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

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

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

We have two data files: one conatins the information about the genetic mutations and the other contains the clinical evidence (text) that human experts/pathologists use to classify the genetic mutations. Both these data files are have a common column called ID
Data file's information:

training_variants (ID , Gene, Variations, Class)

training_text (ID, Text)

There are nine different classes a genetic mutation can be classified on. Its multiclass classification problem that we need to solve

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

Metric(s):

Multi class log-loss

Confusion matrix

Machine Learing Objectives and Constraints

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

Constraints:

Interpretability

Class probabilities are needed.

Penalize the errors in class probabilites => Metric is Log-loss.

No Latency constraints.

Exploratory Data Analysis

Import requuired libraries

In [4]:

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

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

from mlxtend.classifier import StackingClassifier

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

Read Data

In [5]:

```
data_text =pd.read_csv("E:\\ml_downloads\\personalised_cancer_dognosis\\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[5]:

	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...

In [6]:

```
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 [7]:

```
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)
```

```
there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
```

In [8]:

```
gene_variations = pd.read_csv("E:\\ml_downloads\\personalised_cancer_dognosis\\training_variants",
sep=",",engine="python",names=["ID","Gene","Variation","Class"],skiprows=1)
```

In [9]:

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

Out[9]:

	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...

Check if null values present for any column ;

In [10]:

```
result.isnull().sum()
```

Out[10]:

```
ID           0
Gene         0
Variation    0
Class        0
TEXT         5
dtype: int64
```

In [19]:

```
result.loc[result['TEXT'].isnull()]['TEXT']
```

Out[19]:

```
1109    NaN
1277    NaN
1407    NaN
1639    NaN
2755    NaN
Name: TEXT, dtype: object
```

In [20]:

```
result.loc[result['TEXT'].isnull()]['TEXT'] = result['Gene'] + result['Variation']
```

In [25]:

```
result.isnull().sum()
```

Out[25]:

Out[23]:

```
ID           0
Gene         0
Variation    0
Class        0
TEXT         0
dtype: int64
```

In [24]:

```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] + result['Variation']
```

In [27]:

```
result[result['ID'] == 1109]
```

Out[27]:

	ID	Gene	Variation	Class	TEXT
1109	1109	FANCA	S1088F	1	FANCAS1088F

Splitting the data train, CV and test sets ;

In [28]:

```
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 varaible '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
varaible '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
)
```

Undestanding basic stats ;

In [30]:

```
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])
print('Unique Class labels:', result['Class'].unique())
```

```
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
Unique Class labels: [1 2 3 4 5 6 7 8 9]
```

Check data distributions class(output label) wise Its just bar chart that gives how % of datapoints for each class

In [31]:

```
def plotdistribution(df_dist,totalvalues):
    my_colors = 'rgbkymc'
    df_dist.plot(kind='bar')
    plt.xlabel('Class')
    plt.ylabel('Data points per Class')
    plt.title('Distribution of yi in data')
    plt.grid()
    plt.show()
    sorted_yi = np.argsort(-df_dist)
    for i in sorted_yi:
```

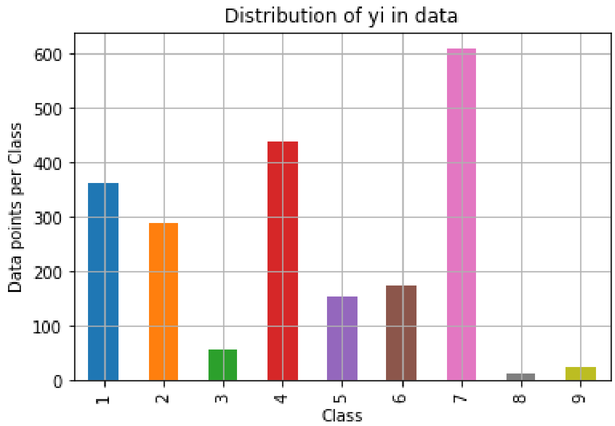
```
        print('Number of data points in class', i+1, ':',df_dist.values[i], '(', np.round((df_dist.
values[i]/totalvalues*100), 3), '%)')
```

In [32]:

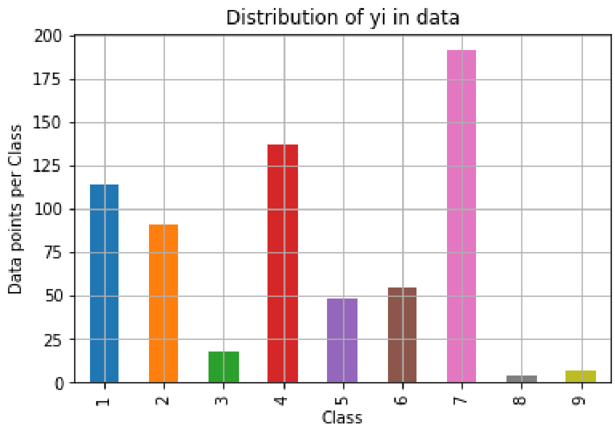
```
train_class_distribution = train_df['Class'].value_counts().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()
```

In [33]:

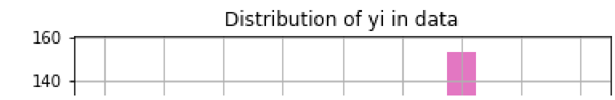
```
#for training input
plotdistribution(train_class_distribution,train_df.shape[0])
plotdistribution(test_class_distribution,test_df.shape[0])
plotdistribution(cv_class_distribution,cv_df.shape[0])
```

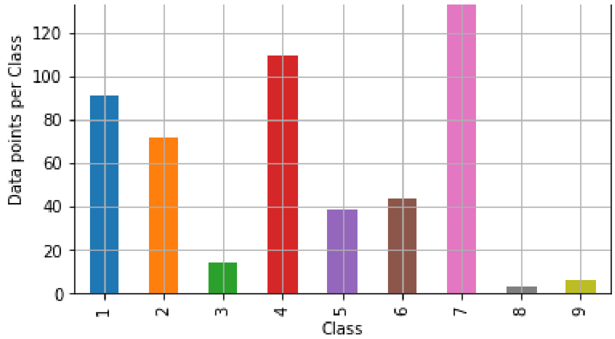



```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298 %)
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 %)
```

We can see that Class 7 has more data points in all 3 data sets, This distribution is example of imbalanced data set.

In [98]:

```
# 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

    recall =(((C.T)/(C.sum(axis=1))).T)
    precision = (C/C.sum(axis=0))

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

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

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

Lets use Random/Dummy Classifier as initial approach to check Log-loss

In [34]:

```
from sklearn.dummy import DummyClassifier
```

In [75]:

```
print(cv_df.shape[0])

print(test_df.shape[0])
```

```
532
665
```

In [118]:

```
X_dummy_cv = np.random.rand(532,5)
y_dummy_cv = np.array(cv_df['Class'])
dummy_classifier_cv = DummyClassifier(strategy="uniform")
dummy_classifier_cv.fit( X_dummy_cv,y_dummy_cv )
```

Out[118]:

```
DummyClassifier(constant=None, random_state=None, strategy='uniform')
```

In [119]:

```
X_dummy_test = np.random.rand(665,5)
y_dummy_test = np.array(test_df['Class'])
dummy_classifier_test = DummyClassifier(strategy="uniform")
dummy_classifier_test.fit( X_dummy_test,y_dummy_test )
```

Out[119]:

```
DummyClassifier(constant=None, random_state=None, strategy='uniform')
```

In [120]:

```
X_dummy_cv
```

Out[120]:

```
array([[0.76506208, 0.19049602, 0.99824126, 0.58236831, 0.51378423],
       [0.1878856 , 0.09542029, 0.08220041, 0.32439533, 0.21840713],
       [0.69145123, 0.93906919, 0.28318562, 0.35557383, 0.47284181],
       ...,
       [0.58335598, 0.62318309, 0.35798179, 0.53939097, 0.58586319],
       [0.85717431, 0.05382636, 0.10195432, 0.45107406, 0.10279438],
       [0.77661274, 0.05129063, 0.23024536, 0.86078249, 0.09496132]])
```

In [121]:

```
X_dummy_test
```

Out[121]:

```
array([[0.92225405, 0.35598576, 0.35187589, 0.64438907, 0.36283455],
       [0.831871  , 0.76534685, 0.04315433, 0.50682673, 0.21186905],
       [0.79750355, 0.02603913, 0.53578586, 0.17336648, 0.70077329],
       ...,
       [0.43504803, 0.60196831, 0.63289124, 0.17429288, 0.10986142],
       [0.48395646, 0.00985991, 0.50176735, 0.44117387, 0.19063105],
       [0.11907138, 0.81202014, 0.2472877 , 0.84599476, 0.92813748]])
```

In [115]:

```
X1_dummy_cv = np.random.rand(532,5)
X1_dummy_test = np.random.rand(665,5)
```

In [116]:

```
y_pred_dummy_cv = dummy_classifier_cv.predict_proba(X1_dummy_cv)
y_pred_dummy_test = dummy_classifier_test.predict_proba(X1_dummy_test)
```

In [117]:

```
print("CV Log loss of Random Classfier" ,log_loss(y_dummy_cv,y_pred_dummy_cv))
print("Test Log Loss of Random Classfier", log_loss(y_dummy_test,y_pred_dummy_test))
```

