Personalized Cancer Diagnosis

Chinmay Sathe

19MT0119

—

NEURAL NETWORKS & DEEP LEARNING LAB
(MCC542)

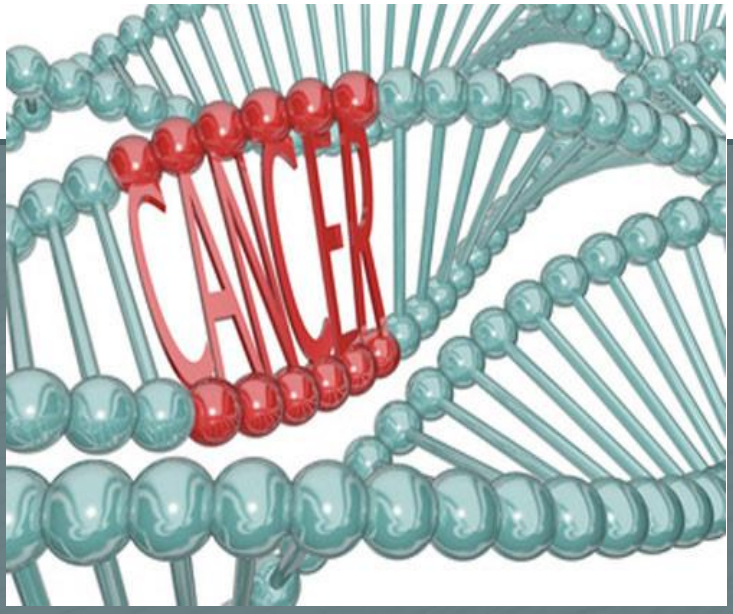
—

Prof. Subhashis Chatterjee

Abstract

Once sequenced, a cancer tumor can have thousands of genetic mutations. But the challenge is distinguishing the mutations that contribute to tumor growth from the neutral mutations. Currently this interpretation of genetic mutations is being done manually.

Objective here is to develop a Machine Learning algorithm that automatically classifies genetic variations.



1 INTRODUCTION

A lot has been said during the past several years about how genetic testing is going to disrupt the way diseases like cancer are treated. But this is only partially happening due to the huge amount of manual work still required.

Currently the interpretation of genetic mutations is being done manually. This is a very time-consuming task where a clinical pathologist has to manually review and classify every single genetic mutation based on evidence from text-based clinical literature.

Iker Heurga, who is the Dir. Engineering and Applied Machine Learning at MSKCC, New York, New York, United States explains the problem as follows:
"We understand that analysing text represents a difficult challenge, but believe it or not is the current state of the art when it comes to interpretation of genetic variants.

The workflow is as follows:

1. A molecular pathologist selects a list of genetic variations of interest that he/she want to analyse.

2. The molecular pathologist searches for evidence in the 'medical literature' that somehow are relevant to the genetic variations of interest.

3. Finally, this molecular pathologist spends a huge amount of time analysing the evidence related to each of the variations to classify them.

Models tried are:
- Naïve Bayes
- Logistic Regression

Performance Metrics used are:
- Log-loss
- Confusion Matrix

Our goal here is to replace the third step, which is also the most time consuming, with a Machine Learning algorithm that, using this knowledge base as a baseline, automatically classifies genetic variations."

In one line the objective can be stated as:
"From gene codes, their variations and text data we have to classify them into any one of the 9 classes."

Thus, this is a **MULTI-CLASS classification problem.**

2 DATA AND FEATURES

2.1 Data

The data is provided by Memorial Sloan Kettering Cancer Centre (MSKCC), which is the largest and oldest private cancer centre in the world. In this dataset, MSKCC has provided an expert-annotated knowledge base where world-class researchers and oncologists have manually annotated thousands of mutations.

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 are having a common column called ID.

The dataset link is provided in notebook attached.

2.2 Features

Data file's information:

training_variants (ID, Gene, Variations, Class)

training_text (ID, Text)

with each file having 3321 data points

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 amino acid change for this mutation

Class: 1-9 the class this genetic mutation has been classified on

Text: Text about the gene code. (only understood by cancer experts)

Gene and variation can be considered as categorical variation.

3 PRE-PROCESSING

3.1 Text Pre-processing

The ‘Text” feature is pre-processed to remove the stop words, special characters, multiple space with a single space. Then all the words were converted to lowercase. All Nan text were replaced with a text formation of 'GENE (space)VARIATION'.

