

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 Sources and some Useful Links

1. <https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25> (<https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25>)
2. <https://www.youtube.com/watch?v=UwbuW7oK8rk> (<https://www.youtube.com/watch?v=UwbuW7oK8rk>)
3. <https://www.youtube.com/watch?v=qxXRKVompl8> (<https://www.youtube.com/watch?v=qxXRKVompl8>)

1.3. Real-world/Business objectives and constraints.

1. Its a task of classification of each data points into one of nine cancer classes correctly.
2. With the Prediction of label we also need probability of each data point belonging to any of the 9 classes.
3. Errors can be very costly.
4. Interpretability is also important.
5. Make the CV log-Loss and Test Log-Loss < 1.0

2. Machine Learning Problem Formulation

2.1. Data

2.1.1 Data Overview

- Source: <https://www.kaggle.com/c/msk-redefining-cancer-treatment/data> (<https://www.kaggle.com/c/msk-redefining-cancer-treatment/data>)
- We have two data files: one 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:
 - training_variants (ID , Gene, Variations, Class)
 - training_text (ID, Text)

2.1.2. Example Data Point

training_variants

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

training_text

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

2.2 Mapping the real world problem to an ML problem

2.2.1 Type of Problem

There are 9 different Classes a Genetic Mutation can be classified into i.e Multiclass Classification Problem.

2.2.2 Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

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

from mlxtend.classifier import StackingClassifier

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

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 mutations used for training.
Fields are

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

3.1.2. Reading Text Data

```
In [194]: # note the separator in this file
data_text = pd.read_csv("training_text", sep="\\|\\|", engine="python", names=["ID", "TEXT"], skiprows=1)
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[194]:

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

3.1.3. Preprocessing of text

```
In [176]: # loading stop words from nltk library
stop_words = set(stopwords.words('english'))

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

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

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

```
In [177]: #text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_time, "seconds")

Time took for preprocessing the text : 1965.2896444840062 seconds
```

merging both gene_variations and text data based on ID

```
In [178]: result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[178]:

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

```
In [179]: result["text_variation"]=result["TEXT"]+result["Variation"]
```

```
In [195]: result['text_backup']=data_text["TEXT"]
```

```
In [181]: result["gene_variation"]=result["Gene"]+" "+result["Variation"]
```

```
In [182]: #feature 1 is Number of Words in Text column
result['Feature_1']= result["text_backup"].apply(lambda x: len(str(x).split()))
```

```
In [183]: #Feature 2 - Count of characters in a statement
result['Feature_2'] = result["text_backup"].apply(lambda x: len(str(x)))
```

```
In [184]: #Feature 3-Diversity_score i.e. Average Length of words used in statement
result['Feature_3'] = result['Feature_2'] / result['Feature_1']
```

```
In [197]: result['Variation_Share'] = result.apply(lambda r: sum([1 for w in r['Variation'].split(' ') if w
in r['text_backup'].split(' ')]), axis=1)
result["Variation_Share"].value_counts()
```

```
Out[197]: 1    1672
0    1577
2     58
3     10
5      2
4      2
Name: Variation_Share, dtype: int64
```

```
In [198]: result.drop("text_backup",axis=1,inplace=True)
```

```
In [200]: result.head()
```

```
Out[200]:
```

	ID	Gene	Variation	Class	TEXT	gene_variation	Feature_1	Feature_2	Feature_3	Gene_Share	V
0	0	FAM58A	Truncating Mutations	1	cyclin dependent kinases cdk regulate variety...	FAM58A Truncating Mutations	4370	30836	7.056293	1	1
1	1	CBL	W802*	2	abstract background non small cell lung cancer...	CBL W802*	4139	27844	6.727229	1	1
2	2	CBL	Q249E	2	abstract background non small cell lung cancer...	CBL Q249E	4139	27844	6.727229	1	1
3	3	CBL	N454D	3	recent evidence demonstrated acquired uniparen...	CBL N454D	3841	28093	7.313981	1	1
4	4	CBL	L399V	4	oncogenic mutations monomeric casitas b lineag...	CBL L399V	4254	31649	7.439821	1	1

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 [201]: 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 'y_true'
[stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true, test_size=0.2)

# split the train data into train and cross validation by maintaining same distribution of output
variable 'y_train' [stratify=y_train]
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test_size=0.2)
```

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

```
In [202]: 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

```

In [203]: # 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 = 'rgbkymc'
train_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.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], '(', np.round(
        (train_class_distribution.values[i]/train_df.shape[0]*100), 3), '%)')

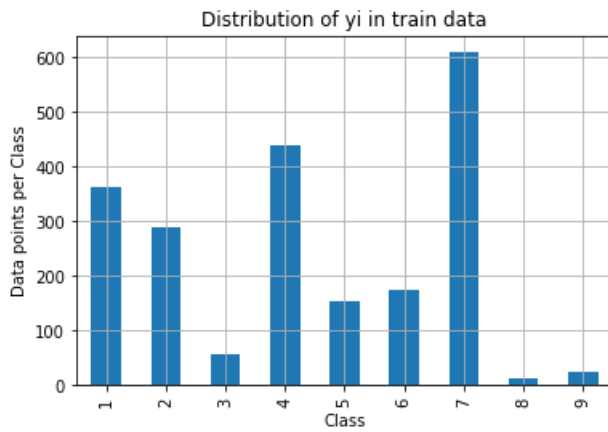
print('-'*80)
my_colors = 'rgbkymc'
test_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.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], '(', np.round(
        (test_class_distribution.values[i]/test_df.shape[0]*100), 3), '%)')

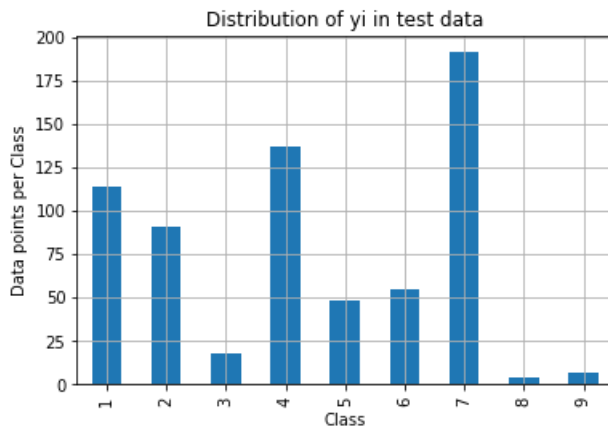
print('-'*80)
my_colors = 'rgbkymc'
cv_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.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], '(', np.round(
        (cv_class_distribution.values[i]/cv_df.shape[0]*100), 3), '%)')

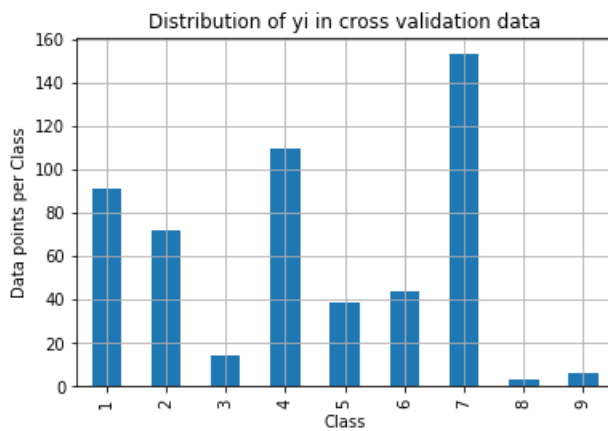
```

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



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



```

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

```

3.2 Prediction using a 'Random' Model

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

```

In [204]: # 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 predicted class j

    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 two dimensi
onal array
    # 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 two dimensi
onal array
    # 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 (Column 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 [205]: # we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to generate 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, eps=1e-15))

# 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=1e-15))

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 2.493666690798053

Log loss on Test Data using Random Model 2.445183264544613

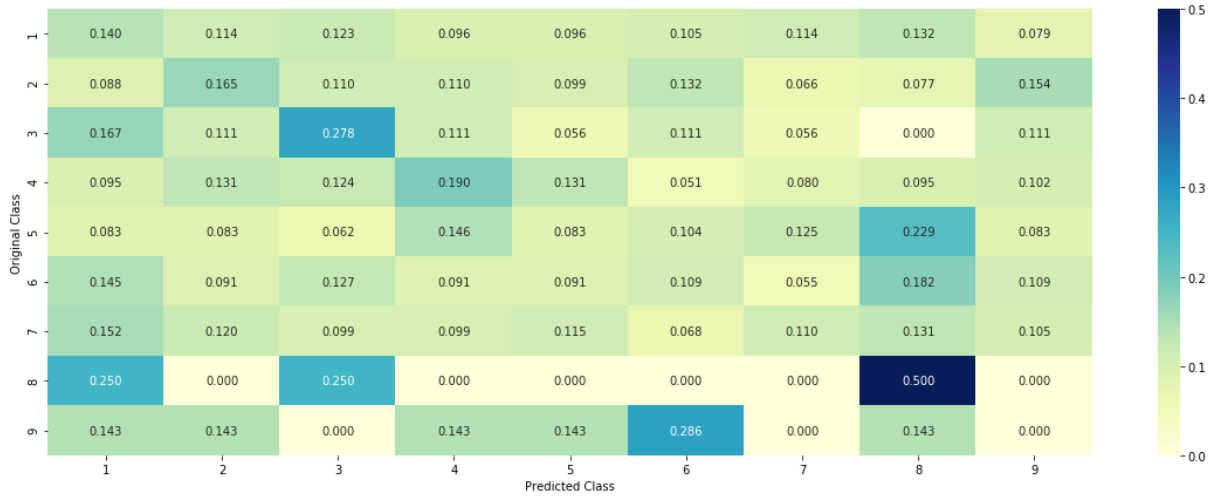
----- Confusion matrix -----



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



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



3.3 Univariate Analysis

```

In [206]: # 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 data dataframe
# build a vector (1*9) , the first element = (number of times it occurred in class1 + 10*alpha / number of times it occurred in total data+90*alpha)
# gv_dict is like a look up table, for every gene it store a (1*9) representation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# -----

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

    # denominator will contain the number of times that particular feature occurred in whole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to particular class
        # 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 times that particular feature
            occurred in whole data
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))

        # we are adding the gene/variation to the dict as key and vec as value

```

```

        gv_dict[i]=vec
    return gv_dict

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    # {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.0681818181818177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.03787878787878788, 0.03787878787878788, 0.03787878787878788],

    # 'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.27040816326530615, 0.061224489795918366, 0.066326530612244902, 0.051020408163265307, 0.051020408163265307, 0.056122448979591837],

    # 'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.0681818181818177, 0.0681818181818177, 0.0625, 0.34659090909090912, 0.0625, 0.056818181818181816],

    # 'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.078787878787878782, 0.1393939393939394, 0.34545454545454546, 0.060606060606060608, 0.060606060606060608, 0.060606060606060608],

    # 'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.46540880503144655, 0.075471698113207544, 0.062893081761006289, 0.069182389937106917, 0.062893081761006289, 0.062893081761006289],

    # 'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.072847682119205295, 0.066225165562913912, 0.066225165562913912, 0.27152317880794702, 0.066225165562913912, 0.066225165562913912],

    # 'BRAF': [0.066666666666666666, 0.17999999999999999, 0.07333333333333334, 0.07333333333333334, 0.09333333333333338, 0.08000000000000002, 0.29999999999999999, 0.066666666666666666, 0.066666666666666666],

    # ...
    # }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

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

```

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

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

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?

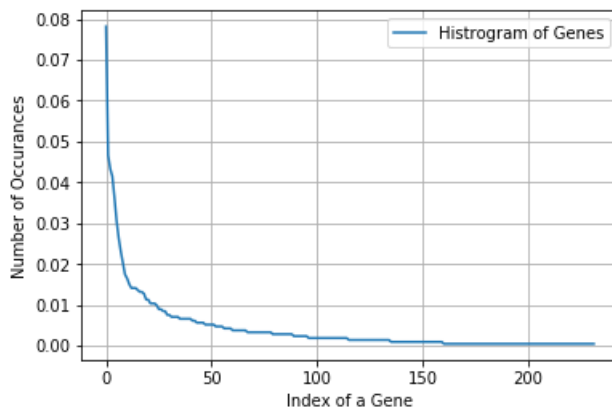
```
In [207]: unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occurred most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 232
BRCA1      166
TP53        99
EGFR        92
PTEN        88
BRCA2       77
KIT          65
BRAF         56
ERBB2        49
ALK          43
PDGFRA       37
Name: Gene, dtype: int64
```

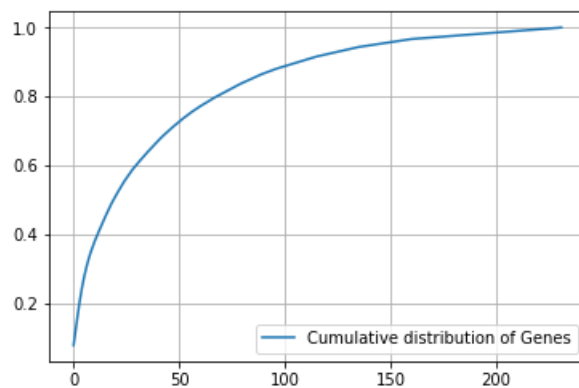
```
In [208]: print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the train data, a
nd they are distributed as follows",)
```

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

```
In [209]: 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 Occurances')
plt.legend()
plt.grid()
plt.show()
```



```
In [210]: c = np.cumsum(h)
plt.plot(c, label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

Ans.there are two ways we can featurize this variable check out this video:

<https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/>

1. One hot Encoding
2. Response coding

We will choose the One hot Encoding featurization based here for now. 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 [211]: # 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 [212]: cv_gene_feature_onehotCoding.shape
```

```
Out[212]: (532, 231)
```

```
In [213]: print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature:", train_gene_feature_onehotCoding.shape)
```

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

```
In [214]: # creating a dataframe of the vectorized features
```

```
df_gene_train = pd.DataFrame(train_gene_feature_onehotCoding.toarray(), columns=gene_vectorizer.get_feature_names())  
df_gene_test = pd.DataFrame(test_gene_feature_onehotCoding.toarray(), columns=gene_vectorizer.get_feature_names())  
df_gene_cv = pd.DataFrame(cv_gene_feature_onehotCoding.toarray(), columns=gene_vectorizer.get_feature_names())
```

How Good is gene feature in predicting y_i


```

In [215]: 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/sklearn.linear_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# 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))

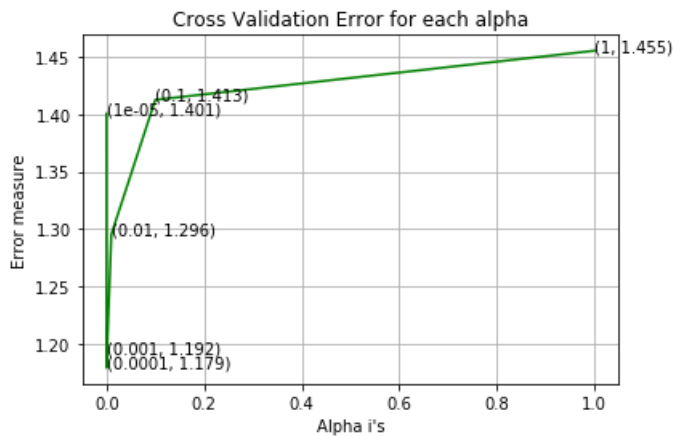
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], np.round(txt, 3)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error 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_train, 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_test, predict_y, labels=clf.classes_, eps=1e-15))

```

For values of alpha = 1e-05 The log loss is: 1.4005555527470852
 For values of alpha = 0.0001 The log loss is: 1.179340161820206
 For values of alpha = 0.001 The log loss is: 1.1920606034079948
 For values of alpha = 0.01 The log loss is: 1.2955107368017145
 For values of alpha = 0.1 The log loss is: 1.412544823106519
 For values of alpha = 1 The log loss is: 1.4552582366725741



For values of best alpha = 0.0001 The train log loss is: 1.0415595201966887
 For values of best alpha = 0.0001 The cross validation log loss is: 1.179340161820206
 For values of best alpha = 0.0001 The test log loss is: 1.2086527043791122

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?

