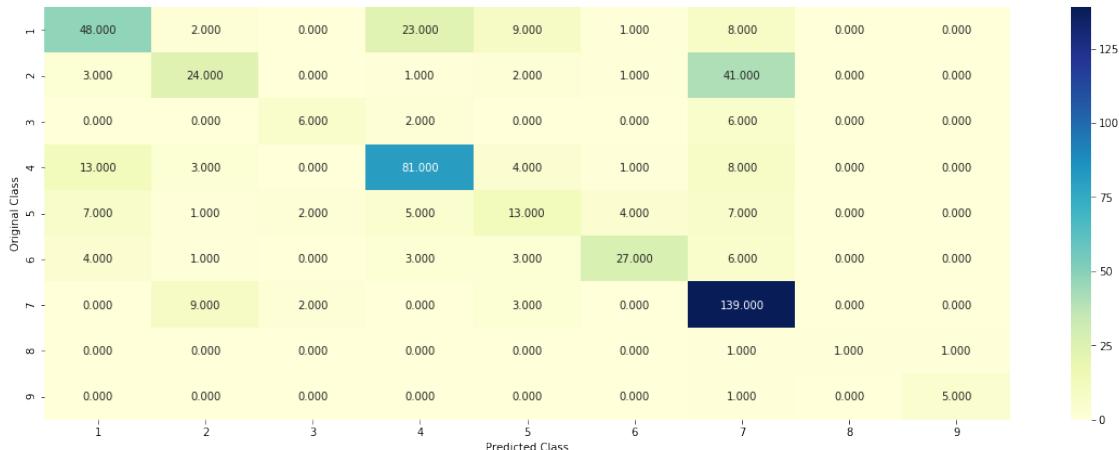
```
In [68]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
# -----
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=optimal,
# 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 Gradient Descent (SGD) algorithm.
# predict(X)             Predict class labels for samples in X.

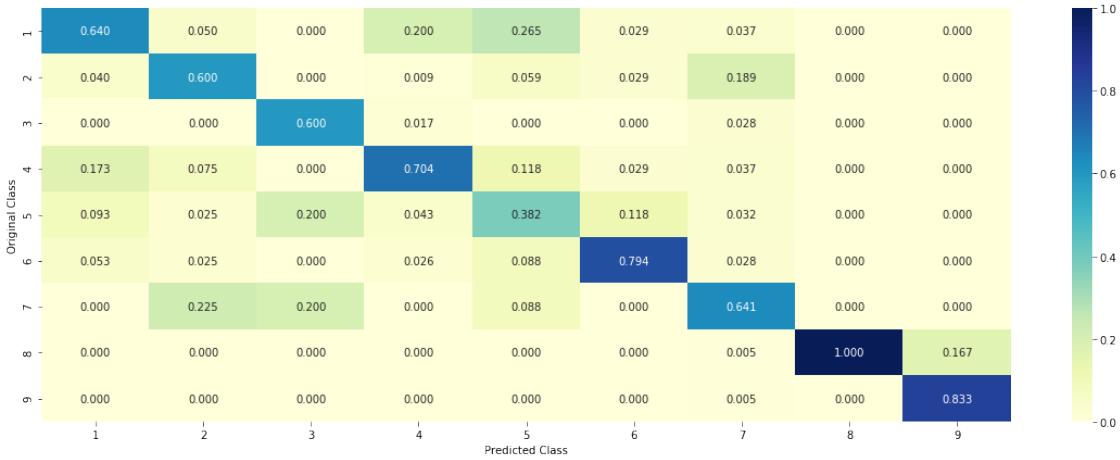
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/
# -----

clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge',
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y)
```

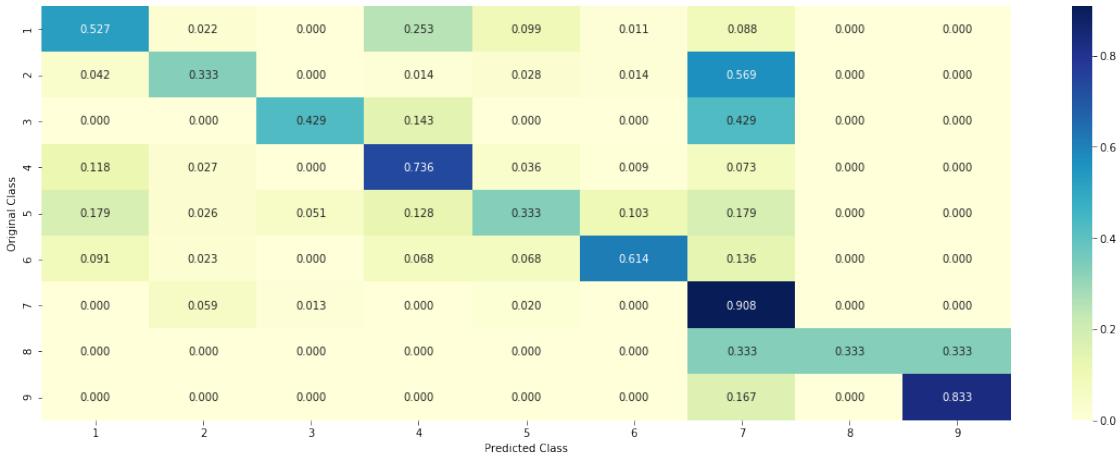
Log loss : 1.0959314791210701  
Number of mis-classified points : 0.3533834586466165  
----- Confusion matrix -----



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



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



#### 4.3.1.3. Feature Importance

```
In [69]: def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
    if ((i > 17) & (i not in removed_ind)) :
        word = train_text_features[i]
```

```

yes_no = True if word in text.split() else False
if yes_no:
    word_present += 1
tabulte_list.append([incresingorder_ind,train_text_features[i], yes_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

#### 4.3.1.3.2. Incorrectly Classified point

```
In [71]: test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding)[test_point_index]))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imptfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['GeneID'])

Predicted Class : 1
Predicted Class Probabilities: [[0.8613 0.0389 0.0097 0.0218 0.0205 0.0103 0.0275 0.0049 0.005
Actual Class : 1
-----
47 Text feature [rptp] present in test data point [True]
185 Text feature [intercellular] present in test data point [True]
202 Text feature [homophilic] present in test data point [True]
218 Text feature [neutravidin] present in test data point [True]
229 Text feature [rptps] present in test data point [True]
233 Text feature [sk18] present in test data point [True]
292 Text feature [pcp2] present in test data point [True]
362 Text feature [aggregation] present in test data point [True]
396 Text feature [frameshifts] present in test data point [True]
421 Text feature [mam] present in test data point [True]
498 Text feature [nonsense] present in test data point [True]
Out of the top 500 features 11 are present in query point
```

### 4.3.2. Without Class balancing

#### 4.3.2.1. Hyper parameter tuning

```
In [72]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal',
# 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 Gradient Descent.
# predict(X)          Predict class labels for samples in X.

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

```

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules
# -----
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=softmax, cv=5)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])      Fit the calibrated model
# get_params([deep])            Get parameters for this estimator.
# predict(X)                  Predict the target of new samples.
# predict_proba(X)             Posterior probabilities of classification
# -----
# video link:
#-----

alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    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=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

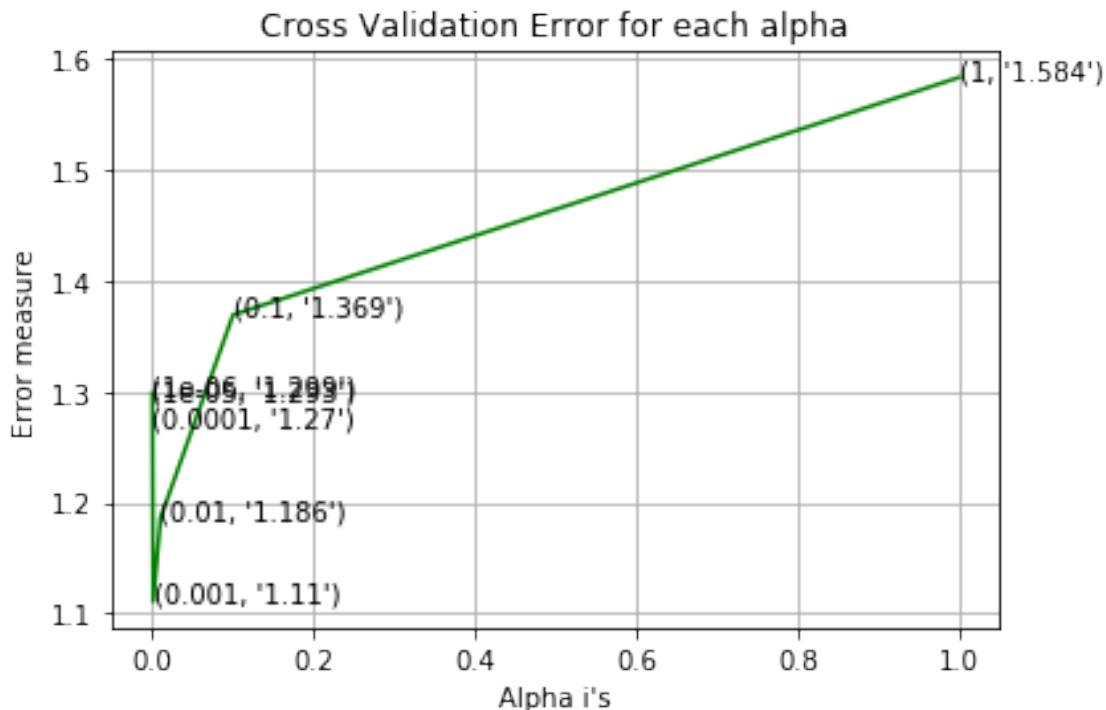
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(predict_y, train_y))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(predict_y, cv_y))

```

```

predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_l
for alpha = 1e-06
Log Loss : 1.29870816298548
for alpha = 1e-05
Log Loss : 1.2930326929598799
for alpha = 0.0001
Log Loss : 1.2695591067881227
for alpha = 0.001
Log Loss : 1.1104493547114254
for alpha = 0.01
Log Loss : 1.185733299990413
for alpha = 0.1
Log Loss : 1.3693690775752794
for alpha = 1
Log Loss : 1.583748636548569

```



For values of best alpha = 0.001 The train log loss is: 0.6336785756924349  
 For values of best alpha = 0.001 The cross validation log loss is: 1.1104493547114254  
 For values of best alpha = 0.001 The test log loss is: 1.108364763910898

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

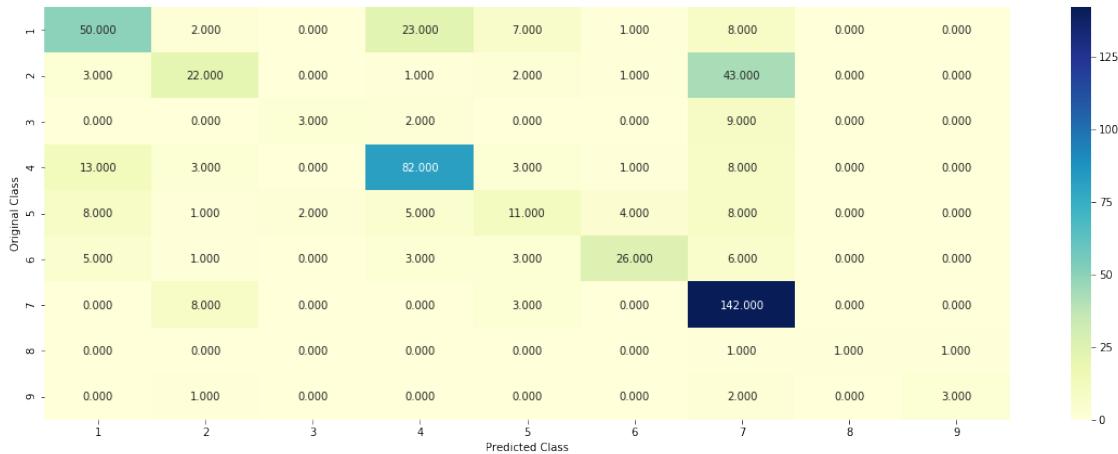
```
In [73]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier
# -----
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
# shuffle=True,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

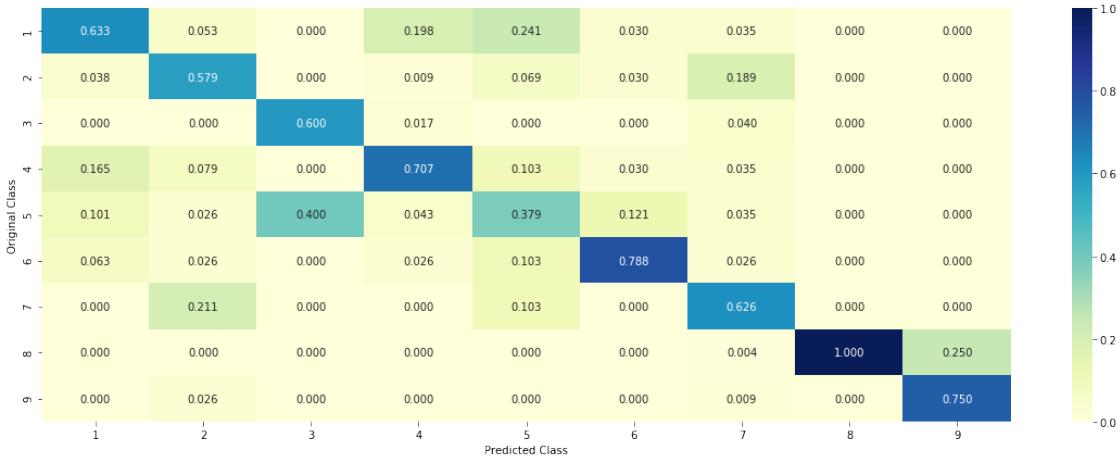
# -----
# video link:
# -----
```

clf = SGDClassifier(alpha=alpha[best\_alpha], penalty='l2', loss='log', random\_state=42)
predict\_and\_plot\_confusion\_matrix(train\_x\_onehotCoding, train\_y, cv\_x\_onehotCoding, cv\_y)

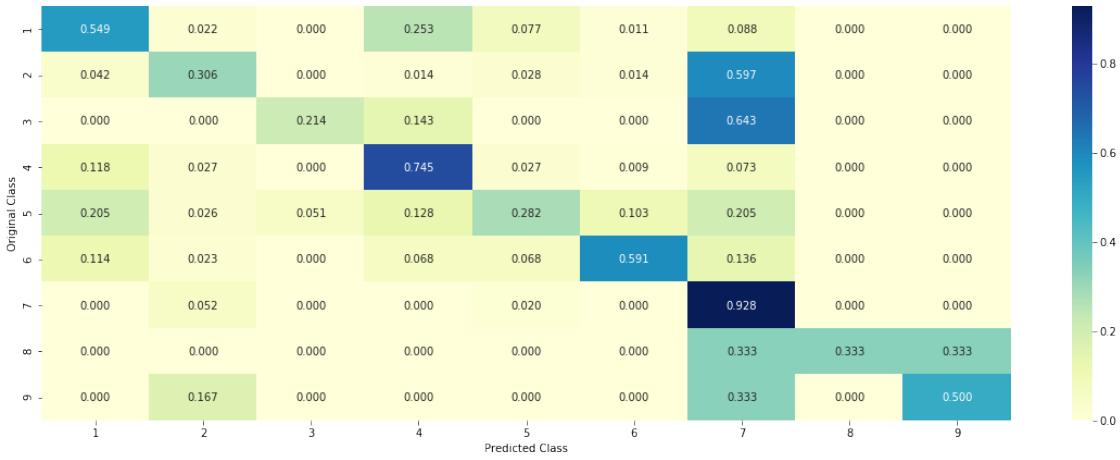
Log loss : 1.1104493547114254  
Number of mis-classified points : 0.3609022556390977  
----- Confusion matrix -----



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



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



#### 4.3.2.3. Feature Importance, Correctly Classified point

```
In [74]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index])[0]))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'])
```

```

Predicted Class : 7
Predicted Class Probabilities: [[0.0628 0.1439 0.0103 0.06   0.0378 0.0249 0.6507 0.0055 0.0041
Actual Class : 2
-----
162 Text feature [ligand] present in test data point [True]
201 Text feature [receptors] present in test data point [True]
254 Text feature [transformation] present in test data point [True]
285 Text feature [technology] present in test data point [True]
359 Text feature [receptor] present in test data point [True]
369 Text feature [hyperplasia] present in test data point [True]
424 Text feature [downstream] present in test data point [True]
478 Text feature [mapk] present in test data point [True]
492 Text feature [signaling] present in test data point [True]
Out of the top 500 features 9 are present in query point

```

#### 4.3.2.4. Feature Importance, Inorrectly Classified point

```

In [75]: test_point_index = 100
          no_feature = 500
          predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
          print("Predicted Class :", predicted_cls[0])
          print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding)[test_point_index]))
          print("Actual Class :", test_y[test_point_index])
          indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
          print("-"*50)
          get_imptfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
Predicted Class : 1
Predicted Class Probabilities: [[0.8577 0.043  0.0055 0.0235 0.0193 0.0093 0.0357 0.0041 0.0018
Actual Class : 1
-----
55 Text feature [rptp] present in test data point [True]
199 Text feature [intercellular] present in test data point [True]
254 Text feature [homophilic] present in test data point [True]
268 Text feature [rptps] present in test data point [True]
279 Text feature [neutravidin] present in test data point [True]
290 Text feature [sk18] present in test data point [True]
359 Text feature [pcp2] present in test data point [True]
445 Text feature [aggregation] present in test data point [True]
478 Text feature [nonsense] present in test data point [True]
Out of the top 500 features 9 are present in query point

```

## 4.4. Linear Support Vector Machines

### 4.4.1. Hyper parameter tuning

```
In [76]: # read more about support vector machines with linear kernels here http://scikit-learn.org/stable/
```

```

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

# Some of methods of SVM()
# fit(X, y, [sample_weight])      Fit the SVM model according to the given training data.
# 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/modules/
# -----
# default parameters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=5)
#
# 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_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    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")
```

```

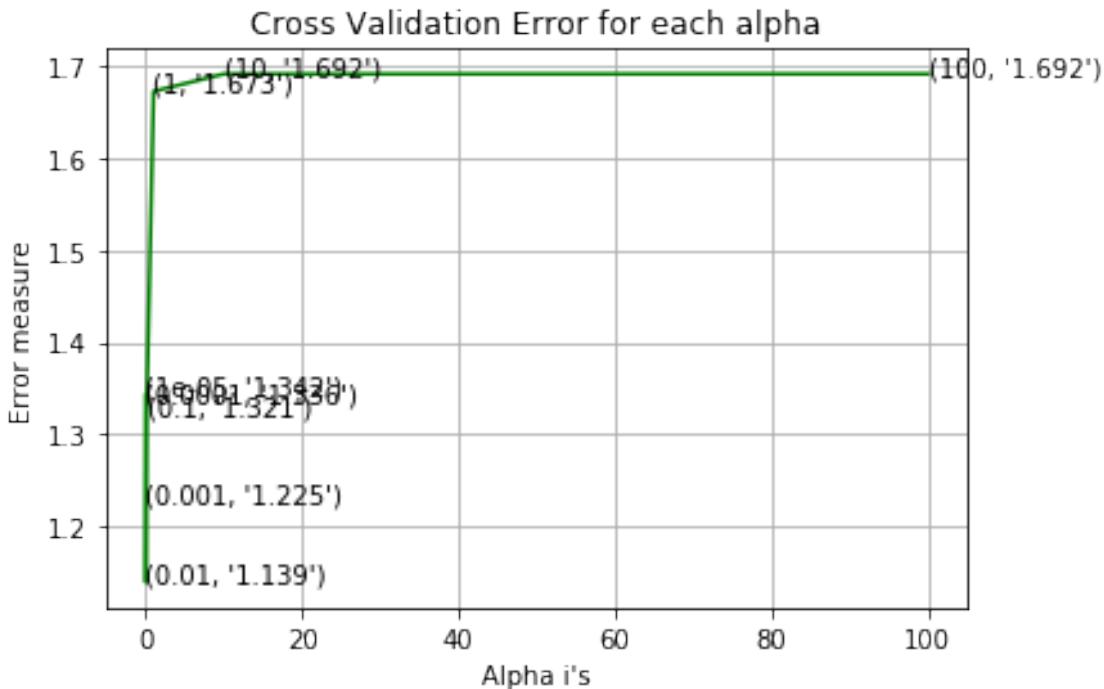
plt.ylabel("Error measure")
plt.show()

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

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,predict_y))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv,predict_y))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test,predict_y))

for C = 1e-05
Log Loss : 1.342303489396577
for C = 0.0001
Log Loss : 1.3356814885170298
for C = 0.001
Log Loss : 1.2253539891642078
for C = 0.01
Log Loss : 1.138722205328733
for C = 0.1
Log Loss : 1.3213414086750566
for C = 1
Log Loss : 1.6726756206225195
for C = 10
Log Loss : 1.6916388700261706
for C = 100
Log Loss : 1.6916387942077529

```



For values of best alpha = 0.01 The train log loss is: 0.7404746947806362

For values of best alpha = 0.01 The cross validation log loss is: 1.138722205328733

For values of best alpha = 0.01 The test log loss is: 1.1320205208734

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

In [77]: # read more about support vector machines with linear kernals here <http://scikit-learn.org/stable/modules/svm.html>

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

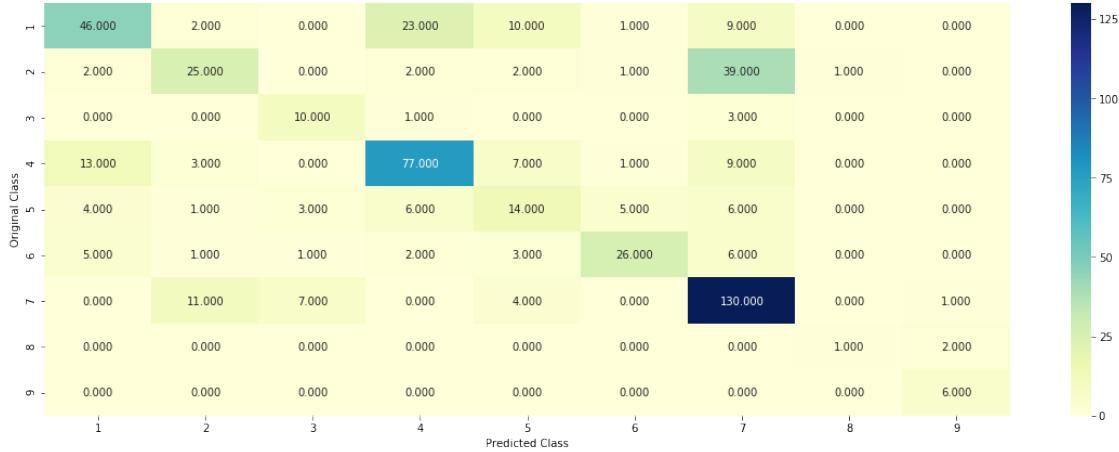
# Some of methods of SVM()
# fit(X, y, [sample_weight])      Fit the SVM model according to the given training data.
# predict(X)                     Perform classification on samples in X.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/support-vector-machines-svm/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

```
# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y)
```

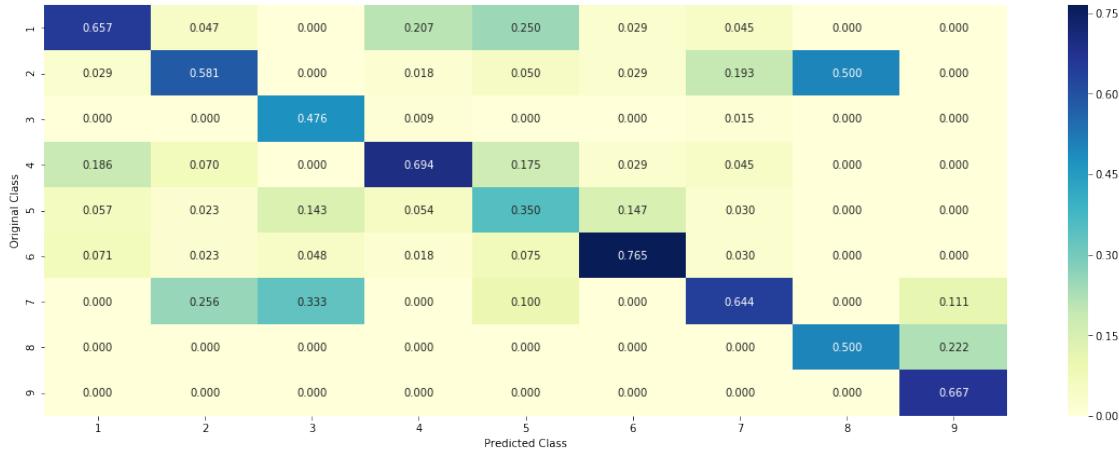
Log loss : 1.138722205328733

Number of mis-classified points : 0.37030075187969924

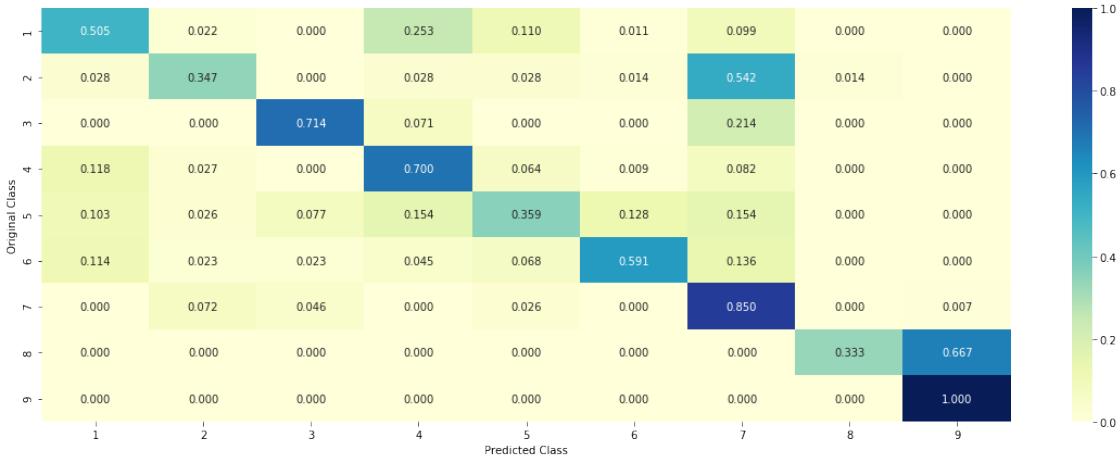
----- Confusion matrix -----



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



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



### 4.3.3. Feature Importance

#### 4.3.3.1. For Correctly classified point

```
In [78]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding)[test_point_index]))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'])
```

Predicted Class : 7

Predicted Class Probabilities: [[0.0834 0.1249 0.0093 0.0823 0.0421 0.0325 0.6158 0.0046 0.0051]]

Actual Class : 2

-----

- 60 Text feature [ligand] present in test data point [True]
- 79 Text feature [receptors] present in test data point [True]
- 108 Text feature [transformation] present in test data point [True]
- 138 Text feature [receptor] present in test data point [True]
- 150 Text feature [technology] present in test data point [True]
- 154 Text feature [egf] present in test data point [True]
- 161 Text feature [activation] present in test data point [True]
- 192 Text feature [downstream] present in test data point [True]
- 336 Text feature [enhanced] present in test data point [True]
- 342 Text feature [oncogene] present in test data point [True]
- 382 Text feature [phosphorylation] present in test data point [True]
- 404 Text feature [errb4] present in test data point [True]

```

444 Text feature [gnas] present in test data point [True]
445 Text feature [map2k1] present in test data point [True]
463 Text feature [mapk] present in test data point [True]
472 Text feature [specimen] present in test data point [True]
486 Text feature [signaling] present in test data point [True]
Out of the top 500 features 17 are present in query point

```

#### 4.3.3.2. For Incorrectly classified point

```

In [79]: test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotC
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imptfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene
Predicted Class : 1
Predicted Class Probabilities: [[0.861  0.0288  0.0049  0.0373  0.02    0.0092  0.0325  0.0042  0.002
Actual Class : 1
-----
8 Text feature [homophilic] present in test data point [True]
9 Text feature [neutravidin] present in test data point [True]
25 Text feature [sk18] present in test data point [True]
27 Text feature [pcp2] present in test data point [True]
31 Text feature [rptp] present in test data point [True]
35 Text feature [mam] present in test data point [True]
42 Text feature [rptps] present in test data point [True]
56 Text feature [ptp] present in test data point [True]
63 Text feature [bescol] present in test data point [True]
71 Text feature [frameshifts] present in test data point [True]
80 Text feature [intercellular] present in test data point [True]
81 Text feature [baculoviral] present in test data point [True]
89 Text feature [aggregation] present in test data point [True]
141 Text feature [aggregates] present in test data point [True]
153 Text feature [cadherin] present in test data point [True]
236 Text feature [sulfo] present in test data point [True]
275 Text feature [mediates] present in test data point [True]
313 Text feature [pcp] present in test data point [True]
361 Text feature [mapping] present in test data point [True]
379 Text feature [aggregate] present in test data point [True]
380 Text feature [adhesive] present in test data point [True]
Out of the top 500 features 21 are present in query point

```

#### 4.5 Random Forest Classifier

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

```
In [80]: # -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth=1,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
# 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/modules/
# -----
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=5)
#
# 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, random_state=42)
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_onehotCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
```

```

cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, )
print("Log Loss :",log_loss(cv_y, sig_clf_probs))

'''fig, ax = plt.subplots()
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_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 = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini',
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

for n_estimators = 100 and max depth =  5
Log Loss : 1.2113533360189637
for n_estimators = 100 and max depth =  10
Log Loss : 1.1572257462011832
for n_estimators = 200 and max depth =  5
Log Loss : 1.2120112084472507
for n_estimators = 200 and max depth =  10
Log Loss : 1.1511553872445246
for n_estimators = 500 and max depth =  5
Log Loss : 1.2128347349092936
for n_estimators = 500 and max depth =  10
Log Loss : 1.134625205904924
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2125296413488056
for n_estimators = 1000 and max depth =  10
Log Loss : 1.132981897683604
for n_estimators = 2000 and max depth =  5
Log Loss : 1.210961376652394
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1313634746534076

```

```

For values of best estimator = 2000 The train log loss is: 0.7147814633013458
For values of best estimator = 2000 The cross validation log loss is: 1.1313634826263137
For values of best estimator = 2000 The test log loss is: 1.1365238738635992

```

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

In [81]: # -----

```

# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth=None,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_nodes=1000,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
# class_weight=None)

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

```

```

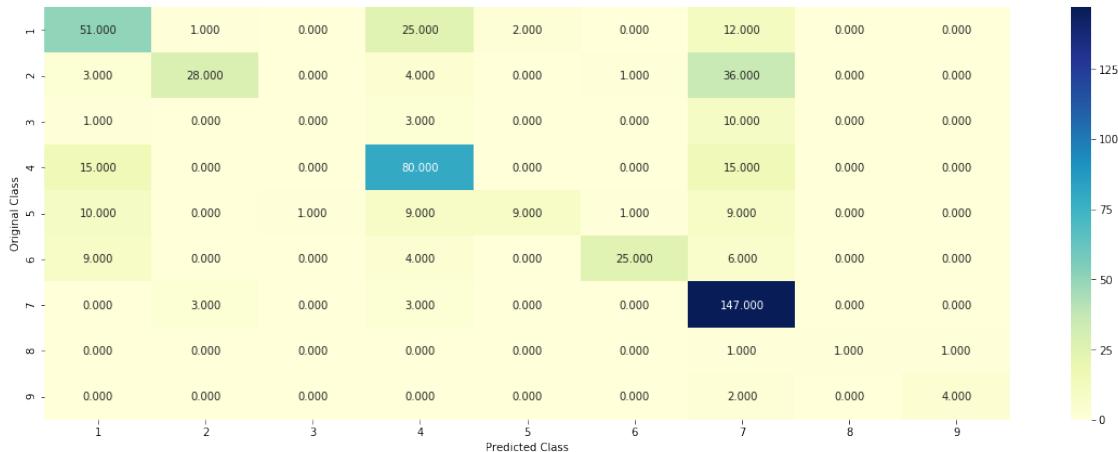
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini',
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y)

```

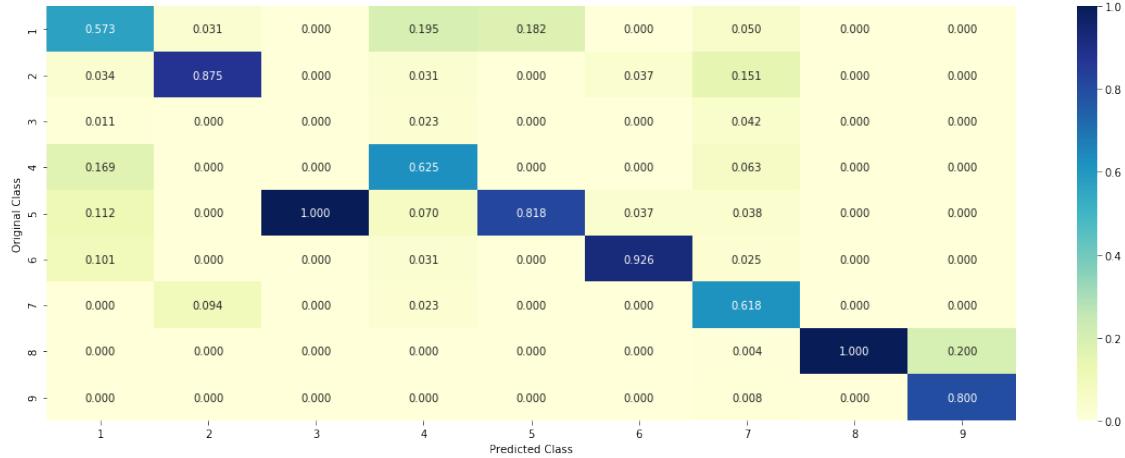
Log loss : 1.1313634826263126

Number of mis-classified points : 0.35150375939849626

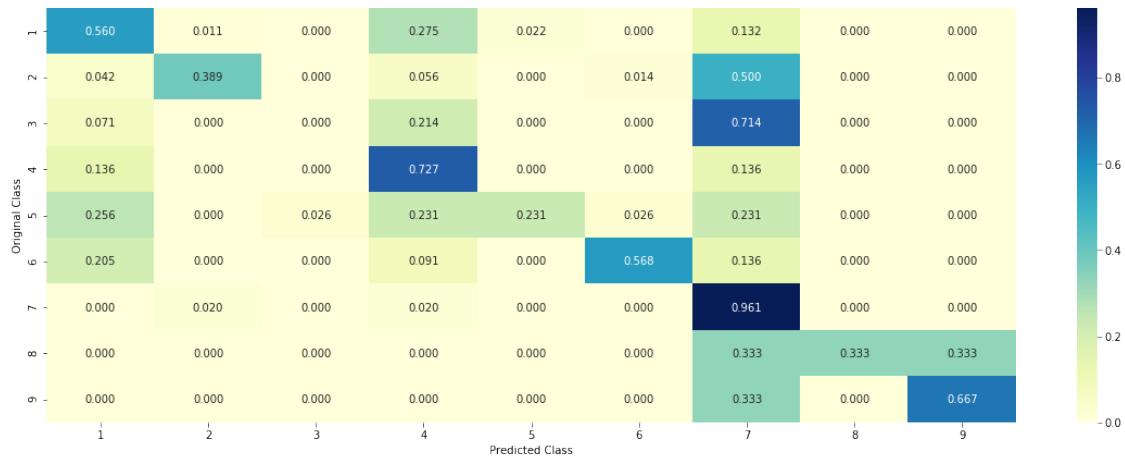
----- Confusion matrix -----



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



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



#### 4.5.3. Feature Importance

##### 4.5.3.1. Correctly Classified point

In [82]: # test\_point\_index = 10

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

```
sig_clf.fit(train_x_onehotCoding, train_y)

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]), 3))
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], test_df['CLASS'].iloc[test_point_index])
```

Predicted Class : 7  
Predicted Class Probabilities: [[0.065 0.3753 0.0168 0.054 0.0395 0.0365 0.4031 0.0048 0.0048 0.0048]  
Actual Class : 2

---

0 Text feature [kinase] present in test data point [True]  
1 Text feature [tyrosine] present in test data point [True]  
3 Text feature [activation] present in test data point [True]  
4 Text feature [suppressor] present in test data point [True]  
6 Text feature [missense] present in test data point [True]  
7 Text feature [inhibitor] present in test data point [True]  
9 Text feature [phosphorylation] present in test data point [True]  
10 Text feature [inhibitors] present in test data point [True]  
11 Text feature [oncogenic] present in test data point [True]  
12 Text feature [function] present in test data point [True]  
13 Text feature [nonsense] present in test data point [True]  
14 Text feature [signaling] present in test data point [True]  
15 Text feature [growth] present in test data point [True]  
16 Text feature [treatment] present in test data point [True]  
17 Text feature [erk] present in test data point [True]  
18 Text feature [akt] present in test data point [True]  
19 Text feature [receptor] present in test data point [True]  
22 Text feature [variants] present in test data point [True]  
23 Text feature [downstream] present in test data point [True]  
26 Text feature [loss] present in test data point [True]  
28 Text feature [inhibition] present in test data point [True]  
29 Text feature [cells] present in test data point [True]  
32 Text feature [trials] present in test data point [True]  
35 Text feature [patients] present in test data point [True]  
39 Text feature [months] present in test data point [True]  
41 Text feature [treated] present in test data point [True]  
43 Text feature [cell] present in test data point [True]  
48 Text feature [protein] present in test data point [True]  
49 Text feature [egfr] present in test data point [True]  
50 Text feature [proliferation] present in test data point [True]  
52 Text feature [ligand] present in test data point [True]  
53 Text feature [kinases] present in test data point [True]

```
56 Text feature [therapy] present in test data point [True]
57 Text feature [ic50] present in test data point [True]
58 Text feature [therapeutic] present in test data point [True]
59 Text feature [oncogene] present in test data point [True]
60 Text feature [drug] present in test data point [True]
61 Text feature [lung] present in test data point [True]
62 Text feature [tki] present in test data point [True]
63 Text feature [expression] present in test data point [True]
64 Text feature [efficacy] present in test data point [True]
65 Text feature [mapk] present in test data point [True]
66 Text feature [dna] present in test data point [True]
67 Text feature [resistance] present in test data point [True]
68 Text feature [transformation] present in test data point [True]
69 Text feature [factor] present in test data point [True]
70 Text feature [proteins] present in test data point [True]
71 Text feature [lines] present in test data point [True]
72 Text feature [trial] present in test data point [True]
73 Text feature [advanced] present in test data point [True]
74 Text feature [response] present in test data point [True]
75 Text feature [clinical] present in test data point [True]
Out of the top 100 features 52 are present in query point
```

#### 4.5.3.2. Incorrectly Classified point

```
In [83]: test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]), 3))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_imptfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index], test_y[test_point_index])

Predicted Class : 1
Predicted Class Probabilities: [[0.4594 0.054 0.0208 0.263 0.056 0.048 0.0847 0.0067 0.0074 0.0074]
Actual Class : 1
-----
2 Text feature [tyrosine] present in test data point [True]
4 Text feature [suppressor] present in test data point [True]
6 Text feature [missense] present in test data point [True]
9 Text feature [phosphorylation] present in test data point [True]
12 Text feature [function] present in test data point [True]
13 Text feature [nonsense] present in test data point [True]
14 Text feature [signaling] present in test data point [True]
15 Text feature [growth] present in test data point [True]
19 Text feature [receptor] present in test data point [True]
```

```

26 Text feature [loss] present in test data point [True]
29 Text feature [cells] present in test data point [True]
43 Text feature [cell] present in test data point [True]
45 Text feature [functional] present in test data point [True]
47 Text feature [inhibited] present in test data point [True]
48 Text feature [protein] present in test data point [True]
50 Text feature [proliferation] present in test data point [True]
52 Text feature [ligand] present in test data point [True]
53 Text feature [kinases] present in test data point [True]
54 Text feature [expressing] present in test data point [True]
63 Text feature [defective] present in test data point [True]
66 Text feature [lung] present in test data point [True]
70 Text feature [expression] present in test data point [True]
75 Text feature [extracellular] present in test data point [True]
76 Text feature [dna] present in test data point [True]
83 Text feature [factor] present in test data point [True]
84 Text feature [proteins] present in test data point [True]
85 Text feature [lines] present in test data point [True]
Out of the top 100 features 27 are present in query point

```

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

```

In [84]: # -----
# 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=None,
# 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/modules/
# -----
# default parameters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv=

```

```

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


alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, r
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, 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 "
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"

```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.103637977276987
for n_estimators = 10 and max depth =  3
Log Loss : 1.8010863768006073
for n_estimators = 10 and max depth =  5
Log Loss : 1.404534126250835
for n_estimators = 10 and max depth =  10
Log Loss : 1.9369920671956589
for n_estimators = 50 and max depth =  2
Log Loss : 1.752894408641626
for n_estimators = 50 and max depth =  3
Log Loss : 1.507140556150414
for n_estimators = 50 and max depth =  5
Log Loss : 1.468738935399488
for n_estimators = 50 and max depth =  10
Log Loss : 1.832929946461027
for n_estimators = 100 and max depth =  2
Log Loss : 1.6387392700000047
for n_estimators = 100 and max depth =  3
Log Loss : 1.5324262357307712
for n_estimators = 100 and max depth =  5
Log Loss : 1.4213535387078071
for n_estimators = 100 and max depth =  10
Log Loss : 1.7845366979080142
for n_estimators = 200 and max depth =  2
Log Loss : 1.6407583616040406
for n_estimators = 200 and max depth =  3
Log Loss : 1.5158616574633044
for n_estimators = 200 and max depth =  5
Log Loss : 1.4403372864705286
for n_estimators = 200 and max depth =  10
Log Loss : 1.7606566386259888
for n_estimators = 500 and max depth =  2
Log Loss : 1.6952502305156436
for n_estimators = 500 and max depth =  3
Log Loss : 1.5934148739185252
for n_estimators = 500 and max depth =  5
Log Loss : 1.472752561714075
for n_estimators = 500 and max depth =  10
Log Loss : 1.8452021343781395
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6619040923245134
for n_estimators = 1000 and max depth =  3
Log Loss : 1.587778743598902
for n_estimators = 1000 and max depth =  5
Log Loss : 1.466440983043204
for n_estimators = 1000 and max depth =  10
Log Loss : 1.816566505491877
```

```

For values of best alpha = 10 The train log loss is: 0.08641895739881247
For values of best alpha = 10 The cross validation log loss is: 1.404534126250835
For values of best alpha = 10 The test log loss is: 1.3801115217789697

```

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

In [85]: # -----

```

# 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=None,
# 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=alpha)
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding)

```

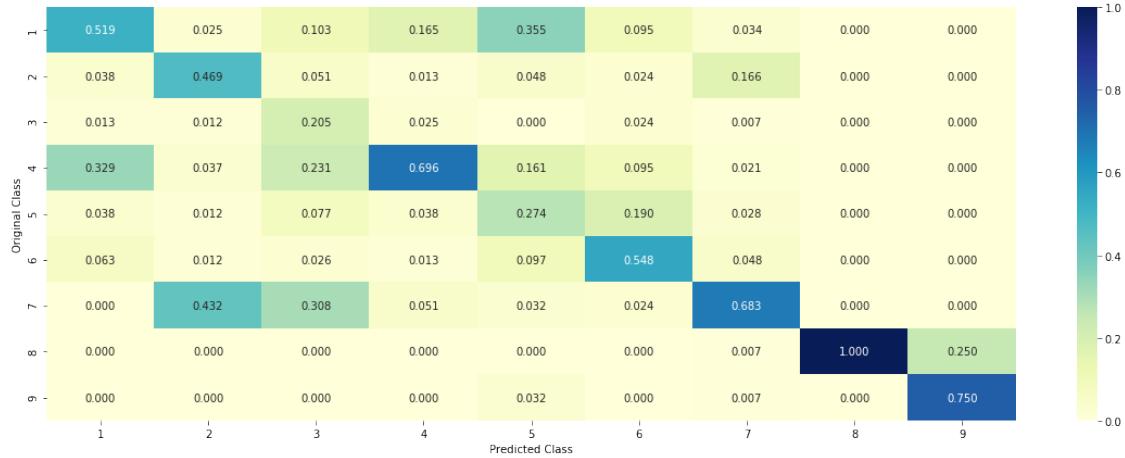
Log loss : 1.404534126250835

Number of mis-classified points : 0.4642857142857143

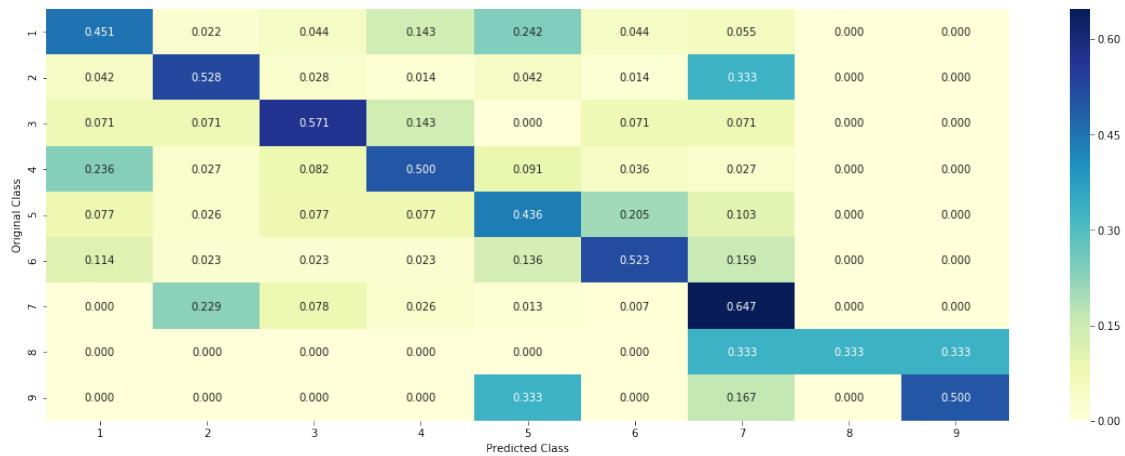
----- Confusion matrix -----



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



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



#### 4.5.5. Feature Importance

##### 4.5.5.1. Correctly Classified point

```
In [86]: clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini',
                                    clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)
```

```

test_point_index = 1
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCoding), 3))
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: [[0.0152 0.4381 0.1017 0.0129 0.0824 0.0315 0.2829 0.0168 0.0184
Actual Class : 2
-----
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
Gene is important feature

```

Variation is important feature

#### 4.5.5.2. Incorrectly Classified point

```
In [87]: 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_responseCoding[test_point_index]), 3))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")

Predicted Class : 1
Predicted Class Probabilities: [[0.4317 0.017  0.1807 0.1123 0.1373 0.0319 0.0065 0.0588 0.0238 0.0001]
Actual Class : 1
-----
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Text is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Gene 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  
Gene is important feature  
Variation is important feature

## 4.7 Stack the models

### 4.7.1 testing with hyper parameter tuning

```
In [88]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn-linear-model-SGDClassifier
# -----
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=optimal,
# 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 Gradient Descent (SGD) algorithm.
# predict(X)          Predict class labels for samples in X.

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


# read more about support vector machines with linear kernels here http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn-svm-SVC
# -----
# default parameters
# SVC(C=1.0, kernel=rbf, degree=3, gamma=auto, coef0=0.0, shrinking=True, probability=False,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr')

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

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


# read more about support vector machines with linear kernels here http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn-ensemble-RandomForestClassifier
# -----
# default parameters
# RandomForestClassifier(n_estimators=10, criterion=gini, max_depth=None,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])      Fit the SVM model according to the given training data.
```

```

# 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
# -----


clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced', random_state=42)
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', random_state=42)
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")

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

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehotCoding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding))))
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_classifier=lr)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifier : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))))
    if best_alpha > log_error:
        best_alpha = log_error

Logistic Regression : Log Loss: 1.15
Support vector machines : Log Loss: 1.67
Naive Bayes : Log Loss: 1.27
-----
Stacking Classifier : for the value of alpha: 0.000100 Log Loss: 2.179

```

Stacking Classifier : for the value of alpha: 0.001000 Log Loss: 2.042  
 Stacking Classifier : for the value of alpha: 0.010000 Log Loss: 1.537  
 Stacking Classifier : for the value of alpha: 0.100000 Log Loss: 1.147  
 Stacking Classifier : for the value of alpha: 1.000000 Log Loss: 1.230  
 Stacking Classifier : for the value of alpha: 10.000000 Log Loss: 1.478

#### 4.7.2 testing the model with the best hyper parameters

```

In [89]: lr = LogisticRegression(C=0.1)
         sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr)
         sclf.fit(train_x_onehotCoding, train_y)

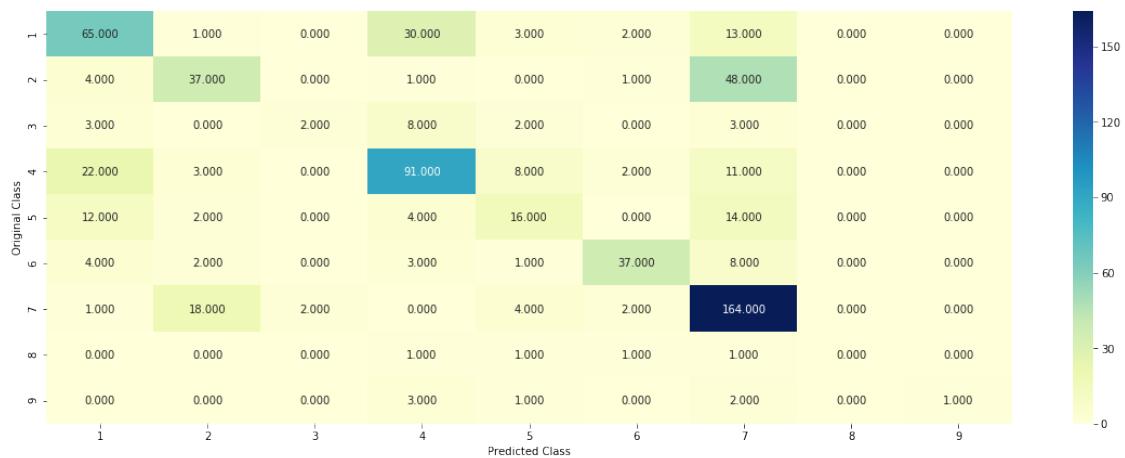
         log_error = log_loss(train_y, sclf.predict_proba(train_x_onehotCoding))
         print("Log loss (train) on the stacking classifier :",log_error)

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

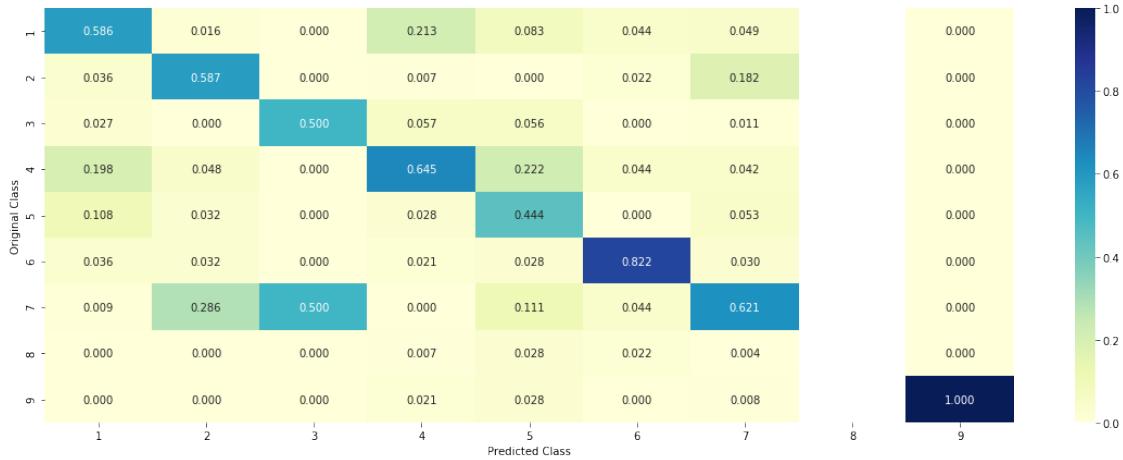
         log_error = log_loss(test_y, sclf.predict_proba(test_x_onehotCoding))
         print("Log loss (test) on the stacking classifier :",log_error)

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

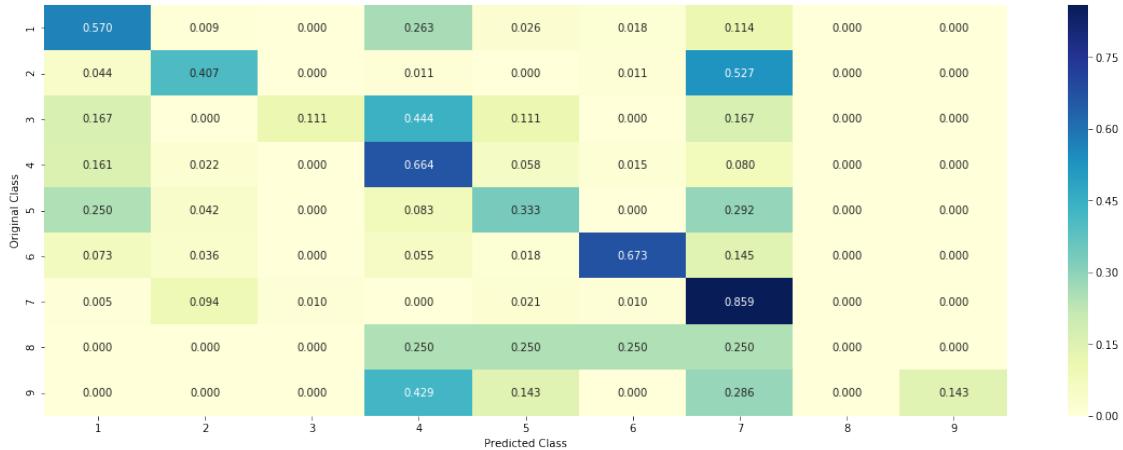
Log loss (train) on the stacking classifier : 0.6747300359186335
Log loss (CV) on the stacking classifier : 1.1466518356124857
Log loss (test) on the stacking classifier : 1.1381149055786157
Number of missclassified point : 0.37894736842105264
----- Confusion matrix -----
  
```



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



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



#### 4.7.3 Maximum Voting classifier

```
In [90]: #Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)])
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier : ", log_loss(train_y, vclf.predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier : ", log_loss(cv_y, vclf.predict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier : ", log_loss(test_y, vclf.predict_proba(test_x_onehotCoding)))
print("Number of missclassified point : ", np.count_nonzero(vclf.predict(test_x_onehotCoding) != test_y))
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

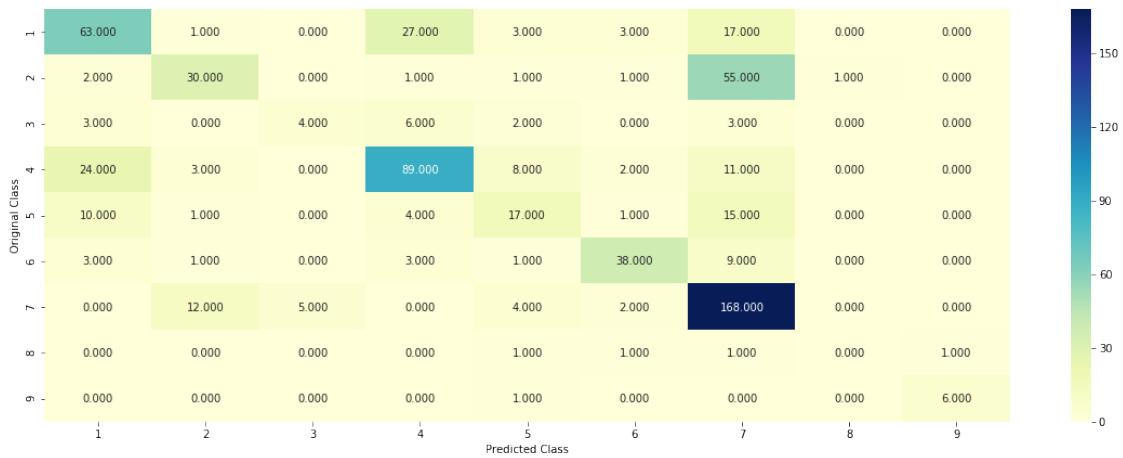
Log loss (train) on the VotingClassifier : 0.9274160819158894

Log loss (CV) on the VotingClassifier : 1.1966930745148814

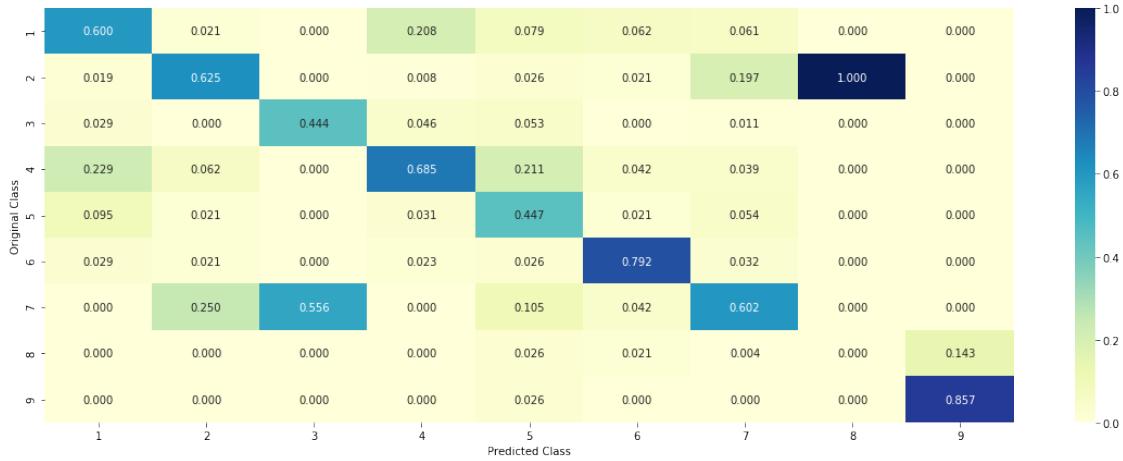
Log loss (test) on the VotingClassifier : 1.2146060549493212

Number of missclassified point : 0.37593984962406013

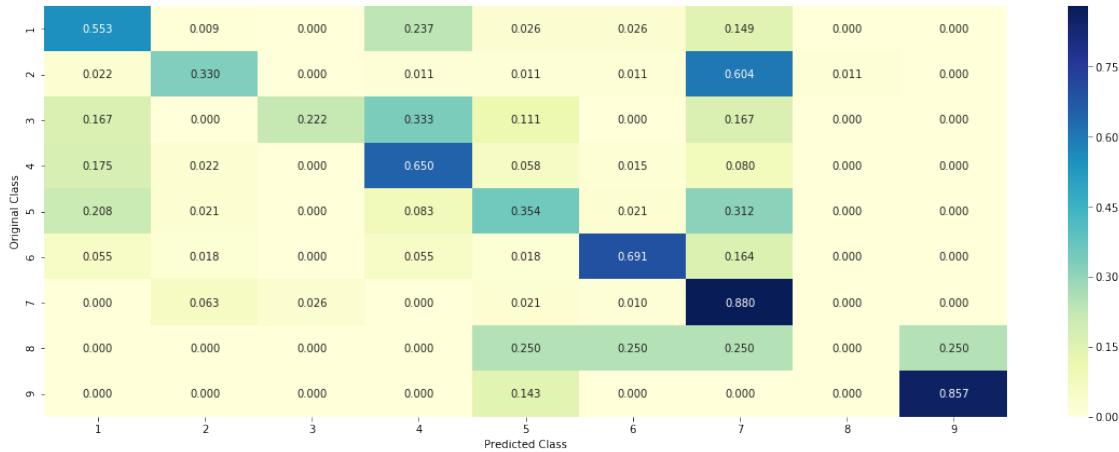
----- Confusion matrix -----



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



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



## 5. Assignments

- <li> Apply All the models with tf-idf features (Replace CountVectorizer with TfidfVectorizer and vice versa)
- <li> Instead of using all the words in the dataset, use only the top 1000 words based of tf-idf
- <li> Apply Logistic regression with CountVectorizer Features, including both unigrams and bigrams
- <li> Try any of the feature engineering techniques discussed in the course to reduce the CV and increase the accuracy

## 5. Tf-Idf Representation

In [53]: #tfidf for gene

```
gene_vectorizer = TfidfVectorizer()
train_gene_feature_tfidfCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_tfidfCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_tfidfCoding = gene_vectorizer.transform(cv_df['Gene'])
```

In [54]: # Tf-Idf encoding of variation feature.

```
variation_vectorizer = TfidfVectorizer()
train_variation_feature_tfidfCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_tfidfCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_tfidfCoding = variation_vectorizer.transform(cv_df['Variation'])
```

In [55]: # building a CountVectorizer with all the words that occurred minimum 3 times in train

```
text_vectorizer = TfidfVectorizer()
train_text_feature_tfidfCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# we use the same vectorizer that was trained on train data
test_text_feature_tfidfCoding = text_vectorizer.transform(test_df['TEXT'])
# we use the same vectorizer that was trained on train data
cv_text_feature_tfidfCoding = text_vectorizer.transform(cv_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()
```

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

```

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

# zip(list(text_features),text_fea_counts) will zip a word with its number of times i
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))

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

Total number of unique words in train data : 127151

```

## Stacking features

```

In [56]: train_gene_var_tfidfCoding = hstack((train_gene_feature_tfidfCoding,train_variation_fea
      test_gene_var_tfidfCoding = hstack((test_gene_feature_tfidfCoding,test_variation_featu
      cv_gene_var_tfidfCoding = hstack((cv_gene_feature_tfidfCoding, cv_variation_feature_tf

      train_x_tfidfCoding = hstack((train_gene_var_tfidfCoding, train_text_feature_tfidfCodin
      train_y = np.array(list(train_df['Class'])))

      test_x_tfidfCoding = hstack((test_gene_var_tfidfCoding, test_text_feature_tfidfCoding))
      test_y = np.array(list(test_df['Class']))

      cv_x_tfidfCoding = hstack((cv_gene_var_tfidfCoding, cv_text_feature_tfidfCoding)).tocs
      cv_y = np.array(list(cv_df['Class']))

```

### 5.1 Machine Learning Models

```

In [57]: 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 will 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))/test_
    plot_confusion_matrix(test_y, pred_y)
    return sig_clf,np.count_nonzero((pred_y- test_y))/test_y.shape[0]

```

#### 5.1.1 Naive Bayes

```

In [68]: 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_tfidfCoding, train_y)

```

```

sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
# to avoid rounding error while multiplying probabilites we use log-probability
print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(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()

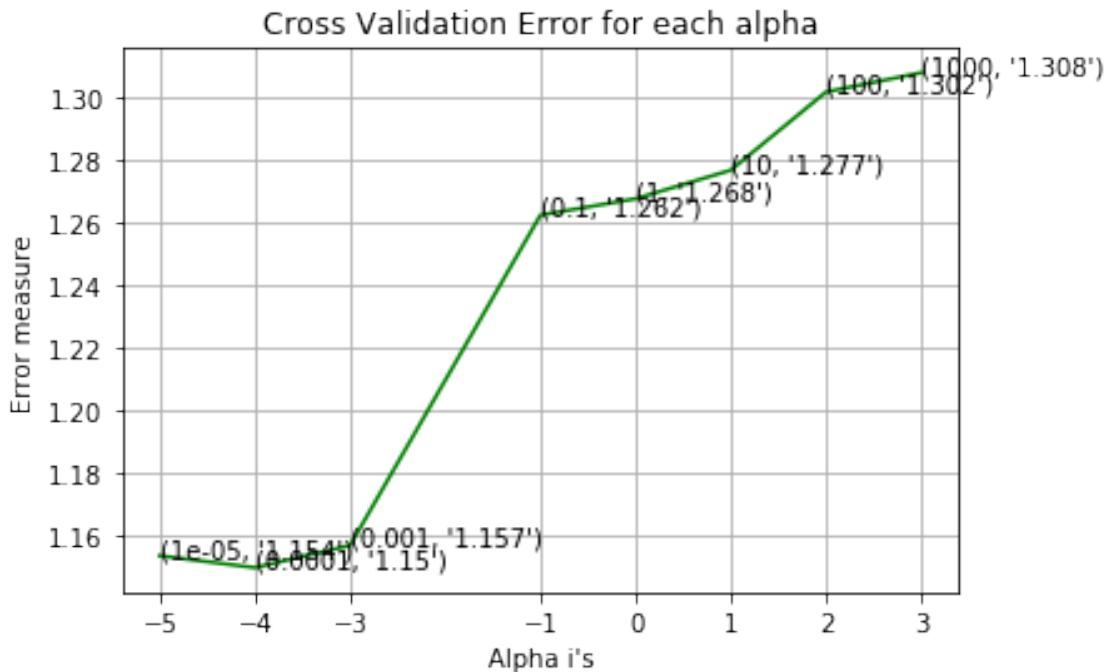
best_alpha = np.argmin(cv_log_error_array)
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(cv_y,predict_y))
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(cv_y,predict_y))
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(test_y,predict_y))

for alpha = 1e-05
Log Loss : 1.15376277848052
for alpha = 0.0001
Log Loss : 1.1499651170315925
for alpha = 0.001
Log Loss : 1.1570268705263178
for alpha = 0.1
Log Loss : 1.262457534066504
for alpha = 1
Log Loss : 1.2676855040880886
for alpha = 10
Log Loss : 1.27680248933506
for alpha = 100
Log Loss : 1.301749764806707
for alpha = 1000

```

Log Loss : 1.3079036945857132



For values of best alpha = 0.0001 The train log loss is: 0.6426009582753089

For values of best alpha = 0.0001 The cross validation log loss is: 1.1499651170315925

For values of best alpha = 0.0001 The test log loss is: 1.1938263431071938

```
In [69]: #alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
alpha = np.random.uniform(0.00005,0.0005,10)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability encoding
    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')
```

```

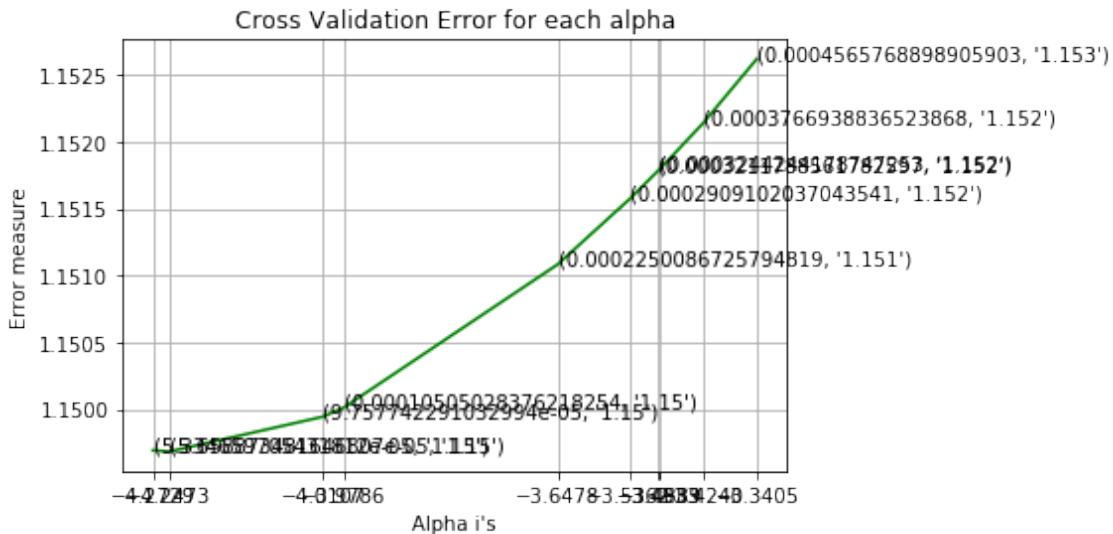
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv_log_error_array[i]))
plt.grid()
plt.xticks(np.log10(alpha))
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

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

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss"
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_1

for alpha = 5.334659705431812e-05
Log Loss : 1.1496925574458239
for alpha = 5.658873481646807e-05
Log Loss : 1.1496839992838097
for alpha = 9.757742291032994e-05
Log Loss : 1.149943707161637
for alpha = 0.00010505028376218254
Log Loss : 1.1500107636347199
for alpha = 0.0002250086725794819
Log Loss : 1.1510822710210182
for alpha = 0.0002909102037043541
Log Loss : 1.1515728534353906
for alpha = 0.0003211788561782597
Log Loss : 1.1517792743026436
for alpha = 0.0003244244178747253
Log Loss : 1.1518007879958452
for alpha = 0.0003766938836523868
Log Loss : 1.1521336186300168
for alpha = 0.0004565768898905903
Log Loss : 1.1526167188792114

```



For values of best alpha =  $5.658873481646807 \times 10^{-5}$  The train log loss is: 0.6371122217662338  
 For values of best alpha =  $5.658873481646807 \times 10^{-5}$  The cross validation log loss is: 1.149683999  
 For values of best alpha =  $5.658873481646807 \times 10^{-5}$  The test log loss is: 1.195156358489712

```
In [70]: #alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
alpha = np.random.uniform(0.000005,0.00005,10)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(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")
```

```

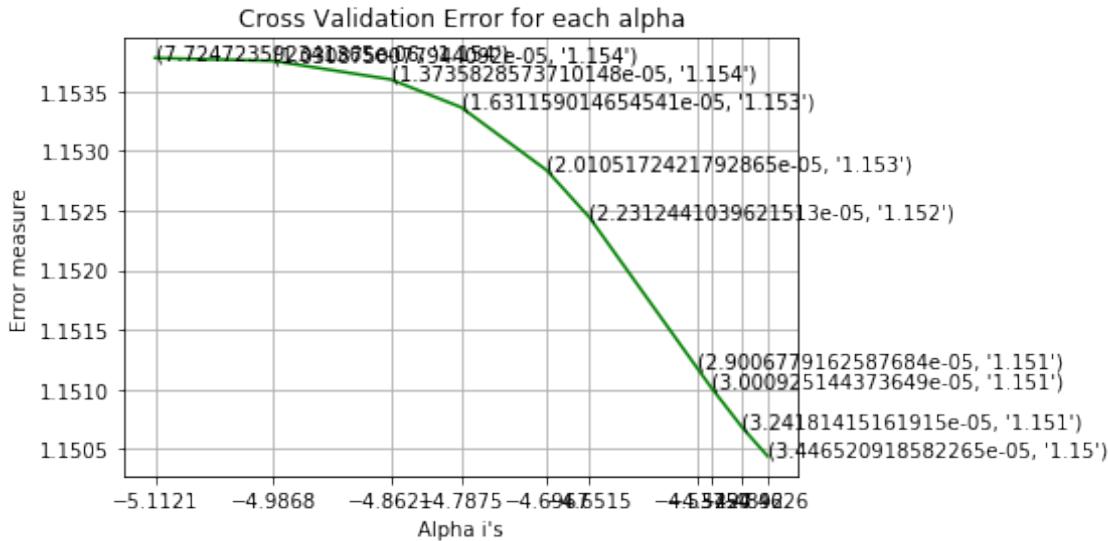
plt.show()

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

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_1

for alpha = 7.724723592341365e-06
Log Loss : 1.1537806036918463
for alpha = 1.0308750077944092e-05
Log Loss : 1.1537562429585189
for alpha = 1.3735828573710148e-05
Log Loss : 1.1536001572111398
for alpha = 1.631159014654541e-05
Log Loss : 1.1533664144160818
for alpha = 2.0105172421792865e-05
Log Loss : 1.1528374098258973
for alpha = 2.2312441039621513e-05
Log Loss : 1.1524420907998558
for alpha = 2.9006779162587684e-05
Log Loss : 1.1511814242626055
for alpha = 3.000925144373649e-05
Log Loss : 1.1510194936135605
for alpha = 3.24181415161915e-05
Log Loss : 1.150677755884181
for alpha = 3.446520918582265e-05
Log Loss : 1.1504403812521464

```

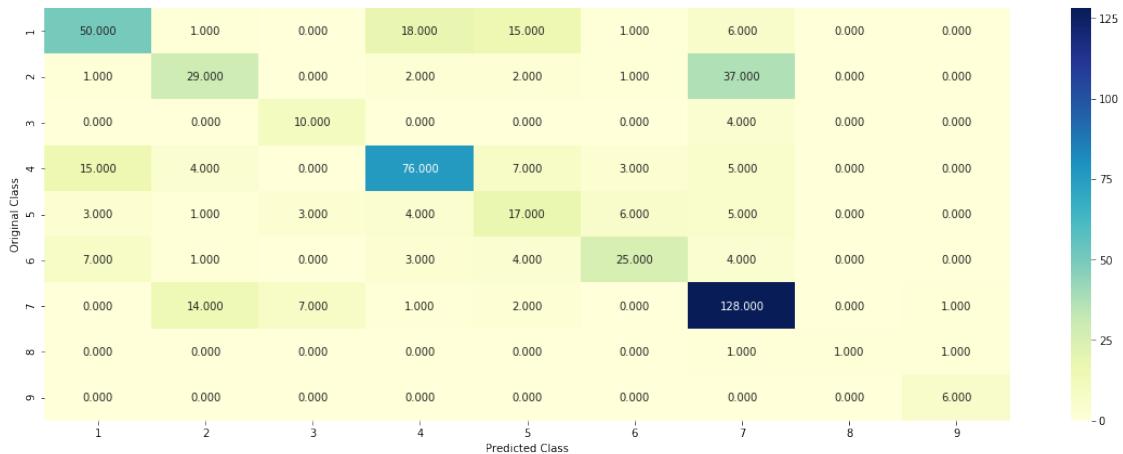


For values of best alpha = 3.446520918582265e-05 The train log loss is: 0.6328703968992225  
 For values of best alpha = 3.446520918582265e-05 The cross validation log loss is: 1.15044038  
 For values of best alpha = 3.446520918582265e-05 The test log loss is: 1.196649072381611

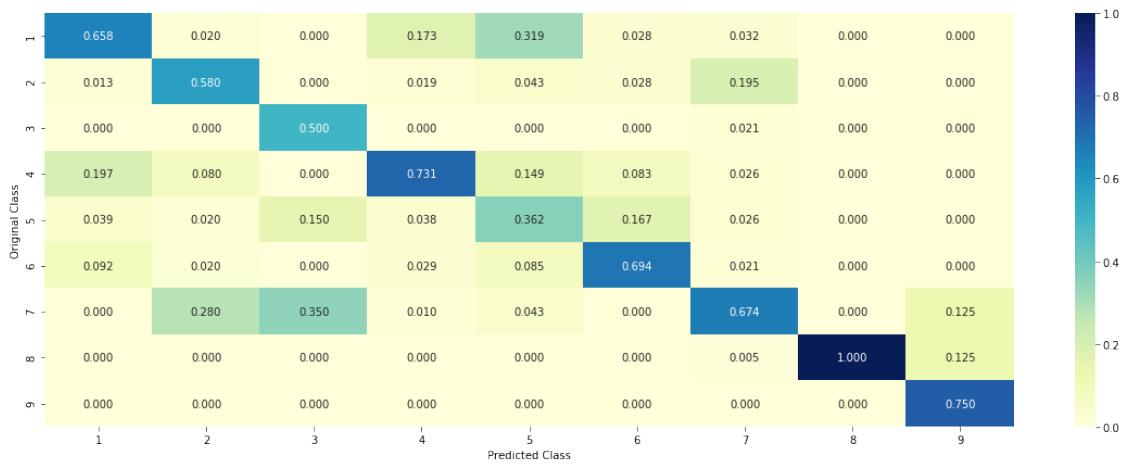
```
In [72]: ##error and confusioon matrix
final_results = []

clf = MultinomialNB(alpha=5.658873481646807e-05)
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
# to avoid rounding error while multiplying probalilities we use log-probability estimat
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_tfidfCoding) != cv_y)))
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_tfidfCoding.toarray()))
list_data = []
list_data.append('Tf_Idf+NB')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_tfidfCoding), labels=clf.classes_))
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_tfidfCoding), labels=clf.classes_))
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_tfidfCoding), labels=clf.classes_))
list_data.append('alpha = '+str(clf.alpha))
list_data.append(np.count_nonzero((sig_clf.predict(cv_x_tfidfCoding) != cv_y))/cv_y.shape[0])
final_results.append(list_data)
```

Log Loss : 1.1496839992838097  
 Number of missclassified point : 0.35714285714285715  
 ----- Confusion matrix -----



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



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



In [73]: final\_results

Out[73]: [['Tf\_Idf+NB',  
0.6371122217662338,  
1.1496839992838097,  
1.195156358489712,  
'alpha = 5.658873481646807e-05',  
0.35714285714285715]]

## Feature Importance

```
In [55]: # this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer()

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

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

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

```

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

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

```

```

In [75]: test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCoding)[test_point_index][0]*100))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imptfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']

```

```

Predicted Class : 7
Predicted Class Probabilities: [[0.0748 0.0627 0.0117 0.0834 0.0329 0.037 0.6907 0.0037 0.0037 0.0037]
Actual Class : 2
-----
1 Text feature [mutations] present in test data point [True]
2 Text feature [cells] present in test data point [True]
5 Text feature [egfr] present in test data point [True]
9 Text feature [cell] present in test data point [True]
12 Text feature [kinase] present in test data point [True]
14 Text feature [mutation] present in test data point [True]
16 Text feature [patients] present in test data point [True]
17 Text feature [al] present in test data point [True]
18 Text feature [et] present in test data point [True]
20 Text feature [figure] present in test data point [True]
24 Text feature [pdgfra] present in test data point [True]
25 Text feature [cancer] present in test data point [True]
29 Text feature [mutant] present in test data point [True]
32 Text feature [braf] present in test data point [True]
35 Text feature [tumor] present in test data point [True]
36 Text feature [activation] present in test data point [True]
```

```

37 Text feature [mutants] present in test data point [True]
41 Text feature [tumors] present in test data point [True]
42 Text feature [fgfr2] present in test data point [True]
43 Text feature [activity] present in test data point [True]
44 Text feature [domain] present in test data point [True]
45 Text feature [expression] present in test data point [True]
46 Text feature [akt] present in test data point [True]
50 Text feature [gefitinib] present in test data point [True]
51 Text feature [resistance] present in test data point [True]
52 Text feature [exon] present in test data point [True]
54 Text feature [also] present in test data point [True]
55 Text feature [signaling] present in test data point [True]
57 Text feature [using] present in test data point [True]
58 Text feature [phosphorylation] present in test data point [True]
61 Text feature [type] present in test data point [True]
62 Text feature [protein] present in test data point [True]
64 Text feature [growth] present in test data point [True]
65 Text feature [gene] present in test data point [True]
66 Text feature [10] present in test data point [True]
70 Text feature [receptor] present in test data point [True]
71 Text feature [treatment] present in test data point [True]
72 Text feature [tyrosine] present in test data point [True]
78 Text feature [kras] present in test data point [True]
79 Text feature [fusion] present in test data point [True]
80 Text feature [inhibitors] present in test data point [True]
81 Text feature [analysis] present in test data point [True]
82 Text feature [lines] present in test data point [True]
83 Text feature [data] present in test data point [True]
85 Text feature [lung] present in test data point [True]
86 Text feature [table] present in test data point [True]
87 Text feature [pathway] present in test data point [True]
88 Text feature [two] present in test data point [True]
92 Text feature [may] present in test data point [True]
93 Text feature [erk] present in test data point [True]
95 Text feature [fgfr3] present in test data point [True]
97 Text feature [identified] present in test data point [True]
99 Text feature [clinical] present in test data point [True]
Out of the top 100 features 53 are present in query point

```

```

In [76]: test_point_index = 15
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCoding[test_point_index])[0]))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)