3.2 Data Splitting

Data was split into two parts first, with 80-20 ratio. First part being training data and second being test data.

Further, the training data was split into two parts, again, with the same ratio of 80-20. First part being training data and second being Cross Validation data.

Overall, Data partitioning is as follows:

1. Training Data (64%) – 2124 Data points
2. Test Data (20%) – 665 Data points
3. CV Data (16%) – 532 Data points

3.3 Data Balance Check

Using simple plots between 'Data points per class' & 'Class', it was found that all three data partitioned have IMBALANCED DATA for each class type.

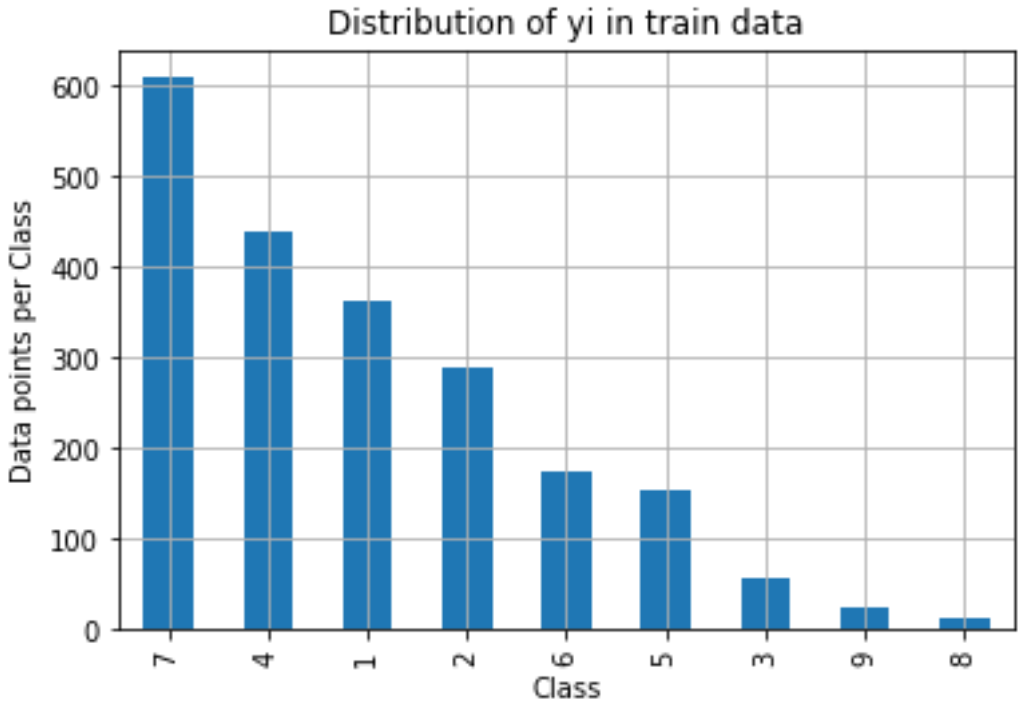


Figure 1: It is clearly visible Class-7 is dominating whole train data set by a large margin. Then comes Class-4, Class-1 till Class-8.