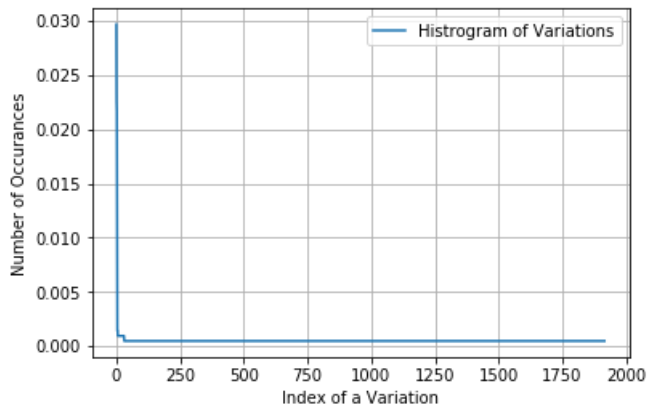
```
In [216]: unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occurred most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1916
Truncating_Mutations    63
Deletion                 48
Amplification            46
Fusions                  26
E17K                     3
Overexpression           3
S308A                    2
V321M                    2
Q61R                     2
Q61K                     2
Name: Variation, dtype: int64
```

```
In [217]: print("Ans: There are", unique_variations.shape[0], "different categories of variations in the tra
in data, and they are distributed as follows",)
```

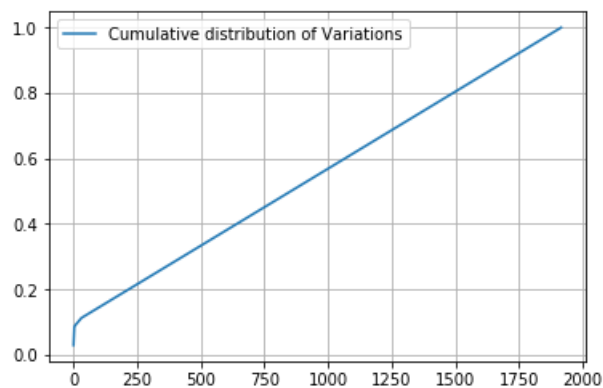
Ans: There are 1916 different categories of variations in the train data, and they are distributed as follows

```
In [218]: 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 Occurances')
plt.legend()
plt.grid()
plt.show()
```



```
In [219]: c = np.cumsum(h)
print(c)
plt.plot(c, label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()

[0.02966102 0.05225989 0.07391714 ... 0.99905838 0.99952919 1.          ]
```



Q9. How to featurize this Variation feature ?

Ans. There are two ways we can featurize this variable check out this video:

<https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/>

1. One hot Encoding
2. Response coding

We will be using One hot Encoding method to featurize the Variation Feature

```
In [220]: # 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'])
```

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 [221]: print("Q12. How many data points are covered by total ", unique_variations.shape[0], " genes in test and cross validation data sets?")
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])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":", (cv_coverage/cv_df.shape[0])*100)
```

Q12. How many data points are covered by total 1916 genes in test and cross validation data sets?

Ans

1. In test data 59 out of 665 : 8.87218045112782

2. In cross validation data 49 out of 532 : 9.210526315789473

```
In [222]: df_var_train = pd.DataFrame(train_variation_feature_onehotCoding.toarray(), columns=variation_vectorizer.get_feature_names())
df_var_test = pd.DataFrame(test_variation_feature_onehotCoding.toarray(), columns=variation_vectorizer.get_feature_names())
df_var_cv = pd.DataFrame(cv_variation_feature_onehotCoding.toarray(), columns=variation_vectorizer.get_feature_names())
```

```

In [223]: 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
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
# =0.0, power_t=0.5,
# 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_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))

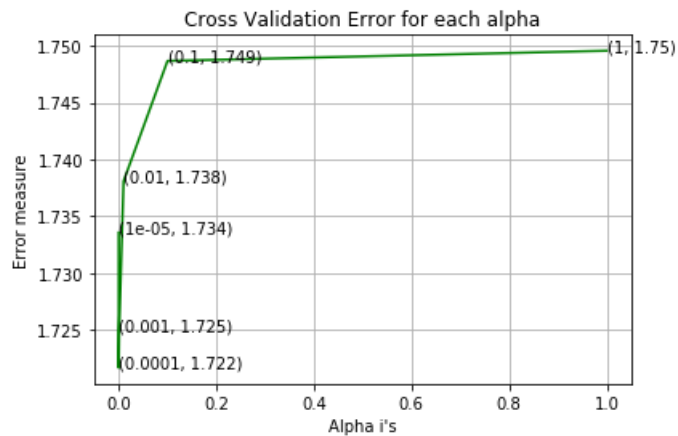
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], np.round(txt, 3)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error 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, labels=clf.classes_, eps=1e-15))
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(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
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(y_test, predict_y, labels=clf.classes_, eps=1e-15))

```

For values of alpha = $1e-05$ The log loss is: 1.7335721680090286
 For values of alpha = 0.0001 The log loss is: 1.72166551571151
 For values of alpha = 0.001 The log loss is: 1.7249727158308956
 For values of alpha = 0.01 The log loss is: 1.7380948794689282
 For values of alpha = 0.1 The log loss is: 1.748669375869514
 For values of alpha = 1 The log loss is: 1.7495674317353667



For values of best alpha = 0.0001 The train log loss is: 0.7456583822237609
 For values of best alpha = 0.0001 The cross validation log loss is: 1.72166551571151
 For values of best alpha = 0.0001 The test log loss is: 1.7242034752511008

3.2.3 Univariate analysis on number of words feature

```

In [224]: alpha = [10 ** x for x in range(-5, 1)]

cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_df["Feature_1"].reshape(-1,1), y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_df.Feature_1.reshape(-1,1), y_train)
    predict_y = sig_clf.predict_proba(cv_df.Feature_1.reshape(-1,1))

    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_df.Feature_1.reshape(-1,1), y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_df.Feature_1.reshape(-1,1), y_train)

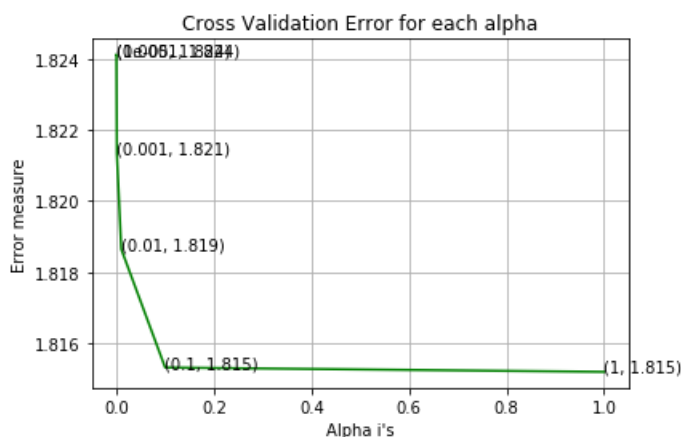
predict_y = sig_clf.predict_proba(train_df.Feature_1.reshape(-1,1))
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_df.Feature_1.reshape(-1,1))
print('For values of best 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_df.Feature_1.reshape(-1,1))
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

```

```

For values of alpha = 1e-05 The log loss is: 1.8241096612789325
For values of alpha = 0.0001 The log loss is: 1.8241096612794818
For values of alpha = 0.001 The log loss is: 1.8213303970404182
For values of alpha = 0.01 The log loss is: 1.8186476370872977
For values of alpha = 0.1 The log loss is: 1.8153239532039447
For values of alpha = 1 The log loss is: 1.8152017039678854

```



```

For values of best alpha = 1 The train log loss is: 1.8145762602898243
For values of best alpha = 1 The cross validation log loss is: 1.8152017039678854
For values of best alpha = 1 The test log loss is: 1.8027729211835348

```

3.2.4 Univariate analysis on gene-variation feature

```
In [225]: gene_variation_vectorizer = CountVectorizer()
train_gene_and_variation_feature_onehotCoding = gene_variation_vectorizer.fit_transform(train_df[
"gene_variation"])
test_gene_and_variation_feature_onehotCoding = gene_variation_vectorizer.transform(test_df["gene_v
ariation"])
cv_gene_and_variation_feature_onehotCoding = gene_variation_vectorizer.transform(cv_df["gene_varia
tion"])
```

```
In [226]: df_geneandvar_train = pd.DataFrame(train_gene_and_variation_feature_onehotCoding.toarray(), column
s=gene_variation_vectorizer.get_feature_names())
df_geneandvar_test = pd.DataFrame(test_gene_and_variation_feature_onehotCoding.toarray(), columns=
gene_variation_vectorizer.get_feature_names())
df_geneandvar_cv = pd.DataFrame(cv_gene_and_variation_feature_onehotCoding.toarray(), columns=gene
_variation_vectorizer.get_feature_names())
```



```
In [227]: alpha = [10 ** x for x in range(-5, 1)]

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

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

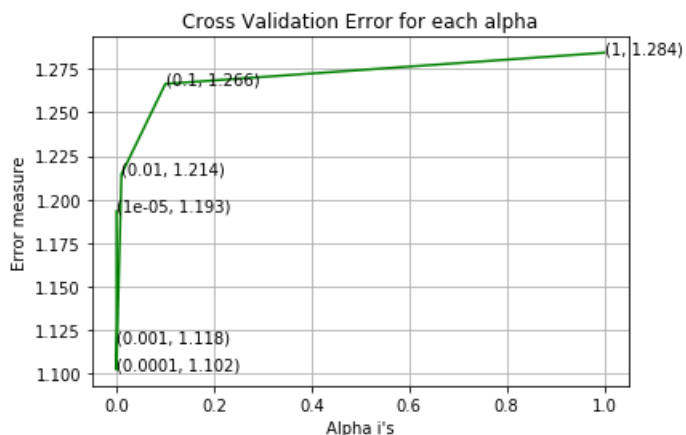
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))

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

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

```
For values of alpha = 1e-05 The log loss is: 1.1932901215272576
For values of alpha = 0.0001 The log loss is: 1.102290132342943
For values of alpha = 0.001 The log loss is: 1.1181306832005078
For values of alpha = 0.01 The log loss is: 1.2144516238587448
For values of alpha = 0.1 The log loss is: 1.2662628626166113
For values of alpha = 1 The log loss is: 1.2841147606722276
```



```
For values of best alpha = 0.0001 The train log loss is: 0.5215962530649423
For values of best alpha = 0.0001 The cross validation log loss is: 1.102290132342943
For values of best alpha = 0.0001 The test log loss is: 1.1352915779532327
```

3.2.5 Univariate analysis on Number of character

```

In [228]: alpha = [10 ** x for x in range(-5, 1)]

cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_df["Feature_2"].reshape(-1,1), y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_df.Feature_2.reshape(-1,1), y_train)
    predict_y = sig_clf.predict_proba(cv_df.Feature_2.reshape(-1,1))

    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.cla
sses_, 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_df.Feature_2.reshape(-1,1), y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_df.Feature_2.reshape(-1,1), y_train)

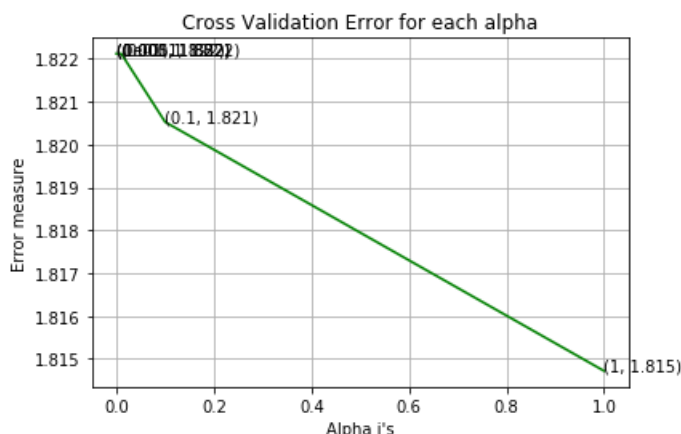
predict_y = sig_clf.predict_proba(train_df.Feature_2.reshape(-1,1))
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_df.Feature_2.reshape(-1,1))
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_df.Feature_2.reshape(-1,1))
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))