```

```

get_imptfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index], test_df['Gene']

Predicted Class : 6
Predicted Class Probabilities: [[0.0722 0.0608 0.0113 0.0808 0.032 0.6436 0.0927 0.0035 0.003
Actual Class : 6
-----
3 Text feature [variants] present in test data point [True]
8 Text feature [mutations] present in test data point [True]
12 Text feature [cancer] present in test data point [True]
15 Text feature [mutation] present in test data point [True]
19 Text feature [variant] present in test data point [True]
22 Text feature [fig] present in test data point [True]
23 Text feature [family] present in test data point [True]
24 Text feature [data] present in test data point [True]
33 Text feature [analysis] present in test data point [True]
35 Text feature [breast] present in test data point [True]
40 Text feature [resistance] present in test data point [True]
43 Text feature [table] present in test data point [True]
44 Text feature [type] present in test data point [True]
45 Text feature [protein] present in test data point [True]
46 Text feature [cells] present in test data point [True]
49 Text feature [binding] present in test data point [True]
53 Text feature [used] present in test data point [True]
55 Text feature [activity] present in test data point [True]
56 Text feature [kinase] present in test data point [True]
58 Text feature [using] present in test data point [True]
59 Text feature [domain] present in test data point [True]
60 Text feature [two] present in test data point [True]
61 Text feature [also] present in test data point [True]
63 Text feature [10] present in test data point [True]
64 Text feature [interaction] present in test data point [True]
74 Text feature [residues] present in test data point [True]
76 Text feature [jak2] present in test data point [True]
77 Text feature [et] present in test data point [True]
78 Text feature [figure] present in test data point [True]
79 Text feature [associated] present in test data point [True]
80 Text feature [clinical] present in test data point [True]
82 Text feature [mek1] present in test data point [True]
85 Text feature [cell] present in test data point [True]
86 Text feature [fedoratinib] present in test data point [True]
87 Text feature [supplementary] present in test data point [True]
90 Text feature [domains] present in test data point [True]
91 Text feature [structure] present in test data point [True]
92 Text feature [one] present in test data point [True]
94 Text feature [number] present in test data point [True]
98 Text feature [functional] present in test data point [True]
99 Text feature [models] present in test data point [True]
Out of the top 100 features 41 are present in query point

```

### 5.1.2. K Nearest Neighbour Classification

```
In [80]: 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_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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

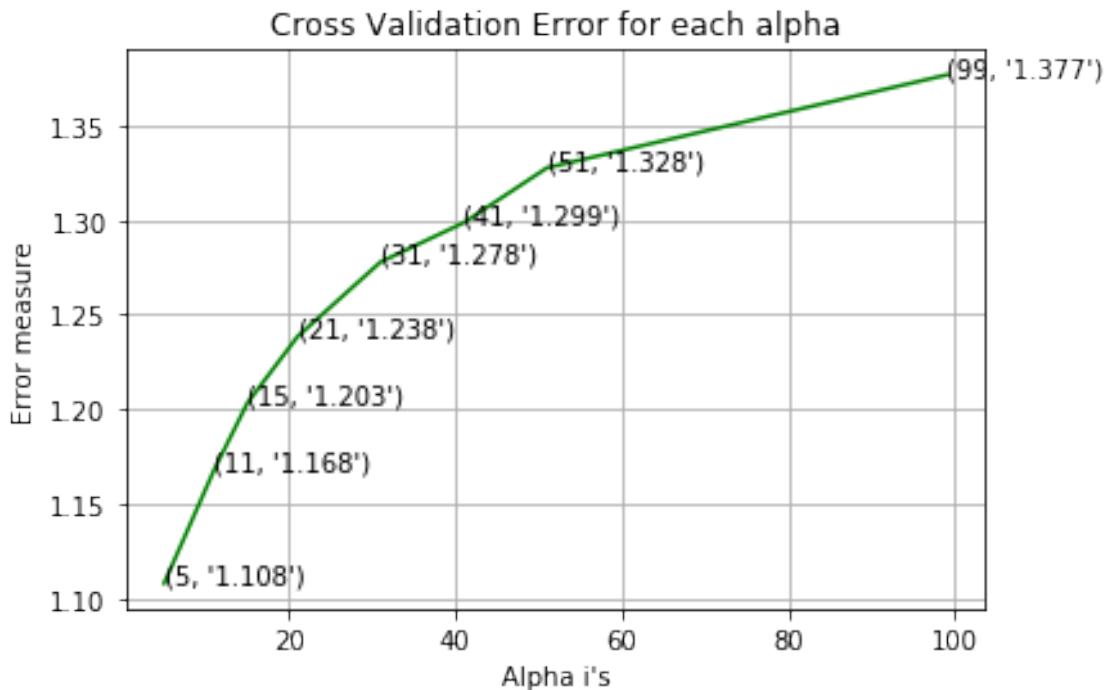
predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_

for alpha = 5
Log Loss : 1.107991255482852
for alpha = 11
Log Loss : 1.1678783627738878
for alpha = 15
Log Loss : 1.2032557565274102
for alpha = 21
```

```

Log Loss : 1.2382715283628698
for alpha = 31
Log Loss : 1.2778164517021868
for alpha = 41
Log Loss : 1.2985929960678644
for alpha = 51
Log Loss : 1.3276173388159187
for alpha = 99
Log Loss : 1.3766930149678058

```



```

For values of best alpha = 5 The train log loss is: 0.9004519693505992
For values of best alpha = 5 The cross validation log loss is: 1.107991255482852
For values of best alpha = 5 The test log loss is: 1.1357853217405187

```

```

In [81]: #testing
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha], algorithm='brute')
sig_clf,temp = predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y, cv_x_t
list_data = []
list_data.append('Tf_Idf+KNN')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_tfidfCoding), eps=1e-15))
list_data.append('K '+str(clf.n_neighbors))

```

```

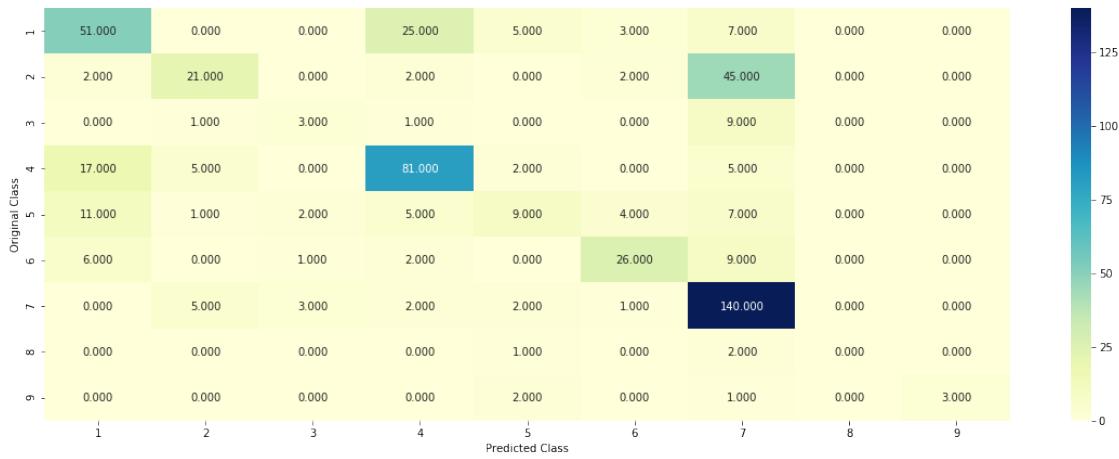
list_data.append(temp)
final_results.append(list_data)

```

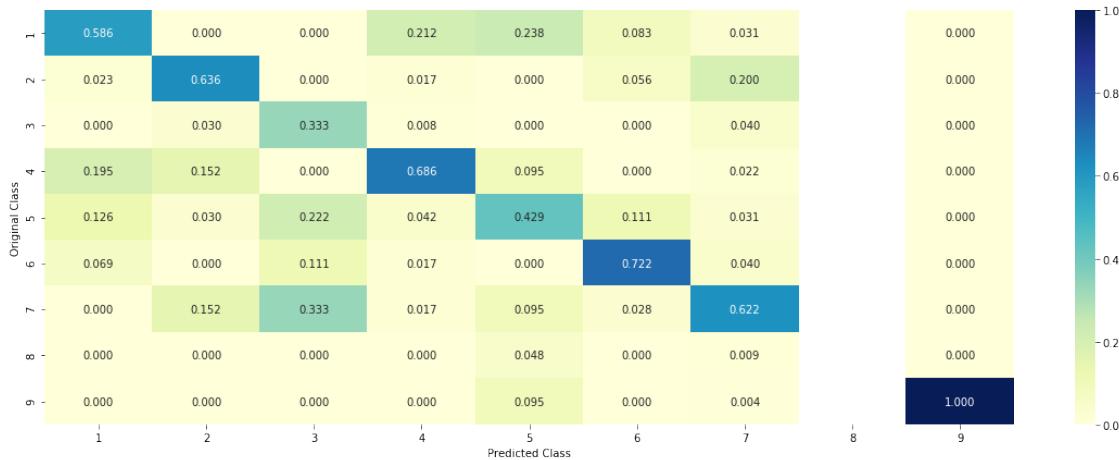
Log loss : 1.107991255482852

Number of mis-classified points : 0.37218045112781956

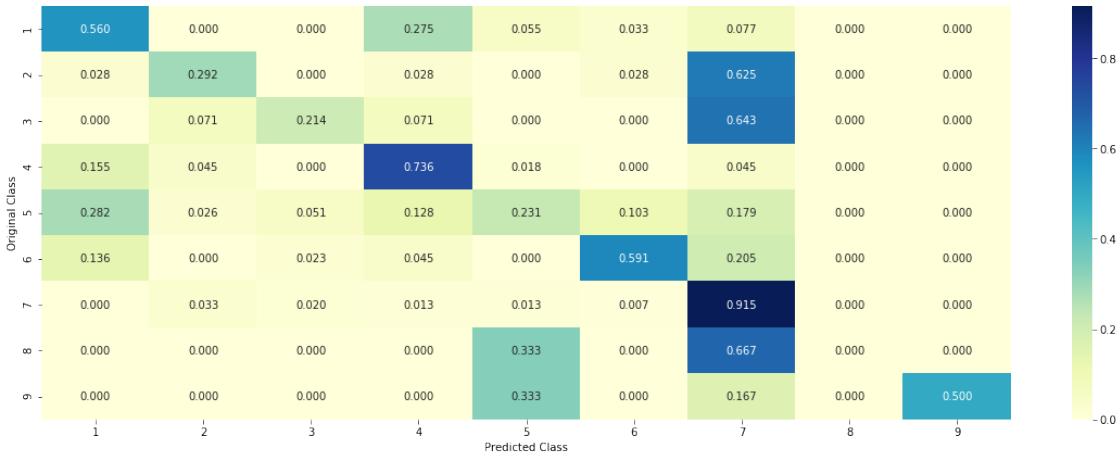
----- Confusion matrix -----



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



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



In [82]: final\_results

```
Out[82]: [['Tf_Idf+NB',
  0.6371122217662338,
  1.1496839992838097,
  1.195156358489712,
  'alpha = 5.658873481646807e-05',
  0.35714285714285715],
 ['Tf_Idf+KNN',
  0.9004519693505992,
  1.107991255482852,
  1.1357853217405187,
  'K 5',
  0.37218045112781956]]
```

## Feature importance

```
In [83]: clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

test_point_index = 45
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_tfidfCoding[test_point_index], alpha[best_alpha])
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to class")
print("Frequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

Predicted Class : 7

Actual Class : 5

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

```
In [84]: clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

test_point_index = 95
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_tfidfCoding[test_point_index], alpha[best_alpha])
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to class")
print("Frequency of nearest points :",Counter(train_y[neighbors[1][0]]))

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

### 5.1.3. Logistic Regression

#### With Class balancing

```
In [85]: 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', random_state=42)
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability encoding
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

```

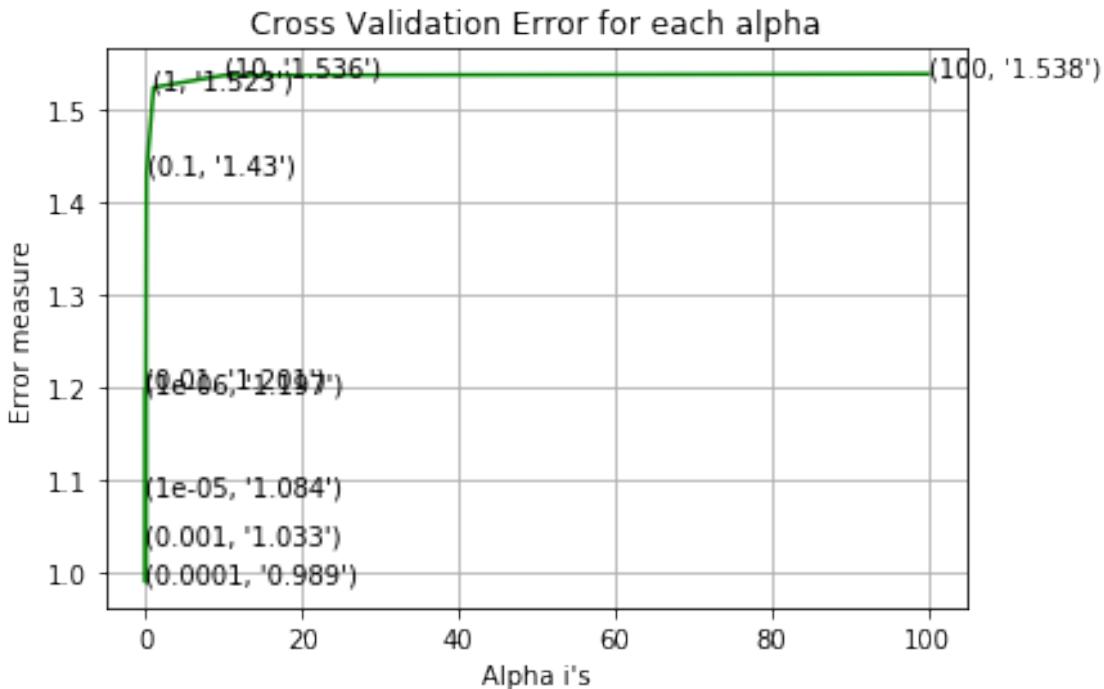
plt.show()

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

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_...
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss i...
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_...

for alpha = 1e-06
Log Loss : 1.1966916389116102
for alpha = 1e-05
Log Loss : 1.0837561566720058
for alpha = 0.0001
Log Loss : 0.9894097373140254
for alpha = 0.001
Log Loss : 1.0329350394113541
for alpha = 0.01
Log Loss : 1.2006080134373862
for alpha = 0.1
Log Loss : 1.4302110792268927
for alpha = 1
Log Loss : 1.5230580606992898
for alpha = 10
Log Loss : 1.5360093178046328
for alpha = 100
Log Loss : 1.5375433101610616

```



For values of best alpha = 0.0001 The train log loss is: 0.42896803046962967  
 For values of best alpha = 0.0001 The cross validation log loss is: 0.9894097373140254  
 For values of best alpha = 0.0001 The test log loss is: 0.9644491319096643

```
In [86]: #alpha = [10 ** x for x in range(-6, 3)]
alpha = np.random.uniform(0.00005, 0.0005, 15)
alpha = np.round(alpha, 7)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
# to avoid rounding error while multiplying probabilites we use log-probability encoding
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', loss='log')
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,predict_y))
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv,predict_y))
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test,predict_y))

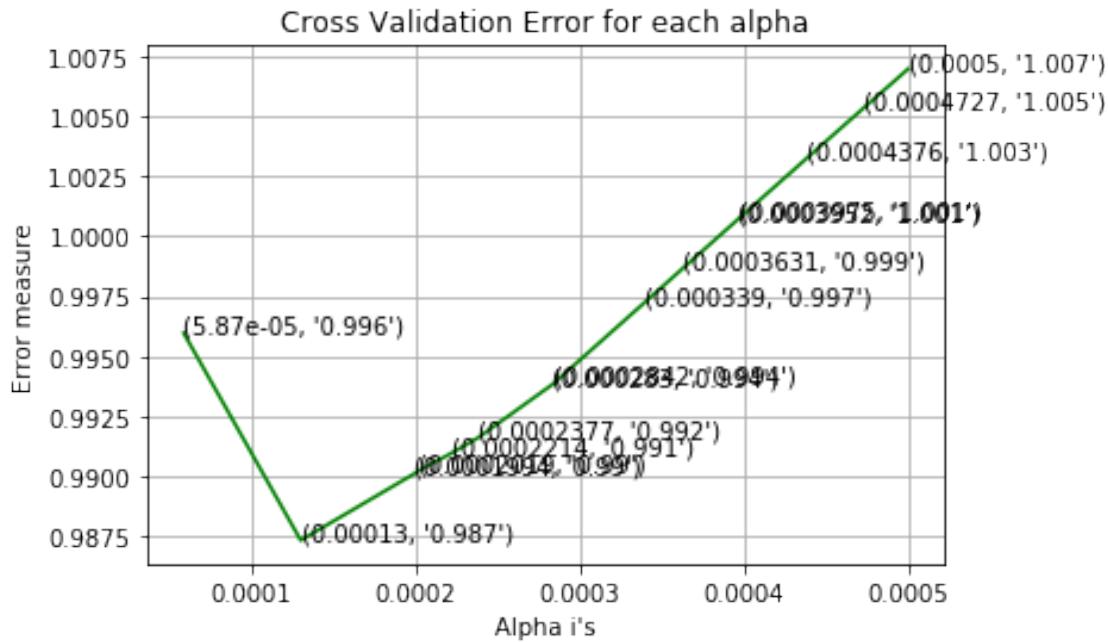
for alpha = 5.87e-05
Log Loss : 0.9959720848300324
for alpha = 0.00013
Log Loss : 0.9873457117066042
for alpha = 0.0001994
Log Loss : 0.9901254867952514
for alpha = 0.0002019
Log Loss : 0.99024049537132
for alpha = 0.0002214
Log Loss : 0.9909562925758523
for alpha = 0.0002377
Log Loss : 0.9915980601029268
for alpha = 0.000283
Log Loss : 0.9938152938931915
for alpha = 0.0002842
Log Loss : 0.9938848965072714
for alpha = 0.000339
Log Loss : 0.9972114142881721
for alpha = 0.0003631
Log Loss : 0.9986821548475427
for alpha = 0.0003952
Log Loss : 1.000633396959546
for alpha = 0.0003975
Log Loss : 1.0007729973985398
for alpha = 0.0004376
Log Loss : 1.003203099730527
for alpha = 0.0004727

```

```

Log Loss : 1.0053209029271017
for alpha = 0.0005
Log Loss : 1.0069574088875826

```



```

For values of best alpha = 0.00013 The train log loss is: 0.4402979991695989
For values of best alpha = 0.00013 The cross validation log loss is: 0.9873457117066042
For values of best alpha = 0.00013 The test log loss is: 0.9624944368737652

```

```

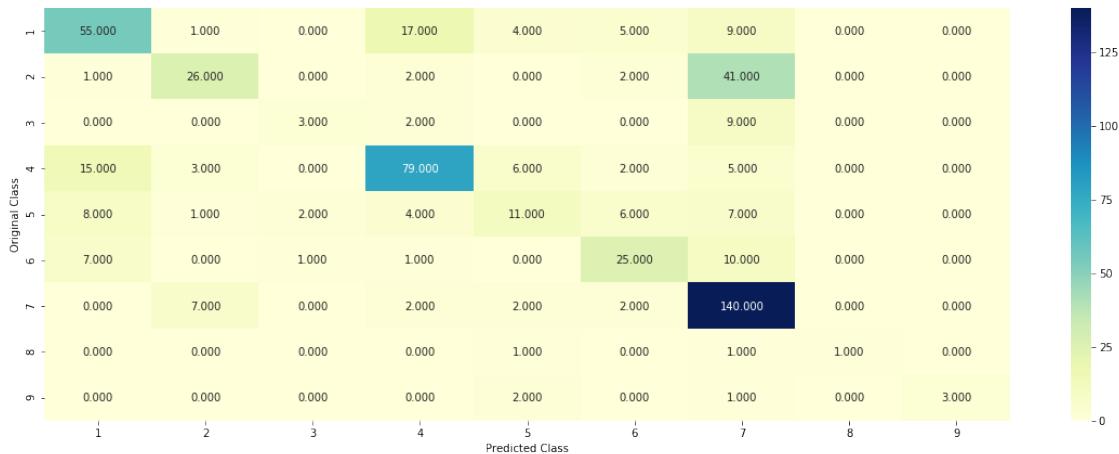
In [87]: clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
                           sig_clf,temp = predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y, cv_x_tfidfCoding))
list_data = []
list_data.append('Tf_Idf+LR+Class_Balancing+SGD')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_tfidfCoding), eps=1e-15))
list_data.append('alpha = '+str(clf.alpha))
list_data.append(temp)
final_results.append(list_data)

```

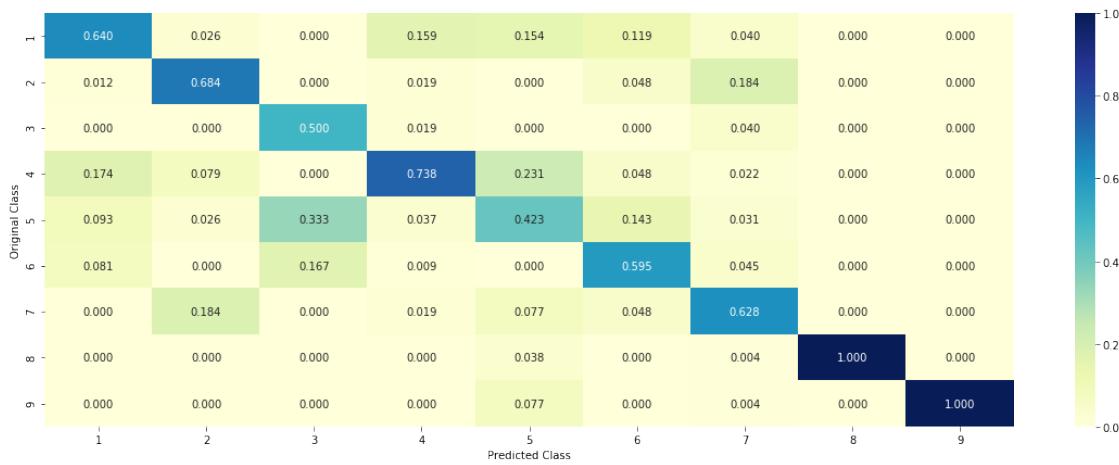
```

Log loss : 0.9873457117066042
Number of mis-classified points : 0.35526315789473684
----- Confusion matrix -----

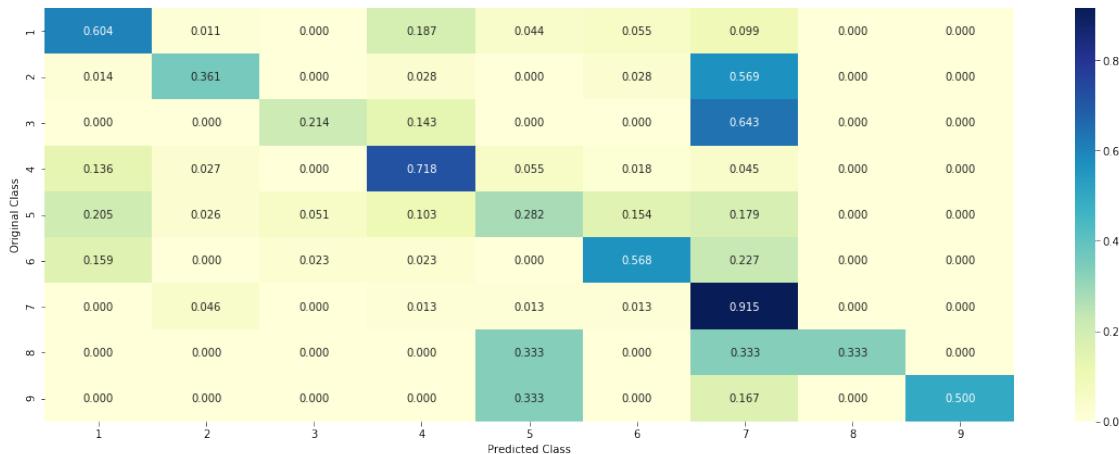
```



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



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



```
In [88]: alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = LogisticRegression(C=i, class_weight='balanced', n_jobs=-1, solver='liblinear')
    #clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log',
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability encoding
    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()

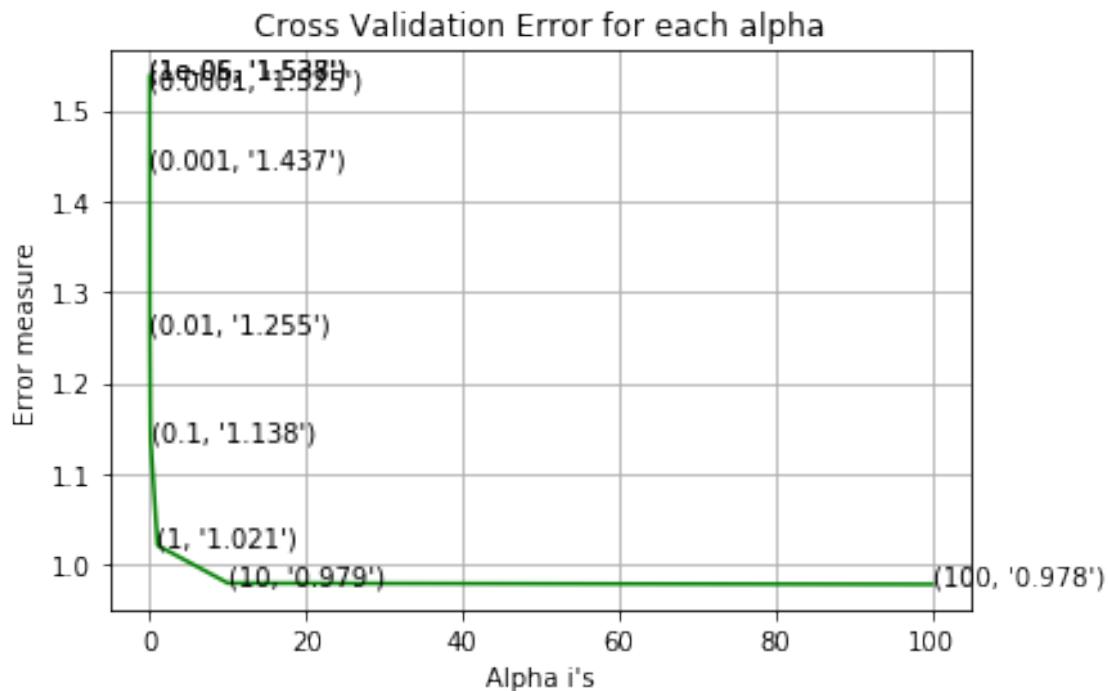
best_alpha = np.argmin(cv_log_error_array)
clf = LogisticRegression(C=i, class_weight='balanced', n_jobs=-1, solver='liblinear')
#clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)
```

```

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_)
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_cv)
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_test)

for alpha = 1e-06
Log Loss : 1.5378301120864848
for alpha = 1e-05
Log Loss : 1.5366082114308264
for alpha = 0.0001
Log Loss : 1.5247184439608028
for alpha = 0.001
Log Loss : 1.436717873997484
for alpha = 0.01
Log Loss : 1.255064285406299
for alpha = 0.1
Log Loss : 1.1382836048786462
for alpha = 1
Log Loss : 1.0210162070652247
for alpha = 10
Log Loss : 0.9794452529861118
for alpha = 100
Log Loss : 0.9781389963217265

```



```

For values of best alpha = 100 The train log loss is: 0.4398146469741785
For values of best alpha = 100 The cross validation log loss is: 0.9805505072745702
For values of best alpha = 100 The test log loss is: 0.9547086478060672

```

```

In [90]: alpha = np.random.uniform(95,125,5)
alpha = np.round(alpha,3)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = LogisticRegression(C=i,class_weight='balanced',n_jobs=-1,solver='liblinear')
    #clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log',
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

best_alpha = np.argmin(cv_log_error_array)
clf = LogisticRegression(C=i,class_weight='balanced',n_jobs=-1,solver='liblinear')
#clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

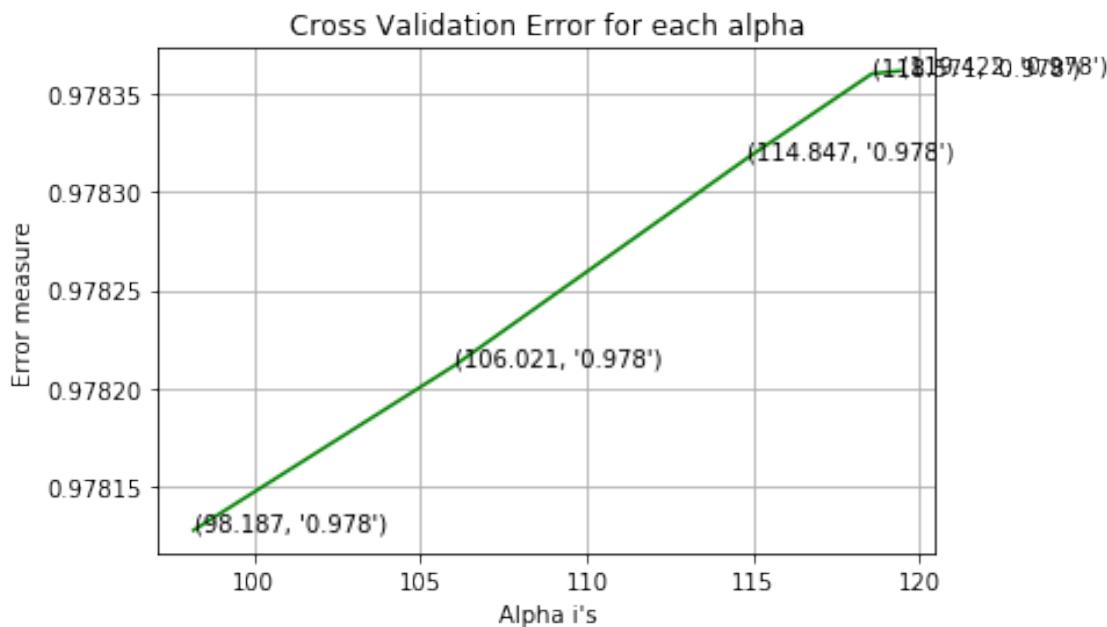
predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss",
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_

```

```

for alpha = 98.187
Log Loss : 0.9781279782951176
for alpha = 106.021
Log Loss : 0.9782115778425603
for alpha = 114.847
Log Loss : 0.9783172480784893
for alpha = 118.571
Log Loss : 0.9783598357694149
for alpha = 119.422
Log Loss : 0.9783612731455397

```



```

For values of best alpha = 98.187 The train log loss is: 0.48221751436365423
For values of best alpha = 98.187 The cross validation log loss is: 0.9869732521430139
For values of best alpha = 98.187 The test log loss is: 0.9607468494913577

```

In [91]: ##test

```

clf = LogisticRegression(C=alpha[best_alpha], class_weight='balanced', n_jobs=-1, solver='liblinear')
#clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
#sig_clf,temp = predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y, cv_x_tfidfCoding)
list_data = []
list_data.append('Tf_Idf+LR+Class_Balancing+Liblinear')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_tfidfCoding), eps=1e-15))

```

```

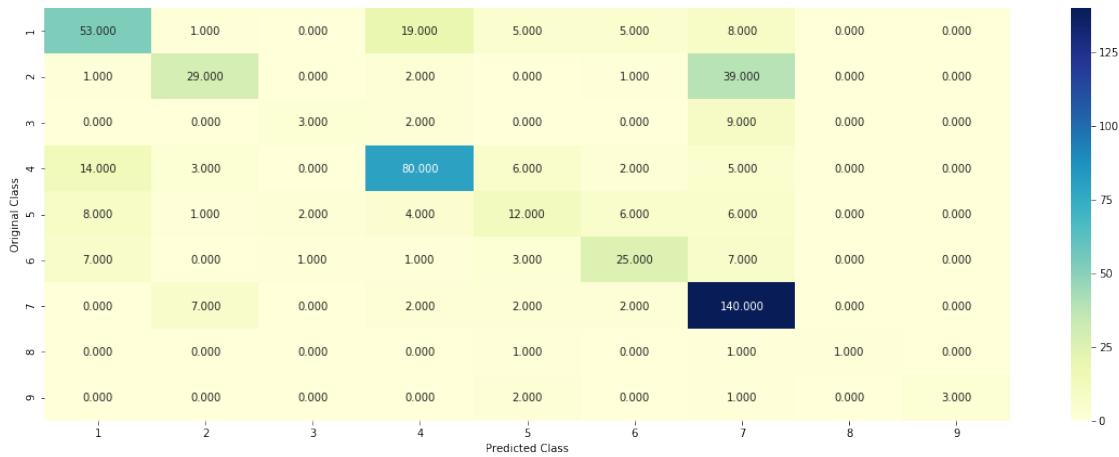
list_data.append('C = '+str(clf.C))
list_data.append(temp)
final_results.append(list_data)

```

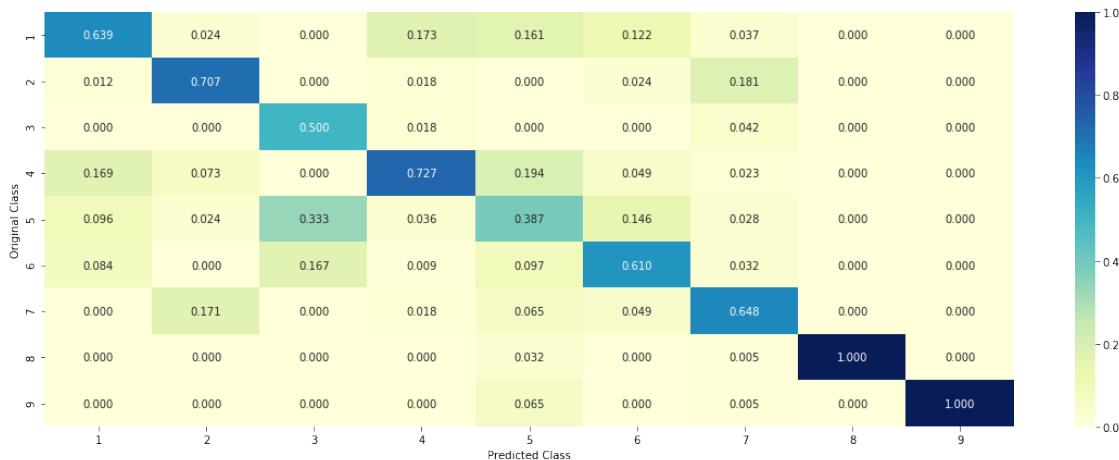
Log loss : 0.9781279782951176

Number of mis-classified points : 0.34962406015037595

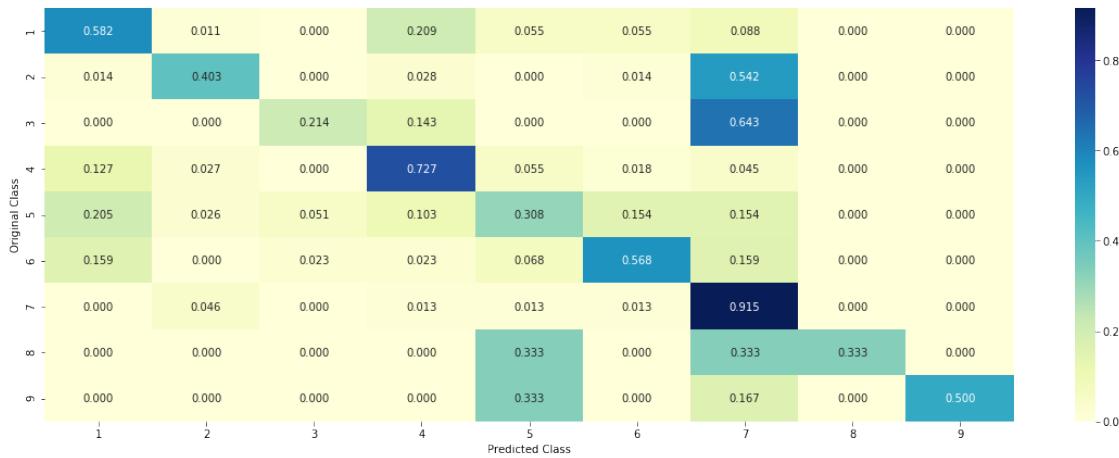
----- Confusion matrix -----



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



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



## Feature Importance:

```
In [59]: 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_tfidfCoding.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 important 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']))
```

```
In [94]: # from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=0.00013, penalty='l2', loss='log',
clf.fit(train_x_tfidfCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCo
```

```

print("Actual Class : ", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imptfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene']
Predicted Class : 7
Predicted Class Probabilities: [[0.0366 0.0545 0.0047 0.0087 0.0347 0.0278 0.8275 0.0026 0.003
Actual Class : 2
-----
1 Text feature [egfr] present in test data point [True]
7 Text feature [cells] present in test data point [True]
10 Text feature [map2k1] present in test data point [True]
27 Text feature [activation] present in test data point [True]
39 Text feature [akt] present in test data point [True]
44 Text feature [pdgfr] present in test data point [True]
51 Text feature [pi3k] present in test data point [True]
52 Text feature [signaling] present in test data point [True]
56 Text feature [ponatinib] present in test data point [True]
61 Text feature [fusion] present in test data point [True]
69 Text feature [mapk] present in test data point [True]
144 Text feature [pathway] present in test data point [True]
160 Text feature [cell] present in test data point [True]
169 Text feature [codon] present in test data point [True]
186 Text feature [mutant] present in test data point [True]
192 Text feature [fgfr4] present in test data point [True]
207 Text feature [braf] present in test data point [True]
249 Text feature [mutants] present in test data point [True]
284 Text feature [akt1] present in test data point [True]
313 Text feature [dovitinib] present in test data point [True]
333 Text feature [erk] present in test data point [True]
338 Text feature [leukemia] present in test data point [True]
352 Text feature [receptors] present in test data point [True]
353 Text feature [ligand] present in test data point [True]
374 Text feature [gefitinib] present in test data point [True]
389 Text feature [growth] present in test data point [True]
390 Text feature [lung] present in test data point [True]
393 Text feature [phosphorylation] present in test data point [True]
394 Text feature [fgfr] present in test data point [True]
397 Text feature [insertion] present in test data point [True]
489 Text feature [tumor] present in test data point [True]
Out of the top 500 features 31 are present in query point

```

```

In [96]: # from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=0.00013, penalty='l2', loss='log',
clf.fit(train_x_tfidfCoding,train_y)
test_point_index = 15
no_feature = 500

```

```

predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCoding)[test_point_index], 3))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imptfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index], test_df['Gene'])

```

Predicted Class : 6  
Predicted Class Probabilities: [[0.0305 0.0396 0.0079 0.0219 0.0216 0.7078 0.1629 0.0034 0.0041 ...]  
Actual Class : 6

---

37 Text feature [fedoratinib] present in test data point [True]  
51 Text feature [mek1] present in test data point [True]  
62 Text feature [ruxolitinib] present in test data point [True]  
105 Text feature [resistance] present in test data point [True]  
120 Text feature [mek] present in test data point [True]  
158 Text feature [1983f] present in test data point [True]  
159 Text feature [y931c] present in test data point [True]  
222 Text feature [shp2] present in test data point [True]  
264 Text feature [ic50] present in test data point [True]  
293 Text feature [substitutions] present in test data point [True]  
295 Text feature [values] present in test data point [True]  
298 Text feature [cyt] present in test data point [True]  
301 Text feature [models] present in test data point [True]  
338 Text feature [substitution] present in test data point [True]  
345 Text feature [sh2] present in test data point [True]  
348 Text feature [helix] present in test data point [True]  
366 Text feature [interaction] present in test data point [True]  
368 Text feature [structure] present in test data point [True]  
376 Text feature [387] present in test data point [True]  
381 Text feature [hydrogen] present in test data point [True]  
386 Text feature [steric] present in test data point [True]  
393 Text feature [screens] present in test data point [True]  
405 Text feature [pseudokinase] present in test data point [True]  
407 Text feature [ferm] present in test data point [True]  
422 Text feature [v617f] present in test data point [True]  
424 Text feature [conformation] present in test data point [True]  
436 Text feature [site] present in test data point [True]  
437 Text feature [residue] present in test data point [True]  
449 Text feature [hedgehog] present in test data point [True]  
457 Text feature [allosteric] present in test data point [True]  
469 Text feature [lestaurtinib] present in test data point [True]  
487 Text feature [atp] present in test data point [True]  
493 Text feature [dasatinib] present in test data point [True]

Out of the top 500 features 33 are present in query point

## Without Class Balancing

```
In [97]: import pickle
pickle.dump(final_results,open('final_results.p','wb'))

In [56]: import pickle
final_results = pickle.load(open('final_results.p','rb'))

In [67]: alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    #clf = LogisticRegression(C=i, class_weight='balanced', n_jobs=-1, solver='liblinear')
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability encoding
    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 = LogisticRegression(C=i, class_weight='balanced', n_jobs=-1, solver='liblinear')
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

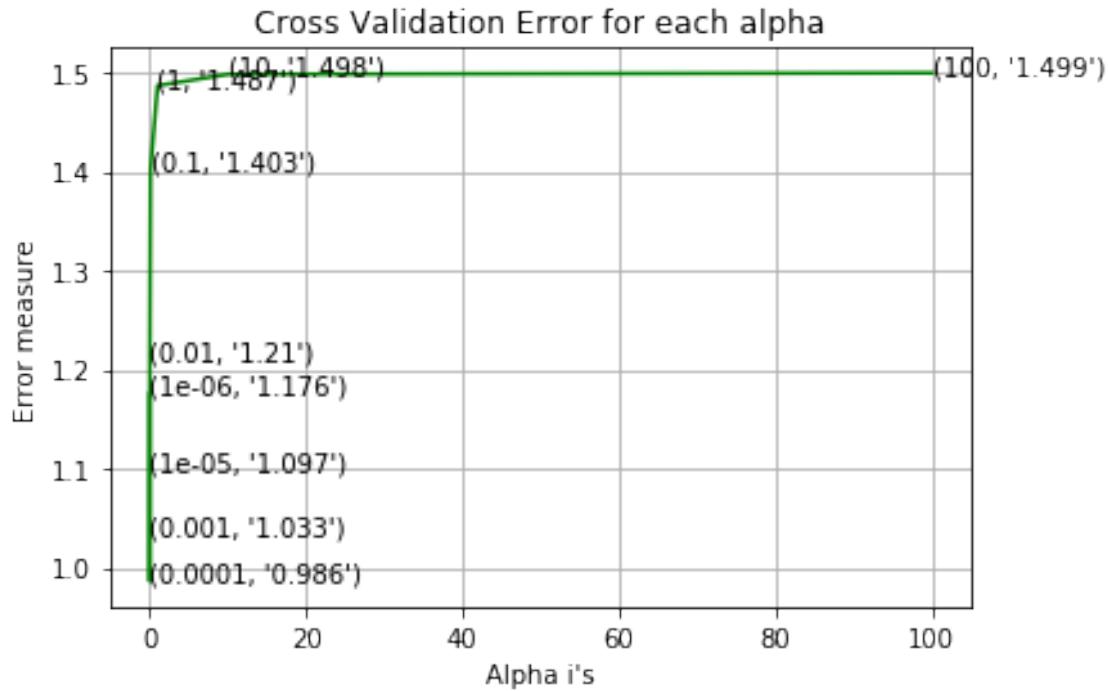
predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(predict_y,train_y))
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(predict_y,cv_y))
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(predict_y,test_y))

for alpha = 1e-06
Log Loss : 1.175515255332565
```

```

for alpha = 1e-05
Log Loss : 1.097339798349783
for alpha = 0.0001
Log Loss : 0.9863027829608768
for alpha = 0.001
Log Loss : 1.0325325018123297
for alpha = 0.01
Log Loss : 1.209840985474834
for alpha = 0.1
Log Loss : 1.4025742520294264
for alpha = 1
Log Loss : 1.4865819710985886
for alpha = 10
Log Loss : 1.4980402026217203
for alpha = 100
Log Loss : 1.4994313349514046

```



For values of best alpha = 0.0001 The train log loss is: 0.41858813182044086  
 For values of best alpha = 0.0001 The cross validation log loss is: 0.9863027829608768  
 For values of best alpha = 0.0001 The test log loss is: 0.9641046306444321

In [68]: `#alpha = [10 ** x for x in range(-6, 3)]  
alpha = np.random.uniform(0.00002, 0.0005, 20)`

```

alpha = np.round(alpha,7)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    #clf = LogisticRegression(C=i, class_weight='balanced', n_jobs=-1, solver='liblinear')
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

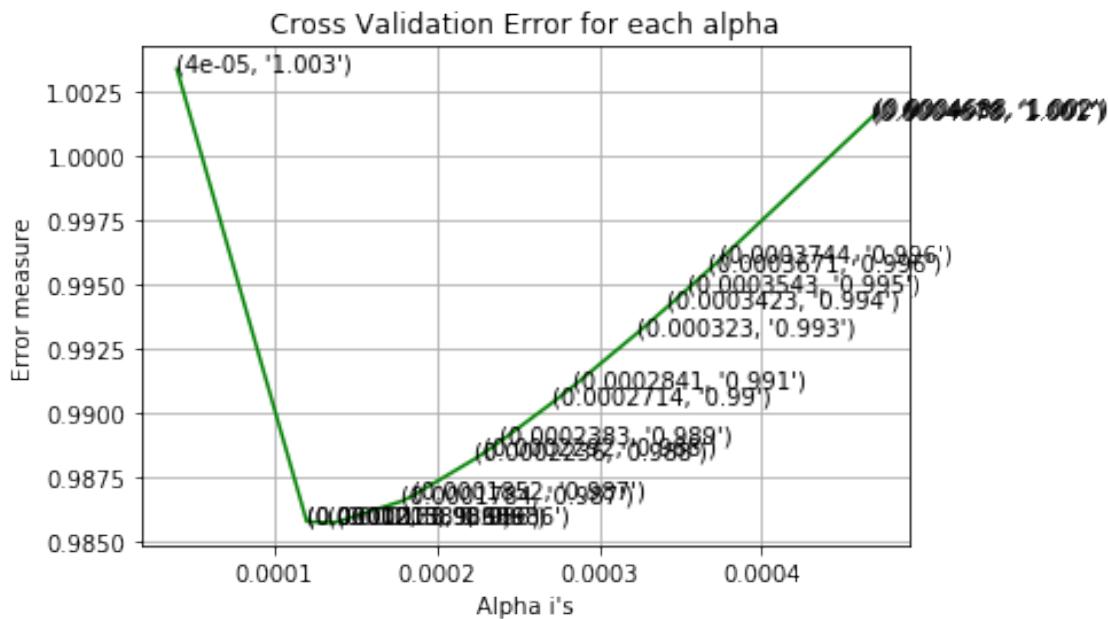
best_alpha = np.argmin(cv_log_error_array)
#clf = LogisticRegression(C=i, class_weight='balanced', n_jobs=-1, solver='liblinear')
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(cv_y, predict_y))
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(cv_y, predict_y))
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(test_y, predict_y))

for alpha = 4e-05
Log Loss : 1.0033755393687762
for alpha = 0.00012
Log Loss : 0.9857997551326462
for alpha = 0.0001205
Log Loss : 0.9857945886793088
for alpha = 0.000133
Log Loss : 0.9857539800519415

```

```
for alpha = 0.0001389
Log Loss : 0.9857863652073604
for alpha = 0.0001784
Log Loss : 0.9865858461443023
for alpha = 0.0001852
Log Loss : 0.9867965555674105
for alpha = 0.0002236
Log Loss : 0.9882340623936018
for alpha = 0.0002292
Log Loss : 0.9884713151810703
for alpha = 0.0002383
Log Loss : 0.9888687932201915
for alpha = 0.0002714
Log Loss : 0.9904205409741019
for alpha = 0.0002841
Log Loss : 0.9910530899172693
for alpha = 0.000323
Log Loss : 0.9930916298948316
for alpha = 0.0003423
Log Loss : 0.9941498839357433
for alpha = 0.0003543
Log Loss : 0.9948207937693316
for alpha = 0.0003671
Log Loss : 0.9955460777322497
for alpha = 0.0003744
Log Loss : 0.995963785274949
for alpha = 0.0004678
Log Loss : 1.001493029480626
for alpha = 0.0004688
Log Loss : 1.001553496079994
for alpha = 0.0004698
Log Loss : 1.001613981411409
```

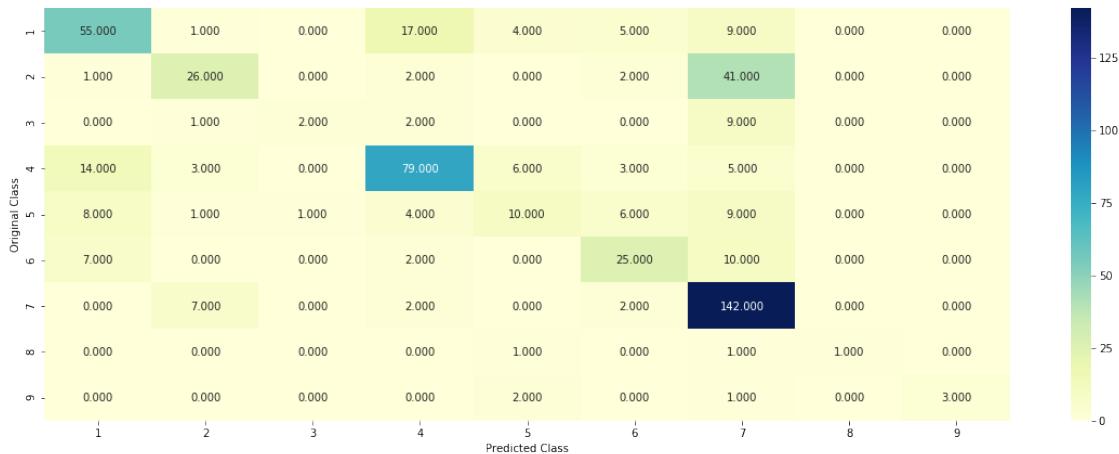


```
For values of best alpha = 0.000133 The train log loss is: 0.4302603251446899  
For values of best alpha = 0.000133 The cross validation log loss is: 0.9857539800519415  
For values of best alpha = 0.000133 The test log loss is: 0.9619229044469587
```

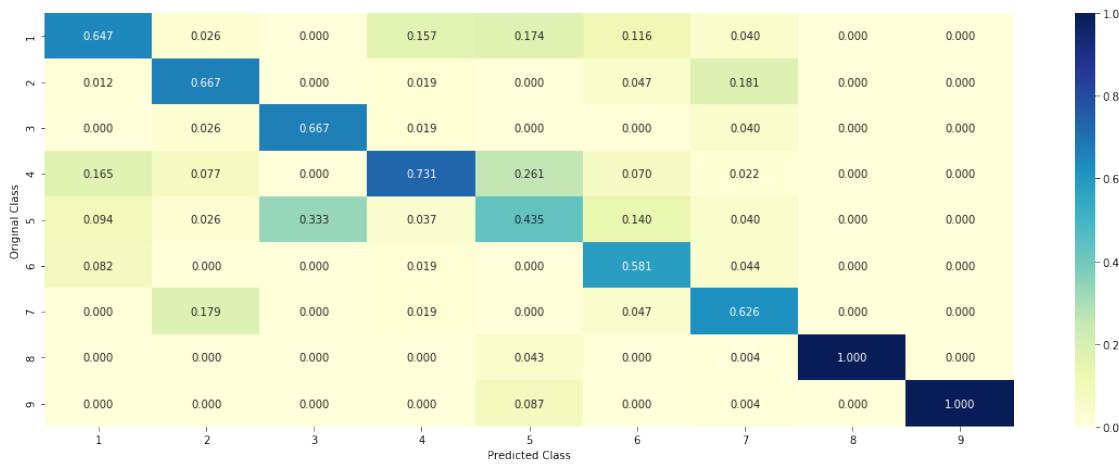
```
In [72]: #testing
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
sig_clf,temp = predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y, cv_x_tfidfCoding, cv_y)

list_data = []
list_data.append('Tf_Idf+LR+Class_Imbalance')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_tfidfCoding), eps=1e-15))
list_data.append('alpha = '+str(clf.alpha))
list_data.append(temp)
final_results.append(list_data)
```

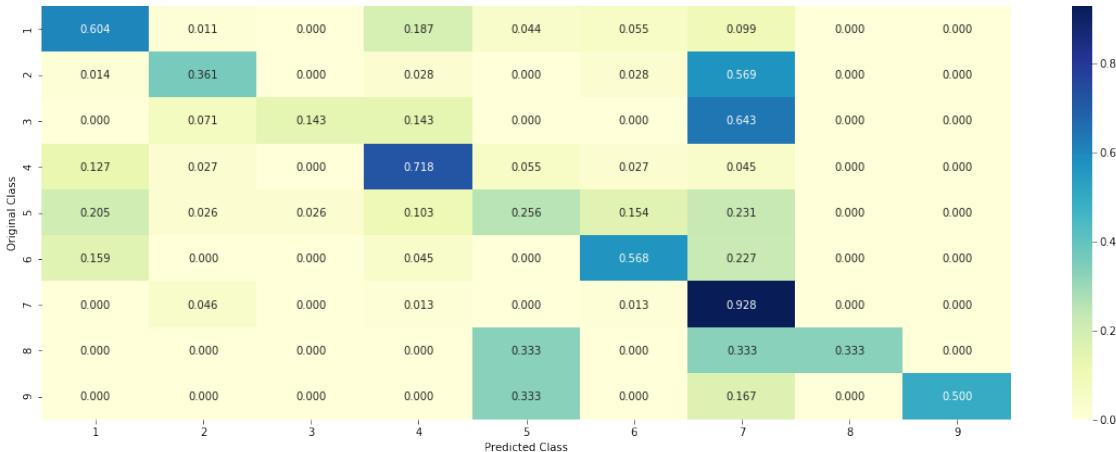
```
Log loss : 0.9857539800519415
Number of mis-classified points : 0.35526315789473684
----- Confusion matrix -----
```



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



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



```
In [77]: # from tabulate import tabulate
clf = SGDClassifier(alpha=0.000133, penalty='l2', loss='log', random_state=42)
clf.fit(train_x_tfidfCoding, train_y)
test_point_index = 51
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCoding[test_point_index])))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index], test_df['Gene'])
```

Predicted Class : 1  
Predicted Class Probabilities: [[0.4948 0.0181 0.0067 0.4503 0.0108 0.0056 0.0104 0.0016 0.0017 0.0011 0.0009 0.0008 0.0007 0.0006 0.0005 0.0004 0.0003 0.0002 0.0001]]  
Actual Class : 1

---

35 Text feature [brca2] present in test data point [True]  
81 Text feature [brct] present in test data point [True]  
199 Text feature [p53] present in test data point [True]  
299 Text feature [structure] present in test data point [True]  
309 Text feature [binding] present in test data point [True]  
348 Text feature [hydrophobic] present in test data point [True]  
364 Text feature [germ] present in test data point [True]  
369 Text feature [dna] present in test data point [True]  
375 Text feature [families] present in test data point [True]  
437 Text feature [transcription] present in test data point [True]  
438 Text feature [hook] present in test data point [True]  
443 Text feature [tryptophan] present in test data point [True]  
446 Text feature [repeats] present in test data point [True]  
447 Text feature [function] present in test data point [True]

```
469 Text feature [surface] present in test data point [True]
475 Text feature [residues] present in test data point [True]
Out of the top 500 features 16 are present in query point
```

Out of the top 500 features 26 are present in query point

#### 5.1.4.Linear Support Vector Machines

```
In [83]: alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', random_state=42, class_weight=None)
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42, class_weight=None)
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

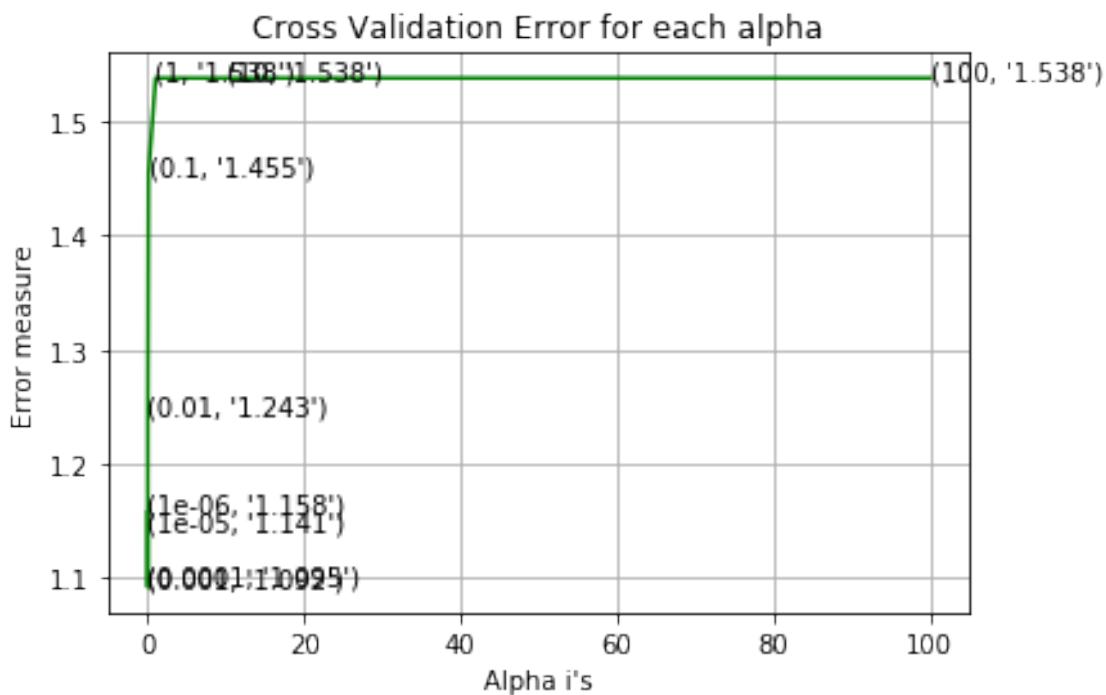
predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(cv_y,predict_y))
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(cv_y,predict_y))
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(test_y,predict_y))

for alpha = 1e-06
Log Loss : 1.1578340892722319
for alpha = 1e-05
Log Loss : 1.1413242262040262
for alpha = 0.0001
Log Loss : 1.0950112820315896
```

```

for alpha = 0.001
Log Loss : 1.0922430908138407
for alpha = 0.01
Log Loss : 1.242840356886754
for alpha = 0.1
Log Loss : 1.4546213897130242
for alpha = 1
Log Loss : 1.5379495421627543
for alpha = 10
Log Loss : 1.5379494572580412
for alpha = 100
Log Loss : 1.5379494641470106

```



For values of best alpha = 0.001 The train log loss is: 0.5103601191588758  
 For values of best alpha = 0.001 The cross validation log loss is: 1.0567716050260378  
 For values of best alpha = 0.001 The test log loss is: 1.0366619473897913

In [84]: `#alpha = [10 ** x for x in range(-6, 3)]  
alpha = np.random.uniform(0.0002,0.005,15)  
alpha = np.round(alpha,6)  
alpha.sort()  
cv_log_error_array = []  
for i in alpha:`

```

print("for alpha =", i)
clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', random_state=42, class_w...
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
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(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state...
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_...
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss i...
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los...

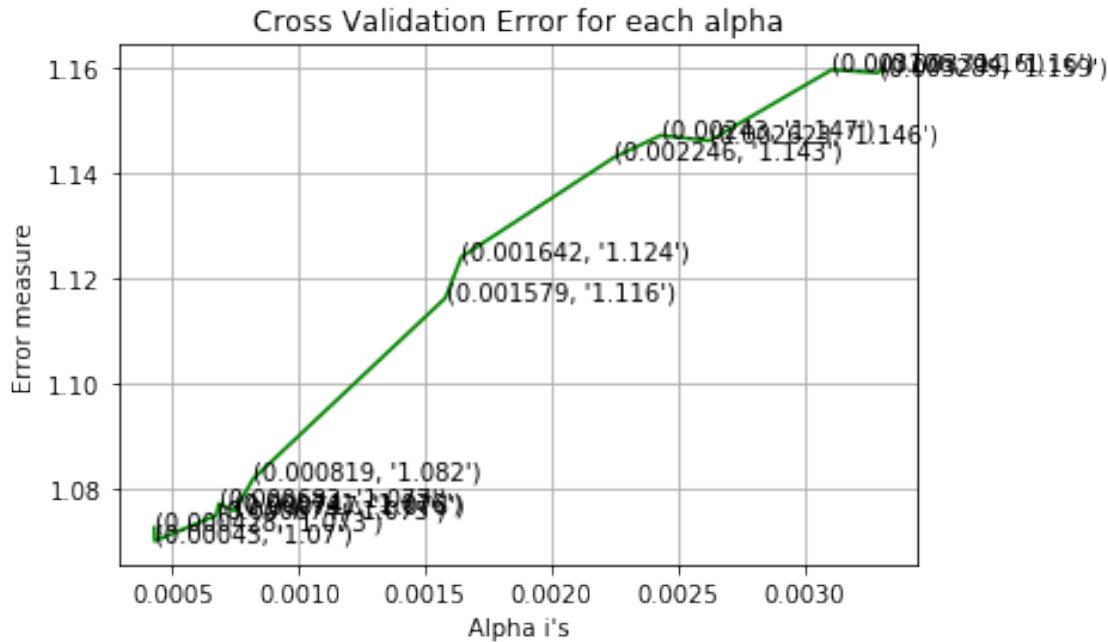
for alpha = 0.000428
Log Loss : 1.0726742908916034
for alpha = 0.00043
Log Loss : 1.0701549368950305
for alpha = 0.000674
Log Loss : 1.0750341788194118
for alpha = 0.000683
Log Loss : 1.0772133304070686
for alpha = 0.000741
Log Loss : 1.0759604762223642
for alpha = 0.000747
Log Loss : 1.0759472786578785
for alpha = 0.000819
Log Loss : 1.0819748888686447

```

```

for alpha = 0.001579
Log Loss : 1.116327163788507
for alpha = 0.001642
Log Loss : 1.1240919033462398
for alpha = 0.002246
Log Loss : 1.143022890862265
for alpha = 0.00243
Log Loss : 1.147193436556522
for alpha = 0.002623
Log Loss : 1.1461727187553723
for alpha = 0.003106
Log Loss : 1.1596619570336184
for alpha = 0.003289
Log Loss : 1.1590687899059888
for alpha = 0.003304
Log Loss : 1.1599593585445689

```



For values of best alpha = 0.00043 The train log loss is: 0.4918548669256699  
 For values of best alpha = 0.00043 The cross validation log loss is: 1.0701549368950305  
 For values of best alpha = 0.00043 The test log loss is: 1.0456041686131503

In [87]: *##testing*

```

clf = SGDClassifier(alpha=0.00043, penalty='l2', loss='hinge', random_state=42, class_w
sig_clf,temp = predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y, cv_x_tf

```

```

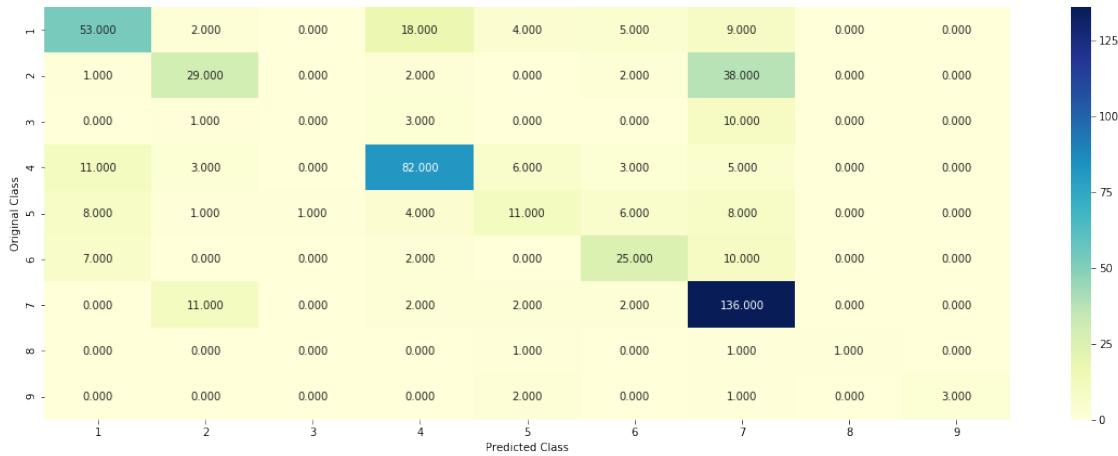
list_data = []
list_data.append('Tf_Idf+LSVC+Class_Balance')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_tfidfCoding), eps=1e-15))
list_data.append('alpha '+str(clf.alpha))
list_data.append(temp)
final_results.append(list_data)

```

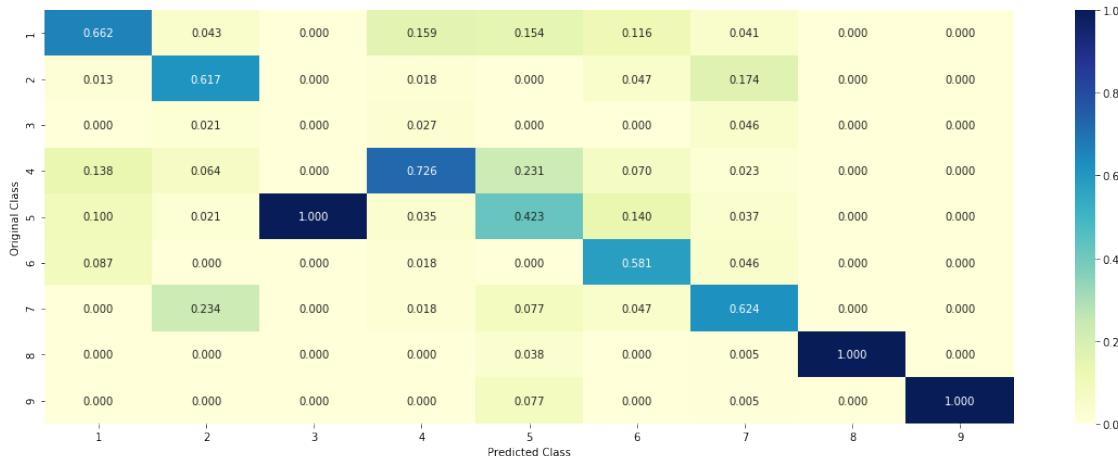
Log loss : 1.0701549368950305

Number of mis-classified points : 0.3609022556390977

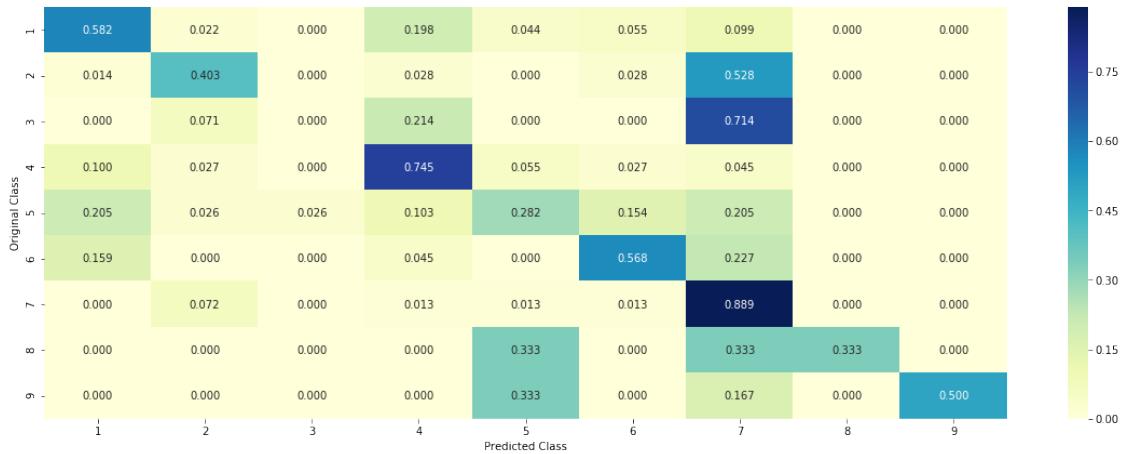
----- Confusion matrix -----



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



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



## Without class balancing

```
In [88]: #alpha = [10 ** x for x in range(-6, 3)]
alpha = np.random.uniform(0.0002,0.005,15)
alpha = np.round(alpha,6)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier( alpha=i, penalty='l2', loss='hinge', random_state=42)
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilities we use log-probability encoding
    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")
```

```

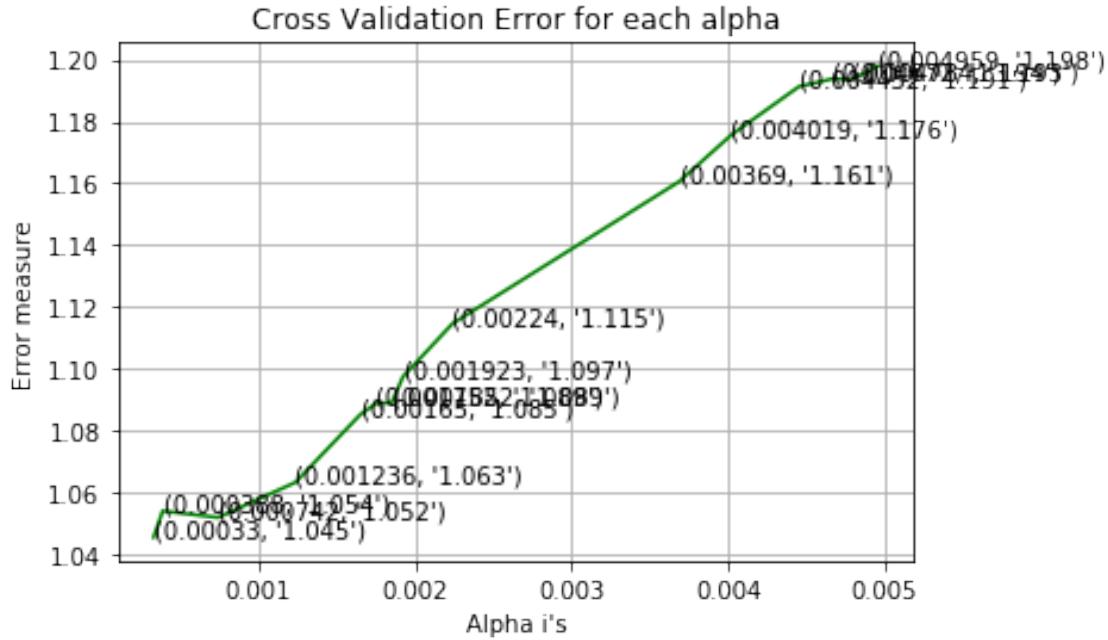
plt.show()

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(train_y, predict_y))
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(cv_y, predict_y))
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(test_y, predict_y))

for alpha = 0.00033
Log Loss : 1.0452986957578458
for alpha = 0.000388
Log Loss : 1.0538320052789394
for alpha = 0.000742
Log Loss : 1.0517249875096872
for alpha = 0.001236
Log Loss : 1.0630980267082315
for alpha = 0.00165
Log Loss : 1.0850377414491315
for alpha = 0.001752
Log Loss : 1.0886386524668055
for alpha = 0.001852
Log Loss : 1.0889501235855652
for alpha = 0.001923
Log Loss : 1.0973291926027333
for alpha = 0.00224
Log Loss : 1.1145411618286551
for alpha = 0.00369
Log Loss : 1.1606057830921932
for alpha = 0.004019
Log Loss : 1.1755733789022957
for alpha = 0.004452
Log Loss : 1.191274392366573
for alpha = 0.004672
Log Loss : 1.1938714288601233
for alpha = 0.004784
Log Loss : 1.193454657404159
for alpha = 0.004959
Log Loss : 1.1980033427984806

```



For values of best alpha = 0.00033 The train log loss is: 0.45334914495750833

For values of best alpha = 0.00033 The cross validation log loss is: 1.0452986957578458

For values of best alpha = 0.00033 The test log loss is: 1.0241909828559526

In [95]: `##testing`

```

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
sig_clf,temp = predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y, cv_x_tfidfCoding, cv_y)

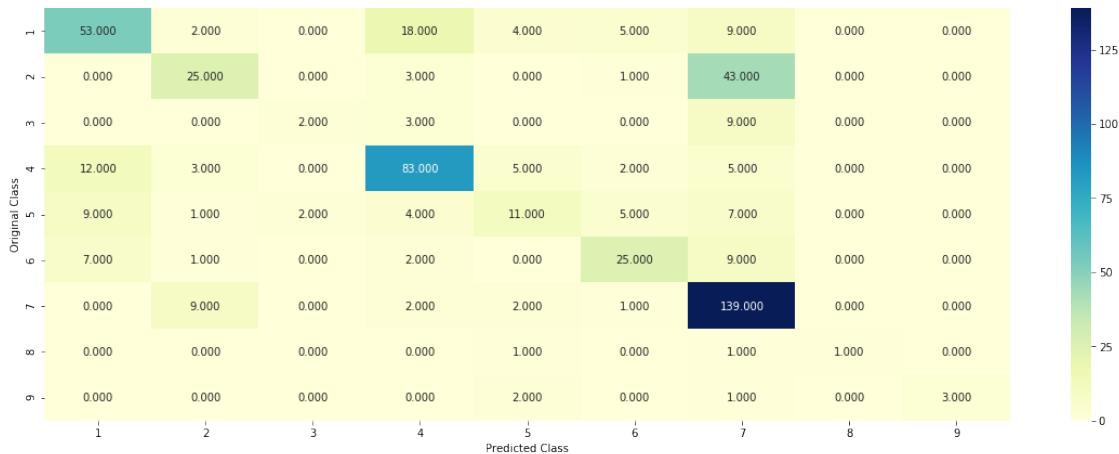
list_data = []
list_data.append('Tf_Idf+LSVC+Class_ImBalance')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_tfidfCoding), eps=1e-15))
list_data.append('alpha '+str(clf.alpha))
list_data.append(temp)
final_results.append(list_data)

```

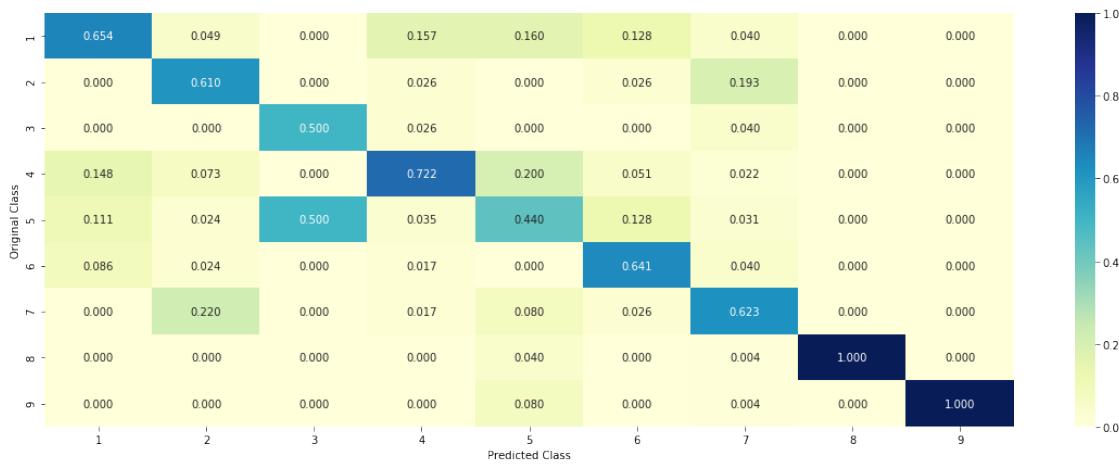
Log loss : 1.0452986957578458

Number of mis-classified points : 0.35714285714285715

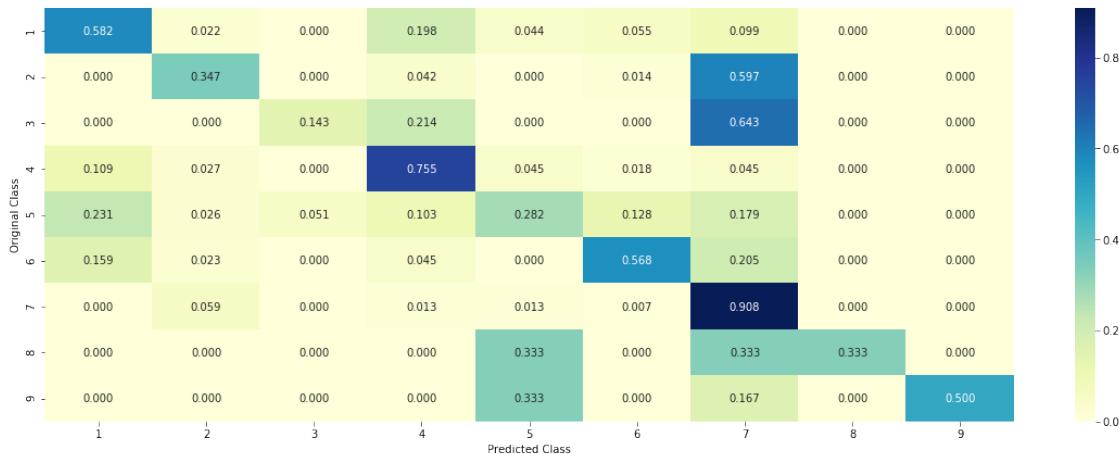
----- Confusion matrix -----



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



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



## Feature Importance

```
In [96]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_tfidfCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCoding[test_point_index])))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'])
```

Predicted Class : 7

Predicted Class Probabilities: [[0.0366 0.0488 0.0074 0.0325 0.0265 0.0166 0.8263 0.0032 0.0021]]

Actual Class : 2

-----

- 0 Text feature [egfr] present in test data point [True]
- 1 Text feature [cells] present in test data point [True]
- 20 Text feature [map2k1] present in test data point [True]
- 75 Text feature [fusion] present in test data point [True]
- 79 Text feature [activation] present in test data point [True]
- 83 Text feature [signaling] present in test data point [True]
- 86 Text feature [mapk] present in test data point [True]
- 209 Text feature [ponatinib] present in test data point [True]
- 213 Text feature [pi3k] present in test data point [True]
- 215 Text feature [mutants] present in test data point [True]
- 223 Text feature [mutant] present in test data point [True]
- 227 Text feature [codon] present in test data point [True]

```

230 Text feature [cell] present in test data point [True]
238 Text feature [akt] present in test data point [True]
252 Text feature [domain] present in test data point [True]
270 Text feature [pdgfra] present in test data point [True]
271 Text feature [pathway] present in test data point [True]
285 Text feature [braf] present in test data point [True]
Out of the top 500 features 18 are present in query point

```

```

In [97]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
          clf.fit(train_x_tfidfCoding, train_y)
          test_point_index = 15
# test_point_index = 100
          no_feature = 500
          predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
          print("Predicted Class :", predicted_cls[0])
          print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCoding)[test_point_index]))
          print("Actual Class :", test_y[test_point_index])
          indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
          print("-"*50)
          get_imptfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index], test_df['Gene'])

```

```

Predicted Class : 6
Predicted Class Probabilities: [[0.0401 0.0322 0.0162 0.036  0.0477 0.5963 0.2206 0.0047 0.0061
                                  0.0011 0.0001 0.0001 0.0001 0.0001 0.0001 0.0001 0.0001 0.0001]
Actual Class : 6
-----
43 Text feature [fedoratinib] present in test data point [True]
79 Text feature [ruxolitinib] present in test data point [True]
112 Text feature [mek1] present in test data point [True]
115 Text feature [resistance] present in test data point [True]
120 Text feature [mek] present in test data point [True]
172 Text feature [shp2] present in test data point [True]
175 Text feature [y931c] present in test data point [True]
177 Text feature [1983f] present in test data point [True]
203 Text feature [ic50] present in test data point [True]
218 Text feature [values] present in test data point [True]
234 Text feature [mutation] present in test data point [True]
239 Text feature [kinase] present in test data point [True]
243 Text feature [interaction] present in test data point [True]
302 Text feature [models] present in test data point [True]
307 Text feature [sh2] present in test data point [True]
313 Text feature [substitutions] present in test data point [True]
317 Text feature [substitution] present in test data point [True]
331 Text feature [residue] present in test data point [True]
333 Text feature [helix] present in test data point [True]
336 Text feature [structure] present in test data point [True]
338 Text feature [cyt] present in test data point [True]
351 Text feature [site] present in test data point [True]

```

```

352 Text feature [pi3k] present in test data point [True]
354 Text feature [atp] present in test data point [True]
364 Text feature [dasatinib] present in test data point [True]
365 Text feature [v617f] present in test data point [True]
368 Text feature [direct] present in test data point [True]
389 Text feature [conformation] present in test data point [True]
393 Text feature [ferm] present in test data point [True]
397 Text feature [steric] present in test data point [True]
399 Text feature [residues] present in test data point [True]
402 Text feature [387] present in test data point [True]
404 Text feature [associated] present in test data point [True]
406 Text feature [receptor] present in test data point [True]
414 Text feature [49] present in test data point [True]
418 Text feature [binding] present in test data point [True]
421 Text feature [phosphorylated] present in test data point [True]
423 Text feature [inhibitors] present in test data point [True]
437 Text feature [allosteric] present in test data point [True]
442 Text feature [modeling] present in test data point [True]
447 Text feature [identified] present in test data point [True]
456 Text feature [hydrogen] present in test data point [True]
463 Text feature [pseudokinase] present in test data point [True]
465 Text feature [sequencing] present in test data point [True]
470 Text feature [nm] present in test data point [True]
473 Text feature [predicted] present in test data point [True]
475 Text feature [screens] present in test data point [True]
480 Text feature [survival] present in test data point [True]
491 Text feature [luciferase] present in test data point [True]
495 Text feature [family] present in test data point [True]
496 Text feature [substrate] present in test data point [True]
Out of the top 500 features 51 are present in query point

```

### 5.1.5.Random Forest Classifier