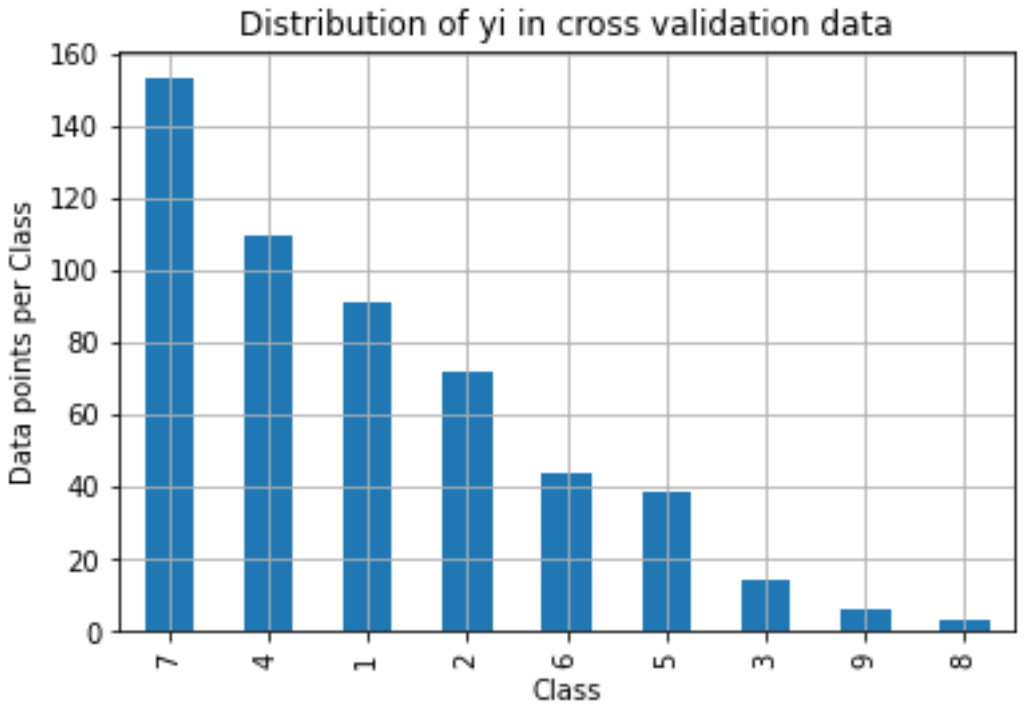


Figure 2: It is clearly visible Class-7 is dominating whole CV data set by a large margin. Then comes Class-4, Class-1 till Class-8.

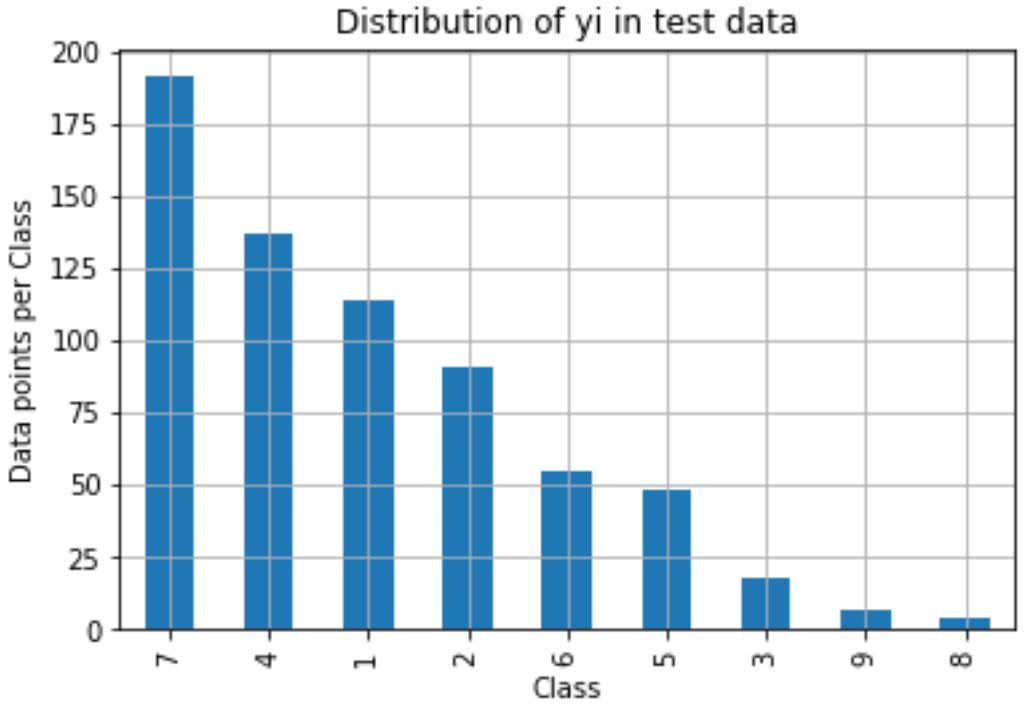


Figure 3: It is clearly visible Class-7 is dominating whole test data set by a large margin. Then comes Class-4, Class-1 till Class-8.

3.4 Encoding the categorical features

Since, the features GENE, VARIATIONS & TEXT were of categorical nature. They were encoded using One-Hot Encoder.