```
CV Log loss of Random Classfier 27.462222584675235
Test Log Loss of Random Classfier 2.1972245773362196
```

We have now random classfier where log loss is > 2.0 for Cross Validation and Test data which is randomly generated; Our Aim will be design model such that Log loss ~ 0 , in practise log-loss < log-loss(random classifier) will be good enough to gauage the performance of model ;

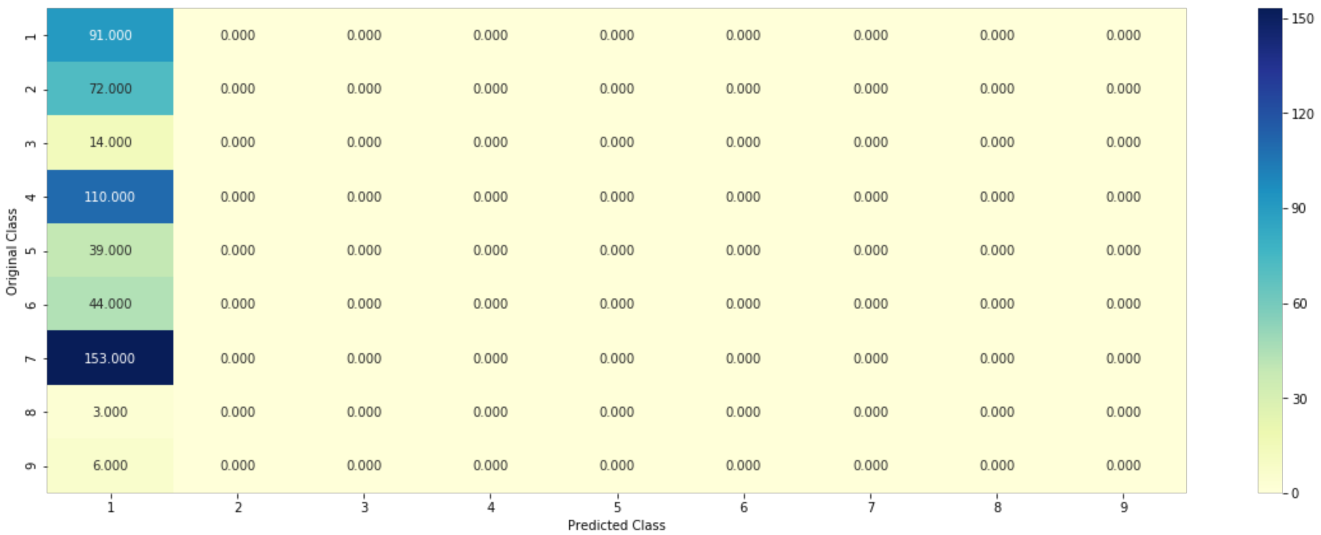
In [106]:

```
y_pred_dummy_cv_cls = np.argmax(y_pred_dummy_cv,axis=1)
```

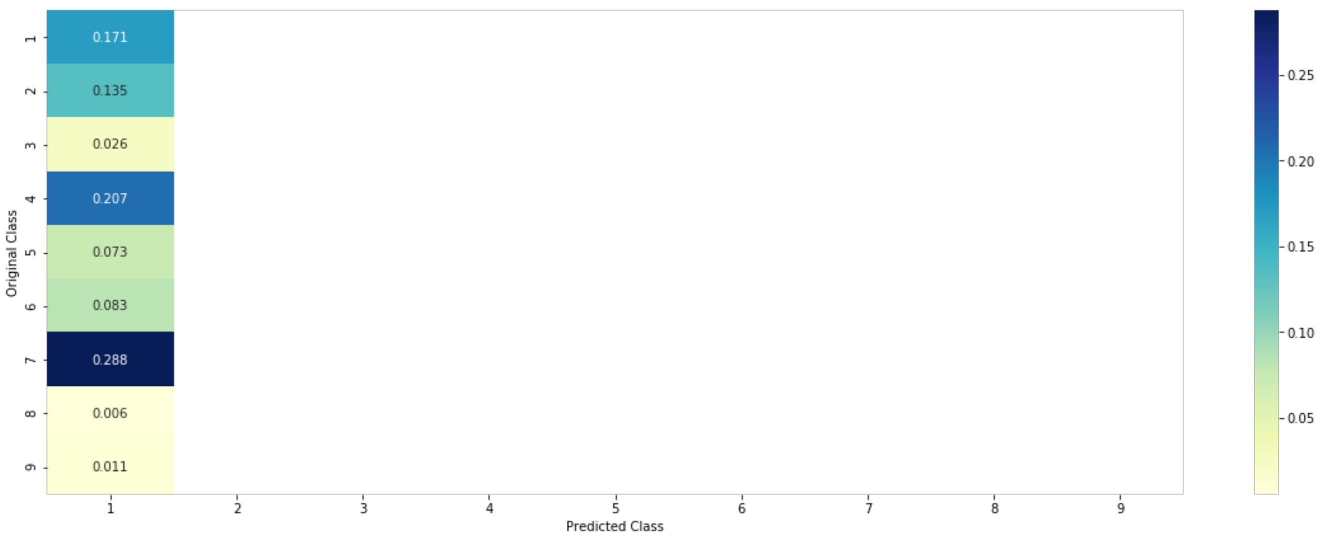
In [107]:

```
plot_confusion_matrix(y_dummy_cv,y_pred_dummy_cv_cls + 1)
```

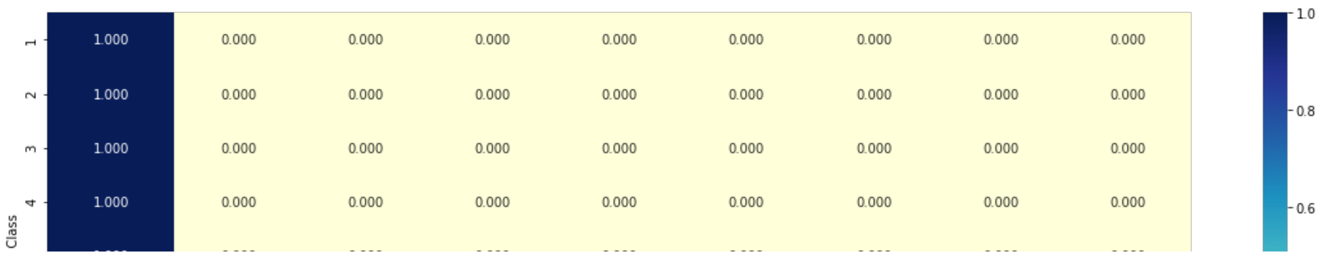
```
-------------------- Confusion matrix --------------------
```

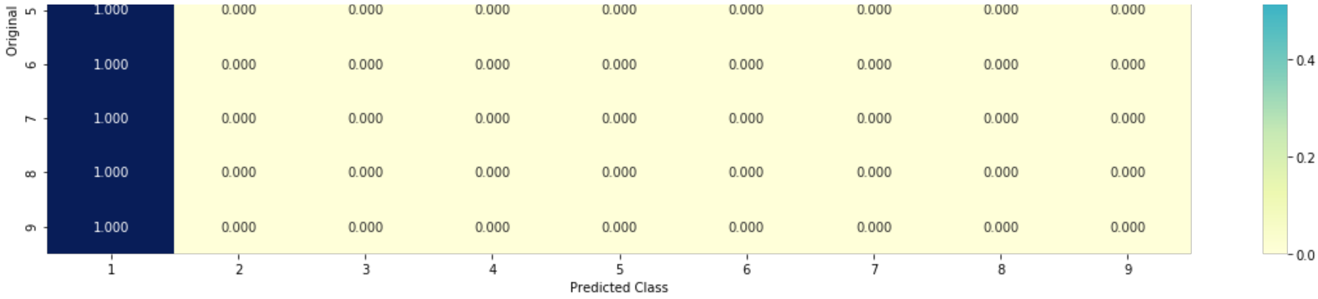



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





Univariants Analysis

Helper functions to use ;

In [122]:

```
def get_feature_counts(alpha,feature,df):
    gv_dict= {}
    valuecnt = df[feature].value_counts()
    for i, d in valuecnt.items():
        vec = []
        for k in range(1,10):
            cls_cnt = df.loc[(df['Class']==k) & (df[feature]==i)]
            vec.append((cls_cnt.shape[0] + alpha*10)/ (d + 90*alpha)) # laplace smoothing
        gv_dict[i]=vec
    return gv_dict

def get_gv_feature(alpha, feature, df):
    gv_dict = get_feature_counts(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 da
ta
    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
```

In [124]:

```
vectorizer = TfidfVectorizer(min_df=3)
train_feature_onehotCoding = vectorizer.fit_transform(train_df['Gene'])
```

In [127]:

```
cnt = CountVectorizer()
train_feature_onehotCoding_1 = cnt.fit_transform(train_df['Gene'])
```

In [134]:

```
train_feature_onehotCoding.toarray()
```

Out[134]:

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

In [135]:

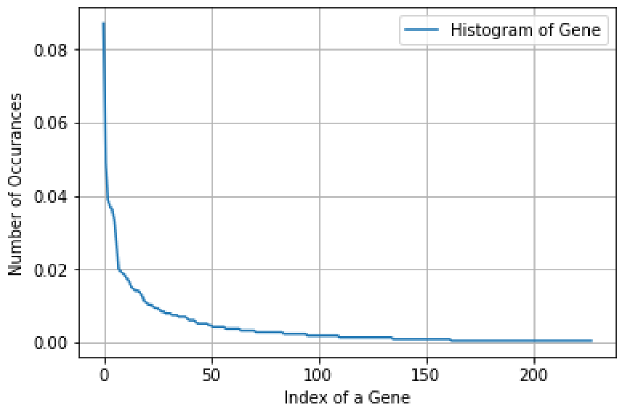
```
def do_univariant_analysis(traindf,testdf,cvdf,feature):

    unique_features = traindf[feature].value_counts()
    print("Ans: There are", unique_features.shape[0] ,"different categories of " + feature + " in t
he train data, and they are distibuted as follows",)

    h  = unique_features.values/sum(unique_features.values)
    plt.plot(h,label = "Histogram of " + feature + "")
    plt.xlabel('Index of a '+ feature + '' )
    plt.ylabel('Number of Occurances')
    plt.legend()
    plt.grid()
    plt.show()