```

```

For values of alpha = 1e-05 The log loss is: 1.822120203214765
For values of alpha = 0.0001 The log loss is: 1.8221202031867352
For values of alpha = 0.001 The log loss is: 1.8221202032096566
For values of alpha = 0.01 The log loss is: 1.822120203210052
For values of alpha = 0.1 The log loss is: 1.820527135525177
For values of alpha = 1 The log loss is: 1.8147197156211177

```



```

For values of best alpha = 1 The train log loss is: 1.8157875187232202
For values of best alpha = 1 The cross validation log loss is: 1.8147197156211177
For values of best alpha = 1 The test log loss is: 1.805133499817863

```

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 [229]: # 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 [230]: 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)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEXT'].split
            )))
        row_index += 1
    return text_feature_responseCoding
```

```
In [267]: #building a CountVectorizer with all the words that occurred minimum 3 times in train data

text_vectorizer = TfidfVectorizer(ngram_range=(1,4),max_features=2000)
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 features) vector
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 : 2000
```

```
In [268]: train_text_fea_counts.shape
```

```
Out[268]: (2000,)
```

```

In [269]: 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 [270]: # don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

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

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

```

```

In [271]: #https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))

```

```

In [272]: print(train_text_feature_onehotCoding.shape)
print(test_text_feature_onehotCoding.shape)
print(cv_text_feature_onehotCoding.shape)

(2124, 2000)
(665, 2000)
(532, 2000)

```

```

In [273]: # Train a Logistic regression+Calibration model using text features which are on-hot encoded
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
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# 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_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))

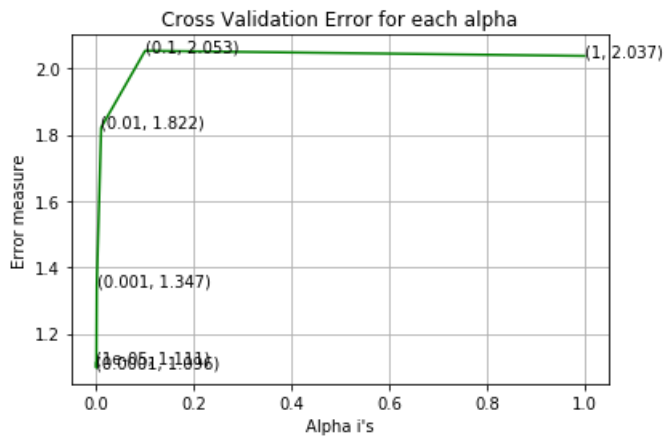
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], np.round(txt, 3)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error 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, labels=clf.classes_, eps=1e-15))
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, labels=clf.classes_, eps=1e-15))
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, labels=clf.classes_, eps=1e-15))

```

For values of alpha = 1e-05 The log loss is: 1.1105363411250055
 For values of alpha = 0.0001 The log loss is: 1.0964906815887412
 For values of alpha = 0.001 The log loss is: 1.346817068493882
 For values of alpha = 0.01 The log loss is: 1.8215906927561887
 For values of alpha = 0.1 The log loss is: 2.0530451970564907
 For values of alpha = 1 The log loss is: 2.0373670509665978



For values of best alpha = 0.0001 The train log loss is: 0.7467683268790596
 For values of best alpha = 0.0001 The cross validation log loss is: 1.0964906815887412
 For values of best alpha = 0.0001 The test log loss is: 1.1013255697297029

```
In [274]: df_text_train = pd.DataFrame(train_text_feature_onehotCoding.toarray(), columns=text_vectorizer.get_feature_names())
df_text_test = pd.DataFrame(test_text_feature_onehotCoding.toarray(), columns=text_vectorizer.get_feature_names())
df_text_cv = pd.DataFrame(cv_text_feature_onehotCoding.toarray(), columns=text_vectorizer.get_feature_names())
```

4. Machine Learning Models

```
In [275]: #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 each class
    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)
```

```
In [276]: 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 [277]: # this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3)

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

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

    word_present = 0
    for i,v in enumerate(indices):
        if (v < fea1_len):
            word = gene_vec.get_feature_names()[v]
            yes_no = True if word == gene else False
            if yes_no:
                word_present += 1
                print(i, "Gene feature [{}]" .format(word,yes_no))
        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 [{}]" .format(word,yes_
no))
        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 [{}]" .format(word,yes_no))

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

```

Stacking all the features

```

In [278]: #target variables
train_y = train_df['Class'].values
test_y = test_df['Class'].values
cv_y = cv_df['Class'].values

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

df_gene_and_var_train = pd.concat([df_gene_var_train, df_geneandvar_train], axis=1)
df_gene_and_var_test = pd.concat([df_gene_var_test, df_geneandvar_test], axis=1)
df_gene_and_var_cv = pd.concat([df_gene_var_cv, df_geneandvar_cv], axis=1)

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

# scaling the text_count feature
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
train_df["Feature_1"] = scaler.fit_transform(train_df["Feature_1"].reshape(-1,1))
test_df["Feature_1"] = scaler.fit_transform(test_df["Feature_1"].reshape(-1,1))
cv_df["Feature_1"] = scaler.fit_transform(cv_df["Feature_1"].reshape(-1,1))

train_df["Feature_2"] = scaler.fit_transform(train_df["Feature_2"].reshape(-1,1))
test_df["Feature_2"] = scaler.fit_transform(test_df["Feature_2"].reshape(-1,1))
cv_df["Feature_2"] = scaler.fit_transform(cv_df["Feature_2"].reshape(-1,1))

train_df["Feature_3"] = scaler.fit_transform(train_df["Feature_3"].reshape(-1,1))
test_df["Feature_3"] = scaler.fit_transform(test_df["Feature_3"].reshape(-1,1))
cv_df["Feature_3"] = scaler.fit_transform(cv_df["Feature_3"].reshape(-1,1))

df_train["Gene_Share"] = train_df.Gene_Share.values
df_train["Variation_Share"] = train_df.Variation_Share.values
df_train["Feature_1"] = train_df.Feature_1.values
df_train["Feature_2"] = train_df.Feature_2.values
df_train["Feature_3"] = train_df.Feature_3.values

df_test["Gene_Share"] = test_df.Gene_Share.values
df_test["Variation_Share"] = test_df.Variation_Share.values
df_test["Feature_1"] = test_df.Feature_1.values
df_test["Feature_2"] = test_df.Feature_2.values
df_test["Feature_3"] = test_df.Feature_3.values

df_cv["Gene_Share"] = cv_df.Gene_Share.values
df_cv["Variation_Share"] = cv_df.Variation_Share.values
df_cv["Feature_1"] = cv_df.Feature_1.values
df_cv["Feature_2"] = cv_df.Feature_2.values
df_cv["Feature_3"] = cv_df.Feature_3.values

train_x_onehotCoding = df_train
test_x_onehotCoding = df_test
cv_x_onehotCoding = df_cv

```

```

In [320]: print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_onehotCoding
.shape)

```

```

One hot encoding features :
(number of data points * number of features) in train data = (2124, 6327)
(number of data points * number of features) in test data = (665, 6327)
(number of data points * number of features) in cross validation data = (532, 6327)

```

4.1. Base Line Model

4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

```

In [280]: # find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive\_bayes.MultinomialNB.html
# -----
# default parameters
# 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.applieidaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algorithm-1/
# -----

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# -----
# default parameters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# 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.applieidaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algorithm-1/
# -----

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 probabilities we use log-probability estimates
    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)

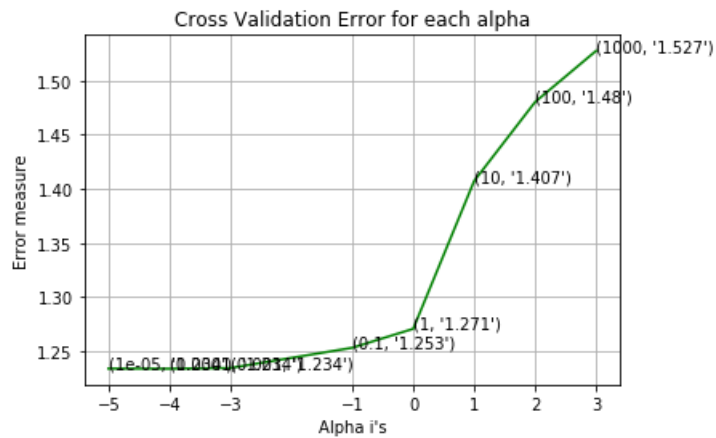
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha],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

```

```

for alpha = 1e-05
Log Loss : 1.233816172023287
for alpha = 0.0001
Log Loss : 1.2337083709494239
for alpha = 0.001
Log Loss : 1.2343450081698089
for alpha = 0.1
Log Loss : 1.253028536291359
for alpha = 1
Log Loss : 1.2705641127871883
for alpha = 10
Log Loss : 1.4071878034064196
for alpha = 100
Log Loss : 1.4801906371270999
for alpha = 1000
Log Loss : 1.5274644180242751

```



For values of best alpha = 0.0001 The train log loss is: 0.6218903432935438
 For values of best alpha = 0.0001 The cross validation log loss is: 1.2337083709494239
 For values of best alpha = 0.0001 The test log loss is: 1.2642411605579797

4.1.1.2. Testing the model with best hyper paramters

```

In [287]: # find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive\_bayes.MultinomialNB.html
# -----
# default parameters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

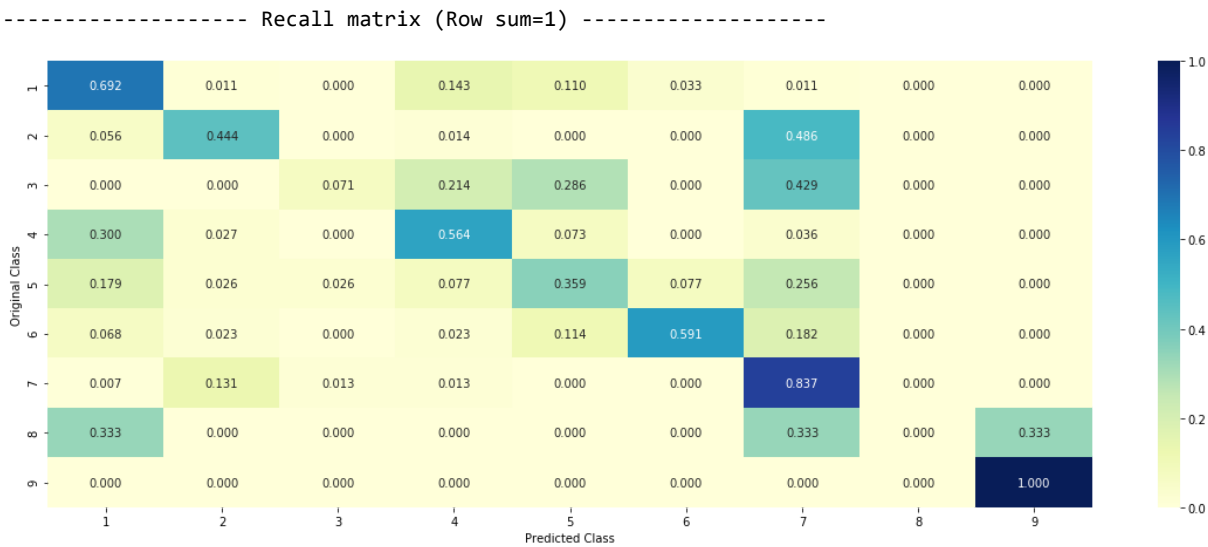
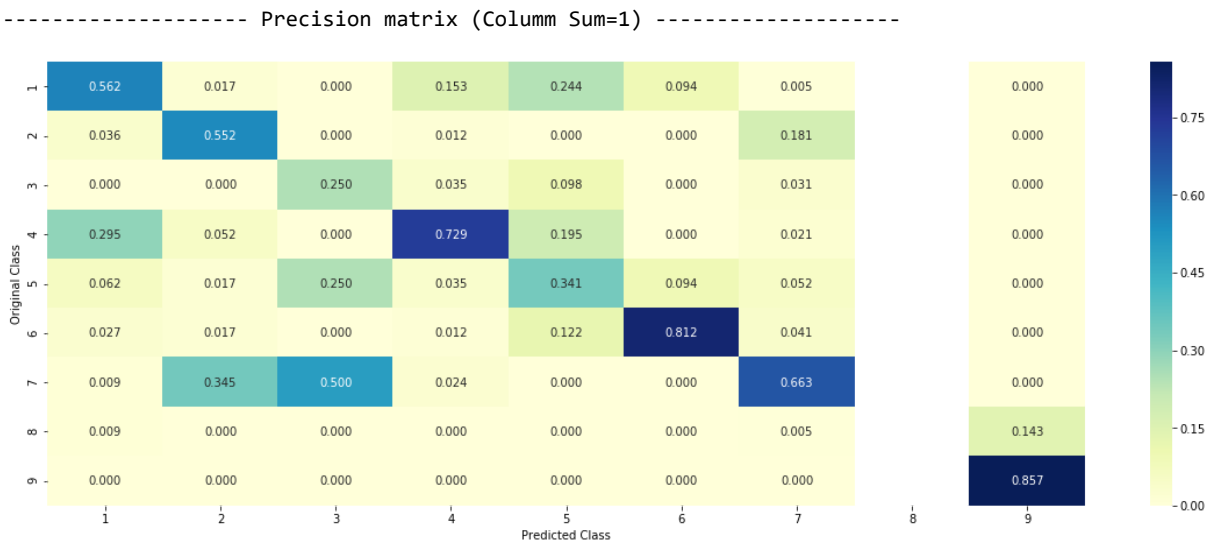
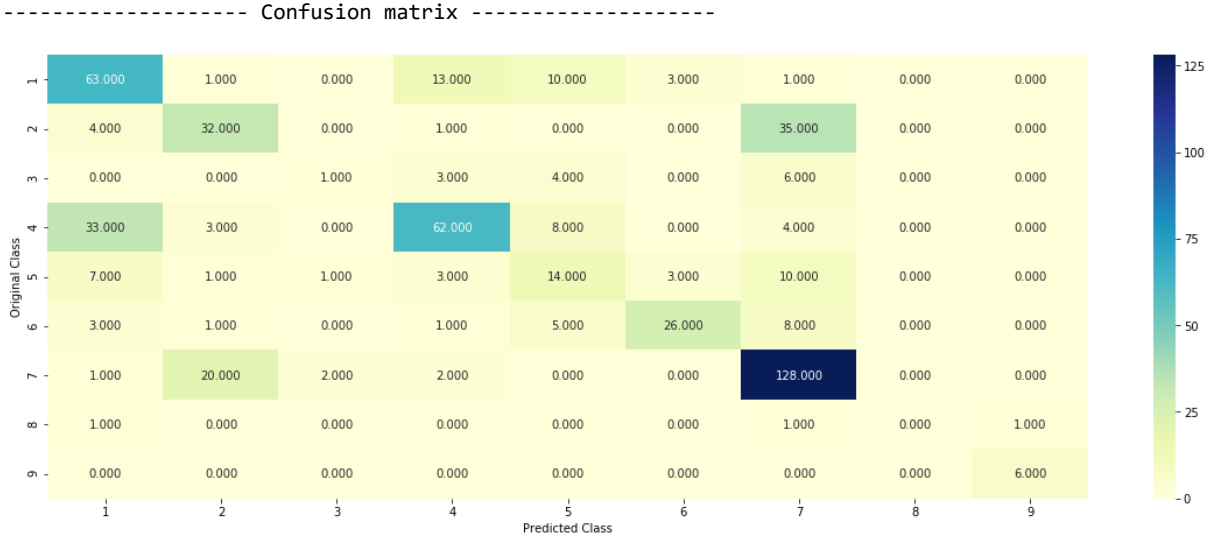
# some of methods of MultinomialNB()
# fit(X, y[, sample_weight]) Fit Naive Bayes classifier according to X, y
# predict(X) Perform classification on an array of test vectors X.
# predict_log_proba(X) Return log-probability estimates for the test vector X.
# -----
# video link: https://www.applieidaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algorithm-1/
# -----