```

In [98]: alpha = [100,200,500,1000,2000]
          max_depth = [5,10,20]
          cv_log_error_array = []
          for i in alpha:
              for j in max_depth:
                  print("for n_estimators =", i,"and max depth = ", j)
                  clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, r
                  clf.fit(train_x_tfidfCoding, train_y)
                  sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
                  sig_clf.fit(train_x_tfidfCoding, train_y)
                  sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
                  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/3)], criterion='gini',
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best estimator = ', alpha[int(best_alpha/3)], 'depth = ',alpha[int(best_alpha/3)],'Log Loss : ', log_loss(y_train,predict_y))
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best estimator = ', alpha[int(best_alpha/3)], 'depth = ',alpha[int(best_alpha/3)],'Log Loss : ', log_loss(y_cv,predict_y))
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best estimator = ', alpha[int(best_alpha/3)], 'depth = ',alpha[int(best_alpha/3)],'Log Loss : ', log_loss(y_test,predict_y))

for n_estimators = 100 and max depth =  5
Log Loss : 1.189099610951161
for n_estimators = 100 and max depth =  10
Log Loss : 1.070456109583718
for n_estimators = 100 and max depth =  20
Log Loss : 1.0510405842467523
for n_estimators = 200 and max depth =  5
Log Loss : 1.1664297094864557
for n_estimators = 200 and max depth =  10
Log Loss : 1.062802204066856
for n_estimators = 200 and max depth =  20
Log Loss : 1.0446907622524244
for n_estimators = 500 and max depth =  5
Log Loss : 1.1589327061746937
for n_estimators = 500 and max depth =  10
Log Loss : 1.059067199346161
for n_estimators = 500 and max depth =  20
Log Loss : 1.0411817888758041
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1587031311205727
for n_estimators = 1000 and max depth =  10
Log Loss : 1.0568596458986803
for n_estimators = 1000 and max depth =  20
Log Loss : 1.0411817888758041

```

```

Log Loss : 1.040570741619973
for n_estimators = 2000 and max depth = 5
Log Loss : 1.1583085642917665
for n_estimators = 2000 and max depth = 10
Log Loss : 1.0540971127819687
for n_estimators = 2000 and max depth = 20
Log Loss : 1.0387118911285929
For values of best estimator = 2000 depth = 500 The train log loss is: 0.48774521894137096
For values of best estimator = 2000 depth = 500 The cross validation log loss is: 1.038711894137096
For values of best estimator = 2000 depth = 500 The test log loss is: 1.0879703249226977

```

```

In [99]: #test
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/3)], criterion='gini',
sig_clf,temp=predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y,cv_x_tfidfCoding))

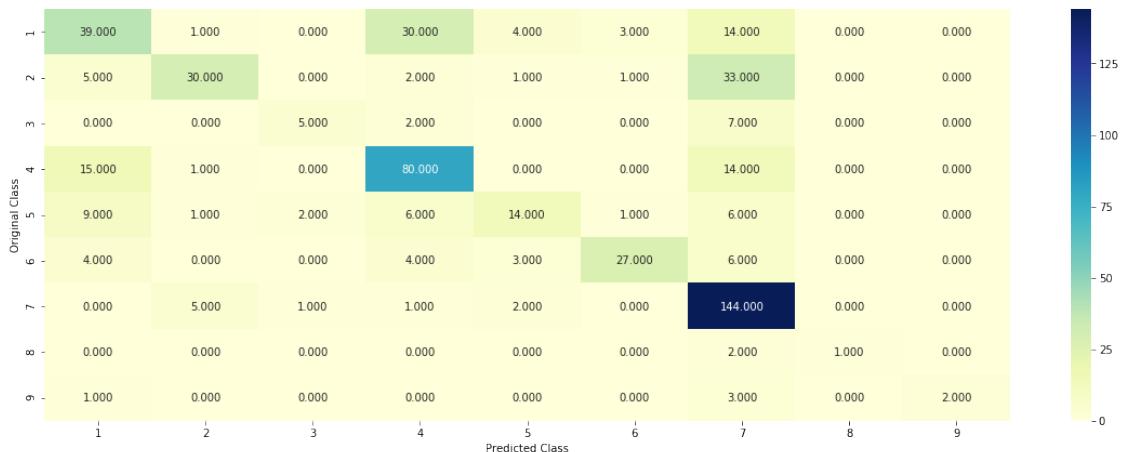
list_data = []
list_data.append('Tf_Idf+RF')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_tfidfCoding), eps=1e-15))
list_data.append('estimators = '+str(clf.estimators_)+ ' ' + 'depth = '+str(clf.max_depth))
list_data.append(temp)
final_results.append(list_data)

```

```

Log loss : 1.0387118911285929
Number of mis-classified points : 0.35714285714285715
----- Confusion matrix -----

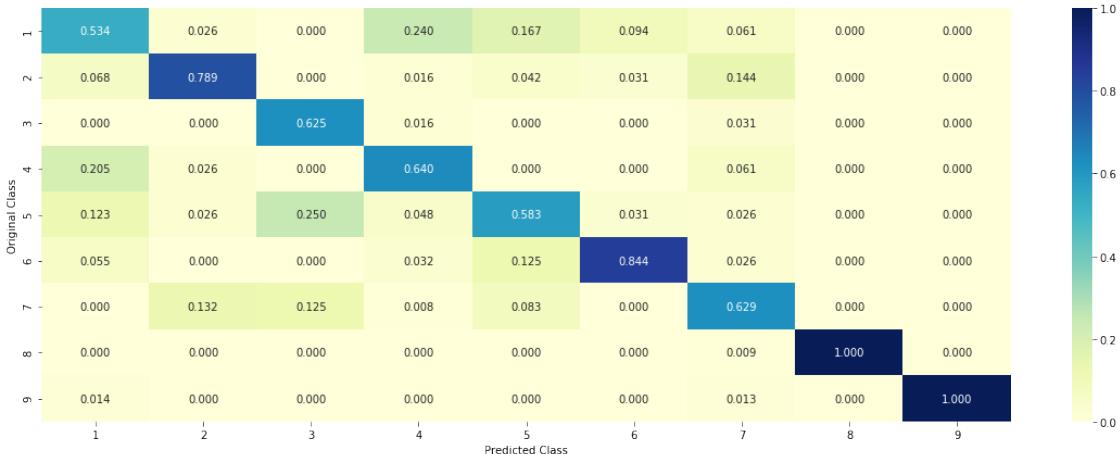
```



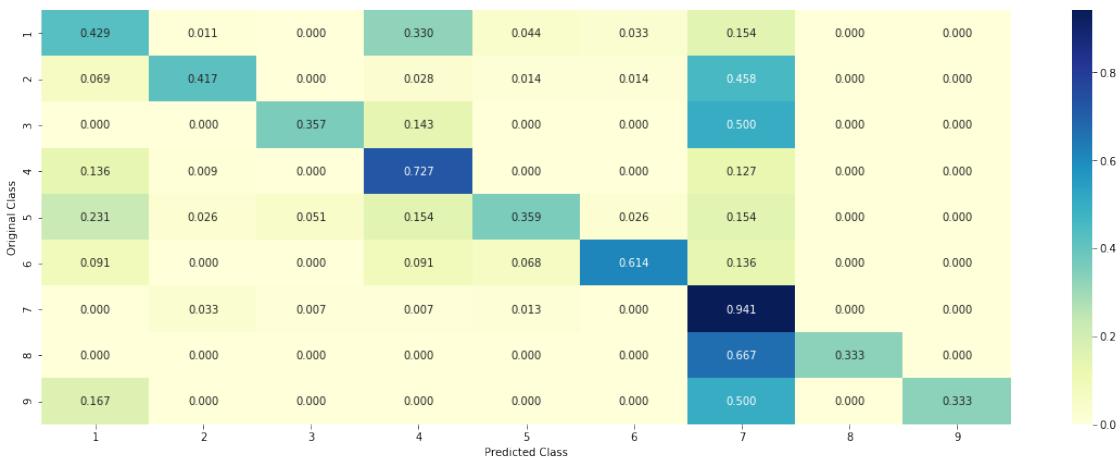
```

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

```



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



```
In [68]: # test_point_index = 10
clf = RandomForestClassifier(n_estimators=200, criterion='gini', max_depth=20, random_
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCoding[test_point_index]), 2))
print("Actual Class :", test_y[test_point_index])
```

```

    indices = np.argsort(-clf.feature_importances_)
    print("-"*50)
    get_imppfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index], test_
Predicted Class : 2
Predicted Class Probabilities: [[0.0985 0.3665 0.0171 0.0657 0.0454 0.0453 0.3501 0.005 0.006
Actual Class : 2
-----
0 Text feature [kinase] present in test data point [True]
2 Text feature [phosphorylation] present in test data point [True]
3 Text feature [activation] present in test data point [True]
5 Text feature [inhibitors] present in test data point [True]
6 Text feature [loss] present in test data point [True]
8 Text feature [function] present in test data point [True]
9 Text feature [oncogenic] present in test data point [True]
11 Text feature [growth] present in test data point [True]
12 Text feature [inhibitor] present in test data point [True]
13 Text feature [variants] present in test data point [True]
14 Text feature [drug] present in test data point [True]
15 Text feature [nonsense] present in test data point [True]
16 Text feature [erk] present in test data point [True]
17 Text feature [kinases] present in test data point [True]
18 Text feature [patients] present in test data point [True]
21 Text feature [tyrosine] present in test data point [True]
22 Text feature [trials] present in test data point [True]
23 Text feature [resistance] present in test data point [True]
24 Text feature [efficacy] present in test data point [True]
25 Text feature [missense] present in test data point [True]
26 Text feature [signaling] present in test data point [True]
27 Text feature [treatment] present in test data point [True]
29 Text feature [dose] present in test data point [True]
30 Text feature [receptor] present in test data point [True]
32 Text feature [cells] present in test data point [True]
33 Text feature [therapeutic] present in test data point [True]
34 Text feature [protein] present in test data point [True]
35 Text feature [proteins] present in test data point [True]
41 Text feature [suppressor] present in test data point [True]
52 Text feature [mice] present in test data point [True]
53 Text feature [treated] present in test data point [True]
54 Text feature [clinical] present in test data point [True]
55 Text feature [pathways] present in test data point [True]
57 Text feature [mutants] present in test data point [True]
60 Text feature [lines] present in test data point [True]
62 Text feature [inhibition] present in test data point [True]
63 Text feature [cell] present in test data point [True]
65 Text feature [factor] present in test data point [True]
68 Text feature [ligand] present in test data point [True]
70 Text feature [recently] present in test data point [True]

```

```

71 Text feature [therapy] present in test data point [True]
73 Text feature [akt] present in test data point [True]
75 Text feature [soft] present in test data point [True]
76 Text feature [pten] present in test data point [True]
78 Text feature [oncogene] present in test data point [True]
85 Text feature [receptors] present in test data point [True]
86 Text feature [proliferation] present in test data point [True]
87 Text feature [downstream] present in test data point [True]
89 Text feature [trial] present in test data point [True]
90 Text feature [interaction] present in test data point [True]
92 Text feature [dna] present in test data point [True]
94 Text feature [sequence] present in test data point [True]
98 Text feature [egfr] present in test data point [True]
99 Text feature [pathway] present in test data point [True]
Out of the top 100 features 54 are present in query point

```

```

In [73]: test_point_index = 70
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCoding[test_point_index]), 3))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_imptfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index], test_y[test_point_index])

```

```

Predicted Class : 7
Predicted Class Probabilities: [[0.137  0.1124  0.0182  0.3098  0.0452  0.0394  0.3253  0.0053  0.0071
                                 0.0015  0.0001]
Actual Class : 1
-----
```

```

0 Text feature [kinase] present in test data point [True]
1 Text feature [activating] present in test data point [True]
2 Text feature [phosphorylation] present in test data point [True]
3 Text feature [activation] present in test data point [True]
4 Text feature [activated] present in test data point [True]
5 Text feature [inhibitors] present in test data point [True]
6 Text feature [loss] present in test data point [True]
7 Text feature [activate] present in test data point [True]
8 Text feature [function] present in test data point [True]
9 Text feature [oncogenic] present in test data point [True]
11 Text feature [growth] present in test data point [True]
12 Text feature [inhibitor] present in test data point [True]
14 Text feature [drug] present in test data point [True]
17 Text feature [kinases] present in test data point [True]
18 Text feature [patients] present in test data point [True]
19 Text feature [inhibited] present in test data point [True]
23 Text feature [resistance] present in test data point [True]
```

```

25 Text feature [missense] present in test data point [True]
26 Text feature [signaling] present in test data point [True]
27 Text feature [treatment] present in test data point [True]
29 Text feature [dose] present in test data point [True]
31 Text feature [assays] present in test data point [True]
32 Text feature [cells] present in test data point [True]
33 Text feature [therapeutic] present in test data point [True]
34 Text feature [protein] present in test data point [True]
35 Text feature [proteins] present in test data point [True]
37 Text feature [expressing] present in test data point [True]
38 Text feature [phospho] present in test data point [True]
39 Text feature [resistant] present in test data point [True]
46 Text feature [survival] present in test data point [True]
53 Text feature [treated] present in test data point [True]
54 Text feature [clinical] present in test data point [True]
55 Text feature [pathways] present in test data point [True]
57 Text feature [mutants] present in test data point [True]
60 Text feature [lines] present in test data point [True]
61 Text feature [functional] present in test data point [True]
62 Text feature [inhibition] present in test data point [True]
63 Text feature [cell] present in test data point [True]
64 Text feature [length] present in test data point [True]
65 Text feature [factor] present in test data point [True]
70 Text feature [recently] present in test data point [True]
71 Text feature [therapy] present in test data point [True]
81 Text feature [whether] present in test data point [True]
86 Text feature [proliferation] present in test data point [True]
87 Text feature [downstream] present in test data point [True]
88 Text feature [phosphorylated] present in test data point [True]
90 Text feature [interaction] present in test data point [True]
92 Text feature [dna] present in test data point [True]
94 Text feature [sequence] present in test data point [True]
97 Text feature [kit] present in test data point [True]
99 Text feature [pathway] present in test data point [True]
Out of the top 100 features 51 are present in query point

```

### 5.1.7.Stack the models

```

In [75]: clf1 = SGDClassifier(alpha=0.00013, penalty='l2', loss='log', class_weight='balanced')
clf1.fit(train_x_tfidfCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

clf2 = SGDClassifier(alpha=0.00033, penalty='l2', loss='hinge', class_weight='balanced')
clf2.fit(train_x_tfidfCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")

```

```

clf3 = KNeighborsClassifier(n_neighbors=5)
clf3.fit(train_x_tfidfCoding, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")

sig_clf1.fit(train_x_tfidfCoding, train_y)
print("Logistic Regression : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_tfidfCoding))))
sig_clf2.fit(train_x_tfidfCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(cv_x_tfidfCoding))))
sig_clf3.fit(train_x_tfidfCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_tfidfCoding))))
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_classifier=lr)
    sclf.fit(train_x_tfidfCoding, train_y)
    print("Stacking Classifier : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sclf.predict_proba(cv_x_tfidfCoding))))
    if best_alpha > log_error:
        best_alpha = log_error

Logistic Regression : Log Loss: 0.99
Support vector machines : Log Loss: 1.07
Naive Bayes : Log Loss: 1.11
-----
Stacking Classifier : for the value of alpha: 0.000100 Log Loss: 2.174
Stacking Classifier : for the value of alpha: 0.001000 Log Loss: 2.001
Stacking Classifier : for the value of alpha: 0.010000 Log Loss: 1.424
Stacking Classifier : for the value of alpha: 0.100000 Log Loss: 1.072
Stacking Classifier : for the value of alpha: 1.000000 Log Loss: 1.199
Stacking Classifier : for the value of alpha: 10.000000 Log Loss: 1.594

```

```

In [76]: alpha = np.random.uniform(0.005,0.5,10)
alpha = np.round(alpha,5)
alpha.sort()
#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_classifier=lr)
    sclf.fit(train_x_tfidfCoding, train_y)
    print("Stacking Classifier : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sclf.predict_proba(cv_x_tfidfCoding))))
    log_error = log_loss(cv_y, sclf.predict_proba(cv_x_tfidfCoding))
    if best_alpha > log_error:
        best_alpha = log_error

```

Stacking Classifier : for the value of alpha: 0.110140 Log Loss: 1.069

```
Stacking Classifier : for the value of alpha: 0.118030 Log Loss: 1.068
Stacking Classifier : for the value of alpha: 0.121800 Log Loss: 1.067
Stacking Classifier : for the value of alpha: 0.127820 Log Loss: 1.067
Stacking Classifier : for the value of alpha: 0.244560 Log Loss: 1.077
Stacking Classifier : for the value of alpha: 0.273980 Log Loss: 1.083
Stacking Classifier : for the value of alpha: 0.315030 Log Loss: 1.090
Stacking Classifier : for the value of alpha: 0.351380 Log Loss: 1.097
Stacking Classifier : for the value of alpha: 0.369550 Log Loss: 1.101
Stacking Classifier : for the value of alpha: 0.411730 Log Loss: 1.109
```

```
In [78]: #testing
lr = LogisticRegression(C=0.121800)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr)
sclf.fit(train_x_tfidfCoding, train_y)

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

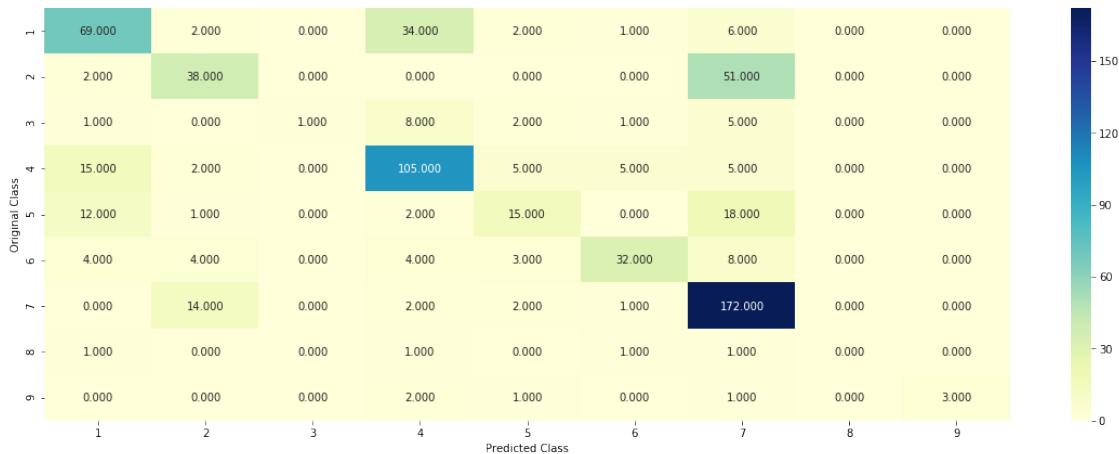
log_error = log_loss(cv_y, sclf.predict_proba(cv_x_tfidfCoding))
print("Log loss (CV) on the stacking classifier :",log_error)

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

print("Number of missclassified point :", np.count_nonzero((sclf.predict(test_x_tfidfCoding)- test_y)))
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_tfidfCoding))

list_data = []
list_data.append('Tf_Idf+Stackig(LR+KNN+LSVC)')
list_data.append(log_loss(y_train, sclf.predict_proba(train_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_cv, sclf.predict_proba(cv_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_test, sclf.predict_proba(test_x_tfidfCoding), eps=1e-15))
list_data.append('C = '+str(0.121800))
list_data.append(np.count_nonzero((sclf.predict(test_x_tfidfCoding)- test_y))/test_y)
final_results.append(list_data)

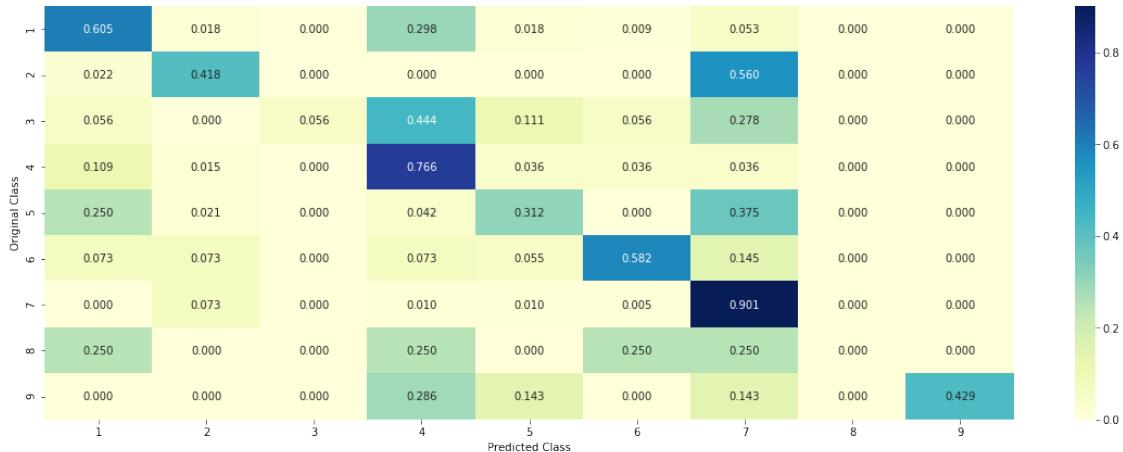
Log loss (train) on the stacking classifier : 0.3642425183281245
Log loss (CV) on the stacking classifier : 1.0671954448305458
Log loss (test) on the stacking classifier : 1.0359894975303312
Number of missclassified point : 0.3458646616541353
----- Confusion matrix -----
```



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



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



### 5.1.8. Maximum Voting classifier

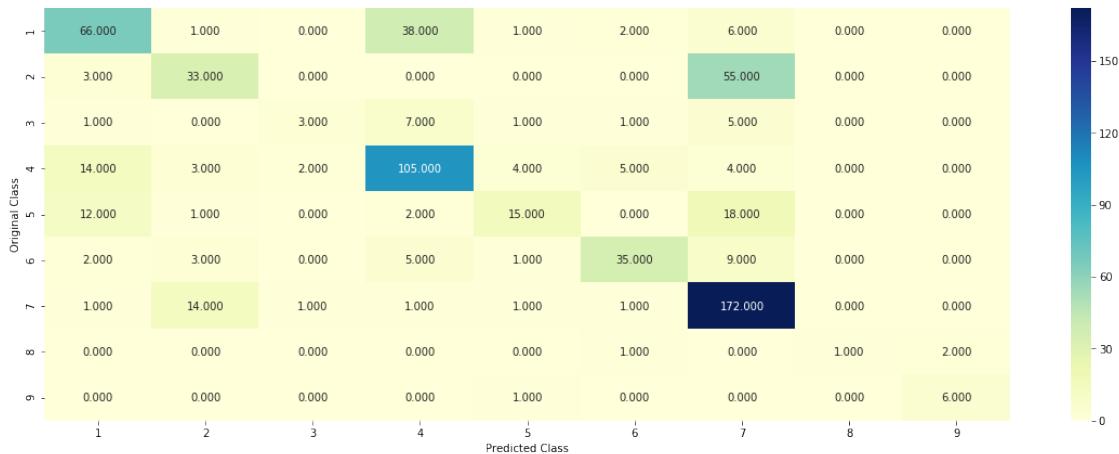
```
In [60]: #Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
clf1 = SGDClassifier(alpha=0.00013, penalty='l2', loss='log', class_weight='balanced')
clf1.fit(train_x_tfidfCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

clf2 = SGDClassifier(alpha=0.00033, penalty='l2', loss='hinge', class_weight='balanced')
clf2.fit(train_x_tfidfCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")

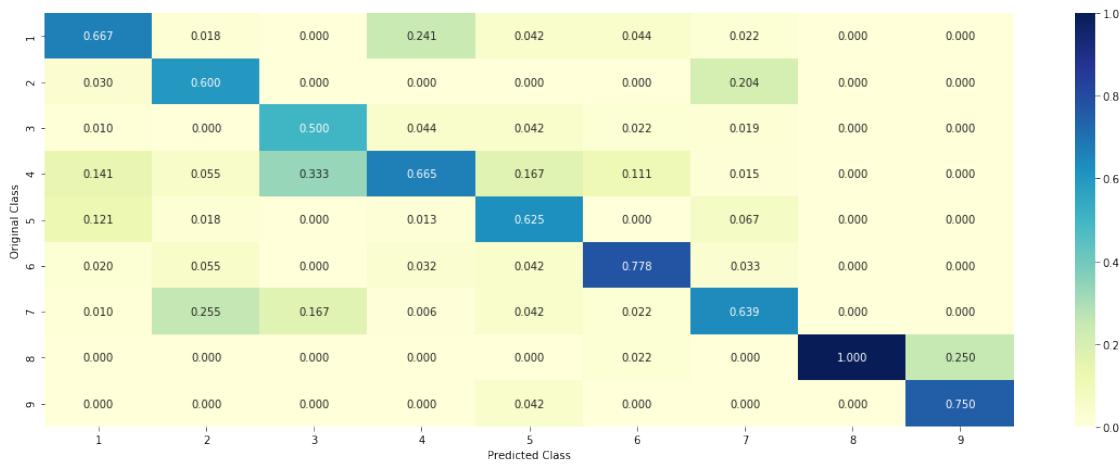
clf3 = KNeighborsClassifier(n_neighbors=5)
clf3.fit(train_x_tfidfCoding, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")

vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)])
vclf.fit(train_x_tfidfCoding, train_y)
print("Log loss (train) on the VotingClassifier : ", log_loss(train_y, vclf.predict_proba(train_x_tfidfCoding)))
print("Log loss (CV) on the VotingClassifier : ", log_loss(cv_y, vclf.predict_proba(cv_x_tfidfCoding)))
print("Log loss (test) on the VotingClassifier : ", log_loss(test_y, vclf.predict_proba(test_x_tfidfCoding)))
print("Number of missclassified point : ", np.count_nonzero(vclf.predict(test_x_tfidfCoding) != test_y))
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_tfidfCoding))

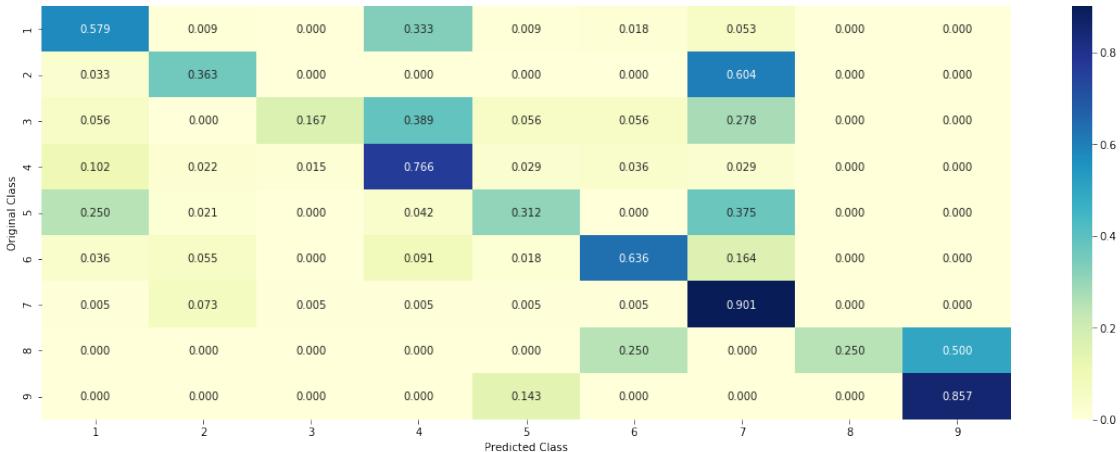
Log loss (train) on the VotingClassifier : 0.5556191761286373
Log loss (CV) on the VotingClassifier : 1.0255114559556837
Log loss (test) on the VotingClassifier : 1.0127488092039683
Number of missclassified point : 0.3443609022556391
----- Confusion matrix -----
```



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



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



## 6. Top 1000 Tf-Idf features

```
In [60]: # building a CountVectorizer with all the words that occurred minimum 3 times in train
text_vectorizer = TfidfVectorizer(min_df=3)
train_text_feature_tfidfCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# we use the same vectorizer that was trained on train data
test_text_feature_tfidfCoding = text_vectorizer.transform(test_df['TEXT'])
# we use the same vectorizer that was trained on train data
cv_text_feature_tfidfCoding = text_vectorizer.transform(cv_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of words)
train_text_fea_counts = train_text_feature_tfidfCoding.sum(axis=0).A1

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it occurred
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))

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

Total number of unique words in train data : 53678

```
In [61]: tfidf_transp = train_text_feature_tfidfCoding.T
feature_max = {}
for i in range(len(train_text_features)):
    feature_max[train_text_features[i]] = tfidf_transp[i].max()
```

```
In [62]: top1000_idx = np.argsort(-np.array(list(feature_max.values())))[0:1000]
```

```
In [63]: train_text_feature_tfidfCoding = train_text_feature_tfidfCoding[:,top1000_idx]
test_text_feature_tfidfCoding = test_text_feature_tfidfCoding[:,top1000_idx]
cv_text_feature_tfidfCoding = cv_text_feature_tfidfCoding[:,top1000_idx]
```

```
In [64]: train_gene_var_tfidfCoding = hstack((train_gene_feature_tfidfCoding,train_variation_featu
test_gene_var_tfidfCoding = hstack((test_gene_feature_tfidfCoding,test_variation_featu
cv_gene_var_tfidfCoding = hstack((cv_gene_feature_tfidfCoding, cv_variation_feature_tf
train_x_tfidfCoding = hstack((train_gene_var_tfidfCoding, train_text_feature_tfidfCoding))
train_y = np.array(list(train_df['Class']))

test_x_tfidfCoding = hstack((test_gene_var_tfidfCoding, test_text_feature_tfidfCoding))
test_y = np.array(list(test_df['Class']))

cv_x_tfidfCoding = hstack((cv_gene_var_tfidfCoding, cv_text_feature_tfidfCoding)).tocs
cv_y = np.array(list(cv_df['Class']))
```

## 6.1 Machine Learning Models

### 6.1.1 Naive Bayes

```
In [84]: 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_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability encoding
    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()

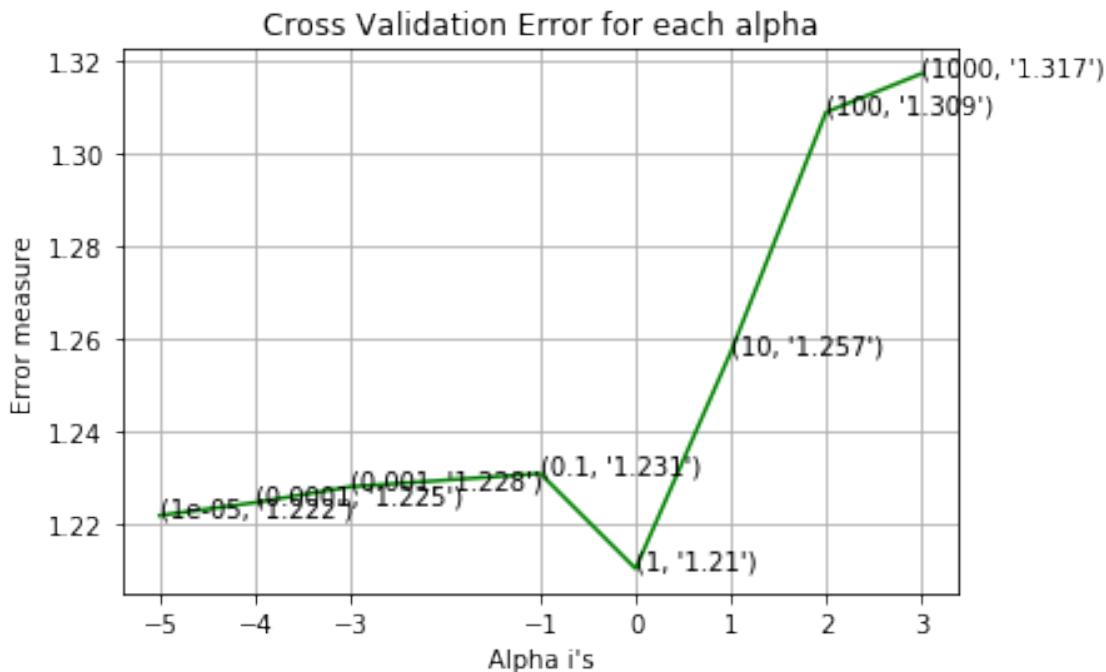
best_alpha = np.argmin(cv_log_error_array)
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)
```

```

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_l
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_l

for alpha = 1e-05
Log Loss : 1.2218154835124873
for alpha = 0.0001
Log Loss : 1.2245947881893156
for alpha = 0.001
Log Loss : 1.2280097082423465
for alpha = 0.1
Log Loss : 1.2308608621708208
for alpha = 1
Log Loss : 1.2103503652361725
for alpha = 10
Log Loss : 1.2570740407647683
for alpha = 100
Log Loss : 1.3088530714316662
for alpha = 1000
Log Loss : 1.3171493119701125

```



```

For values of best alpha = 1 The train log loss is: 0.852172858970628
For values of best alpha = 1 The cross validation log loss is: 1.2103503652361725
For values of best alpha = 1 The test log loss is: 1.1981215375530594

```

```

In [85]: #alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
          alpha = np.random.uniform(0.5,1.5,10)
          alpha.sort()
          cv_log_error_array = []
          for i in alpha:
              print("for alpha =", i)
              clf = MultinomialNB(alpha=i)
              clf.fit(train_x_tfidfCoding, train_y)
              sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
              sig_clf.fit(train_x_tfidfCoding, train_y)
              sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
              cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
              # to avoid rounding error while multiplying probabilites we use log-probability
              print("Log Loss :",log_loss(cv_y, sig_clf_probs))

              fig, ax = plt.subplots()
              ax.plot(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_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_

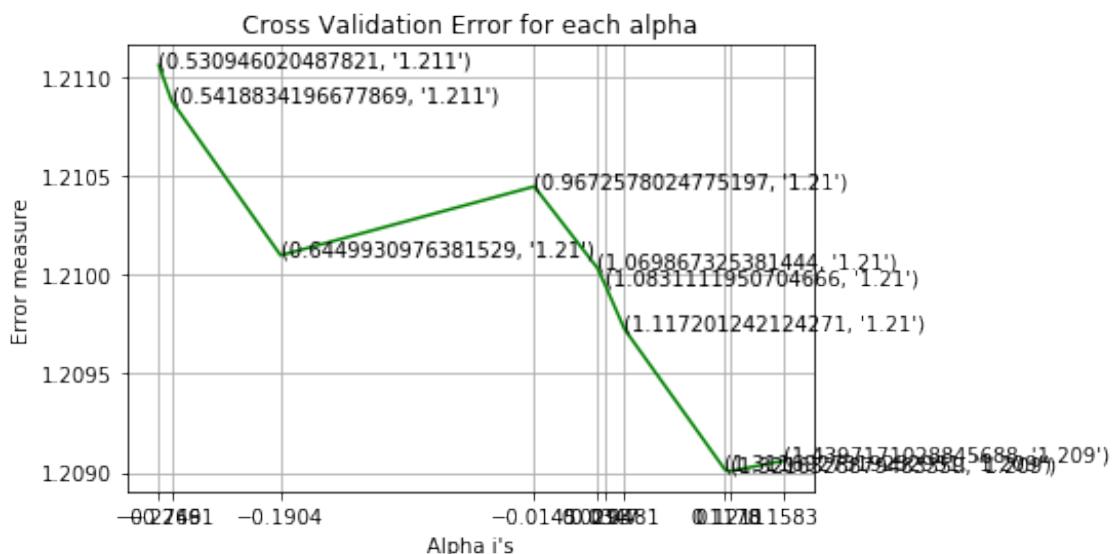
for alpha = 0.530946020487821
Log Loss : 1.2110595994956441

```

```

for alpha = 0.5418834196677869
Log Loss : 1.210880176664092
for alpha = 0.6449930976381529
Log Loss : 1.2100985878156307
for alpha = 0.9672578024775197
Log Loss : 1.2104466021792089
for alpha = 1.069867325381444
Log Loss : 1.2100351371681517
for alpha = 1.0831111950704666
Log Loss : 1.2099506728136395
for alpha = 1.117201242124271
Log Loss : 1.2097263263821174
for alpha = 1.3116927319252951
Log Loss : 1.2090161391283065
for alpha = 1.3216328879483359
Log Loss : 1.2090073411089732
for alpha = 1.4397171028845688
Log Loss : 1.2090598730774664

```



For values of best alpha = 1.3216328879483359 The train log loss is: 0.8851381077249583  
 For values of best alpha = 1.3216328879483359 The cross validation log loss is: 1.2090073411089732  
 For values of best alpha = 1.3216328879483359 The test log loss is: 1.1954407270352287

```
In [86]: 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 will 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))/test_y.shape[0])
plot_confusion_matrix(test_y, pred_y)
return sig_clf,np.count_nonzero((pred_y- test_y))/test_y.shape[0]

```

In [87]: ##error and confusioon matrix

```

clf = MultinomialNB(alpha=1.3216328879483359)
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estimation
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_tfidfCoding)-cv_y)))
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_tfidfCoding.toarray()))
list_data = []
list_data.append('Tf_Idf+NB')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_tfidfCoding), labels=clf.classes_))
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_tfidfCoding), labels=clf.classes_))
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_tfidfCoding), labels=clf.classes_))
list_data.append('alpha = '+str(clf.alpha))
list_data.append(np.count_nonzero((sig_clf.predict(cv_x_tfidfCoding)- cv_y))/cv_y.shape[0])
final_results.append(list_data)

```

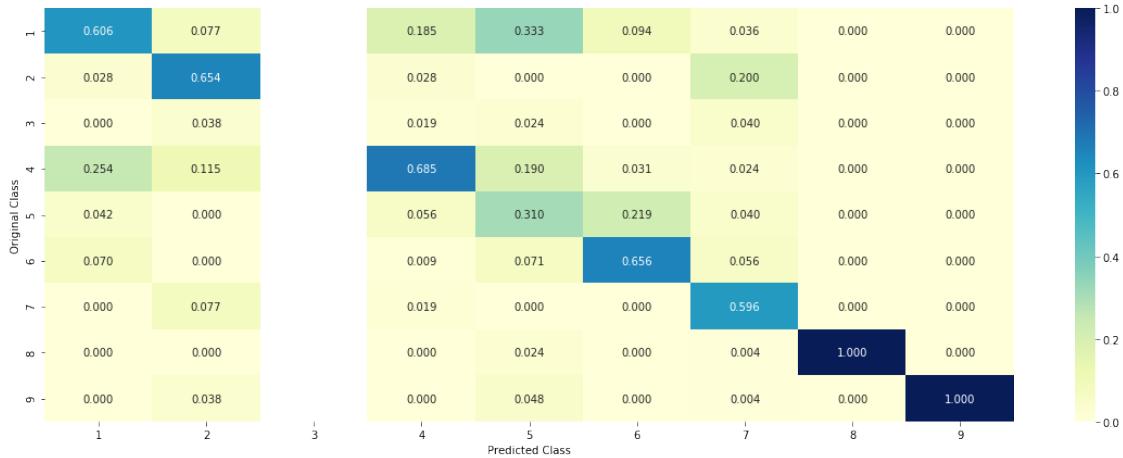
Log Loss : 1.2090073411089732

Number of missclassified point : 0.39849624060150374

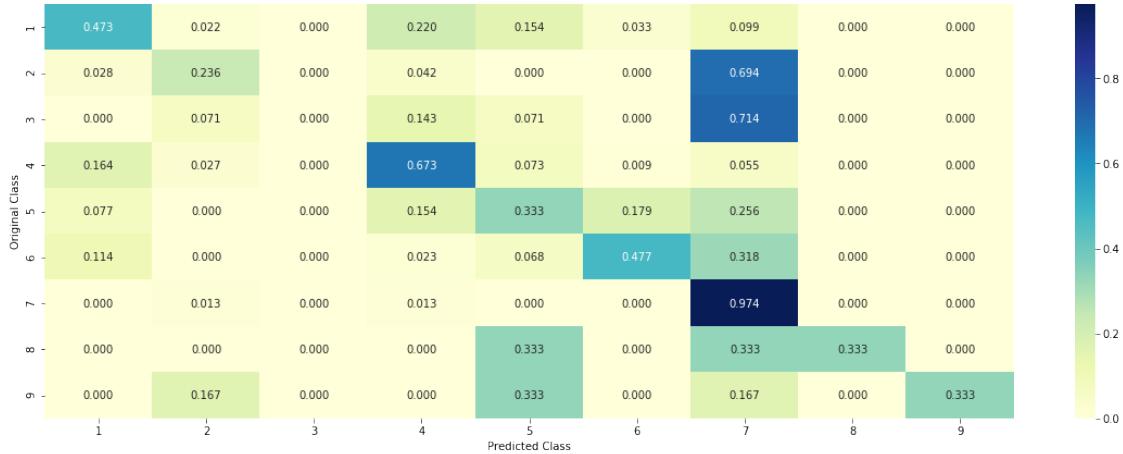
----- Confusion matrix -----



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



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



## Feature Importance

```
In [89]: # 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_imfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3)
```

```

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

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

word_present = 0
for i,v in enumerate(indices):
    if (v < fea1_len):
        word = gene_vec.get_feature_names()[v]
        yes_no = True if word == gene else False
        if yes_no:
            word_present += 1
            print(i, "Gene feature [{}] present in test data point [{}].format(w
    elif (v < fea1_len+fea2_len):
        word = var_vec.get_feature_names()[v-(fea1_len)]
        yes_no = True if word == var else False
        if yes_no:
            word_present += 1
            print(i, "variation feature [{}] present in test data point [{}].format(w
    else:
        temp1 = list(np.array(text_vec.get_feature_names())[top1000_idx])
        word = temp1[v-(fea1_len+fea2_len)]
        yes_no = True if word in text.split() else False
        if yes_no:
            word_present += 1
            print(i, "Text feature [{}] present in test data point [{}].format(w

print("Out of the top ",no_features," features ", word_present, "are present in query point")

```

```

In [93]: test_point_index = 75
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCoding)[test_point_index], 3))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imptfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index], test_df['Gene'].iloc[test_point_index])

```

Predicted Class : 4  
Predicted Class Probabilities: [[0.154 0.074 0.0289 0.5804 0.0498 0.0474 0.0533 0.0049 0.0071 0.0001]  
Actual Class : 4  
-----  
Out of the top 100 features 0 are present in query point

```
In [89]: test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCoding[test_point_index]), 3))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imptfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index], test_df['Genre'])

Predicted Class : 7
Predicted Class Probabilities: [[0.0418 0.0547 0.0179 0.0408 0.0323 0.0369 0.7687 0.0018 0.005
Actual Class : 2
-----
Out of the top 100 features 0 are present in query point
```

### 6.1.2. K Nearest Neighbour Classification

```
In [95]: alpha = [3,5, 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_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability encoding
    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()

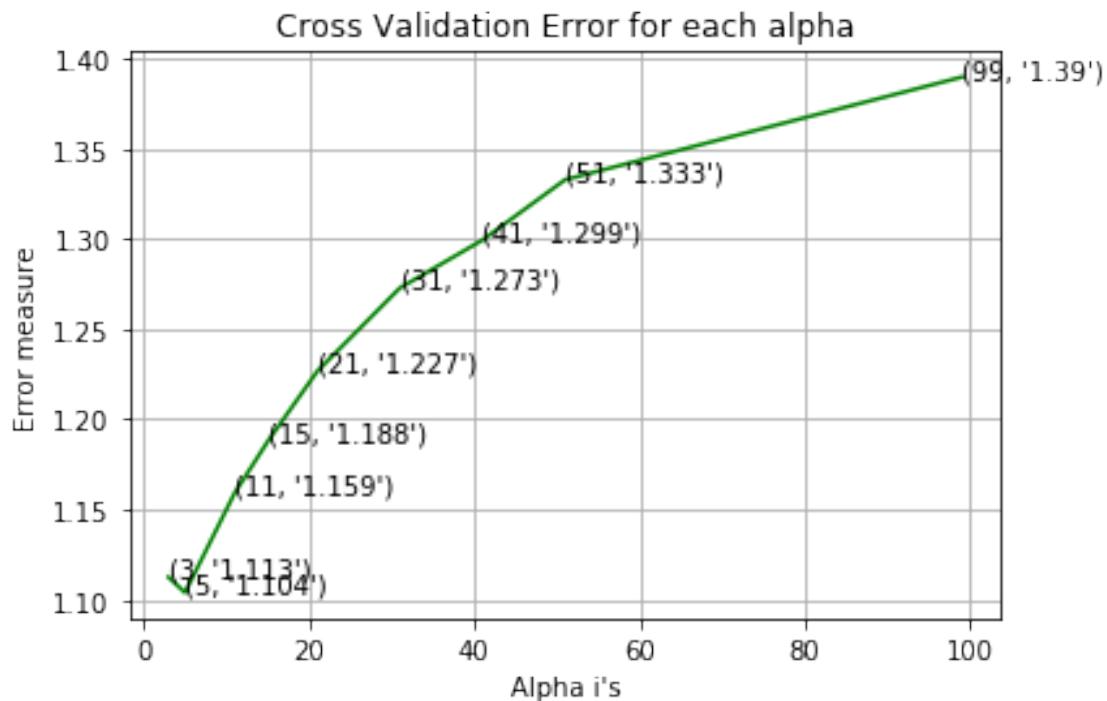
best_alpha = np.argmin(cv_log_error_array)
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)
```

```

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_)
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_cv)
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_test)

for alpha = 3
Log Loss : 1.1126090451296378
for alpha = 5
Log Loss : 1.104016554347388
for alpha = 11
Log Loss : 1.1591222687963092
for alpha = 15
Log Loss : 1.1876959827443956
for alpha = 21
Log Loss : 1.2266182852403424
for alpha = 31
Log Loss : 1.2727826330764145
for alpha = 41
Log Loss : 1.299349990086429
for alpha = 51
Log Loss : 1.3328085978650193
for alpha = 99
Log Loss : 1.3896642539637867

```



```

For values of best alpha = 5 The train log loss is: 0.9056379119075273
For values of best alpha = 5 The cross validation log loss is: 1.104016554347388
For values of best alpha = 5 The test log loss is: 1.1274954040369372

```

In [96]: #testing

```

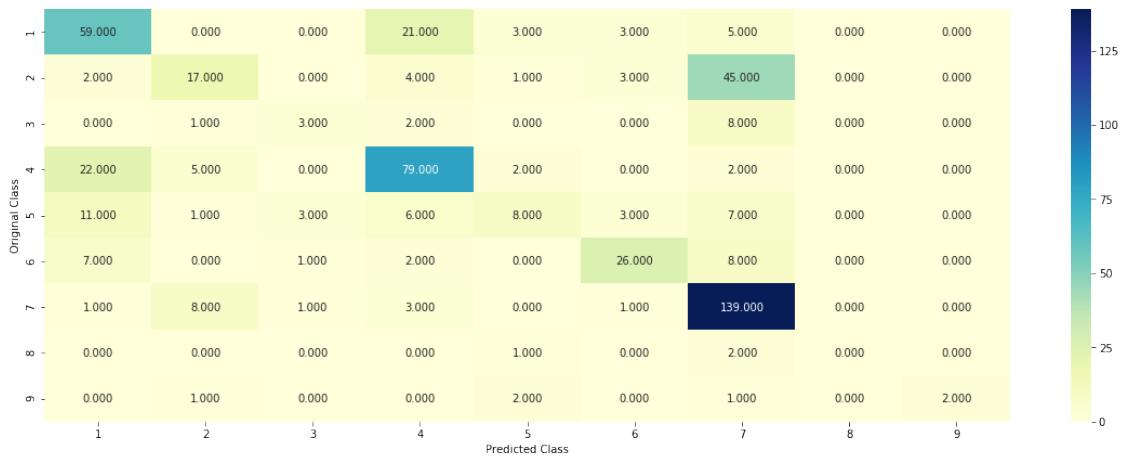
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha], algorithm='brute')
sig_clf,temp = predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y, cv_x_t
list_data = []
list_data.append('Tf_Idf_Top1000+KNN')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_tfidfCoding), eps=1e-15))
list_data.append('K '+str(clf.n_neighbors))
list_data.append(temp)
final_results.append(list_data)

```

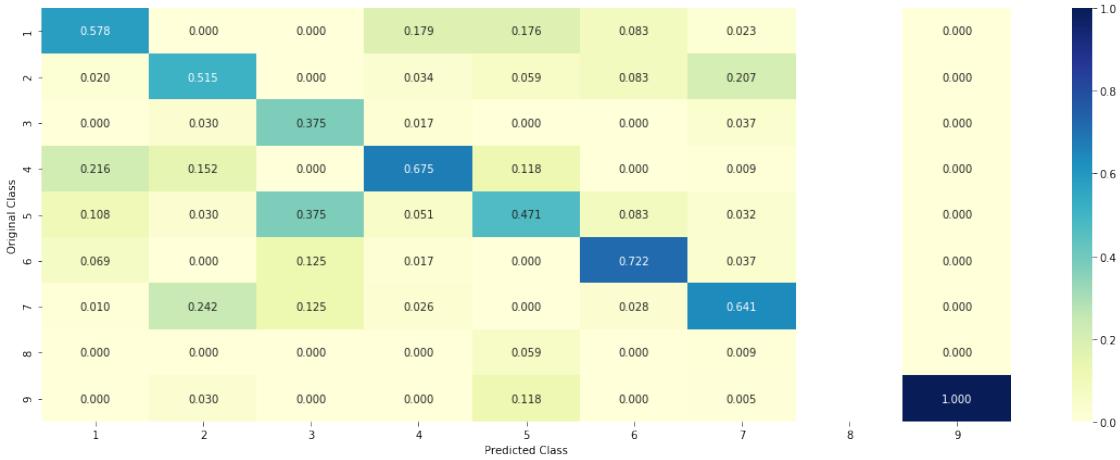
Log loss : 1.104016554347388

Number of mis-classified points : 0.37406015037593987

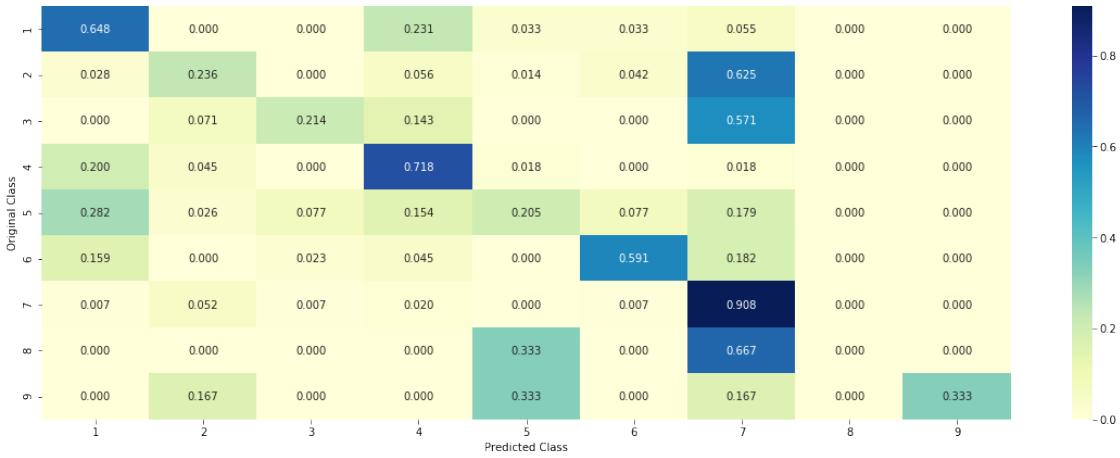
----- Confusion matrix -----



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



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



## Feature importance

```
In [100]: clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

test_point_index = 45
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_tfidfCoding[test_point_index], alpha[best_alpha])
```

```

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

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

In [101]: clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

test_point_index = 95
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_tfidfCoding[test_point_index], alpha[best_alpha])
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to class")
print("Frequency of nearest points :",Counter(train_y[neighbors[1][0]]))

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

```

### 6.1.3. Logistic Regression

#### With Class balancing

```

In [102]: 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', max_iter=1000)
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

```

```

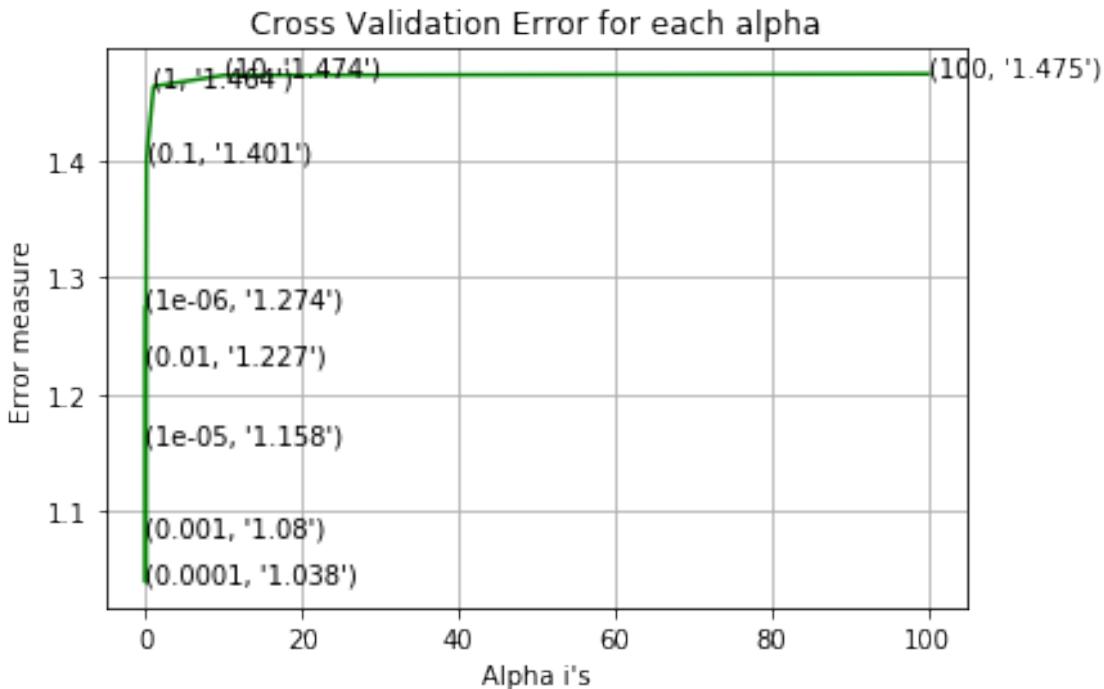
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

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

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y))

for alpha = 1e-06
Log Loss : 1.2742413547252438
for alpha = 1e-05
Log Loss : 1.1583464587718548
for alpha = 0.0001
Log Loss : 1.0384716717285567
for alpha = 0.001
Log Loss : 1.0795816909566425
for alpha = 0.01
Log Loss : 1.2268286208991612
for alpha = 0.1
Log Loss : 1.4012743680439574
for alpha = 1
Log Loss : 1.4640809054632058
for alpha = 10
Log Loss : 1.4735366023268988
for alpha = 100
Log Loss : 1.4747096422215427

```



```
For values of best alpha = 0.0001 The train log loss is: 0.45744771918464255
For values of best alpha = 0.0001 The cross validation log loss is: 1.0384716717285567
For values of best alpha = 0.0001 The test log loss is: 1.0036573672843676
```

```
In [103]: #alpha = [10 ** x for x in range(-6, 3)]
alpha = np.random.uniform(0.00005, 0.0005, 15)
alpha = np.round(alpha, 7)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log',
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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

```

        ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

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

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_

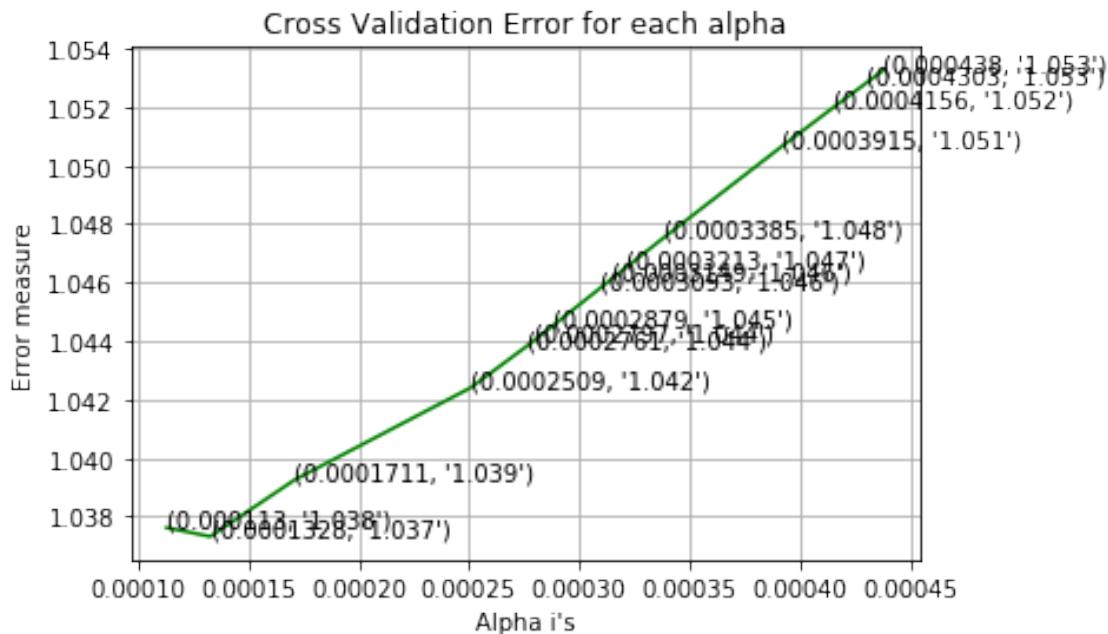
for alpha = 0.000113
Log Loss : 1.0376239349257512
for alpha = 0.0001328
Log Loss : 1.0373209165039003
for alpha = 0.0001711
Log Loss : 1.0392628664722428
for alpha = 0.0002509
Log Loss : 1.0424100715569196
for alpha = 0.0002761
Log Loss : 1.043817475394664
for alpha = 0.0002797
Log Loss : 1.0440286105489225
for alpha = 0.0002879
Log Loss : 1.0445145401700895
for alpha = 0.0003093
Log Loss : 1.0457979157880797
for alpha = 0.0003149
Log Loss : 1.0461342739551749
for alpha = 0.0003213
Log Loss : 1.0465180061869872
for alpha = 0.0003385
Log Loss : 1.047543493474002
for alpha = 0.0003915
Log Loss : 1.0506335152989867
for alpha = 0.0004156
Log Loss : 1.0520038154362383
for alpha = 0.0004303

```

```

Log Loss : 1.0528296862104571
for alpha = 0.000438
Log Loss : 1.0532593491471727

```



```

For values of best alpha = 0.0001328 The train log loss is: 0.4726759091669523
For values of best alpha = 0.0001328 The cross validation log loss is: 1.0373209165039003
For values of best alpha = 0.0001328 The test log loss is: 1.0034447431644684

```

In [104]: #testing

```

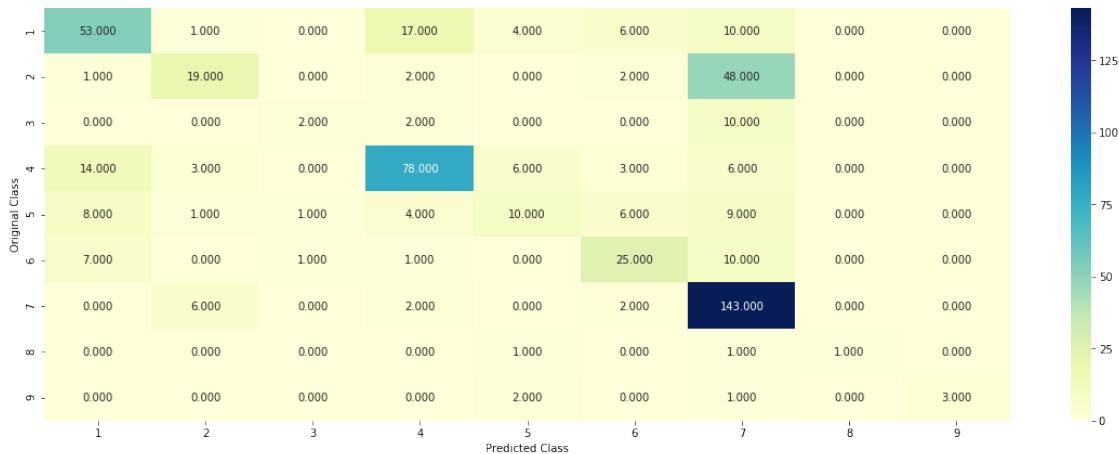
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
                    max_iter=1000, tol=1e-05)
sig_clf,temp = predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y, cv_x_tfidfCoding, cv_y)
list_data = []
list_data.append('Tf_Idf_Top1000+LR+Class_Balancing+SGD')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_tfidfCoding), eps=1e-15))
list_data.append('alpha = '+str(clf.alpha))
list_data.append(temp)
final_results.append(list_data)

```

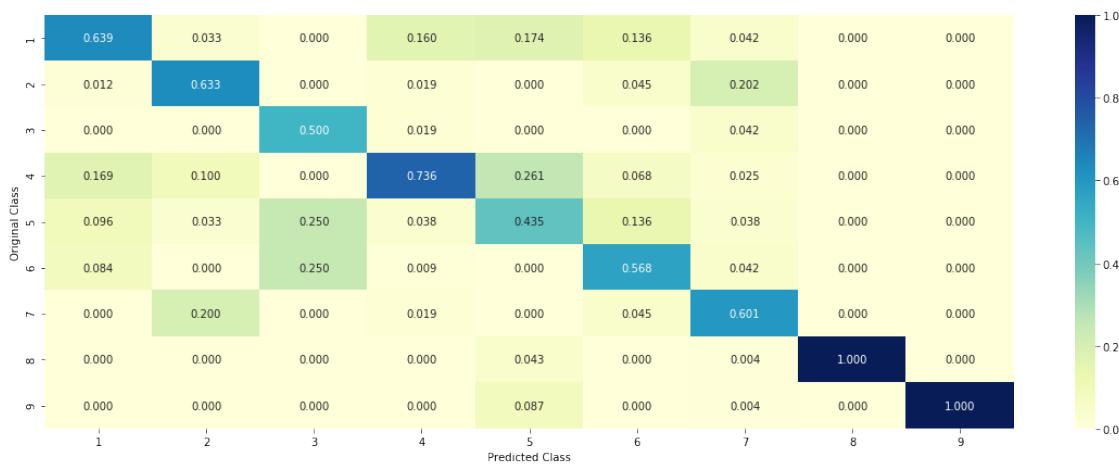
```

Log loss : 1.0373209165039003
Number of mis-classified points : 0.37218045112781956
----- Confusion matrix -----

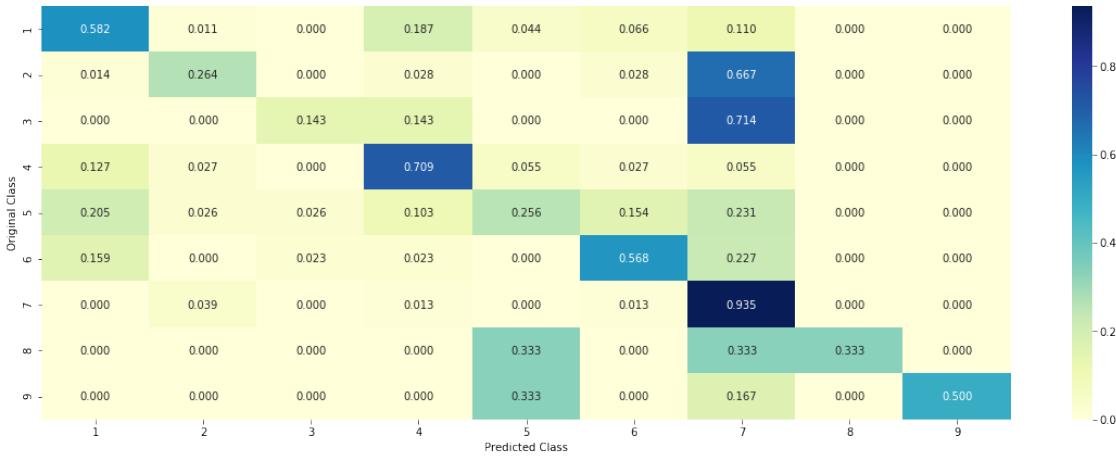
```



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



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



## Feature Importance

```
In [105]: # from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=0.0001328, penalty='l2', loss='log')
clf.fit(train_x_tfidfCoding, train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCoding[test_point_index])[0]))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imptfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index], test_df['Genre'])

Predicted Class : 7
Predicted Class Probabilities: [[0.052  0.0386  0.0041  0.0087  0.0331  0.0197  0.8376  0.0029  0.0033  0.0021]
                                 ...]
Actual Class : 2
-----
Out of the top 500 features 0 are present in query point
```

```
In [106]: # from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=0.0001328, penalty='l2', loss='log')
clf.fit(train_x_tfidfCoding, train_y)
test_point_index = 52
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCoding[test_point_index])[0]))
print("Actual Class :", test_y[test_point_index])
```

```

indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imffeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Genre'])

Predicted Class : 7
Predicted Class Probabilities: [[0.0367 0.0759 0.0123 0.0532 0.1089 0.042 0.6591 0.0052 0.0062]
Actual Class : 7
-----
488 Text feature [002755] present in test data point [True]
Out of the top 500 features 1 are present in query point

```

## Without Class Balancing

```

In [107]: alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    #clf = LogisticRegression(C=i, class_weight='balanced', n_jobs=-1, solver='liblinear')
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

best_alpha = np.argmin(cv_log_error_array)
#clf = LogisticRegression(C=i, class_weight='balanced', n_jobs=-1, solver='liblinear')
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

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

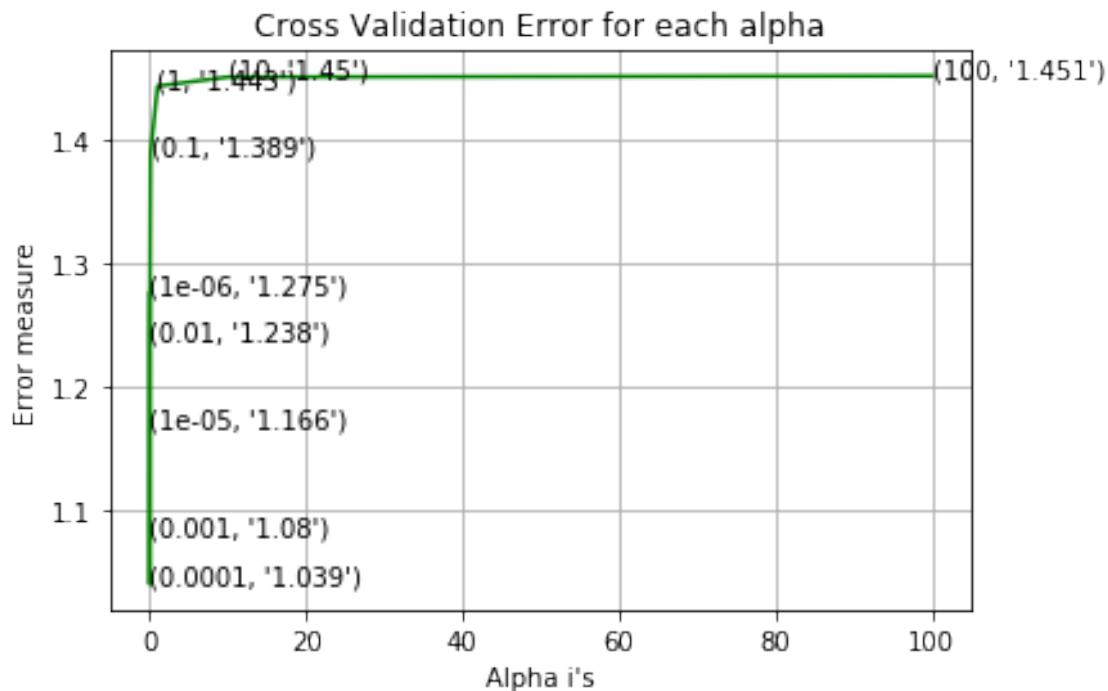
```

```

predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss_cv)
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss)

for alpha = 1e-06
Log Loss : 1.2750734917903674
for alpha = 1e-05
Log Loss : 1.1660016940944902
for alpha = 0.0001
Log Loss : 1.0391147837461054
for alpha = 0.001
Log Loss : 1.0800023577589466
for alpha = 0.01
Log Loss : 1.2380292873621375
for alpha = 0.1
Log Loss : 1.3885619241486853
for alpha = 1
Log Loss : 1.442868186242606
for alpha = 10
Log Loss : 1.4502012739906016
for alpha = 100
Log Loss : 1.4510984530252005

```



```
For values of best alpha = 0.0001 The train log loss is: 0.44707505287015314
For values of best alpha = 0.0001 The cross validation log loss is: 1.0391147837461054
For values of best alpha = 0.0001 The test log loss is: 1.0065184466969561
```

```
In [108]: #alpha = [10 ** x for x in range(-6, 3)]
alpha = np.random.uniform(0.00002, 0.0005, 20)
alpha = np.round(alpha, 7)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    #clf = LogisticRegression(C=i, class_weight='balanced', n_jobs=-1, solver='liblinear')
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

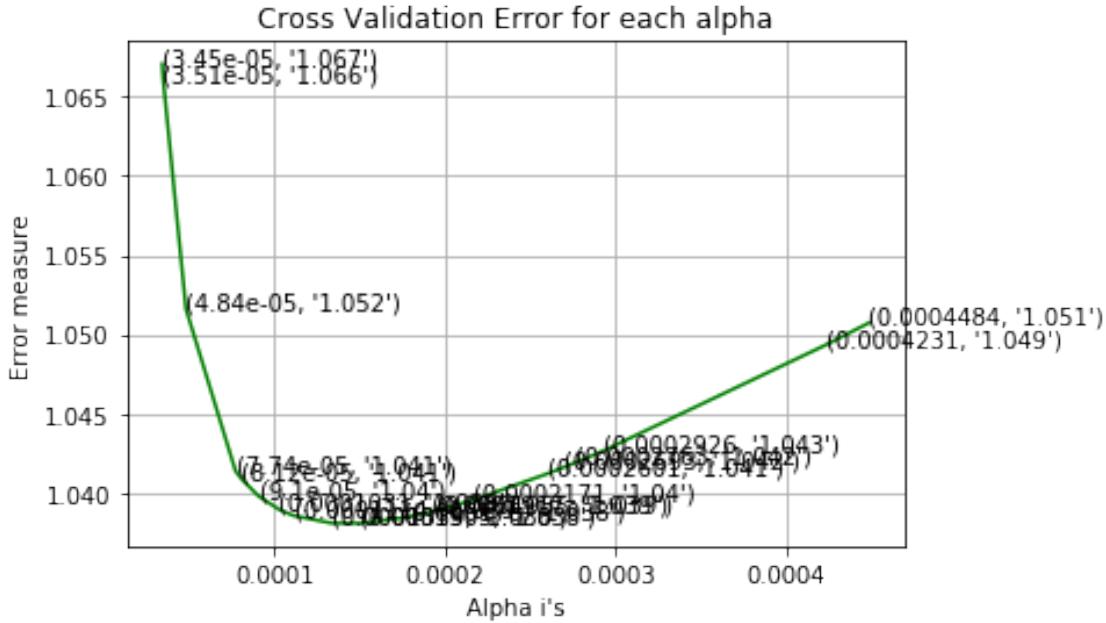
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], str(txt)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error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

best_alpha = np.argmin(cv_log_error_array)
#clf = LogisticRegression(C=i, class_weight='balanced', n_jobs=-1, solver='liblinear')
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(predict_y, cv_y))
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(predict_y, cv_y))
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(predict_y, test_y))

for alpha = 3.45e-05
Log Loss : 1.0670742346263986
```

```
for alpha = 3.51e-05
Log Loss : 1.066057062831967
for alpha = 4.84e-05
Log Loss : 1.0516091309842406
for alpha = 7.74e-05
Log Loss : 1.0413810334405824
for alpha = 8.12e-05
Log Loss : 1.0408420319020082
for alpha = 9.1e-05
Log Loss : 1.0397828067870847
for alpha = 0.0001031
Log Loss : 1.0389355253801063
for alpha = 0.0001114
Log Loss : 1.0385577756005775
for alpha = 0.0001339
Log Loss : 1.038081593466502
for alpha = 0.0001505
Log Loss : 1.038076578939728
for alpha = 0.000176
Log Loss : 1.0384289368422561
for alpha = 0.0001917
Log Loss : 1.038797645252271
for alpha = 0.0001955
Log Loss : 1.0389003481895909
for alpha = 0.0002171
Log Loss : 1.0395657820638144
for alpha = 0.0002601
Log Loss : 1.0412036817627637
for alpha = 0.0002693
Log Loss : 1.0415951587808367
for alpha = 0.0002763
Log Loss : 1.0419009589101702
for alpha = 0.0002926
Log Loss : 1.0426370922924915
for alpha = 0.0004231
Log Loss : 1.0493311088218704
for alpha = 0.0004484
Log Loss : 1.0507243649923408
```



For values of best alpha = 0.0001505 The train log loss is: 0.4688483155455666

For values of best alpha = 0.0001505 The cross validation log loss is: 1.038076578939728

For values of best alpha = 0.0001505 The test log loss is: 1.0050061762121292

In [109]: #testing

```

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=1)
sig_clf,temp = predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y, cv_x_tfidfCoding)

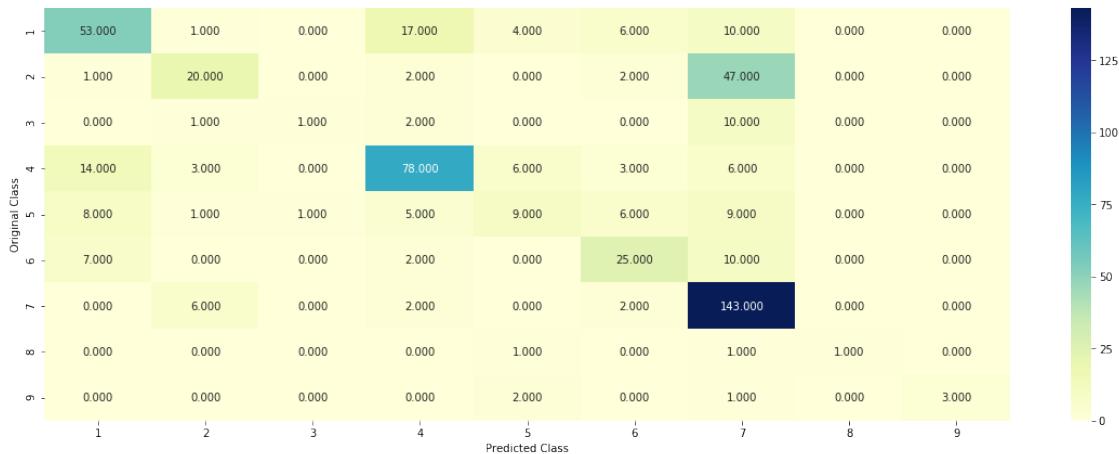
list_data = []
list_data.append('Tf_Idf_Top1000+LR+Class_Imbalance')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_tfidfCoding), eps=1e-15))
list_data.append('alpha = '+str(clf.alpha))
list_data.append(temp)
final_results.append(list_data)

```

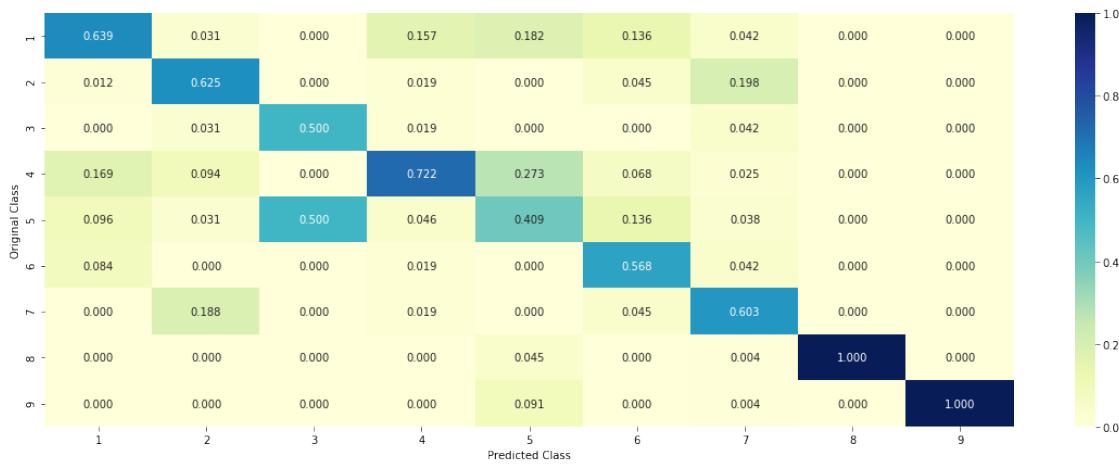
Log loss : 1.038076578939728

Number of mis-classified points : 0.37406015037593987

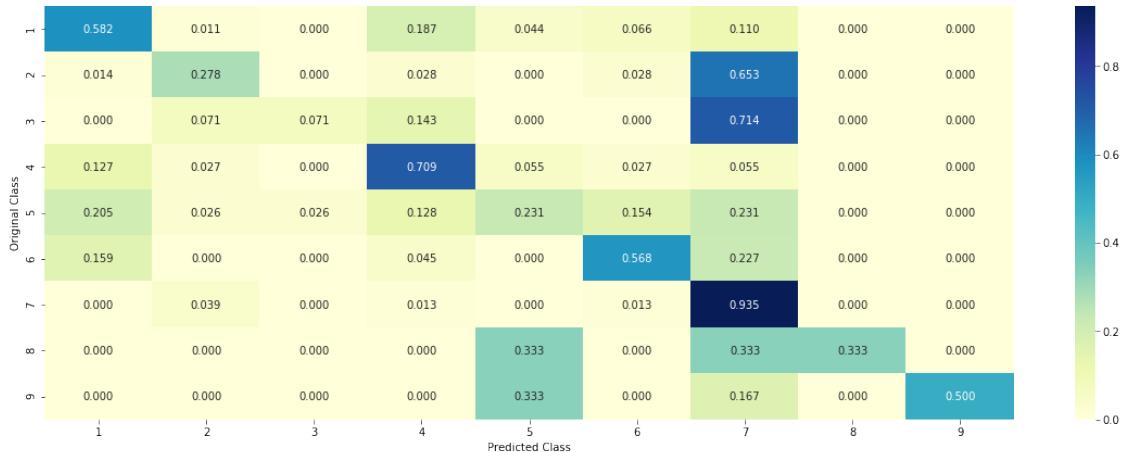
----- Confusion matrix -----



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



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



#### 6.1.4.Linear Support Vector Machines

##### With Balanced Classes

```
In [70]: alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', random_state=42, class_weight=None)
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilités we use log-probability encoding
    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='hinge', random_state=42, class_weight=None)
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