    #CDF of above plot ;

    c = np.cumsum(h)
    plt.plot(c,label='Cumulative distribution of '+ feature +'')
    plt.grid()
    plt.legend()
    plt.show()

    #response-coding of the Gene feature
    # alpha is used for laplace smoothingb
    alpha = 1
    # train gene feature
    train_feature_responseCoding = np.array(get_gv_feature(alpha, feature, traindf))
    # test gene feature
    test_feature_responseCoding = np.array(get_gv_feature(alpha, feature, testdf))
    # cross validation gene feature
    cv_feature_responseCoding = np.array(get_gv_feature(alpha, feature, cvdf))

    print("train_feature_responseCoding is converted feature using respone coding method. The
shape of "+feature+" feature:", train_feature_responseCoding.shape)

    # one-hot encoding of given feature.
    vectorizer = TfidfVectorizer(min_df=3)
    train_feature_onehotCoding = vectorizer.fit_transform(traindf[feature])
    test_feature_onehotCoding = vectorizer.transform(testdf[feature])
    cv_feature_onehotCoding = vectorizer.transform(cvdf[feature])

    print("train_feature_onehotCoding is converted feature using one-hot encoding method. The shap
e of "+feature+" feature:", train_feature_onehotCoding.shape)

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

    cv_log_error_array=[]
    for i in alpha_hyperparam:
        clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
        clf.fit(train_feature_onehotCoding, y_train)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_feature_onehotCoding, y_train)
        predict_y = sig_clf.predict_proba(cv_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_hyperparam, cv_log_error_array,c='g')
    for i, txt in enumerate(np.round(cv_log_error_array,3)):
        ax.annotate((alpha_hyperparam[i],np.round(txt,3)), (alpha_hyperparam[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_hyperparam[best_alpha], penalty='l2', loss='log', random_state=
42)
    clf.fit(train_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

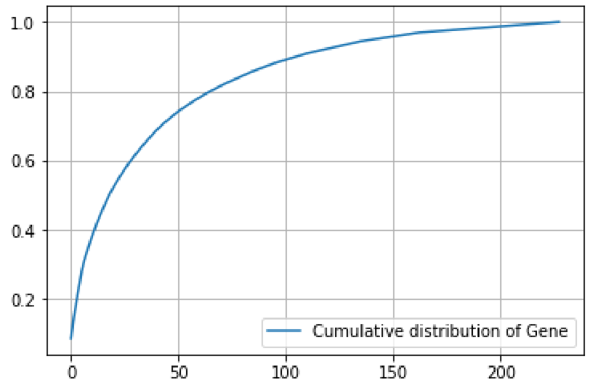
```
    sig_clf.fit(train_feature_onehotCoding, y_train)

    predict_y = sig_clf.predict_proba(train_feature_onehotCoding)

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


    #print("How many data points in Test and CV datasets are covered by the {} {} in train
dataset?".format(unique_gene.shape[0], feature ))

    test_coverage=testdf[testdf[feature].isin(list(set(traindf[feature])))].shape[0]
    cv_coverage=cvdf[cvdf[feature].isin(list(set(traindf[feature])))].shape[0]

    test_coverage=testdf[testdf[feature].isin(list(set(traindf[feature])))].shape[0]
    cv_coverage=cvdf[cvdf[feature].isin(list(set(traindf[feature])))].shape[0]

    print('In test data',test_coverage, 'out of',testdf.shape[0], ":",(test_coverage/testdf.shape[0
])*100)
    print('In cross validation data',cv_coverage, 'out of ',cvdf.shape[0],":" ,(cv_coverage/cvdf.sh
ape[0])*100)

    return
(train_feature_onehotCoding,test_feature_onehotCoding,cv_feature_onehotCoding,train_feature_respons
eCoding,test_feature_responseCoding,cv_feature_responseCoding)
```

In [136]:

```
train_gene_feature_onehotCoding,test_gene_feature_onehotCoding,cv_gene_feature_onehotCoding,train_
gene_feature_responseCoding,test_gene_feature_responseCoding,cv_gene_feature_responseCoding =  do_
univariant_analysis(train_df,test_df,cv_df,'Gene')
```

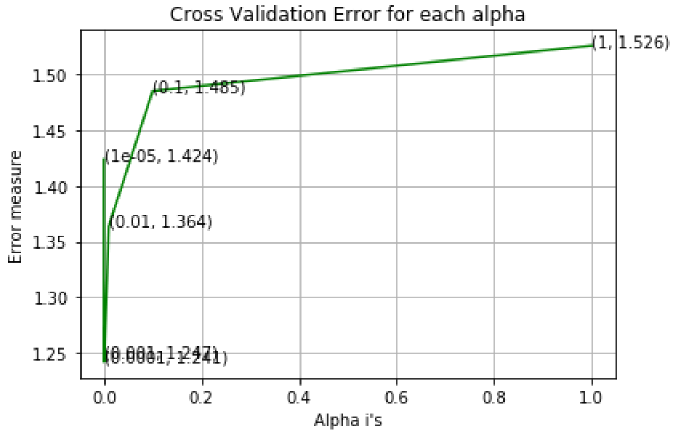
Ans: There are 228 different categories of Gene in the train data, and they are distibuted as foll
ows





train_feature_responseCoding is converted feature using respone coding method. The shape of Gene f
eature: (2124, 9)

```
eature: (2124, 9)
train_feature_onehotCoding is converted feature using one-hot encoding method. The shape of Gene f
eature: (2124, 135)
For values of alpha =  1e-05 The log loss is: 1.4238334608732697
For values of alpha =  0.0001 The log loss is: 1.2413947202634246
For values of alpha =  0.001 The log loss is: 1.2467178725414017
For values of alpha =  0.01 The log loss is: 1.3637461120241208
For values of alpha =  0.1 The log loss is: 1.4854615846658097
For values of alpha =  1 The log loss is: 1.5262743976929538
```

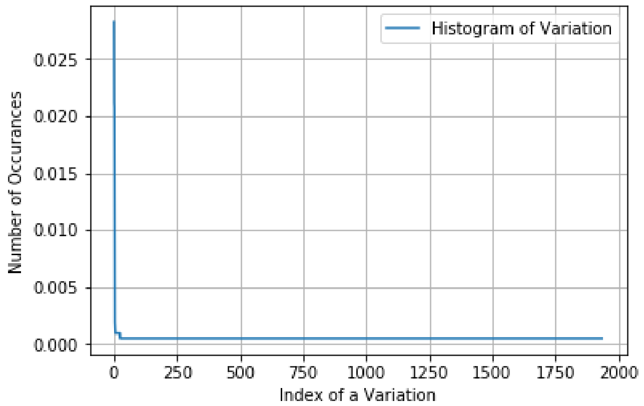


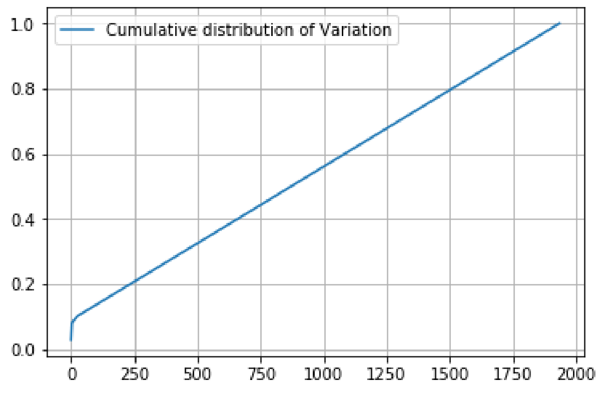

```
For values of best alpha = 0.0001 , The train log loss is:1.0999089780672509
For values of best alpha = 0.0001  The cross validation log loss is:1.2413947202634246
For values of best alpha = 0.0001  The test validation log loss is:1.2138095027235971
In test data 638 out of 665 : 95.93984962406014
In cross validation data 513 out of  532 : 96.42857142857143
```

In [137]:

```
train_vari_feature_onehotCoding,test_vari_feature_onehotCoding,cv_vari_feature_onehotCoding,train_
vari_feature_responseCoding,test_vari_feature_responseCoding,cv_vari_feature_responseCoding = do_u
nivariant_analysis(train_df,test_df,cv_df,'Variation')
```

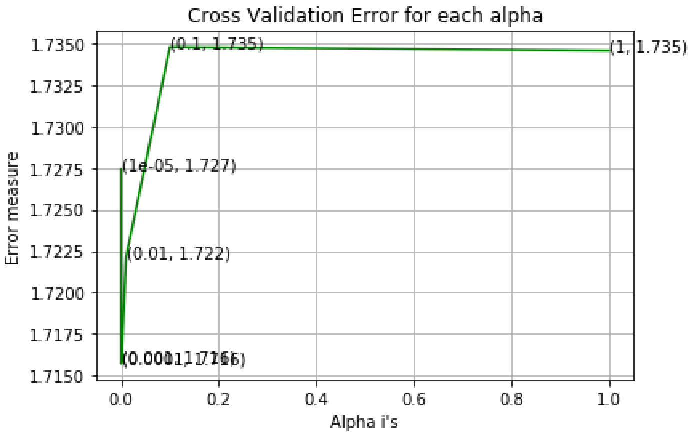
```
Ans: There are 1935 different categories of Variation in the train data, and they are distibuted a
s follows
```






```
train_feature_responseCoding is converted feature using respone coding method. The shape of
```

```
Variation feature: (2124, 9)
train_feature_onehotCoding is converted feature using one-hot encoding method. The shape of
Variation feature: (2124, 22)
For values of alpha =  1e-05 The log loss is: 1.7274108391340375
For values of alpha =  0.0001 The log loss is: 1.7157075783113975
For values of alpha =  0.001 The log loss is: 1.715820236087464
For values of alpha =  0.01 The log loss is: 1.7221054565338185
For values of alpha =  0.1 The log loss is: 1.7347762058040637
For values of alpha =  1 The log loss is: 1.7345531156561076
```




```
For values of best alpha = 0.0001 , The train log loss is:1.6938068428519726
For values of best alpha = 0.0001  The cross validation log loss is:1.7157075783113975
For values of best alpha = 0.0001  The test validation log loss is:1.7091357512544432
In test data 66 out of 665 : 9.924812030075188
In cross validation data 61 out of  532 : 11.466165413533833
```