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# -----
# default parameters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# 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 probabilities we use log-probability estimates
print("Log Loss :", log_loss(cv_y, sig_clf_probs))
print("Number of misclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotCoding) - cv_y)) / cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.as_matrix()))

```

Log Loss : 1.2337083709494239
Number of missclassified point : 0.37593984962406013



4.1.1.3. Feature Importance, Correctly classified point

```
In [289]: test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding.iloc[test_point_index, :].reshape(1, -1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding.iloc[test_point_index, :].reshape(1, -1)), 4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:, :no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index], test_df['Gene'].iloc[test_point_index], test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0558 0.0437 0.0091 0.6975 0.0316 0.0292 0.1271 0.0028 0.003 ]]
Actual Class : 4
-----
75 Text feature [47] present in test data point [True]
88 Text feature [42] present in test data point [True]
Out of the top 100 features 2 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

```
In [290]: test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding.iloc[test_point_index, :].reshape(1, -1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding.iloc[test_point_index, :].reshape(1, -1)), 4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:, :no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index], test_df['Gene'].iloc[test_point_index], test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.0899 0.0671 0.0139 0.1098 0.4708 0.0446 0.1952 0.0043 0.0045]]
Actual Class : 1
-----
90 Text feature [276300] present in test data point [True]
Out of the top 100 features 1 are present in query point
```

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

```

In [291]: # find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -----
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# 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/k-nearest-neighbors-geometric-intuition-with-a-toy-example-1/
#-----

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# -----
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# 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_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 probabilities we use log-probability estimates
    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_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

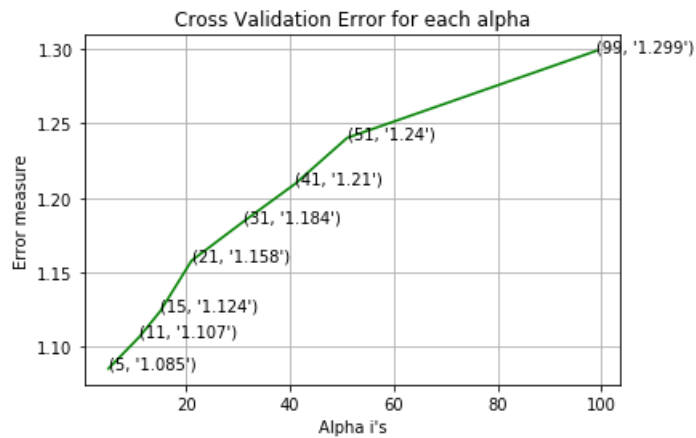
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
    predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
    ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
    redict_y, labels=clf.classes_, eps=1e-15))

```

```

for alpha = 5
Log Loss : 1.0852829601316618
for alpha = 11
Log Loss : 1.1068069609892106
for alpha = 15
Log Loss : 1.1239734112934245
for alpha = 21
Log Loss : 1.157792733394395
for alpha = 31
Log Loss : 1.1840769306698482
for alpha = 41
Log Loss : 1.2095648188645158
for alpha = 51
Log Loss : 1.2403831491653186
for alpha = 99
Log Loss : 1.2988340945993473

```



```

For values of best alpha = 5 The train log loss is: 0.8589002413672985
For values of best alpha = 5 The cross validation log loss is: 1.0852829601316618
For values of best alpha = 5 The test log loss is: 1.0951620282817154

```

4.2.2. Testing the model with best hyper paramters

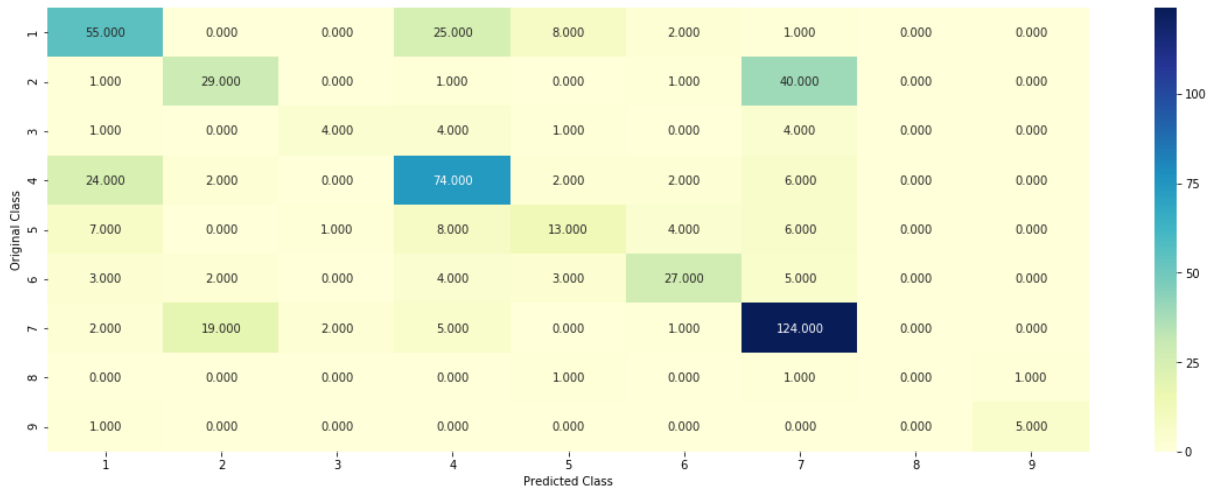

```
In [292]: # find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -----
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, 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/k-nearest-neighbors-geometric-intuition-with-a-toy-example-1/
#-----
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

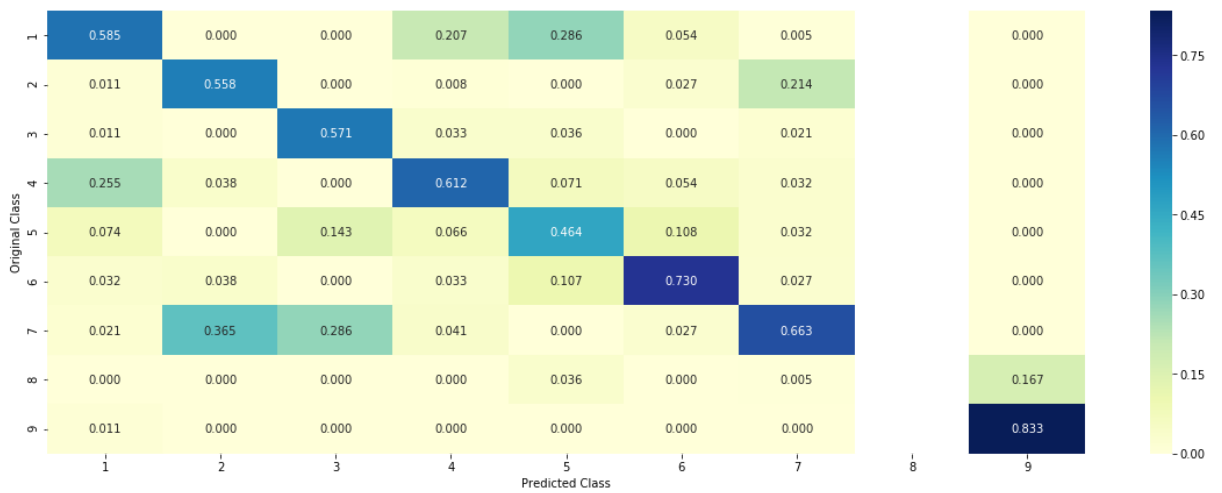
Log loss : 1.0852829601316618

Number of mis-classified points : 0.37781954887218044

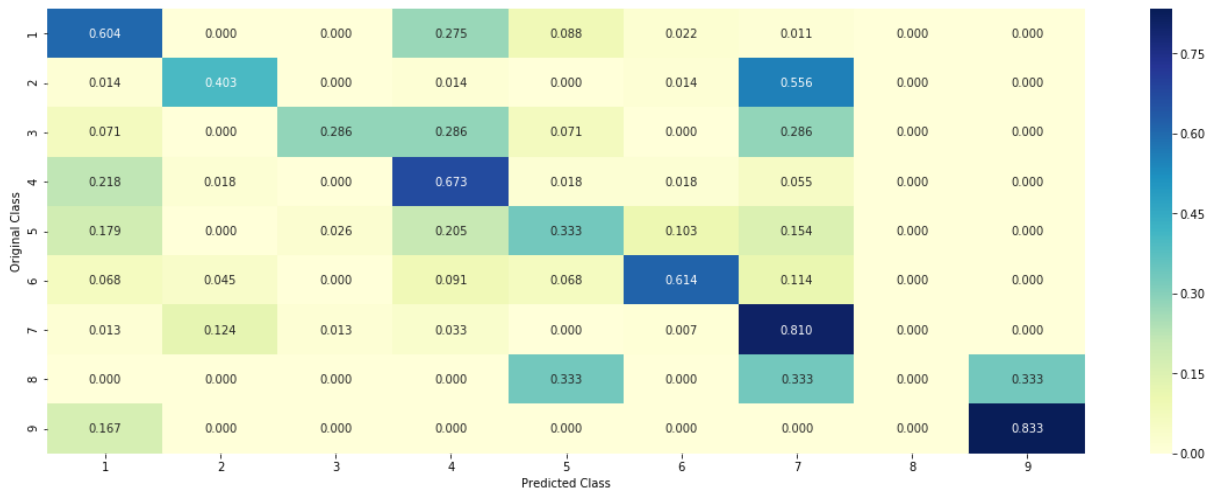
----- Confusion matrix -----



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



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



4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

```

In [298]: # 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, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# 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/geometric-intuition-1/
#-----

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# -----
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# 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 probabilities we use log-probability estimates
    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()

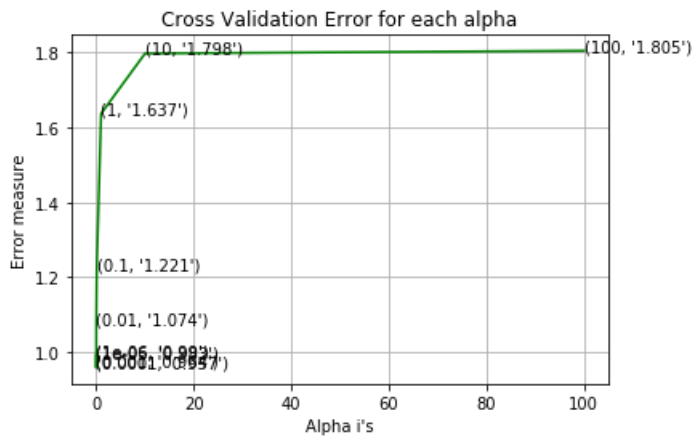
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```

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

```
for alpha = 1e-06
Log Loss : 0.9904781143992294
for alpha = 1e-05
Log Loss : 0.9832920905787479
for alpha = 0.0001
Log Loss : 0.9572675305625858
for alpha = 0.001
Log Loss : 0.9644174727158827
for alpha = 0.01
Log Loss : 1.0743182035485732
for alpha = 0.1
Log Loss : 1.2214073279561124
for alpha = 1
Log Loss : 1.6370128495942953
for alpha = 10
Log Loss : 1.797721121776716
for alpha = 100
Log Loss : 1.8051691959223746
```



```
For values of best alpha = 0.0001 The train log loss is: 0.4246640193264304
For values of best alpha = 0.0001 The cross validation log loss is: 0.9572675305625858
For values of best alpha = 0.0001 The test log loss is: 0.9994144760254319
```