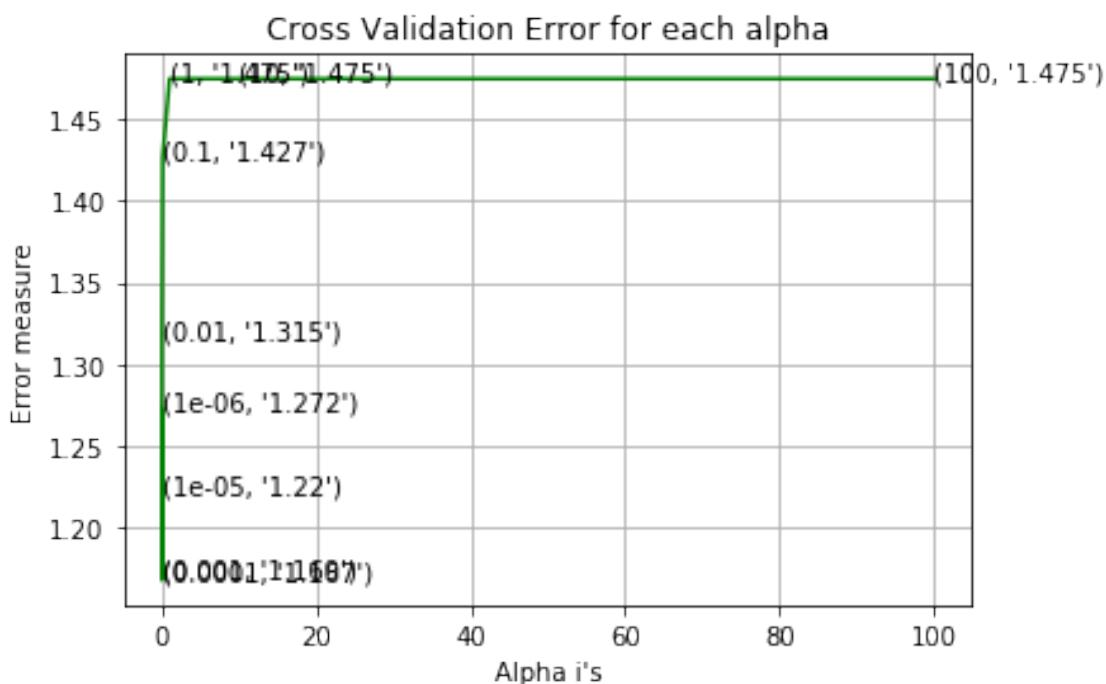
```

sig_clf.fit(train_x_tfidfCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss)
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_cv)
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_test)

for alpha = 1e-06
Log Loss : 1.272066047261281
for alpha = 1e-05
Log Loss : 1.2200501732568565
for alpha = 0.0001
Log Loss : 1.1673526217848709
for alpha = 0.001
Log Loss : 1.1688993247520392
for alpha = 0.01
Log Loss : 1.3153858223713897
for alpha = 0.1
Log Loss : 1.4265077476615182
for alpha = 1
Log Loss : 1.4750563516625652
for alpha = 10
Log Loss : 1.4750563739341356
for alpha = 100
Log Loss : 1.4750562740034543

```



```

For values of best alpha = 0.0001 The train log loss is: 0.5359711950851
For values of best alpha = 0.0001 The cross validation log loss is: 1.1763727085856153
For values of best alpha = 0.0001 The test log loss is: 1.1439517744655447

In [71]: alpha = [10 ** x for x in range(-6, 3)]
          alpha = np.random.uniform(0.00005, 0.0005, 15)
          alpha = np.round(alpha, 7)
          alpha.sort()
          cv_log_error_array = []
          for i in alpha:
              print("for alpha =", i)
              clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', random_state=42, class_weight=None)
              clf.fit(train_x_tfidfCoding, train_y)
              sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
              sig_clf.fit(train_x_tfidfCoding, train_y)
              sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
              cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=0.1))
              # to avoid rounding error while multiplying probabilites we use log-probability
              print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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

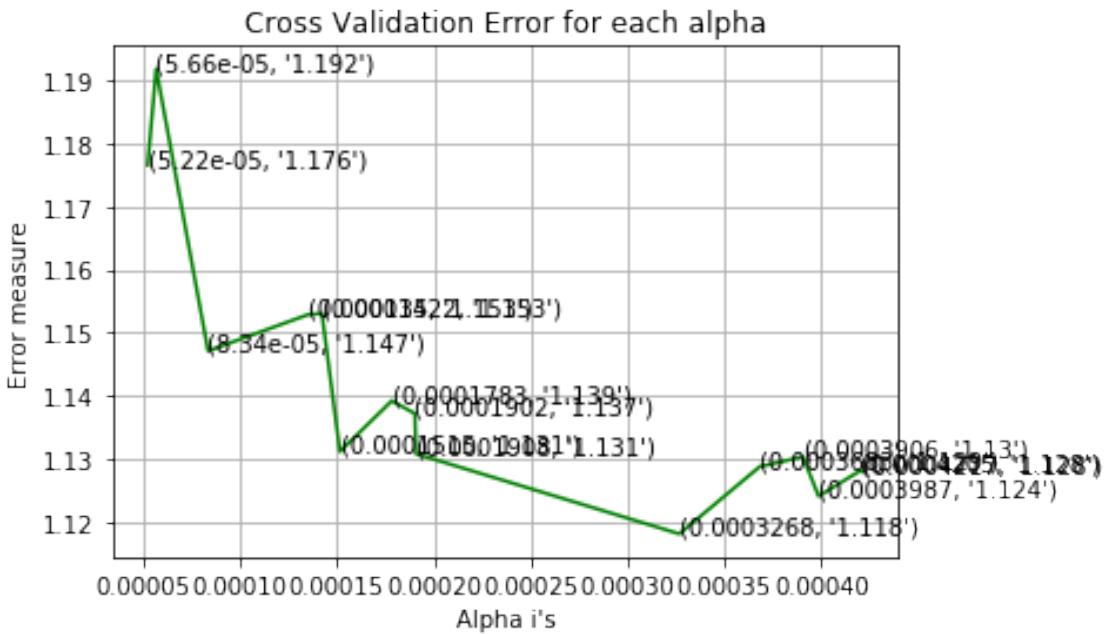
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42, class_weight=None)
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(cv_y, predict_y))
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(cv_y, predict_y))
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(cv_y, predict_y))

for alpha = 5.22e-05

```

```
Log Loss : 1.1764979826757143
for alpha = 5.66e-05
Log Loss : 1.1917067925133678
for alpha = 8.34e-05
Log Loss : 1.1471592719946184
for alpha = 0.000135
Log Loss : 1.152888317385696
for alpha = 0.0001422
Log Loss : 1.1530646788053136
for alpha = 0.0001515
Log Loss : 1.1312665968697224
for alpha = 0.0001783
Log Loss : 1.1392792922279584
for alpha = 0.0001902
Log Loss : 1.137263428105586
for alpha = 0.0001908
Log Loss : 1.1309038550273058
for alpha = 0.0003268
Log Loss : 1.1182642886147933
for alpha = 0.0003683
Log Loss : 1.1288804681922344
for alpha = 0.0003906
Log Loss : 1.1304471288797926
for alpha = 0.0003987
Log Loss : 1.124220865046015
for alpha = 0.0004205
Log Loss : 1.1282959970998367
for alpha = 0.0004217
Log Loss : 1.1280171563272299
```



For values of best alpha = 0.0003268 The train log loss is: 0.4842222603930427

For values of best alpha = 0.0003268 The cross validation log loss is: 1.1185399308691886

For values of best alpha = 0.0003268 The test log loss is: 1.0798408837074438

```
In [72]: alpha = [10 ** x for x in range(-6, 3)]
alpha = np.random.uniform(0.0002,0.0005,10)
alpha = np.round(alpha,7)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', random_state=42, class_weight=None)
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=0.1))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

```

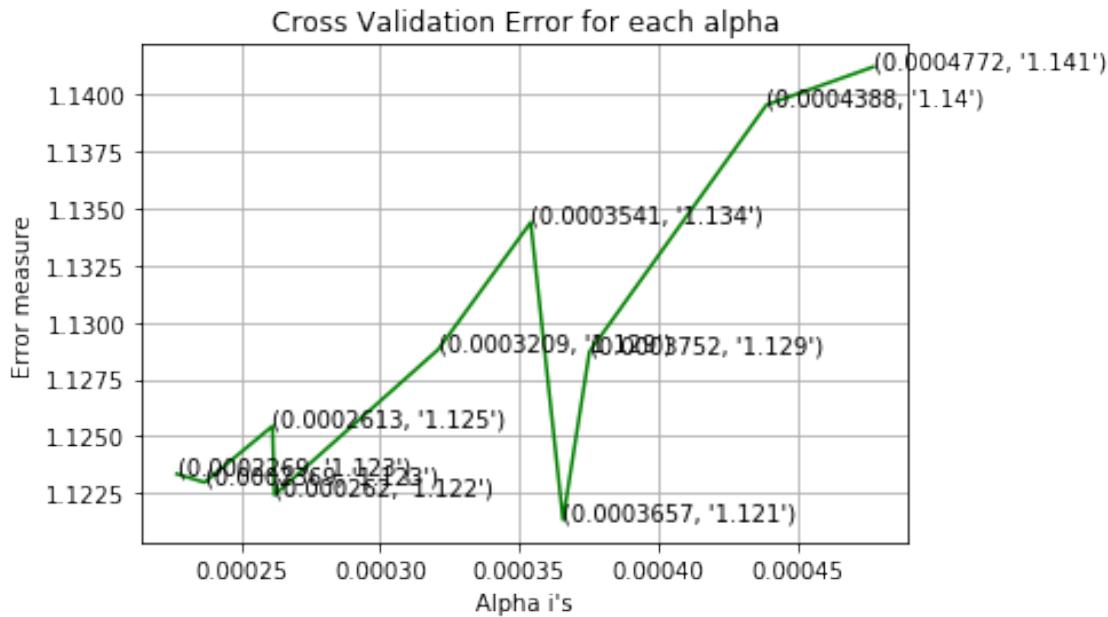
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(train_y,predict_y))
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(cv_y,predict_y))
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(test_y,predict_y))

for alpha = 0.0002269
Log Loss : 1.1233531125257687
for alpha = 0.0002369
Log Loss : 1.1229787254873083
for alpha = 0.0002613
Log Loss : 1.125449554166514
for alpha = 0.000262
Log Loss : 1.1224253512243767
for alpha = 0.0003209
Log Loss : 1.1288189009866831
for alpha = 0.0003541
Log Loss : 1.1343959099063583
for alpha = 0.0003657
Log Loss : 1.1213361176066843
for alpha = 0.0003752
Log Loss : 1.1287141650222747
for alpha = 0.0004388
Log Loss : 1.1395535346142467
for alpha = 0.0004772
Log Loss : 1.1412228990771658

```



For values of best alpha = 0.0003657 The train log loss is: 0.486221624308985  
 For values of best alpha = 0.0003657 The cross validation log loss is: 1.100680933615641  
 For values of best alpha = 0.0003657 The test log loss is: 1.0715242928706294

In [73]: *##testing*

```

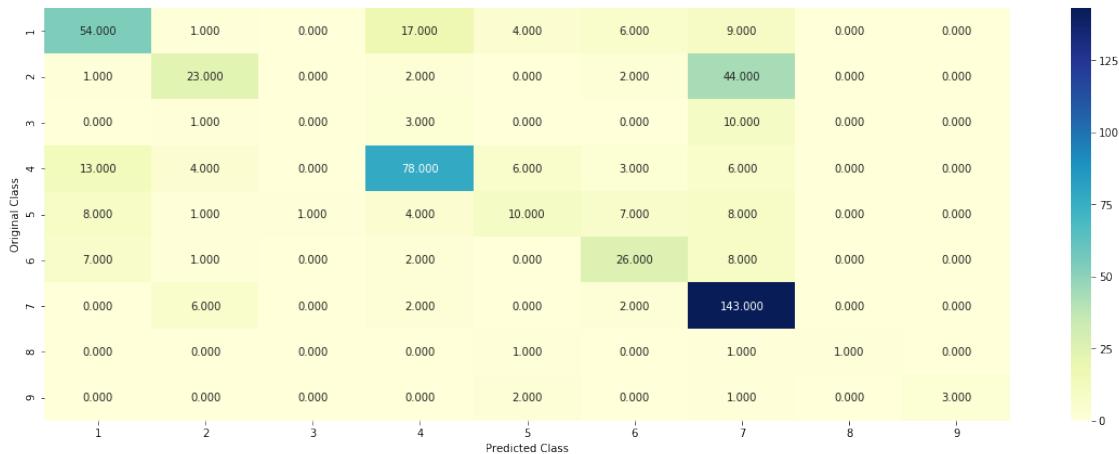
clf = SGDClassifier(alpha=0.0003657, penalty='l2', loss='hinge', random_state=42, class_
sig_clf,temp = predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y, cv_x_tf
list_data = []
list_data.append('Tf_Idf_Top1000+LSVC+Class_Balance')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_tfidfCoding), eps=1e-15))
list_data.append('alpha '+str(clf.alpha))
list_data.append(temp)
final_results.append(list_data)

```

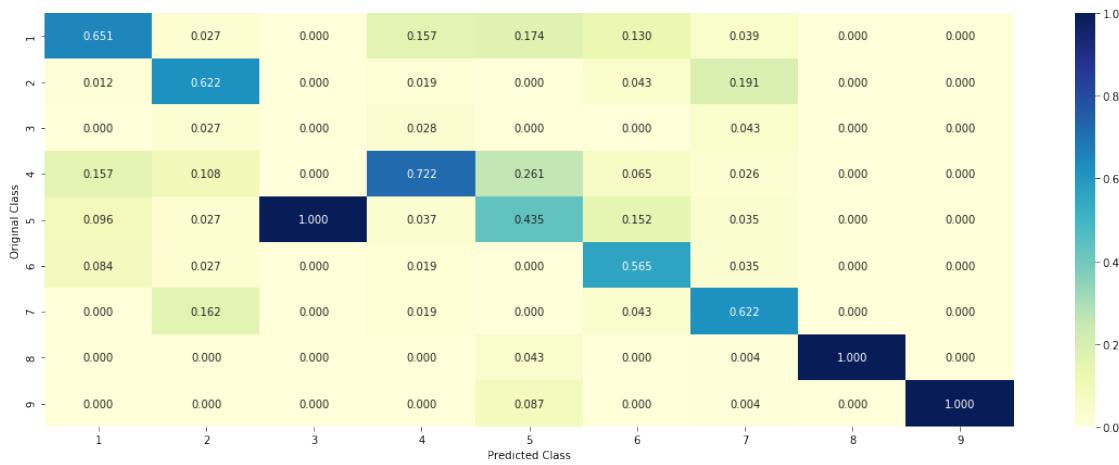
```

Log loss : 1.1213361176066843
Number of mis-classified points : 0.36466165413533835
----- Confusion matrix -----

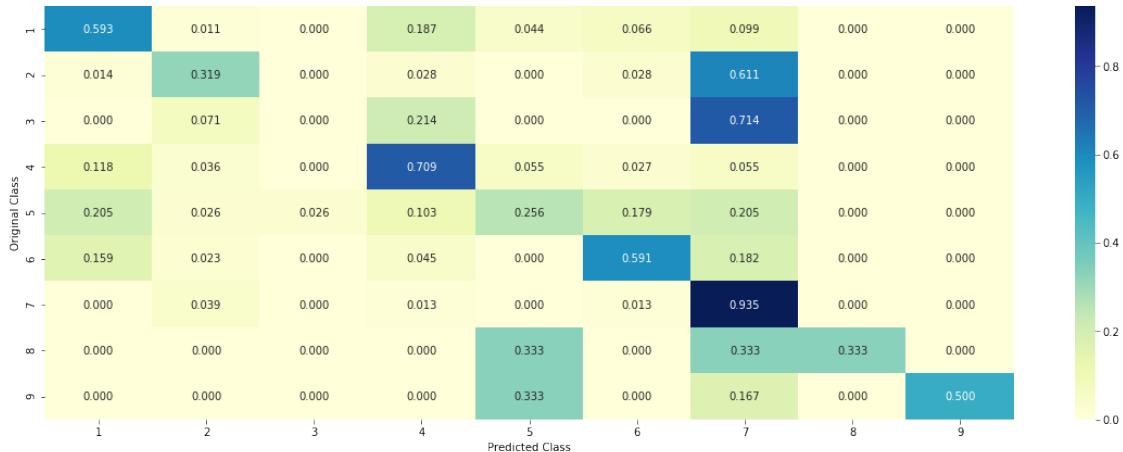
```



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



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



## Without balancing

```
In [77]: alpha = [10 ** x for x in range(-6, 3)]
#alpha = np.random.uniform(0.00005,0.0005,15)
#alpha = np.round(alpha,7)
#alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', random_state=42)
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability encoding
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

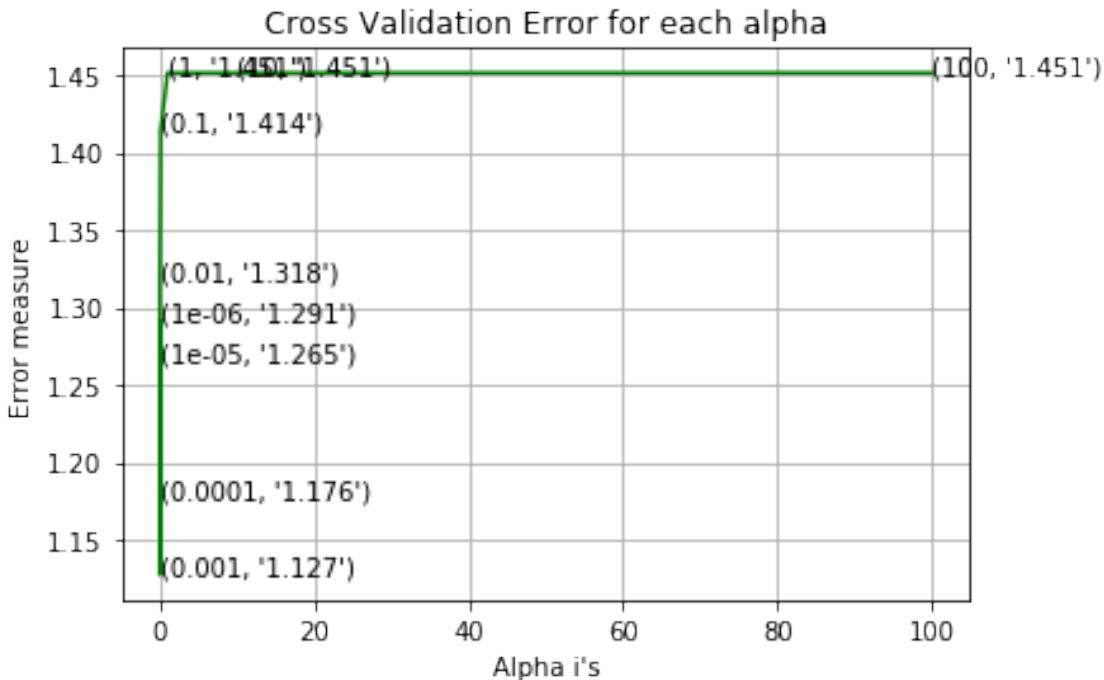
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")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

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
```

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

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss"
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_1

for alpha = 1e-06
Log Loss : 1.2913022198678832
for alpha = 1e-05
Log Loss : 1.2646274147266237
for alpha = 0.0001
Log Loss : 1.1763727085856153
for alpha = 0.001
Log Loss : 1.1273123917601031
for alpha = 0.01
Log Loss : 1.3182786911382816
for alpha = 0.1
Log Loss : 1.4141001133681201
for alpha = 1
Log Loss : 1.4513669995356056
for alpha = 10
Log Loss : 1.4513671141081745
for alpha = 100
Log Loss : 1.4513669933679882
```



```
For values of best alpha = 0.001 The train log loss is: 0.5657929490246093
For values of best alpha = 0.001 The cross validation log loss is: 1.1273123917601031
For values of best alpha = 0.001 The test log loss is: 1.0924211129794321
```

```
In [78]: #alpha = [10 ** x for x in range(-6, 3)]
alpha = np.random.uniform(0.0005,0.005,15)
alpha = np.round(alpha,7)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', random_state=42)
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

```

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

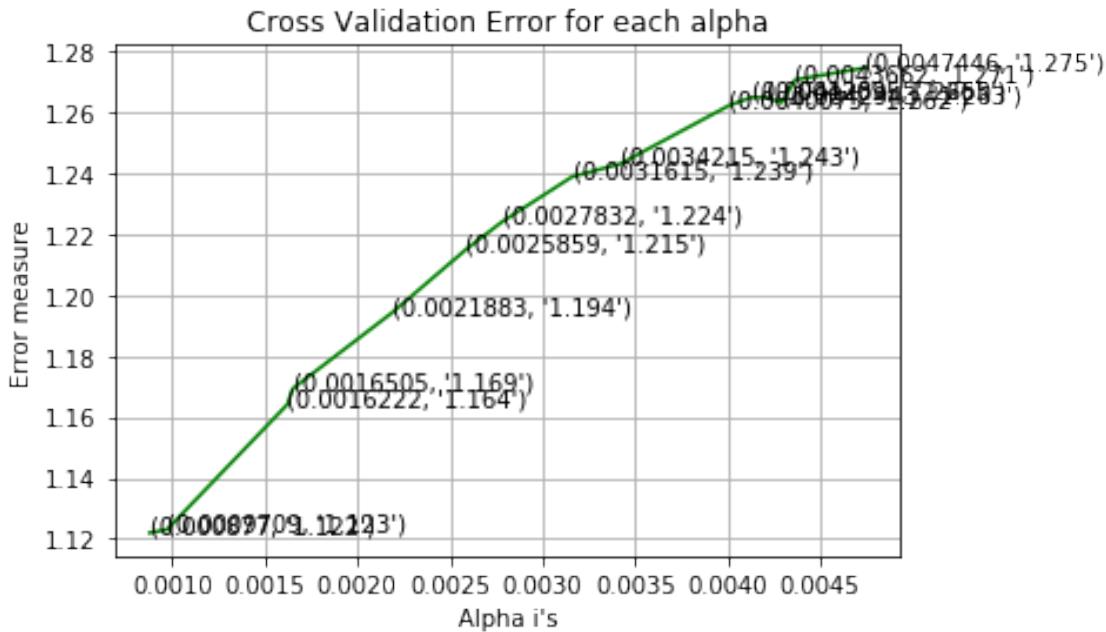
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(train_y,predict_y))
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(cv_y,predict_y))
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(test_y,predict_y))

for alpha = 0.000877
Log Loss : 1.121922882387426
for alpha = 0.0009709
Log Loss : 1.1230191361123818
for alpha = 0.0016222
Log Loss : 1.1641303895328938
for alpha = 0.0016505
Log Loss : 1.1692646933359756
for alpha = 0.0021883
Log Loss : 1.1942473484170808
for alpha = 0.0025859
Log Loss : 1.2149138822198011
for alpha = 0.0027832
Log Loss : 1.2240984164478699
for alpha = 0.0031615
Log Loss : 1.2388728156121223
for alpha = 0.0034215
Log Loss : 1.243096043210234
for alpha = 0.0040075
Log Loss : 1.2622053942106137
for alpha = 0.0041282
Log Loss : 1.2649006499973943
for alpha = 0.0042095
Log Loss : 1.2648502136059976
for alpha = 0.0042943
Log Loss : 1.263196976233171
for alpha = 0.0043662
Log Loss : 1.2706970672501794

```

```
for alpha = 0.0047446
Log Loss : 1.2746046597447256
```



For values of best alpha = 0.000877 The train log loss is: 0.5470806388177991  
 For values of best alpha = 0.000877 The cross validation log loss is: 1.121922882387426  
 For values of best alpha = 0.000877 The test log loss is: 1.088234388865766

```
In [79]: #alpha = [10 ** x for x in range(-6, 3)]
alpha = np.random.uniform(0.0001,0.0005,15)
alpha = np.round(alpha,7)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', random_state=42)
    clf.fit(train_x_tfidfCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability encoding
    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='hinge', random_state=None)
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,predict_y))
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv,predict_y))
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test,predict_y))

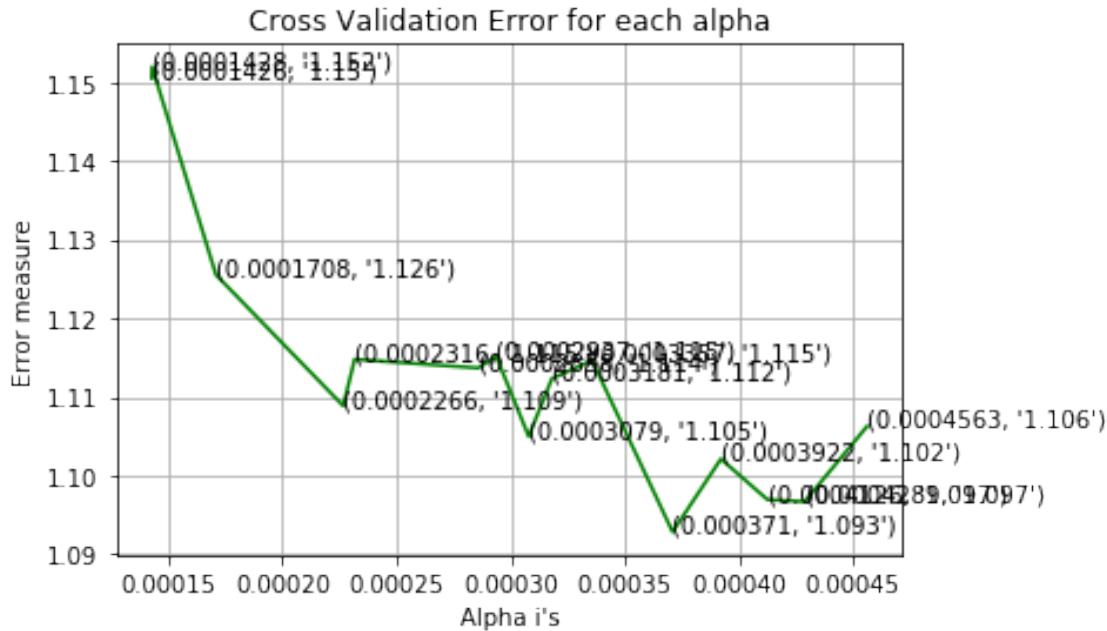
for alpha = 0.0001426
Log Loss : 1.1504878545491222
for alpha = 0.0001428
Log Loss : 1.1519354849469567
for alpha = 0.0001708
Log Loss : 1.125519126087881
for alpha = 0.0002266
Log Loss : 1.1088749890270753
for alpha = 0.0002316
Log Loss : 1.1147884457424864
for alpha = 0.0002858
Log Loss : 1.1136829525783034
for alpha = 0.0002937
Log Loss : 1.1151494771210757
for alpha = 0.0003079
Log Loss : 1.1049979301186286
for alpha = 0.0003181
Log Loss : 1.1123353712181743
for alpha = 0.0003357
Log Loss : 1.1146455152719763
for alpha = 0.000371
Log Loss : 1.0928612451433617
for alpha = 0.0003922
Log Loss : 1.1021306061416494
for alpha = 0.0004126

```

```

Log Loss : 1.0969596000349988
for alpha = 0.0004289
Log Loss : 1.0967077157352842
for alpha = 0.0004563
Log Loss : 1.1062791928362754

```



```

For values of best alpha = 0.000371 The train log loss is: 0.4835755837921846
For values of best alpha = 0.000371 The cross validation log loss is: 1.0928612451433617
For values of best alpha = 0.000371 The test log loss is: 1.0766410758120712

```

```

In [80]: ##testing
clf = SGDClassifier(alpha=0.000371, penalty='l2', loss='hinge', random_state=42)
sig_clf,temp = predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y, cv_x_tfi

list_data = []
list_data.append('Tf_Idf+LSVC+ImClass_Balance')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_tfidfCoding), eps=1e-15))
list_data.append('alpha '+str(clf.alpha))
list_data.append(temp)
final_results.append(list_data)

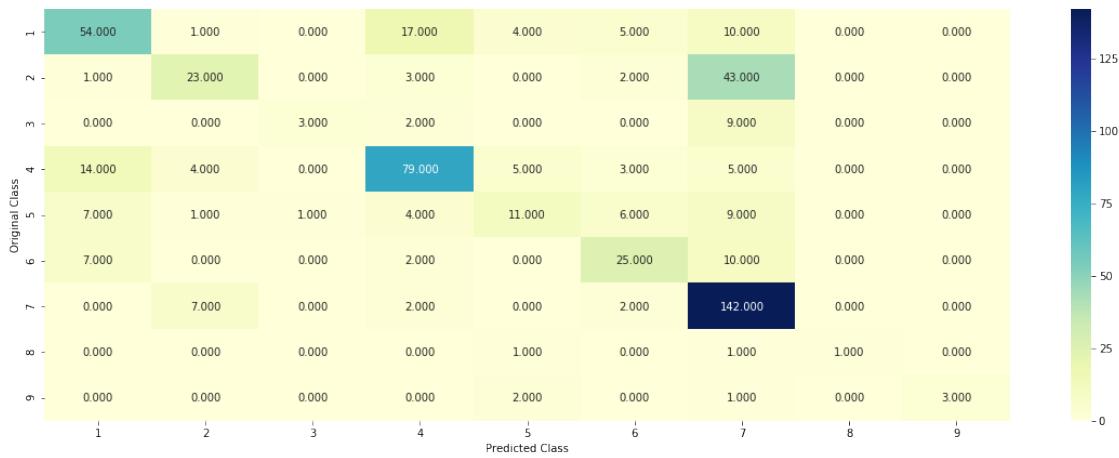
```

```

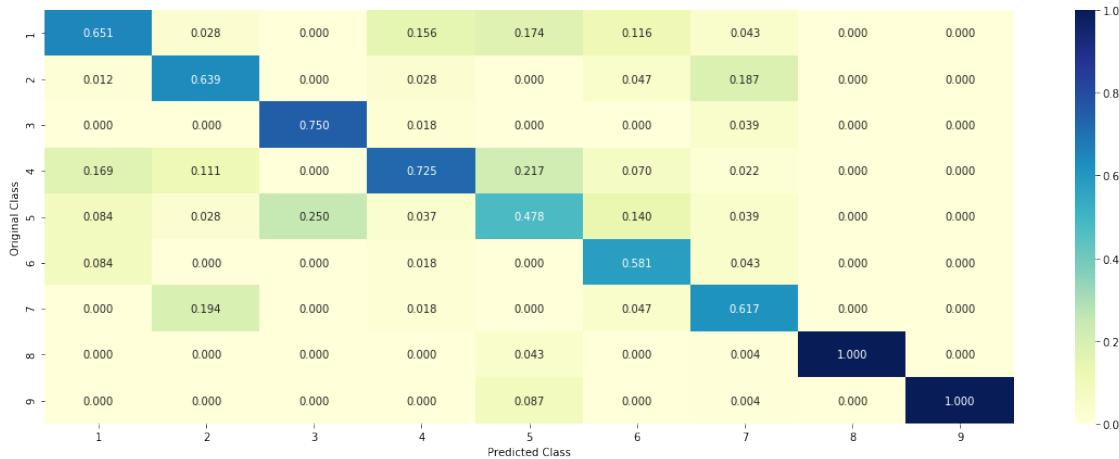
Log loss : 1.0928612451433617
Number of mis-classified points : 0.35902255639097747

```

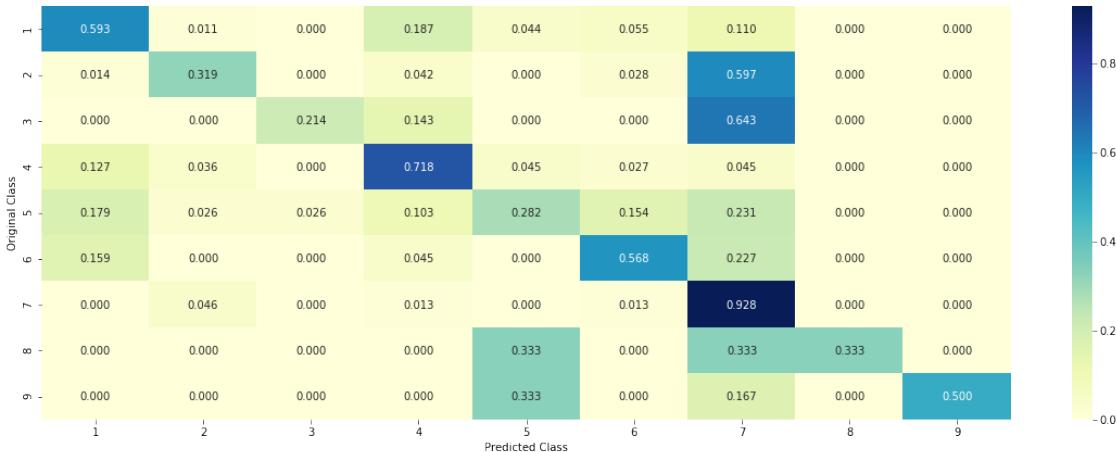
----- Confusion matrix -----



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



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



## Feature Importance

```
In [90]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_tfidfCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCoding[test_point_index])))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'])
```

Predicted Class : 7  
Predicted Class Probabilities: [[0.0436 0.0461 0.011 0.0413 0.0242 0.0141 0.8137 0.0035 0.0026]  
Actual Class : 2

---

0 Text feature [egfr] present in test data point [True]  
42 Text feature [map2k1] present in test data point [True]  
46 Text feature [akt] present in test data point [True]  
48 Text feature [pi3k] present in test data point [True]  
55 Text feature [ponatinib] present in test data point [True]  
151 Text feature [fusion] present in test data point [True]  
160 Text feature [egf] present in test data point [True]  
169 Text feature [pdgfr] present in test data point [True]  
355 Text feature [gnas] present in test data point [True]  
357 Text feature [dovitinib] present in test data point [True]  
363 Text feature [lung] present in test data point [True]  
374 Text feature [akt1] present in test data point [True]

```

378 Text feature [erk] present in test data point [True]
415 Text feature [fgfr4] present in test data point [True]
417 Text feature [ttf] present in test data point [True]
Out of the top 500 features 15 are present in query point

```

```

In [92]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
          clf.fit(train_x_tfidfCoding, train_y)
          test_point_index = 94
# test_point_index = 100
          no_feature = 500
          predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
          print("Predicted Class :", predicted_cls[0])
          print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCoding[test_point_index]), 3))
          print("Actual Class :", test_y[test_point_index])
          indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
          print("-"*50)
          get_imfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index], test_df['Genre'].iloc[test_point_index])

```

Predicted Class : 7  
Predicted Class Probabilities: [[0.064 0.165 0.0117 0.0438 0.0409 0.0264 0.6389 0.0042 0.0051 0.0001 0.0001 0.0001 0.0001 0.0001]]  
Actual Class : 7

---

```

40 Text feature [flt3] present in test data point [True]
151 Text feature [fusion] present in test data point [True]
154 Text feature [gists] present in test data point [True]
162 Text feature [stat3] present in test data point [True]
166 Text feature [mast] present in test data point [True]
169 Text feature [pdgfr] present in test data point [True]
341 Text feature [d816v] present in test data point [True]
354 Text feature [rg] present in test data point [True]
Out of the top 500 features 8 are present in query point

```

### 6.1.5.Random Forest Classifier

```

In [93]: alpha = [100,200,500,1000,2000]
          max_depth = [5,10,20]
          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, random_state=42)
                  clf.fit(train_x_tfidfCoding, train_y)
                  sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
                  sig_clf.fit(train_x_tfidfCoding, train_y)
                  sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding)
                  cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
                  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_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 = RandomForestClassifier(n_estimators=alpha[int(best_alpha/3)], criterion='gini',
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding)
print('For values of best estimator = ', alpha[int(best_alpha/3)], 'depth = ',alpha[int(best_alpha/3)])
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding)
print('For values of best estimator = ', alpha[int(best_alpha/3)], 'depth = ',alpha[int(best_alpha/3)])
predict_y = sig_clf.predict_proba(test_x_tfidfCoding)
print('For values of best estimator = ', alpha[int(best_alpha/3)], 'depth = ',alpha[int(best_alpha/3)])

for n_estimators = 100 and max depth =  5
Log Loss : 1.2259488637032034
for n_estimators = 100 and max depth =  10
Log Loss : 1.1116466214762053
for n_estimators = 100 and max depth =  20
Log Loss : 1.0539424608684294
for n_estimators = 200 and max depth =  5
Log Loss : 1.2159587752863987
for n_estimators = 200 and max depth =  10
Log Loss : 1.1040051345152648
for n_estimators = 200 and max depth =  20
Log Loss : 1.0483142904486877
for n_estimators = 500 and max depth =  5
Log Loss : 1.20354726195443
for n_estimators = 500 and max depth =  10
Log Loss : 1.0960645024261577
for n_estimators = 500 and max depth =  20
Log Loss : 1.0385435717021625
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2064648977485972
for n_estimators = 1000 and max depth =  10
Log Loss : 1.0968086109156803

```

```

for n_estimators = 1000 and max depth = 20
Log Loss : 1.0391507689640218
for n_estimators = 2000 and max depth = 5
Log Loss : 1.2066484150351522
for n_estimators = 2000 and max depth = 10
Log Loss : 1.0991647443965207
for n_estimators = 2000 and max depth = 20
Log Loss : 1.0382605281606945
For values of best estimator = 2000 depth = 500 The train log loss is: 0.5437952439636112
For values of best estimator = 2000 depth = 500 The cross validation log loss is: 1.0382605281606945
For values of best estimator = 2000 depth = 500 The test log loss is: 1.0417213057842747

```

In [100]: #test

```

clf = RandomForestClassifier(n_estimators=2000, criterion='gini', max_depth=20, random_state=42)
sig_clf,temp=predict_and_plot_confusion_matrix(train_x_tfidfCoding, train_y, cv_x_tfidfCoding, cv_y)

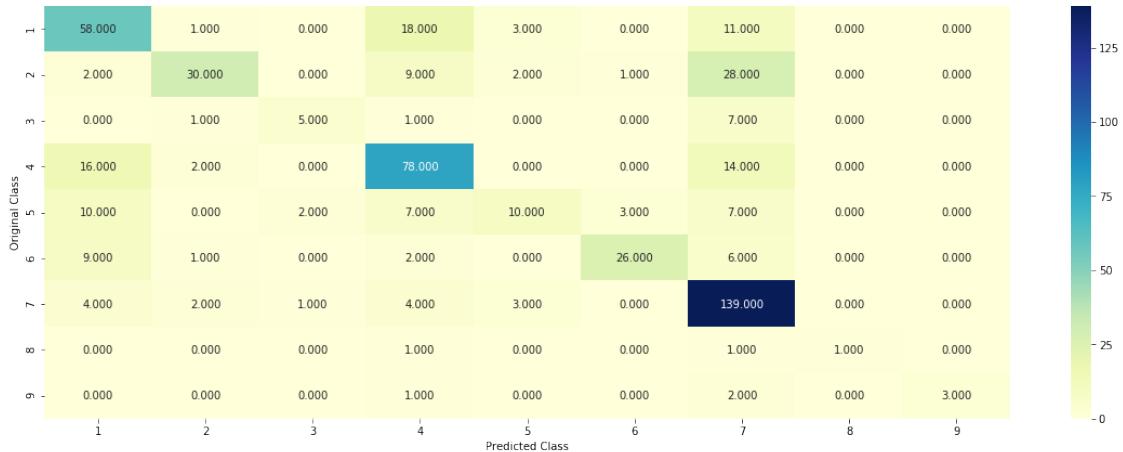
list_data = []
list_data.append('Tf_Idf_Top1000+RF')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_tfidfCoding), eps=1e-15))
list_data.append('estimators = '+str(2000)+ ' ' + 'depth = '+str(20))
list_data.append(temp)
final_results.append(list_data)

```

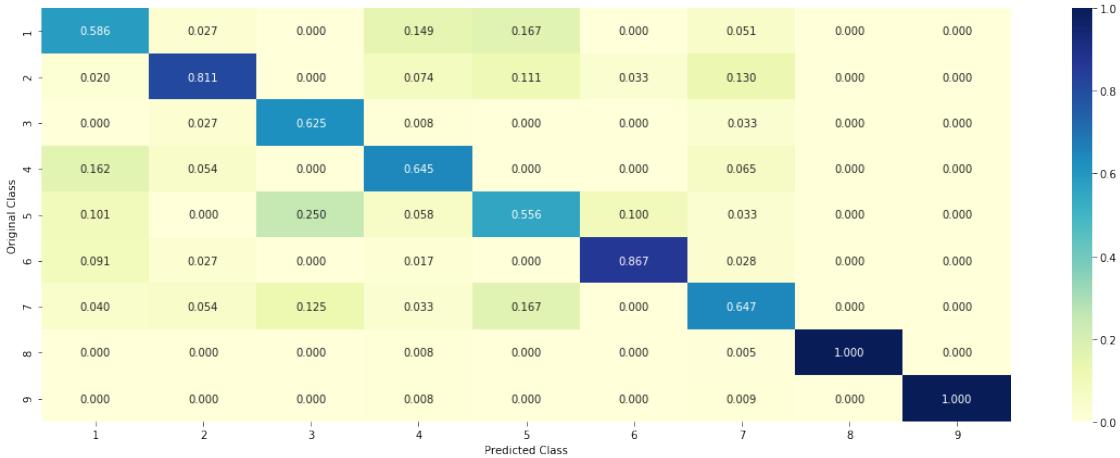
Log loss : 1.0382605281606945

Number of mis-classified points : 0.34210526315789475

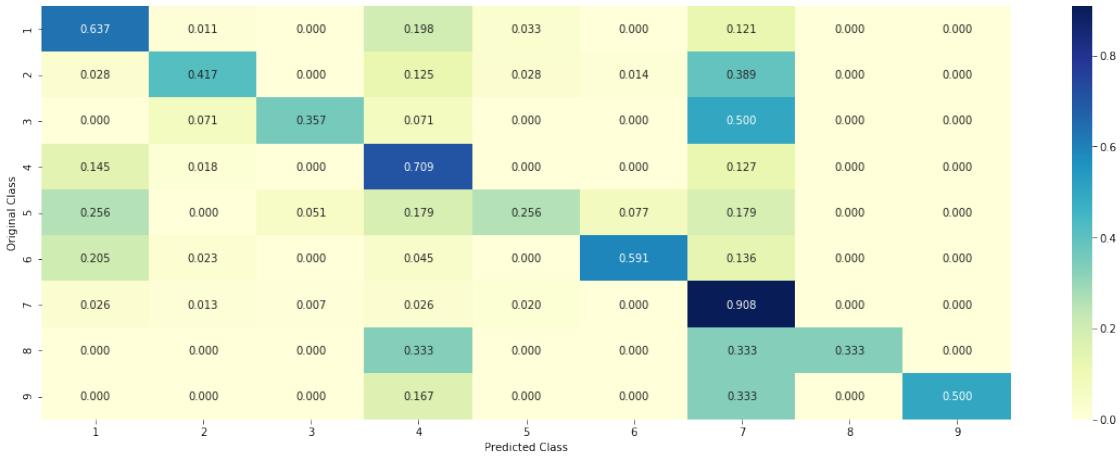
----- Confusion matrix -----



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



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



## Feature importance

```
In [101]: # test_point_index = 10
clf = RandomForestClassifier(n_estimators=200, criterion='gini', max_depth=20, random_state=42)
clf.fit(train_x_tfidfCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding, train_y)

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

```

print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfC)
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_imptfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index], te
Predicted Class : 7
Predicted Class Probabilities: [[0.0596 0.2382 0.0153 0.033  0.0453 0.0388 0.56   0.005  0.004
Actual Class : 2
-----
0 Text feature [patients] present in test data point [True]
1 Text feature [akt] present in test data point [True]
2 Text feature [variants] present in test data point [True]
5 Text feature [resistance] present in test data point [True]
6 Text feature [pten] present in test data point [True]
9 Text feature [mutations] present in test data point [True]
10 Text feature [erk] present in test data point [True]
11 Text feature [splicing] present in test data point [True]
13 Text feature [al] present in test data point [True]
14 Text feature [et] present in test data point [True]
15 Text feature [lung] present in test data point [True]
16 Text feature [breast] present in test data point [True]
18 Text feature [ovarian] present in test data point [True]
29 Text feature [egfr] present in test data point [True]
30 Text feature [fusion] present in test data point [True]
31 Text feature [nm] present in test data point [True]
38 Text feature [tki] present in test data point [True]
40 Text feature [braf] present in test data point [True]
42 Text feature [pi3k] present in test data point [True]
43 Text feature [fusions] present in test data point [True]
44 Text feature [wildtype] present in test data point [True]
55 Text feature [tp53] present in test data point [True]
56 Text feature [pik3ca] present in test data point [True]
61 Text feature [kras] present in test data point [True]
63 Text feature [pdgfra] present in test data point [True]
65 Text feature [prostate] present in test data point [True]
73 Text feature [erlotinib] present in test data point [True]
74 Text feature [akt1] present in test data point [True]
75 Text feature [egf] present in test data point [True]
76 Text feature [gefitinib] present in test data point [True]
79 Text feature [bladder] present in test data point [True]
85 Text feature [renal] present in test data point [True]
96 Text feature [pdgfr] present in test data point [True]
97 Text feature [fgfgr1] present in test data point [True]
98 Text feature [methylation] present in test data point [True]
Out of the top 100 features 35 are present in query point

```

In [102]: test\_point\_index = 52

```

no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidfCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidfCoding[test_point_index]), 3))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_imptfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index], test_y[test_point_index])

```

Predicted Class : 7  
Predicted Class Probabilities: [[0.0462 0.0554 0.0139 0.0363 0.0361 0.0305 0.7727 0.0043 0.0046 0.0046]  
Actual Class : 7

---

4 Text feature [kit] present in test data point [True]  
5 Text feature [resistance] present in test data point [True]  
9 Text feature [mutations] present in test data point [True]  
10 Text feature [erk] present in test data point [True]  
15 Text feature [lung] present in test data point [True]  
16 Text feature [breast] present in test data point [True]  
18 Text feature [ovarian] present in test data point [True]  
19 Text feature [ras] present in test data point [True]  
22 Text feature [classification] present in test data point [True]  
30 Text feature [fusion] present in test data point [True]  
31 Text feature [nm] present in test data point [True]  
39 Text feature [specimens] present in test data point [True]  
40 Text feature [braf] present in test data point [True]  
45 Text feature [3d] present in test data point [True]  
49 Text feature [melanoma] present in test data point [True]  
60 Text feature [foci] present in test data point [True]  
61 Text feature [kras] present in test data point [True]  
64 Text feature [alk] present in test data point [True]  
69 Text feature [colorectal] present in test data point [True]  
80 Text feature [mek1] present in test data point [True]  
93 Text feature [gastric] present in test data point [True]  
Out of the top 100 features 21 are present in query point

### 5.1.7.Stack the models

```

In [105]: clf1 = SGDClassifier(alpha=0.0001328, penalty='l2', loss='log', class_weight='balanced')
          clf1.fit(train_x_tfidfCoding, train_y)
          sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

          clf2 = SGDClassifier(alpha=0.000371, penalty='l2', loss='hinge', random_state=0)
          clf2.fit(train_x_tfidfCoding, train_y)
          sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")

```

```

clf3 = KNeighborsClassifier(n_neighbors=5)
clf3.fit(train_x_tfidfCoding, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")

sig_clf1.fit(train_x_tfidfCoding, train_y)
print("Logistic Regression : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_tfidfCoding))))
sig_clf2.fit(train_x_tfidfCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(cv_x_tfidfCoding))))
sig_clf3.fit(train_x_tfidfCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_tfidfCoding))))
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_classifier=lr)
    sclf.fit(train_x_tfidfCoding, train_y)
    print("Stacking Classifier : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sclf.predict_proba(cv_x_tfidfCoding))))
    if best_alpha > log_error:
        best_alpha = log_error

Logistic Regression : Log Loss: 1.04
Support vector machines : Log Loss: 1.09
Naive Bayes : Log Loss: 1.10
-----
Stacking Classifier : for the value of alpha: 0.000100 Log Loss: 2.174
Stacking Classifier : for the value of alpha: 0.001000 Log Loss: 2.006
Stacking Classifier : for the value of alpha: 0.010000 Log Loss: 1.443
Stacking Classifier : for the value of alpha: 0.100000 Log Loss: 1.104
Stacking Classifier : for the value of alpha: 1.000000 Log Loss: 1.251
Stacking Classifier : for the value of alpha: 10.000000 Log Loss: 1.670

```

```

In [106]: #alpha = [0.0001,0.001,0.01,0.1,1,10]
alpha = np.random.uniform(0.005,0.5,10)
alpha = np.round(alpha,5)
alpha.sort()
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr)
    sclf.fit(train_x_tfidfCoding, train_y)
    print("Stacking Classifier : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sclf.predict_proba(cv_x_tfidfCoding))))
    log_error = log_loss(cv_y, sclf.predict_proba(cv_x_tfidfCoding))
    if best_alpha > log_error:
        best_alpha = log_error

```

Stacking Classifier : for the value of alpha: 0.049100 Log Loss: 1.147

```
Stacking Classifier : for the value of alpha: 0.082210 Log Loss: 1.111
Stacking Classifier : for the value of alpha: 0.103280 Log Loss: 1.103
Stacking Classifier : for the value of alpha: 0.120860 Log Loss: 1.101
Stacking Classifier : for the value of alpha: 0.294000 Log Loss: 1.126
Stacking Classifier : for the value of alpha: 0.294300 Log Loss: 1.127
Stacking Classifier : for the value of alpha: 0.355400 Log Loss: 1.140
Stacking Classifier : for the value of alpha: 0.366200 Log Loss: 1.142
Stacking Classifier : for the value of alpha: 0.406190 Log Loss: 1.150
Stacking Classifier : for the value of alpha: 0.407360 Log Loss: 1.151
```

In [107]: #testing

```
lr = LogisticRegression(C=0.120860)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr)
sclf.fit(train_x_tfidfCoding, train_y)

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

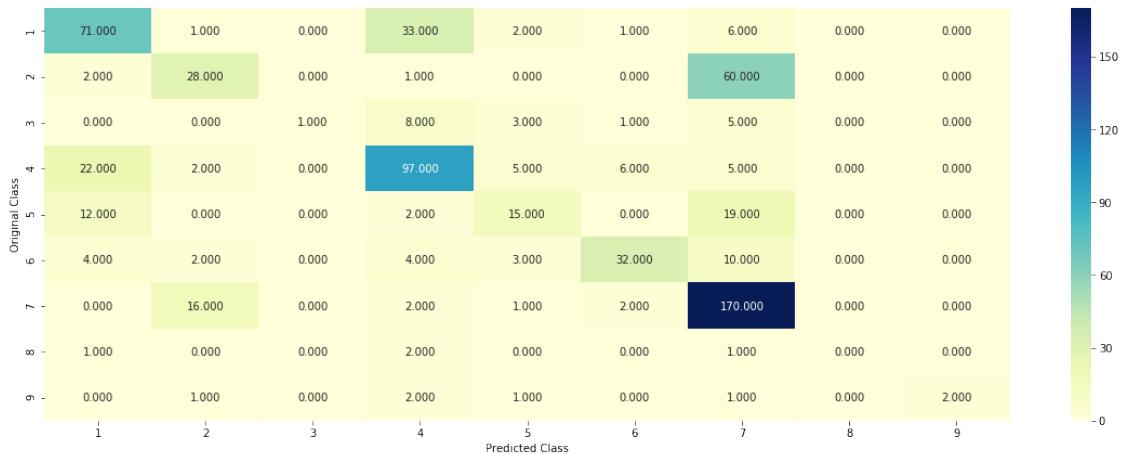
log_error = log_loss(cv_y, sclf.predict_proba(cv_x_tfidfCoding))
print("Log loss (CV) on the stacking classifier :",log_error)

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

print("Number of missclassified point :", np.count_nonzero((sclf.predict(test_x_tfidfCoding)- test_y)))
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_tfidfCoding))

list_data = []
list_data.append('Tf_Idf_Top1000+Stackig(LR+KNN+LSVC)')
list_data.append(log_loss(y_train, sclf.predict_proba(train_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_cv, sclf.predict_proba(cv_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_test, sclf.predict_proba(test_x_tfidfCoding), eps=1e-15))
list_data.append('C = '+str(0.121800))
list_data.append(np.count_nonzero((sclf.predict(test_x_tfidfCoding)- test_y))/test_y)
final_results.append(list_data)

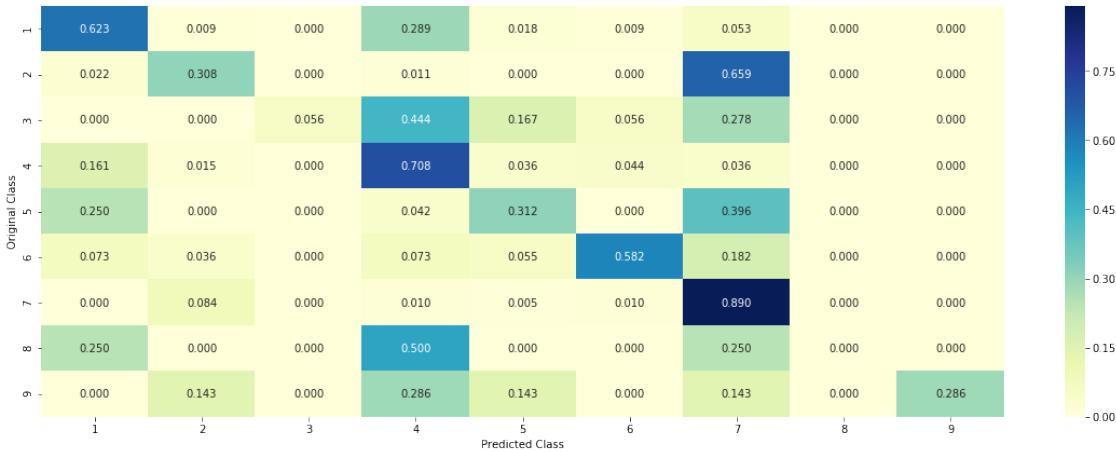
Log loss (train) on the stacking classifier : 0.3748167505829858
Log loss (CV) on the stacking classifier : 1.1007163551349828
Log loss (test) on the stacking classifier : 1.080307626525738
Number of missclassified point : 0.3744360902255639
----- Confusion matrix -----
```



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



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

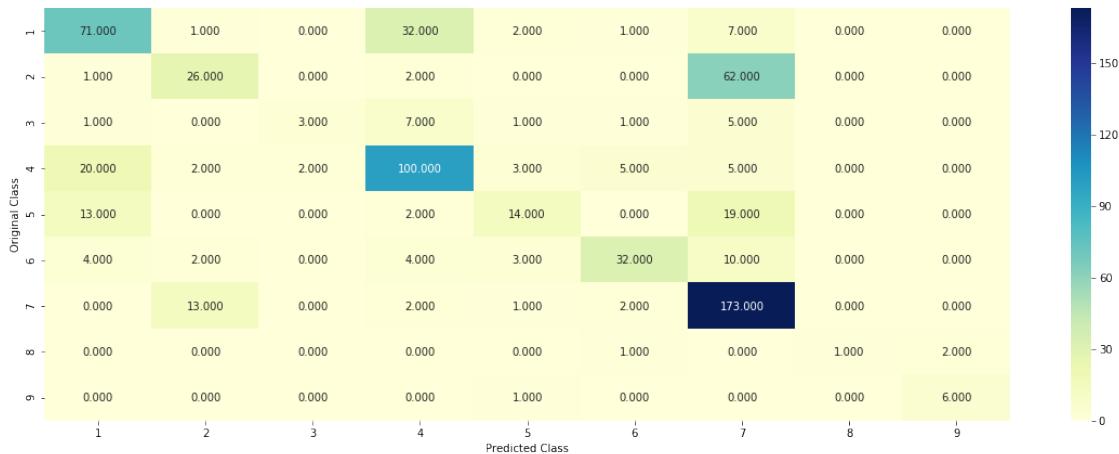


### 6.1.8. Maximum Voting classifier

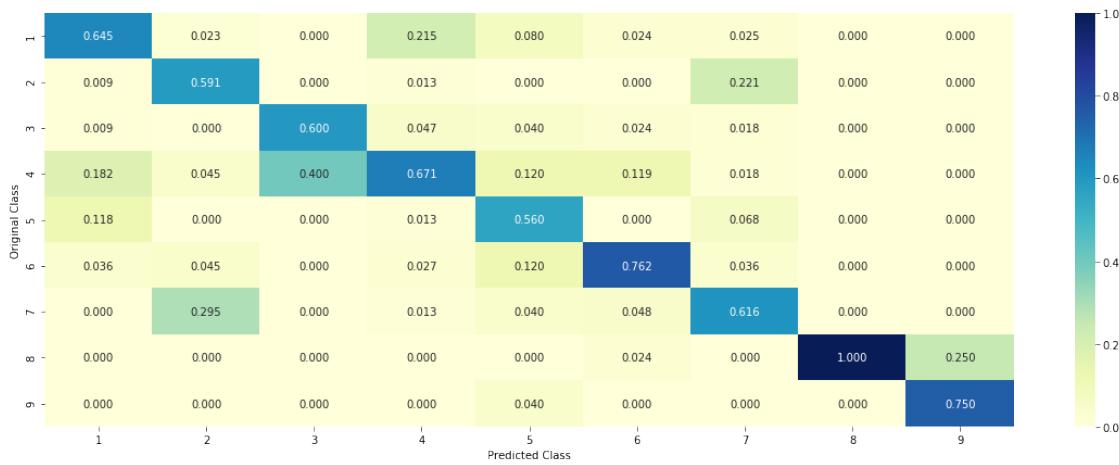
```
In [111]: from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)])
vclf.fit(train_x_tfidfCoding, train_y)
print("Log loss (train) on the VotingClassifier : ", log_loss(train_y, vclf.predict_proba(train_x_tfidfCoding)))
print("Log loss (CV) on the VotingClassifier : ", log_loss(cv_y, vclf.predict_proba(cv_x_tfidfCoding)))
print("Log loss (test) on the VotingClassifier : ", log_loss(test_y, vclf.predict_proba(test_x_tfidfCoding)))
print("Number of missclassified point : ", np.count_nonzero((vclf.predict(test_x_tfidfCoding)) - test_y))
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_tfidfCoding))

list_data = []
list_data.append('Tf_Idf+MaxVote(LR+KNN+LSVC)')
list_data.append(log_loss(y_train, sclf.predict_proba(train_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_cv, sclf.predict_proba(cv_x_tfidfCoding), eps=1e-15))
list_data.append(log_loss(y_test, sclf.predict_proba(test_x_tfidfCoding), eps=1e-15))
list_data.append(' ')
list_data.append(np.count_nonzero((sclf.predict(test_x_tfidfCoding)) - test_y))/test_y
final_results.append(list_data)

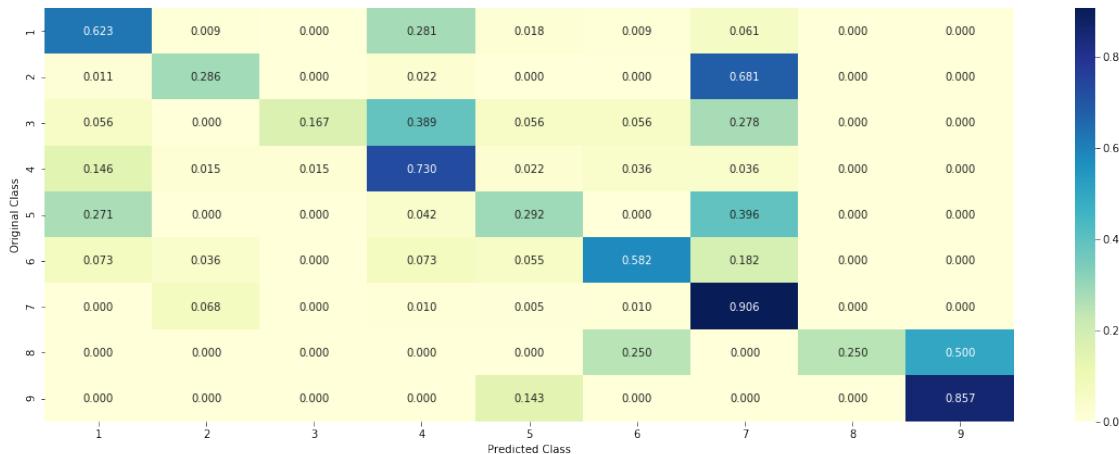
Log loss (train) on the VotingClassifier : 0.5665707047992947
Log loss (CV) on the VotingClassifier : 1.0513859372110699
Log loss (test) on the VotingClassifier : 1.0399771081290223
Number of missclassified point : 0.3593984962406015
----- Confusion matrix -----
```



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



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



## 7.Uni-Bi grams for Bow and Tfifdf

```
In [58]: # 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 [59]: # tfidf encoding of Gene feature.
gene_vectorizer = TfidfVectorizer()
train_gene_feature_tfidfCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_tfidfCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_tfidfCoding = gene_vectorizer.transform(cv_df['Gene'])

In [60]: # one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])

In [61]: # tfidf_transpidf encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_tfidfCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_tfidfCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_tfidfCoding = variation_vectorizer.transform(cv_df['Variation'])

In [62]: # building a CountVectorizer with all the words that occurred minimum 3 times in train
text_vectorizer = CountVectorizer(min_df=3,ngram_range=(1,2))
train_text_feature_onehotCoding12 = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features12 = text_vectorizer.get_feature_names()

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

```

train_textfea_counts12 = train_textfeature_onehotCoding12.sum(axis=0).A1

# zip(list(text_features),text_fea_counts) will zip a word with its number of times i
textfea_dict12 = dict(zip(list(train_text_features12),train_textfea_counts12))

print("Total number of unique words in train data :", len(train_text_features12))

# don't forget to normalize every feature
train_textfeature_onehotCoding12 = normalize(train_textfeature_onehotCoding12, axis=0)

# we use the same vectorizer that was trained on train data
test_textfeature_onehotCoding12 = text_vectorizer.transform(test_df['TEXT'])
# don't forget to normalize every feature
test_textfeature_onehotCoding12 = normalize(test_textfeature_onehotCoding12, axis=0)

# we use the same vectorizer that was trained on train data
cv_textfeature_onehotCoding12 = text_vectorizer.transform(cv_df['TEXT'])
# don't forget to normalize every feature
cv_textfeature_onehotCoding12 = normalize(cv_textfeature_onehotCoding12, axis=0)

```

Total number of unique words in train data : 771668

```

In [63]: # building a CountVectorizer with all the words that occurred minimum 3 times in train
text_vectorizer = TfidfVectorizer(min_df=3,ngram_range=(1,2))
train_textfeature_tfidfCoding12 = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features12 = text_vectorizer.get_feature_names()

# train_textfeature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*nu
train_textfea_counts12 = train_textfeature_tfidfCoding12.sum(axis=0).A1

# zip(list(text_features),text_fea_counts) will zip a word with its number of times i
textfea_dict12 = dict(zip(list(train_text_features12),train_textfea_counts12))

print("Total number of unique words in train data :", len(train_text_features12))

# we use the same vectorizer that was trained on train data
test_textfeature_tfidfCoding12 = text_vectorizer.transform(test_df['TEXT'])

# we use the same vectorizer that was trained on train data
cv_textfeature_tfidfCoding12 = text_vectorizer.transform(cv_df['TEXT'])

```

Total number of unique words in train data : 771668

## Stacking

```
In [64]: #Bow
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_onehotCoding12 = hstack((train_gene_var_onehotCoding, train_text_feature_onehotC
train_y = np.array(list(train_df['Class'])))

test_x_onehotCoding12 = hstack((test_gene_var_onehotCoding, test_text_feature_onehotC
test_y = np.array(list(test_df['Class'])))

cv_x_onehotCoding12 = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding12
cv_y = np.array(list(cv_df['Class'])))

In [65]: #Tf-Idf
train_gene_var_tfidfCoding = hstack((train_gene_feature_tfidfCoding,train_variation_f
test_gene_var_tfidfCoding = hstack((test_gene_feature_tfidfCoding,test_variation_featu
cv_gene_var_tfidfCoding = hstack((cv_gene_feature_tfidfCoding, cv_variation_feature_tfi

train_x_tfidfCoding12 = hstack((train_gene_var_tfidfCoding, train_text_feature_tfidfC
train_y = np.array(list(train_df['Class'])))

test_x_tfidfCoding12 = hstack((test_gene_var_tfidfCoding, test_text_feature_tfidfCodin
test_y = np.array(list(test_df['Class'])))

cv_x_tfidfCoding12 = hstack((cv_gene_var_tfidfCoding, cv_text_feature_tfidfCoding12))
cv_y = np.array(list(cv_df['Class']))
```

## 7.1. Logistic Regression

### 7.1.1. With BoW

#### 7.1.1.1. With Class Balancing

```
In [121]: alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', r
    clf.fit(train_x_onehotCoding12, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding12, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding12)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

```

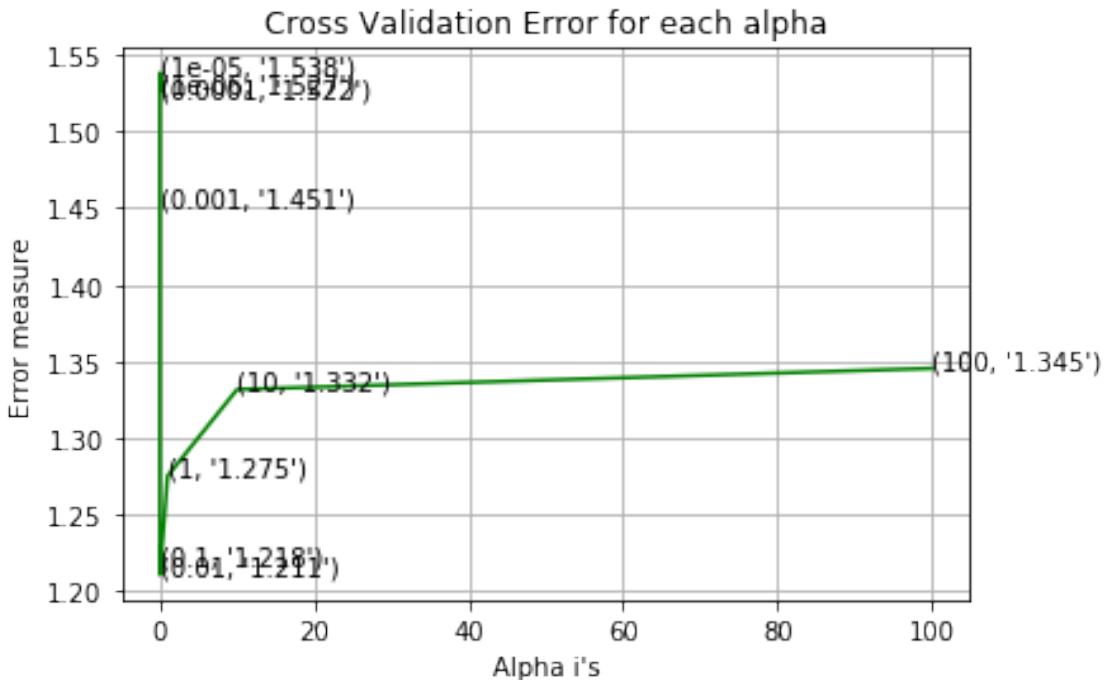
        ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
clf.fit(train_x_onehotCoding12, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding12, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_
predict_y = sig_clf.predict_proba(test_x_onehotCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_

for alpha = 1e-06
Log Loss : 1.526523578765235
for alpha = 1e-05
Log Loss : 1.5381540319014362
for alpha = 0.0001
Log Loss : 1.522395551332507
for alpha = 0.001
Log Loss : 1.4511168491746227
for alpha = 0.01
Log Loss : 1.210832062018533
for alpha = 0.1
Log Loss : 1.217565359775422
for alpha = 1
Log Loss : 1.2753156732923774
for alpha = 10
Log Loss : 1.3317153731701312
for alpha = 100
Log Loss : 1.3454995107329675

```



```
For values of best alpha = 0.01 The train log loss is: 0.8537315993842852
For values of best alpha = 0.01 The cross validation log loss is: 1.210832062018533
For values of best alpha = 0.01 The test log loss is: 1.2172759801902582
```

```
In [123]: #alpha = [10 ** x for x in range(-6, 3)]
alpha = np.random.uniform(0.0005,0.05,15)
alpha = np.round(alpha,6)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', )
    clf.fit(train_x_onehotCoding12, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding12, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding12)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

```

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

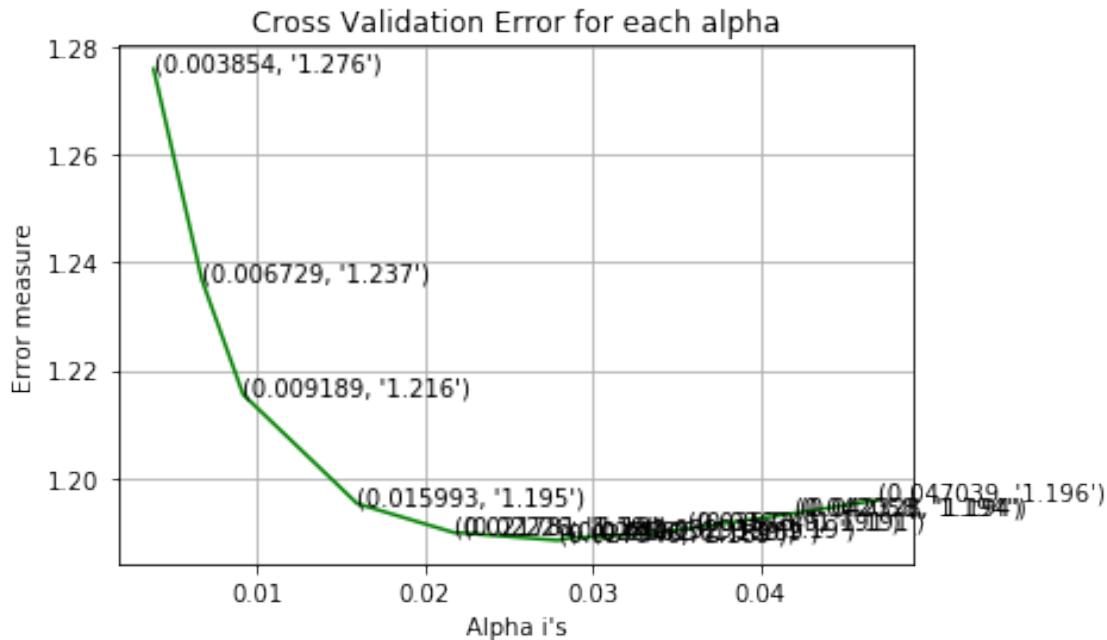
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log')
clf.fit(train_x_onehotCoding12, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding12, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(train_y, predict_y))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(cv_y, predict_y))
predict_y = sig_clf.predict_proba(test_x_onehotCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(test_y, predict_y))

for alpha = 0.003854
Log Loss : 1.2758359485556872
for alpha = 0.006729
Log Loss : 1.2367762597292238
for alpha = 0.009189
Log Loss : 1.215522689177353
for alpha = 0.015993
Log Loss : 1.1952938258779133
for alpha = 0.021781
Log Loss : 1.1899025608413587
for alpha = 0.02226
Log Loss : 1.1898585260609569
for alpha = 0.027946
Log Loss : 1.1885000305485867
for alpha = 0.028561
Log Loss : 1.1888935169783645
for alpha = 0.030036
Log Loss : 1.1891870265930138
for alpha = 0.032939
Log Loss : 1.1895491194379844
for alpha = 0.03573
Log Loss : 1.1906871954528666
for alpha = 0.036249
Log Loss : 1.190899987906779
for alpha = 0.042058
Log Loss : 1.193521472604063
for alpha = 0.042328
Log Loss : 1.193648235605448

```

```
for alpha = 0.047039
Log Loss : 1.1959383277214997
```

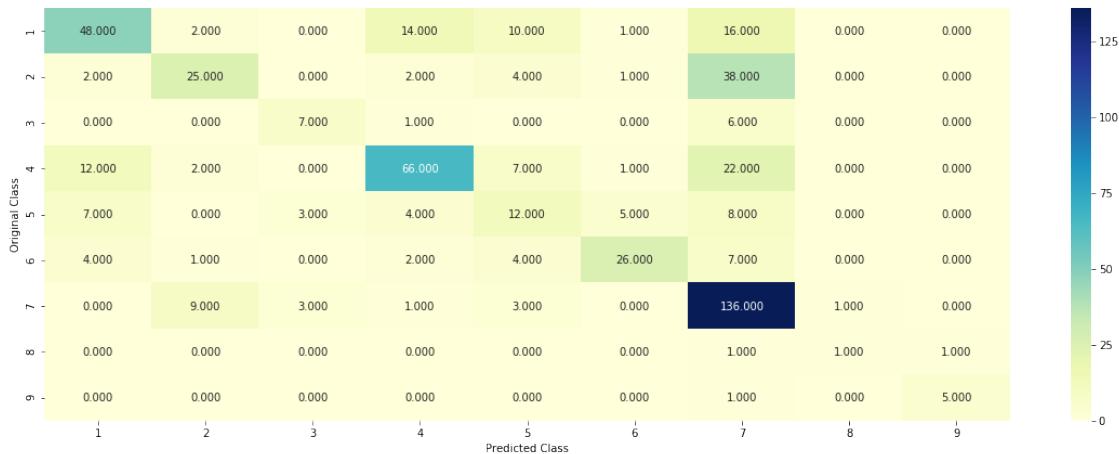


```
For values of best alpha = 0.027946 The train log loss is: 0.7894568411369551
For values of best alpha = 0.027946 The cross validation log loss is: 1.1885000305485867
For values of best alpha = 0.027946 The test log loss is: 1.1873909016363215
```

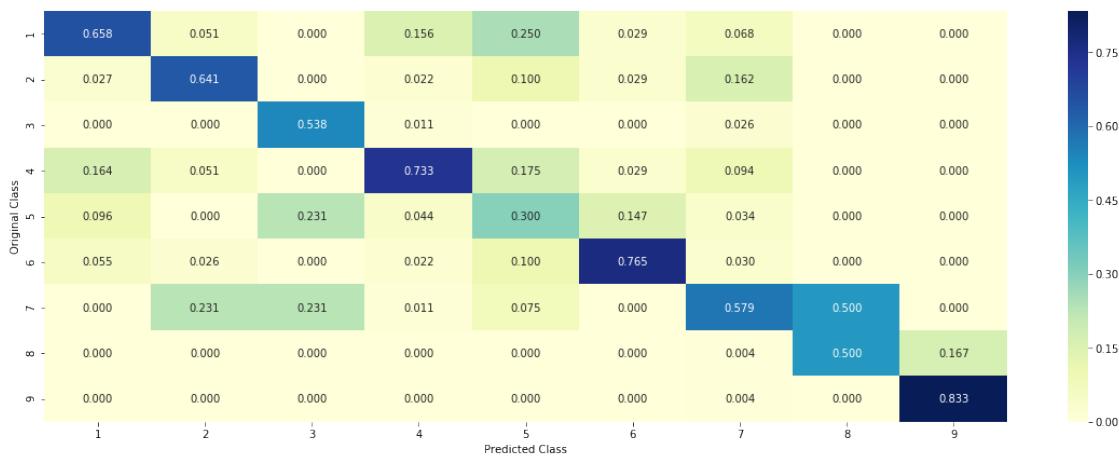
```
In [127]: #testing
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', max_iter=1000)
sig_clf,temp = predict_and_plot_confusion_matrix(train_x_onehotCoding12, train_y, cv)

list_data = []
list_data.append('Bow+Uni-Bi_gram+SGD-LR+Class_Balance')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding12), eps=1e-15))
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding12), eps=1e-15))
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding12), eps=1e-15))
list_data.append('alpha = '+str(clf.alpha))
list_data.append(np.count_nonzero((sig_clf.predict(test_x_onehotCoding12)- test_y))/len(test_y))
final_results.append(list_data)

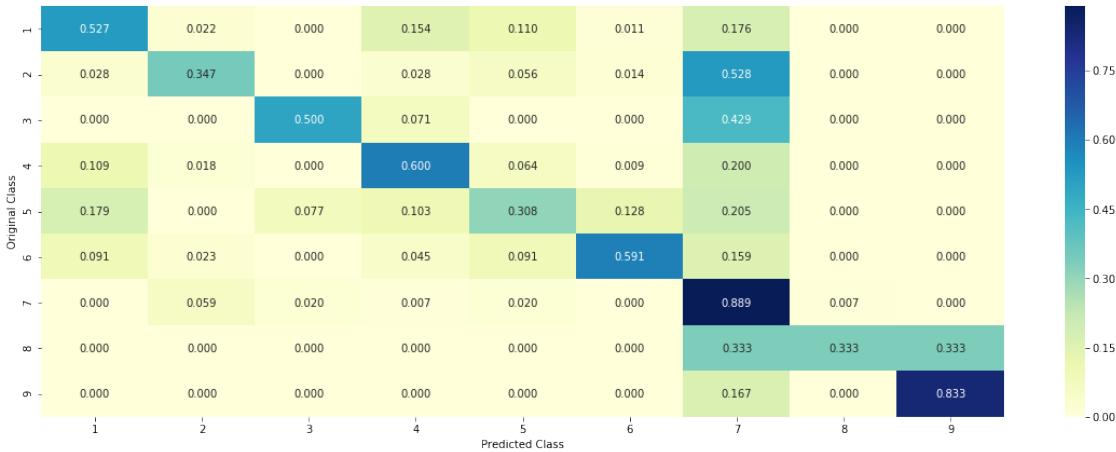
Log loss : 1.1885000305485867
Number of mis-classified points : 0.38721804511278196
----- Confusion matrix -----
```



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



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



### 7.1.1.2. Without Class Balancing

```
In [128]: alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding12, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding12, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding12)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilités we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

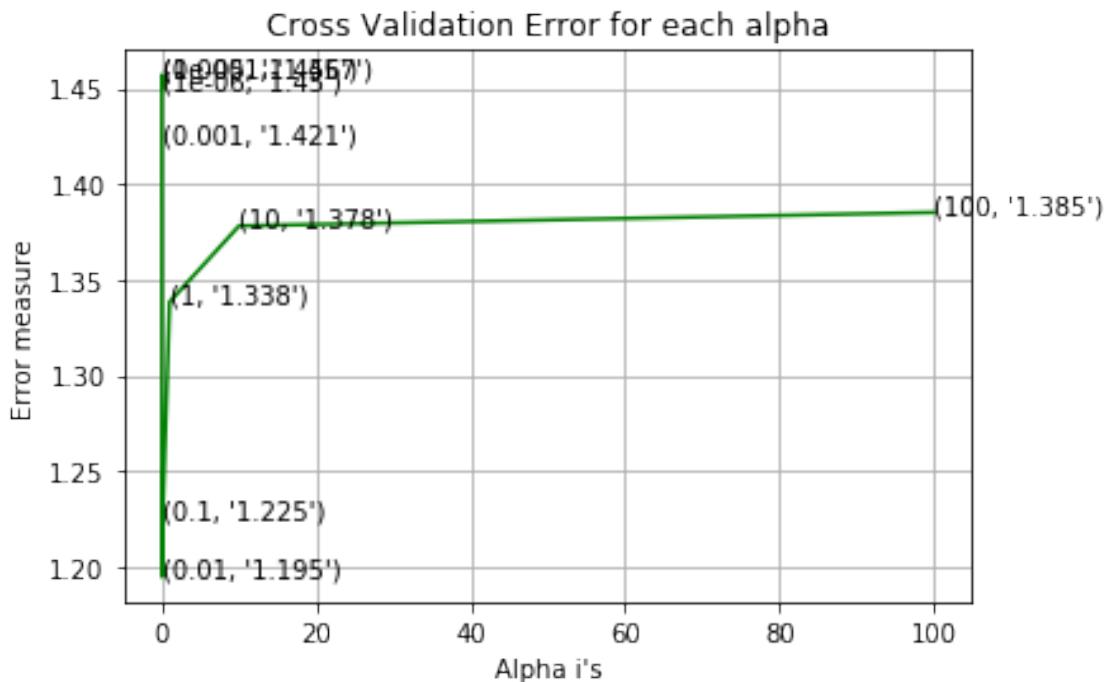
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding12, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding12, train_y)
```

```

predict_y = sig_clf.predict_proba(train_x_onehotCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log lo
predict_y = sig_clf.predict_proba(test_x_onehotCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_)

for alpha = 1e-06
Log Loss : 1.450243518830451
for alpha = 1e-05
Log Loss : 1.4563904979410016
for alpha = 0.0001
Log Loss : 1.4571791175509323
for alpha = 0.001
Log Loss : 1.421490785904314
for alpha = 0.01
Log Loss : 1.1945005275034963
for alpha = 0.1
Log Loss : 1.2249788391978451
for alpha = 1
Log Loss : 1.3383697476356289
for alpha = 10
Log Loss : 1.3781644667597925
for alpha = 100
Log Loss : 1.385198442913747

```



```

For values of best alpha = 0.01 The train log loss is: 0.8540764462576208
For values of best alpha = 0.01 The cross validation log loss is: 1.1945005275034963
For values of best alpha = 0.01 The test log loss is: 1.2070189591636151

```

```

In [129]: alpha = np.random.uniform(0.0005,0.05,15)
alpha = np.round(alpha,6)
alpha.sort()
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_onehotCoding12, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding12, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding12)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding12, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding12, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(predict_y,train_y))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(predict_y,cv_y))
predict_y = sig_clf.predict_proba(test_x_onehotCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(predict_y,test_y))