In [138]:

```python
# building a TfidfVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = TfidfVectorizer(min_df=3)
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])

train_text_features= text_vectorizer.vocabulary_.keys()
print(train_text_feature_onehotCoding.shape)
text_fea_dict = text_vectorizer.vocabulary_
print("Total number of unique words in train data :", len(train_text_features))
```

```
(2124, 53747)
Total number of unique words in train data : 53747
```

In [139]:

```python
import collections

def extract_dictionary_paddle(df):

    for i, row in df.iterrows():
        return collections.Counter(row['TEXT'].split())
def classwise_feat_dict(df):
    dict_list = []
    #as total classes are 1 to 9
    for i in range(1,10):
        cls_text = df[df['Class']==i]
        # build a word dict based on the words in that class
        dict_list.append((i,extract_dictionary_paddle(cls_text)))
    final_dict_list = dict(dict_list)
    return final_dict_list
```

In [140]:

```python
cls_feat_dict = classwise_feat_dict(train_df)
total_dict = extract_dictionary_paddle(train_df)
```

In [141]:

```
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(((cls_feat_dict.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 [142]:

```
#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
test_text_feature_responseCoding  = get_text_responsecoding(test_df)
cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)
```

In [143]:

```
train_text_feature_responseCoding =
(train_text_feature_responseCoding.T/train_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding =
(test_text_feature_responseCoding.T/test_text_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_responseCoding.
sum(axis=1)).T
```

In [144]:

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

In [145]:

```
train_text_feature_onehotCoding.shape
```

Out[145]:

```
(2124, 53747)
```

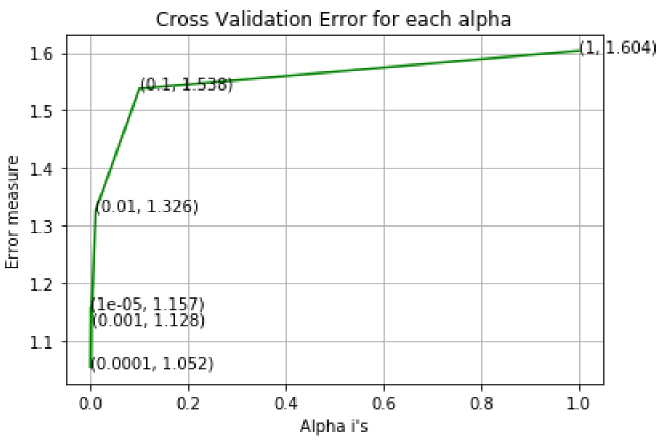
Calculating log-loss for Text feature ;

In [146]:

```
# Train a Logistic regression+Calibration model using text features whicha re on-hot encoded
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_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.clas
ses_, 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: 1.15656723270288
For values of alpha =  0.0001 The log loss is: 1.0521353483999012
For values of alpha =  0.001 The log loss is: 1.1278381003624662
For values of alpha =  0.01 The log loss is: 1.325773207092883
For values of alpha =  0.1 The log loss is: 1.5378187124269684
For values of alpha =  1 The log loss is: 1.6036611171553028
```




```
For values of best alpha =  0.0001 The train log loss is: 0.6840995945614776
For values of best alpha =  0.0001 The cross validation log loss is: 1.0521353483999012
For values of best alpha =  0.0001 The test log loss is: 1.0253319798339153
```

In [147]:

```
def get_intersec_text(df):
    vect = CountVectorizer(analyzer='word',min_df=3)
    vect.fit_transform(df['TEXT'])
    df_feat = vect.get_feature_names()
    total_interset_len = len(set(text_fea_dict.keys()) & set(vect.vocabulary_.keys()))
    return len(set(vect.vocabulary_.keys())),total_interset_len
```

In [148]:

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

```
97.142 % of word of test data appeared in train data
```

In [149]:

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

```
98.156 % of word of Cross Validation appeared in train data
```

Machine Learning Models

Stack the train,test and CV of each feature to create input to ML models ;

e.g train_x_onehot_encoding = train_gene + train_vari + train_text

In [150]:

```
train_gene_var_onehotCoding =
hstack((train_gene_feature_onehotCoding,train_vari_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_vari_feature_onehotCoding
))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_vari_feature_onehotCoding))
```

In [151]:

```
train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_onehotCoding)).tocs
r()
```

In [154]:

```
train_x_onehotCoding.shape
```

Out[154]:

```
(2124, 53904)
```

In [155]:

```
train_y = np.array(list(train_df['Class']))

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

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

In [156]:

```
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, 53904)
(number of data points * number of features) in test data =  (665, 53904)
(number of data points * number of features) in cross validation data = (532, 53904)
```

Base Line Model

In [179]:

```
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(min_df=3)

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

    fea1_len = len(gene_vec.get_feature_names())
    fea2_len = len(var_vec.get_feature_names())
    fea3_len = len(text_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 in gene else False
            if yes_no:
                word_present += 1
                print(i, "Gene feature [{}] present in test data point [{}]".format(word,yes_no))
        elif (v < fea1_len+fea2_len):
            word = var_vec.get_feature_names()[v-(fea1_len)]
            yes_no = True if word in var else False
            if yes_no:
                word_present += 1
                print(i, "variation feature [{}] present in test data point [{}]".format(word,yes_r
o))
        else:
            word = text_vec.get_feature_names()[v-(fea1_len+fea2_len)]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
                print(i, "Text feature [{}] present in test data point [{}]".format(word,yes_no))

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

In [165]:

```
##Some helper functions to use ;

def
CalculateLoss_PredictAndPlot(alpha,algo,method,X_train,X_test,X_cv,y_train,y_test,y_cv,test_point_i
ndex,no_feature):

    #alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
    cv_log_error_array = []
    for i in alpha:
        print("for alpha =", i)
        if algo=='MultinomialNB':
            clf = MultinomialNB(alpha=i)
        if  algo=='LogisticRegressionBalanced':
            clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log',
random_state=42)
        if  algo=='LinearSVM':
            clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='hinge', rand
om_state=42)
        if algo=='KNeighborsClassifier':
            clf = KNeighborsClassifier(n_neighbors=i)
        clf.fit(X_train, y_train)
        sig_clf = CalibratedClassifierCV(clf, method=method)
        sig_clf.fit(X_train, y_train)
        sig_clf_probs = sig_clf.predict_proba(X_cv)
        cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=clf.classes_, eps=1e-15))
        # to avoid rounding error while multiplying probabilites we use log-probability estimates
        print("Log Loss on CV :",log_loss(y_cv, 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()

    #Get the best alpha where loss is minimum
    best_alpha = np.argmin(cv_log_error_array)
    if algo=='MultinomialNB':
        clf = MultinomialNB(alpha=alpha[best_alpha])
    if  algo=='LogisticRegressionBalanced':
        clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='l
og', random_state=42)
    if  algo=='LinearSVM':
        clf = SGDClassifier( class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='
hinge', random_state=42)
    if algo=='KNeighborsClassifier':
        clf = KNeighborsClassifier(n_neighbors=i)
```

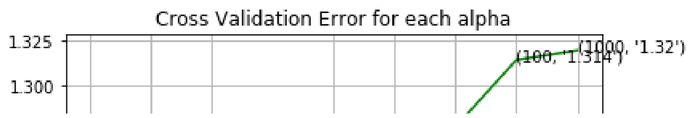
```
    clf.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(clf, method=method)
    sig_clf.fit(X_train, y_train)


    predict_y = sig_clf.predict_proba(X_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(X_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(X_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))

    print("Number of missclassified points in Cross Validation :", np.count_nonzero((sig_clf.predict(X_cv)- y_cv))/y_cv.shape[0])

    plot_confusion_matrix(y_cv, sig_clf.predict(X_cv.toarray()))