4.3.1.2. Testing the model with best hyper paramters

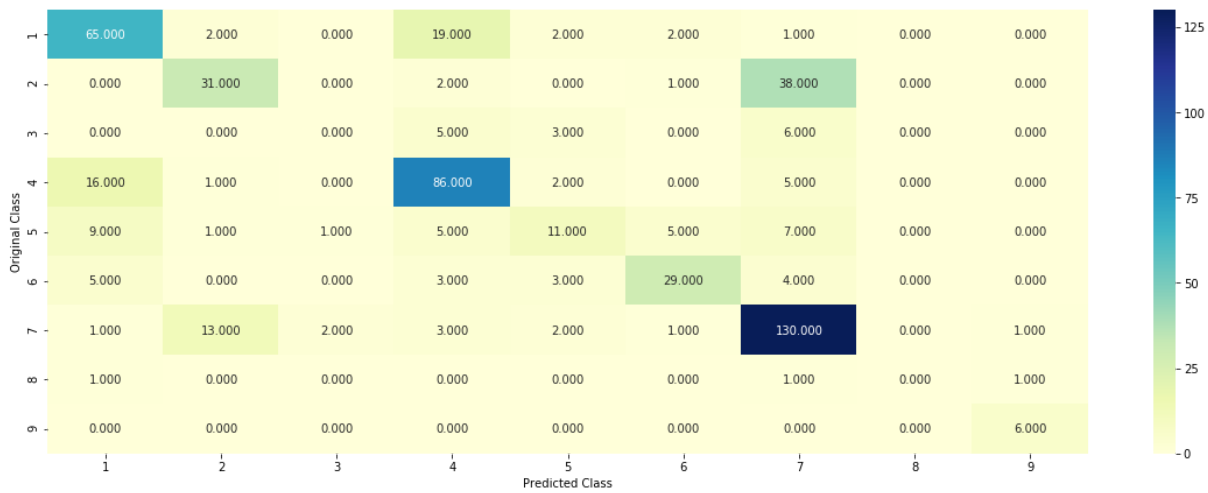
```
In [299]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# 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.

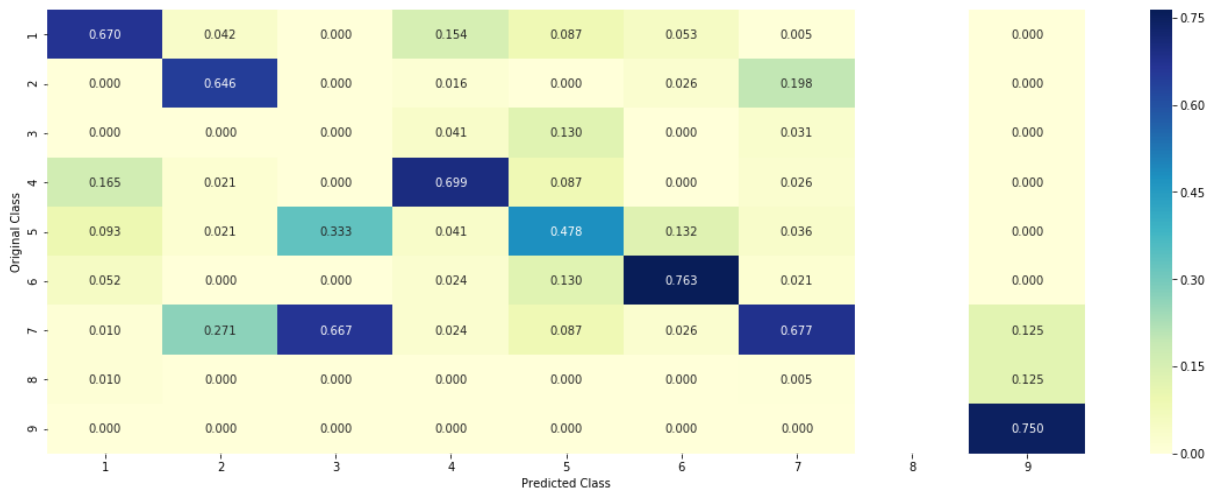
#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1/
#-----
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

Log loss : 0.9572675305625858
Number of mis-classified points : 0.32706766917293234

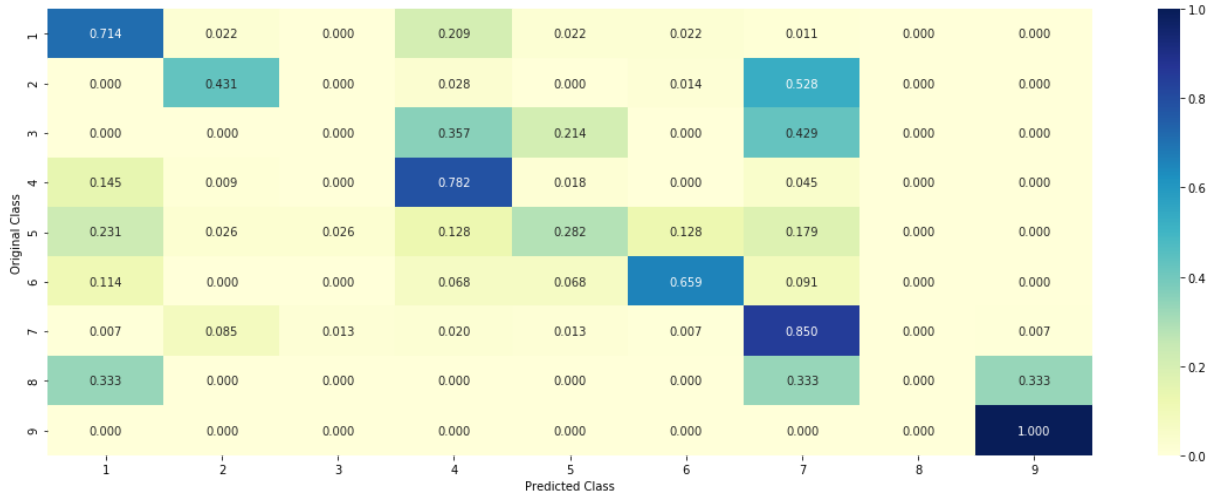
----- Confusion matrix -----



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



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



4.3.1.3. Feature Importance

```
In [300]: def get_imp_feature_names(text, indices, removed_ind = []):
word_present = 0
tabulte_list = []
increasingorder_ind = 0
for i in indices:
    if i < train_gene_feature_onehotCoding.shape[1]:
        tabulte_list.append([increasingorder_ind, "Gene", "Yes"])
    elif i < 18:
        tabulte_list.append([increasingorder_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([increasingorder_ind, train_text_features[i], yes_no])
        increasingorder_ind += 1
print(word_present, "most important features are present in our query point")
print("-"*50)
print("The features that are most important of the ", predicted_cls[0], " class:")
print (tabulate(tabulte_list, headers=["Index", 'Feature name', 'Present or Not']))
```

4.3.1.3.1. Correctly Classified point

```
In [302]: # from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', 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.iloc[test_point_index, :].reshape(1, -1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding.iloc[test_point_index, :].reshape(1, -1)), 4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:, :no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index], test_df['Gene'].iloc[test_point_index], test_df['Variation'].iloc[test_point_index], no_feature)

Predicted Class : 4
Predicted Class Probabilities: [[0.0153 0.0083 0.1122 0.8122 0.0204 0.0098 0.0174 0.0019 0.0025]]
Actual Class : 4
-----
25 Text feature [200] present in test data point [True]
449 Text feature [31] present in test data point [True]
460 Text feature [44] present in test data point [True]
Out of the top 500 features 3 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

```

In [303]: test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding.iloc[test_point_index, :].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding.iloc[test_point_index, :].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:, :no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index], test_df['Gene'].iloc[test_point_index], test_df['Variation'].iloc[test_point_index], no_feature)

Predicted Class : 5
Predicted Class Probabilities: [[0.1418 0.0297 0.0068 0.2824 0.4927 0.0289 0.0113 0.0031 0.0034]]
Actual Class : 1
-----
61 Text feature [144] present in test data point [True]
152 Text feature [169] present in test data point [True]
248 Text feature [183] present in test data point [True]
349 Text feature [276300] present in test data point [True]
350 Text feature [449] present in test data point [True]
386 Text feature [217] present in test data point [True]
434 Text feature [38] present in test data point [True]
489 Text feature [458] present in test data point [True]
Out of the top 500 features 8 are present in query point

```

4.3.2. Without Class balancing


```

In [304]: # 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, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# 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/geometric-intuition-1/
#-----

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# -----
# default parameters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# 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)

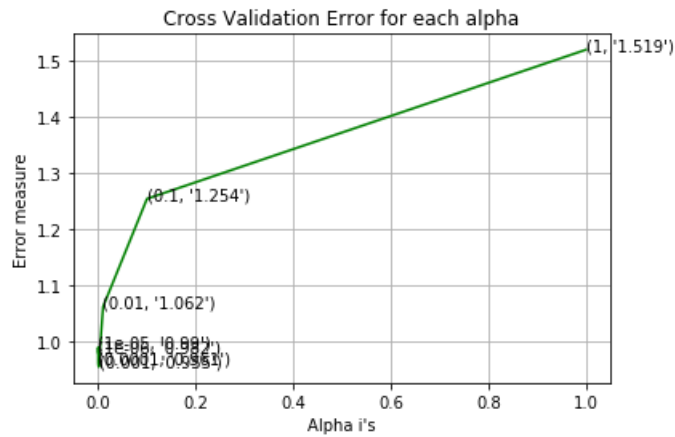
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha],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

```

```

for alpha = 1e-06
Log Loss : 0.9815371302328314
for alpha = 1e-05
Log Loss : 0.990057605040871
for alpha = 0.0001
Log Loss : 0.9612007738504138
for alpha = 0.001
Log Loss : 0.9554467161147713
for alpha = 0.01
Log Loss : 1.0623456849935953
for alpha = 0.1
Log Loss : 1.2541979620273025
for alpha = 1
Log Loss : 1.5191871641718004

```



```

For values of best alpha = 0.001 The train log loss is: 0.5775468155510546
For values of best alpha = 0.001 The cross validation log loss is: 0.9554467161147713
For values of best alpha = 0.001 The test log loss is: 0.995044452598985

```

4.3.2.2. Testing model with best hyper parameters

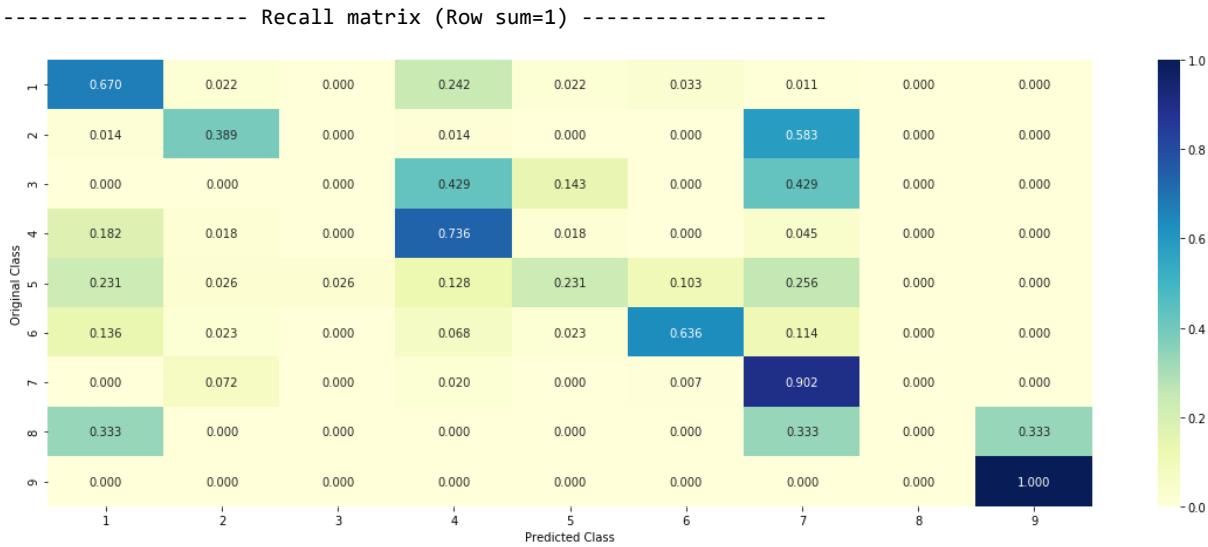
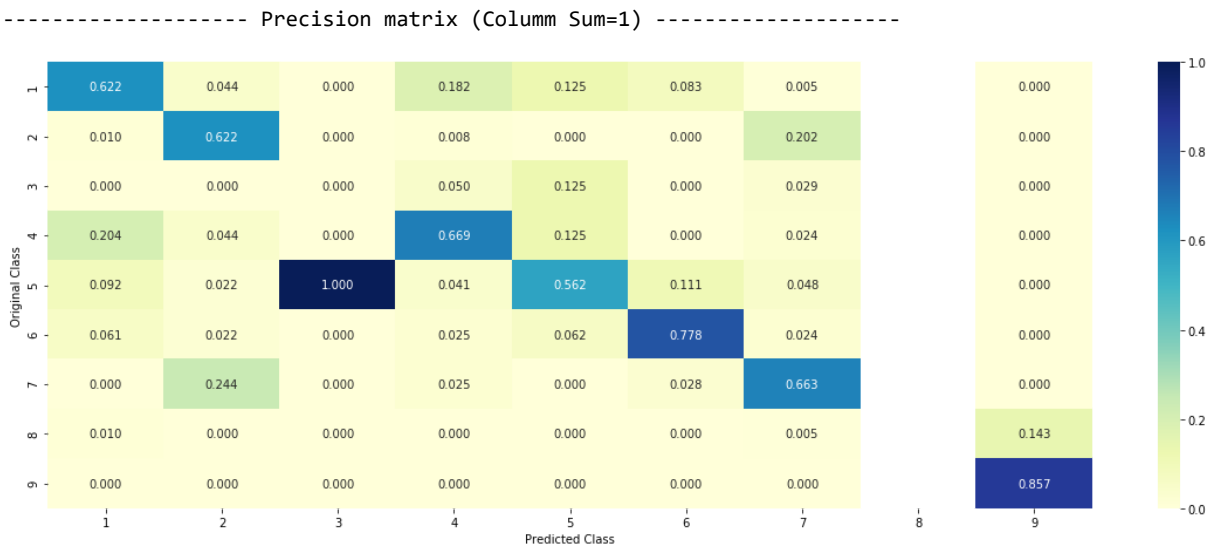
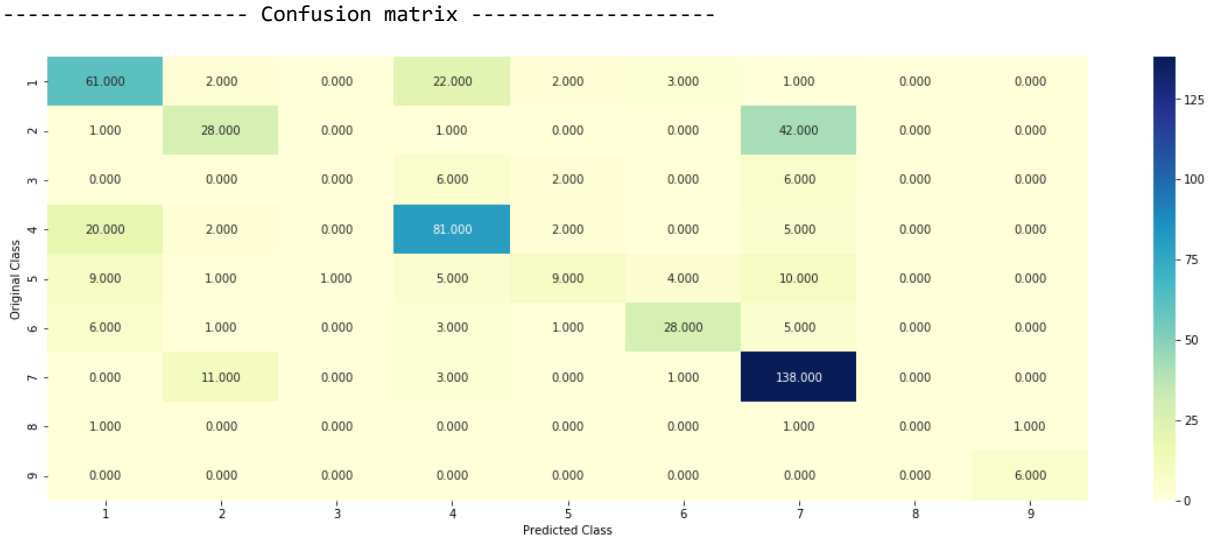
```
In [305]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# 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:
#-----

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, clf)
```