4 MODELS

4.1 Naïve Bayes Model

This model is generally chosen as a baseline for other models. Using Hyperparameter Tuning for alpha, best fitting alpha was found as:

Alpha = 0.001

With log-loss as follows:

Train = 0.4669036753556783
Test = 1.1693677084388148
Validation = 1.2489742954899758

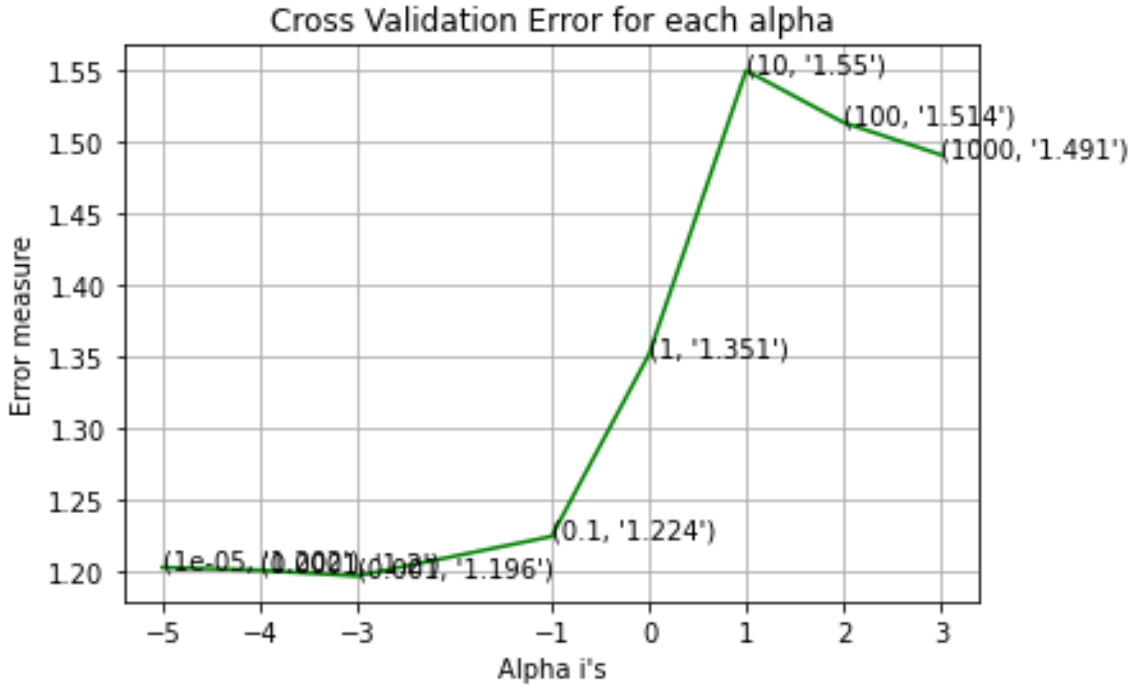


Figure 4: The alpha can be tracked in the graph to be 0.001 when it gives min. Error measure

4.2 Logistic Regression Model (with Balanced Data)

Model was implemented using One Vs Rest strategy. Using Hyperparameter Tuning for alpha, best fitting alpha was found as:

Alpha = 0.0001

With log-loss as follows:

Train = 0.4075301221619987
Test = 0.9740580770216842
Validation = 1.0351697685089027