    predicted_cls = sig_clf.predict(X_test[test_point_index])
    print("Predicted Class :", predicted_cls[0])
    print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(X_test[test_point_index]),4))
    print("Actual Class :", y_test[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)
```

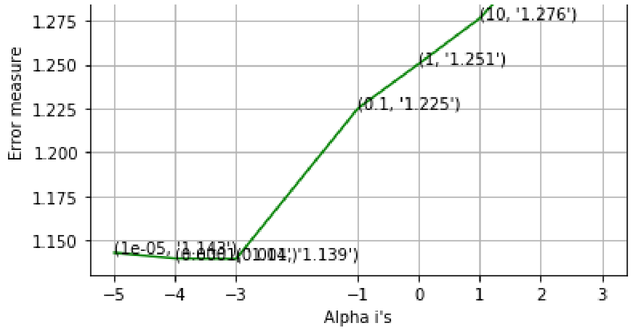
Naive Bayes

In [181]:

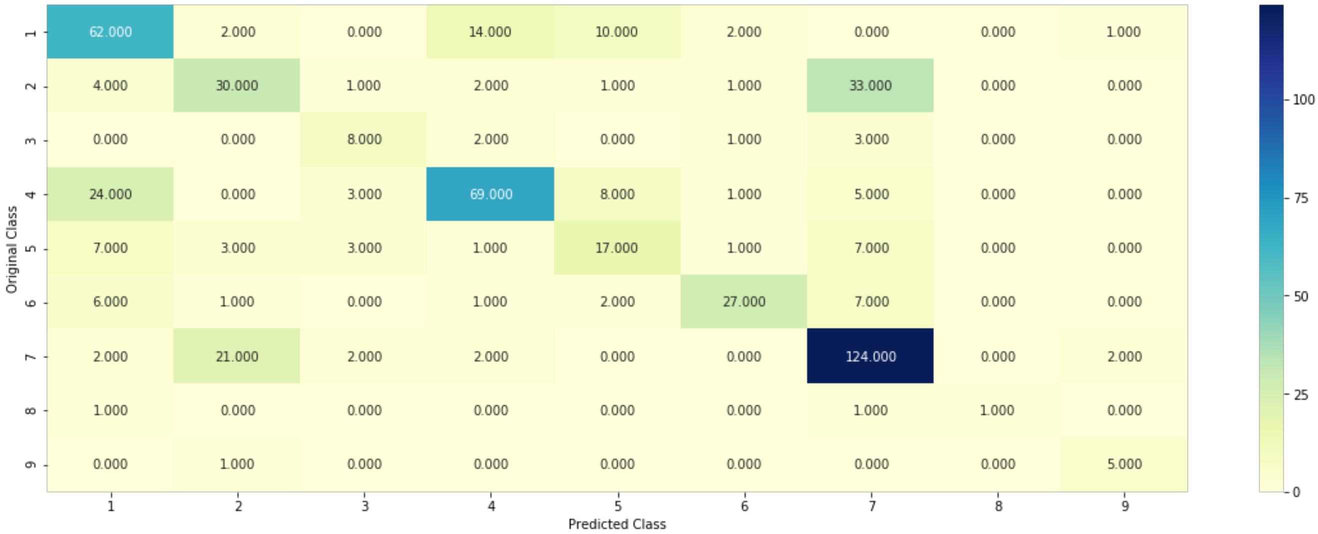
```
#Lets fit model using Naive Bayes ;
# First find best hyper parameter alpha ;then fit model using best alpha;
# then finally find loss on train , test and cv
# lastly plot confusion matrix to draw conclusions using GNB algo;


alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
algo = 'MultinomialNB'
method = 'sigmoid'
test_point_index = 1
no_feature = 100

CalculateLoss_PredictAndPlot(alpha,algo,method,train_x_onehotCoding,test_x_onehotCoding,cv_x_onehotCoding,y_train,y_test,y_cv,test_point_index,no_feature)
```

```
for alpha = 1e-05
Log Loss on CV : 1.1429917580619757
for alpha = 0.0001
Log Loss on CV : 1.1396982177756838
for alpha = 0.001
Log Loss on CV : 1.1393990457022654
for alpha = 0.1
Log Loss on CV : 1.2250816021006667
for alpha = 1
Log Loss on CV : 1.250513540348645
for alpha = 10
Log Loss on CV : 1.2763397190539703
for alpha = 100
Log Loss on CV : 1.314227452844711
for alpha = 1000
Log Loss on CV : 1.319628993940158
```

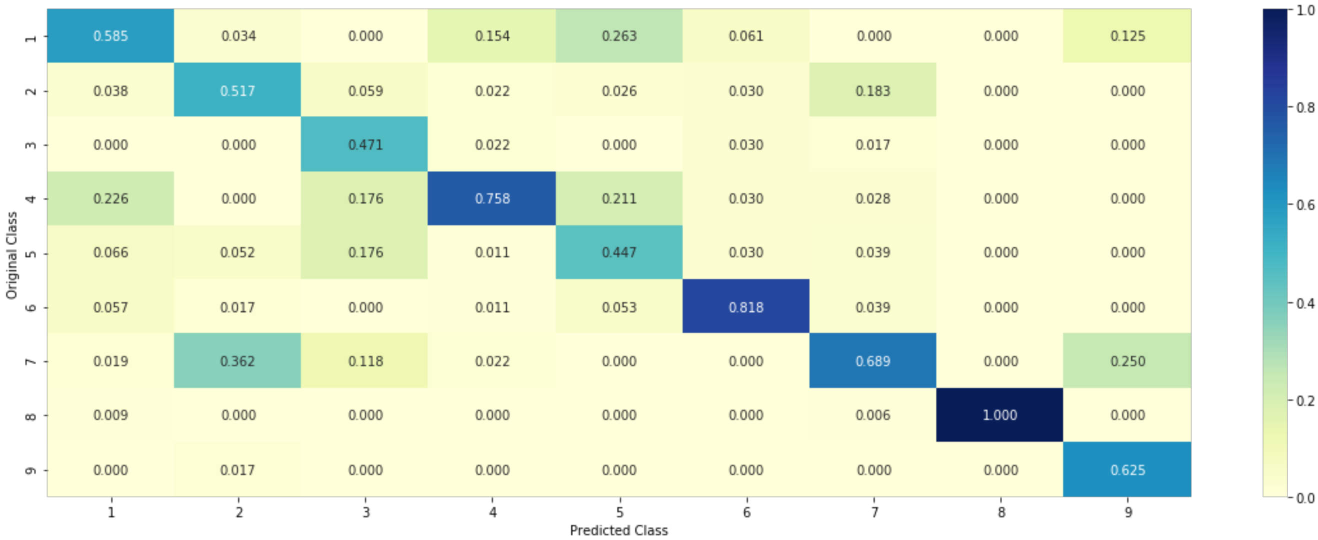




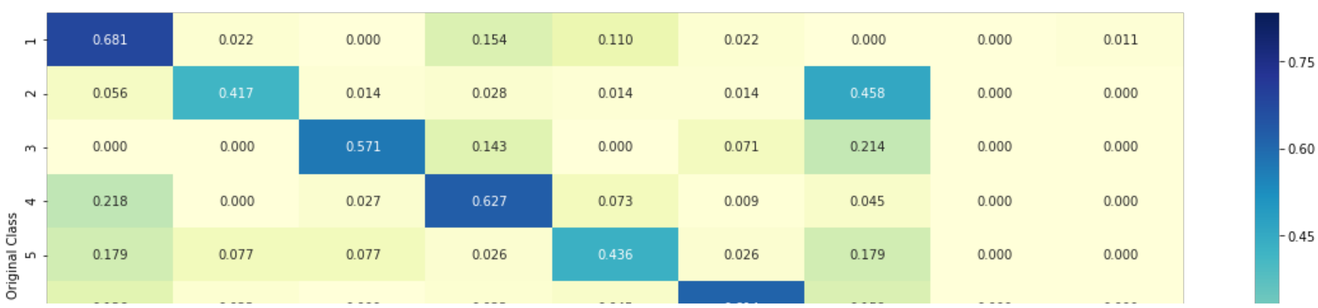

```
For values of best alpha =  0.001 The train log loss is: 0.8365409069321498
For values of best alpha =  0.001 The cross validation log loss is: 1.1393990457022654
For values of best alpha =  0.001 The test log loss is: 1.1274276046650675
Number of missclassified points in Cross Validation : 0.35526315789473684
-------------------- Confusion matrix --------------------
```

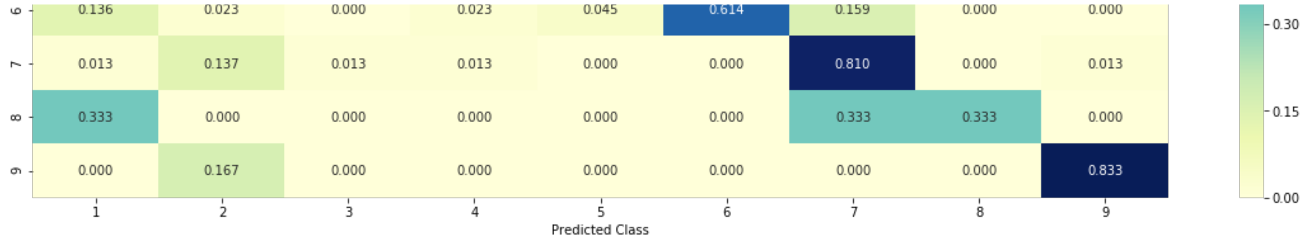



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





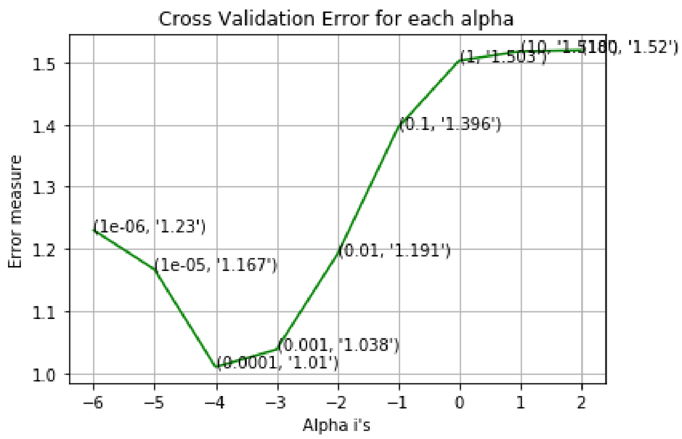

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0602 0.0594 0.0149 0.7139 0.0342 0.0357 0.075  0.0044 0.0024]]
Actual Class : 4
--------------------------------------------------
73 Text feature [affect] present in test data point [True]
95 Text feature [implementation] present in test data point [True]
Out of the top  100  features  2 are present in query point
```