Log loss : 0.9554467161147713
Number of mis-classified points : 0.34022556390977443



4.3.2.3. Feature Importance, Correctly Classified point

```
In [306]: 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.iloc[test_point_index, :].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding.iloc[test_point_index, :].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:, :no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0099 0.0071 0.0348 0.9187 0.0124 0.0065 0.0049 0.0023 0.0034]]
Actual Class : 4
-----
78 Text feature [200] present in test data point [True]
334 Text feature [44] present in test data point [True]
430 Text feature [42] present in test data point [True]
Out of the top 500 features 3 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

```
In [307]: test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding.iloc[test_point_index, :].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding.iloc[test_point_index, :].reshape(1, -1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:, :no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.1657 0.0239 0.0103 0.3466 0.4094 0.0305 0.0059 0.0028 0.0048]]
Actual Class : 1
-----
49 Text feature [144] present in test data point [True]
170 Text feature [169] present in test data point [True]
322 Text feature [276300] present in test data point [True]
335 Text feature [183] present in test data point [True]
350 Text feature [217] present in test data point [True]
362 Text feature [449] present in test data point [True]
414 Text feature [458] present in test data point [True]
479 Text feature [38] present in test data point [True]
Out of the top 500 features 8 are present in query point
```

4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

```

In [308]: # read more about support vector machines with linear kernels here http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

# 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/mathematical-derivation-copy-8/
# -----

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# -----
# default parameters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# 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', 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 = SVC(C=i, kernel='linear', probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

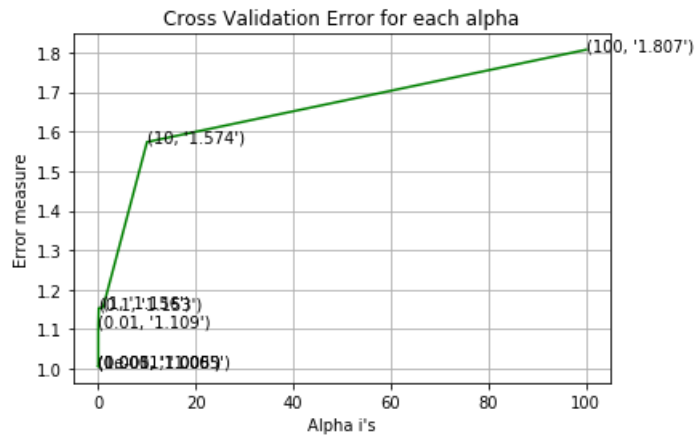
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha],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

```

```

print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
for C = 1e-05
Log Loss : 1.0050996043348637
for C = 0.0001
Log Loss : 1.0049497899468287
for C = 0.001
Log Loss : 1.0057111730892607
for C = 0.01
Log Loss : 1.1087508187419475
for C = 0.1
Log Loss : 1.15333090064045
for C = 1
Log Loss : 1.1555740267992154
for C = 10
Log Loss : 1.5735290247210454
for C = 100
Log Loss : 1.8068729052762467

```



For values of best alpha = 0.0001 The train log loss is: 0.48413403866838484
 For values of best alpha = 0.0001 The cross validation log loss is: 1.0049497899468287
 For values of best alpha = 0.0001 The test log loss is: 1.0301017311061773

4.4.2. Testing model with best hyper parameters

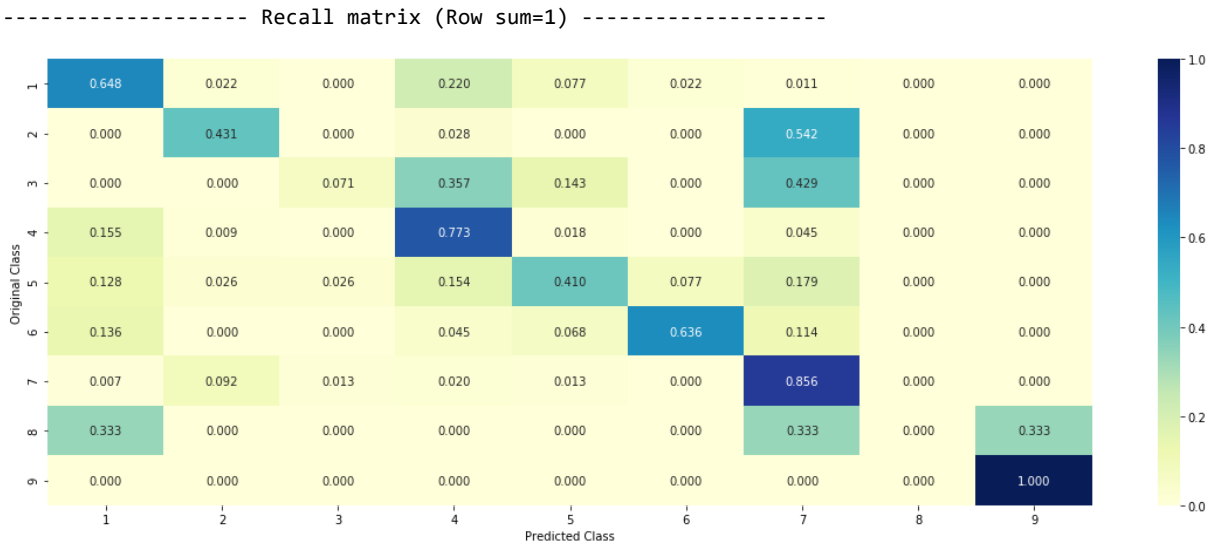
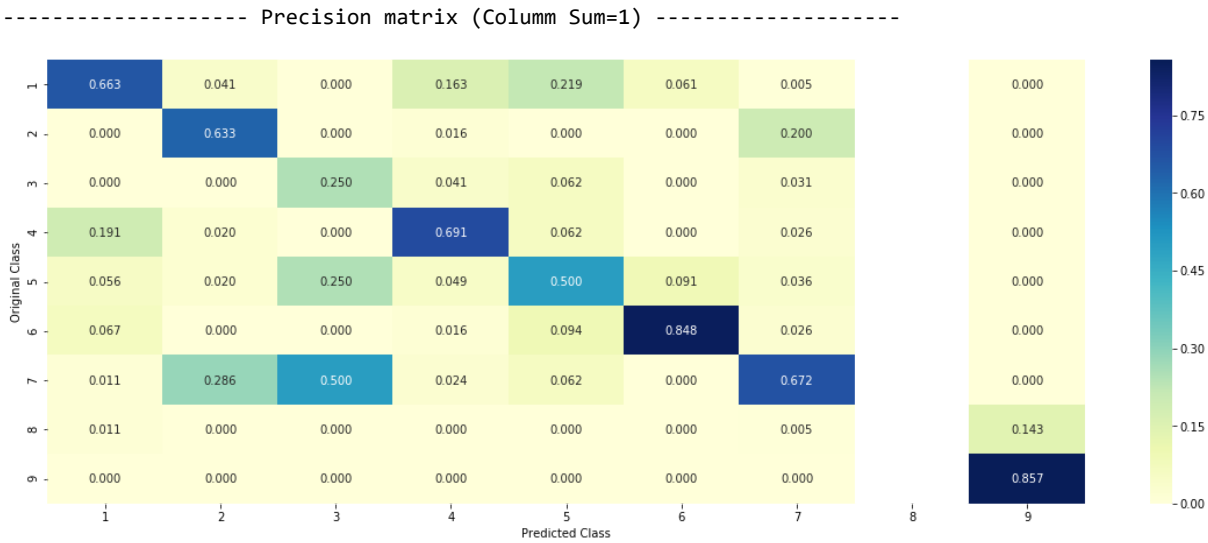
```
In [309]: # read more about support vector machines with linear kernels here http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

# 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.applidaicourse.com/course/applied-ai-course-online/lessons/mathematical-derivation-copy-8/
# -----

# 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,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```


Log loss : 1.0049497899468287
Number of mis-classified points : 0.32894736842105265



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

```
In [311]: 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.iloc[test_point_index, :].reshape(1, -1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding.iloc[test_point_index, :].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)

Predicted Class : 4
Predicted Class Probabilities: [[0.0313 0.016  0.0338 0.8273 0.0258 0.0116 0.0495 0.002  0.0027]]
Actual Class : 4
-----
5 Text feature [200] present in test data point [True]
467 Text feature [44] present in test data point [True]
486 Text feature [31] present in test data point [True]
Out of the top 500 features 3 are present in query point
```

4.3.3.2. For Incorrectly classified point

```
In [312]: test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding.iloc[test_point_index, :].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding.iloc[test_point_index, :].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)

Predicted Class : 5
Predicted Class Probabilities: [[0.0818 0.0313 0.0092 0.333  0.4988 0.0263 0.0134 0.0031 0.0031]]
Actual Class : 1
-----
41 Text feature [144] present in test data point [True]
228 Text feature [169] present in test data point [True]
247 Text feature [183] present in test data point [True]
329 Text feature [217] present in test data point [True]
424 Text feature [458] present in test data point [True]
449 Text feature [276300] present in test data point [True]
473 Text feature [36] present in test data point [True]
Out of the top 500 features 7 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

```

In [313]: # -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=
0, warm_start=False,
# 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/random-forest-and-their-construction-2/
# -----

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# -----
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# 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=4
2, n_jobs=-1)
        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()
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', max_depth=ma
x_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)

```

```

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

```

```

for n_estimators = 100 and max depth = 5
Log Loss : 1.1865379885037166
for n_estimators = 100 and max depth = 10
Log Loss : 1.2113440910020055
for n_estimators = 200 and max depth = 5
Log Loss : 1.1771037364330132
for n_estimators = 200 and max depth = 10
Log Loss : 1.2016233974573611
for n_estimators = 500 and max depth = 5
Log Loss : 1.1708950541902716
for n_estimators = 500 and max depth = 10
Log Loss : 1.1929406664084856
for n_estimators = 1000 and max depth = 5
Log Loss : 1.16636908224111
for n_estimators = 1000 and max depth = 10
Log Loss : 1.1875663609912825
for n_estimators = 2000 and max depth = 5
Log Loss : 1.1677238032096364
for n_estimators = 2000 and max depth = 10
Log Loss : 1.1875390569194275
For values of best estimator = 1000 The train log loss is: 0.865313677705407
For values of best estimator = 1000 The cross validation log loss is: 1.16636908224111
For values of best estimator = 1000 The test log loss is: 1.1736019632678938

```

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

```

In [314]: # -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=
0, warm_start=False,
# 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).

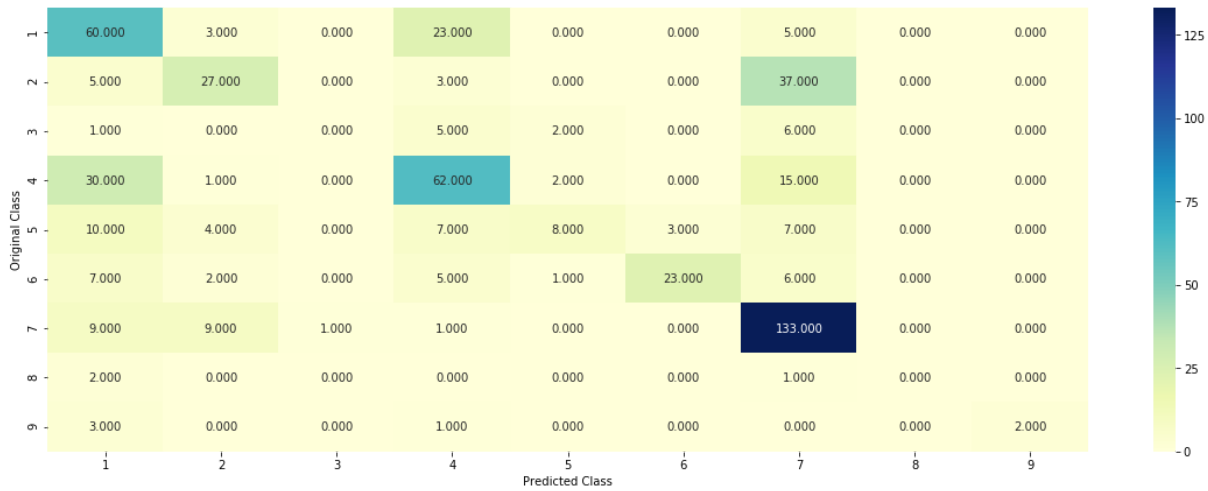
# -----
# video Link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-construction-2/
# -----

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=ma
x_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)