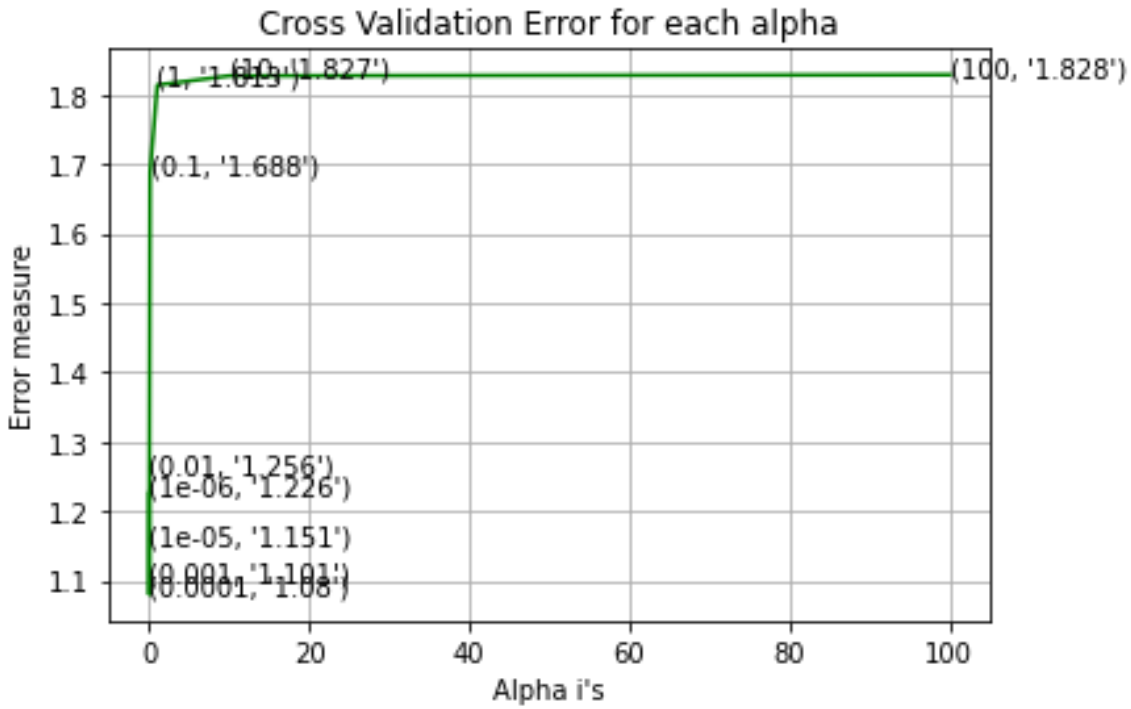


Figure 5: The alpha can be tracked in the graph to be 0.0001 when it gives min. Error measure

5 RESULTS AND DISCUSSIONS

5.1 Using Naïve Bayes

Using this model, the following results were obtained:

1. Log Loss: 1.2489742954899758

2. Number of misclassified points: 0.38533834586466165

3. Confusion Matrix:

This matrix was drawn using seaborn libraries heatmap feature.

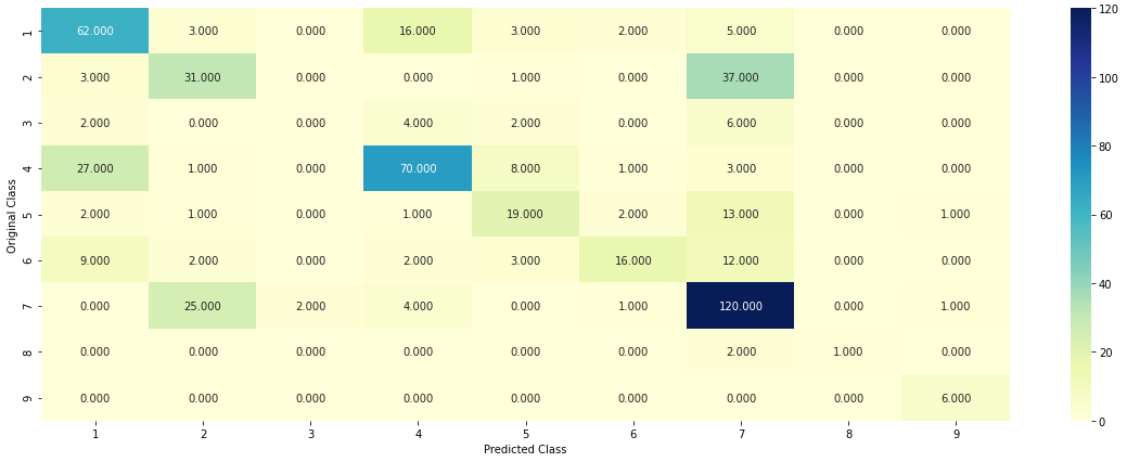


Figure 6: The confusion matrix shows that model seems to be doing well for class-7, class-4 and class-1

5.2 Using Logistic Regression

Using this model, the following results were obtained:

1. Log Loss: 1.0351697685089027

2. Number of misclassified points: 0.3233082706766917

3. Confusion Matrix:

This matrix was drawn using seaborn libraries heatmap feature.