Logistic Regression with Class balancing

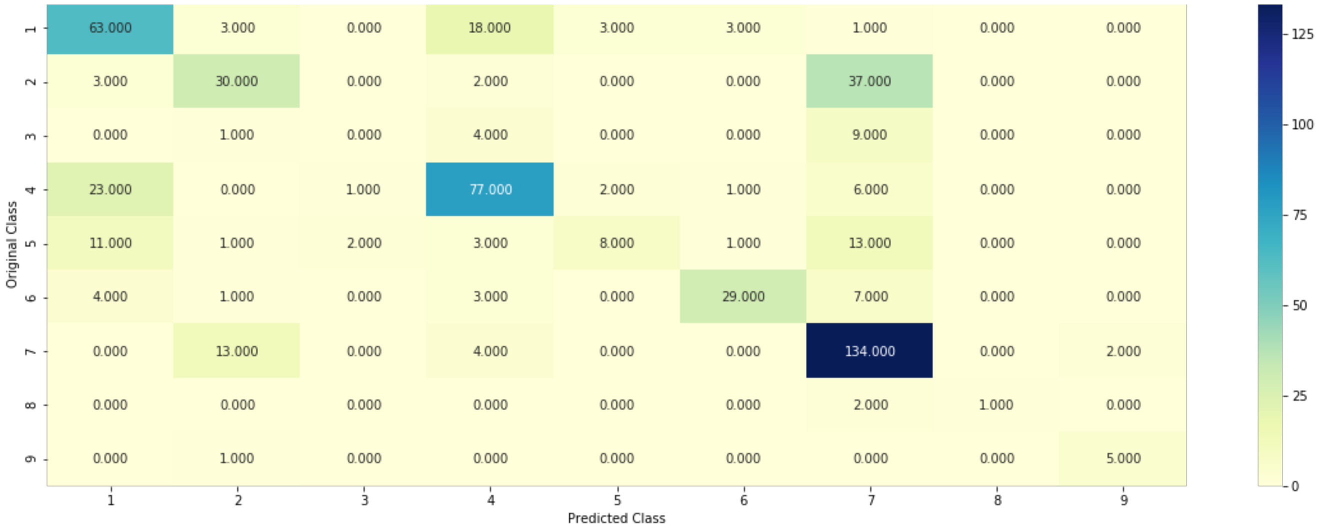
In [182]:

```
alpha = [10 ** x for x in range(-6, 3)]
algo = 'LogisticRegressionBalanced'
method = 'sigmoid'
test_point_index = 1
no_feature = 100

CalculateLoss_PredictAndPlot(alpha,algo,method,train_x_onehotCoding,test_x_onehotCoding,cv_x_onehot
Coding,y_train,y_test,y_cv,test_point_index,no_feature)
```

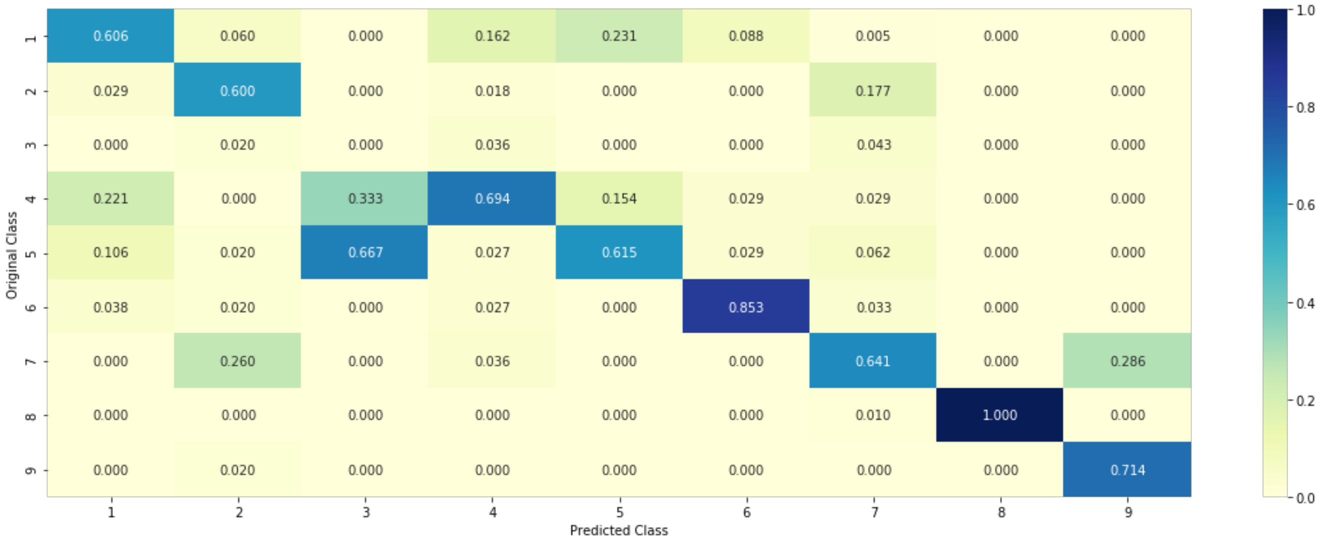
```
for alpha = 1e-06
Log Loss on CV : 1.2302443159660155
for alpha = 1e-05
Log Loss on CV : 1.1667084913677321
for alpha = 0.0001
Log Loss on CV : 1.0098329734272407
for alpha = 0.001
Log Loss on CV : 1.0377677848121085
for alpha = 0.01
Log Loss on CV : 1.1907930509441123
for alpha = 0.1
Log Loss on CV : 1.3956357214795383
for alpha = 1
Log Loss on CV : 1.5030338962233751
for alpha = 10
Log Loss on CV : 1.5181447281779874
for alpha = 100
Log Loss on CV : 1.5199312865020367
```



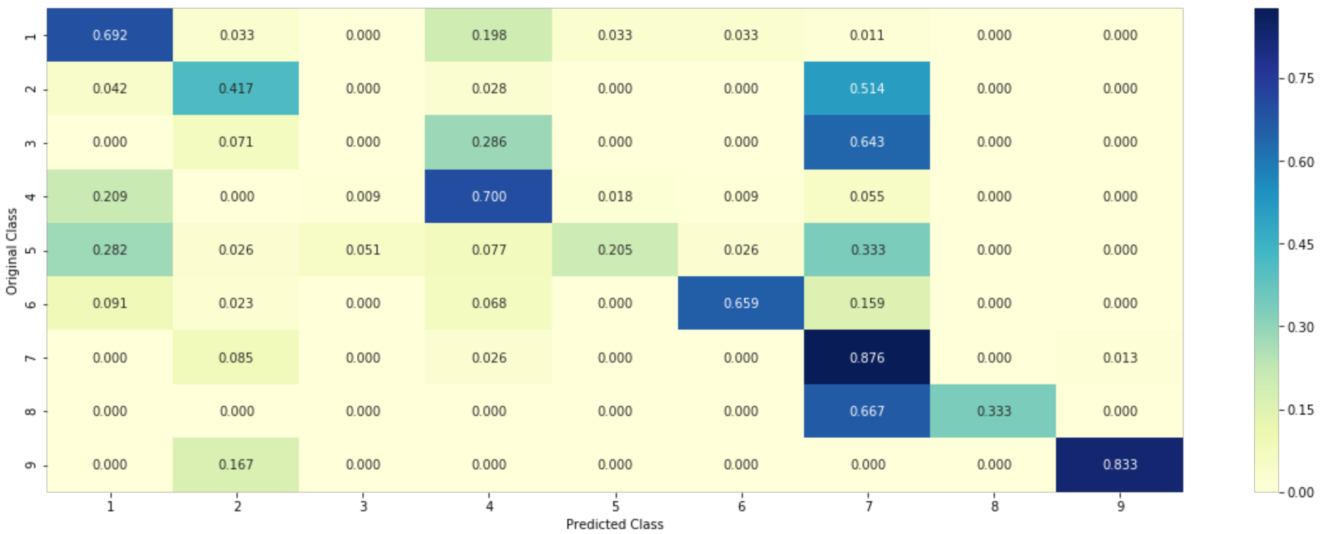

```
For values of best alpha =  0.0001 The train log loss is: 0.680388841421323
For values of best alpha =  0.0001 The cross validation log loss is: 1.0098329734272407
For values of best alpha =  0.0001 The test log loss is: 0.9520150818980699
Number of missclassified points in Cross Validation : 0.34774436090225563
-------------------- Confusion matrix --------------------
```