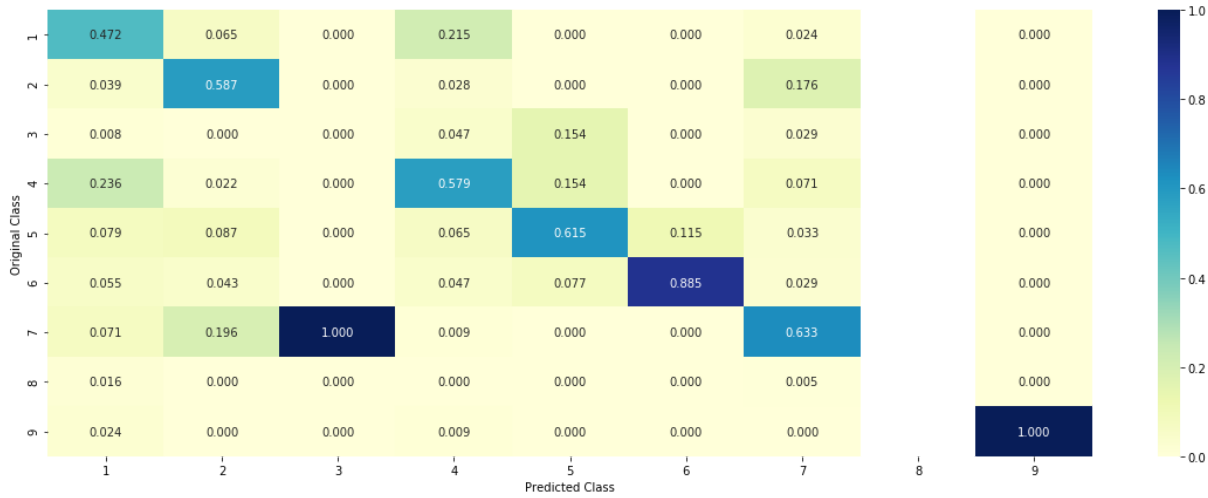
```

Log loss : 1.16636908224111
Number of mis-classified points : 0.40789473684210525

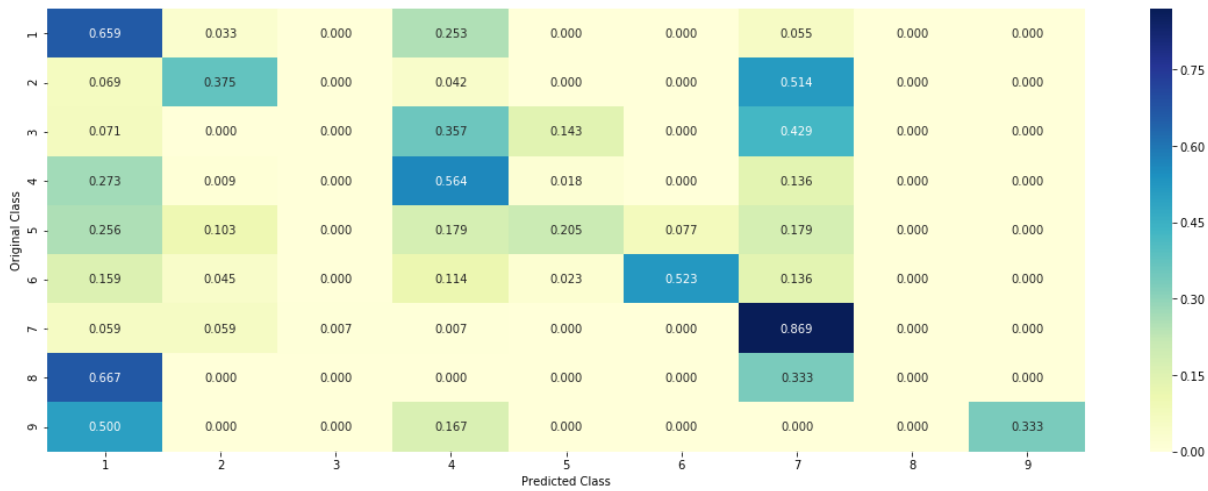
----- Confusion matrix -----



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



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

```
In [315]: # test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=ma
x_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
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.iloc[test_point_index, :].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding.iloc[te
st_point_index, :].reshape(1, -1)),4))
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['Gene'].
iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)

Predicted Class : 4
Predicted Class Probabilities: [[0.0259 0.0089 0.0237 0.8624 0.0283 0.024 0.021 0.0024 0.0036]]
Actual Class : 4
-----
25 Text feature [42] present in test data point [True]
Out of the top 100 features 1 are present in query point
```

4.5.3.2. Inorrectly Classified point

```
In [316]: test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding.iloc[test_point_index, :].reshape(1, -1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding.iloc[te
st_point_index, :].reshape(1, -1)),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].
iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)

Predicted Class : 4
Predicted Class Probabilities: [[0.2868 0.0169 0.018 0.3335 0.1777 0.1312 0.0249 0.0038 0.0072]]
Actuall Class : 1
-----
25 Text feature [42] present in test data point [True]
Out of the top 100 features 1 are present in query point
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

```

In [317]: # 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, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# 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/geometric-intuition-1/
#-----

# read more about support vector machines with linear kernels here http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

# 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/mathematical-derivation-copy-8/
# -----

# read more about support vector machines with linear kernels here http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
# -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False,
# 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/random-forest-and-their-construction-2/
# -----

clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced', random_state=0)
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=0)
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, use_probab=True)
    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))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error

```

```

Logistic Regression : Log Loss: 0.97
Support vector machines : Log Loss: 1.17
Naive Bayes : Log Loss: 1.23

```

```

-----
Stacking Classifier : for the value of alpha: 0.000100 Log Loss: 2.176
Stacking Classifier : for the value of alpha: 0.001000 Log Loss: 2.020
Stacking Classifier : for the value of alpha: 0.010000 Log Loss: 1.467
Stacking Classifier : for the value of alpha: 0.100000 Log Loss: 1.110
Stacking Classifier : for the value of alpha: 1.000000 Log Loss: 1.309
Stacking Classifier : for the value of alpha: 10.000000 Log Loss: 1.788

```

4.7.2 testing the model with the best hyper parameters

```
In [318]: lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba=True)
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))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding))
```

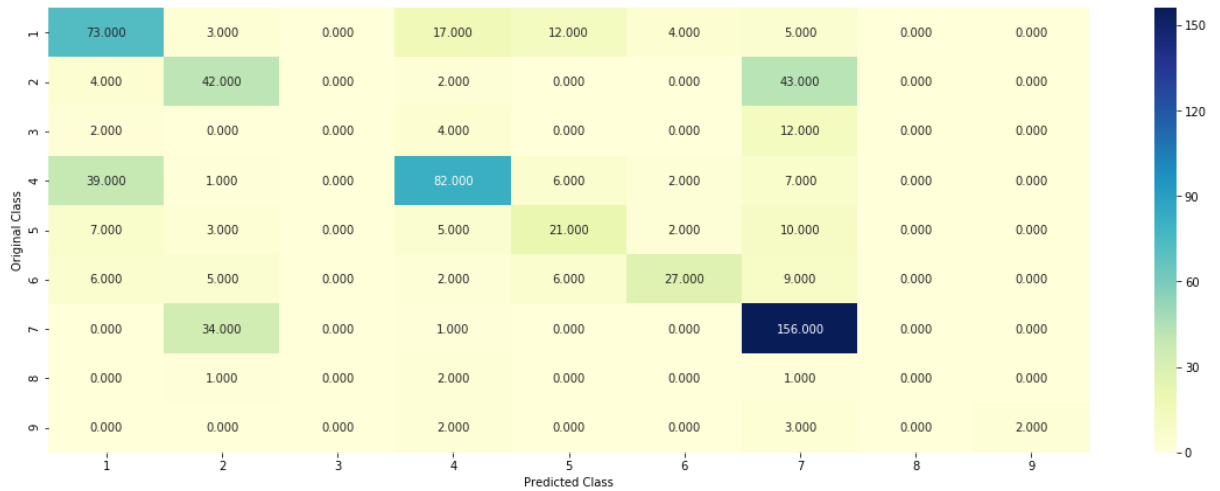
Log loss (train) on the stacking classifier : 0.5373966434229661

Log loss (CV) on the stacking classifier : 1.1096531140060404

Log loss (test) on the stacking classifier : 1.1483411935398586

Number of misclassified point : 0.39398496240601505

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



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



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



4.7.3 Maximum Voting classifier

```
In [319]: #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)], voting
='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba(train_x_o
nehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_x_onehotCodi
ng)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba(test_x_oneh
otCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCoding)- tes
t_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

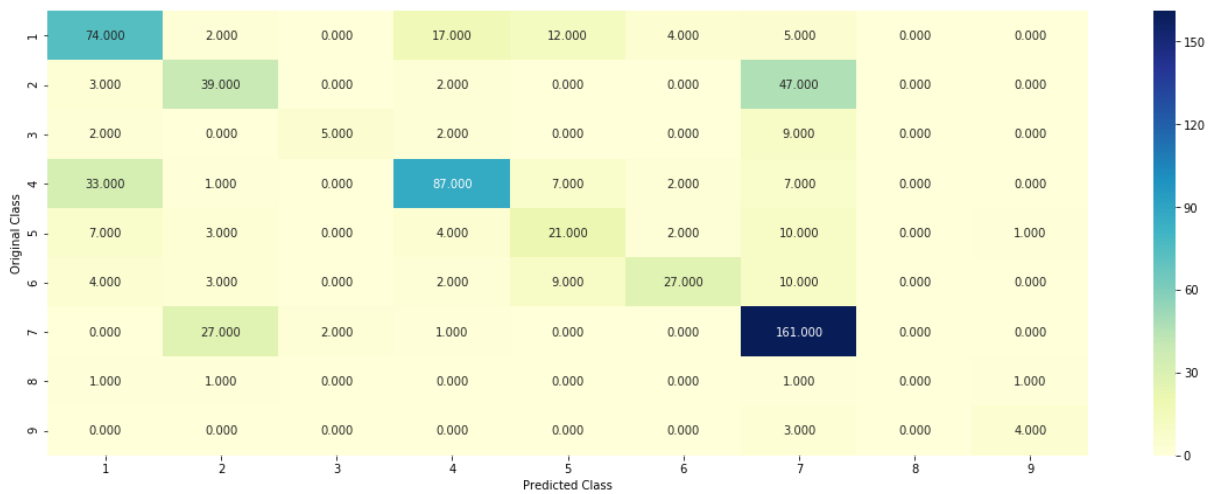
Log loss (train) on the VotingClassifier : 0.6874087688400785

Log loss (CV) on the VotingClassifier : 1.0753424620453356

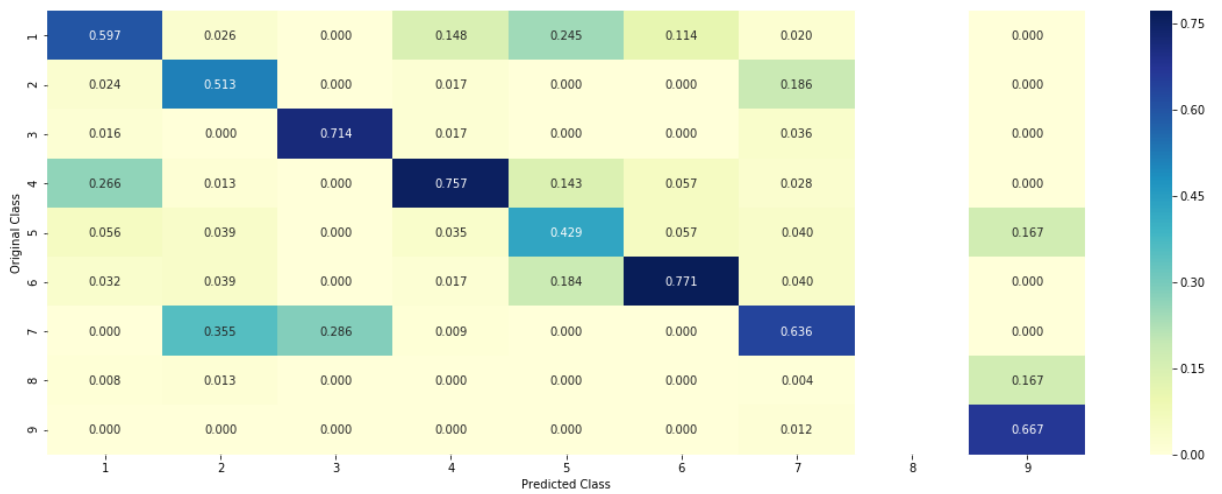
Log loss (test) on the VotingClassifier : 1.100137241027604

Number of missclassified point : 0.37142857142857144

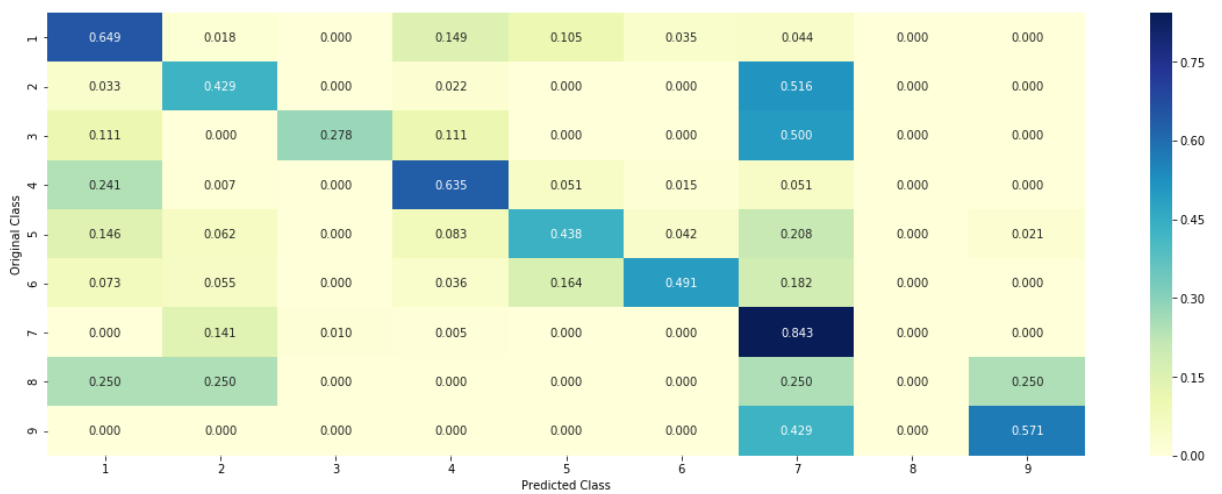
----- Confusion matrix -----



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



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



Using Word2Vectors