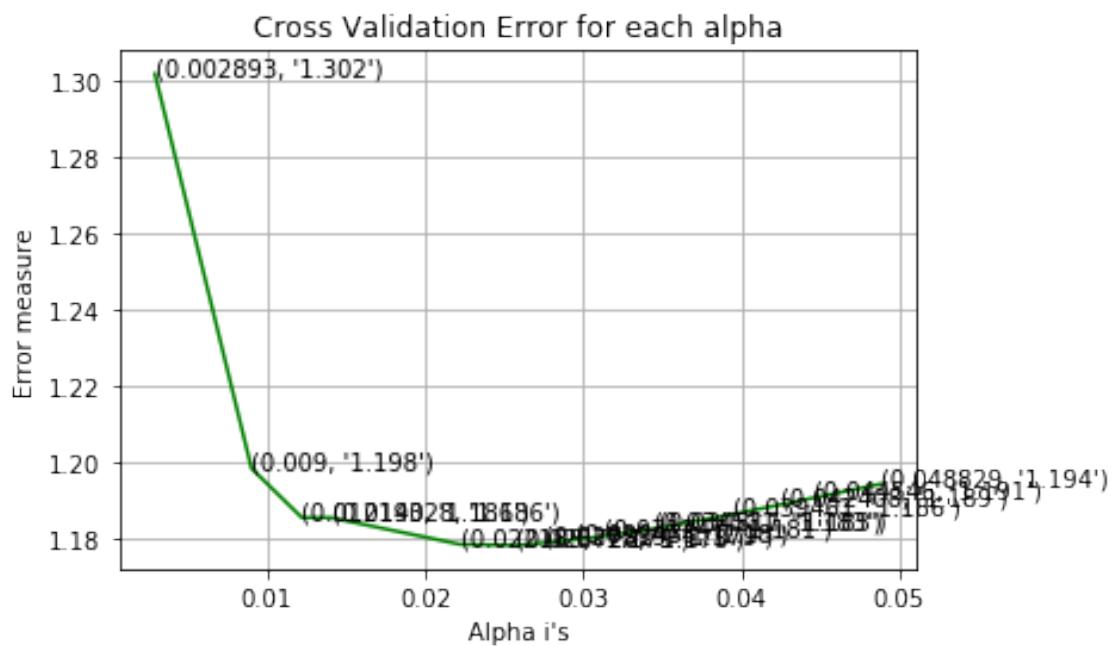
for alpha = 0.002893
Log Loss : 1.301519974118112
for alpha = 0.009
Log Loss : 1.1984333793410649

```

```

for alpha = 0.012193
Log Loss : 1.1855219046920584
for alpha = 0.014028
Log Loss : 1.1855277088996579
for alpha = 0.022185
Log Loss : 1.1785682199338936
for alpha = 0.025728
Log Loss : 1.1782991936848162
for alpha = 0.027519
Log Loss : 1.1789264507731831
for alpha = 0.029435
Log Loss : 1.1797850778451444
for alpha = 0.031247
Log Loss : 1.180691455287569
for alpha = 0.034511
Log Loss : 1.1826627094017814
for alpha = 0.034817
Log Loss : 1.1828789939460198
for alpha = 0.039461
Log Loss : 1.1864239264627239
for alpha = 0.042408
Log Loss : 1.188866473129122
for alpha = 0.044546
Log Loss : 1.1906985988784056
for alpha = 0.048829
Log Loss : 1.1943356471717925

```



```

For values of best alpha = 0.025728 The train log loss is: 0.774720129376928
For values of best alpha = 0.025728 The cross validation log loss is: 1.1782991936848162
For values of best alpha = 0.025728 The test log loss is: 1.1761356735449886

```

In [130]: #testing

```

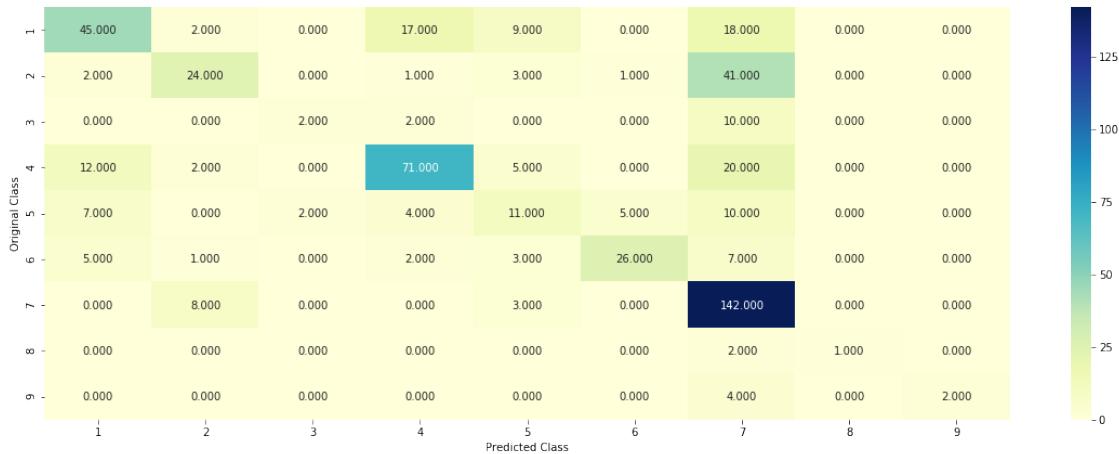
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
sig_clf,temp = predict_and_plot_confusion_matrix(train_x_onehotCoding12, train_y, cv=cv,
list_data = []
list_data.append('Bow+Uni-Bi_gram+SGD-LR+ImClass_Balance')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding12), eps=1e-15))
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding12), eps=1e-15))
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding12), eps=1e-15))
list_data.append('alpha = '+str(clf.alpha))
list_data.append(np.count_nonzero((sig_clf.predict(test_x_onehotCoding12)- test_y))/len(test_y))
final_results.append(list_data)

```

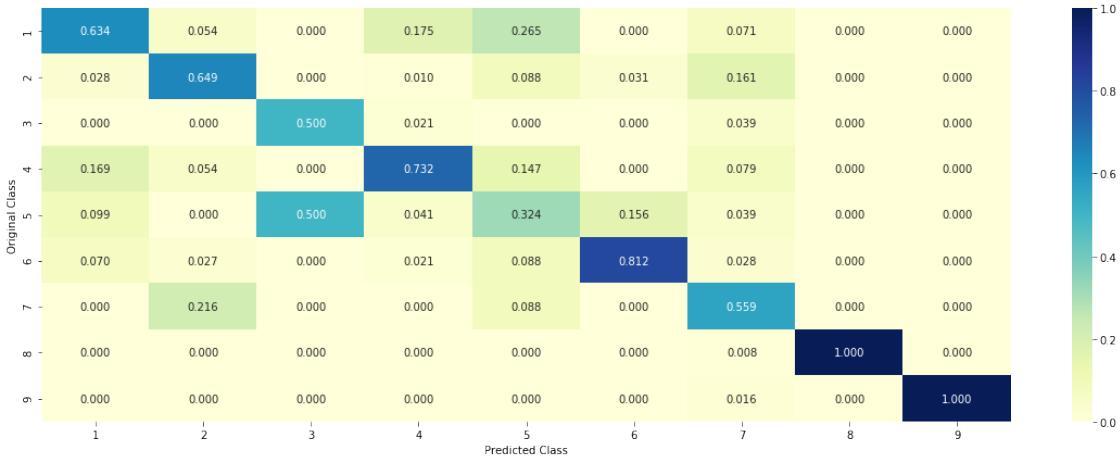
Log loss : 1.1782991936848162

Number of mis-classified points : 0.39097744360902253

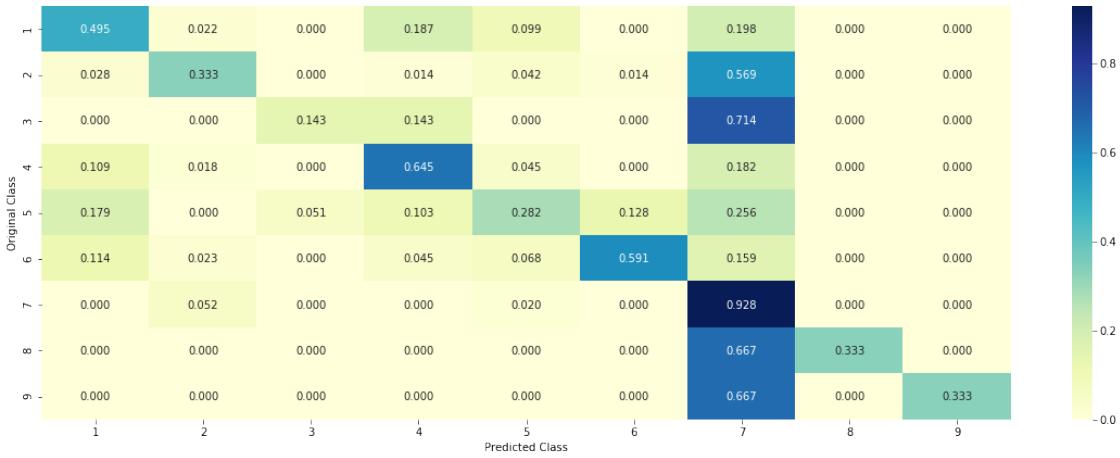
----- Confusion matrix -----



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



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



## Feature Importance

```
In [133]: # 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_imfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    #text_count_vec = TfidfVectorizer(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(v,i)
    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 [{}].format(v,i)
    else:
        temp1 = list(train_text_features12)
        word = temp1[v-(fea1_len+fea2_len)]
        l = word.split()
        if len(l) ==2:
            if l[0] in text.split() and l[1] in text.split():
                yes_no = True
            else:
                yes_no = False
        else:
            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(v,i)

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

```

```

In [135]: # from tabulate import tabulate
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding12,train_y)
test_point_index = 52
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding12[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding12)[0]))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index])

```

Predicted Class : 7  
Predicted Class Probabilities: [[0.0306 0.0506 0.0083 0.0285 0.0242 0.01 0.8426 0.004 0.0012]  
Actual Class : 7

---

37 Text feature [diagnosis poor] present in test data point [True]  
45 Text feature [level clearly] present in test data point [True]  
48 Text feature [fig 20] present in test data point [True]  
56 Text feature [elevated high] present in test data point [True]  
63 Text feature [together may] present in test data point [True]  
74 Text feature [constitutive activation] present in test data point [True]  
76 Text feature [suggesting hras] present in test data point [True]  
77 Text feature [lines among] present in test data point [True]  
97 Text feature [constitutive] present in test data point [True]  
104 Text feature [3t3] present in test data point [True]  
112 Text feature [signaling 15] present in test data point [True]  
120 Text feature [whereas nras] present in test data point [True]  
123 Text feature [may led] present in test data point [True]  
127 Text feature [form resulting] present in test data point [True]  
131 Text feature [relatively open] present in test data point [True]  
132 Text feature [address mechanism] present in test data point [True]  
133 Text feature [proteins 13] present in test data point [True]  
136 Text feature [addition activated] present in test data point [True]  
149 Text feature [transforming potential] present in test data point [True]  
158 Text feature [increased cells] present in test data point [True]  
174 Text feature [role mutation] present in test data point [True]  
177 Text feature [especially compared] present in test data point [True]  
184 Text feature [substitution glutamine] present in test data point [True]  
190 Text feature [activation survival] present in test data point [True]  
199 Text feature [mutant respectively] present in test data point [True]  
203 Text feature [nude] present in test data point [True]  
218 Text feature [downstream signaling] present in test data point [True]  
227 Text feature [cascade activation] present in test data point [True]  
230 Text feature [14 nsclc] present in test data point [True]  
233 Text feature [kit resulting] present in test data point [True]  
240 Text feature [15 confirmed] present in test data point [True]  
243 Text feature [canonical ras] present in test data point [True]  
255 Text feature [nm low] present in test data point [True]  
256 Text feature [carcinoma colorectal] present in test data point [True]  
257 Text feature [two kras] present in test data point [True]  
258 Text feature [product indicated] present in test data point [True]  
262 Text feature [result phosphorylation] present in test data point [True]  
280 Text feature [type respectively] present in test data point [True]  
282 Text feature [nude mice] present in test data point [True]  
284 Text feature [performed phosphorylation] present in test data point [True]  
291 Text feature [clearly increased] present in test data point [True]  
293 Text feature [pathway proliferation] present in test data point [True]  
298 Text feature [signaling previous] present in test data point [True]  
317 Text feature [transformation 21] present in test data point [True]

```

319 Text feature [signaling result] present in test data point [True]
322 Text feature [mutant lines] present in test data point [True]
333 Text feature [pathway judged] present in test data point [True]
336 Text feature [induced proliferation] present in test data point [True]
341 Text feature [hras mutant] present in test data point [True]
344 Text feature [rate decreased] present in test data point [True]
349 Text feature [fusion generated] present in test data point [True]
356 Text feature [formation kinase] present in test data point [True]
362 Text feature [increased high] present in test data point [True]
363 Text feature [16 next] present in test data point [True]
365 Text feature [survival 26] present in test data point [True]
375 Text feature [growth transformed] present in test data point [True]
393 Text feature [japan dna] present in test data point [True]
395 Text feature [activation transforming] present in test data point [True]
421 Text feature [mutant hras] present in test data point [True]
427 Text feature [mutations extracellular] present in test data point [True]
454 Text feature [formation examined] present in test data point [True]
461 Text feature [subcutaneously] present in test data point [True]
464 Text feature [carcinoma breast] present in test data point [True]
469 Text feature [signal activation] present in test data point [True]
470 Text feature [independent high] present in test data point [True]
472 Text feature [independent growth] present in test data point [True]
473 Text feature [activation associated] present in test data point [True]
486 Text feature [kit lane] present in test data point [True]
Out of the top 500 features 68 are present in query point

```

```

In [134]: # from tabulate import tabulate
          clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
          clf.fit(train_x_onehotCoding12, train_y)
          test_point_index = 1
          no_feature = 500
          predicted_cls = sig_clf.predict(test_x_onehotCoding12[test_point_index])
          print("Predicted Class : ", predicted_cls[0])
          print("Predicted Class Probabilities: ", np.round(sig_clf.predict_proba(test_x_onehotCoding12)[test_point_index]))
          print("Actual Class : ", test_y[test_point_index])
          indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
          print("-"*50)
          get_imfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index], test_df['Genre'].iloc[test_point_index])

```

```

Predicted Class : 7
Predicted Class Probabilities: [[0.127  0.1841  0.0235  0.12    0.0635  0.0521  0.4159  0.0066  0.0072]
Actual Class : 2
-----
```

```

37 Text feature [diagnosis poor] present in test data point [True]
44 Text feature [migration increased] present in test data point [True]
47 Text feature [case new] present in test data point [True]
52 Text feature [ligand figure] present in test data point [True]
```

63 Text feature [together may] present in test data point [True]  
65 Text feature [vs 55] present in test data point [True]  
67 Text feature [activity plays] present in test data point [True]  
71 Text feature [mutation find] present in test data point [True]  
77 Text feature [lines among] present in test data point [True]  
79 Text feature [proliferation overexpression] present in test data point [True]  
83 Text feature [supporting oncogenic] present in test data point [True]  
102 Text feature [carcinoma endometrial] present in test data point [True]  
108 Text feature [lead development] present in test data point [True]  
112 Text feature [signaling 15] present in test data point [True]  
122 Text feature [pdgfra mutated] present in test data point [True]  
123 Text feature [may led] present in test data point [True]  
127 Text feature [form resulting] present in test data point [True]  
129 Text feature [strong egfr] present in test data point [True]  
133 Text feature [proteins 13] present in test data point [True]  
139 Text feature [mutant receptors] present in test data point [True]  
140 Text feature [absence ligand] present in test data point [True]  
141 Text feature [activation receptor] present in test data point [True]  
153 Text feature [remains fully] present in test data point [True]  
157 Text feature [receptor activation] present in test data point [True]  
158 Text feature [increased cells] present in test data point [True]  
165 Text feature [mutation refractory] present in test data point [True]  
166 Text feature [receptors revealed] present in test data point [True]  
169 Text feature [options immunohistochemistry] present in test data point [True]  
174 Text feature [role mutation] present in test data point [True]  
175 Text feature [alterations increasing] present in test data point [True]  
196 Text feature [41 49] present in test data point [True]  
202 Text feature [found lymph] present in test data point [True]  
205 Text feature [pathway importantly] present in test data point [True]  
208 Text feature [addition overexpression] present in test data point [True]  
212 Text feature [grade primary] present in test data point [True]  
215 Text feature [18 52] present in test data point [True]  
218 Text feature [downstream signaling] present in test data point [True]  
235 Text feature [direct target] present in test data point [True]  
253 Text feature [activity 104] present in test data point [True]  
254 Text feature [evaluation revealed] present in test data point [True]  
257 Text feature [two kras] present in test data point [True]  
259 Text feature [egfr immunohistochemistry] present in test data point [True]  
275 Text feature [receptor distinct] present in test data point [True]  
284 Text feature [performed phosphorylation] present in test data point [True]  
290 Text feature [dimerization activation] present in test data point [True]  
292 Text feature [within class] present in test data point [True]  
293 Text feature [pathway proliferation] present in test data point [True]  
296 Text feature [case 51] present in test data point [True]  
300 Text feature [arm treated] present in test data point [True]  
303 Text feature [receptor likely] present in test data point [True]  
305 Text feature [78 71] present in test data point [True]  
313 Text feature [particular braf] present in test data point [True]

```

315 Text feature [combination approaches] present in test data point [True]
316 Text feature [response 16] present in test data point [True]
317 Text feature [transformation 21] present in test data point [True]
322 Text feature [mutant lines] present in test data point [True]
332 Text feature [similar mutations] present in test data point [True]
338 Text feature [mutations decrease] present in test data point [True]
342 Text feature [exon right] present in test data point [True]
351 Text feature [spontaneous receptor] present in test data point [True]
356 Text feature [formation kinase] present in test data point [True]
362 Text feature [increased high] present in test data point [True]
364 Text feature [figure oncogene] present in test data point [True]
369 Text feature [nodules revealed] present in test data point [True]
373 Text feature [activity 65] present in test data point [True]
374 Text feature [suggests single] present in test data point [True]
376 Text feature [17 67] present in test data point [True]
382 Text feature [also oncogenic] present in test data point [True]
385 Text feature [analysis pretreatment] present in test data point [True]
386 Text feature [sequencing would] present in test data point [True]
398 Text feature [developed tumor] present in test data point [True]
414 Text feature [recently number] present in test data point [True]
424 Text feature [kinase reported] present in test data point [True]
428 Text feature [ligand] present in test data point [True]
434 Text feature [mutated receptors] present in test data point [True]
447 Text feature [models 16] present in test data point [True]
454 Text feature [formation examined] present in test data point [True]
460 Text feature [fgfr2 activation] present in test data point [True]
464 Text feature [carcinoma breast] present in test data point [True]
468 Text feature [59 mutations] present in test data point [True]
470 Text feature [independent high] present in test data point [True]
472 Text feature [independent growth] present in test data point [True]
473 Text feature [activation associated] present in test data point [True]
475 Text feature [table akt] present in test data point [True]
Out of the top 500 features 84 are present in query point

```

### 7.1.2. Tf-Idf

#### 7.1.2.1. With Class balancing

```

In [136]: 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', max_iter=1000)
    clf.fit(train_x_tfidfCoding12, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding12, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding12)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))

```

```

# to avoid rounding error while multiplying probabilites we use log-probability
print("Log Loss :",log_loss(cv_y, sig_clf_probs))

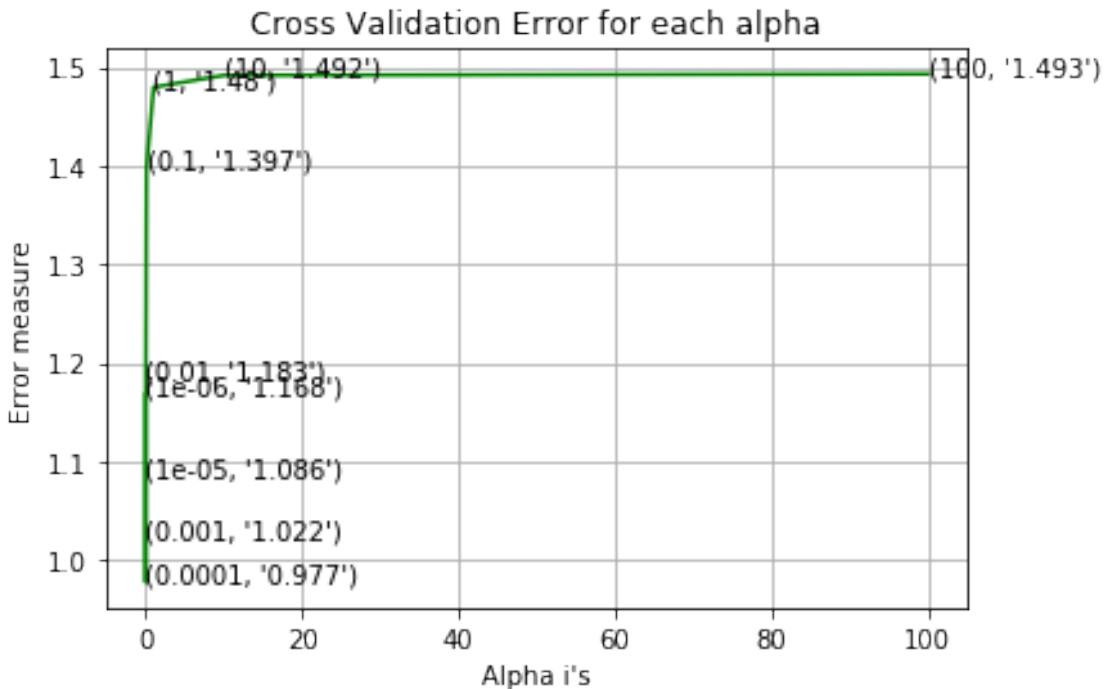
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")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

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', max_iter=1000, tol=1e-3)
clf.fit(train_x_tfidfCoding12, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding12, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(cv_y,predict_y))
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(cv_y,predict_y))
predict_y = sig_clf.predict_proba(test_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(test_y,predict_y))

for alpha = 1e-06
Log Loss : 1.167813994752195
for alpha = 1e-05
Log Loss : 1.0859260903127572
for alpha = 0.0001
Log Loss : 0.9770410081298855
for alpha = 0.001
Log Loss : 1.0219622612364356
for alpha = 0.01
Log Loss : 1.1830710021645425
for alpha = 0.1
Log Loss : 1.3971659458173822
for alpha = 1
Log Loss : 1.4795953617851796
for alpha = 10
Log Loss : 1.4916098352061153
for alpha = 100
Log Loss : 1.4930605454576433

```



For values of best alpha = 0.0001 The train log loss is: 0.41946078084539595  
 For values of best alpha = 0.0001 The cross validation log loss is: 0.9770410081298855  
 For values of best alpha = 0.0001 The test log loss is: 0.9602327285356053

```
In [137]: #alpha = [10 ** x for x in range(-6, 3)]
alpha = np.random.uniform(0.00005, 0.0005, 17)
alpha = np.round(alpha, 7)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log',
    clf.fit(train_x_tfidfCoding12, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding12, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding12)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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

```

        ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
clf.fit(train_x_tfidfCoding12, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding12, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_
predict_y = sig_clf.predict_proba(test_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_

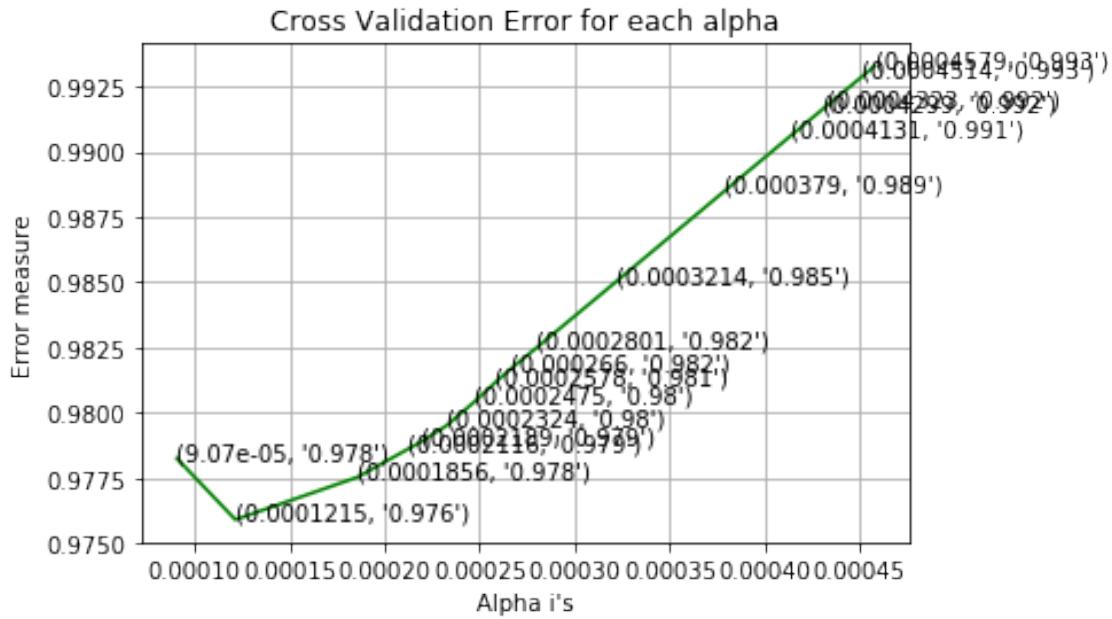
for alpha = 9.07e-05
Log Loss : 0.9782562489786956
for alpha = 0.0001215
Log Loss : 0.9759096404144472
for alpha = 0.0001856
Log Loss : 0.977531947894117
for alpha = 0.0002116
Log Loss : 0.9785838745959481
for alpha = 0.0002189
Log Loss : 0.9788752415251704
for alpha = 0.0002324
Log Loss : 0.979500692363654
for alpha = 0.0002475
Log Loss : 0.9804347332913858
for alpha = 0.0002578
Log Loss : 0.9811331228055793
for alpha = 0.000266
Log Loss : 0.9816585355369283
for alpha = 0.0002801
Log Loss : 0.9824999093245492
for alpha = 0.0003214
Log Loss : 0.9849963583770669
for alpha = 0.000379
Log Loss : 0.9885017175834626
for alpha = 0.0004131
Log Loss : 0.9905728331500386
for alpha = 0.0004299

```

```

Log Loss : 0.9915901761003181
for alpha = 0.0004323
Log Loss : 0.9917353133436668
for alpha = 0.0004514
Log Loss : 0.9928883720940982
for alpha = 0.0004579
Log Loss : 0.9932799102081188

```



```

For values of best alpha = 0.0001215 The train log loss is: 0.42742003270934587
For values of best alpha = 0.0001215 The cross validation log loss is: 0.9759096404144472
For values of best alpha = 0.0001215 The test log loss is: 0.9586037738595685

```

```

In [138]: clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
                           sig_clf,temp = predict_and_plot_confusion_matrix(train_x_tfidfCoding12, train_y, cv_x_tfidfCoding12, cv_y,
                           list_data = []
                           list_data.append('Tf_Idf+Uni-Bi_grams+SGD-LR+Class_Balancing')
                           list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_tfidfCoding12), eps=1e-15))
                           list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_tfidfCoding12), eps=1e-15))
                           list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_tfidfCoding12), eps=1e-15))
                           list_data.append('alpha = '+str(clf.alpha))
                           list_data.append(temp)
                           final_results.append(list_data)

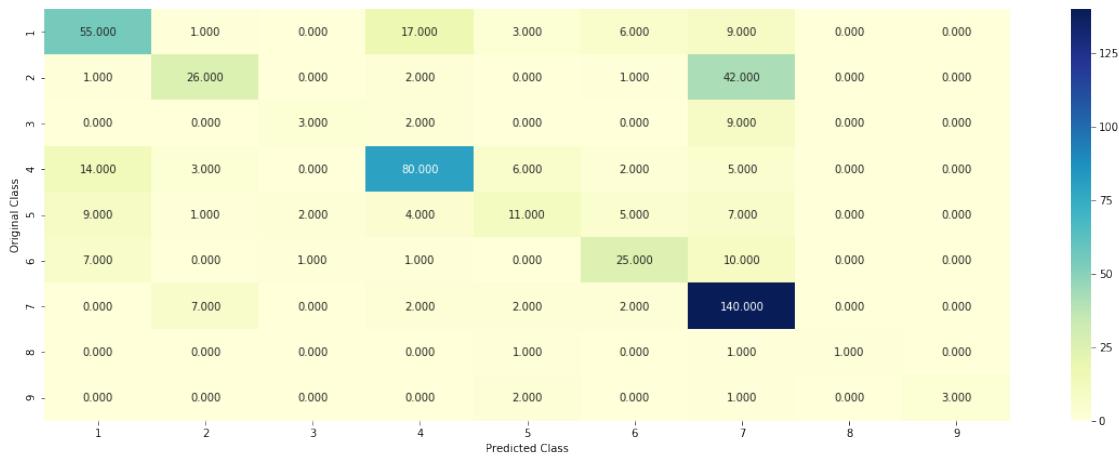
```

```

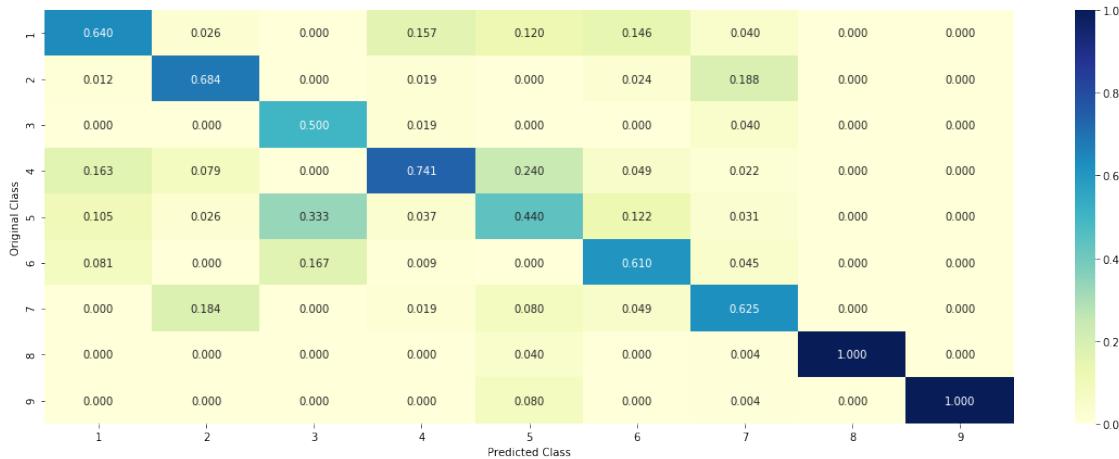
Log loss : 0.9759096404144472
Number of mis-classified points : 0.3533834586466165

```

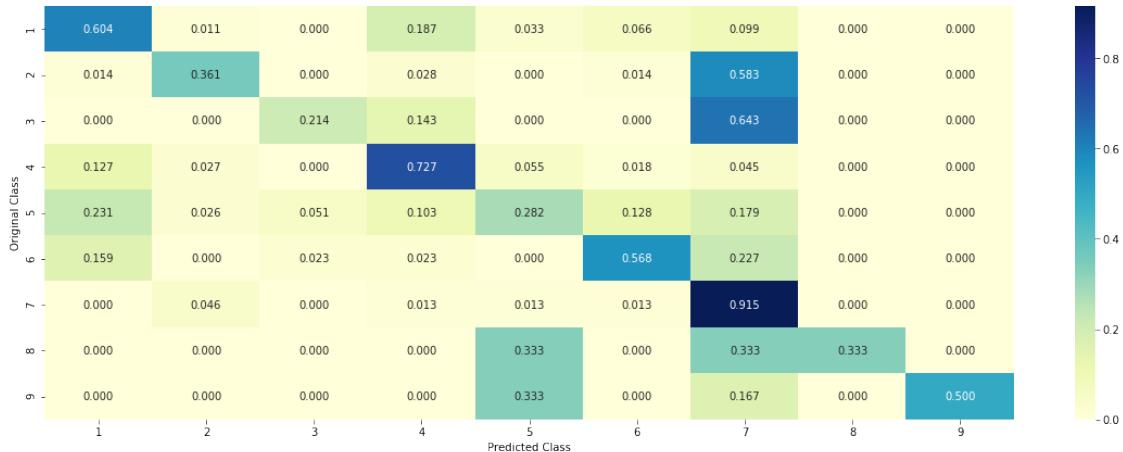
----- Confusion matrix -----



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



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



### 7.1.2.2. Without Class balancing

```
In [139]: alpha = [10 ** x for x in range(-6, 3)]
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_tfidfCoding12, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding12, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding12)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

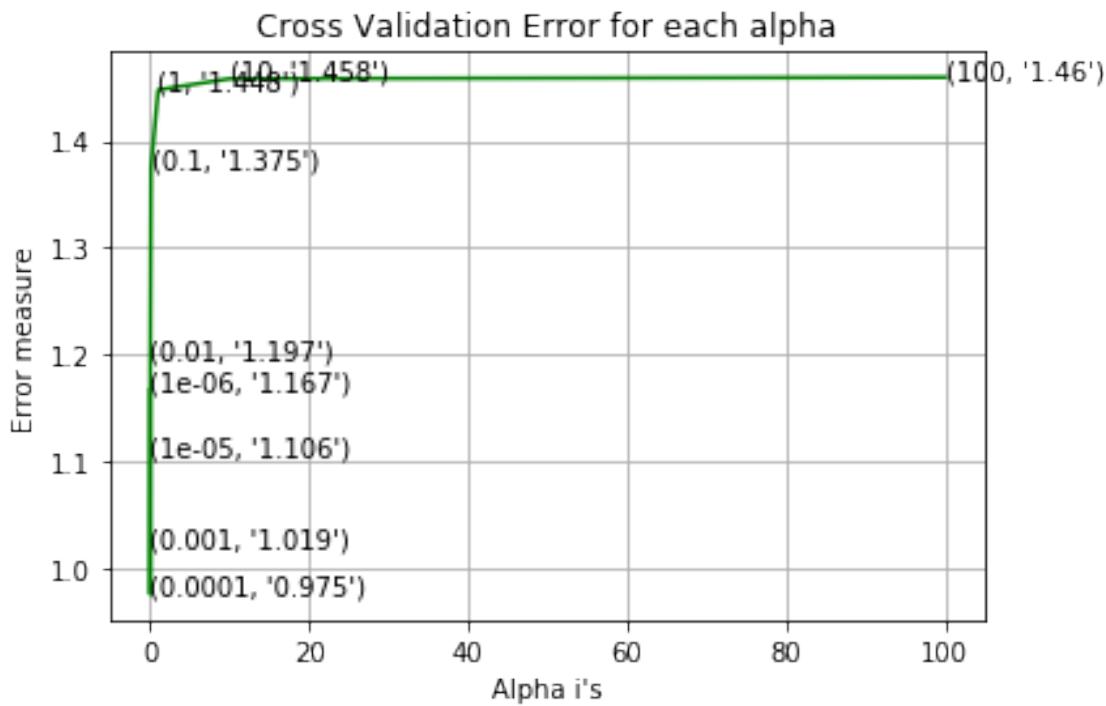
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")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

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_tfidfCoding12, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding12, train_y)
```

```

predict_y = sig_clf.predict_proba(train_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_
predict_y = sig_clf.predict_proba(test_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_
for alpha = 1e-06
Log Loss : 1.1669339573075512
for alpha = 1e-05
Log Loss : 1.1061686148227823
for alpha = 0.0001
Log Loss : 0.9753367488599498
for alpha = 0.001
Log Loss : 1.0190817915263672
for alpha = 0.01
Log Loss : 1.1971524734782997
for alpha = 0.1
Log Loss : 1.374913456151834
for alpha = 1
Log Loss : 1.4478846176199176
for alpha = 10
Log Loss : 1.4583499502886337
for alpha = 100
Log Loss : 1.4596348632167264

```



```
For values of best alpha = 0.0001 The train log loss is: 0.41037641510669026
For values of best alpha = 0.0001 The cross validation log loss is: 0.9753367488599498
For values of best alpha = 0.0001 The test log loss is: 0.9597865096875005
```

```
In [140]: #alpha = [10 ** x for x in range(-6, 3)]
alpha = np.random.uniform(0.00006, 0.0006, 17)
alpha = np.round(alpha, 7)
alpha.sort()
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_tfidfCoding12, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding12, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding12)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps_
# to avoid rounding error while multiplying probabilites we use log-probability
print("Log Loss :", log_loss(cv_y, sig_clf_probs))

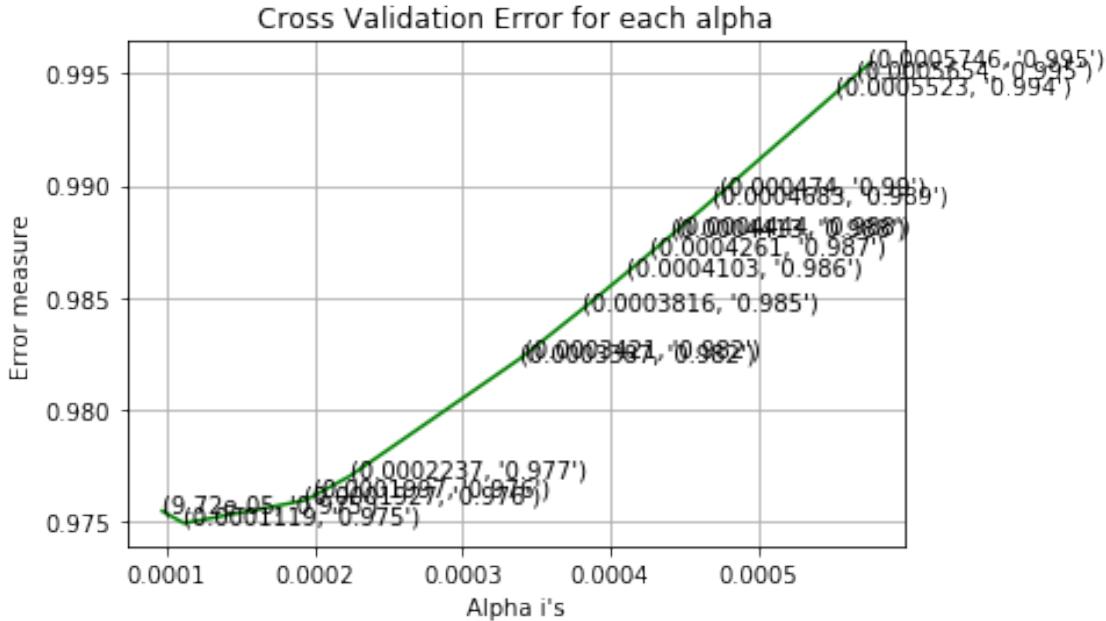
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], str(txt)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error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

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_tfidfCoding12, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding12, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_
predict_y = sig_clf.predict_proba(test_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_

for alpha = 9.72e-05
Log Loss : 0.9754786367014693
```

```
for alpha = 0.0001119
Log Loss : 0.974932412790238
for alpha = 0.0001927
Log Loss : 0.9759387691382561
for alpha = 0.0001997
Log Loss : 0.9761655860119923
for alpha = 0.0002237
Log Loss : 0.9770260721768155
for alpha = 0.0003387
Log Loss : 0.9822682429442415
for alpha = 0.0003421
Log Loss : 0.9824421861223568
for alpha = 0.0003816
Log Loss : 0.9845162927441337
for alpha = 0.0004103
Log Loss : 0.9860747387388313
for alpha = 0.0004261
Log Loss : 0.9869473607537149
for alpha = 0.0004413
Log Loss : 0.987795056334726
for alpha = 0.0004444
Log Loss : 0.9879688289128805
for alpha = 0.0004683
Log Loss : 0.9893172936722927
for alpha = 0.000474
Log Loss : 0.9896409255120722
for alpha = 0.0005523
Log Loss : 0.9941371286750882
for alpha = 0.0005654
Log Loss : 0.9948945843656894
for alpha = 0.0005746
Log Loss : 0.9954268225324446
```



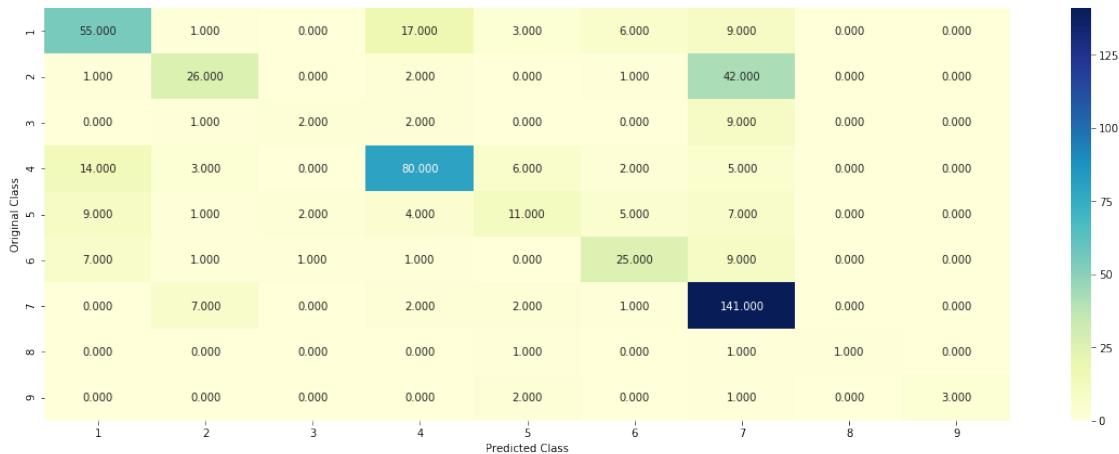
```

For values of best alpha = 0.0001119 The train log loss is: 0.4142853147094344
For values of best alpha = 0.0001119 The cross validation log loss is: 0.974932412790238
For values of best alpha = 0.0001119 The test log loss is: 0.9586194212148953

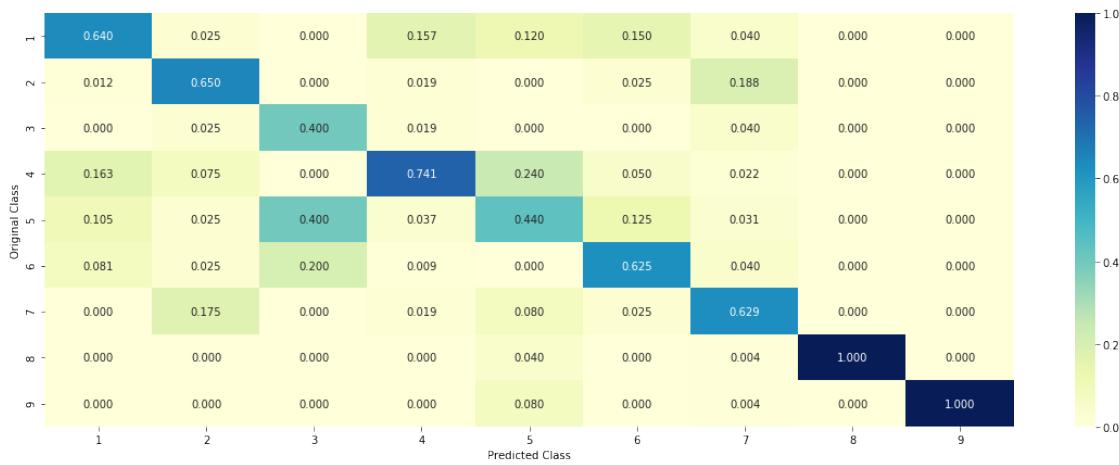
In [141]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
          sig_clf,temp = predict_and_plot_confusion_matrix(train_x_tfidfCoding12, train_y, cv_x_tfidfCoding12, cv_y, test_x_tfidfCoding12, test_y, list_data)
          list_data.append('Tf_Idf+Uni-Bi_grams+SGD-LR+ImClass_Balancing')
          list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_tfidfCoding12), eps=1e-15))
          list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_tfidfCoding12), eps=1e-15))
          list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_tfidfCoding12), eps=1e-15))
          list_data.append('alpha = '+str(clf.alpha))
          list_data.append(temp)
          final_results.append(list_data)

Log loss : 0.974932412790238
Number of mis-classified points : 0.3533834586466165
----- Confusion matrix -----

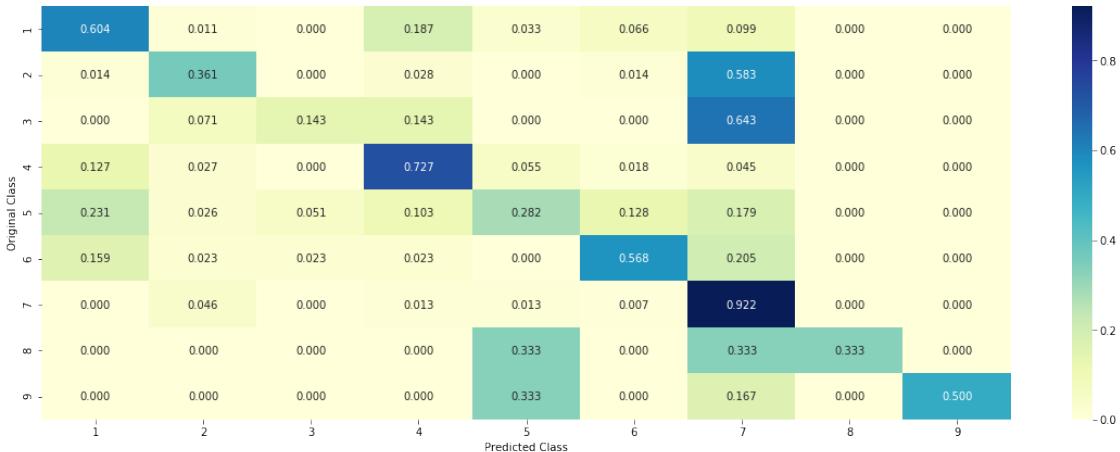
```



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



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



## SVM

```
In [143]: alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='hinge', random_state=42)
    clf.fit(train_x_tfidfCoding12, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding12, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding12)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps_
# to avoid rounding error while multiplying probabilites we use log-probability
print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

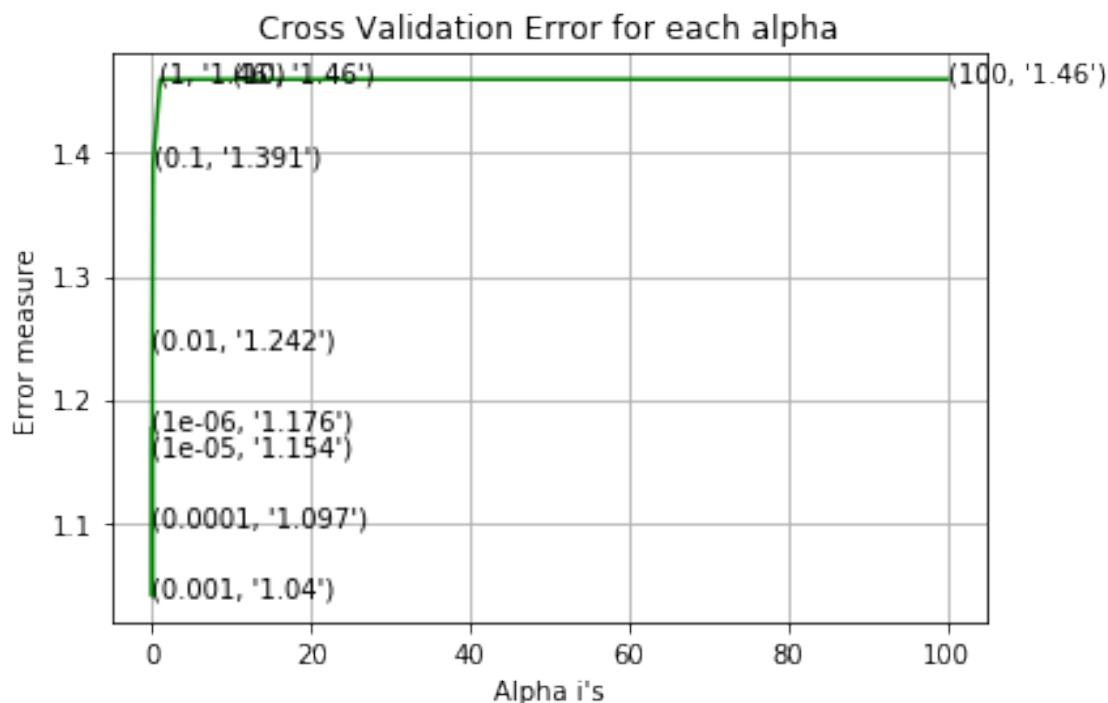
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_tfidfCoding12, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding12, train_y)
```

```

predict_y = sig_clf.predict_proba(train_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log los_
predict_y = sig_clf.predict_proba(test_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_]

for alpha = 1e-06
Log Loss : 1.1764676359139894
for alpha = 1e-05
Log Loss : 1.1544075133867213
for alpha = 0.0001
Log Loss : 1.0971634770135468
for alpha = 0.001
Log Loss : 1.040093599852211
for alpha = 0.01
Log Loss : 1.2423214510124556
for alpha = 0.1
Log Loss : 1.3908804252231661
for alpha = 1
Log Loss : 1.4600177085104444
for alpha = 10
Log Loss : 1.4600176862750838
for alpha = 100
Log Loss : 1.4600176274640293

```



```

For values of best alpha =  0.001 The train log loss is: 0.49590348460883915
For values of best alpha =  0.001 The cross validation log loss is: 1.040093599852211
For values of best alpha =  0.001 The test log loss is: 1.0188660330275956

```

## Logistic Regression:

```

In [67]: alpha = [10 ** x for x in range(-6, 3)]
#alpha = np.random.uniform(3,20,7)
#alpha = np.round(alpha,3)
#alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = LogisticRegression(C=i,solver='sag',class_weight='balanced',n_jobs=-1)
    #clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log',
    clf.fit(train_x_tfidfCoding12, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding12, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding12)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability encoding
    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 = LogisticRegression(C=i,solver='sag',class_weight='balanced',n_jobs=-1)
#clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
clf.fit(train_x_tfidfCoding12, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding12, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(predict_y,train_y))
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding12)

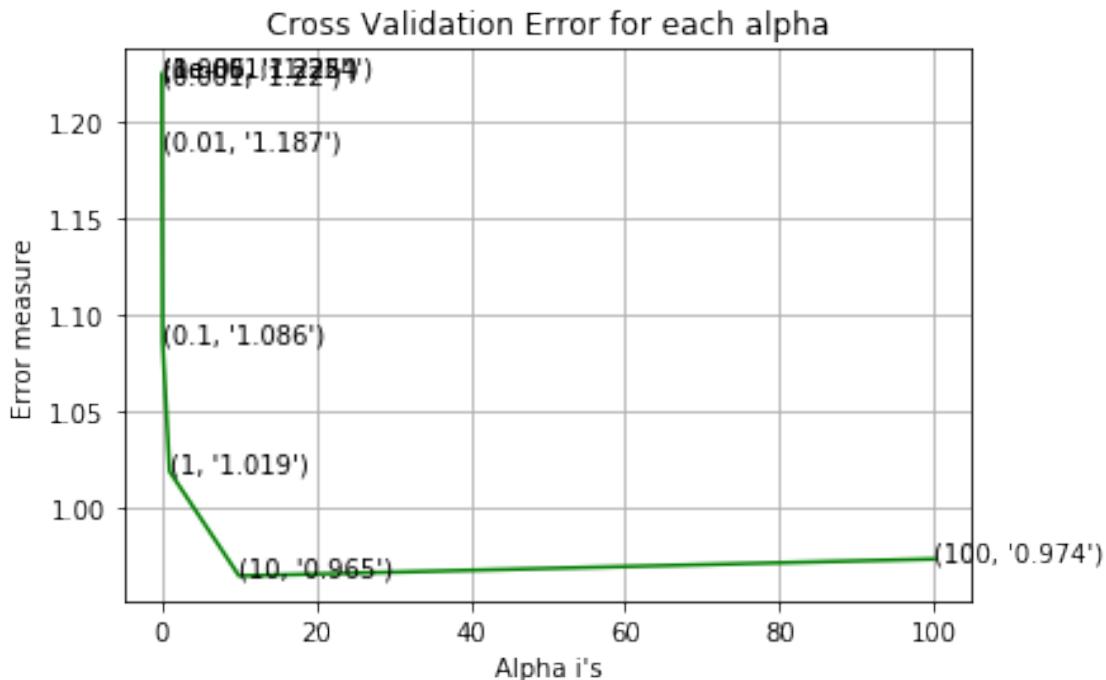
```

```

print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is: ", cv_log_loss)
predict_y = sig_clf.predict_proba(test_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss)

for alpha = 1e-06
Log Loss : 1.2251603653211331
for alpha = 1e-05
Log Loss : 1.2248147970034824
for alpha = 0.0001
Log Loss : 1.224326042747839
for alpha = 0.001
Log Loss : 1.220080593387868
for alpha = 0.01
Log Loss : 1.1867628535253558
for alpha = 0.1
Log Loss : 1.08643307615089
for alpha = 1
Log Loss : 1.0191937769087949
for alpha = 10
Log Loss : 0.9649968648008004
for alpha = 100
Log Loss : 0.9737234734387823

```



For values of best alpha = 10 The train log loss is: 0.41624758346410834  
 For values of best alpha = 10 The cross validation log loss is: 0.9654868260584009

```
For values of best alpha = 10 The test log loss is: 0.9456785873515301
```

```
In [70]: alpha = [10 ** x for x in range(-6, 3)]
alpha = np.random.uniform(3,20,7)
alpha = np.round(alpha,3)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = LogisticRegression(C=i,solver='sag',class_weight='balanced',n_jobs=-1)
    #clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log',
    clf.fit(train_x_tfidfCoding12, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding12, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding12)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

best_alpha = np.argmin(cv_log_error_array)
clf = LogisticRegression(C=i,solver='sag',class_weight='balanced',n_jobs=-1)
#clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
clf.fit(train_x_tfidfCoding12, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding12, train_y)

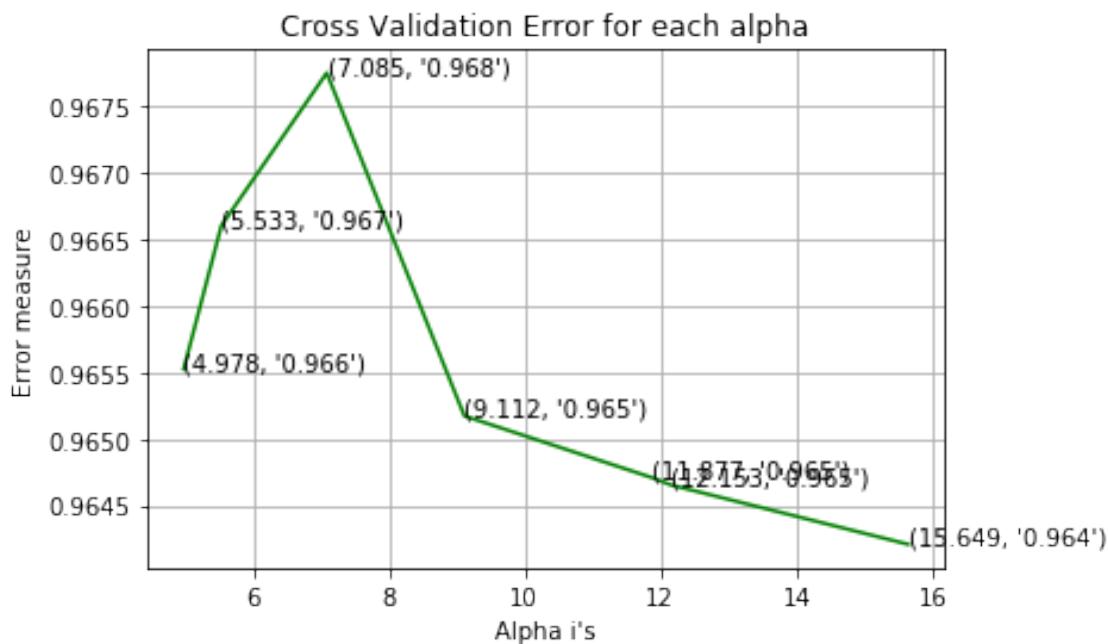
predict_y = sig_clf.predict_proba(train_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_
predict_y = sig_clf.predict_proba(test_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_

for alpha = 4.978
Log Loss : 0.9655235078661184
for alpha = 5.533
```

```

Log Loss : 0.9666005694878834
for alpha = 7.085
Log Loss : 0.9677481605904835
for alpha = 9.112
Log Loss : 0.9651718129376363
for alpha = 11.877
Log Loss : 0.9647082159285334
for alpha = 12.153
Log Loss : 0.9646537023461265
for alpha = 15.649
Log Loss : 0.9642075695543909

```



```

For values of best alpha = 15.649 The train log loss is: 0.4295225090195933
For values of best alpha = 15.649 The cross validation log loss is: 0.9695123746748465
For values of best alpha = 15.649 The test log loss is: 0.947870304819179

```

```

In [71]: alpha = [10 ** x for x in range(-6, 3)]
alpha = np.random.uniform(12,50,5)
alpha = np.round(alpha,3)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = LogisticRegression(C=i,solver='sag',class_weight='balanced',n_jobs=-1)

```

```

#clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log',
clf.fit(train_x_tfidfCoding12, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding12, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding12)
cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-7))
# to avoid rounding error while multiplying probabilites we use log-probability encoding
print("Log Loss :",log_loss(cv_y, sig_clf_probs))

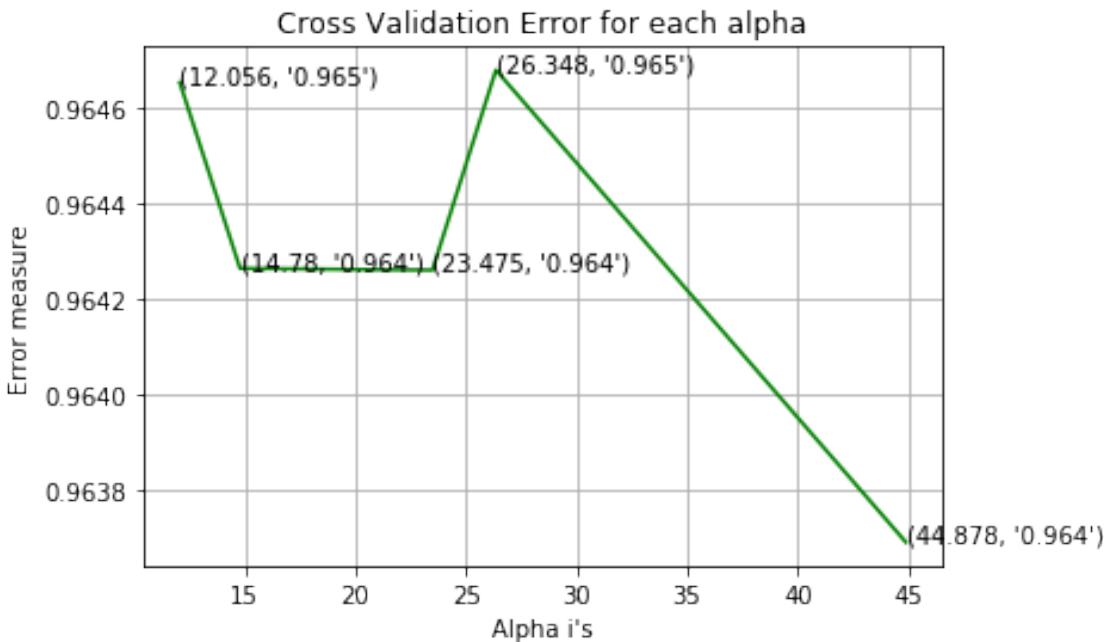
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")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

best_alpha = np.argmin(cv_log_error_array)
clf = LogisticRegression(C=i,solver='sag',class_weight='balanced',n_jobs=-1)
#clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
clf.fit(train_x_tfidfCoding12, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding12, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(cv_y, predict_y))
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(cv_y, predict_y))
predict_y = sig_clf.predict_proba(test_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(test_y, predict_y))

for alpha = 12.056
Log Loss : 0.9646540350482602
for alpha = 14.78
Log Loss : 0.9642652161495139
for alpha = 23.475
Log Loss : 0.9642617233144963
for alpha = 26.348
Log Loss : 0.9646801112294364
for alpha = 44.878
Log Loss : 0.963692732633654

```



For values of best alpha = 44.878 The train log loss is: 0.4792489641706121  
 For values of best alpha = 44.878 The cross validation log loss is: 0.9922615294099696  
 For values of best alpha = 44.878 The test log loss is: 0.9530989551594877

```
In [76]: alpha = [10 ** x for x in range(-6, 3)]
alpha = np.random.uniform(45,100,6)
alpha = np.round(alpha,3)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = LogisticRegression(C=i,solver='sag',class_weight='balanced',n_jobs=-1)
    #clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log',
    clf.fit(train_x_tfidfCoding12, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding12, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding12)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability encoding
    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]))
```

```

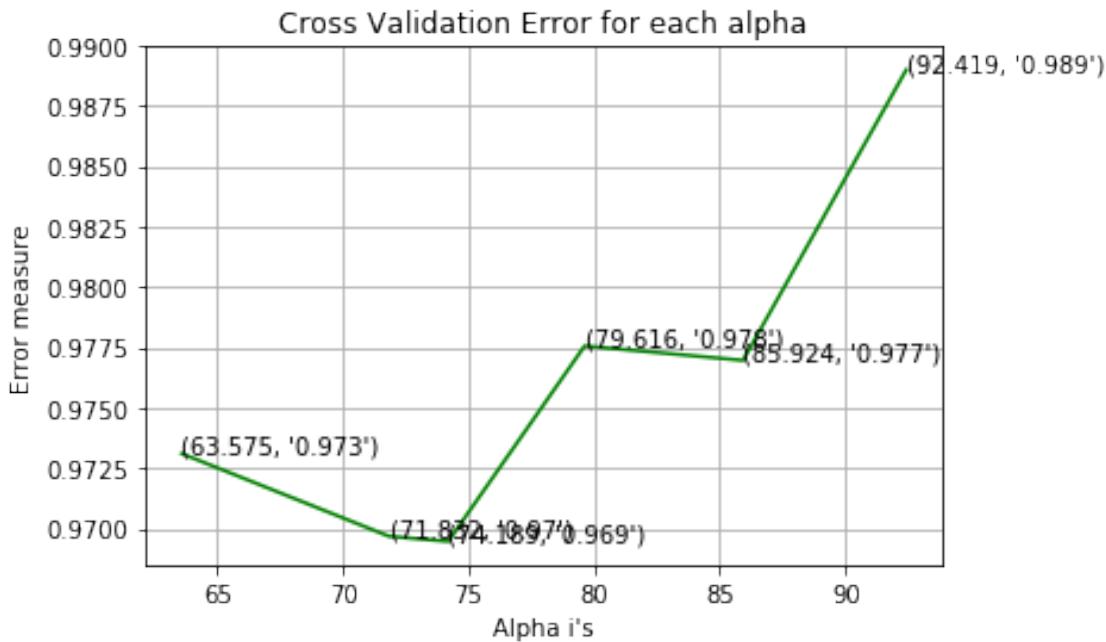
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

best_alpha = np.argmin(cv_log_error_array)
clf = LogisticRegression(C=i,solver='sag',class_weight='balanced',n_jobs=-1)
#clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
clf.fit(train_x_tfidfCoding12, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding12, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss"
predict_y = sig_clf.predict_proba(test_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_1

for alpha = 63.575
Log Loss : 0.9731066797806623
for alpha = 71.832
Log Loss : 0.9696839978678216
for alpha = 74.189
Log Loss : 0.9694771368063494
for alpha = 79.616
Log Loss : 0.9775578840714221
for alpha = 85.924
Log Loss : 0.9769634740222813
for alpha = 92.419
Log Loss : 0.988977620987819

```



For values of best alpha = 74.189 The train log loss is: 0.4580033599640141

For values of best alpha = 74.189 The cross validation log loss is: 0.9854325019445621

For values of best alpha = 74.189 The test log loss is: 0.9512956228142643

In [79]: *##testing*

```

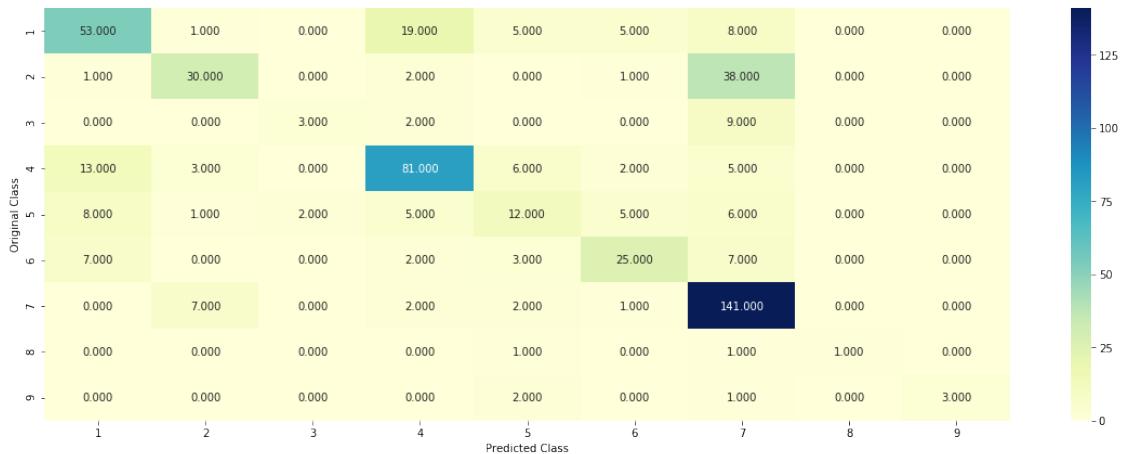
clf = LogisticRegression(C=44.878, penalty='l2', class_weight='balanced')
sig_clf,temp = predict_and_plot_confusion_matrix(train_x_tfidfCoding12, train_y, cv_x,
list_data = []
list_data.append('Tf_Idf+Uni-Bi_grams+SAG-LR+Class_Balancing')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_tfidfCoding12), eps=1e-15))
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_tfidfCoding12), eps=1e-15))
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_tfidfCoding12), eps=1e-15))
list_data.append('C = '+str(clf.C))
list_data.append(temp)
final_results.append(list_data)

```

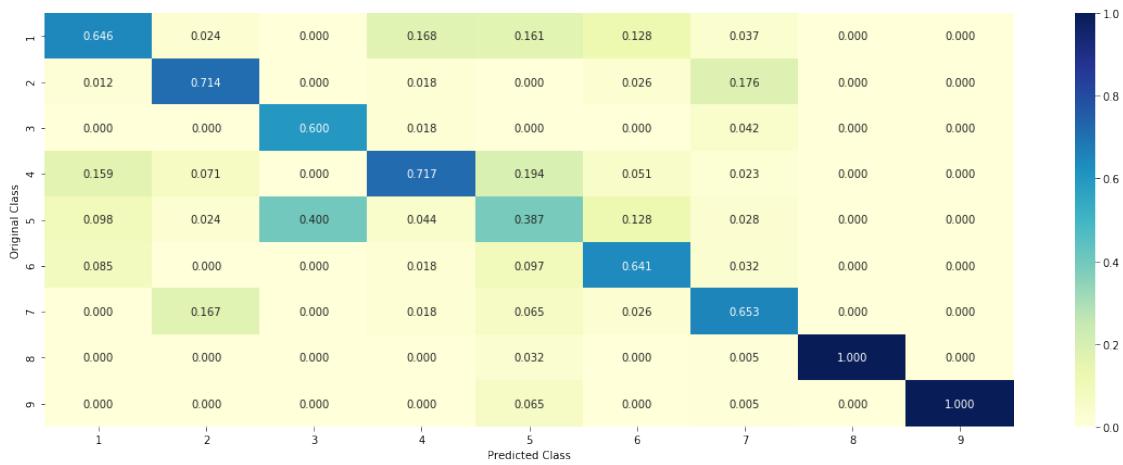
Log loss : 0.9661476609913215

Number of mis-classified points : 0.34398496240601506

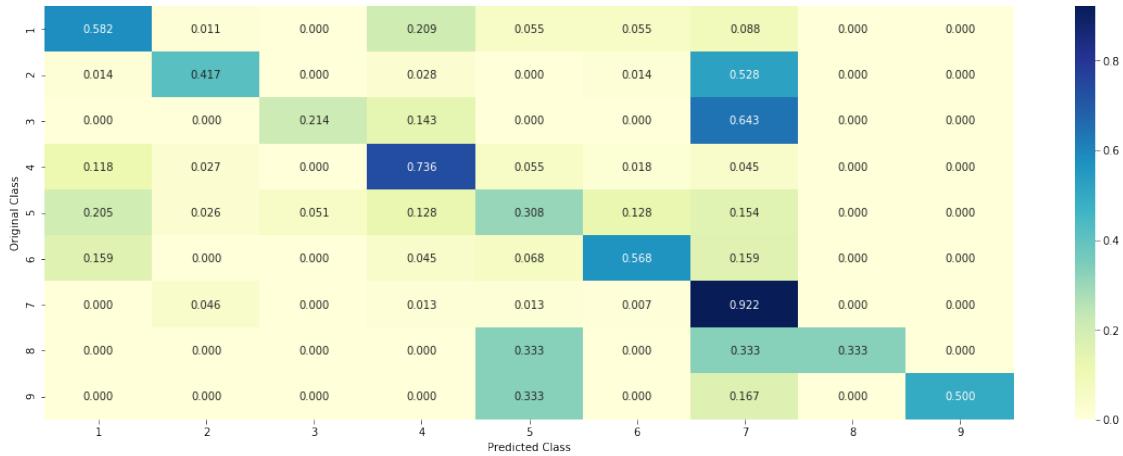
----- Confusion matrix -----



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



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



### 0.0.1 Liblinear + Dual formulation

```
In [88]: alpha = [10 ** x for x in range(-6, 3)]
#alpha = np.random.uniform(12,50,5)
#alpha = np.round(alpha,3)
#alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = LogisticRegression(C=i,solver='liblinear',class_weight='balanced',dual=True)
    #clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log',
    clf.fit(train_x_tfidfCoding12, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding12, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding12)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

best_alpha = np.argmin(cv_log_error_array)
```

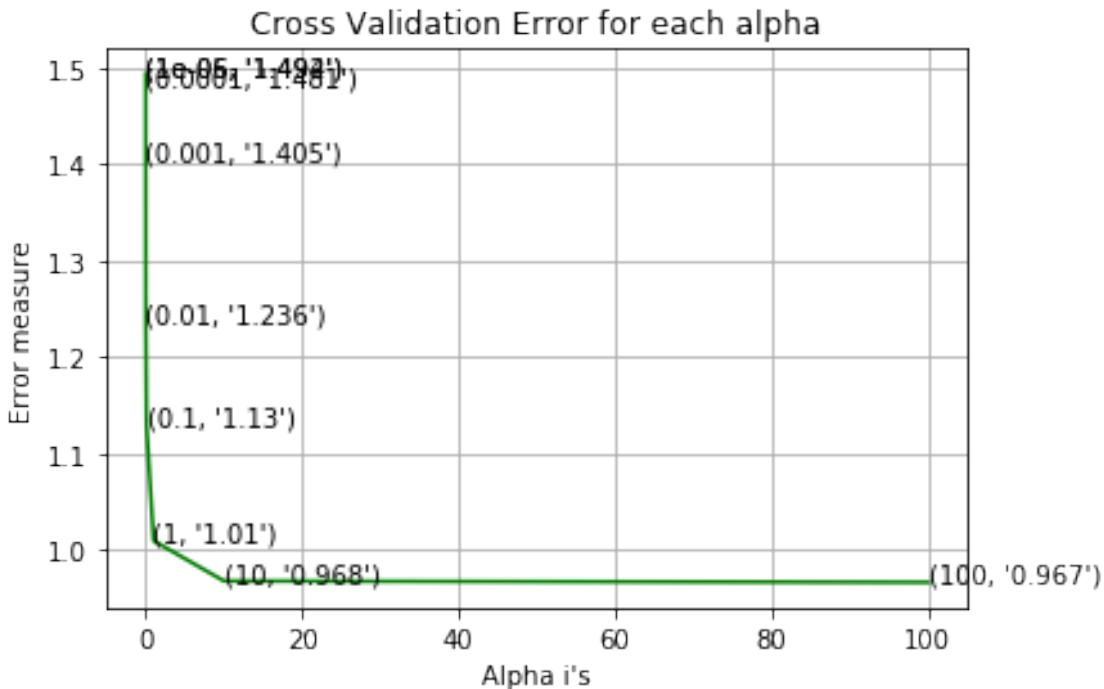
```

clf = LogisticRegression(C=i,solver='liblinear',class_weight='balanced',dual=True,n_jobs=-1)
#clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
clf.fit(train_x_tfidfCoding12, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding12, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss)
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss)
predict_y = sig_clf.predict_proba(test_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss)

for alpha = 1e-06
Log Loss : 1.4935640851620444
for alpha = 1e-05
Log Loss : 1.4921351458316154
for alpha = 0.0001
Log Loss : 1.4806039820890449
for alpha = 0.001
Log Loss : 1.4050881769898391
for alpha = 0.01
Log Loss : 1.2362999237832646
for alpha = 0.1
Log Loss : 1.1299238643542424
for alpha = 1
Log Loss : 1.009883298356866
for alpha = 10
Log Loss : 0.968157855282615
for alpha = 100
Log Loss : 0.9667551077756726

```



```
For values of best alpha = 100 The train log loss is: 0.4323053238714277
For values of best alpha = 100 The cross validation log loss is: 0.9692523109578295
For values of best alpha = 100 The test log loss is: 0.9488828459869497
```

```
In [89]: alpha = [10 ** x for x in range(-6, 3)]
alpha = np.random.uniform(80,200,15)
alpha = np.round(alpha,3)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = LogisticRegression(C=i,solver='liblinear',class_weight='balanced',dual=True)
    #clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log',
    clf.fit(train_x_tfidfCoding12, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding12, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding12)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
```

```

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

best_alpha = np.argmin(cv_log_error_array)
clf = LogisticRegression(C=i,solver='liblinear',class_weight='balanced',dual=True,n_j...
#clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
clf.fit(train_x_tfidfCoding12, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding12, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_...
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_...
predict_y = sig_clf.predict_proba(test_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_...

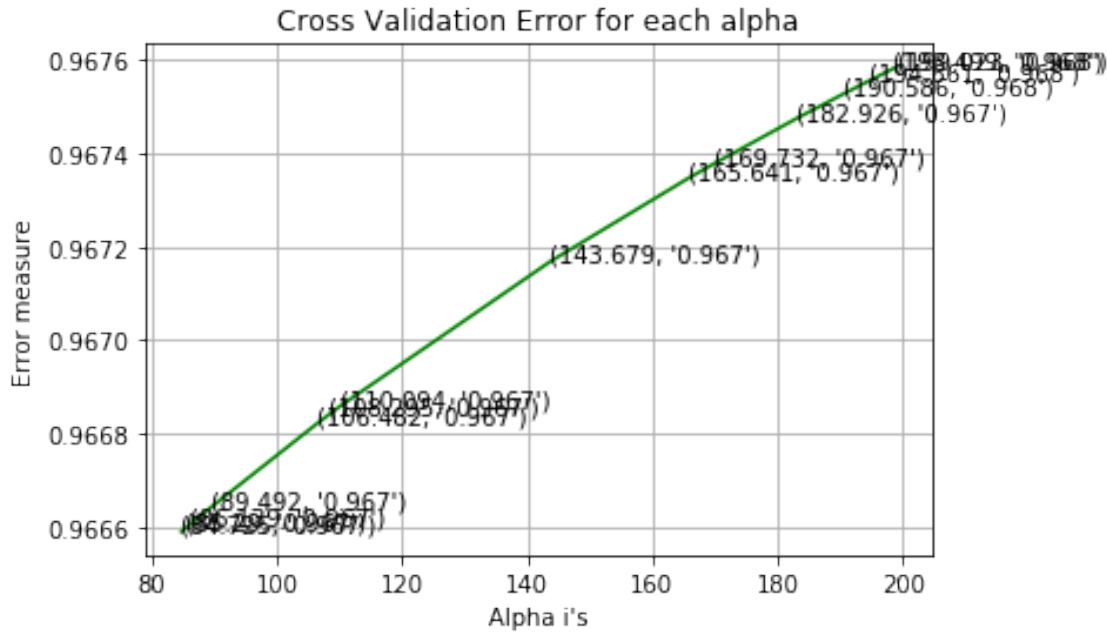
for alpha = 84.725
Log Loss : 0.9665895797975521
for alpha = 85.29
Log Loss : 0.9665959431277928
for alpha = 86.139
Log Loss : 0.9666053204019728
for alpha = 89.492
Log Loss : 0.9666423423352526
for alpha = 106.482
Log Loss : 0.9668220961726366
for alpha = 108.295
Log Loss : 0.9668404359361702
for alpha = 110.094
Log Loss : 0.9668585372770508
for alpha = 143.679
Log Loss : 0.9671682327769623
for alpha = 165.641
Log Loss : 0.9673457162378406
for alpha = 169.732
Log Loss : 0.9673768108510106
for alpha = 182.926
Log Loss : 0.9674736725279803
for alpha = 190.586
Log Loss : 0.9675273942478451
for alpha = 194.661

```

```

Log Loss : 0.9675552807225936
for alpha = 198.499
Log Loss : 0.9675812131247746
for alpha = 199.023
Log Loss : 0.9675847207361715

```



```

For values of best alpha = 84.725 The train log loss is: 0.4117964325110095
For values of best alpha = 84.725 The cross validation log loss is: 0.9670589699915485
For values of best alpha = 84.725 The test log loss is: 0.9477388267259347

```

```

In [91]: alpha = [10 ** x for x in range(-6, 3)]
alpha = np.random.uniform(10,50,6)
alpha = np.round(alpha,3)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = LogisticRegression(C=i,solver='liblinear',class_weight='balanced',dual=True)
    #clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log',
    clf.fit(train_x_tfidfCoding12, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding12, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding12)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))

```

```

# to avoid rounding error while multiplying probabilites we use log-probability etc.
print("Log Loss :",log_loss(cv_y, sig_clf_probs))

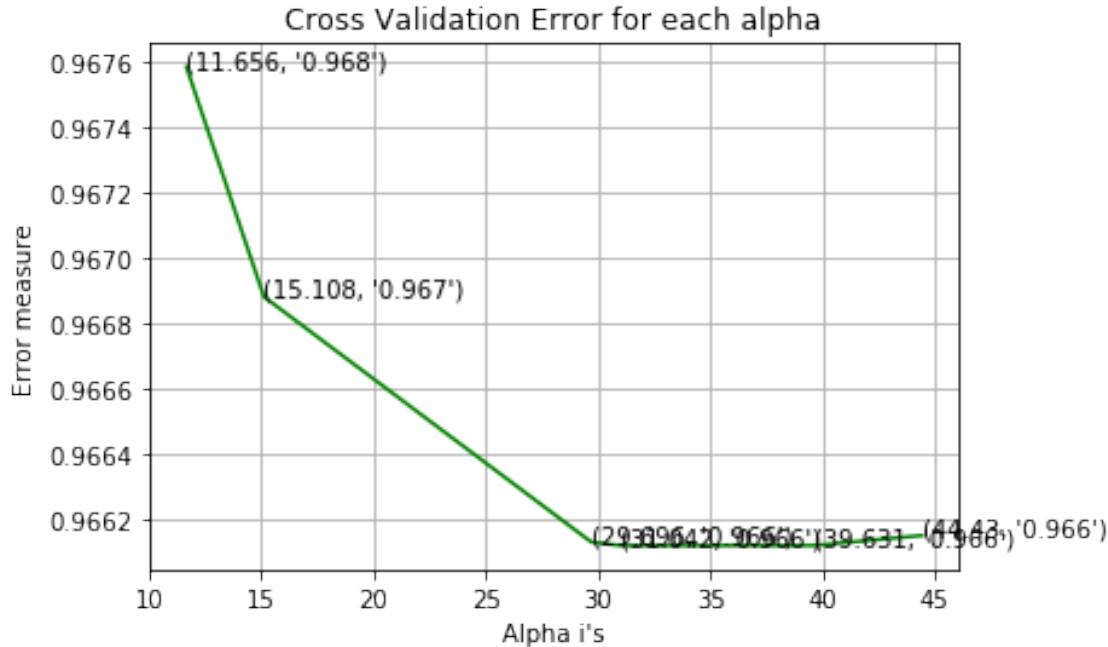
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")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

best_alpha = np.argmin(cv_log_error_array)
clf = LogisticRegression(C=i,solver='liblinear',class_weight='balanced',dual=True,n_jobs=1)
#clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
clf.fit(train_x_tfidfCoding12, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding12, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(cv_y, predict_y))
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(cv_y, predict_y))
predict_y = sig_clf.predict_proba(test_x_tfidfCoding12)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(test_y, predict_y))

for alpha = 11.656
Log Loss : 0.9675859070103057
for alpha = 15.108
Log Loss : 0.9668832526193721
for alpha = 29.696
Log Loss : 0.9661329418482317
for alpha = 31.042
Log Loss : 0.9661220628379997
for alpha = 39.631
Log Loss : 0.9661223633441779
for alpha = 44.43
Log Loss : 0.966152896486159