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




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



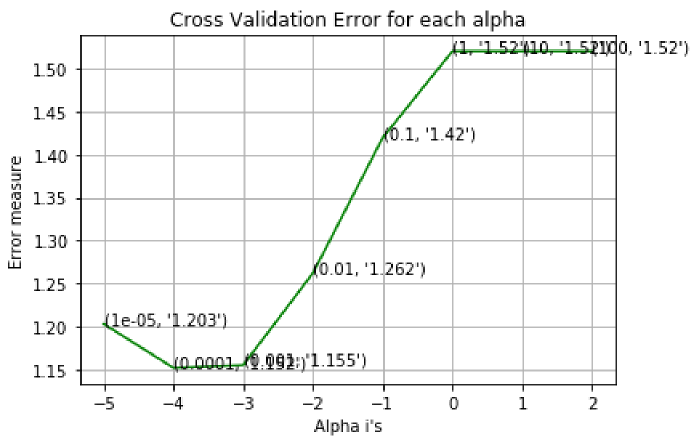

```
Predicted Class : 4
Predicted Class Probabilities: [[0.0248 0.012  0.0624 0.8428 0.0286 0.0086 0.0151 0.0036 0.0021]]
Actual Class : 4
--------------------------------------------------
56 Text feature [implementation] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

In [183]:

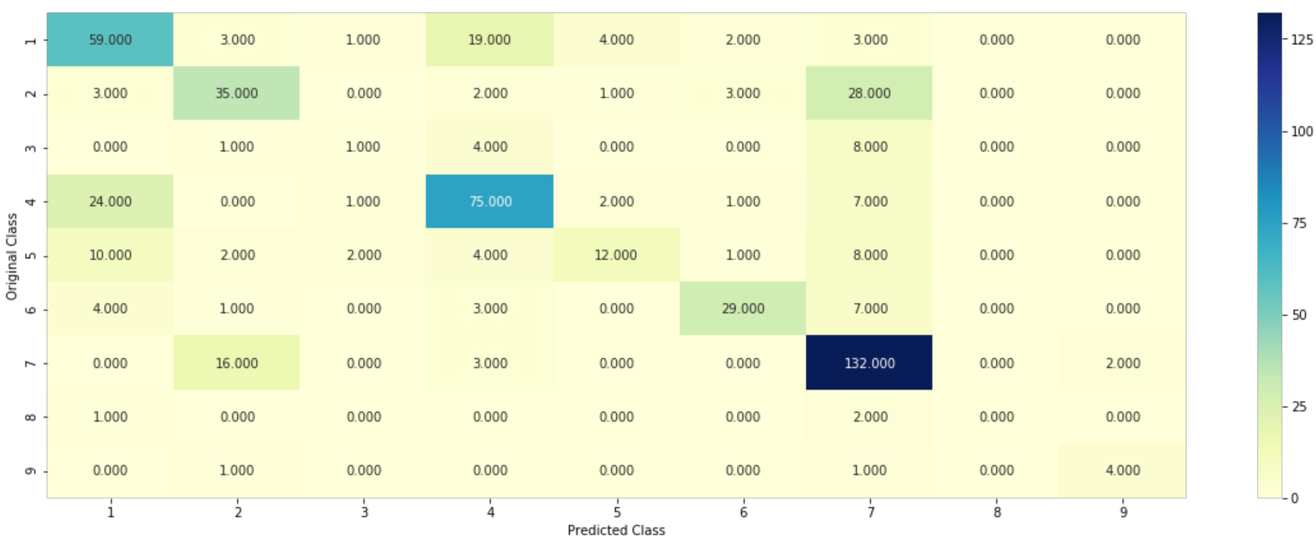
```
alpha = [10 ** x for x in range(-5, 3)]
```

```
alpha = [10 ** x for x in range(-5, 3)]
algo = 'LinearSVM'
method = 'sigmoid'
test_point_index = 100
no_feature = 100

CalculateLoss_PredictAndPlot(alpha,algo,method,train_x_onehotCoding,test_x_onehotCoding,cv_x_onehotCoding,y_train,y_test,y_cv,test_point_index,no_feature)
```

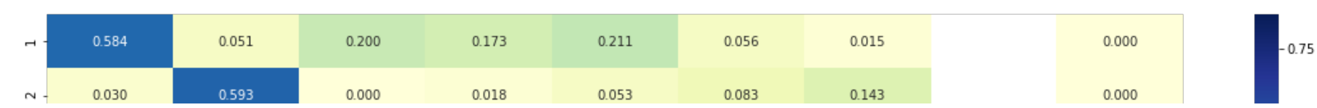
```
for alpha = 1e-05
Log Loss on CV : 1.202881672301905
for alpha = 0.0001
Log Loss on CV : 1.151842955750128
for alpha = 0.001
Log Loss on CV : 1.1552599160490415
for alpha = 0.01
Log Loss on CV : 1.2623486574308813
for alpha = 0.1
Log Loss on CV : 1.419673197587919
for alpha = 1
Log Loss on CV : 1.5204100278915218
for alpha = 10
Log Loss on CV : 1.520409925050722
for alpha = 100
Log Loss on CV : 1.5204098074246037
```

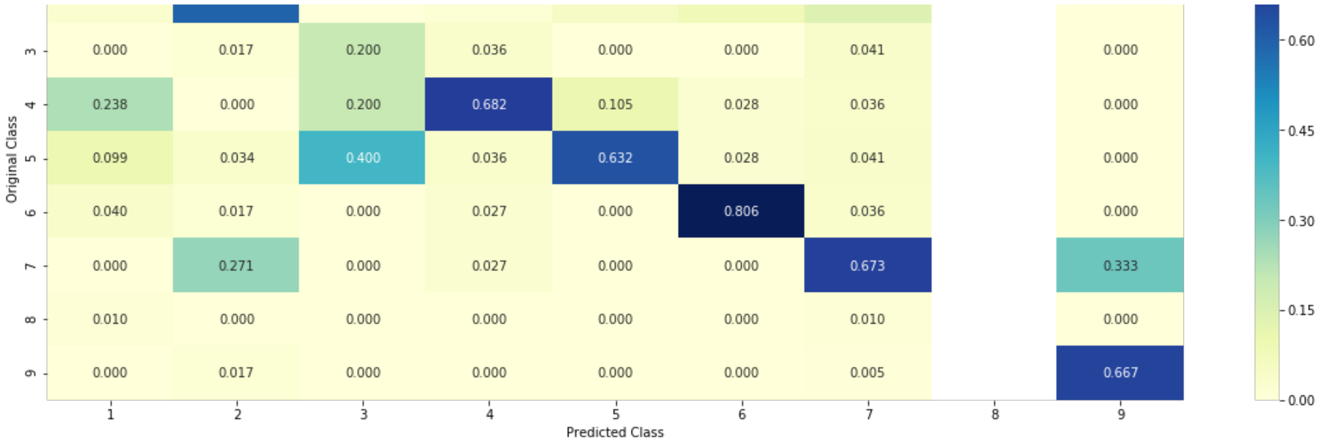



```
For values of best alpha =  0.0001 The train log loss is: 0.8460379028679363
For values of best alpha =  0.0001 The cross validation log loss is: 1.151842955750128
For values of best alpha =  0.0001 The test log loss is: 1.10873020249378
Number of missclassified points in Cross Validation : 0.34774436090225563
-------------------- Confusion matrix --------------------
```

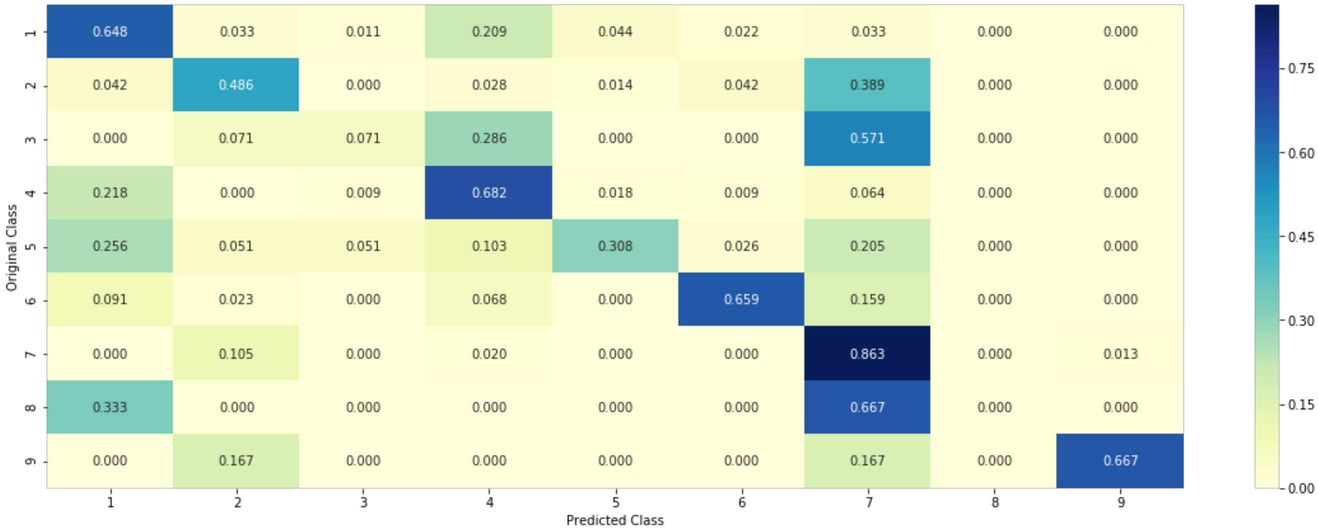



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```






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




```
Predicted Class : 7
Predicted Class Probabilities: [[0.1125 0.1665 0.135  0.1177 0.0612 0.0662 0.3328 0.0051 0.0032]]
Actual Class : 3
--------------------------------------------------
43 Text feature [secondary] present in test data point [True]
69 Text feature [dna] present in test data point [True]
71 Text feature [seeded] present in test data point [True]
76 Text feature [analyses] present in test data point [True]
Out of the top  100  features  4 are present in query point
```

In [167]:

```
clf = SGDClassifier( class_weight='balanced', alpha=0.0001, penalty='l2', loss='hinge', random_stat
e=42)
```

In [168]:

```
clf.fit(train_x_onehotCoding,train_y)
```

Out[168]:

```
SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
       epsilon=0.1, eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', max_iter=None, n_iter=None,
       n_jobs=1, penalty='l2', power_t=0.5, random_state=42, shuffle=True,
       tol=None, verbose=0, warm_start=False)
```

In [176]:

```
(np.argsort(-clf.coef_)[6])[:100]
```

Out[176]:

```
array([20980, 12459,    84, 27858, 34051, 43465,    31,   146,   107,
```

```
        140,    61, 18763,  7125,   141,   154, 23780, 31787,    40,
         68, 35582, 33886,    52,    80, 10739, 46758, 24748, 23371,
      44779,  7117,  6521, 30899, 50196, 19551,   137, 31799, 19597,
          4, 43483, 21329, 35777, 18236, 19074, 42029, 47056,    16,
        143,    55,    95, 15912, 20176, 23050, 26137, 52991, 15357,
      45796,    62, 42336, 28215, 20900,    53, 25007, 37159,  8111,
      43334, 46398,   149, 20288, 46914, 30450, 19583, 19792, 47093,
      53740, 10945, 23364,   152,  9705, 25494, 14838, 35571, 19095,
      12388, 17743, 36147,  8744,   120, 53100, 45811, 46399,  7118,
       9983,    20, 24121, 18194, 24530, 48385, 34922,  8199, 10684,
      27969], dtype=int64)
```

Stacking Classifier

In [184]:

```
##### Find best alpha for stacking classifier ;
clf1 = SGDClassifier(alpha=0.0001, 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=0.0001, penalty='l2', loss='hinge', class_weight='balanced', random_stat
e=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_onehot
Coding))))
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]

for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_p
robas=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sc
lf.predict_proba(cv_x_onehotCoding))))
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
```

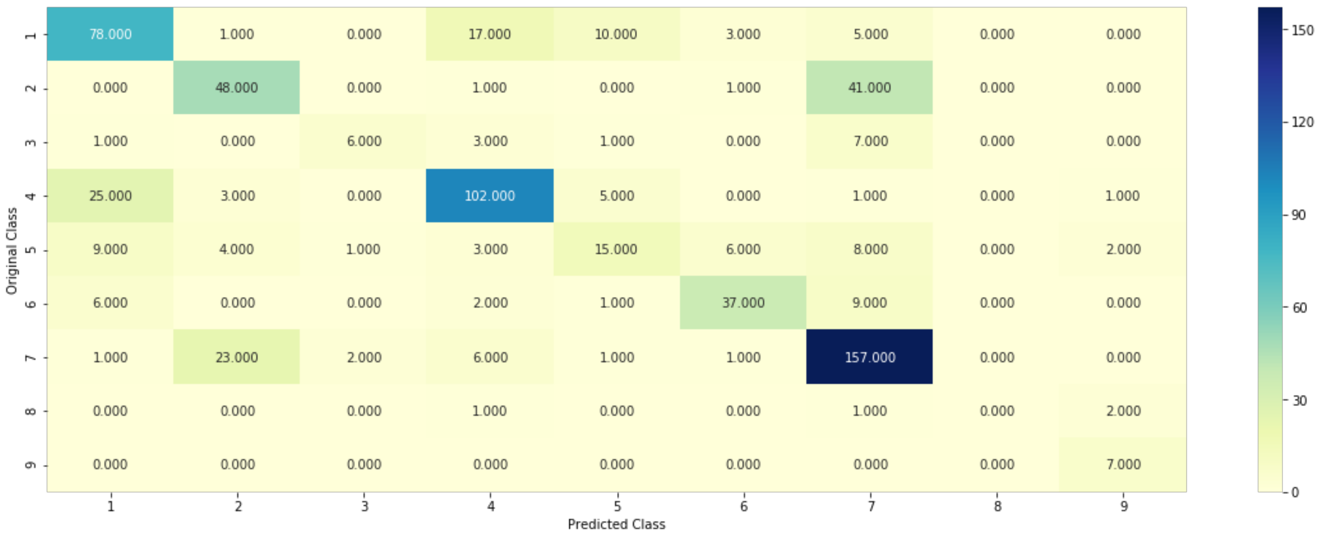
```
Logistic Regression :  Log Loss: 0.99
Support vector machines : Log Loss: 1.14
Naive Bayes : Log Loss: 1.14
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.176
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.021
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.473
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.065
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.012
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.047
```

In [186]:

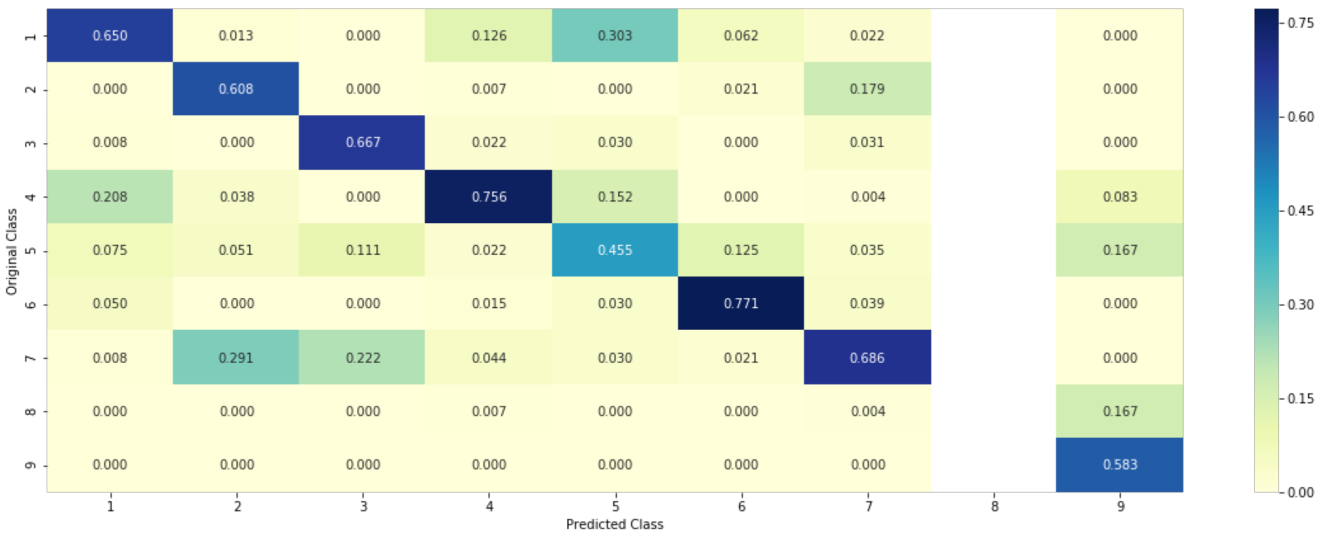
```
##From above cell we can see that best alpha is 0.1 which has lowest log loss ;using the same for
testing ;
lr = LogisticRegression(C=1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba
s=True)
sclf.fit(train_x_onehotCoding, train_y)
```

```
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))
```

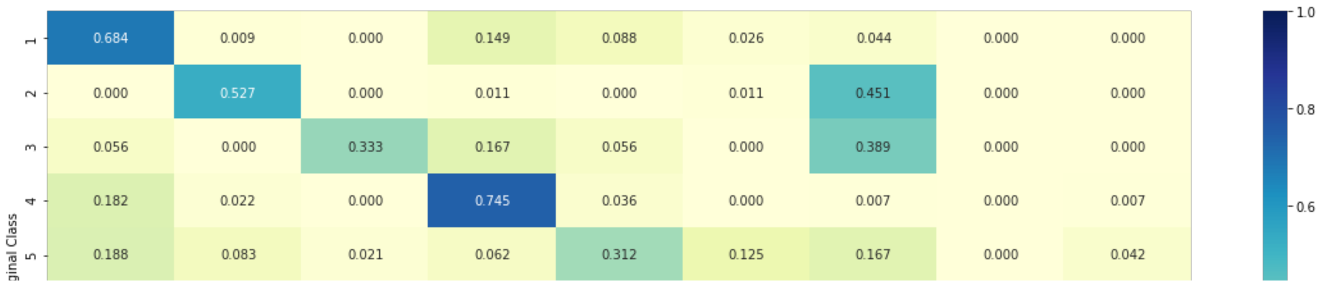
```
Log loss (train) on the stacking classifier : 0.5394468140969174
Log loss (CV) on the stacking classifier : 1.0117213438650667
Log loss (test) on the stacking classifier : 0.9402773228252584
Number of missclassified point : 0.3233082706766917
-------------------- Confusion matrix --------------------
```

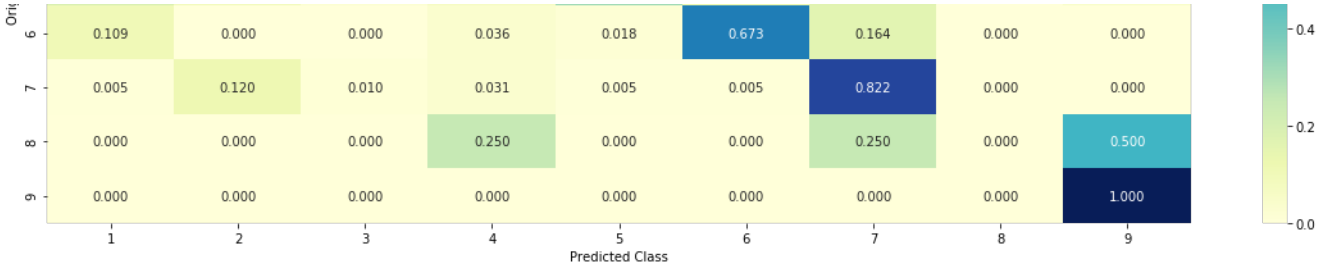



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





Conclusion ;

Here is summary of solution we reached so far ;

1. Our aim was to find probabilities of output classes 1-9
2. We first tried to find simulation based on Random classifier and calculated its log loss that given us general idea about below how much log loss value we can conclulde model is good performing;
3. we started analysing each feature seperately for Gene,variation and text
4. We used one hot encoding technique(Tf-Idf) based
5. finally , we tried with few ML models : NB , Logistic Regression and Linear SVM to get different log -loss values
6. We finally combined above all models with best hyperparameter alpha and created stacked classifier , which finally achieved

   log-loss of 0.94 which can be good indication that model is able to classifiy pretty correctly(perfect claasifier log-loss =0) .