```
In [302]: def cleanhtml(sentence): #function to clean the word of any html-tags
          cleanr = re.compile('<.*?>')
          cleantext = re.sub(cleanr, ' ', sentence)
          return cleantext
def cleanpunc(sentence): #function to clean the word of any punctuation or special characters
  cleaned = re.sub(r'[?!\|\\\'|"]|#]',r'',sentence)
  cleaned = re.sub(r'[,.|)|(|\\|/]',r' ',cleaned)
  return cleaned
```

```
In [315]: # Train your own Word2Vec model using your own text corpus for train Data
import gensim
i=0
list_of_sent_train=[]
for sent in train_df['TEXT'].values:
  filtered_sentence=[]
  sent=cleanhtml(sent)
  for w in sent.split():
    for cleaned_words in cleanpunc(w).split():
      if(cleaned_words.isalpha()):
        filtered_sentence.append(cleaned_words.lower())
      else:
        continue
  list_of_sent_train.append(filtered_sentence)
```

```
In [316]: # min_count = 5 considers only words that occurred atleast 5 times
import gensim
from gensim import models
from gensim.models import Word2Vec, KeyedVectors
w2v_model=Word2Vec(list_of_sent_train,min_count=5,size=200,workers=4)
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occurred minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:200])
```

```
number of words that occurred minimum 5 times 31649
sample words ['hub', 'proteins', 'connected', 'binding', 'interactions', 'many', 'mediator', 'sig
nal', 'transduction', 'induced', 'transforming', 'growth', 'factor', 'beta', 'tgf', 'b', 'serves',
'protein', 'different', 'cellular', 'responses', 'mediated', 'product', 'cell', 'type', 'context',
'dependent', 'nucleated', 'complexes', 'acting', 'concert', 'hypothesis', 'perturbation', 'spectru
m', 'mutation', 'single', 'hot', 'spots', 'distinct', 'consequences', 'methodology', 'principal',
'findings', 'mutated', 'amino', 'acids', 'surface', 'domain', 'identified', 'variants', 'reduced',
'subsets', 'including', 'sara', 'ski', 'mutations', 'defective', 'e', 'g', 'nucleocytoplasmic', 's
huttling', 'compromised', 'modulating', 'expression', 'levels', 'reporter', 'gene', 'six', 'endoge
nous', 'responsive', 'genes', 'however', 'mutants', 'differences', 'signaling', 'example', 'activa
tion', 'inducibility', 'contrast', 'inducibility', 'severely', 'increased', 'caused', 'higher', 'ba
sal', 'lower', 'conclusions', 'significance', 'interacting', 'range', 'quantitative', 'changes',
'approach', 'useful', 'unraveling', 'critical', 'specific', 'biological', 'citation', 'schiro', 'm
m', 'stauber', 'se', 'peterson', 'tl', 'krueger', 'c', 'darnell', 'sj', 'et', 'al', 'differentiall
y', 'affect', 'regulated', 'plos', 'one', 'doi', 'journal', 'pone', 'editor', 'dhyam', 'chandra',
'roswell', 'park', 'cancer', 'institute', 'united', 'states', 'america', 'received', 'june', 'acce
pted', 'august', 'published', 'september', 'copyright', 'open', 'access', 'article', 'distribute
d', 'terms', 'creative', 'commons', 'attribution', 'license', 'permits', 'unrestricted', 'use', 'd
istribution', 'reproduction', 'medium', 'provided', 'original', 'author', 'source', 'credited', 'f
unding', 'work', 'supported', 'muscular', 'dystrophy', 'association', 'www', 'mda', 'org', 'gran
t', 'juvenile', 'diabetes', 'research', 'foundation', 'jdrf', 'national', 'gov', 'funders', 'rol
e', 'study', 'design', 'data', 'collection', 'analysis', 'decision', 'publish', 'preparation', 'ma
nuscript', 'competing', 'interests', 'authors', 'declared', 'exist', 'mail', 'hoffmann', 'oncolog
y', 'wisc', 'edu', 'introduction', 'transient', 'ubiquitous', 'molecular', 'mechanism', 'regulatin
g']
```

```
In [317]: # average Word2Vec
# compute average word2vec for each review.
sent_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sent in list_of_sent_train: # for each review/sentence
    sent_vec = np.zeros(200) # as word vectors are of zero length
    cnt_words = 0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_train.append(sent_vec)
print(len(sent_vectors_train))
print(len(sent_vectors_train[0]))
```

2124
200

Test Data

```
In [318]: # Train your own Word2Vec model using your own text corpus for test Data
import gensim
i=0
list_of_sent_test=[]
for sent in test_df['TEXT'].values:
    filtered_sentence=[]
    sent=cleanhtml(sent)
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if(cleaned_words.isalpha()):
                filtered_sentence.append(cleaned_words.lower())
            else:
                continue
    list_of_sent_test.append(filtered_sentence)
```

```
In [319]: # min_count = 5 considers only words that occurred at least 5 times
w2v_model=Word2Vec(list_of_sent_test,min_count=5,size=200,workers=4)
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occurred minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

number of words that occurred minimum 5 times 19076
sample words ['purpose', 'kit', 'major', 'oncogenic', 'driver', 'gastrointestinal', 'stromal', 'tumors', 'gists', 'imatinib', 'sunitinib', 'regorafenib', 'approved', 'therapies', 'however', 'efficacy', 'often', 'limited', 'acquisition', 'polyclonal', 'secondary', 'resistance', 'mutations', 'located', 'activation', 'loop', 'exons', 'particularly', 'problematic', 'explored', 'inhibitory', 'activity', 'ponatinib', 'preclinical', 'models', 'describe', 'initial', 'characterization', 'gist', 'patients', 'experimental', 'design', 'cellular', 'vivo', 'activities', 'mutant', 'evaluated', 'using', 'accelerated', 'mutagenesis']

```
In [320]: # average Word2Vec
# compute average word2vec for each review.
sent_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sent in list_of_sent_test: # for each review/sentence
    sent_vec = np.zeros(200) # as word vectors are of zero length
    cnt_words = 0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_test.append(sent_vec)
print(len(sent_vectors_test))
print(len(sent_vectors_test[0]))
```

665
200

CV data

```
In [321]: # Train your own Word2Vec model using your own text corpus for test Data
import gensim
i=0
list_of_sent_cv=[]
for sent in cv_df['TEXT'].values:
    filtered_sentence=[]
    sent=cleanhtml(sent)
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if(cleaned_words.isalpha()):
                filtered_sentence.append(cleaned_words.lower())
            else:
                continue
    list_of_sent_cv.append(filtered_sentence)
```

```
In [322]: # min_count = 5 considers only words that occurred atleast 5 times
w2v_model=Word2Vec(list_of_sent_cv,min_count=5,size=200,workers=4)
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occurred minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

number of words that occurred minimum 5 times 16875
sample words ['activation', 'canonical', 'tgf', 'signaling', 'pathway', 'provides', 'growth', 'inhibitory', 'signals', 'normal', 'intestinal', 'epithelium', 'colorectal', 'cancers', 'crcs', 'frequently', 'harbor', 'somatic', 'mutations', 'members', 'extent', 'contribute', 'tumorigenesis', 'unclear', 'cohort', 'primary', 'crc', 'cell', 'lines', 'sequenced', 'analyzed', 'allelic', 'loss', 'single', 'nucleotide', 'polymorphism', 'snp', 'microarray', 'analysis', 'mutation', 'spectra', 'compared', 'genes', 'pathogenicity', 'assessed', 'relationships', 'clinicopathologic', 'features', 'examined', 'prevalence']

```
In [323]: # average Word2Vec
# compute average word2vec for each review.
sent_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sent in list_of_sent_cv: # for each review/sentence
    sent_vec = np.zeros(200) # as word vectors are of zero length
    cnt_words = 0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_cv.append(sent_vec)
print(len(sent_vectors_cv))
print(len(sent_vectors_cv[0]))
```

532
200


```
In [324]: # 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_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 = hstack((train_gene_var_onehotCoding,sent_vectors_train)).tocsr()
train_y = np.array(list(train_df['Class']))

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

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

```

In [325]: # 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, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# 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/geometric-intuition-1/
#-----

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# -----
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# 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 probabilities we use log-probability estimates
    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)

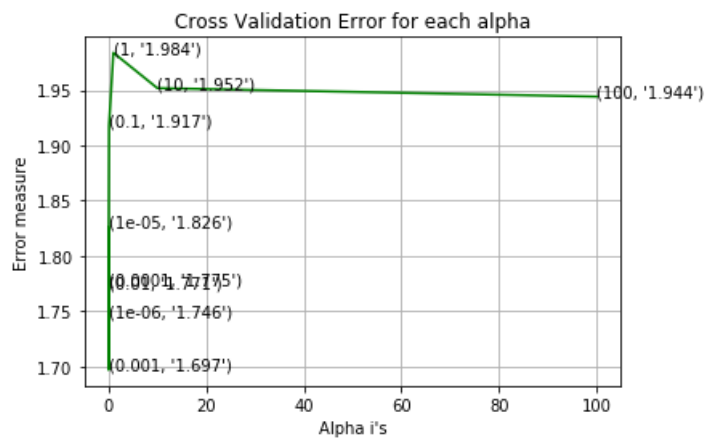
predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha],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

```

```

for alpha = 1e-06
Log Loss : 1.7458173513766055
for alpha = 1e-05
Log Loss : 1.8258168695785402
for alpha = 0.0001
Log Loss : 1.7745596603630562
for alpha = 0.001
Log Loss : 1.6970338684358992
for alpha = 0.01
Log Loss : 1.7708477037124097
for alpha = 0.1
Log Loss : 1.917326871281069
for alpha = 1
Log Loss : 1.9837670827139646
for alpha = 10
Log Loss : 1.951773139339752
for alpha = 100
Log Loss : 1.9441062620102016

```



For values of best alpha = 0.001 The train log loss is: 0.8271757247127755
 For values of best alpha = 0.001 The cross validation log loss is: 1.6970338684358992
 For values of best alpha = 0.001 The test log loss is: 1.465678659743184

```

In [326]: # 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, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

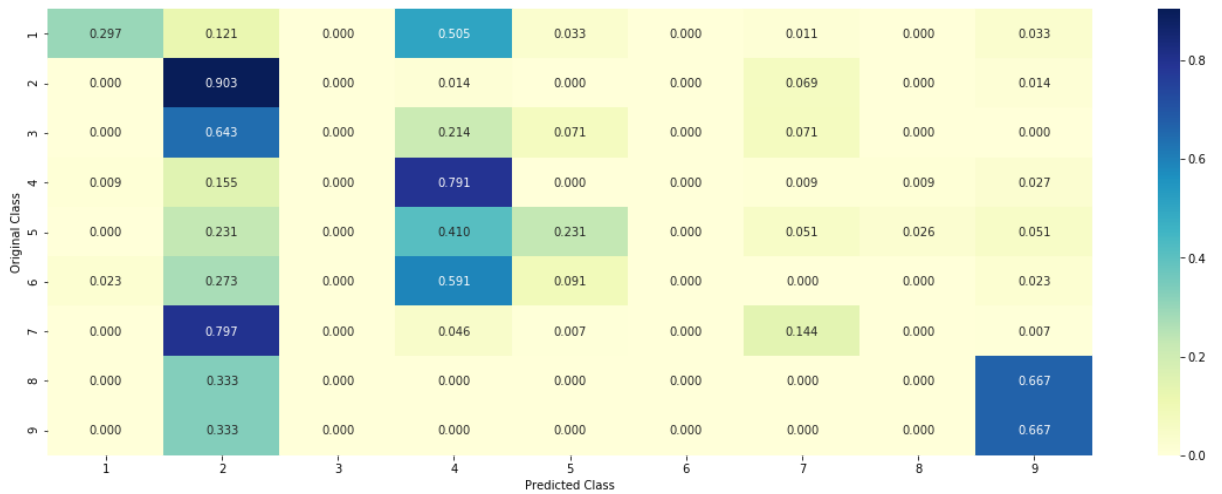
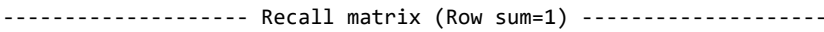
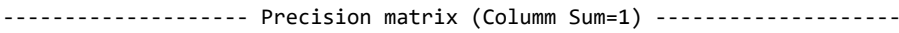
# some of methods
# fit(X, y[, coef_init, intercept_init, ...])    Fit linear model with Stochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1/
#-----
clf = SGDClassifier(class_weight='balanced', 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, clf)

```

Number of mis-classified points : 0.5977443609022557

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



Conclusion Of 1st Section:-

We tried many models but with Tf-Idf Vectoriser in Bi-grams and for 2000 most important words, we were best able to classify Cancer into its actual classes with minimum error :-

- Stacking Model and Maximum Vote classifier model are hard to interpret.
- Best Model For this Dataset is :- **Logistic Regression with Class Balancing**
- Train Log-Loss:-0.4246
- CV Log-Loss:-0.9572
- Test Log-Loss:-0.9994
- Percentage Misclassified:-0.3270

Model	Train	CV	Test	Percentage Misclassified
Naive Bayes	0.6218	1.2337	1.2642	0.3759
KNN (Best K=11)	0.8589	1.0852	1.0951	0.3778
LR (With Class Balancing)	0.4246	0.9572	0.9994	0.3270
LR (Without Class Balancing)	0.5775	0.9554	0.9950	0.3402
LINEAR SVM	0.4841	1.0049	1.0301	0.3289
Random Forest	0.8653	1.1663	1.1736	0.4078
Stacking Clasifier (Log Reg+ SVM + Naive Bayes)	0.5373	1.1096	1.1483	0.3939
Maximum Voting Classifier (Log Reg+ SVM + RF)	0.6874	1.0753	1.001	0.3714
Word2Vec(Logistic Regression)	0.8271	1.6970	1.7656	0.5977