```



For values of best alpha = 31.042 The train log loss is: 0.45729119257114387

For values of best alpha = 31.042 The cross validation log loss is: 0.9729217323869387

For values of best alpha = 31.042 The test log loss is: 0.951868948143535

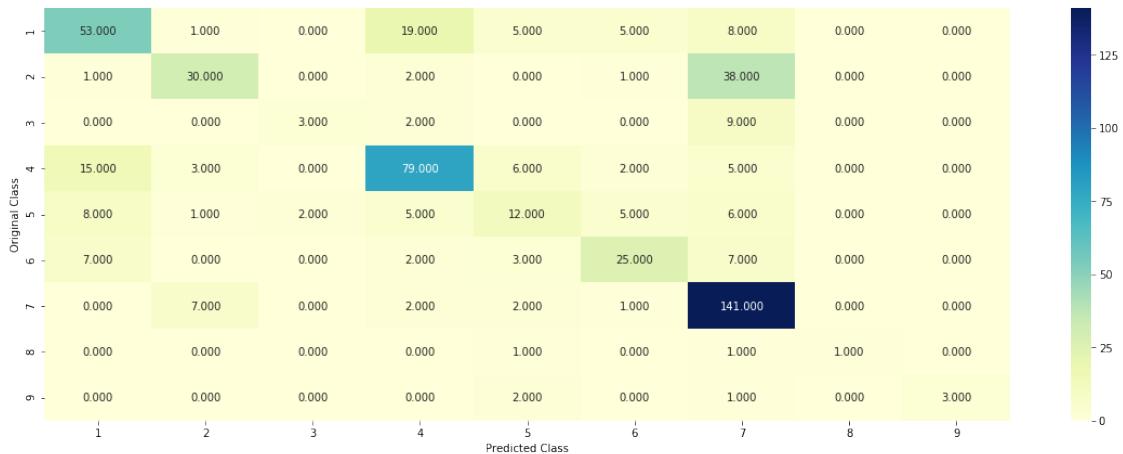
```
In [92]: ##testing
clf = LogisticRegression(C=31.042, penalty='l2', class_weight='balanced', dual=True)
sig_clf,temp = predict_and_plot_confusion_matrix(train_x_tfidfCoding12, train_y, cv_x)

list_data = []
list_data.append('Tf_Idf+Uni-Bi_grams+(Liblinear+dual-LR)+Class_Balancing')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_tfidfCoding12), eps=1e-15))
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_tfidfCoding12), eps=1e-15))
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_tfidfCoding12), eps=1e-15))
list_data.append('C = '+str(clf.C))
list_data.append(temp)
final_results.append(list_data)
```

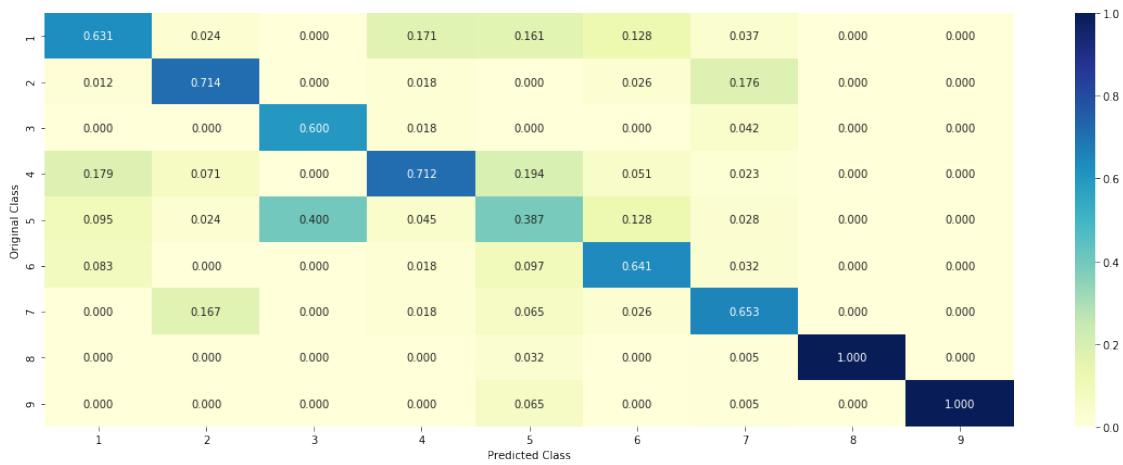
Log loss : 0.966122052036285

Number of mis-classified points : 0.34774436090225563

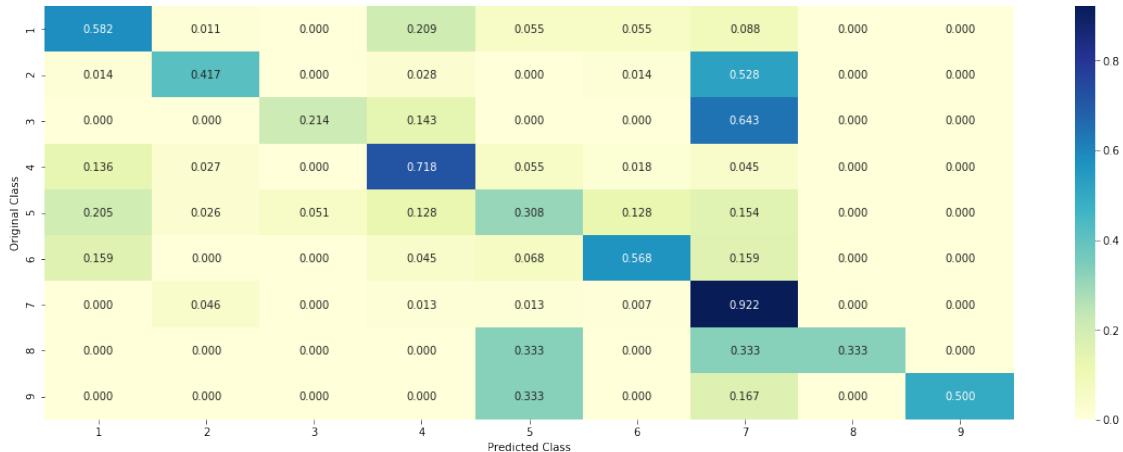
----- Confusion matrix -----



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



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



## 8. Some Feature transforms

### 8.1. With Lemmatizing the words in Text

```
In [66]: from nltk import word_tokenize
        from nltk.stem import WordNetLemmatizer
        class LemmaTokenizer(object):
            def __init__(self):
                self.wnl = WordNetLemmatizer()
            def __call__(self, doc):
                return [self.wnl.lemmatize(t.lower()) for t in word_tokenize(doc)]
```

```
In [67]: vec_lemma = TfidfVectorizer(tokenizer=LemmaTokenizer(),min_df=3)
train_text_feature_tfidfCoding_lemma = vec_lemma.fit_transform(train_df.TEXT)
test_text_feature_tfidfCoding_lemma = vec_lemma.transform(test_df.TEXT)
cv_text_feature_tfidfCoding_lemma = vec_lemma.transform(cv_df.TEXT)
```

```
In [68]: vec_lemma = TfidfVectorizer(tokenizer=LemmaTokenizer(),min_df=3,ngram_range=(1,2))
train_text_feature_tfidfCoding12_lemma = vec_lemma.fit_transform(train_df.TEXT)
test_text_feature_tfidfCoding12_lemma = vec_lemma.transform(test_df.TEXT)
cv_text_feature_tfidfCoding12_lemma = vec_lemma.transform(cv_df.TEXT)
```

```
In [69]: train_text_feature_tfidfCoding_lemma
```

```
Out[69]: <2124x51393 sparse matrix of type '<class 'numpy.float64'>'  
with 3172807 stored elements in Compressed Sparse Row format>
```

```
In [70]: train_text_feature_tfidfCoding12_lemma
```

```
Out[70]: <2124x746692 sparse matrix of type '<class 'numpy.float64'>'  
with 11666773 stored elements in Compressed Sparse Row format>
```

```

In [71]: vec_lemma = CountVectorizer(tokenizer=LemmaTokenizer(),min_df=3)
train_text_feature_onehotCoding_lemma = vec_lemma.fit_transform(train_df.TEXT)
test_text_feature_onehotCoding_lemma = vec_lemma.transform(test_df.TEXT)
cv_text_feature_onehotCoding_lemma = vec_lemma.transform(cv_df.TEXT)

In [72]: train_text_feature_onehotCoding_lemma = normalize(train_text_feature_onehotCoding_lemma)
test_text_feature_onehotCoding_lemma = normalize(test_text_feature_onehotCoding_lemma)
cv_text_feature_onehotCoding_lemma = normalize(cv_text_feature_onehotCoding_lemma, axis=1)

In [73]: train_gene_var_tfidfCoding = hstack((train_gene_feature_tfidfCoding,train_variation_feature_tfidfCoding))
test_gene_var_tfidfCoding = hstack((test_gene_feature_tfidfCoding,test_variation_feature_tfidfCoding))
cv_gene_var_tfidfCoding = hstack((cv_gene_feature_tfidfCoding, cv_variation_feature_tfidfCoding))

train_x_tfidfCoding_lemma = hstack((train_gene_var_tfidfCoding, train_text_feature_tfidfCoding_lemma))
train_y = np.array(list(train_df['Class']))

test_x_tfidfCoding_lemma = hstack((test_gene_var_tfidfCoding, test_text_feature_tfidfCoding_lemma))
test_y = np.array(list(test_df['Class']))

cv_x_tfidfCoding_lemma = hstack((cv_gene_var_tfidfCoding, cv_text_feature_tfidfCoding_lemma))
cv_y = np.array(list(cv_df['Class']))

In [74]: train_gene_var_tfidfCoding = hstack((train_gene_feature_tfidfCoding,train_variation_feature_tfidfCoding))
test_gene_var_tfidfCoding = hstack((test_gene_feature_tfidfCoding,test_variation_feature_tfidfCoding))
cv_gene_var_tfidfCoding = hstack((cv_gene_feature_tfidfCoding, cv_variation_feature_tfidfCoding))

train_x_tfidfCoding12_lemma = hstack((train_gene_var_tfidfCoding, train_text_feature_tfidfCoding_lemma))
train_y = np.array(list(train_df['Class']))

test_x_tfidfCoding12_lemma = hstack((test_gene_var_tfidfCoding, test_text_feature_tfidfCoding_lemma))
test_y = np.array(list(test_df['Class']))

cv_x_tfidfCoding12_lemma = hstack((cv_gene_var_tfidfCoding, cv_text_feature_tfidfCoding_lemma))
cv_y = np.array(list(cv_df['Class']))

In [75]: train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding, cv_variation_feature_onehotCoding))

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

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

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

```

### 8.1.1. Logistic Regression

## With TFIDF

```
In [72]: alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', r
    clf.fit(train_x_tfidfCoding_lemma, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding_lemma, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding_lemma)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-9))
    # 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', loss='log', r
clf.fit(train_x_tfidfCoding_lemma, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding_lemma, train_y)

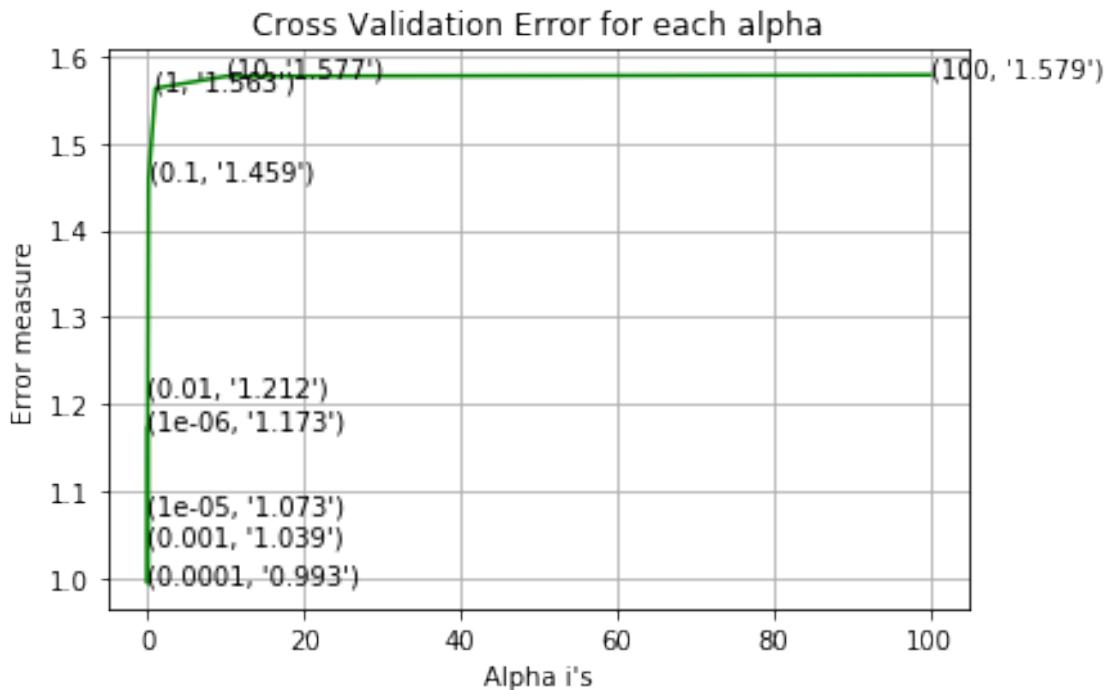
predict_y = sig_clf.predict_proba(train_x_tfidfCoding_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_
predict_y = sig_clf.predict_proba(test_x_tfidfCoding_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_

for alpha = 1e-06
Log Loss : 1.1725105211198363
for alpha = 1e-05
Log Loss : 1.0727790763164933
for alpha = 0.0001
Log Loss : 0.9933270023493858
for alpha = 0.001
Log Loss : 1.0393769868276823
for alpha = 0.01
Log Loss : 1.2115156373164764
```

```

for alpha = 0.1
Log Loss : 1.4587280775248854
for alpha = 1
Log Loss : 1.5634400391565477
for alpha = 10
Log Loss : 1.5771467640687522
for alpha = 100
Log Loss : 1.5787354931004236

```



```

For values of best alpha = 0.0001 The train log loss is: 0.4343344928180016
For values of best alpha = 0.0001 The cross validation log loss is: 0.9933270023493858
For values of best alpha = 0.0001 The test log loss is: 0.9696408040338836

```

```

In [74]: #alpha = [10 ** x for x in range(-6, 3)]
alpha = np.random.uniform(0.00005, 0.0005, 15)
alpha = np.round(alpha, 7)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', r
    clf.fit(train_x_tfidfCoding_lemma, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")

```

```

sig_clf.fit(train_x_tfidfCoding_lemma, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding_lemma)
cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-7))
# to avoid rounding error while multiplying probabilites we use log-probability encoding
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', loss='log')
clf.fit(train_x_tfidfCoding_lemma, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding_lemma, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(cv_y,predict_y))
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(cv_y,predict_y))
predict_y = sig_clf.predict_proba(test_x_tfidfCoding_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(test_y,predict_y))

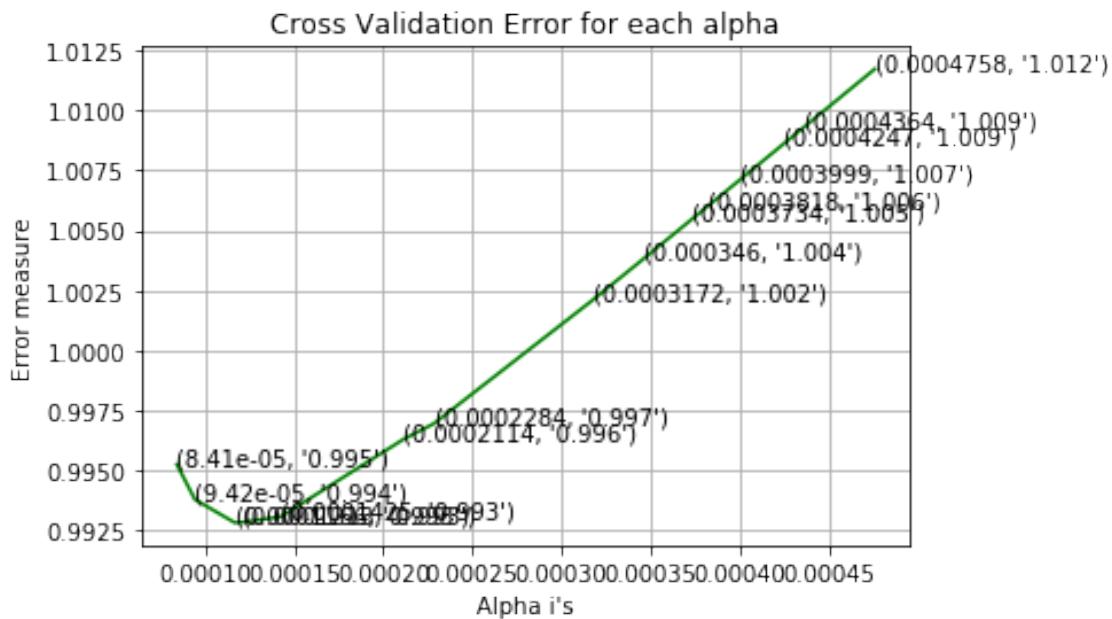
for alpha = 8.41e-05
Log Loss : 0.9952393681569353
for alpha = 9.42e-05
Log Loss : 0.993776938146513
for alpha = 0.0001166
Log Loss : 0.9928062427237282
for alpha = 0.0001213
Log Loss : 0.9928267387421457
for alpha = 0.0001425
Log Loss : 0.993054366054183
for alpha = 0.0002114
Log Loss : 0.9962897016086292
for alpha = 0.0002284
Log Loss : 0.9969391244513419
for alpha = 0.0003172
Log Loss : 1.0020762567080106
for alpha = 0.000346
Log Loss : 1.0038136727505582

```

```

for alpha = 0.0003734
Log Loss : 1.0054827077433133
for alpha = 0.0003818
Log Loss : 1.0059945541703275
for alpha = 0.0003999
Log Loss : 1.007098682712382
for alpha = 0.0004247
Log Loss : 1.0086144241065604
for alpha = 0.0004364
Log Loss : 1.009329970064607
for alpha = 0.0004758
Log Loss : 1.011734282052671

```



For values of best alpha = 0.0001166 The train log loss is: 0.44114113420696877  
 For values of best alpha = 0.0001166 The cross validation log loss is: 0.9928062427237282  
 For values of best alpha = 0.0001166 The test log loss is: 0.9685944047195433

```
In [75]: alpha = [10 ** x for x in range(-6, 3)]
#alpha = np.random.uniform(0.00005, 0.0005, 15)
#alpha = np.round(alpha, 7)
#alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = LogisticRegression(C=i, class_weight='balanced', dual=True, solver='liblinear')
```

```

#clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log',
clf.fit(train_x_tfidfCoding_lemma, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding_lemma, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding_lemma)
cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-7))
# to avoid rounding error while multiplying probabilites we use log-probability encoding
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 = LogisticRegression(C=i,class_weight='balanced',dual=True,solver='liblinear')
#clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
clf.fit(train_x_tfidfCoding_lemma, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding_lemma, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(cv_y, predict_y))
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(cv_y, predict_y))
predict_y = sig_clf.predict_proba(test_x_tfidfCoding_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(test_y, predict_y))

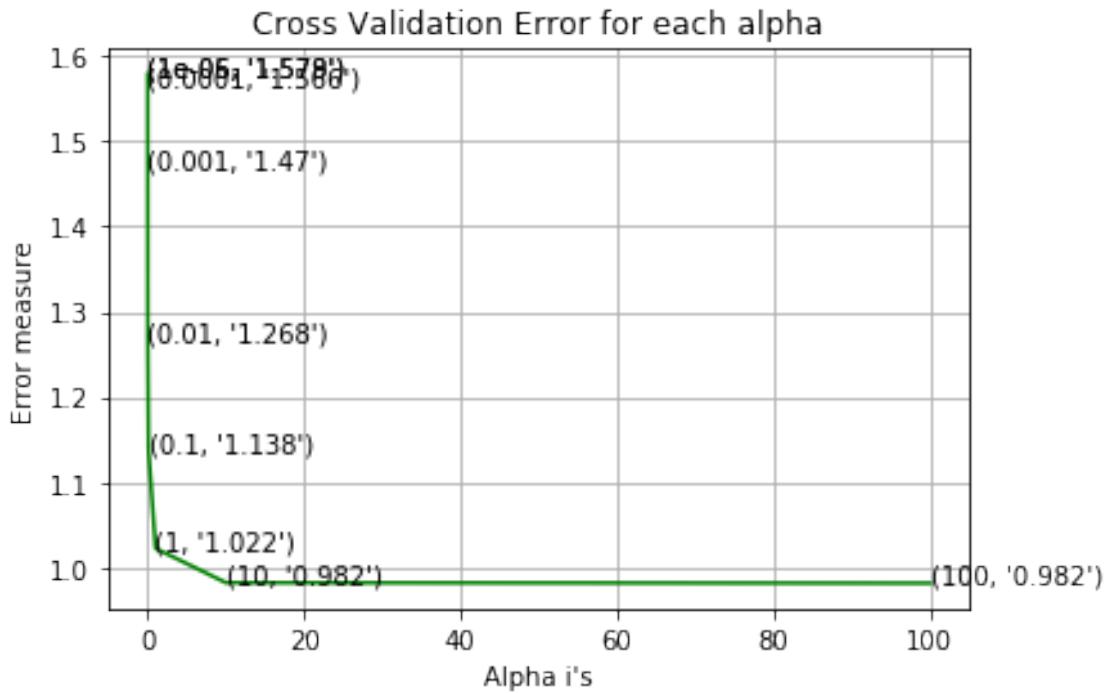
for alpha = 1e-06
Log Loss : 1.5790122647320304
for alpha = 1e-05
Log Loss : 1.5778514137704804
for alpha = 0.0001
Log Loss : 1.5663031067739186
for alpha = 0.001
Log Loss : 1.4699186449520276
for alpha = 0.01
Log Loss : 1.2675405376332065
for alpha = 0.1
Log Loss : 1.137803085272022
for alpha = 1
Log Loss : 1.022355305752015

```

```

for alpha = 10
Log Loss : 0.9821385654654367
for alpha = 100
Log Loss : 0.981599624098709

```



```

For values of best alpha = 100 The train log loss is: 0.4430328158048224
For values of best alpha = 100 The cross validation log loss is: 0.9831708950185931
For values of best alpha = 100 The test log loss is: 0.9582775078803201

```

```

In [76]: alpha = [10 ** x for x in range(-6, 3)]
#alpha = np.random.uniform(0.00005, 0.0005, 15)
#alpha = np.round(alpha, 7)
#alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = LogisticRegression(C=i, class_weight='balanced', dual=False, solver='liblinear')
    #clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log',
    clf.fit(train_x_tfidfCoding_lemma, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding_lemma, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding_lemma)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))

```

```

# to avoid rounding error while multiplying probabilites we use log-probability etc.
print("Log Loss :",log_loss(cv_y, sig_clf_probs))

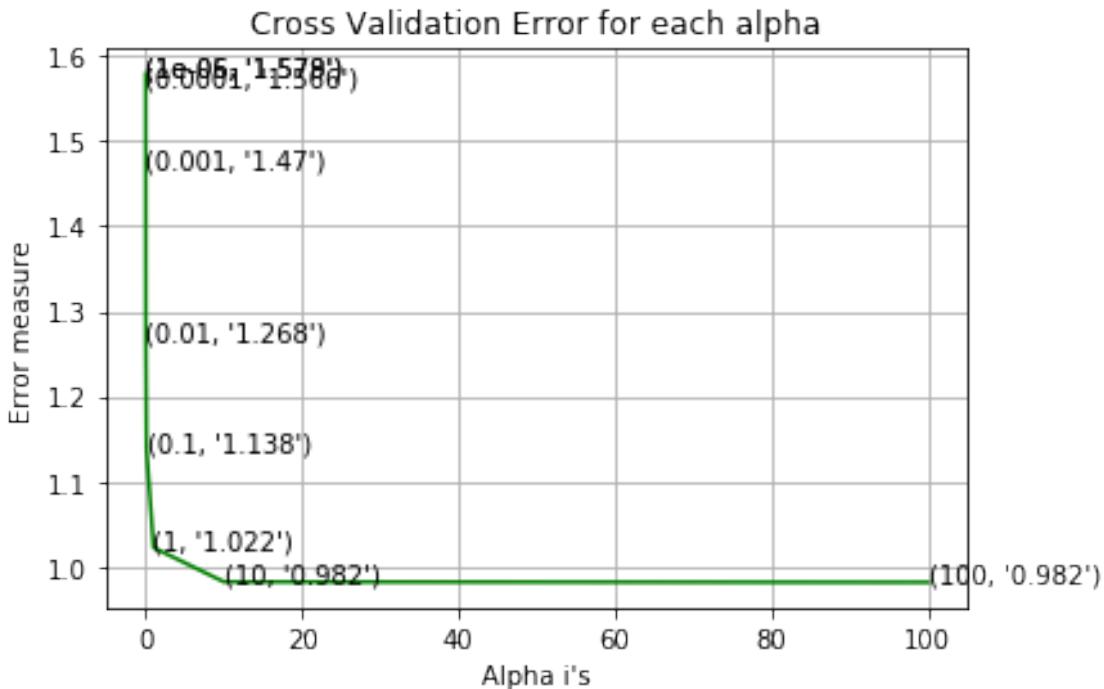
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")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

best_alpha = np.argmin(cv_log_error_array)
clf = LogisticRegression(C=i,class_weight='balanced',dual=False,solver='liblinear')
#clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
clf.fit(train_x_tfidfCoding_lemma, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding_lemma, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_
predict_y = sig_clf.predict_proba(test_x_tfidfCoding_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_

for alpha = 1e-06
Log Loss : 1.5790260487600183
for alpha = 1e-05
Log Loss : 1.577852328707823
for alpha = 0.0001
Log Loss : 1.566302671880441
for alpha = 0.001
Log Loss : 1.4699194864889424
for alpha = 0.01
Log Loss : 1.267541428193986
for alpha = 0.1
Log Loss : 1.13780751445501
for alpha = 1
Log Loss : 1.0223534085238728
for alpha = 10
Log Loss : 0.9821448963028034
for alpha = 100
Log Loss : 0.9815948690215434

```



For values of best alpha = 100 The train log loss is: 0.44302766665021975

For values of best alpha = 100 The cross validation log loss is: 0.9831701796113002

For values of best alpha = 100 The test log loss is: 0.9582747725630155

```
In [77]: alpha = [10 ** x for x in range(-6, 3)]
alpha = np.random.uniform(3,80,15)
alpha = np.round(alpha,3)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = LogisticRegression(C=i,class_weight='balanced',dual=True,solver='liblinear')
    #clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log',
    clf.fit(train_x_tfidfCoding_lemma, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding_lemma, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding_lemma)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
```

```

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

best_alpha = np.argmin(cv_log_error_array)
clf = LogisticRegression(C=i,class_weight='balanced',dual=True,solver='liblinear')
#clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
clf.fit(train_x_tfidfCoding_lemma, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding_lemma, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss"
predict_y = sig_clf.predict_proba(test_x_tfidfCoding_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_

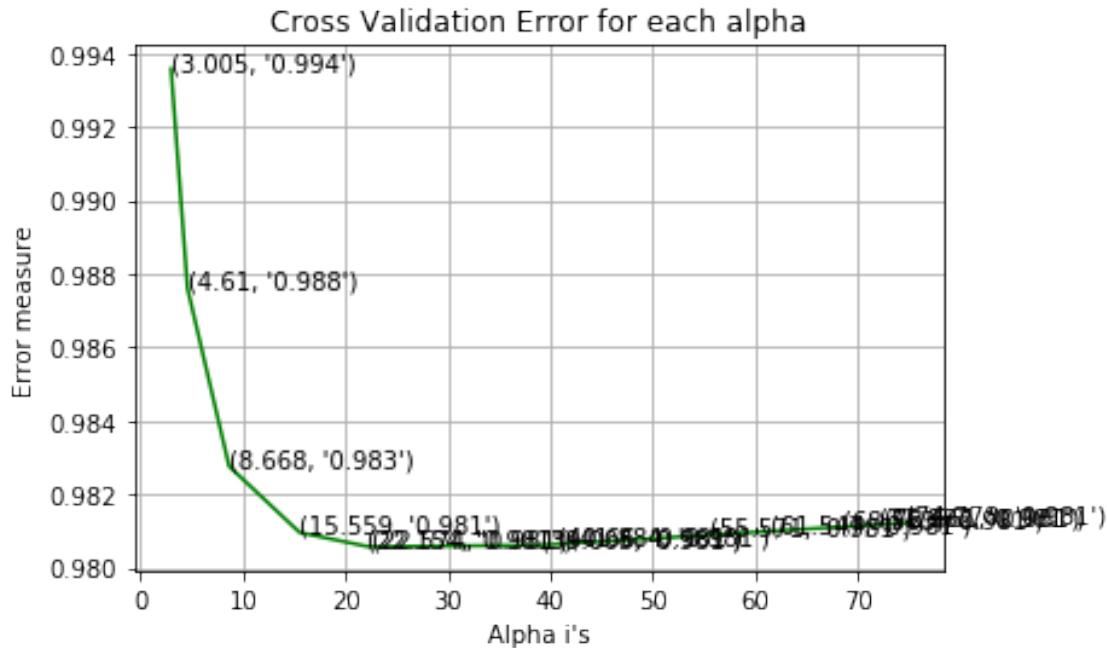
for alpha = 3.005
Log Loss : 0.9935788958393461
for alpha = 4.61
Log Loss : 0.9876043296306164
for alpha = 8.668
Log Loss : 0.9827589127377531
for alpha = 15.559
Log Loss : 0.9809540236022158
for alpha = 22.154
Log Loss : 0.9805754855190898
for alpha = 22.574
Log Loss : 0.9805656894320169
for alpha = 39.005
Log Loss : 0.980609781398943
for alpha = 40.66
Log Loss : 0.9806341442713474
for alpha = 41.884
Log Loss : 0.9806529131640569
for alpha = 55.571
Log Loss : 0.9808857825542455
for alpha = 61.547
Log Loss : 0.9809905492331642
for alpha = 68.563
Log Loss : 0.9811113364198727
for alpha = 71.982

```

```

Log Loss : 0.9811687949395094
for alpha = 72.661
Log Loss : 0.9811800867414179
for alpha = 74.778
Log Loss : 0.9812151022192704

```



```

For values of best alpha = 22.574 The train log loss is: 0.42102639411316267
For values of best alpha = 22.574 The cross validation log loss is: 0.981152725224176
For values of best alpha = 22.574 The test log loss is: 0.9570353248893179

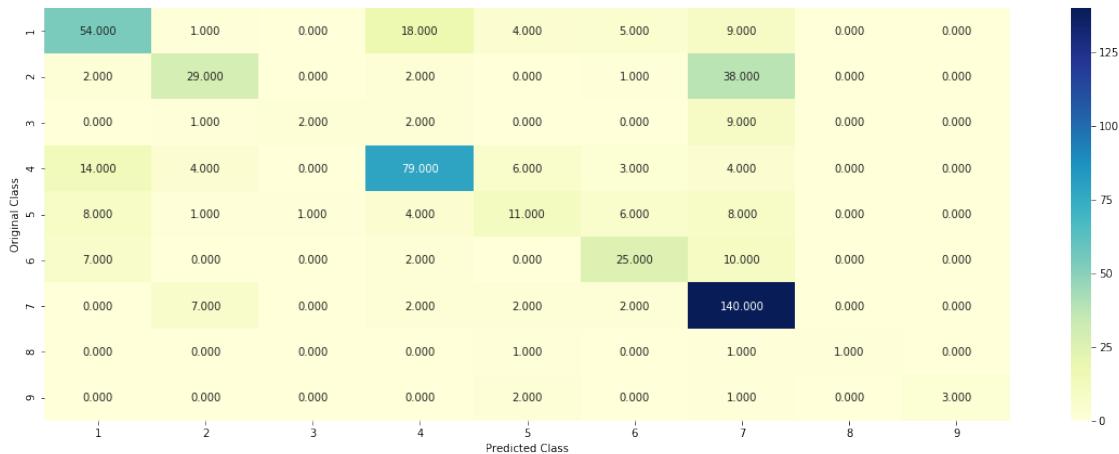
```

```
In [78]: clf = LogisticRegression(C=22.574, class_weight='balanced', dual=True, solver='liblinear')
sig_clf,temp = predict_and_plot_confusion_matrix(train_x_tfidfCoding_lemma, train_y, c)
list_data = []
list_data.append('Tf_Idf+Wordnetlemm+(Liblinear+dual+LR)+Class_Balance')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_tfidfCoding_lemma), eps=1e-15))
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_tfidfCoding_lemma), eps=1e-15))
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_tfidfCoding_lemma), eps=1e-15))
list_data.append('C = '+str(clf.C))
list_data.append(temp)
final_results.append(list_data)
```

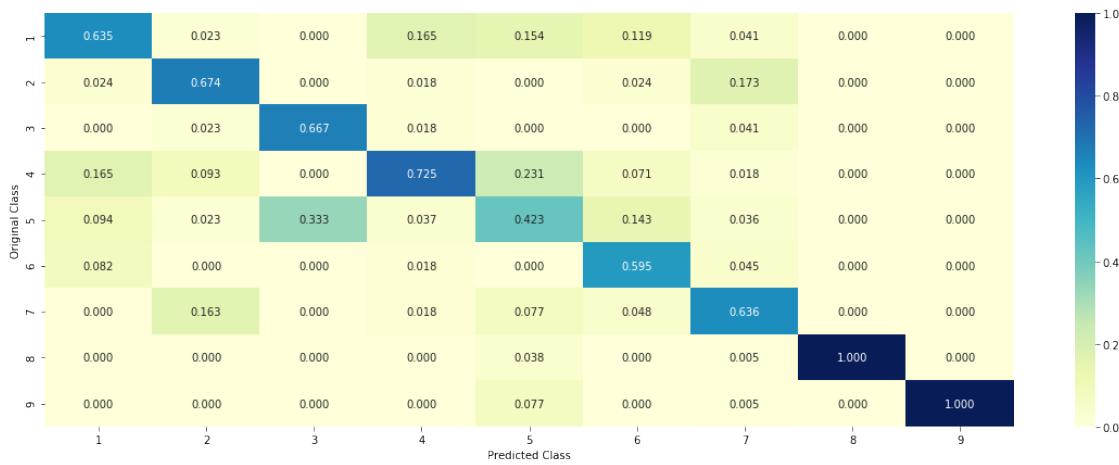
```

Log loss : 0.9805656405108626
Number of mis-classified points : 0.3533834586466165
----- Confusion matrix -----

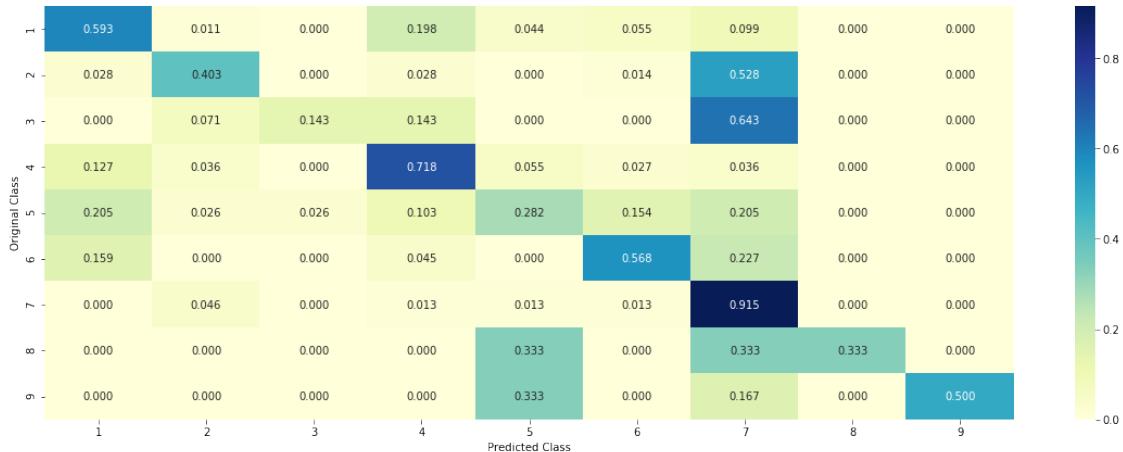
```



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



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



## With BoW

```
In [82]: alpha = [10 ** x for x in range(-6, 3)]
#alpha = np.random.uniform(0.00005, 0.0005, 15)
#alpha = np.round(alpha,7)
#alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = LogisticRegression(C=i, class_weight='balanced', dual=True, solver='liblinear')
    #clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log',
    clf.fit(train_x_onehotCoding_lemma, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding_lemma, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding_lemma)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

best_alpha = np.argmin(cv_log_error_array)
```

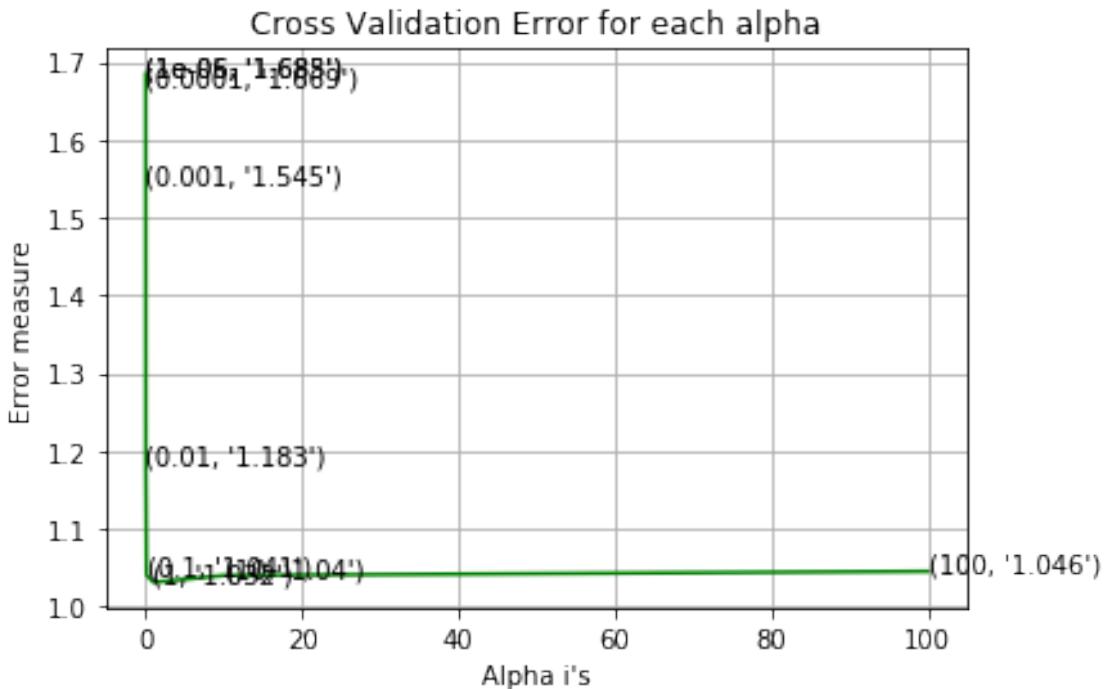
```

clf = LogisticRegression(C=i,class_weight='balanced',dual=True,solver='liblinear')
#clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
clf.fit(train_x_onehotCoding_lemma, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding_lemma, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_onehotCoding_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss"
predict_y = sig_clf.predict_proba(test_x_onehotCoding_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_

for alpha = 1e-06
Log Loss : 1.6846279951530376
for alpha = 1e-05
Log Loss : 1.6832308184883178
for alpha = 0.0001
Log Loss : 1.669333491230861
for alpha = 0.001
Log Loss : 1.5447253446110916
for alpha = 0.01
Log Loss : 1.1831837280664048
for alpha = 0.1
Log Loss : 1.041209381608986
for alpha = 1
Log Loss : 1.0315058003069701
for alpha = 10
Log Loss : 1.040332126435828
for alpha = 100
Log Loss : 1.0456315491277204

```



For values of best alpha = 1 The train log loss is: 0.4877379054414307

For values of best alpha = 1 The cross validation log loss is: 1.0397282367119518

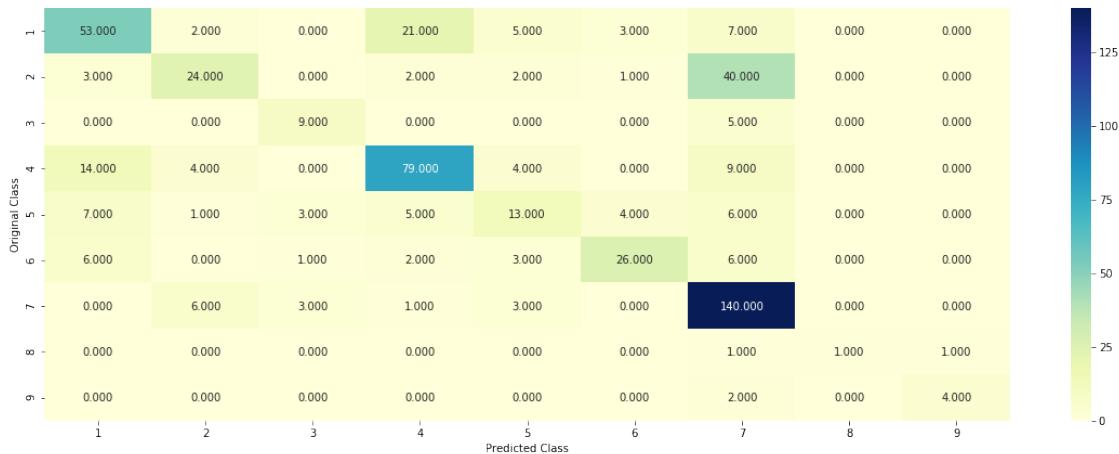
For values of best alpha = 1 The test log loss is: 1.0317701357402798

```
In [83]: clf = LogisticRegression(C=1, class_weight='balanced', dual=True, solver='liblinear')
sig_clf, temp = predict_and_plot_confusion_matrix(train_x_onehotCoding_lemma, train_y,
list_data = []
list_data.append('BoW+Wordnetlemm+(Liblinear+dual+LR)+Class_Balance')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding_lemma),
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding_lemma), eps=1e-10)
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding_lemma), eps=1e-10)
list_data.append('C = '+str(clf.C))
list_data.append(temp)
final_results.append(list_data)
```

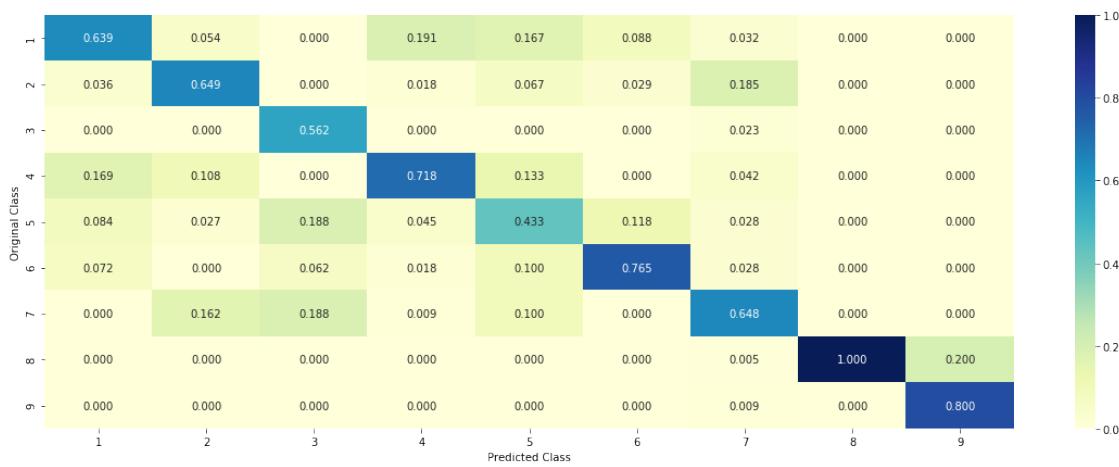
Log loss : 1.0315058751140653

Number of mis-classified points : 0.34398496240601506

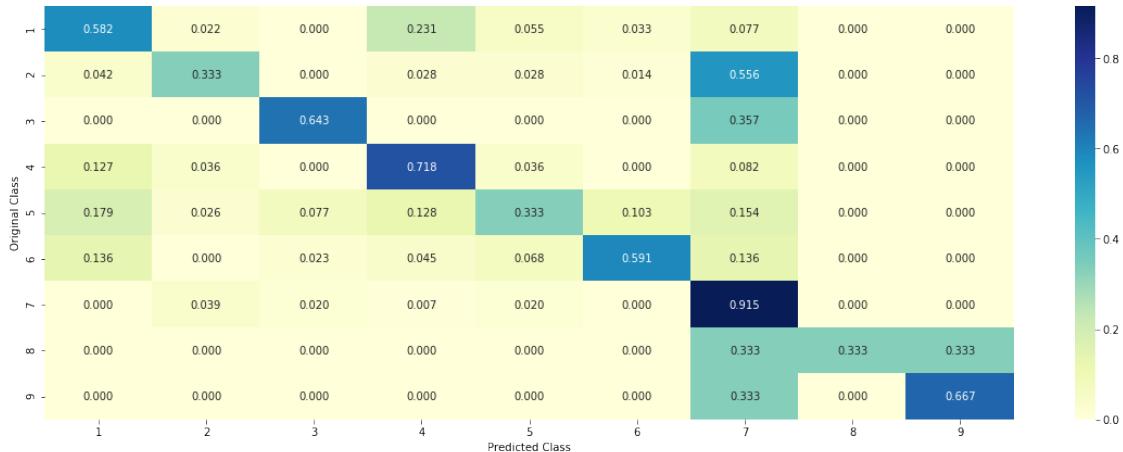
----- Confusion matrix -----



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



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



## With Uni-Bi grams

```
In [87]: 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', random_state=42)
    clf.fit(train_x_tfidfCoding12_lemma, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding12_lemma, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding12_lemma)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability encoding
    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()

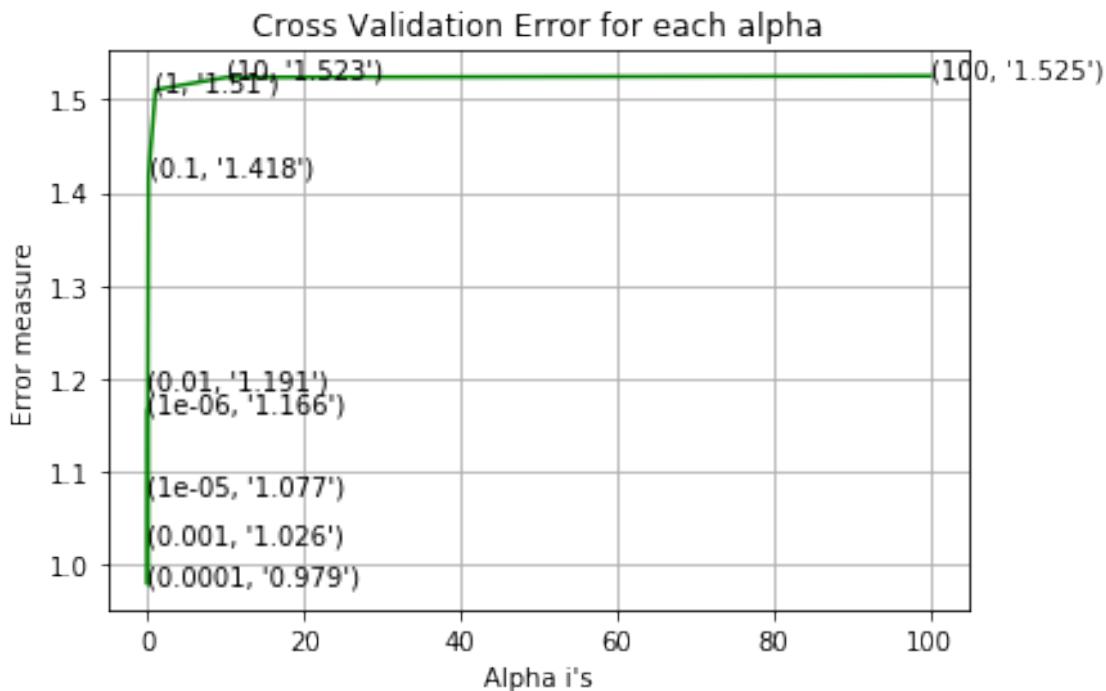
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_tfidfCoding12_lemma, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding12_lemma, train_y)
```

```

predict_y = sig_clf.predict_proba(train_x_tfidfCoding12_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_)
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding12_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_cv)
predict_y = sig_clf.predict_proba(test_x_tfidfCoding12_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_test)

for alpha = 1e-06
Log Loss : 1.166199643759739
for alpha = 1e-05
Log Loss : 1.077257377221936
for alpha = 0.0001
Log Loss : 0.9794795953312689
for alpha = 0.001
Log Loss : 1.025543991442806
for alpha = 0.01
Log Loss : 1.1912433471213488
for alpha = 0.1
Log Loss : 1.4179379590323204
for alpha = 1
Log Loss : 1.5102020029380896
for alpha = 10
Log Loss : 1.5234648930120178
for alpha = 100
Log Loss : 1.525040269159958

```



```

For values of best alpha = 0.0001 The train log loss is: 0.42207818048235696
For values of best alpha = 0.0001 The cross validation log loss is: 0.9794795953312689
For values of best alpha = 0.0001 The test log loss is: 0.9616531897088071

In [88]: alpha = [10 ** x for x in range(-6, 3)]
          alpha = np.random.uniform(0.00005, 0.0005, 20)
          alpha = np.round(alpha, 7)
          alpha.sort()
          cv_log_error_array = []
          for i in alpha:
              print("for alpha =", i)
              clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', r...
              clf.fit(train_x_tfidfCoding12_lemma, train_y)
              sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
              sig_clf.fit(train_x_tfidfCoding12_lemma, train_y)
              sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding12_lemma)
              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()

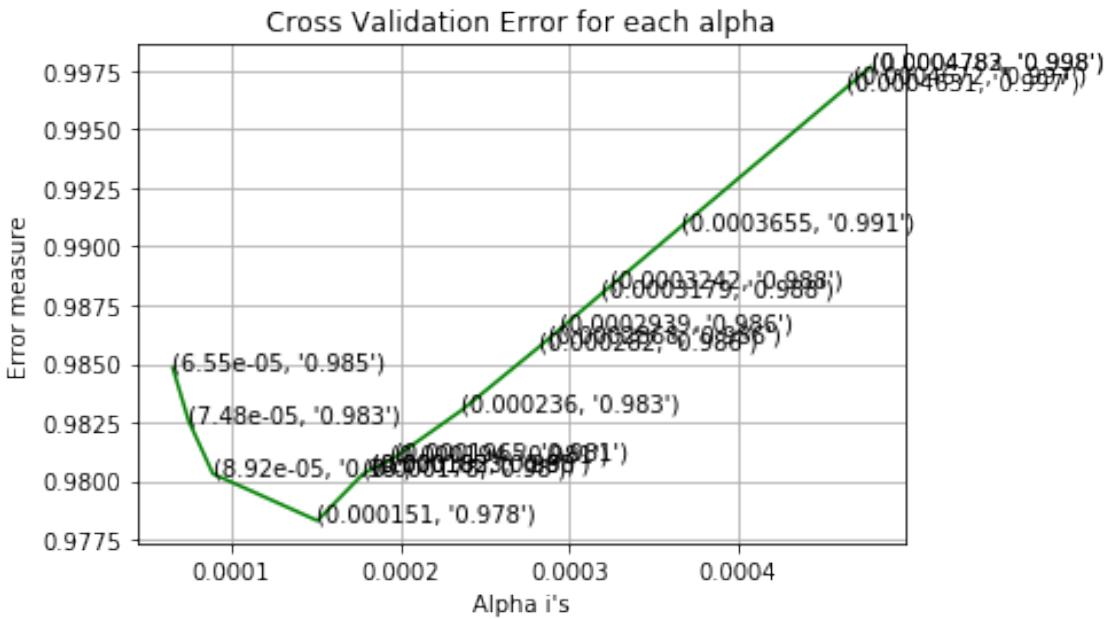
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', r...
clf.fit(train_x_tfidfCoding12_lemma, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding12_lemma, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding12_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_...
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding12_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", ...
predict_y = sig_clf.predict_proba(test_x_tfidfCoding12_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_...

for alpha = 6.55e-05

```

```
Log Loss : 0.9848209581331964
for alpha = 7.48e-05
Log Loss : 0.9825700120157895
for alpha = 8.92e-05
Log Loss : 0.9803057511324388
for alpha = 0.000151
Log Loss : 0.9783069901593616
for alpha = 0.000178
Log Loss : 0.9802977436922015
for alpha = 0.0001823
Log Loss : 0.980473078452225
for alpha = 0.000183
Log Loss : 0.9804958043773845
for alpha = 0.000194
Log Loss : 0.9808759163220442
for alpha = 0.0001965
Log Loss : 0.9809883386891163
for alpha = 0.000236
Log Loss : 0.9829983506151976
for alpha = 0.000282
Log Loss : 0.9857069646882679
for alpha = 0.0002868
Log Loss : 0.9860029863249947
for alpha = 0.0002939
Log Loss : 0.9864406847890889
for alpha = 0.0003179
Log Loss : 0.9879138089574063
for alpha = 0.0003242
Log Loss : 0.9882984499792187
for alpha = 0.0003655
Log Loss : 0.9908063134748588
for alpha = 0.0004631
Log Loss : 0.996727305319625
for alpha = 0.0004672
Log Loss : 0.9969761867491199
for alpha = 0.0004782
Log Loss : 0.9976435476772695
for alpha = 0.0004783
Log Loss : 0.9976496115361158
```



For values of best alpha = 0.000151 The train log loss is: 0.4406039441886821

For values of best alpha = 0.000151 The cross validation log loss is: 0.9783069901593616

For values of best alpha = 0.000151 The test log loss is: 0.95945553450423

```
In [89]: alpha = [10 ** x for x in range(-6, 3)]
#alpha = np.random.uniform(0.00005,0.0005,20)
#alpha = np.round(alpha,7)
#alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    #clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log',
    clf = LogisticRegression(C=i,class_weight='balanced',dual=True,solver='liblinear')
    clf.fit(train_x_tfidfCoding12_lemma, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding12_lemma, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding12_lemma)
    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()
```

```

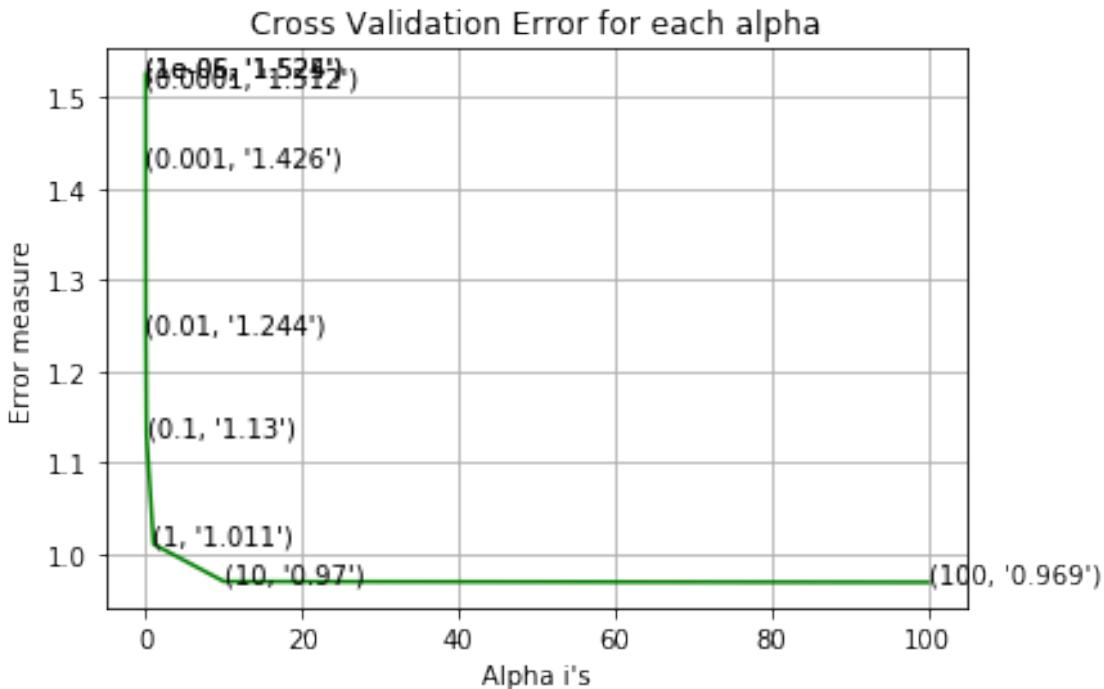
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

best_alpha = np.argmin(cv_log_error_array)
#clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
clf = LogisticRegression(C=alpha[best_alpha], class_weight='balanced', dual=True, solver='liblinear')
clf.fit(train_x_tfidfCoding12_lemma, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding12_lemma, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding12_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,predict_y))
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding12_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv,predict_y))
predict_y = sig_clf.predict_proba(test_x_tfidfCoding12_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test,predict_y))

for alpha = 1e-06
Log Loss : 1.5253415500187415
for alpha = 1e-05
Log Loss : 1.524070382120419
for alpha = 0.0001
Log Loss : 1.511862823603336
for alpha = 0.001
Log Loss : 1.4264205465018682
for alpha = 0.01
Log Loss : 1.244010119202379
for alpha = 0.1
Log Loss : 1.1301390767350503
for alpha = 1
Log Loss : 1.0108608129478078
for alpha = 10
Log Loss : 0.970000204071783
for alpha = 100
Log Loss : 0.9690787836107145

```



For values of best alpha = 100 The train log loss is: 0.3787665372989494

For values of best alpha = 100 The cross validation log loss is: 0.9690788242902345

For values of best alpha = 100 The test log loss is: 0.9516237248417849

```
In [90]: alpha = [10 ** x for x in range(-6, 3)]
alpha = np.random.uniform(5,150,20)
alpha = np.round(alpha,7)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    #clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log',
    clf = LogisticRegression(C=i, class_weight='balanced', dual=True, solver='liblinear')
    clf.fit(train_x_tfidfCoding12_lemma, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding12_lemma, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding12_lemma)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
```

```

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

best_alpha = np.argmin(cv_log_error_array)
#clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
clf = LogisticRegression(C=alpha[best_alpha], class_weight='balanced', dual=True, solver='liblinear')
clf.fit(train_x_tfidfCoding12_lemma, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding12_lemma, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding12_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,predict_y))
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding12_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv,predict_y))
predict_y = sig_clf.predict_proba(test_x_tfidfCoding12_lemma)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test,predict_y))

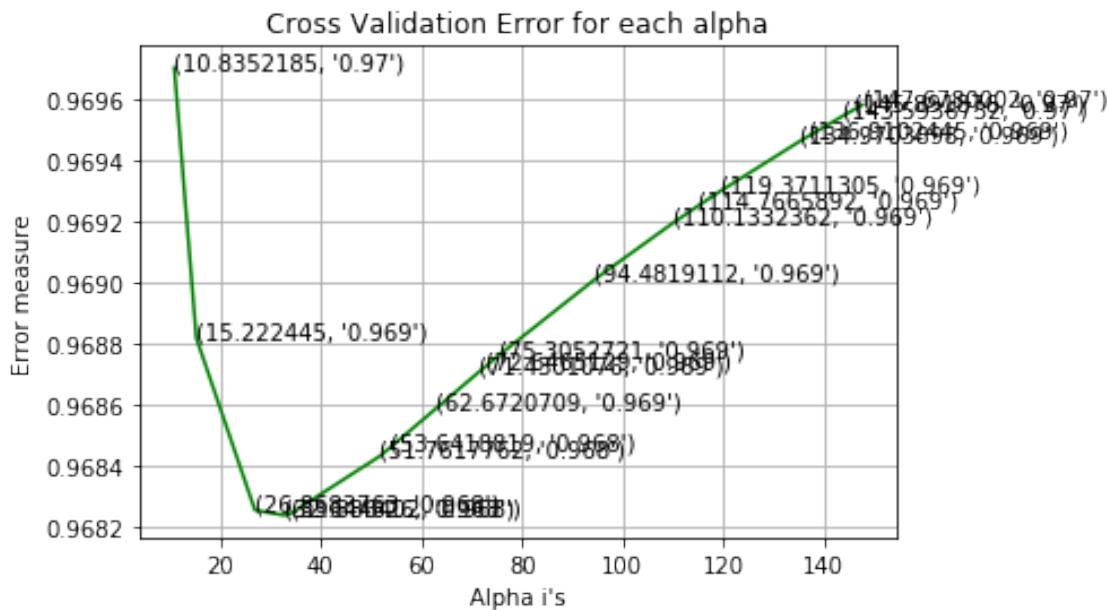
for alpha = 10.8352185
Log Loss : 0.9697060266517539
for alpha = 15.222445
Log Loss : 0.9688177829380734
for alpha = 26.8683763
Log Loss : 0.9682518941239794
for alpha = 32.644826
Log Loss : 0.9682353309642812
for alpha = 33.886402
Log Loss : 0.9682398694561938
for alpha = 51.7617762
Log Loss : 0.9684301777771828
for alpha = 53.6418819
Log Loss : 0.9684560294190749
for alpha = 62.6720709
Log Loss : 0.9685832692471412
for alpha = 71.4301076
Log Loss : 0.9687065045016026
for alpha = 72.6465129
Log Loss : 0.9687234235821763
for alpha = 75.3052721
Log Loss : 0.9687600263278746
for alpha = 94.4819112
Log Loss : 0.9690111540739916
for alpha = 110.1332362

```

```

Log Loss : 0.96919777918077
for alpha = 114.7665892
Log Loss : 0.9692499574332806
for alpha = 119.3711305
Log Loss : 0.9693004455354026
for alpha = 134.9703898
Log Loss : 0.9694624421499135
for alpha = 136.9102445
Log Loss : 0.9694816235544842
for alpha = 143.5936732
Log Loss : 0.9695463617120776
for alpha = 145.891876
Log Loss : 0.9695681259838823
for alpha = 147.6780002
Log Loss : 0.969584853514492

```



For values of best alpha = 32.644826 The train log loss is: 0.39313063660967296  
 For values of best alpha = 32.644826 The cross validation log loss is: 0.9682353279222855  
 For values of best alpha = 32.644826 The test log loss is: 0.9490206683599927

```

In [91]: clf = LogisticRegression(C=32.644826,class_weight='balanced',dual=True,solver='liblinear')
sig_clf,temp = predict_and_plot_confusion_matrix(train_x_tfidfCoding12_lemma, train_y)
list_data = []
list_data.append('Tf_Idf_uni-bigram+lemma+(Liblinear+dual+LR)+Class_Balance')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_tfidfCoding12_lemma)))

```

```

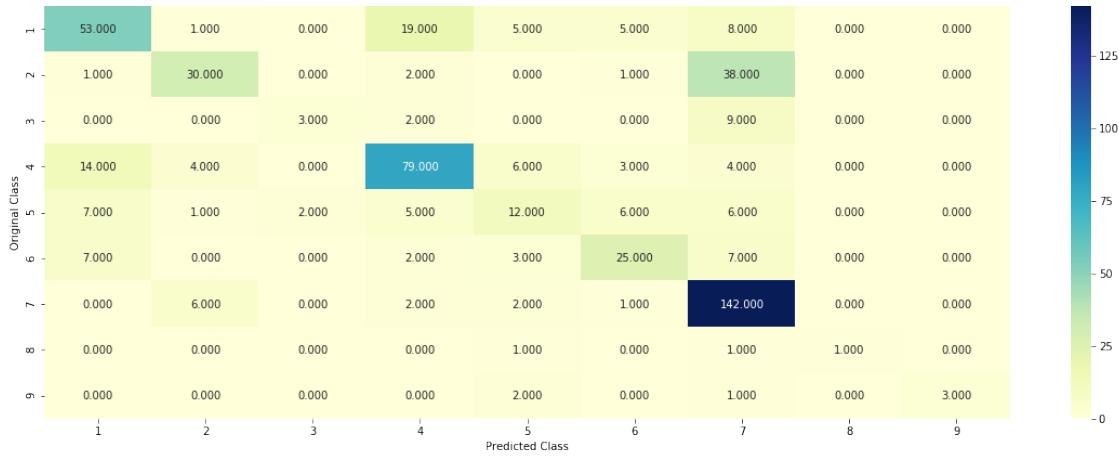
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_tfidfCoding12_lemma), eps=0.001))
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_tfidfCoding12_lemma), eps=0.001))
list_data.append('C = '+str(clf.C))
list_data.append(temp)
final_results.append(list_data)

```

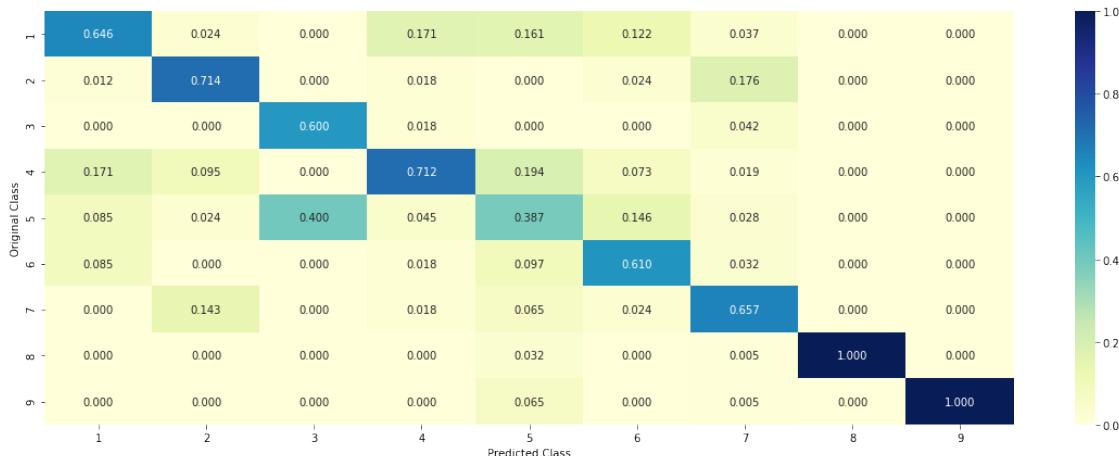
Log loss : 0.9682353300456913

Number of mis-classified points : 0.3458646616541353

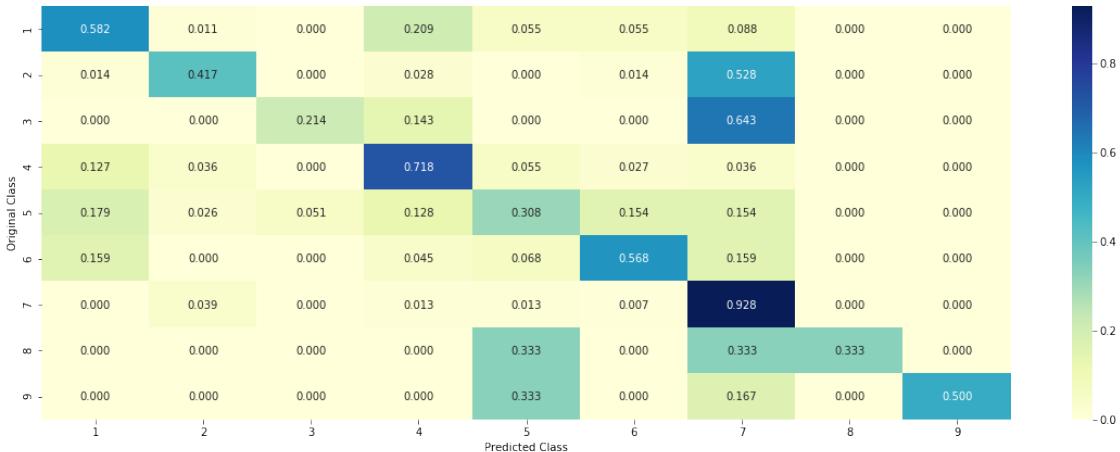
----- Confusion matrix -----



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



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



## 8.2. Some Word2Vec transformations

Tried with glove, google and bionlp word2vec pretrained models and get some good results with bionlp so using bionlp (<http://bio.nlplab.org/#word-vectors> , <http://evexdb.org/pmresources/vec-space-models/> )

```
In [ ]: ! wget http://evexdb.org/pmresources/vec-space-models/wikipedia-pubmed-and-PMC-w2v.bin
```

```
In [66]: import gensim
from gensim.models import KeyedVectors
w2v_model = gensim.models.KeyedVectors.load_word2vec_format('wikipedia-pubmed-and-PMC-w2v.bin')
```

```
In [67]: vc = [i for i in data.Gene if i not in w2v_model.vocab.keys()]
```

```
In [68]: vc
```

```
Out[68]: ['KMT2D',
 'KMT2D',
 'KMT2C',
 'KMT2C',
 'KMT2C',
 'KMT2C',
 'KNSTRN',
 'KNSTRN',
 'KNSTRN',
 'KNSTRN']
```

find that 3 genes are not there in pmc data word2vec. For this checked in [www.genecards.org](http://www.genecards.org) and got some aliases and changed with that.

for KMT2D - ALL

KMT2C - MLL3

KNSTRN - SKAP

```
In [69]: def Gene_vector(gene):
    if gene == 'KMT2D':
        gene = 'ALL'
    if gene == 'KMT2C':
        gene = 'MLL3'
    if gene == 'KNSTRN':
        gene = 'SKAP'
    return w2v_model[gene]
def Gene_avg_vector(gene):
    if gene == 'KMT2D':
        gene = 'ALL'
    if gene == 'KMT2C':
        gene = 'MLL3'
    if gene == 'KNSTRN':
        gene = 'SKAP'
    return np.mean(w2v_model[gene])

In [70]: #W2v for Gene feature
train_gene_feature_w2v = np.array(train_df.Gene.apply(lambda x: Gene_vector(x)).tolist())
test_gene_feature_w2v = np.array(test_df.Gene.apply(lambda x: Gene_vector(x)).tolist())
cv_gene_feature_w2v = np.array(cv_df.Gene.apply(lambda x: Gene_vector(x)).tolist())

In [71]: train_gene_feature_avgw2v = train_df.Gene.apply(lambda x: Gene_avg_vector(x)).values.reshape(-1,1)
test_gene_feature_avgw2v = test_df.Gene.apply(lambda x: Gene_avg_vector(x)).values.reshape(-1,1)
cv_gene_feature_avgw2v = cv_df.Gene.apply(lambda x: Gene_avg_vector(x)).values.reshape(-1,1)

In [72]: plot_df = pd.DataFrame(np.hstack((train_gene_feature_avgw2v,y_train.reshape(-1,1))))
```

```
In [73]: temp_df = plot_df.groupby(1).describe()
temp_df[0].sort_values('75%')

Out[73]:
```

	count	mean	std	min	25%	50%	75%	\
1	8.0	12.0	-0.007021	0.009565	-0.025787	-0.007320	-0.005693	-0.000589
2.0	9.0	24.0	-0.002970	0.011035	-0.027127	-0.007407	0.000113	0.000113
3.0	2.0	289.0	-0.002322	0.010786	-0.038673	-0.007421	-0.000556	0.002595
4.0	7.0	609.0	-0.001764	0.009648	-0.038673	-0.007421	-0.001360	0.005617
5.0	5.0	57.0	0.004156	0.011425	-0.023000	-0.004053	0.006543	0.014311
6.0	4.0	155.0	0.008159	0.010359	-0.023520	0.000450	0.014311	0.014995
7.0	1.0	439.0	0.008354	0.012725	-0.028747	0.004648	0.006710	0.016127
8.0	1.0	363.0	0.008136	0.014343	-0.038673	0.000282	0.008991	0.017592
9.0	6.0	176.0	0.008779	0.012266	-0.023520	0.005374	0.014995	0.017592
			max					
	1	8.0	0.004505					
	2.0	9.0	0.025762					
	3.0	2.0	0.025762					
	4.0	7.0	0.016594					

```
3.0  0.025762
5.0  0.030619
4.0  0.030619
1.0  0.047780
6.0  0.027838
```

```
In [187]: from sklearn.manifold import TSNE
```

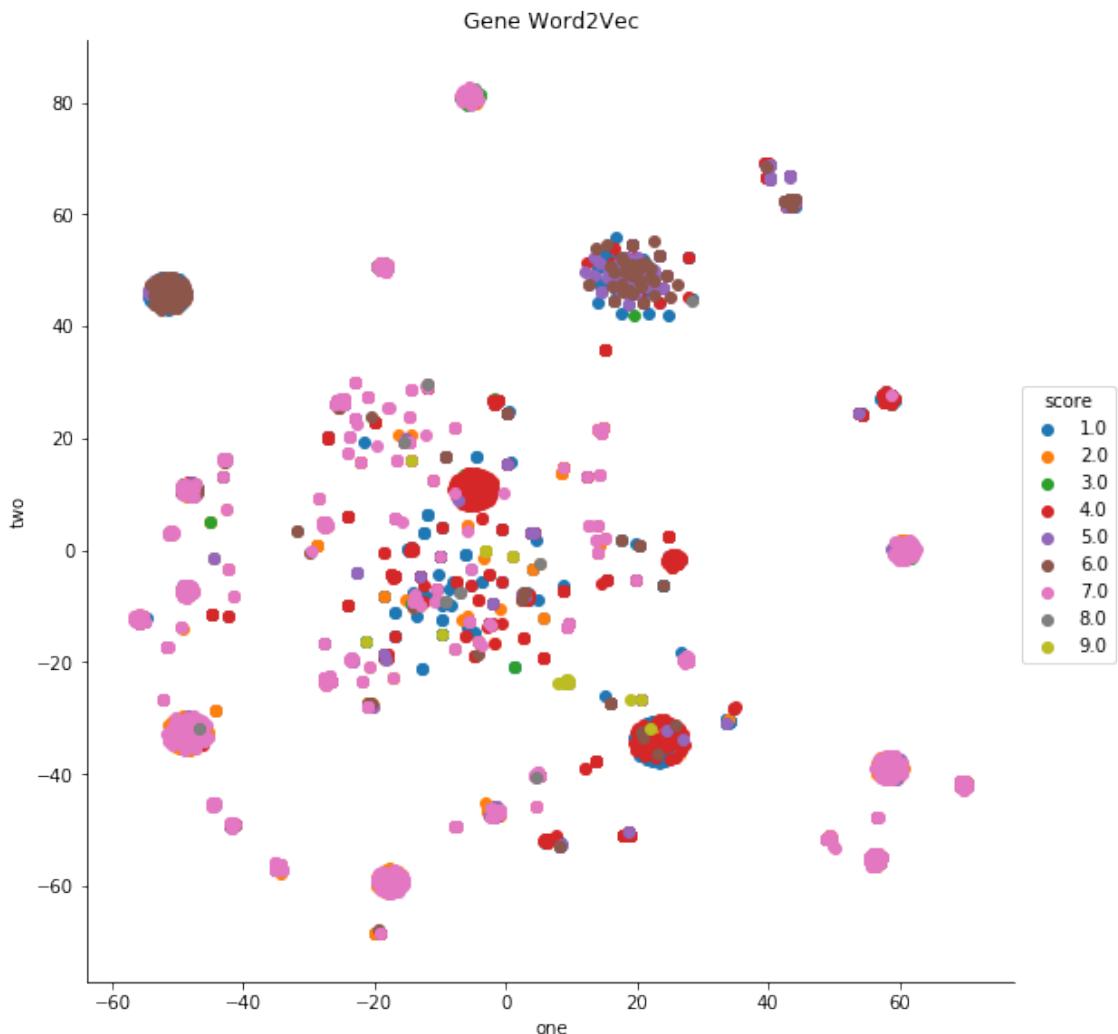
```
X_embedded = TSNE(n_components=2, perplexity=35, n_iter=3500).fit_transform(train_gene)
```

```
In [188]: plot_df = pd.DataFrame(np.hstack((X_embedded, y_train.reshape(-1,1))), columns=['one',
```

```
In [189]: %matplotlib inline
```

```
sns.FacetGrid(plot_df, hue="score", size=8).\\
map(plt.scatter, 'one', 'two').add_legend()
plt.title('Gene Word2Vec')
```

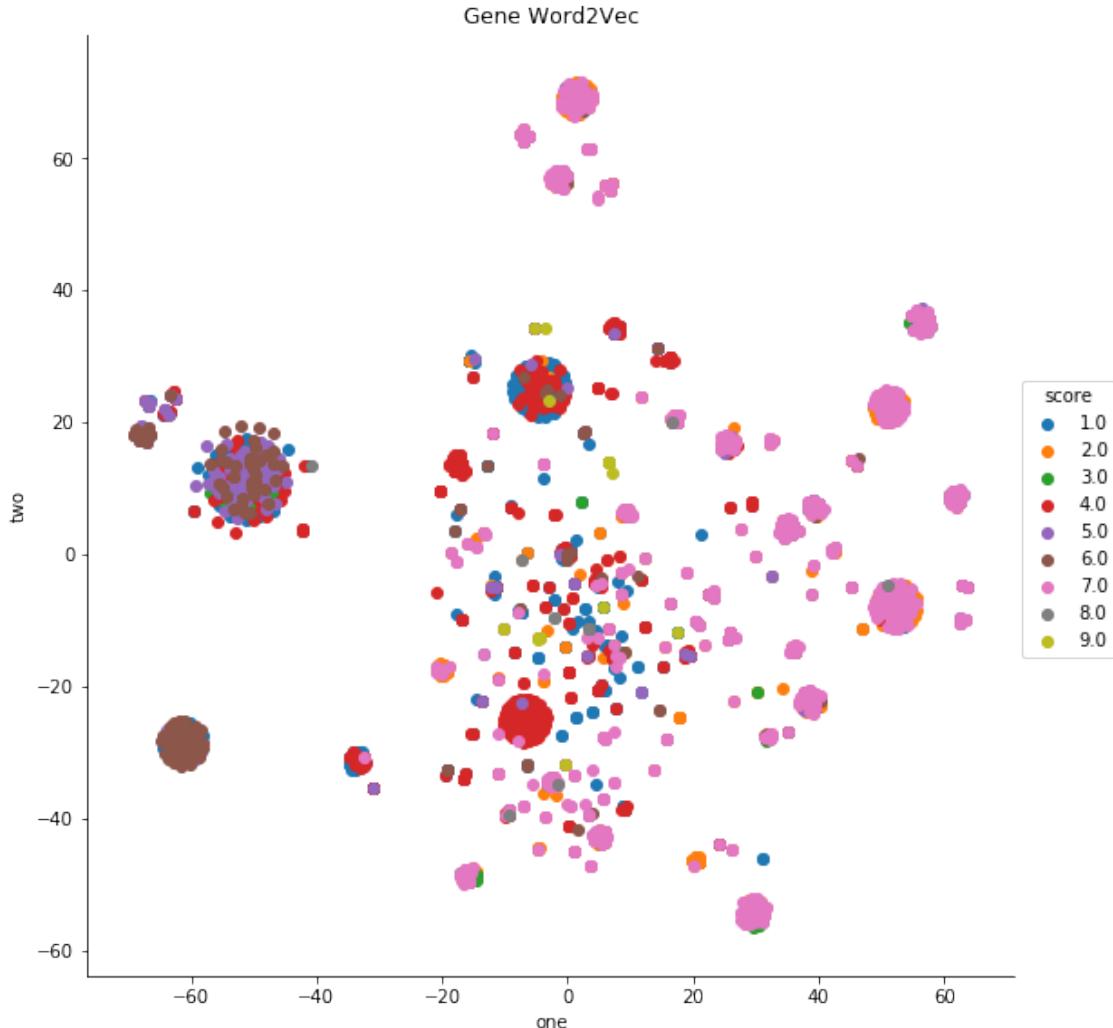
```
Out[189]: Text(0.5,1,'Gene Word2Vec')
```



```
In [190]: from sklearn.manifold import TSNE
```

```
X_embedded = TSNE(n_components=2, perplexity=55, n_iter=3500).fit_transform(train_gene)
plot_df = pd.DataFrame(np.hstack((X_embedded, y_train.reshape(-1,1))), columns=['one', '%matplotlib inline
sns.FacetGrid(plot_df, hue="score", size=8).\
map(plt.scatter, 'one', 'two').add_legend()
plt.title('Gene Word2Vec')
```

```
Out[190]: Text(0.5,1,'Gene Word2Vec')
```