Figure 7: The confusion matrix shows that model seems to be doing well for class-7, class-4 and class-1

5.3 Comparison between both the implementations

S.NO.	MODEL	Train_loss	CV_loss	Test_loss	Misclassified(%)
1	Naive Bayes	0.4669036753556783	1.2489742954899758	1.1693677084388148	38.53383458646616
2	LR With Class Balancing	0.4075301221619987	1.0351697685089027	0.9740580770216842	32.33082706766917

Table 1: From the column Misclassified (%) the percentage of misclassified data can be seen.

5.4 Conclusion

Based on the model trainings following points can be concluded:

1. Logistic Regression model works better than the Naïve Bayes model by a margin of nearly 6 % while misclassifying data.
2. Logistic Regression also works better in terms of Log-Loss.
3. Still, Logistic Regression is not as efficient as required in the field of medical science.
4. In future, better feature pre-processing techniques can be introduced with more amount of data to decrease the misclassification errors.

6 CODE

Personalized Cancer Diagnosis

by Chinmay Sathe

Link for data : https://www.kaggle.com/c/msk-redefining-cancer-treatment/data (https://www.kaggle.com/c/msk-redefining-cancer-treatment/data)

Files used are :

- training_text.zip
- training_variants.zip

1. Reading Data

[23]:
```
!unzip /kaggle/input/msk-redefining-cancer-treatment/training_text.zip
!unzip /kaggle/input/msk-redefining-cancer-treatment/training_variants.zip
```

```
Archive:  /kaggle/input/msk-redefining-cancer-treatment/training_text.zip
  inflating: training_text
Archive:  /kaggle/input/msk-redefining-cancer-treatment/training_variants.zip
  inflating: training_variants
```

+ Code + Markdown

[24]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
import nltk
nltk.download('stopwords')
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 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
```

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

[25]:
```
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[25]:

	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

[26]:
```
# the id and text data is separated by || thus the change in code below
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[26]:

	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...

2.Exploratory Data Analysis

2.1 Pre-Processing Text

[27]:
```
# the function nlp_preprocessing removes the stopwords, special char,multiple space with space
# it also converts everything to lowercase
stop_words = set(stopwords.words('english'))

def nlp_preprocessing(total_text, index, column):
    if type(total_text) is not int:
        string = ""
        total_text = re.sub('[^a-zA-Z0-9\n]', ' ', total_text)
        total_text = re.sub('\s+',' ', total_text)
        total_text = total_text.lower()
        for word in total_text.split():
            if not word in stop_words:
                string += word + " "
        data_text[column][index] = string
```

[28]:
```
#calling the above function for each text field.
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
```

[29]:
```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

Out[29]:

	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...

[30]:
```
#replacing all NaN text with a text formation of 'GENE VARIATION'
result[result.isnull().any(axis=1)]
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

2.2 Splitting Data into Train (64%), Cross-Validation (16%) and Test(20%)

[31]:
```
y_true = result['Class'].values
result.Gene = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

# 1. split the data into test and train
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true, test_size=0.2)
# 2. split the train data into train and cross validation
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test_size=0.2)
```

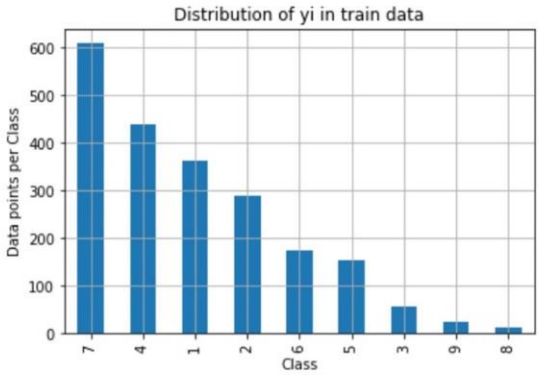
[32]:
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
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
```

2.3 Checking if Data is balanced or not

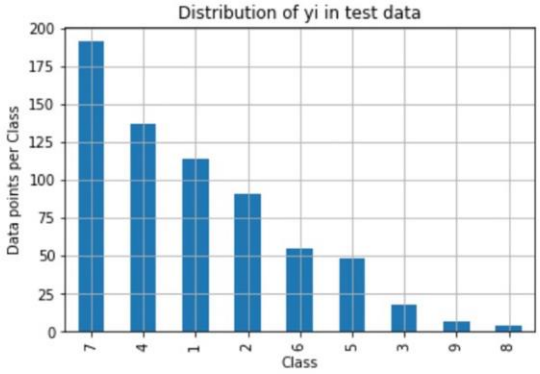
[33]:

```
# 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()

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()
```