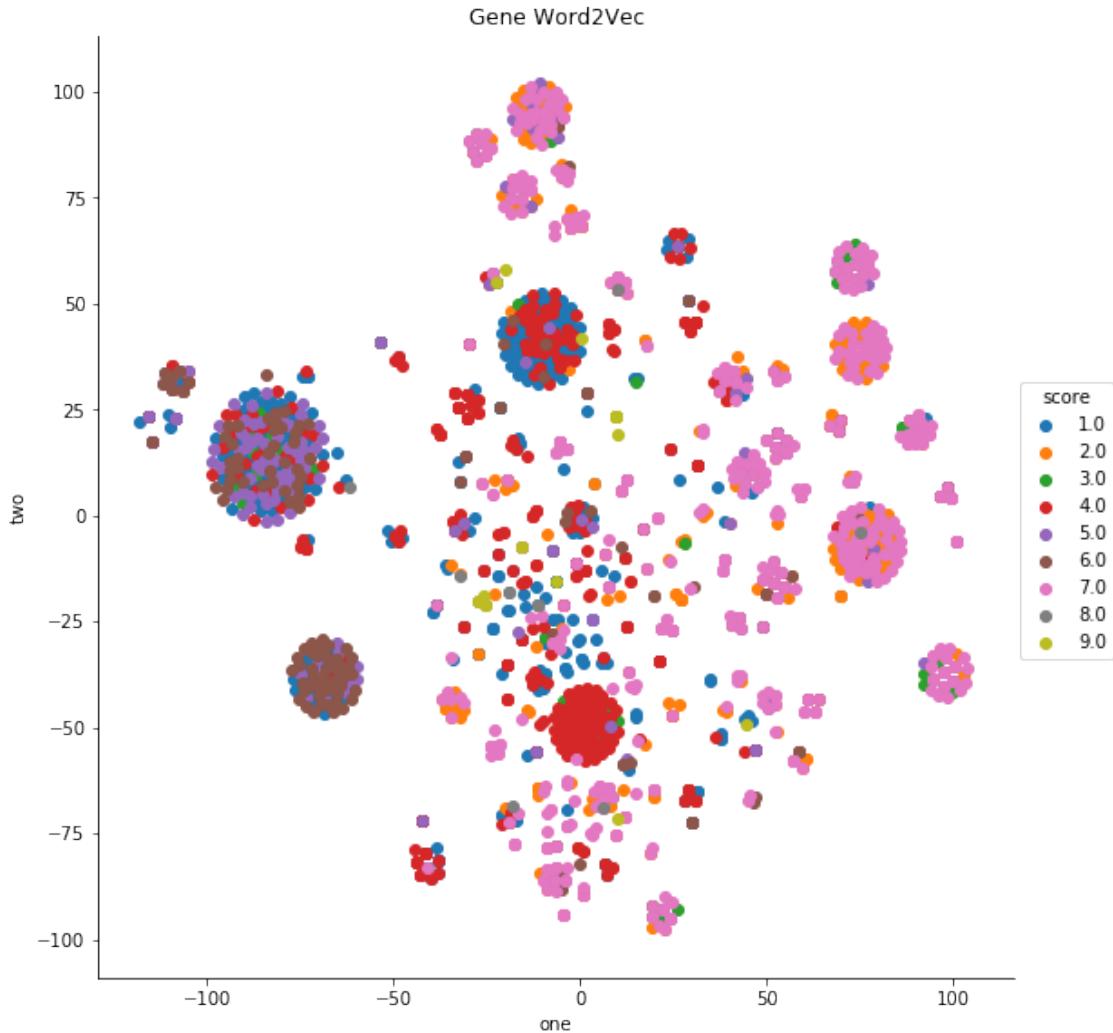


```
In [191]: from sklearn.manifold import TSNE
```

```
X_embedded = TSNE(n_components=2, perplexity=75, n_iter=4000).fit_transform(train_gene)
plot_df = pd.DataFrame(np.hstack((X_embedded, y_train.reshape(-1,1))), columns=['one', '%matplotlib inline
sns.FacetGrid(plot_df, hue="score", size=8).\
map(plt.scatter, 'one', 'two').add_legend()
plt.title('Gene Word2Vec')
```

```
map(plt.scatter,'one','two').add_legend()  
plt.title('Gene Word2Vec')
```

Out[191]: Text(0.5,1,'Gene Word2Vec')



Word vectors are not that good at classifying some types. making some ml modeles to check

```
In [83]: train_gene_var_tfidfCoding1 = hstack((train_gene_feature_w2v,train_variation_feature_tf  
test_gene_var_tfidfCoding1 = hstack((test_gene_feature_w2v,test_variation_feature_tf  
cv_gene_var_tfidfCoding1 = hstack((cv_gene_feature_w2v, cv_variation_feature_tfidfCodi  
  
train_x_tfidfCoding12_w2v = hstack((train_gene_var_tfidfCoding1,train_text_feature_tf  
train_y = np.array(list(train_df['Class']))  
  
test_x_tfidfCoding12_w2v = hstack((test_gene_var_tfidfCoding1,test_text_feature_tf  
test_y = np.array(list(test_df['Class'])))
```

```

cv_x_tfidfCoding12_w2v = hstack((cv_gene_var_tfidfCoding1, cv_text_feature_tfidfCoding1))
cv_y = np.array(list(cv_df['Class']))

In [193]: alpha = [10 ** x for x in range(-6, 3)]
#alpha = np.random.uniform(0.00005, 0.0005, 20)
#alpha = np.round(alpha, 7)
#alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', max_iter=1000)
    #clf = LogisticRegression(C=i, class_weight='balanced', dual=True, solver='liblinear')
    clf.fit(train_x_tfidfCoding12_w2v, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding12_w2v, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding12_w2v)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', max_iter=1000)
#clf = LogisticRegression(C=alpha[best_alpha], class_weight='balanced', dual=True, solver='liblinear')
clf.fit(train_x_tfidfCoding12_w2v, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding12_w2v, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding12_w2v)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(cv_y, predict_y))
predict_y = sig_clf.predict_proba(cv_x_tfidfCoding12_w2v)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(cv_y, predict_y))
predict_y = sig_clf.predict_proba(test_x_tfidfCoding12_w2v)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(cv_y, predict_y))

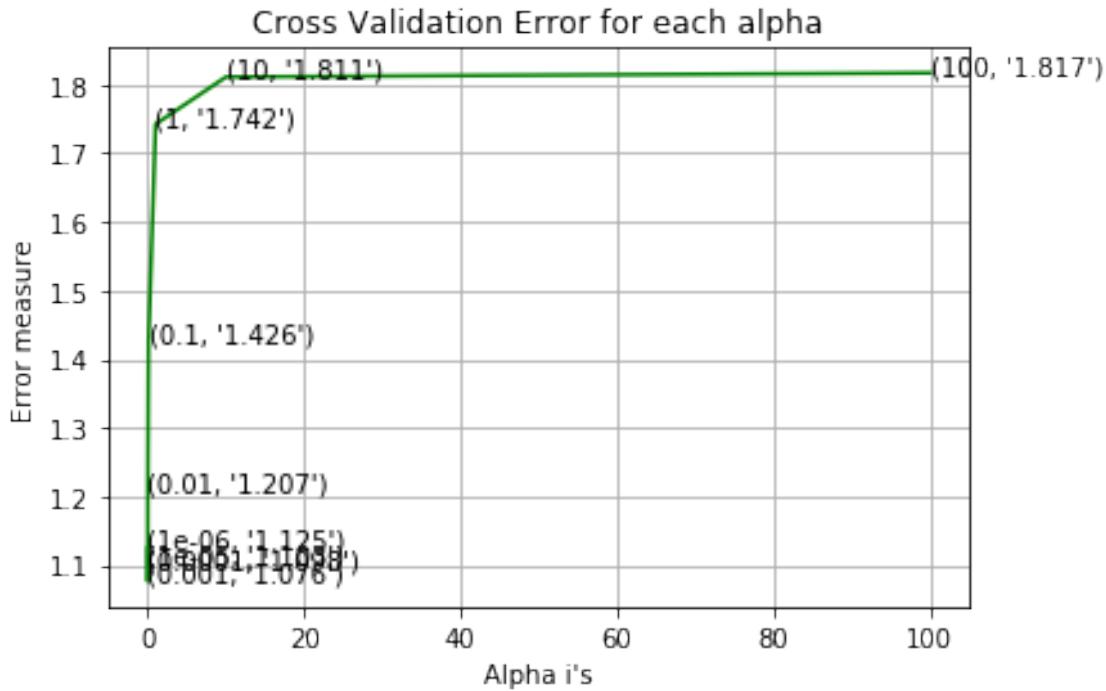
for alpha = 1e-06
Log Loss : 1.124732531108742

```

```

for alpha = 1e-05
Log Loss : 1.1033808940958034
for alpha = 0.0001
Log Loss : 1.0975128218792758
for alpha = 0.001
Log Loss : 1.0764025493080294
for alpha = 0.01
Log Loss : 1.207476055643181
for alpha = 0.1
Log Loss : 1.4262010152592337
for alpha = 1
Log Loss : 1.7422006780274262
for alpha = 10
Log Loss : 1.8110026694373917
for alpha = 100
Log Loss : 1.8168492010595712

```



For values of best alpha = 0.001 The train log loss is: 0.7731255468402026  
 For values of best alpha = 0.001 The cross validation log loss is: 1.0764025493080294  
 For values of best alpha = 0.001 The test log loss is: 1.0634330476554723

In [83]: `train_gene_var_tfidfCoding1 = hstack((train_gene_feature_w2v,train_variation_feature_tfidfCoding1))`  
`test_gene_var_tfidfCoding1 = hstack((test_gene_feature_w2v,test_variation_feature_tfidfCoding1))`

```

cv_gene_var_tfidfCoding1 = hstack((cv_gene_feature_w2v, cv_variation_feature_tfidfCoding))

train_x_tfidfCoding12_w2v = hstack((train_gene_var_tfidfCoding1, train_text_feature_tfidfCoding))
train_y = np.array(list(train_df['Class']))

test_x_tfidfCoding12_w2v = hstack((test_gene_var_tfidfCoding1, test_text_feature_tfidfCoding))
test_y = np.array(list(test_df['Class']))

cv_x_tfidfCoding12_w2v = hstack((cv_gene_var_tfidfCoding1, cv_text_feature_tfidfCoding))
cv_y = np.array(list(cv_df['Class']))

In [195]: alpha = [10 ** x for x in range(-6, 3)]
#alpha = np.random.uniform(0.00005, 0.0005, 20)
#alpha = np.round(alpha, 7)
#alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', max_iter=1000)
    #clf = LogisticRegression(C=i, class_weight='balanced', dual=True, solver='liblinear')
    clf.fit(train_x_tfidfCoding12_w2v, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidfCoding12_w2v, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidfCoding12_w2v)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log')
#clf = LogisticRegression(C=alpha[best_alpha], class_weight='balanced', dual=True, solver='liblinear')
clf.fit(train_x_tfidfCoding12_w2v, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidfCoding12_w2v, train_y)

predict_y = sig_clf.predict_proba(train_x_tfidfCoding12_w2v)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(predict_y, train_y))

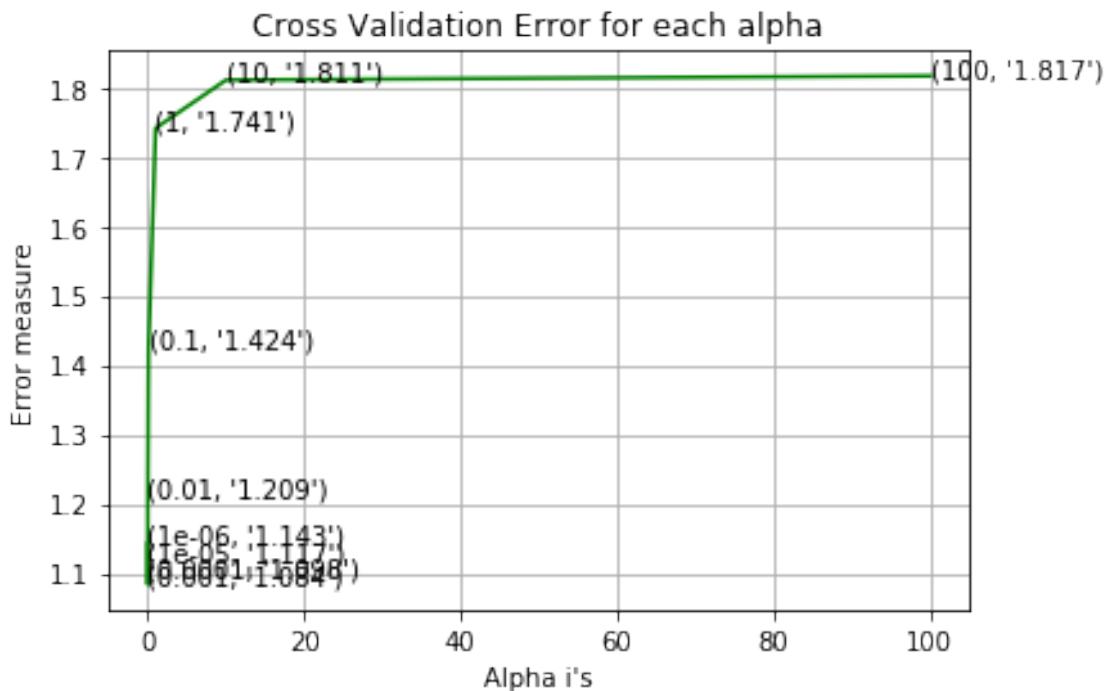
```

```

predict_y = sig_clf.predict_proba(cv_x_tfidfCoding12_w2v)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss_cv)
predict_y = sig_clf.predict_proba(test_x_tfidfCoding12_w2v)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss)

for alpha = 1e-06
Log Loss : 1.1428609508234124
for alpha = 1e-05
Log Loss : 1.1168497584484731
for alpha = 0.0001
Log Loss : 1.0976865550302808
for alpha = 0.001
Log Loss : 1.083781084871808
for alpha = 0.01
Log Loss : 1.2089479523962916
for alpha = 0.1
Log Loss : 1.4244416779085578
for alpha = 1
Log Loss : 1.741388863112138
for alpha = 10
Log Loss : 1.8108015203815786
for alpha = 100
Log Loss : 1.8166952978199693

```



```
For values of best alpha = 0.001 The train log loss is: 0.7857700590033223
For values of best alpha = 0.001 The cross validation log loss is: 1.083781084871808
For values of best alpha = 0.001 The test log loss is: 1.06866809990059
```

Didnt get any good scores using that

### 8.3. Some Transformation on Variation Feature

```
In [88]: data.Variation.value_counts().head(15)
```

```
Out[88]: Truncating Mutations      93
          Deletion                 74
          Amplification            71
          Fusions                  34
          Overexpression            6
          G12V                     4
          Q61L                     3
          T58I                     3
          Q61R                     3
          E17K                     3
          Q61H                     3
          Promoter Hypermethylation 2
          G13D                     2
          I31M                     2
          R170W                    2
          Name: Variation, dtype: int64
```

if we see above values of variation we can Truncating Mutations,Deletion,Amplification,Fusions,Overexpression are types and for type Q61L found from this link <https://cancer.sanger.ac.uk/cosmic/mutation/overview?id=583> that it is Substitution at position 61 for amino acid Q to L so from every variation got variation type and if variation type is substitution then got position of substitution and what are replaced values.

1.First transformation is if variation is type Q61L then we will get position amino before state amino after state. and that type is substitution. so we can compile with `r'^([a-z*])(1,7)([a-z*])$'`

```
In [99]: data[data['Variation'].str.contains('trunc') == True]
```

```
Out[99]:      ID   Gene     Variation  Class
    1707  1707  PPM1D    422_605trunc    7
    2123  2123  CCND1    256_286trunc    7
    3096  3096  NOTCH2    1_2009trunc    1
    3097  3097  NOTCH2  2010_2471trunc    2
```

2.For Trunc getting feature name as trunc and another feature as difference between two truncated values

```
In [104]: data[data['Variation'].str.contains('del') == True].head(10)
```

```
Out[104]:      ID  Gene          Variation  Class
    138  138  EGFR        L747_T751delinsP    7
    139  139  EGFR        S752_I759del    2
    149  149  EGFR        K745_A750del    7
    166  166  EGFR        E746_A750del    7
   169  169  EGFR  Exon 19 deletion/insertion    7
   171  171  EGFR        A859_L883delinsV    2
   174  174  EGFR        A750_E758del    7
   184  184  EGFR        A750_E758delinsP    7
   187  187  EGFR        L747_P753delinsS    7
   188  188  EGFR  Exon 19 deletion    7
```

3.Getting start amino acid end amino acid and difference

```
In [105]: data[data['Variation'].str.contains('ins')] == True].head(10)
```

```
Out[105]:      ID  Gene          Variation  Class
    138  138  EGFR        L747_T751delinsP    7
    146  146  EGFR        E746_T751insIP    7
    147  147  EGFR        D770_N771insD    7
    165  165  EGFR        D770_N771insNPG    7
   169  169  EGFR  Exon 19 deletion/insertion    7
   171  171  EGFR        A859_L883delinsV    2
   175  175  EGFR        V769_D770insGVV    7
   184  184  EGFR        A750_E758delinsP    7
   187  187  EGFR        L747_P753delinsS    7
   193  193  EGFR        H773insLGNP    7
```

```
In [106]: data[data['Variation'].str.contains('splice')] == True].head(10)
```

```
Out[106]:      ID  Gene          Variation  Class
    285  285  EIF1AX        A113_splice    7
    868  868  HLA-A        596_619splice    1
   1118 1118    MET        X1007_splice    7
   1129 1129    MET        X963_splice    7
   1132 1132    MET        X1009_splice    7
   1138 1138    MET        981_1028splice    7
   1140 1140    MET        X1008_splice    7
   1148 1148    MET        963_D1010splice    7
   1247 1247  PIK3R1        X475_splice    4
   1257 1257  PIK3R1        X582_splice    1
```

```
In [127]: data[data['Variation'].str.contains('fs')] == True].head()
```

```
Out[127]:      ID  Gene          Variation  Class
     72    72  RAD50        L234fs      1
    113   113  MSH6        F1088Lfs*5    4
    114   114  MSH6        F1088Sfs*2    4
    120   120  PBRM1        N1333Gfs*      4
    808   808  ERCC2        S746fs      1
```

so found some types in the data as below.

```
In [74]: dict_var = { 'truncation' :['truncating mutations','trunc'], 'delins' : ['insertions/insertions','deletions/deletion'], 'fusion': ['fusions','fusion','fus','fs*','fs'], 'deletion' : ['deletions/deletion'], 'insertions': ['insertions','insertion','ins'], 'amplification': ['amplification'], 'splice': ['splice'], 'duplication' : ['duplications','duplication','dup'] }
```

```
In [75]: def variation_transform(var):
    var = var.lower()
    words = var.split(' ')
    pattern = re.match(re.compile(r'^([a-zA-Z]*) (\d{1,7}) ([a-zA-Z]*)$',),words[0])
    final_out = {'amino_before':'none','amino_after':'none','location_no':0,'operation':None}
    if len(words)==1 and pattern :
        final_out['operation'] = 'substitution'
        if pattern.group(1) != '*':
            final_out['amino_before'] = pattern.group(1)
        if pattern.group(3) != '*':
            final_out['amino_after'] = pattern.group(3)
        final_out['location_no'] = pattern.group(2)
    else:
        temp_txt = var
        for key,val in dict_var.items():
            for i in val:
                if i in var.lower():
                    final_out['operation'] = key
                    temp_txt = var.replace(i,' ')
                    break
                if final_out['operation'] != 'other':
                    break
            if not temp_txt.isspace():
                for x in ['mutations','_','-']:
                    temp_txt = temp_txt.replace(x,' ')
                if not temp_txt.isspace():
                    temp_txt = temp_txt.strip()
                    temp_words = temp_txt.split(' ')
                    if len(temp_words) != 1 :
                        if len(temp_words[-1]) < 3:
                            del temp_words[-1]
                    temp_txt = ' '.join(temp_words)
                    final_out['text_remaining'] = temp_txt
    return final_out
```

```
In [76]: variation_train = pd.DataFrame.from_dict(list(train_df.Variation.apply(variation_transform)))
variation_test = pd.DataFrame.from_dict(list(test_df.Variation.apply(variation_transform)))
variation_cv = pd.DataFrame.from_dict(list(cv_df.Variation.apply(variation_transform)))
```

```
In [77]: variation_train.head()
```

```
Out[77]:   amino_after amino_before location_no      operation text_remaining
          0             y                 n           1819  substitution         none
```

1	h	y	68	substitution	none
2	k	e	70	substitution	none
3	s	p	1139	substitution	none
4	none	r	2505	substitution	none

```
In [78]: var_vec = CountVectorizer(token_pattern=r"(?u)\b\w+\b", binary=True)
train_amino_before_onehotcoding = var_vec.fit_transform(variation_train.amino_before)
test_amino_before_onehotcoding = var_vec.transform(variation_test.amino_before)
cv_amino_before_onehotcoding = var_vec.transform(variation_cv.amino_before)

In [79]: var_vec = CountVectorizer(token_pattern=r"(?u)\b\w+\b", binary=True)
train_amino_after_onehotcoding = var_vec.fit_transform(variation_train.amino_after)
test_amino_after_onehotcoding = var_vec.transform(variation_test.amino_after)
cv_amino_after_onehotcoding = var_vec.transform(variation_cv.amino_after)

In [80]: variation_train['Score'] = y_train

In [81]: from sklearn.preprocessing import MinMaxScaler
var_minmax = MinMaxScaler()
train_amino_location = var_minmax.fit_transform(variation_train.location_no.astype(int))
test_amino_location = var_minmax.transform(variation_test.location_no.astype(int).reshape(-1,1))
cv_amino_location = var_minmax.transform(variation_cv.location_no.astype(int).reshape(-1,1))

In [82]: var_vec = CountVectorizer(binary=True)
train_operation_onehotcoding = var_vec.fit_transform(variation_train.operation)
test_operation_onehotcoding = var_vec.transform(variation_test.operation)
cv_operation_onehotcoding = var_vec.transform(variation_cv.operation)

In [136]: var_vec_opr = TfidfVectorizer()
train_operation_tfidf = var_vec_opr.fit_transform(variation_train.operation)
test_operation_tfidf = var_vec_opr.transform(variation_test.operation)
cv_operation_tfidf = var_vec_opr.transform(variation_cv.operation)

In [84]: var_vec = TfidfVectorizer()
train_remaining_tfidf = var_vec.fit_transform(variation_train.text_remaining)
test_remaining_tfidf = var_vec.transform(variation_test.text_remaining)
cv_remaining_tfidf = var_vec.transform(variation_cv.text_remaining)
```

### Stacking Features:

```
In [94]: train_x_mix = hstack((train_gene_feature_tfidfCoding,train_amino_before_onehotcoding,
                           train_amino_location,train_operation_tfidf,train_remaining_tfidf,train_text_tfidf))
test_x_mix = hstack((test_gene_feature_tfidfCoding,test_amino_before_onehotcoding,test_amino_location,
                     test_operation_tfidf,test_remaining_tfidf,test_text_tfidf))
cv_x_mix = hstack((cv_gene_feature_tfidfCoding, cv_amino_before_onehotcoding, cv_amino_location,
                   cv_operation_tfidf, cv_remaining_tfidf, cv_text_tfidf))
```

## Machine Learning Models

```
In [96]: alpha = [10 ** x for x in range(-6, 3)]
#alpha = np.random.uniform(0.00005,0.0005,20)
#alpha = np.round(alpha,7)
#alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', r
    clf.fit(train_x_mix, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_mix, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_mix)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=
    # to avoid rounding error while multiplying probabilités we use log-probability e
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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

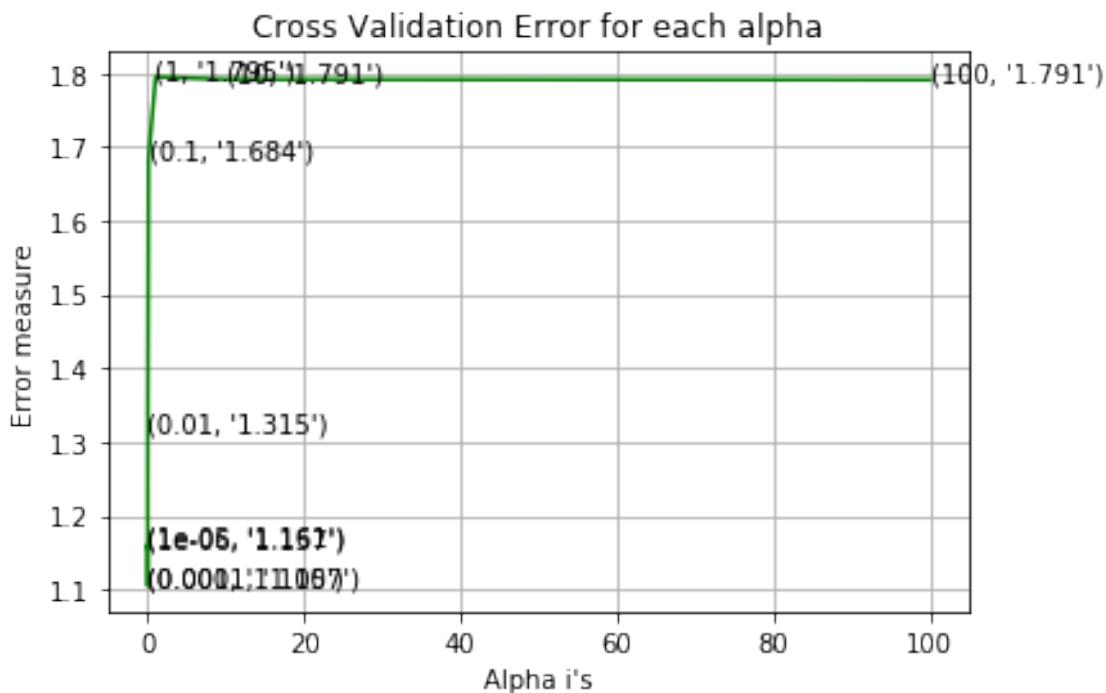
predict_y = sig_clf.predict_proba(train_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss"
predict_y = sig_clf.predict_proba(test_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_

for alpha = 1e-06
Log Loss : 1.1572679671689152
for alpha = 1e-05
Log Loss : 1.1610483754032719
for alpha = 0.0001
Log Loss : 1.106569846257389
for alpha = 0.001
```

```

Log Loss : 1.104524998484568
for alpha = 0.01
Log Loss : 1.3154045376851977
for alpha = 0.1
Log Loss : 1.6836712841622923
for alpha = 1
Log Loss : 1.79469180612326
for alpha = 10
Log Loss : 1.7914570224459718
for alpha = 100
Log Loss : 1.790967929744358

```



```

For values of best alpha = 0.001 The train log loss is: 0.8153879888893428
For values of best alpha = 0.001 The cross validation log loss is: 1.104524998484568
For values of best alpha = 0.001 The test log loss is: 1.0582360008368255

```

```

In [98]: train_x_mix = hstack((train_gene_feature_tfidfCoding,train_amino_before_onehotcoding,
                           train_operation_tfidf,train_remaining_tfidf,train_text_feature_tfidf))
test_x_mix = hstack((test_gene_feature_tfidfCoding,test_amino_before_onehotcoding,test_operation_tfidf,
                     test_remaining_tfidf,test_text_feature_tfidf))
cv_x_mix = hstack((cv_gene_feature_tfidfCoding,cv_amino_before_onehotcoding,cv_amino_after_onehotcoding,
                   cv_operation_tfidf,cv_remaining_tfidf,cv_text_feature_tfidfCoding))

```

```

In [99]: alpha = [10 ** x for x in range(-6, 3)]
#alpha = np.random.uniform(0.00005,0.0005,20)
#alpha = np.round(alpha,7)
#alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_mix, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_mix, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_mix)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability encoding
    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', loss='log', random_state=42)
clf.fit(train_x_mix, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_mix, train_y)

predict_y = sig_clf.predict_proba(train_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(cv_y, predict_y))
predict_y = sig_clf.predict_proba(cv_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(cv_y, predict_y))
predict_y = sig_clf.predict_proba(test_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(test_y, predict_y))

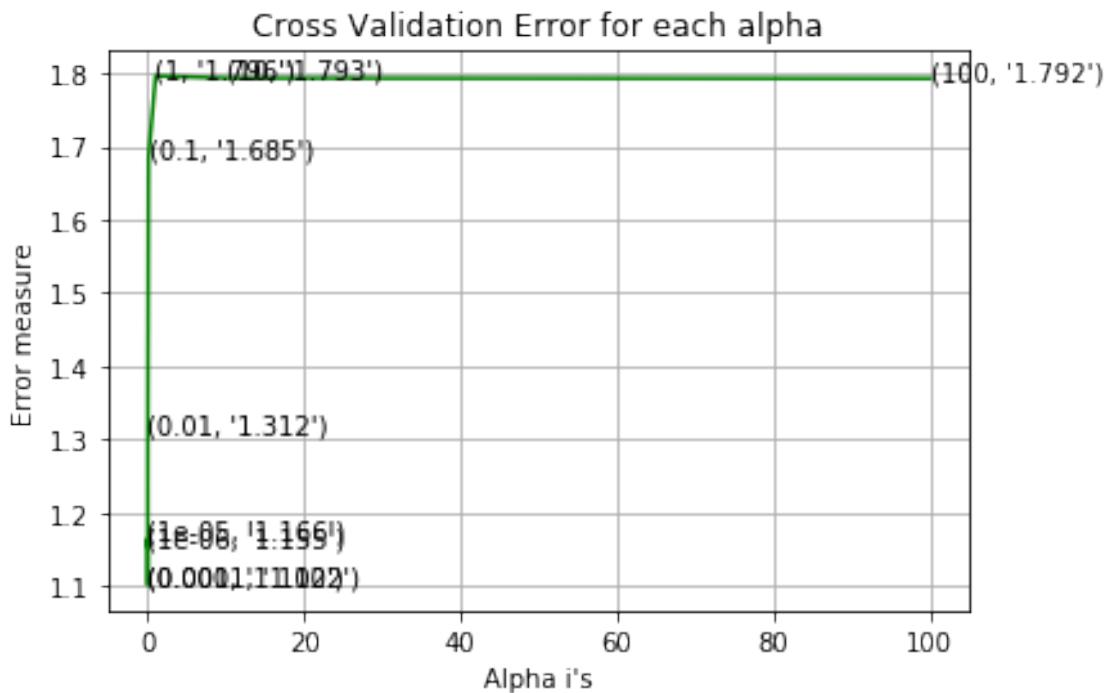
for alpha = 1e-06
Log Loss : 1.1552529266608342
for alpha = 1e-05
Log Loss : 1.165656297453131
for alpha = 0.0001
Log Loss : 1.101948116645287
for alpha = 0.001
Log Loss : 1.1020601268677532

```

```

for alpha = 0.01
Log Loss : 1.311587506629397
for alpha = 0.1
Log Loss : 1.6845419005400664
for alpha = 1
Log Loss : 1.795652910372611
for alpha = 10
Log Loss : 1.7928080609941783
for alpha = 100
Log Loss : 1.7923721408648678

```



```

For values of best alpha = 0.0001 The train log loss is: 0.7694724997064499
For values of best alpha = 0.0001 The cross validation log loss is: 1.101948116645287
For values of best alpha = 0.0001 The test log loss is: 1.064722377733783

```

```

In [100]: train_x_mix = hstack((train_gene_feature_tfidfCoding,train_amino_location,
                           train_gene_feature_avgw2v,train_operation_tfidf,train_text_feature_tfidf))
test_x_mix = hstack((test_gene_feature_tfidfCoding,test_amino_location,
                     test_gene_feature_avgw2v,test_operation_tfidf,test_text_feature_tfidf))
cv_x_mix = hstack((cv_gene_feature_tfidfCoding, cv_amino_location,
                   cv_gene_feature_avgw2v, cv_operation_tfidf, cv_text_feature_tfidf))

In [101]: alpha = [10 ** x for x in range(-6, 3)]
#alpha = np.random.uniform(0.00005, 0.0005, 20)

```

```

#alpha = np.round(alpha,7)
#alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', max_iter=1000)
    clf.fit(train_x_mix, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_mix, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_mix)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', max_iter=1000)
clf.fit(train_x_mix, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_mix, train_y)

predict_y = sig_clf.predict_proba(train_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(cv_y, predict_y))
predict_y = sig_clf.predict_proba(cv_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(cv_y, predict_y))
predict_y = sig_clf.predict_proba(test_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(test_y, predict_y))

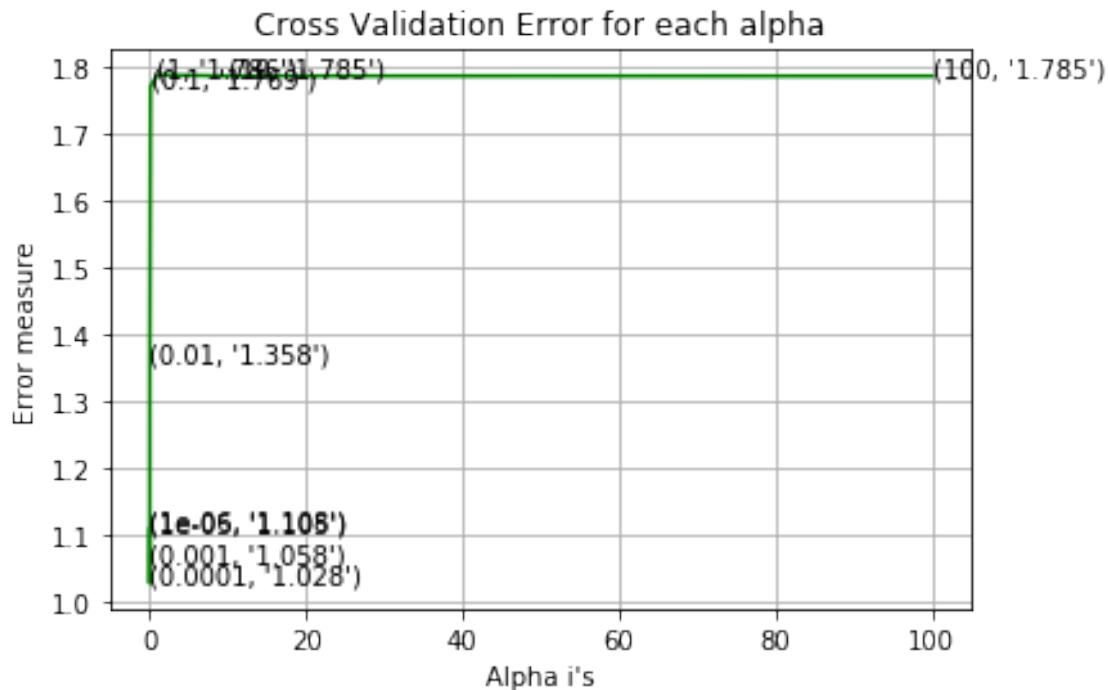
for alpha = 1e-06
Log Loss : 1.1080445309787814
for alpha = 1e-05
Log Loss : 1.1050611097403344
for alpha = 0.0001
Log Loss : 1.0275949490270742
for alpha = 0.001
Log Loss : 1.0576565879188808
for alpha = 0.01
Log Loss : 1.3579693032200653

```

```

for alpha = 0.1
Log Loss : 1.76891926027938
for alpha = 1
Log Loss : 1.7862657387792211
for alpha = 10
Log Loss : 1.7847453914264644
for alpha = 100
Log Loss : 1.784521282510309

```



```

For values of best alpha = 0.0001 The train log loss is: 0.7321143880223072
For values of best alpha = 0.0001 The cross validation log loss is: 1.0275949490270742
For values of best alpha = 0.0001 The test log loss is: 1.0125731245075953

```

```

In [102]: alpha = [10 ** x for x in range(-6, 3)]
alpha = np.random.uniform(0.0005, 0.005, 20)
alpha = np.round(alpha, 6)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log',
    clf.fit(train_x_mix, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")

```

```

        sig_clf.fit(train_x_mix, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_mix)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-7))
        # to avoid rounding error while multiplying probabilites we use log-probability
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

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

predict_y = sig_clf.predict_proba(train_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(train_y,predict_y))
predict_y = sig_clf.predict_proba(cv_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(cv_y,predict_y))
predict_y = sig_clf.predict_proba(test_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(test_y,predict_y))

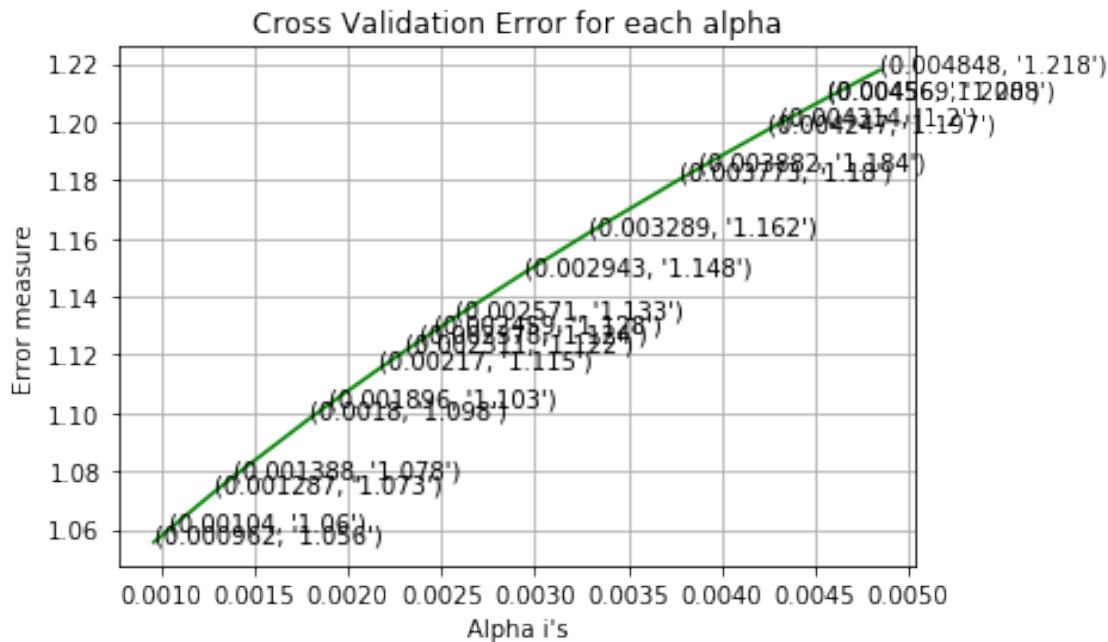
for alpha = 0.000962
Log Loss : 1.0555731957389414
for alpha = 0.00104
Log Loss : 1.0598335309981797
for alpha = 0.001287
Log Loss : 1.0729319868004528
for alpha = 0.001388
Log Loss : 1.0781179816585353
for alpha = 0.0018
Log Loss : 1.0983105207120722
for alpha = 0.001896
Log Loss : 1.1028135147165894
for alpha = 0.00217
Log Loss : 1.1152997118462584
for alpha = 0.002311
Log Loss : 1.121532098746538
for alpha = 0.002378
Log Loss : 1.1244512553294899

```

```

for alpha = 0.002459
Log Loss : 1.127945592241774
for alpha = 0.002571
Log Loss : 1.1327168797795224
for alpha = 0.002943
Log Loss : 1.1480975197431567
for alpha = 0.003289
Log Loss : 1.1618165562266625
for alpha = 0.003773
Log Loss : 1.1801574605775806
for alpha = 0.003882
Log Loss : 1.1841607477352032
for alpha = 0.004247
Log Loss : 1.1972474552177357
for alpha = 0.004314
Log Loss : 1.1995981997455478
for alpha = 0.00456
Log Loss : 1.2080982048747775
for alpha = 0.004569
Log Loss : 1.2084053531832788
for alpha = 0.004848
Log Loss : 1.217797407361465

```



For values of best alpha = 0.000962 The train log loss is: 0.8413680281122422  
 For values of best alpha = 0.000962 The cross validation log loss is: 1.0555731957389414

```
For values of best alpha = 0.000962 The test log loss is: 1.0273949675019105
```

```
In [103]: alpha = [10 ** x for x in range(-6, 3)]
alpha = np.random.uniform(0.0001, 0.0006, 10)
alpha = np.round(alpha, 6)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', max_iter=1000)
    clf.fit(train_x_mix, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_mix, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_mix)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', max_iter=1000)
clf.fit(train_x_mix, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_mix, train_y)

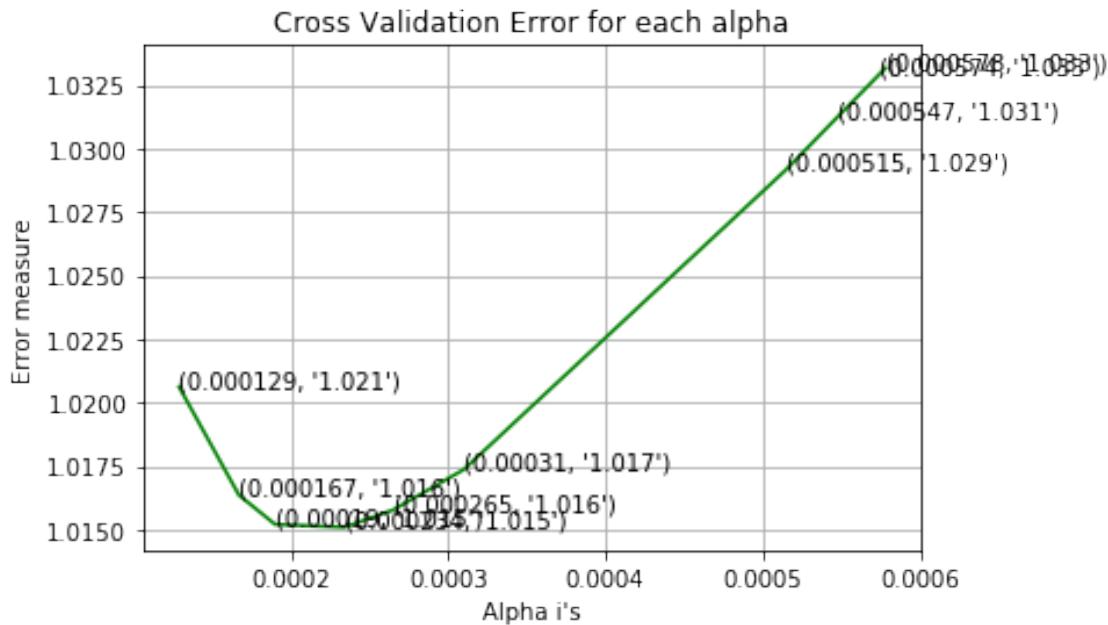
predict_y = sig_clf.predict_proba(train_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(predict_y, train_y))
predict_y = sig_clf.predict_proba(cv_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(predict_y, cv_y))
predict_y = sig_clf.predict_proba(test_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(predict_y, test_y))

for alpha = 0.000129
Log Loss : 1.0206145953991124
for alpha = 0.000167
Log Loss : 1.0163558403889612
for alpha = 0.00019
```

```

Log Loss : 1.0151995171247383
for alpha = 0.000234
Log Loss : 1.0150846947511012
for alpha = 0.000265
Log Loss : 1.0157648301601483
for alpha = 0.00031
Log Loss : 1.0173589423149747
for alpha = 0.000515
Log Loss : 1.029175820672752
for alpha = 0.000547
Log Loss : 1.0312096143425993
for alpha = 0.000574
Log Loss : 1.03291486364564
for alpha = 0.000578
Log Loss : 1.0331663005870153

```



For values of best alpha = 0.000234 The train log loss is: 0.7355025088235884  
 For values of best alpha = 0.000234 The cross validation log loss is: 1.0150846947511012  
 For values of best alpha = 0.000234 The test log loss is: 0.9953564559160516

```
In [104]: clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', max_iter=1000)
sig_clf,temp = predict_and_plot_confusion_matrix(train_x_mix, train_y, cv_x_mix, cv_y)
list_data = []
list_data.append('Tf_Idf+some_var_transform+LR+Class_Balance')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_mix), eps=1e-15))
```

```

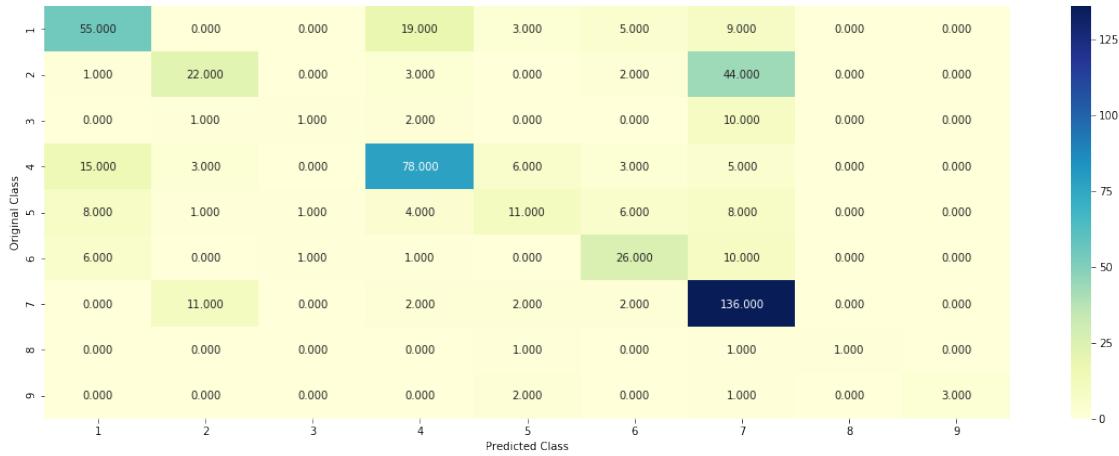
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_mix), eps=1e-15))
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_mix), eps=1e-15))
list_data.append('alpha = '+str(clf.alpha))
list_data.append(temp)
final_results.append(list_data)

```

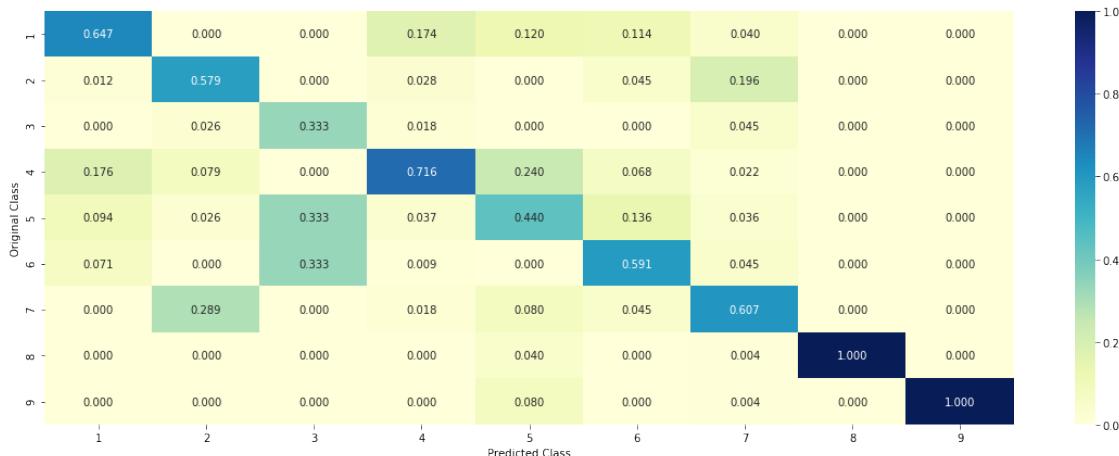
Log loss : 1.0150846947511012

Number of mis-classified points : 0.37406015037593987

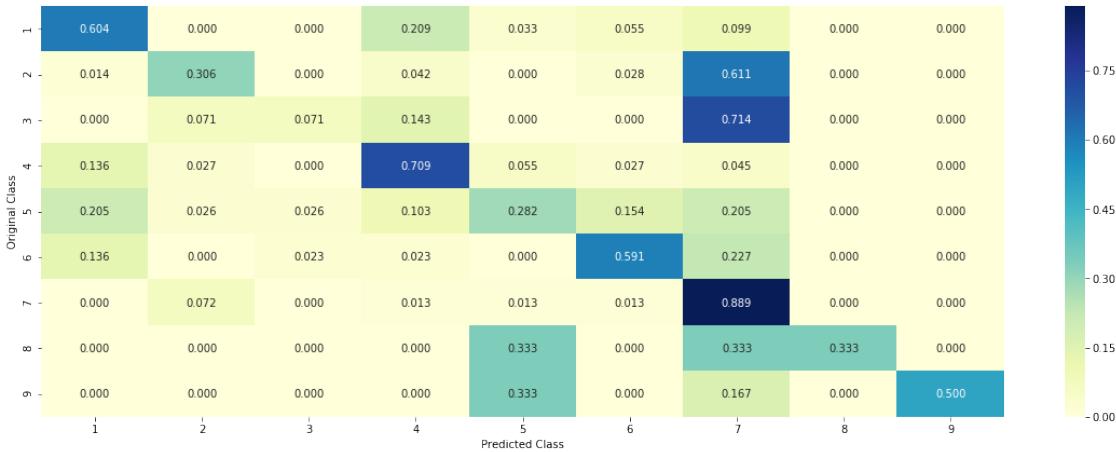
----- Confusion matrix -----



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



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



## SVM

```
In [105]: alpha = [10 ** x for x in range(-6, 3)]
#alpha = np.random.uniform(0.0005,0.005,20)
#alpha = np.round(alpha,6)
#alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='hinge')
    clf.fit(train_x_mix, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_mix, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_mix)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

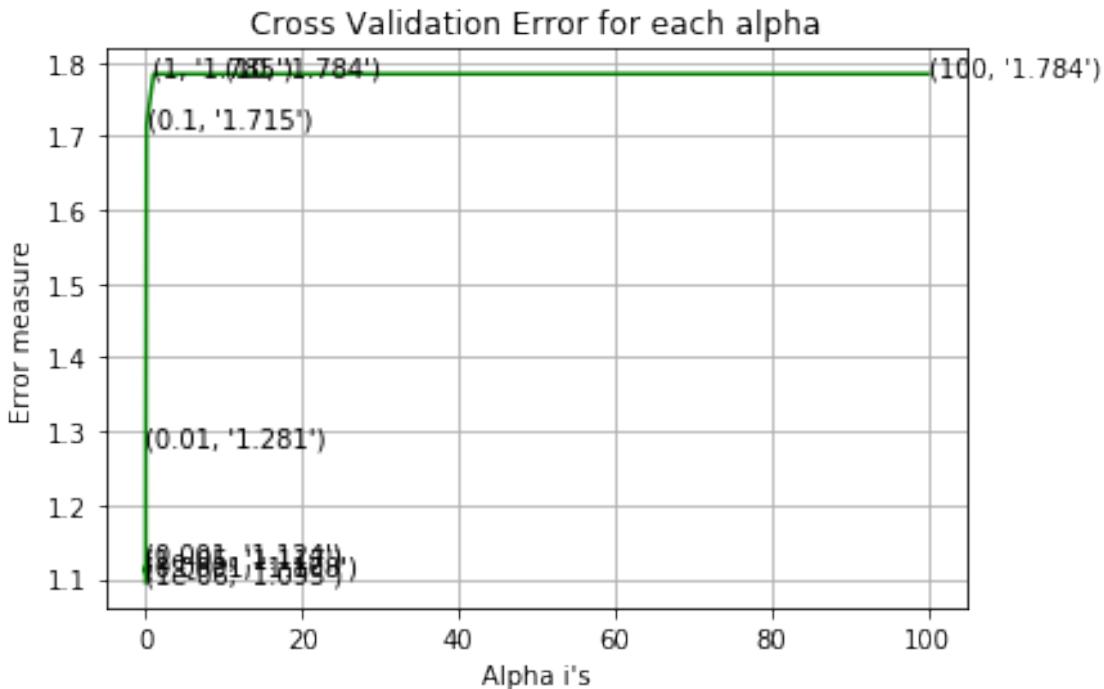
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")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

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge')
```

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

predict_y = sig_clf.predict_proba(train_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss"
predict_y = sig_clf.predict_proba(test_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_]

for alpha = 1e-06
Log Loss : 1.0951331204791235
for alpha = 1e-05
Log Loss : 1.1169038559448836
for alpha = 0.0001
Log Loss : 1.1078747888858307
for alpha = 0.001
Log Loss : 1.1244054402972607
for alpha = 0.01
Log Loss : 1.2810298081828306
for alpha = 0.1
Log Loss : 1.7150246556914535
for alpha = 1
Log Loss : 1.7845086161052728
for alpha = 10
Log Loss : 1.78448976314774
for alpha = 100
Log Loss : 1.7844897548208094
```



```
For values of best alpha = 1e-06 The train log loss is: 0.842376438063761
For values of best alpha = 1e-06 The cross validation log loss is: 1.0951331204791235
For values of best alpha = 1e-06 The test log loss is: 1.1036266770497034
```

**Logistic Regression** With var transform and added additional colum of variance response coding and gene avgwordvector from pubmed data.

```
In [106]: train_x_mix = hstack((train_amino_location,train_variation_feature_responseCoding,
                           train_gene_feature_avgw2v,train_operation_tfidf,train_text_feature_
test_x_mix = hstack((test_amino_location,test_variation_feature_responseCoding,
                     test_gene_feature_avgw2v,test_operation_tfidf,test_text_feature_
cv_x_mix = hstack((cv_amino_location, cv_variation_feature_responseCoding,
                   cv_gene_feature_avgw2v, cv_operation_tfidf, cv_text_feature_tfidfC

In [107]: alpha = [10 ** x for x in range(-6, 3)]
#alpha = np.random.uniform(0.0005,0.005,20)
#alpha = np.round(alpha,6)
#alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', n_iter_no_change=100)
    #clf = RandomForestClassifier(n_estimators=i, max_depth=20, n_jobs=-1)
```

```

clf.fit(train_x_mix, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_mix, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_mix)
cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
# to avoid rounding error while multiplying probabilites we use log-probability
print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', max_iter=1000)
#clf = RandomForestClassifier(n_estimators=alpha[best_alpha],max_depth=20,n_jobs=-1)
clf.fit(train_x_mix, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_mix, train_y)

predict_y = sig_clf.predict_proba(train_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(train_y,predict_y))
predict_y = sig_clf.predict_proba(cv_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(cv_y,predict_y))
predict_y = sig_clf.predict_proba(test_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(test_y,predict_y))

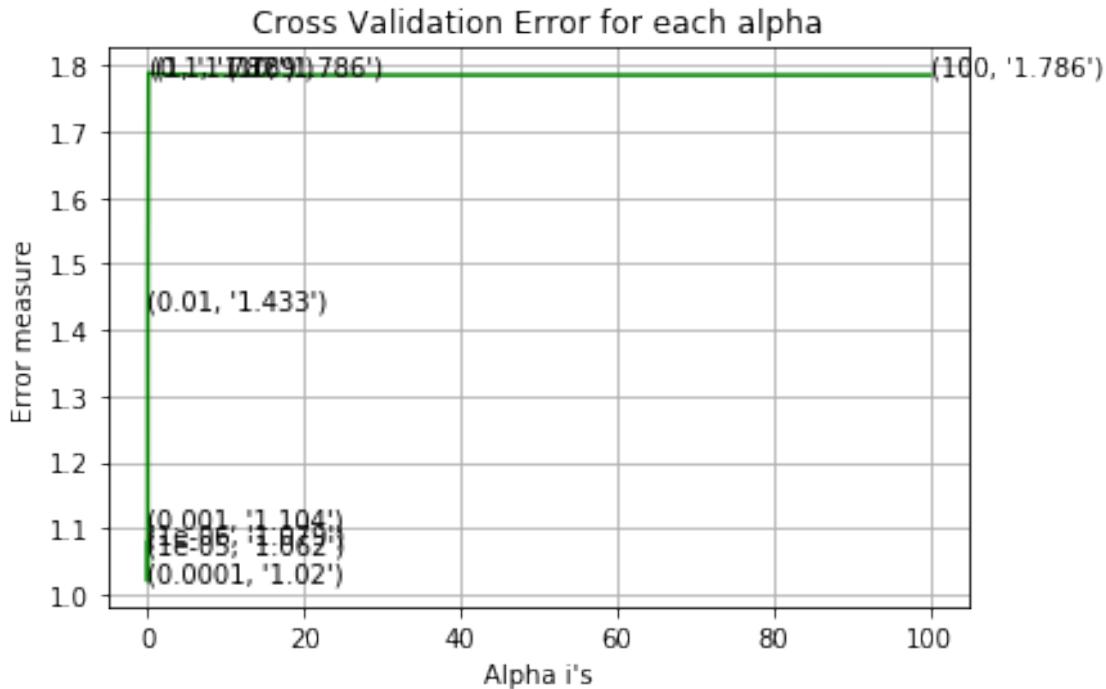
for alpha = 1e-06
Log Loss : 1.0789786892806823
for alpha = 1e-05
Log Loss : 1.0622995540604239
for alpha = 0.0001
Log Loss : 1.019670085842367
for alpha = 0.001
Log Loss : 1.1036029919079786
for alpha = 0.01
Log Loss : 1.4328854202361305
for alpha = 0.1
Log Loss : 1.7889199460157772
for alpha = 1
Log Loss : 1.7872814310437921
for alpha = 10

```

```

Log Loss : 1.785728326404347
for alpha = 100
Log Loss : 1.785537209906527

```



```

For values of best alpha = 0.0001 The train log loss is: 0.6828045441533561
For values of best alpha = 0.0001 The cross validation log loss is: 1.019670085842367
For values of best alpha = 0.0001 The test log loss is: 1.004530080577885

```

with tfidf of values and amino location+addition of avgwordvector of gene

```

In [108]: train_x_mix = hstack((train_gene_feature_tfidfCoding,train_amino_location,train_variation
                           train_gene_feature_avgw2v,train_operation_tfidf,train_text_feature_tfidf)
                           test_x_mix = hstack((test_gene_feature_tfidfCoding,test_amino_location,test_variation
                           test_gene_feature_avgw2v,test_operation_tfidf,test_text_feature_tfidf)
                           cv_x_mix = hstack((cv_gene_feature_tfidfCoding, cv_amino_location, cv_variation_feature_tfidf
                           cv_gene_feature_avgw2v, cv_operation_tfidf, cv_text_feature_tfidf))

```

```

In [109]: alpha = [10 ** x for x in range(-6, 3)]
          #alpha = np.random.uniform(0.00005, 0.0005, 15)
          #alpha = np.round(alpha, 6)
          #alpha.sort()
          cv_log_error_array = []
          for i in alpha:

```

```

print("for alpha =", i)
clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', )
#clf = RandomForestClassifier(n_estimators=i, max_depth=20, n_jobs=-1)
clf.fit(train_x_mix, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_mix, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_mix)
cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-7))
# to avoid rounding error while multiplying probabilites we use log-probability
print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log')
#clf = RandomForestClassifier(n_estimators=alpha[best_alpha], max_depth=20, n_jobs=-1)
clf.fit(train_x_mix, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_mix, train_y)

predict_y = sig_clf.predict_proba(train_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(train_y,predict_y))
predict_y = sig_clf.predict_proba(cv_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(cv_y,predict_y))
predict_y = sig_clf.predict_proba(test_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(test_y,predict_y))

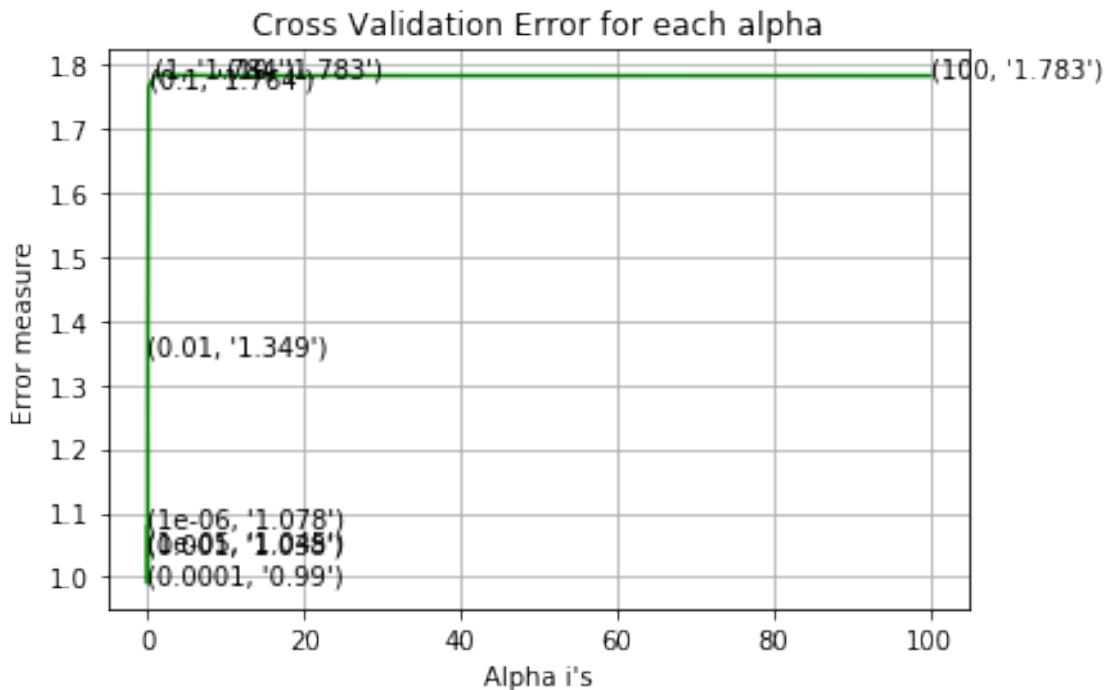
for alpha = 1e-06
Log Loss : 1.0783454048785017
for alpha = 1e-05
Log Loss : 1.045282616532428
for alpha = 0.0001
Log Loss : 0.9900830455402387
for alpha = 0.001
Log Loss : 1.0380823113055675
for alpha = 0.01
Log Loss : 1.3488719221178533
for alpha = 0.1
Log Loss : 1.7644080836648688

```

```

for alpha = 1
Log Loss : 1.783941007290487
for alpha = 10
Log Loss : 1.782799564353334
for alpha = 100
Log Loss : 1.782613616612465

```



For values of best alpha = 0.0001 The train log loss is: 0.451051440732323  
 For values of best alpha = 0.0001 The cross validation log loss is: 0.9900830455402387  
 For values of best alpha = 0.0001 The test log loss is: 0.9660451286904723

```
In [110]: alpha = [10 ** x for x in range(-6, 3)]
alpha = np.random.uniform(0.00005, 0.0005, 15)
alpha = np.round(alpha, 7)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', max_iter=1000)
    #clf = RandomForestClassifier(n_estimators=i, max_depth=20, n_jobs=-1)
    clf.fit(train_x_mix, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_mix, train_y)
```

```

sig_clf_probs = sig_clf.predict_proba(cv_x_mix)
cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-7))
# to avoid rounding error while multiplying probabilites we use log-probability
print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', max_iter=1000)
#clf = RandomForestClassifier(n_estimators=alpha[best_alpha],max_depth=20,n_jobs=-1)
clf.fit(train_x_mix, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_mix, train_y)

predict_y = sig_clf.predict_proba(train_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(train_y,predict_y))
predict_y = sig_clf.predict_proba(cv_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(cv_y,predict_y))
predict_y = sig_clf.predict_proba(test_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(test_y,predict_y))

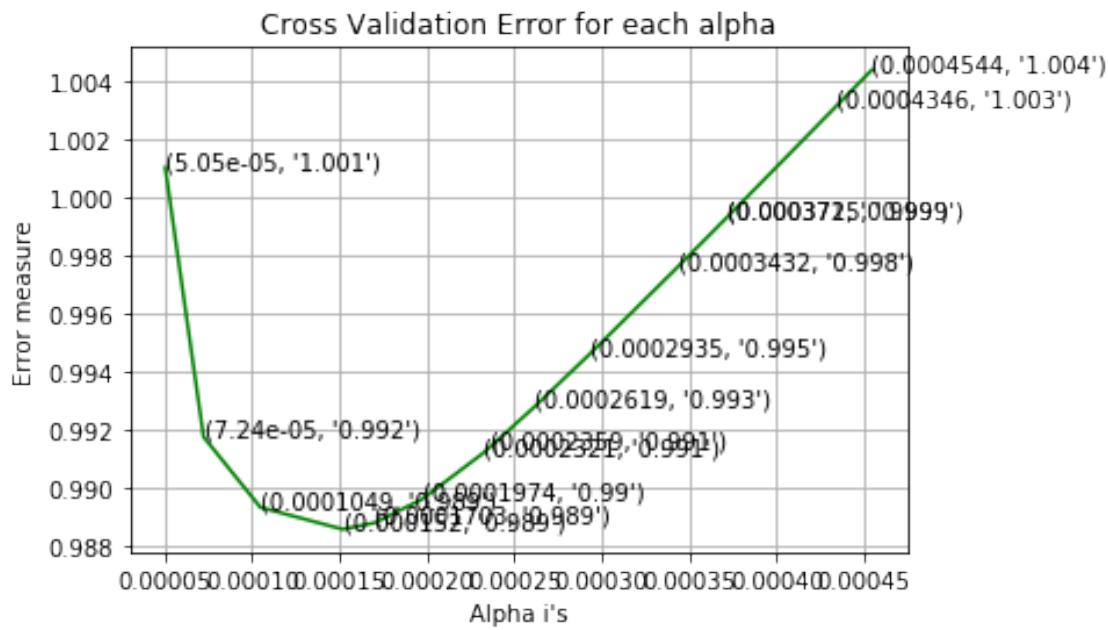
for alpha = 5.05e-05
Log Loss : 1.0010131769276331
for alpha = 7.24e-05
Log Loss : 0.9917421316320164
for alpha = 0.0001049
Log Loss : 0.9893139539555773
for alpha = 0.000152
Log Loss : 0.9885610968052642
for alpha = 0.0001703
Log Loss : 0.9887849630220442
for alpha = 0.0001974
Log Loss : 0.9895982662483537
for alpha = 0.0002321
Log Loss : 0.9911770011774639
for alpha = 0.0002359
Log Loss : 0.9913753720394258
for alpha = 0.0002619
Log Loss : 0.9927951344813574

```

```

for alpha = 0.0002935
Log Loss : 0.9946142660508183
for alpha = 0.0003432
Log Loss : 0.9975759839202812
for alpha = 0.0003715
Log Loss : 0.999293340019823
for alpha = 0.000372
Log Loss : 0.9993238203783651
for alpha = 0.0004346
Log Loss : 1.0031602794149665
for alpha = 0.0004544
Log Loss : 1.004374901832886

```



For values of best alpha = 0.000152 The train log loss is: 0.467970069026504  
 For values of best alpha = 0.000152 The cross validation log loss is: 0.9885610968052642  
 For values of best alpha = 0.000152 The test log loss is: 0.9631869525334625

```
In [111]: clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', max_iter=1000)
sig_clf,temp = predict_and_plot_confusion_matrix(train_x_mix, train_y, cv_x_mix, cv_y)
list_data = []
list_data.append('Tf_Idf+gene_agvw2v+alloc+var_transform+LR')
list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_mix), eps=1e-15))
list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_mix), eps=1e-15))
list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_mix), eps=1e-15))
list_data.append('alpha = '+str(clf.alpha))
```

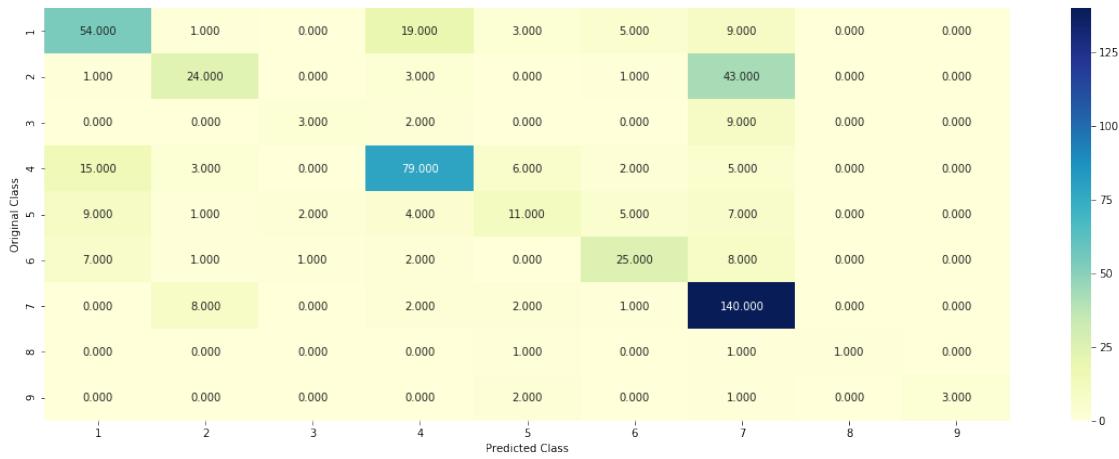
```

        list_data.append(temp)
        final_results.append(list_data)
    
```

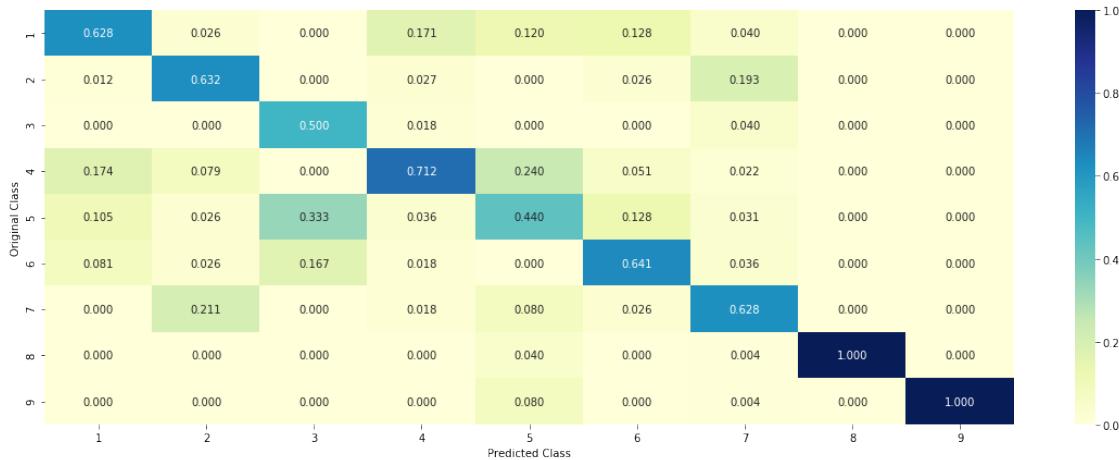
Log loss : 0.9885610968052642

Number of mis-classified points : 0.3609022556390977

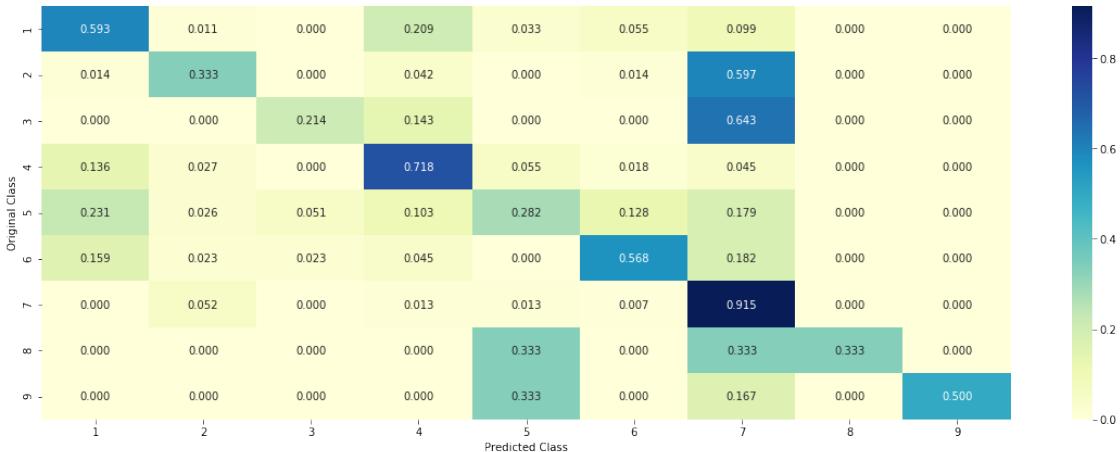
----- Confusion matrix -----



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



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



```
In [97]: train_x_mix = hstack((train_gene_feature_tfidfCoding,train_variation_feature_tfidfCoding,
                           train_gene_feature_avgw2v,train_operation_tfidf,train_text_feature_tfidfCoding))
test_x_mix = hstack((test_gene_feature_tfidfCoding,test_variation_feature_tfidfCoding,
                     test_gene_feature_avgw2v,test_operation_tfidf,test_text_feature_tfidfCoding))
cv_x_mix = hstack((cv_gene_feature_tfidfCoding, cv_variation_feature_tfidfCoding,
                   cv_gene_feature_avgw2v, cv_operation_tfidf, cv_text_feature_tfidfCoding))

In [113]: alpha = [10 ** x for x in range(-6, 3)]
          #alpha = np.random.uniform(0.00005,0.0005,15)
          #alpha = np.round(alpha,7)
          #alpha.sort()
          cv_log_error_array = []
          for i in alpha:
              print("for alpha =", i)
              clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log',
#clf = RandomForestClassifier(n_estimators=i,max_depth=20,n_jobs=-1)
              clf.fit(train_x_mix, train_y)
              sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
              sig_clf.fit(train_x_mix, train_y)
              sig_clf_probs = sig_clf.predict_proba(cv_x_mix)
              cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=0.0001))
              # to avoid rounding error while multiplying probabilités we use log-probability
              print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

```

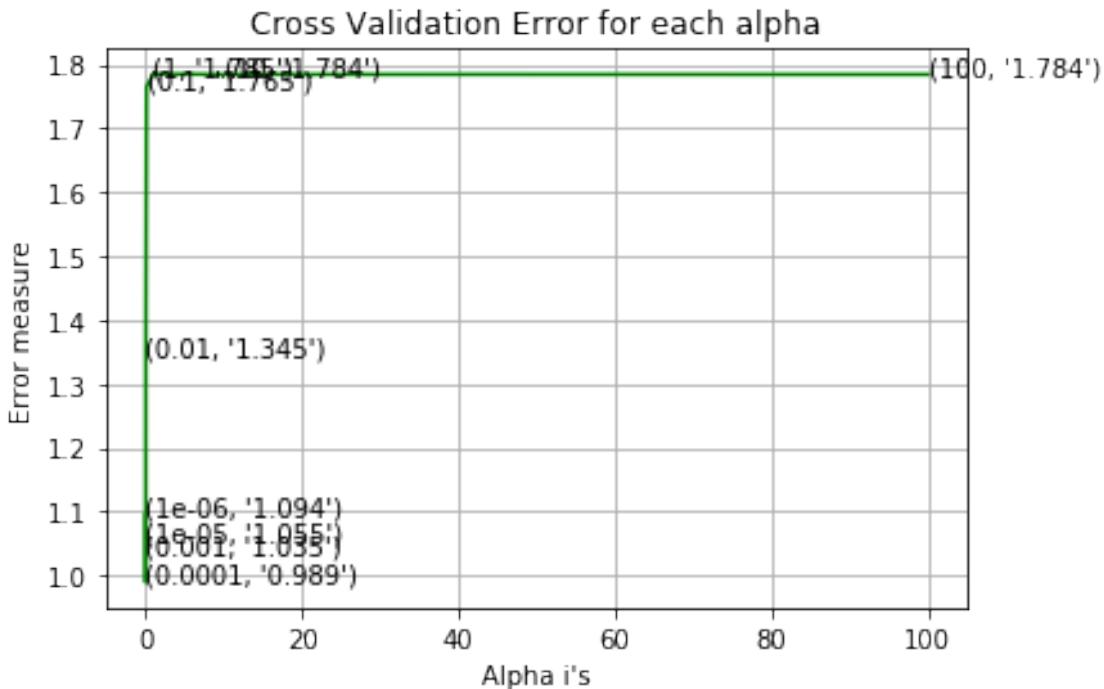
plt.ylabel("Error measure")
plt.show()

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log')
#clf = RandomForestClassifier(n_estimators=alpha[best_alpha], max_depth=20, n_jobs=-1)
clf.fit(train_x_mix, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_mix, train_y)

predict_y = sig_clf.predict_proba(train_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, predict_y))
predict_y = sig_clf.predict_proba(cv_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(test_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, predict_y))

for alpha = 1e-06
Log Loss : 1.0939952885314643
for alpha = 1e-05
Log Loss : 1.0551469676593386
for alpha = 0.0001
Log Loss : 0.9891796355326503
for alpha = 0.001
Log Loss : 1.0348094073343594
for alpha = 0.01
Log Loss : 1.34496797267533
for alpha = 0.1
Log Loss : 1.7646190837677709
for alpha = 1
Log Loss : 1.785112428139833
for alpha = 10
Log Loss : 1.784416545472734
for alpha = 100
Log Loss : 1.784298422390178

```



```
For values of best alpha = 0.0001 The train log loss is: 0.45164930273181725
For values of best alpha = 0.0001 The cross validation log loss is: 0.9891796355326503
For values of best alpha = 0.0001 The test log loss is: 0.962638334802327
```

```
In [114]: alpha = [10 ** x for x in range(-6, 3)]
alpha = np.random.uniform(0.00005, 0.0005, 15)
alpha = np.round(alpha, 7)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', max_iter=1000)
    #clf = RandomForestClassifier(n_estimators=i, max_depth=20, n_jobs=-1)
    clf.fit(train_x_mix, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_mix, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_mix)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
```

```

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

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', l1_ratio=None, max_iter=1000, tol=0.001, shuffle=True, random_state=None, verbose=0, epsilon=0.1, n_jobs=-1, warm_start=False, validate_params=True)
#clf = RandomForestClassifier(n_estimators=alpha[best_alpha], max_depth=20, n_jobs=-1)
clf.fit(train_x_mix, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_mix, train_y)

predict_y = sig_clf.predict_proba(train_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,predict_y))
predict_y = sig_clf.predict_proba(cv_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv,predict_y))
predict_y = sig_clf.predict_proba(test_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test,predict_y))

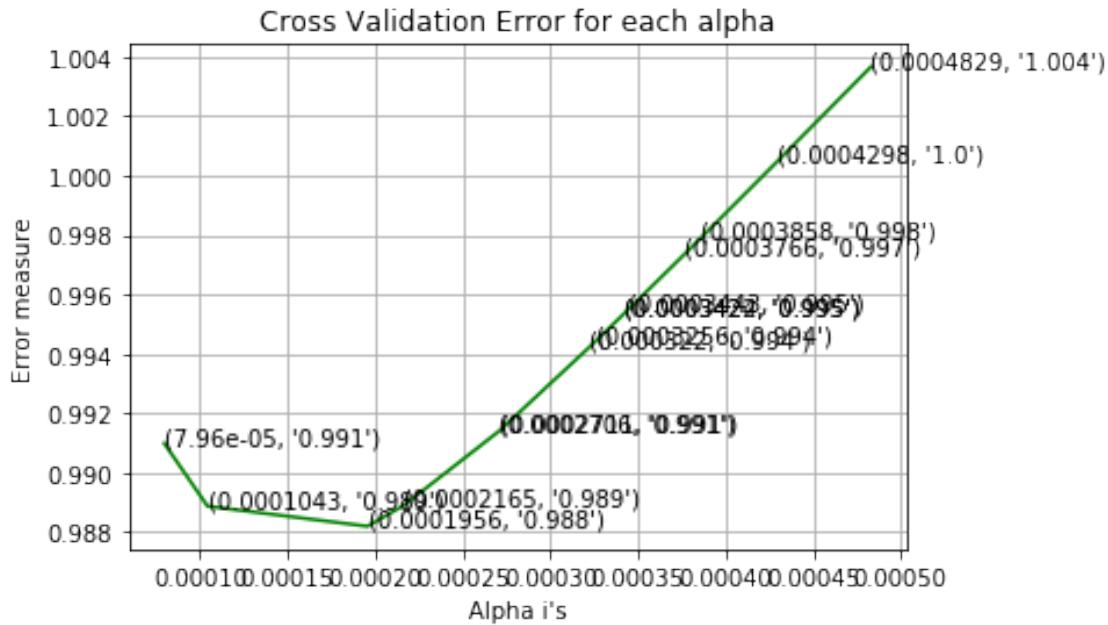
for alpha = 7.96e-05
Log Loss : 0.9909841734807477
for alpha = 0.0001043
Log Loss : 0.9888659096858611
for alpha = 0.0001956
Log Loss : 0.9881899631745082
for alpha = 0.0002165
Log Loss : 0.9888994288192036
for alpha = 0.0002706
Log Loss : 0.9913900229876919
for alpha = 0.0002711
Log Loss : 0.9914162714722965
for alpha = 0.000322
Log Loss : 0.994209026748974
for alpha = 0.0003256
Log Loss : 0.9944129075739225
for alpha = 0.0003422
Log Loss : 0.995360617763009
for alpha = 0.0003424
Log Loss : 0.9953721072724496
for alpha = 0.0003443
Log Loss : 0.9954813378759501
for alpha = 0.0003766
Log Loss : 0.9973572132457053
for alpha = 0.0003858

```

```

Log Loss : 0.9978967388534998
for alpha = 0.0004298
Log Loss : 1.0004970558638862
for alpha = 0.0004829
Log Loss : 1.0036633058534845

```



```

For values of best alpha = 0.0001956 The train log loss is: 0.48220046620586954
For values of best alpha = 0.0001956 The cross validation log loss is: 0.9881899631745082
For values of best alpha = 0.0001956 The test log loss is: 0.9601033528194042

```

```

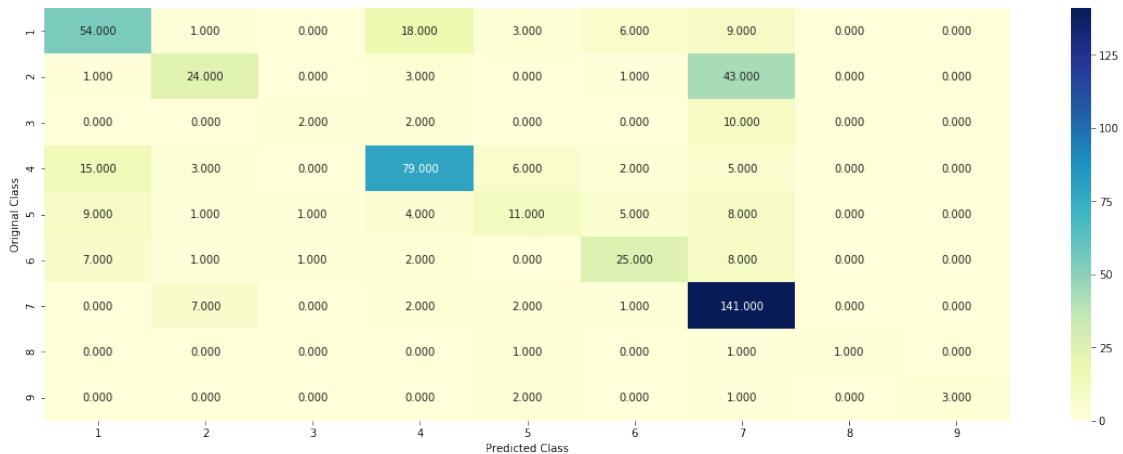
In [115]: clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
                           sig_clf,temp = predict_and_plot_confusion_matrix(train_x_mix, train_y, cv_x_mix, cv_y,
                           list_data = []
                           list_data.append('Tf_Idf+gene_agvw2v+var_transform+LR')
                           list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_mix), eps=1e-15))
                           list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_mix), eps=1e-15))
                           list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_mix), eps=1e-15))
                           list_data.append('alpha = '+str(clf.alpha))
                           list_data.append(temp)
                           final_results.append(list_data)

```

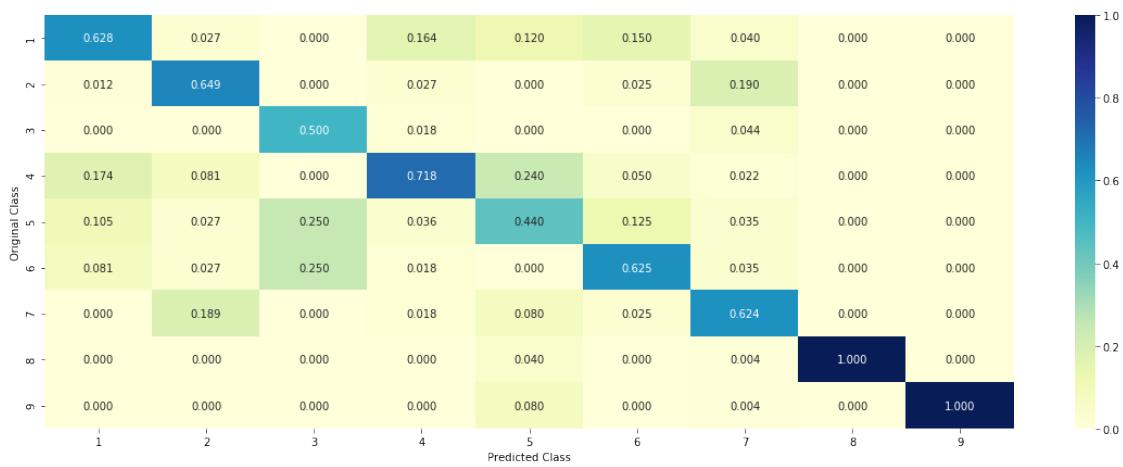
```

Log loss : 0.9881899631745082
Number of mis-classified points : 0.3609022556390977
----- Confusion matrix -----

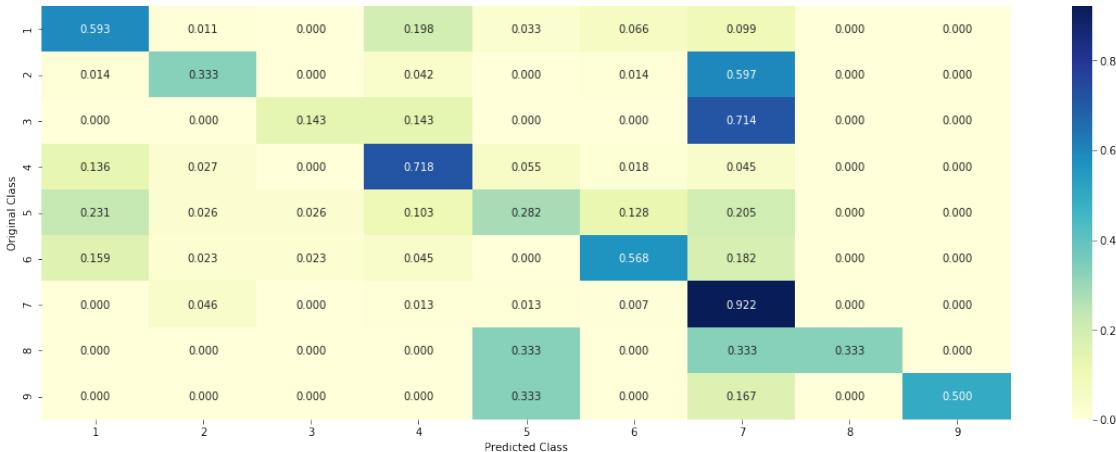
```



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



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



With TF-IDF and variance feature operation mode.

```
In [132]: train_x_mix = hstack((train_gene_feature_tfidfCoding,train_variation_feature_tfidfCoding,
                             train_operation_tfidf,train_text_feature_tfidfCoding12))
test_x_mix = hstack((test_gene_feature_tfidfCoding,test_variation_feature_tfidfCoding,
                     test_operation_tfidf,test_text_feature_tfidfCoding12))
cv_x_mix = hstack((cv_gene_feature_tfidfCoding,cv_variation_feature_tfidfCoding,
                   cv_operation_tfidf,cv_text_feature_tfidfCoding12))

In [117]: alpha = [10 ** x for x in range(-6, 3)]
#alpha = np.random.uniform(0.00005,0.0005,15)
#alpha = np.round(alpha,7)
#alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log',
#clf = RandomForestClassifier(n_estimators=i,max_depth=20,n_jobs=-1)
    clf.fit(train_x_mix, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_mix, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_mix)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

```

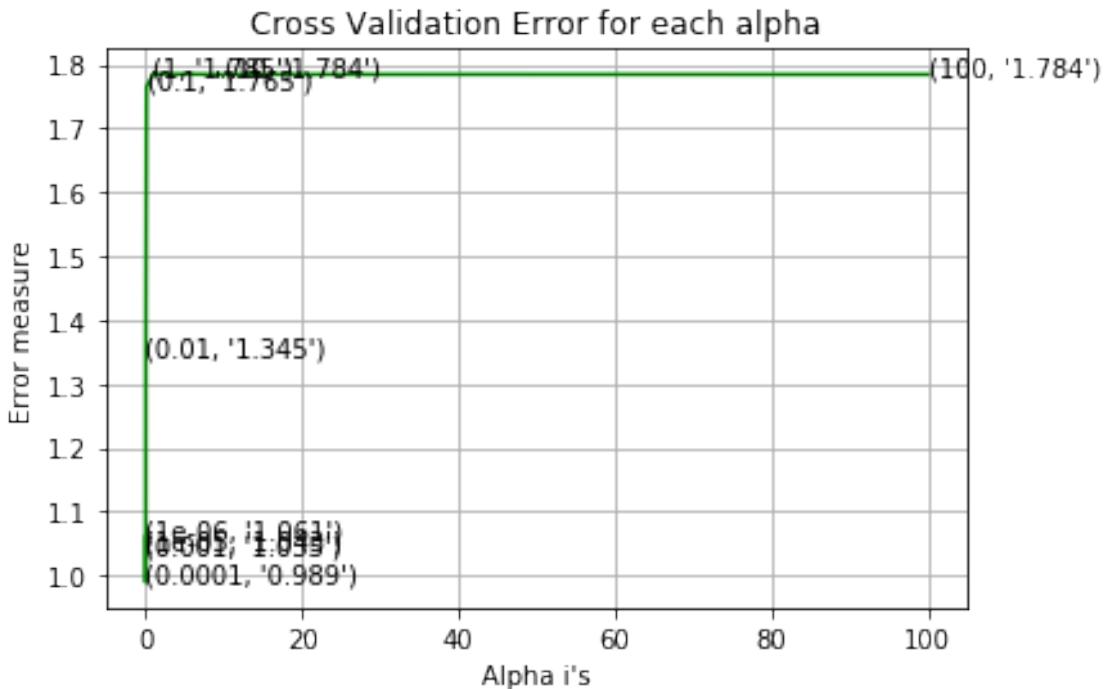
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', l1_ratio=None, max_iter=1000, tol=0.001, loss='log', fit_intercept=True, epsilon=0.1, n_jobs=-1, random_state=None, verbose=0, warm_start=False, average=False)
#clf = RandomForestClassifier(n_estimators=alpha[best_alpha], max_depth=20, n_jobs=-1)
clf.fit(train_x_mix, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_mix, train_y)

predict_y = sig_clf.predict_proba(train_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(train_y, predict_y))
predict_y = sig_clf.predict_proba(cv_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(cv_y, predict_y))
predict_y = sig_clf.predict_proba(test_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(test_y, predict_y))

for alpha = 1e-06
Log Loss : 1.0605283759520376
for alpha = 1e-05
Log Loss : 1.0433775875551214
for alpha = 0.0001
Log Loss : 0.9892444121651003
for alpha = 0.001
Log Loss : 1.0348706380604842
for alpha = 0.01
Log Loss : 1.3450568482987801
for alpha = 0.1
Log Loss : 1.764647590669802
for alpha = 1
Log Loss : 1.785127022559603
for alpha = 10
Log Loss : 1.7844307069871068
for alpha = 100
Log Loss : 1.7843125693731292

```



For values of best alpha = 0.0001 The train log loss is: 0.45165758602284317  
 For values of best alpha = 0.0001 The cross validation log loss is: 0.9892444121651003  
 For values of best alpha = 0.0001 The test log loss is: 0.9624703287647086

```
In [118]: alpha = [10 ** x for x in range(-6, 3)]
#alpha = np.random.uniform(0.00005,0.0005,15)
#alpha = np.round(alpha,7)
#alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', max_iter=1000)
    #clf = LogisticRegression(class_weight='balanced', C=i, solver='liblinear')
    clf.fit(train_x_mix, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_mix, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_mix)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
```

```

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

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', l1_ratio=None, max_iter=1000, tol=1e-05, shuffle=True, verbose=0, epsilon=0.1, random_state=None, n_jobs=1, loss='log', warm_start=False, average=False, multi_class='ovr', fit_intercept=True, verbose=0, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5, class_weight=None, n_iter=None, max_iter=None, tol=None, shuffle=False, verbose=0, epsilon=0.1, random_state=None, n_jobs=1, loss='log', warm_start=False, average=False, multi_class='ovr', fit_intercept=True, verbose=0, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5, class_weight=None, n_iter=None, max_iter=None, tol=None, shuffle=False)
#clf = LogisticRegression(class_weight='balanced', C=alpha[best_alpha], solver='liblinear', max_iter=1000, tol=1e-05, C=alpha[best_alpha], dual=True, multi_class='ovr', fit_intercept=True, verbose=0, random_state=None, penalty='l2', l1_ratio=None, max_iter=1000, tol=1e-05, shuffle=True, verbose=0, epsilon=0.1, random_state=None, n_jobs=1, loss='log', warm_start=False, average=False, multi_class='ovr', fit_intercept=True, verbose=0, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5, class_weight=None, n_iter=None, max_iter=None, tol=None, shuffle=False)
clf.fit(train_x_mix, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_mix, train_y)

predict_y = sig_clf.predict_proba(train_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,predict_y))
predict_y = sig_clf.predict_proba(cv_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(y_cv,predict_y))
predict_y = sig_clf.predict_proba(test_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test,predict_y))

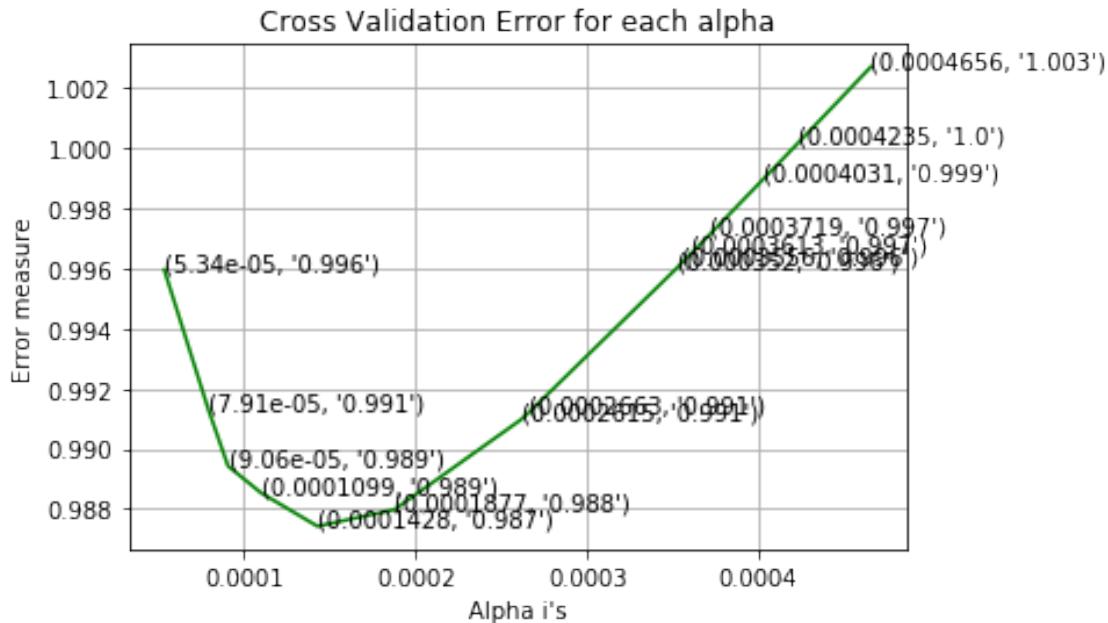
for alpha = 5.34e-05
Log Loss : 0.9959503789613564
for alpha = 7.91e-05
Log Loss : 0.9913118694679116
for alpha = 9.06e-05
Log Loss : 0.9894297447694269
for alpha = 0.0001099
Log Loss : 0.9885346287655487
for alpha = 0.0001428
Log Loss : 0.9874270825792686
for alpha = 0.0001877
Log Loss : 0.9879791151066899
for alpha = 0.0002615
Log Loss : 0.9909688571134118
for alpha = 0.0002663
Log Loss : 0.991215778294893
for alpha = 0.000352
Log Loss : 0.9959797521442814
for alpha = 0.0003556
Log Loss : 0.996188236233669
for alpha = 0.0003613
Log Loss : 0.9965191995290914
for alpha = 0.0003719
Log Loss : 0.9971372342265112
for alpha = 0.0004031

```

```

Log Loss : 0.9989711501202482
for alpha = 0.0004235
Log Loss : 1.000178780288698
for alpha = 0.0004656
Log Loss : 1.0026851505797192

```



```

For values of best alpha = 0.0001428 The train log loss is: 0.46517085125881996
For values of best alpha = 0.0001428 The cross validation log loss is: 0.9874270825792686
For values of best alpha = 0.0001428 The test log loss is: 0.9605095088297325

```

```

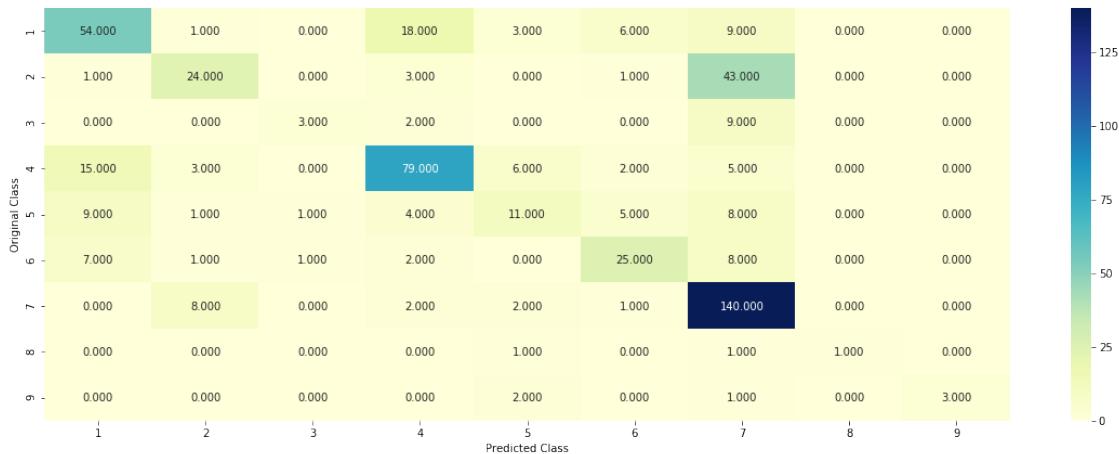
In [119]: clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
                           sig_clf,temp = predict_and_plot_confusion_matrix(train_x_mix, train_y, cv_x_mix, cv_y,
                           list_data = []
                           list_data.append('Tf_Idf+var_transform+LR')
                           list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_mix), eps=1e-15))
                           list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_mix), eps=1e-15))
                           list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_mix), eps=1e-15))
                           list_data.append('alpha = '+str(clf.alpha))
                           list_data.append(temp)
                           final_results.append(list_data)

```

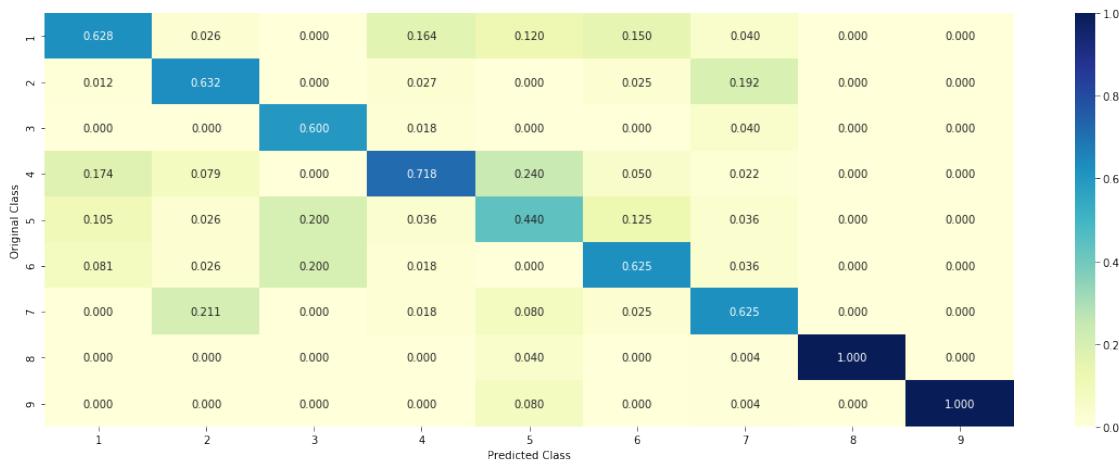
```

Log loss : 0.9874270825792686
Number of mis-classified points : 0.3609022556390977
----- Confusion matrix -----

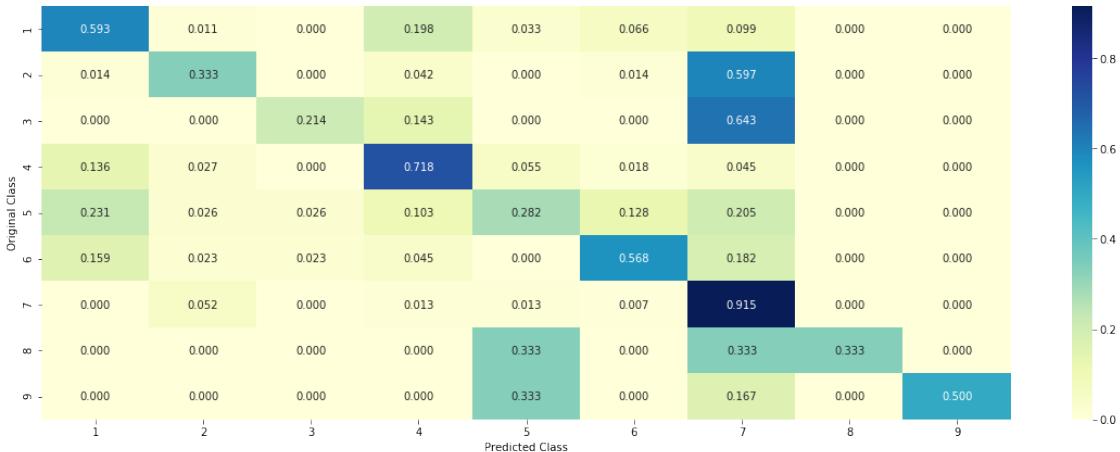
```



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



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



```
In [120]: alpha = [10 ** x for x in range(-6, 3)]
#alpha = np.random.uniform(0.00005, 0.0005, 15)
#alpha = np.round(alpha, 7)
#alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    #clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log',
    clf = LogisticRegression(class_weight='balanced', C=i, solver='liblinear')
    clf.fit(train_x_mix, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_mix, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_mix)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

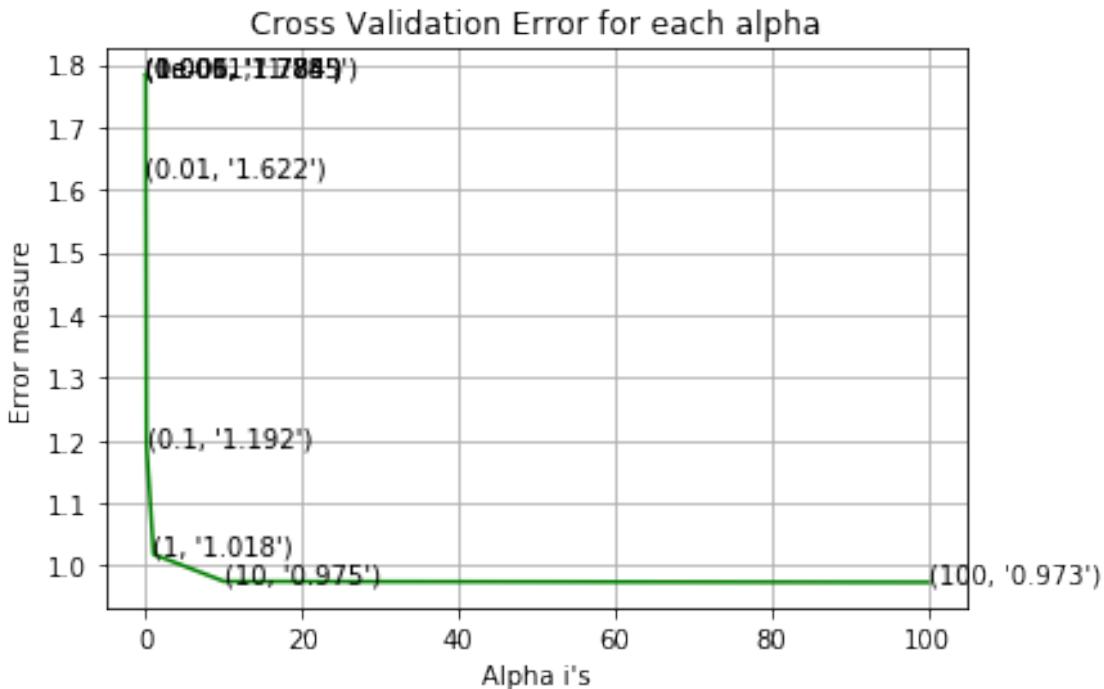
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")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

best_alpha = np.argmin(cv_log_error_array)
#clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
clf = LogisticRegression(class_weight='balanced', C=alpha[best_alpha], solver='liblinear')
```

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

predict_y = sig_clf.predict_proba(train_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss"
predict_y = sig_clf.predict_proba(test_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_]

for alpha = 1e-06
Log Loss : 1.784299973799127
for alpha = 1e-05
Log Loss : 1.784335309544237
for alpha = 0.0001
Log Loss : 1.7846518508255669
for alpha = 0.001
Log Loss : 1.7848764464987565
for alpha = 0.01
Log Loss : 1.6216663679475554
for alpha = 0.1
Log Loss : 1.1923489567215455
for alpha = 1
Log Loss : 1.0180119957574678
for alpha = 10
Log Loss : 0.9746497488932047
for alpha = 100
Log Loss : 0.9731493877536548
```



For values of best alpha = 100 The train log loss is: 0.3798556221390054

For values of best alpha = 100 The cross validation log loss is: 0.9731493877536548

For values of best alpha = 100 The test log loss is: 0.9379883111072473

```
In [121]: alpha = [10 ** x for x in range(-6, 3)]
alpha = np.random.uniform(3,85,15)
alpha = np.round(alpha,7)
alpha.sort()
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    #clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log',
    clf = LogisticRegression(class_weight='balanced', C=i, solver='liblinear')
    clf.fit(train_x_mix, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_mix, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_mix)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps_
    # to avoid rounding error while multiplying probabilites we use log-probability
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
```

```

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

best_alpha = np.argmin(cv_log_error_array)
#clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2',
clf = LogisticRegression(class_weight='balanced',C=alpha[best_alpha],solver='liblinear')
clf.fit(train_x_mix, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_mix, train_y)

predict_y = sig_clf.predict_proba(train_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_
predict_y = sig_clf.predict_proba(cv_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log lo_
predict_y = sig_clf.predict_proba(test_x_mix)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_)

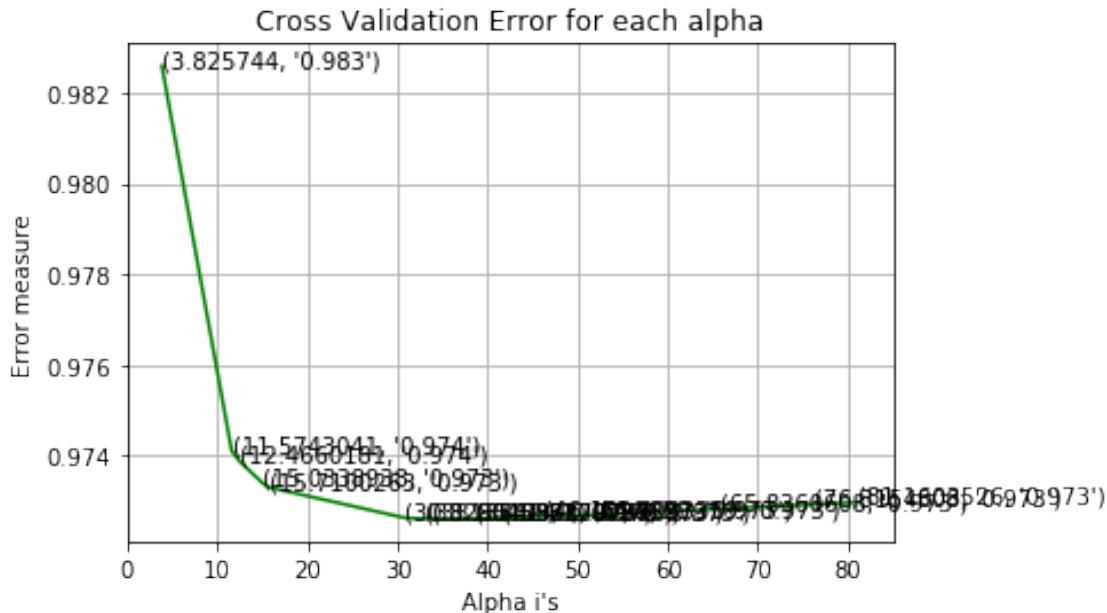
for alpha = 3.825744
Log Loss : 0.9825985727351075
for alpha = 11.5743041
Log Loss : 0.9741036902696403
for alpha = 12.4660181
Log Loss : 0.9738682437082615
for alpha = 15.0338938
Log Loss : 0.9733848359314272
for alpha = 15.7100263
Log Loss : 0.9732961340998615
for alpha = 30.8285178
Log Loss : 0.9726207934279304
for alpha = 33.1358249
Log Loss : 0.9726009690549103
for alpha = 33.9045971
Log Loss : 0.9726021290027661
for alpha = 38.5382797
Log Loss : 0.972608062157796
for alpha = 41.6700036
Log Loss : 0.9726323396800788
for alpha = 46.1890629
Log Loss : 0.9726592894043548
for alpha = 51.6033359
Log Loss : 0.9726787700667852
for alpha = 65.8360608

```

```

Log Loss : 0.9728186778208354
for alpha = 76.3154808
Log Loss : 0.9729191281079113
for alpha = 81.1603526
Log Loss : 0.9729651770321597

```



```

For values of best alpha = 33.1358249 The train log loss is: 0.39397955110729116
For values of best alpha = 33.1358249 The cross validation log loss is: 0.9726009690549103
For values of best alpha = 33.1358249 The test log loss is: 0.9350926673607827

```

```

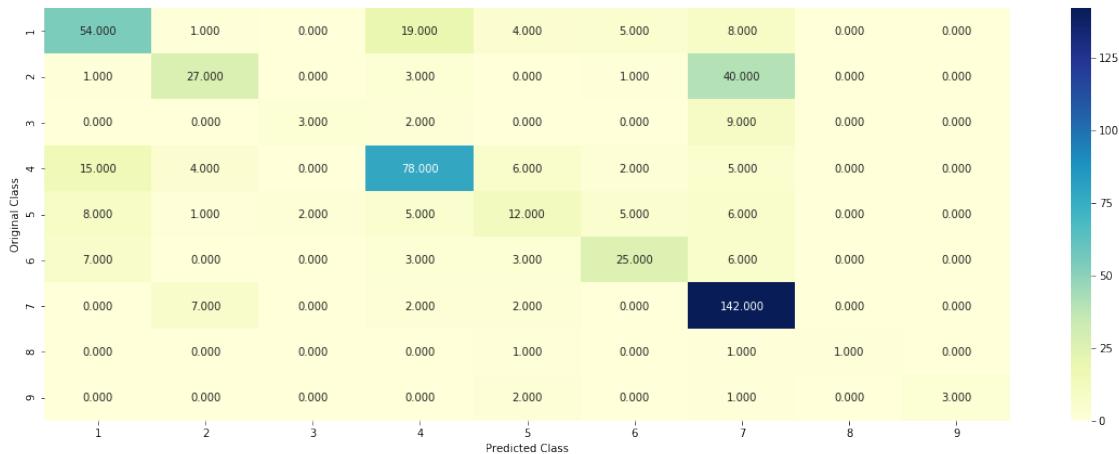
In [122]: clf = LogisticRegression(class_weight='balanced', C=alpha[best_alpha], solver='liblinear')
          sig_clf,temp = predict_and_plot_confusion_matrix(train_x_mix, train_y, cv_x_mix, cv_y)
          list_data = []
          list_data.append('Tf_Idf+var_transform+LR-Liblinear')
          list_data.append(log_loss(y_train, sig_clf.predict_proba(train_x_mix), eps=1e-15))
          list_data.append(log_loss(y_cv, sig_clf.predict_proba(cv_x_mix), eps=1e-15))
          list_data.append(log_loss(y_test, sig_clf.predict_proba(test_x_mix), eps=1e-15))
          list_data.append('alpha = '+str(clf.C))
          list_data.append(temp)
          final_results.append(list_data)

```

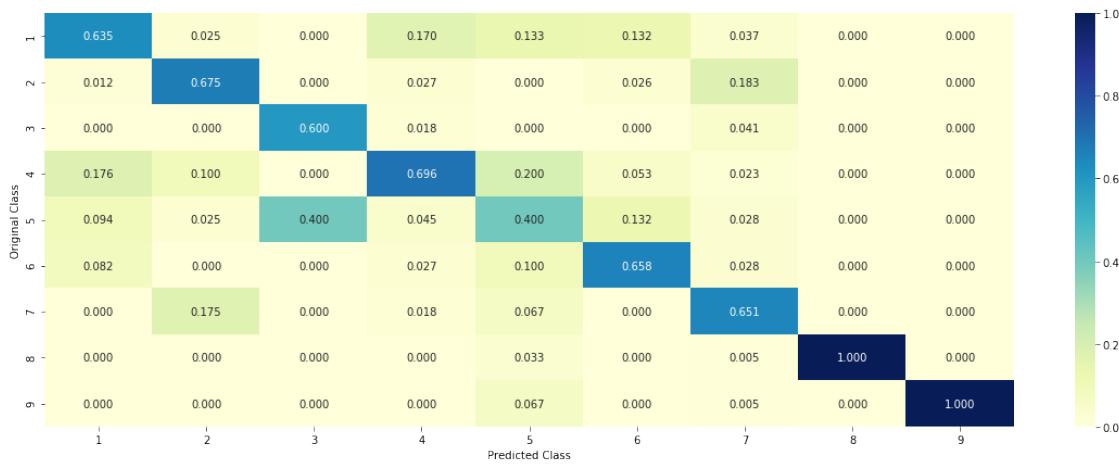
```

Log loss : 0.9726009690549103
Number of mis-classified points : 0.35150375939849626
----- Confusion matrix -----

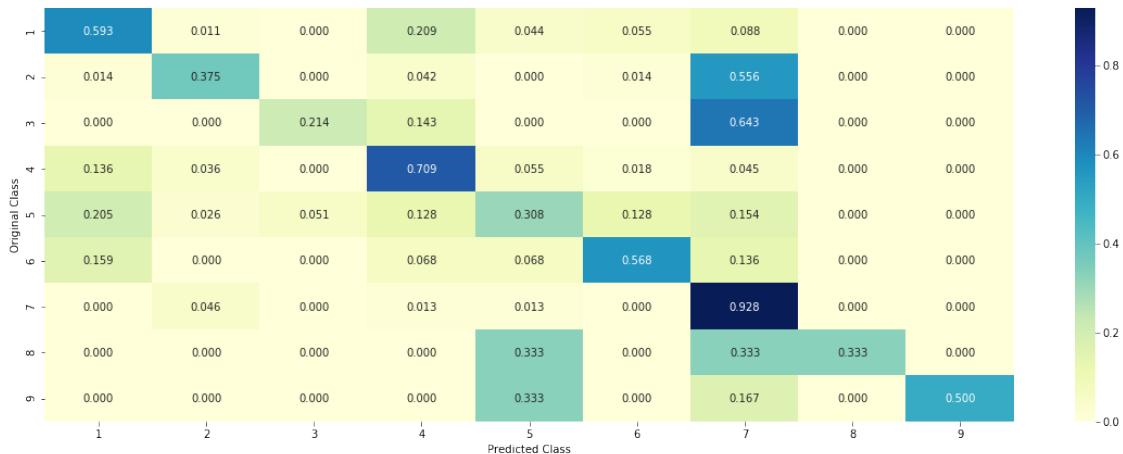
```



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



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

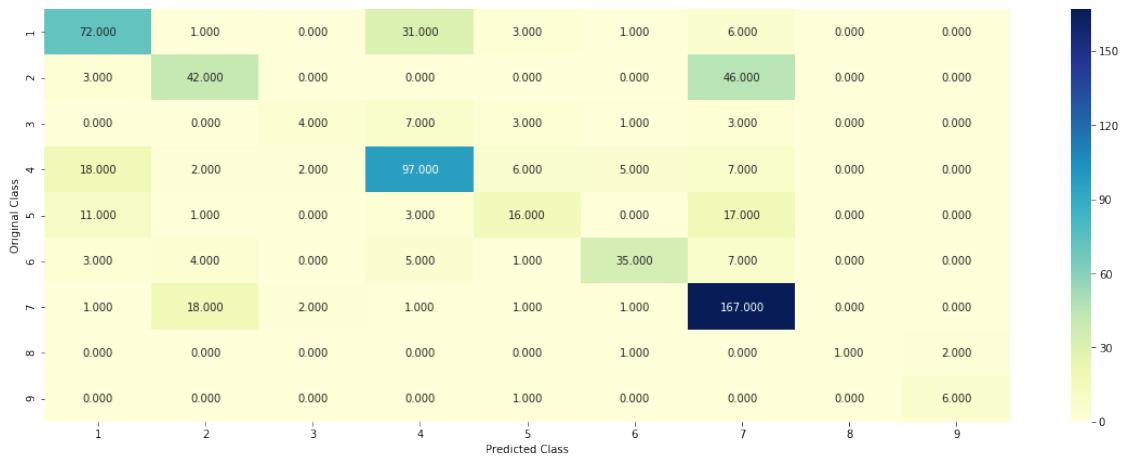


```
In [123]: clf = LogisticRegression(class_weight='balanced', C=alpha[best_alpha], solver='liblinear')
sig_clf,temp = predict_and_plot_confusion_matrix(train_x_mix, train_y, test_x_mix, test_y)
#final_results.append(list_data)
```

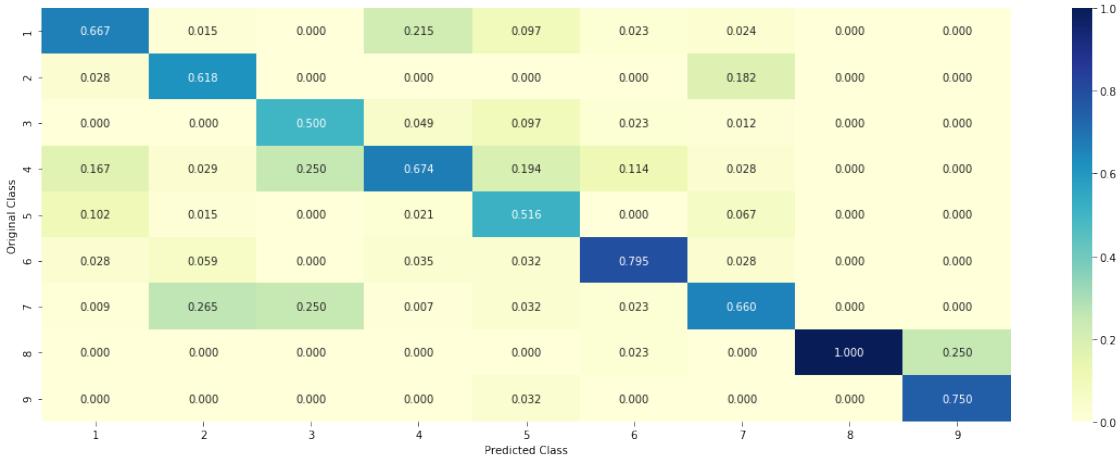
Log loss : 0.9350926673607827

Number of mis-classified points : 0.3383458646616541

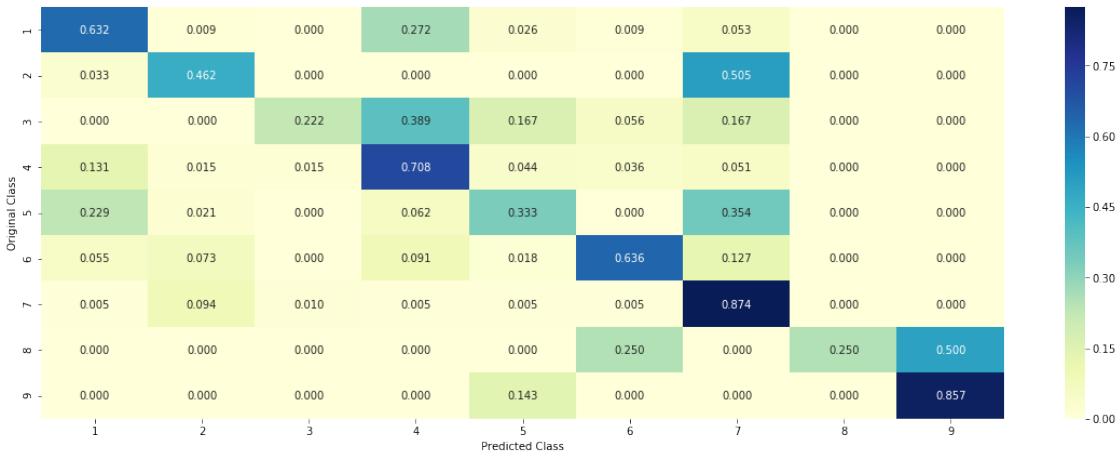
----- Confusion matrix -----



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



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



```
In [98]: result_df = pd.DataFrame(final_results,columns=['Model','TrainLogLoss','CVLogLoss',
                                                       'TestLogLoss','Hyperparams','CV%Missclassification'])
```

```
In [106]: result_df
```

```
Out[106]:
```

	Model	TrainLogLoss	\
0	Tf_Idf+NB	0.637112	
1	Tf_Idf+KNN	0.900452	
2	Tf_Idf+LR+Class_Balancing+SGD	0.440298	
3	Tf_Idf+LR+Class_Balancing+Liblinear	0.382558	
4	Tf_Idf+LR+Class_Balancing	0.430260	
5	Tf_Idf+LSVC+Class_Balance	0.491855	

6	Tf_Idf+LSVC+ImClass_Balance	0.453349
7	Tf_Idf+RF	0.487745
8	Tf_Idf+Stackig(LR+KNN+LSVC)	0.364243
9	Tf_Idf+MaxVote	0.555619
10	Tf_Idf_Top1000+NB	0.885138
11	Tf_Idf_Top1000+KNN	0.905638
12	Tf_Idf_Top1000+LR+Class_Balancing+SGD	0.472676
13	Tf_Idf_Top1000+LR+Class_Imbalance	0.468848
14	Tf_Idf_Top1000+LSVC+Class_Balance	0.518984
15	Tf_Idf+LSVC+ImClass_Balance	0.483576
16	Tf_Idf_Top1000+RF	0.543795
17	Tf_Idf_Top1000+Stackig(LR+KNN+LSVC)	0.374817
18	Tf_Idf+MaxVote(LR+KNN+LSVC)	0.374817
19	Bow+Uni-Bi_gram+SGD-LR+Class_Balance	0.789457
20	Bow+Uni-Bi_gram+SGD-LR+ImClass_Balance	0.774720
21	Tf_Idf+Uni-Bi_grams+SGD-LR+Class_Balancing	0.427420
22	Tf_Idf+Uni-Bi_grams+SGD-LR+ImClass_Balancing	0.414285
23	Tf_Idf+Uni-Bi_grams+SAG-LR+Class_Balancing	0.387444
24	Tf_Idf+Uni-Bi_grams+(Liblinear+dual-LR)+Class_...	0.393327
25	Tf_Idf+Wordnetlemm+(Liblinear+dual+LR)+Class_B...	0.407944
26	BoW+Wordnetlemm+(Liblinear+dual+LR)+Class_Balance	0.545439
27	Tf_Idf_uni-bigram+lemma+(Liblinear+dual+LR)+Cl...	0.393131
29	Tf_Idf+some_var_transform+LR+Class_Balance	0.735503
30	Tf_Idf+gene_agvw2v+aloc+var_transform+LR	0.467970
31	Tf_Idf+gene_agvw2v+var_transform+LR	0.482200
32	Tf_Idf+var_transform+LR	0.465171
33	Tf_Idf+var_transform+LR-Liblinear	0.393980

	CVLogLoss	TestLogloss	Hyperparams	\
0	1.149684	1.195156	alpha = 5.658873481646807e-05	
1	1.107991	1.135785	K 5	
2	0.987346	0.962494	alpha = 0.00013	
3	0.978128	0.955732	C = 98.187	
4	0.985754	0.961923	alpha = 0.000133	
5	1.070155	1.045604	alpha = 0.00043	
6	1.045299	1.024191	alpha = 0.00033	
7	1.038712	1.087970	Estimators = 2000 depth 20	
8	1.067195	1.035989	C = 0.1218	
9	1.025511	1.012749		
10	1.209007	1.195441	alpha = 1.3216328879483359	
11	1.104017	1.127495	K 5	
12	1.037321	1.003445	alpha = 0.0001328	
13	1.038077	1.005006	alpha = 0.0001505	
14	1.121336	1.094028	alpha 0.0003657	
15	1.092861	1.076641	alpha 0.000371	
16	1.038261	1.041721	estimators = 2000 depth = 20	
17	1.100716	1.080308	C = 0.1218	
18	1.100716	1.080308		

19	1.188500	1.187391	alpha = 0.027946
20	1.178299	1.176136	alpha = 0.025728
21	0.975910	0.958604	alpha = 0.0001215
22	0.974932	0.958619	alpha = 0.0001119
23	0.966148	0.949079	C = 44.878
24	0.966122	0.948363	C = 31.042
25	0.980566	0.957212	C = 22.574
26	1.031506	1.024977	C = 1
27	0.968235	0.949021	C = 32.644826
29	1.015085	0.995356	alpha = 0.000234
30	0.988561	0.963187	alpha = 0.000152
31	0.988190	0.960103	alpha = 0.0001956
32	0.987427	0.960510	alpha = 0.0001428
33	0.972601	0.935093	alpha = 33.1358249

#### CV%Missclassification

0	0.357143
1	0.372180
2	0.355263
3	0.349624
4	0.355263
5	0.360902
6	0.357143
7	0.357143
8	0.345865
9	0.344361
10	0.398496
11	0.374060
12	0.372180
13	0.374060
14	0.364662
15	0.359023
16	0.342105
17	0.374436
18	0.374436
19	0.407519
20	0.396992
21	0.353383
22	0.353383
23	0.343985
24	0.347744
25	0.353383
26	0.343985
27	0.345865
29	0.374060
30	0.360902
31	0.360902
32	0.360902

### Error analysis on Test data

```
In [102]: clf = LogisticRegression(class_weight='balanced', C=33.1358249, solver='liblinear')
clf.fit(train_x_mix, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_mix, train_y)

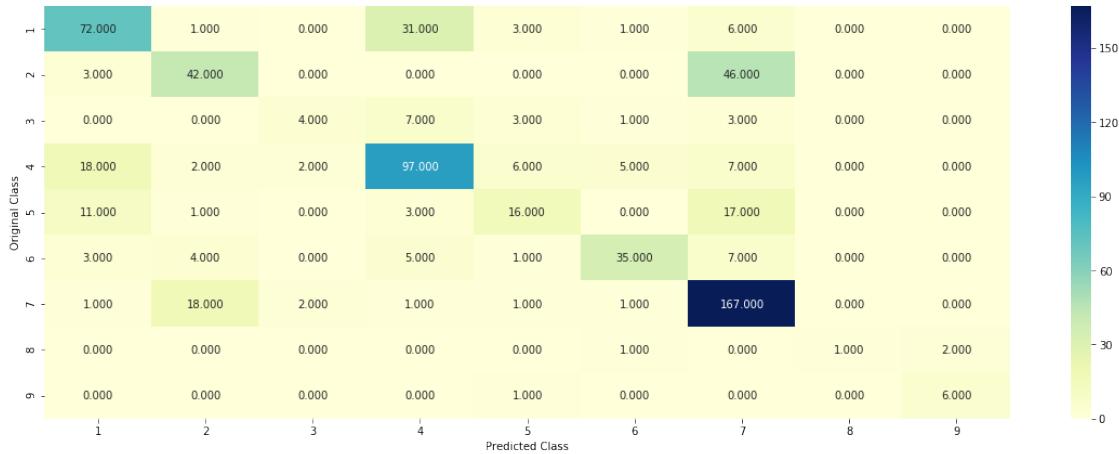
Out[102]: CalibratedClassifierCV(base_estimator=LogisticRegression(C=33.1358249, class_weight=
                           fit_intercept=True, intercept_scaling=1, max_iter=100,
                           multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
                           solver='liblinear', tol=0.0001, verbose=0, warm_start=False),
                           cv=3, method='sigmoid')

In [103]: pred_test = sig_clf.predict(test_x_mix)
test_df['pred_score'] = pred_test
# for calculating log_loss we will provide the array of probabilities belongs to each class
print("Log loss of test data :", log_loss(test_y, sig_clf.predict_proba(test_x_mix)))
# calculating the number of data points that are misclassified
print("Number of mis-classified points :", np.count_nonzero((pred_test - y_test)) / y_test.sum())
plot_confusion_matrix(y_test, pred_test)
```

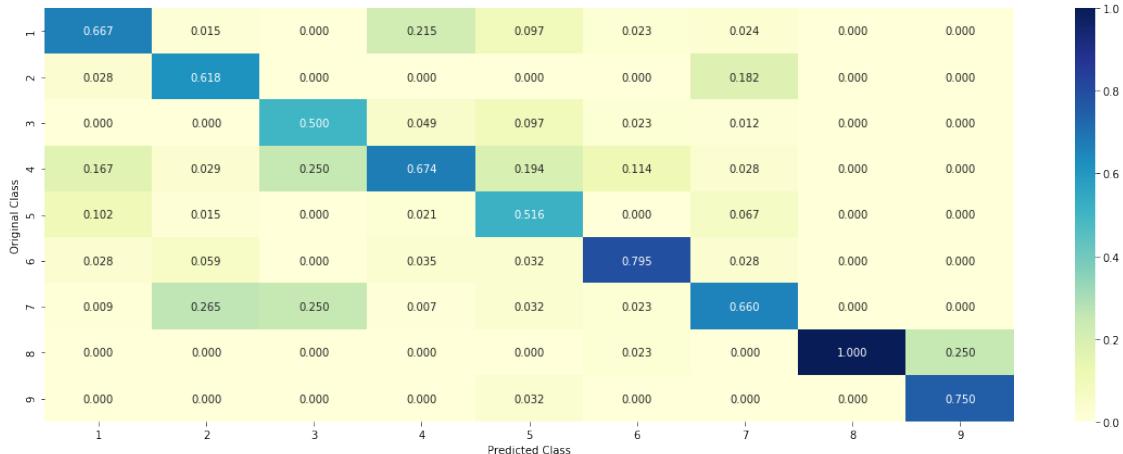
Log loss of test data : 0.9350926673607827

Number of mis-classified points : 0.3383458646616541

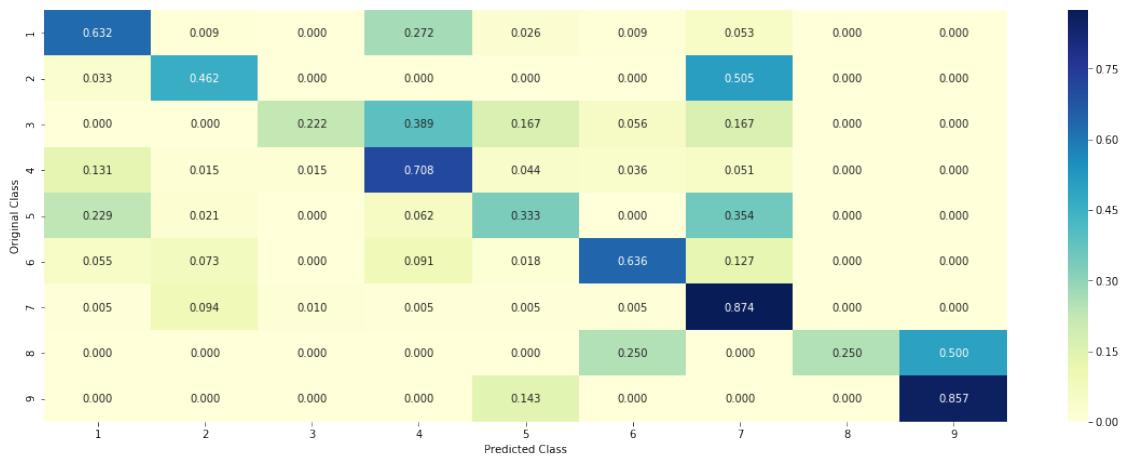
----- Confusion matrix -----



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



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



```
In [104]: test_df['proba'] = list(sig_clf.predict_proba(test_x_mix))
```

```
In [105]: error_test = test_df[test_df.Class!=test_df.pred_score]
```

```
In [106]: #no of errors by each class
error_test.Class.value_counts()
```

```
Out[106]: 2    49
          1    42
          4    40
          5    32
          7    24
```

```
6    20
3    14
8     3
9     1
Name: Class, dtype: int64
```

```
In [107]: #no of test points by each class
test_df.Class.value_counts()
```

```
Out[107]: 7    191
4    137
1    114
2     91
6     55
5     48
3     18
9      7
8      4
Name: Class, dtype: int64
```

```
In [108]: # % of error for each class
error_prct = error_test.Class.value_counts()/test_df.Class.value_counts() *100
error_prct.sort_values()
```

```
Out[108]: 7    12.565445
9    14.285714
4    29.197080
6    36.363636
1    36.842105
2    53.846154
5    66.666667
8    75.000000
3    77.777778
Name: Class, dtype: float64
```

```
In [109]: dict(error_prct/error_prct.sum())
```

```
Out[109]: {1: 0.09152304399918644,
2: 0.1337644489218879,
3: 0.19321531510939366,
4: 0.07253129659998092,
5: 0.16561312723662308,
6: 0.09033443303815807,
7: 0.031215039688578174,
8: 0.186314768141201,
9: 0.03548852726499066}
```

We can observe that for 3,8,5,2 errors are very high and also points in train and test data also low compared to others

```
In [110]: error_test[error_test.Class==8]
```

```
Out[110]:      ID  Gene Variation  Class  \
1770  1770    IDH2      R172M      8
750    750    ERBB2      T798I      8
1768  1768    IDH2      R172G      8

                                         TEXT  pred_score  \
1770  cancer genome characterization efforts provide...      9
750    purpose mutations associated resistance kinase...      6
1768  heterozygous mutations either r132 residue iso...      9

                           proba
1770  [0.08759971552698849, 0.09380602553795729, 0.0...
750    [0.019657044505234804, 0.20334103147916172, 0...
1768  [0.09464985036172145, 0.08188177709885049, 0.0...
```

```
In [111]: error_test[error_test.Class==3]
```

```
Out[111]:      ID  Gene Variation  Class  \
3        3    CBL      N454D      3
196     196   EGFR      C628Y      3
616     616  FBXW7      S562L      3
2587    2587  BRCA1      S186Y      3
996     996   TSC1      T417I      3
2206    2206   PTEN      T131S      3
1698    1698   PMS2      I18V       3
1002    1002   TSC1      G305R      3
295     295  CHEK2      P85L       3
482     482   TP53      G334R      3
2509    2509  BRCA1      V271L      3
1898    1898   MTOR      A41T       3
3113    3113  RAD51C      G264S      3
1862    1862   MTOR      R2430M     3

                                         TEXT  pred_score  \
3    recent evidence demonstrated acquired uniparen...      4
196  feature many gliomas amplification epidermal g...      7
616  background melanoma heterogeneous tumor subgro...      4
2587 mutations brca1 brca2 account majority heredit...      5
996  50 transitional cell carcinomas bladder show l...      4
2206 pten phosphatase tensin homolog phosphatase un...      4
1698 identification high risk disease causing const...      5
1002 tuberous sclerosis complex tsc autosomal domin...      4
295  checkpoint kinase 2 chek2 chk2 emerges importa...      6
482  abstract purpose adrenocortical carcinoma acc ...      4
2509 mutations brca1 brca2 account majority heredit...      5
1898 genes encoding components pi3k akt mtor signal...      7
```

3113	strong evidence overtly inactivating mutations...	4
1862	genes encoding components pi3k akt mtor signal...	7

	proba
3	[0.11012062676533511, 0.1657867905606478, 0.01...
196	[0.0509071218417158, 0.11654689185114486, 0.00...
616	[0.31177058449671596, 0.06998627419854121, 0.0...
2587	[0.0444995012341162, 0.010663010570992016, 0.2...
996	[0.2119975914926698, 0.04543411711965084, 0.07...
2206	[0.015129389019545953, 0.013925059601095859, 0...
1698	[0.2803510704777019, 0.025450732318678528, 0.0...
1002	[0.08414163791065872, 0.029875366925429272, 0...
295	[0.13336244600171912, 0.11859150282456161, 0.0...
482	[0.13948116912837374, 0.026547427189130456, 0...
2509	[0.05220618912668703, 0.012716257034596563, 0...
1898	[0.019147737162793377, 0.09338359349431331, 0...
3113	[0.1964438357020701, 0.11464518752775336, 0.02...
1862	[0.019147737162793377, 0.09338359349431331, 0...

In [112]: error\_test[error\_test.Class==5].drop('TEXT',axis=1)

Out[112]:

ID	Gene	Variation	Class	pred_score	\
2605	2605	BRCA1	V1665M	5	4
1509	1509	ALK	T1151M	5	7
760	760	ERBB2	R143Q	5	2
1702	1702	PMS2	T485K	5	4
2607	2607	BRCA1	M1775V	5	1
888	888	PDGFRA	S478P	5	7
2872	2872	BRCA2	L2865V	5	1
2636	2636	BRCA1	R1751Q	5	1
730	730	ERBB2	V750E	5	7
1185	1185	PIK3CA	Q60K	5	7
1539	1539	ALK	T1343I	5	7
1217	1217	PIK3CA	I391M	5	7
2553	2553	BRCA1	D1778N	5	1
1197	1197	PIK3CA	A1066V	5	7
2678	2678	BRCA1	D1778G	5	1
1457	1457	FGFR2	T730S	5	7
1276	1276	PIK3R2	S273C	5	7
3208	3208	RB1	C712R	5	1
1357	1357	AKT1	K39N	5	7
2532	2532	BRCA1	N1647K	5	1
2565	2565	BRCA1	H1805P	5	1
2006	2006	MAP2K1	D65N	5	7
538	538	SMAD2	W368H	5	1
1359	1359	AKT1	D32Y	5	7
1417	1417	FGFR3	F384L	5	7
1500	1500	FGFR2	E475K	5	7

1154	1154	KMT2C	S3660L	5	4
764	764	ERBB2	V777M	5	7
2147	2147	PTCH1	R571W	5	1
709	709	ERBB2	S760A	5	7
1459	1459	FGFR2	V755I	5	7
536	536	SMAD2	F346V	5	1

				proba
2605			[0.3213780495710148, 0.011830932914130197, 0.0...	
1509			[0.017643324862938698, 0.11758422939444678, 0...	
760			[0.03703419806370475, 0.3734283513396397, 0.01...	
1702			[0.22675862580593506, 0.029657914425030695, 0...	
2607			[0.48668852159938264, 0.013068986116580214, 0...	
888			[0.031717417026539775, 0.03459185780909833, 0...	
2872			[0.5341889475806343, 0.021011824957045774, 0.0...	
2636			[0.4263051849345633, 0.012847630709232317, 0.0...	
730			[0.037202641839789184, 0.12761640231419336, 0...	
1185			[0.02674751539565014, 0.14199054487929974, 0.0...	
1539			[0.021263849039252537, 0.0966047656793894, 0.2...	
1217			[0.01657306383698307, 0.11176608071128906, 0.0...	
2553			[0.4991084232327833, 0.01139202404020985, 0.01...	
1197			[0.034782628710693674, 0.1815339132630941, 0.0...	
2678			[0.4991084232327833, 0.01139202404020985, 0.01...	
1457			[0.0946831895921358, 0.05770402786485756, 0.00...	
1276			[0.047980496127297094, 0.04422222105361728, 0...	
3208			[0.3759264016717813, 0.07066331725411126, 0.01...	
1357			[0.02600009018580396, 0.07765178822893352, 0.0...	
2532			[0.44226370843831075, 0.013389030497367496, 0...	
2565			[0.44226370843831075, 0.013389030497367496, 0...	
2006			[0.03098010591019619, 0.229454489131365, 0.009...	
538			[0.532760597654175, 0.03317205396274788, 0.011...	
1359			[0.02600009018580396, 0.07765178822893352, 0.0...	
1417			[0.0385028549578268, 0.21629743029617857, 0.00...	
1500			[0.31872001884857143, 0.05235225037161617, 0.0...	
1154			[0.2385524726760918, 0.12874775726683313, 0.01...	
764			[0.023906538636965013, 0.1331014567096753, 0.0...	
2147			[0.24903217258604904, 0.18079864112099617, 0.0...	
709			[0.023585927192648504, 0.13395406938596321, 0...	
1459			[0.0946831895921358, 0.05770402786485756, 0.00...	
536			[0.5168417172360128, 0.03878693577023747, 0.01...	

In [114]: `print(error_test[error_test.Class==5].proba.values)`

```
[array([0.32137805, 0.01183093, 0.02061398, 0.349291 , 0.22595242,
       0.06285671, 0.00428796, 0.00178876, 0.0020002 ]),
 array([0.01764332, 0.11758423, 0.24857448, 0.01080236, 0.04023725,
       0.00888571, 0.55105323, 0.00215888, 0.00306054]),
 array([0.0370342 , 0.37342835, 0.01064545, 0.04800591, 0.0811078 ,
```

```

    0.15296218, 0.2874939 , 0.00380109, 0.00552112])
array([0.22675863, 0.02965791, 0.01043229, 0.50262591, 0.18358747,
       0.02183297, 0.01776084, 0.00318923, 0.00415474])
array([0.48668852, 0.01306899, 0.01835305, 0.19076315, 0.11859485,
       0.15834902, 0.00955641, 0.00214223, 0.00248378])
array([0.03171742, 0.03459186, 0.06577031, 0.02672026, 0.05883023,
       0.01952527, 0.75588542, 0.00293304, 0.0040262 ])
array([0.53418895, 0.02101182, 0.00743331, 0.047111 , 0.34276906,
       0.0300951 , 0.01166222, 0.00246189, 0.00326666])
array([0.42630518, 0.01284763, 0.0226233 , 0.19259336, 0.21164857,
       0.12434491, 0.00561878, 0.00197663, 0.00204165])
array([0.03720264, 0.1276164 , 0.01004203, 0.03846598, 0.32841779,
       0.05774376, 0.39102189, 0.00472548, 0.00476403])
array([0.02674752, 0.14199054, 0.00592582, 0.01722481, 0.09291819,
       0.06789406, 0.63956767, 0.0031424 , 0.00458899])
array([0.02126385, 0.09660477, 0.20985432, 0.01376131, 0.07291007,
       0.01159911, 0.56814553, 0.00248155, 0.00337951])
array([0.01657306, 0.11176608, 0.0059978 , 0.0155423 , 0.05160886,
       0.08107165, 0.71026586, 0.00327102, 0.00390336])
array([0.49910842, 0.01139202, 0.01872358, 0.15656914, 0.2201446 ,
       0.08544905, 0.00458902, 0.00187916, 0.00214501])
array([0.03478263, 0.18153391, 0.00657536, 0.01342826, 0.07056659,
       0.07824518, 0.60717562, 0.00321509, 0.00447736])
array([0.49910842, 0.01139202, 0.01872358, 0.15656914, 0.2201446 ,
       0.08544905, 0.00458902, 0.00187916, 0.00214501])
array([0.09468319, 0.05770403, 0.00847876, 0.02027275, 0.07308318,
       0.07550607, 0.66305085, 0.00276551, 0.00445566])
array([0.0479805 , 0.04422222, 0.01277899, 0.10053621, 0.1966221 ,
       0.02498831, 0.56404277, 0.00365515, 0.00517377])
array([0.3759264 , 0.07066332, 0.0156127 , 0.14846746, 0.0534952 ,
       0.11305168, 0.20799874, 0.00556838, 0.00921612])
array([0.02600009, 0.07765179, 0.06911845, 0.01710515, 0.09707739,
       0.01249346, 0.68247724, 0.01359684, 0.0044796 ])
array([0.44226371, 0.01338903, 0.01975009, 0.10745773, 0.37018476,
       0.03855427, 0.00431109, 0.0018464 , 0.00224291])
array([0.44226371, 0.01338903, 0.01975009, 0.10745773, 0.37018476,
       0.03855427, 0.00431109, 0.0018464 , 0.00224291])
array([0.03098011, 0.22945449, 0.009869 , 0.03448533, 0.16555499,
       0.0540219 , 0.46729127, 0.00338459, 0.00495832])
array([0.5327606 , 0.03317205, 0.01171289, 0.08974253, 0.06930689,
       0.227027 , 0.02747651, 0.00369923, 0.0051023 ])
array([0.02600009, 0.07765179, 0.06911845, 0.01710515, 0.09707739,
       0.01249346, 0.68247724, 0.01359684, 0.0044796 ])
array([0.03850285, 0.21629743, 0.00822235, 0.02514967, 0.09429598,
       0.02707558, 0.58338426, 0.00293219, 0.00413969])
array([0.31872002, 0.05235225, 0.00899186, 0.02011261, 0.05486625,
       0.20190801, 0.33639921, 0.00259627, 0.00405351])
array([0.23855247, 0.12874776, 0.01896121, 0.32602442, 0.06101204,
       0.01952527, 0.75588542, 0.00293304, 0.0040262 ])

```

```

    0.09398337, 0.11831645, 0.00548583, 0.00891645])
array([0.02390654, 0.13310146, 0.00914765, 0.03469783, 0.06031199,
       0.11053393, 0.62125308, 0.0030052 , 0.00404233])
array([0.24903217, 0.18079864, 0.0208678 , 0.13626738, 0.06615151,
       0.14120204, 0.18898646, 0.00695712, 0.00973689])
array([0.02358593, 0.13395407, 0.00693674, 0.03636865, 0.03767477,
       0.11052467, 0.64460598, 0.00261195, 0.00373724])
array([0.09468319, 0.05770403, 0.00847876, 0.02027275, 0.07308318,
       0.07550607, 0.66305085, 0.00276551, 0.00445566])
array([0.51684172, 0.03878694, 0.01188834, 0.16874977, 0.07112912,
       0.15389849, 0.02910835, 0.00389088, 0.00570639])

```

In [120]: #class 5 predicted as  
error\_test[error\_test.Class==5].pred\_score.value\_counts()

Out[120]: 7 17  
1 11  
4 3  
2 1  
Name: pred\_score, dtype: int64

In [121]: train\_df[train\_df.Class==5]

Out[121]:

	ID	Gene	Variation	Class	\
1762	1762	IDH1	I99M	5	
2732	2732	BRAF	I463S	5	
2908	2908	NF2	Q324L	5	
2472	2472	BRCA1	L358R	5	
3282	3282	RET	M980T	5	
106	106	MSH6	G39E	5	
582	582	SMAD4	R441P	5	
2899	2899	NF2	T352M	5	
2615	2615	BRCA1	T47D	5	
3289	3289	RET	V648I	5	
2026	2026	MAP2K1	N382H	5	
2773	2773	BRAF	G464E	5	
1594	1594	VHL	L128F	5	
1412	1412	FGFR3	C582F	5	
1318	1318	MLH1	R217C	5	
2528	2528	BRCA1	N550H	5	
2625	2625	BRCA1	K45Q	5	
2536	2536	BRCA1	L165P	5	
1237	1237	PIM1	P123M	5	
2467	2467	BRCA1	H1421Y	5	
2868	2868	BRCA2	I2285V	5	
2070	2070	TET2	R1262A	5	
1428	1428	FGFR3	N653H	5	
2519	2519	BRCA1	E1060A	5	

2259	2259	PTEN	Q110R	5
2081	2081	TET2	S1290A	5
2423	2423	BRCA1	A1830T	5
312	312	ELF3	S308A	5
1913	1913	SMO	E518K	5
1738	1738	MSH2	A272V	5
...	...	...	...	...
2741	2741	BRAF	W531C	5
2610	2610	BRCA1	N1819S	5
154	154	EGFR	H870R	5
2823	2823	BRCA2	N372H	5
3260	3260	RET	T338I	5
1356	1356	AKT1	P42T	5
873	873	PDGFRA	R481G	5
2018	2018	MAP2K1	E120Q	5
2656	2656	BRCA1	M1663K	5
1165	1165	PIK3CA	I31M	5
2671	2671	BRCA1	F1695L	5
1958	1958	ATM	D1853N	5
2555	2555	BRCA1	A1669S	5
1203	1203	PIK3CA	A1020V	5
265	265	EGFR	P848L	5
2824	2824	BRCA2	R2520Q	5
991	991	TSC1	K82T	5
507	507	TP53	T123A	5
2604	2604	BRCA1	E1214K	5
1744	1744	MSH2	S860L	5
2533	2533	BRCA1	Q1785H	5
2575	2575	BRCA1	P1637L	5
776	776	ERBB3	V714M	5
2647	2647	BRCA1	S1301R	5
2608	2608	BRCA1	R1589H	5
2498	2498	BRCA1	R496C	5
2611	2611	BRCA1	V1808A	5
704	704	ERBB2	L49H	5
2546	2546	BRCA1	S1486C	5
2630	2630	BRCA1	T1720A	5

#### TEXT

1762 introduction somatic mutations human cytosolic...  
 2732 recently mutations b raf gene identified varie...  
 2908 neurofibromatosis 2 nf2 tumor predisposition s...  
 2472 mutations brca1 brca2 account majority heredit...  
 3282 many missense mutations ret proto oncogene fou...  
 106 identification high risk disease causing const...  
 582 smad proteins mediate transforming growth fact...  
 2899 despite intense study neurofibromatosis type 2...  
 2615 brca1 accumulates nuclear foci phase reassembl...

3289 germline somatic ret oncogene mutations found ...  
2026 performed exome sequencing detect somatic muta...  
2773 recently mutations b raf gene identified varie...  
1594 examined biogenesis von hippel lindau vhl tumo...  
1412 frequent genetic alterations discovered fgfrs ...  
1318 germ line mutations hmlh1 gene frequent cause ...  
2528 abstract brca1 tumor suppressor gene found mut...  
2625 mutations brca1 brca2 account majority heredit...  
2536 mutations brca1 brca2 account majority heredit...  
1237 pim1 serine threonine kinase involved several ...  
2467 brca1 tumour suppressor pleiotropic actions ge...  
2868 mutation screening breast ovarian cancer predi...  
2070 tet proteins oxidize 5 methylcytosine 5mc dna ...  
1428 frequent genetic alterations discovered fgfrs ...  
2519 mutations brca1 brca2 account majority heredit...  
2259 abstract look direct role ultraviolet radiatio...  
2081 tet proteins oxidize 5 methylcytosine 5mc dna ...  
2423 genetic screening breast ovarian cancer suscep...  
312 ets transcription factor family comprised near...  
1913 inappropriate hedgehog hh signaling directly l...  
1738 background aims inherited deleterious mutation...  
...  
2741 noonan leopard cardiofaciocutaneous syndromes ...  
2610 abstract brca1 gene individuals risk breast ov...  
154 abstract epidermal growth factor receptor egfr...  
2823 assessment influence many rare brca2 missense ...  
3260 germline somatic ret oncogene mutations found ...  
1356 protein kinase v akt murine thymoma viral onco...  
873 fip1l1 pdgfra fusion seen fraction cases presu...  
2018 histiocytic neoplasms clonal hematopoietic dis...  
2656 genetic screening breast ovarian cancer suscep...  
1165 large scale sequencing efforts uncovering comp...  
2671 abstract brca1 gene individuals risk breast ov...  
1958 involvement atm gene specifically important ro...  
2555 abstract brca1 gene individuals risk breast ov...  
1203 large scale sequencing efforts uncovering comp...  
265 epidermal growth factor receptor egfr transmem...  
2824 assessment influence many rare brca2 missense ...  
991 tuberous sclerosis complex disease caused muta...  
507 p53 tumor suppressor gene acquires missense mu...  
2604 mutations brca1 brca2 account majority heredit...  
1744 identification high risk disease causing const...  
2533 genetic screening breast ovarian cancer suscep...  
2575 germline inactivating mutations brca1 result h...  
776 human epidermal growth factor receptor family ...  
2647 mutations brca1 brca2 account majority heredit...  
2608 abstract germline inactivating mutations brca1...  
2498 significant proportion inherited breast cancer...

```

2611 abstract germline mutations inactivate tumor s...
704 assessed somatic alleles six receptor tyrosine...
2546 mutations brca1 brca2 account majority heredit...
2630 abstract germline mutations inactivate tumor s...

[155 rows x 5 columns]

```

```
In [115]: #no of points that are greater than 0.2 probability for class 5
points_no = 0
for i in error_test[error_test.Class==5].proba.values:
    if i[4] >0.2:
        points_no = points_no +1
print(points_no)
```

8

## Feature Importance of Model

```
In [144]: train_x_mix = hstack((train_gene_feature_tfidfCoding,train_variation_feature_tfidfCoding,
                           train_operation_tfidf,train_text_feature_tfidfCoding12)).tocsr()
test_x_mix = hstack((test_gene_feature_tfidfCoding,test_variation_feature_tfidfCoding,
                     test_operation_tfidf,test_text_feature_tfidfCoding12)).tocsr()
cv_x_mix = hstack((cv_gene_feature_tfidfCoding, cv_variation_feature_tfidfCoding,
                   cv_operation_tfidf, cv_text_feature_tfidfCoding12)).tocsr()
```

```
In [133]: clf = LogisticRegression(class_weight='balanced',C=33.1358249,solver='liblinear')
clf.fit(train_x_mix, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_mix, train_y)
```

```
Out[133]: CalibratedClassifierCV(base_estimator=LogisticRegression(C=33.1358249, class_weight=
                           fit_intercept=True, intercept_scaling=1, max_iter=100,
                           multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
                           solver='liblinear', tol=0.0001, verbose=0, warm_start=False),
                           cv=3, method='sigmoid')
```

```
In [148]: # this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    #text_count_vec = TfidfVectorizer(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(v,word))
    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 [{}].format(v,word))
    elif (v<fea1_len+fea2_len+len(var_vec_opr.get_feature_names())):
        word =var_vec_opr.get_feature_names()[vfea1_lenfea2_len]
        yes_no = True if word == gene else False
        if yes_no:
            word_present += 1
            print(i, "Variation feature [{}] present in test data point [{}].format(v,word))
    else:
        temp1 = list(train_text_features12)
        word = temp1[v-(fea1_len+fea2_len)+len(var_vec_opr.get_feature_names())]
        l = word.split()
        if len(l) ==2:
            if l[0] in text.split() and l[1] in text.split():
                yes_no = True
            else:
                yes_no = False
        else:
            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(v,word))

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

```

```

In [149]: test_point_index = 5
no_feature = 500
predicted_cls = sig_clf.predict(test_x_mix[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_mix[test_point_index]),2))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)

```

```

get_imptfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index], test_df['Geno

Predicted Class : 6
Predicted Class Probabilities: [[0.0398 0.0099 0.0249 0.0997 0.0461 0.7693 0.0062 0.0018 0.002
Actual Class : 6
-----
126 Text feature [e2] present in test data point [True]
132 Text feature [resistance] present in test data point [True]
151 Text feature [brca1] present in test data point [True]
158 Text feature [deleterious] present in test data point [True]
173 Text feature [ovarian] present in test data point [True]
195 Text feature [i26a] present in test data point [True]
197 Text feature [e2 interaction] present in test data point [True]
235 Text feature [resistance associated] present in test data point [True]
238 Text feature [pancreatic] present in test data point [True]
266 Text feature [bard1] present in test data point [True]
267 Text feature [brca1] present in test data point [True]
269 Text feature [wapcre] present in test data point [True]
274 Text feature [brca1flex2] present in test data point [True]
277 Text feature [polymorphism] present in test data point [True]
278 Text feature [mutant enriched] present in test data point [True]
285 Text feature [substrate binding] present in test data point [True]
319 Text feature [cell tumors] present in test data point [True]
326 Text feature [ligase activity] present in test data point [True]
327 Text feature [type p53] present in test data point [True]
355 Text feature [ovarian cancer] present in test data point [True]
363 Text feature [brca1fh] present in test data point [True]
368 Text feature [brca1 e2] present in test data point [True]
372 Text feature [kinase] present in test data point [True]
376 Text feature [resistance alleles] present in test data point [True]
378 Text feature [brca1 bard1] present in test data point [True]
394 Text feature [family history] present in test data point [True]
397 Text feature [zn2] present in test data point [True]
400 Text feature [p53ls1] present in test data point [True]
401 Text feature [p53ls1 r270h] present in test data point [True]
403 Text feature [s1598f] present in test data point [True]
407 Text feature [r270h] present in test data point [True]
408 Text feature [history] present in test data point [True]
415 Text feature [met mutations] present in test data point [True]
417 Text feature [diverse mutant] present in test data point [True]
419 Text feature [low copy] present in test data point [True]
424 Text feature [deleterious mutation] present in test data point [True]
425 Text feature [brca1 mutated] present in test data point [True]
426 Text feature [ability induce] present in test data point [True]
428 Text feature [s2] present in test data point [True]
432 Text feature [r270h wapcre] present in test data point [True]
433 Text feature [p53flex7] present in test data point [True]
436 Text feature [low penetrance] present in test data point [True]

```

```

445 Text feature [interaction] present in test data point [True]
447 Text feature [substitutions] present in test data point [True]
455 Text feature [t50] present in test data point [True]
456 Text feature [penetrance] present in test data point [True]
460 Text feature [inhibitors] present in test data point [True]
462 Text feature [p53flex7 flex7] present in test data point [True]
463 Text feature [flex7] present in test data point [True]
466 Text feature [tumor suppression] present in test data point [True]
472 Text feature [ubch5a] present in test data point [True]
474 Text feature [substitution] present in test data point [True]
483 Text feature [binding site] present in test data point [True]
484 Text feature [ligase] present in test data point [True]
488 Text feature [ubiquitin] present in test data point [True]
497 Text feature [cases controls] present in test data point [True]
Out of the top 500 features 56 are present in query point

```

```

In [152]: test_point_index = 51
no_feature = 500
predicted_cls = sig_clf.predict(test_x_mix[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_mix[test_point_index]), 3))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_imptfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index], test_df['Gene'])

```

```

Predicted Class : 4
Predicted Class Probabilities: [[0.4728 0.0185 0.0045 0.4736 0.009 0.0031 0.0142 0.0021 0.0021 0.0021]
Actual Class : 1
-----
```

```

0 Text feature [brca1] present in test data point [True]
220 Text feature [protein] present in test data point [True]
237 Text feature [pten] present in test data point [True]
239 Text feature [missense] present in test data point [True]
262 Text feature [brct] present in test data point [True]
266 Text feature [p53 cancer] present in test data point [True]
270 Text feature [mice] present in test data point [True]
272 Text feature [mm] present in test data point [True]
295 Text feature [atpase] present in test data point [True]
303 Text feature [p53 mutants] present in test data point [True]
309 Text feature [dn] present in test data point [True]
311 Text feature [rad50] present in test data point [True]
321 Text feature [atm] present in test data point [True]
333 Text feature [recombination] present in test data point [True]
336 Text feature [repair] present in test data point [True]
343 Text feature [activity] present in test data point [True]
359 Text feature [cancer mutants] present in test data point [True]
```

```
360 Text feature [nuclear] present in test data point [True]
367 Text feature [muts] present in test data point [True]
374 Text feature [loss] present in test data point [True]
413 Text feature [functional] present in test data point [True]
421 Text feature [tp53] present in test data point [True]
422 Text feature [tumours] present in test data point [True]
424 Text feature [yeast] present in test data point [True]
430 Text feature [mutants] present in test data point [True]
431 Text feature [breast ovarian] present in test data point [True]
436 Text feature [es] present in test data point [True]
440 Text feature [families] present in test data point [True]
441 Text feature [carriers] present in test data point [True]
444 Text feature [function] present in test data point [True]
454 Text feature [es cells] present in test data point [True]
470 Text feature [hr] present in test data point [True]
476 Text feature [g1 checkpoint] present in test data point [True]
486 Text feature [mre11] present in test data point [True]
489 Text feature [instability] present in test data point [True]
499 Text feature [ovarian cancer] present in test data point [True]
Out of the top 500 features 36 are present in query point
```