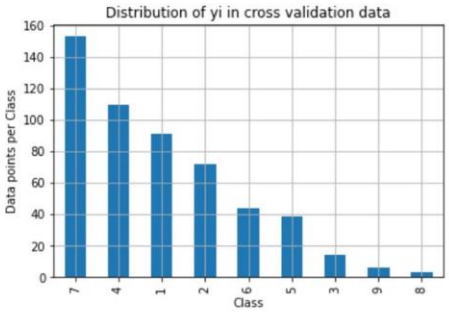


[34]:

```
test_class_distribution = test_df['Class'].value_counts()

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()
```



[35]:
```
cv_class_distribution = cv_df['Class'].value_counts()

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()
```



2.4 Using One-Hot Encoder for GENE

[36]:
```
def get_gv_fea_dict(alpha, feature, df):
    value_count = train_df[feature].value_counts()
    gv_dict = dict()
    for i, denominator in value_count.items():
        vec = []
        for k in range(1,10):
            cls_cnt = train_df.loc[(train_df['Class']==k) & (train_df[feature]==i)]
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))
        gv_dict[i]=vec
    return gv_dict

def get_gv_feature(alpha, feature, df):
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    value_count = train_df[feature].value_counts()
    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])
    return gv_fea
```

[37]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))

# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer()
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'])
```

2.5 Using One-Hot Encoder for VARIATION

[38]:
```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_df))

# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
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'])
```

2.6 Encoding for TEXT

[39]:
```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = TfidfVectorizer(max_features=1000)
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 occured
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 : 1000
```

[40]:
```
#Normalizing train,test and cv
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEXT'])
test_text_feature_onehotCoding = normalize(test_text_feature_onehotCoding, axis=0)

cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])
cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding, axis=0)
```

4. Models

4.1 Some MISC Functions

[41]:

```
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    A =(((C.T)/(C.sum(axis=1))).T)
    B =(C/C.sum(axis=0))
    labels = [1,2,3,4,5,6,7,8,9]
    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()

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

    # for calculating log_loss we willl provide the array of probabilities belongs to 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)

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)

def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec =TfidfVectorizer()
    var_count_vec = TfidfVectorizer()
    text_count_vec = TfidfVectorizer(max_features=1000)

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

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

    word_present = 0
    for i,v in enumerate(indices):
        if (v < fea1_len):
            word = gene_vec.get_feature_names()[v]
            yes_no = True if word == gene else False
            if yes_no:
                word_present += 1
                print(i, "Gene feature [{}] present in test data point [{}]".format(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 [{}] present in test data point [{}]".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 [{}] present in test data point [{}]".format(word,yes_no))

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

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, train_text_feature_onehotCoding)).tocsr()
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']))
```

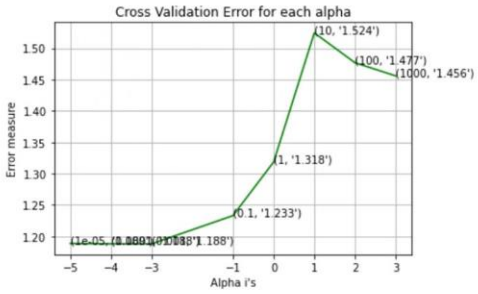
4.2 Naive Bayes

4.2.1. Hyper parameter tuning

[42]:

```
alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability 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.cl
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, label
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.clas
# Variables that will be used in the end to make comparison table of all models
nb_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding), labels=clf.classes_, eps=1e-15)
nb_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=clf.classes_, eps=1e-15)
nb_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding), labels=clf.classes_, eps=1e-15)
```

```
for alpha = 1e-05
Log Loss : 1.1885544007577409
for alpha = 0.0001
Log Loss : 1.18792263008605
for alpha = 0.001
Log Loss : 1.1878970789307897
for alpha = 0.1
Log Loss : 1.2329187302184466
for alpha = 1
Log Loss : 1.318432157940304
for alpha = 10
Log Loss : 1.524313905205868
for alpha = 100
Log Loss : 1.4767034830545769
for alpha = 1000
Log Loss : 1.4557124563620019
```



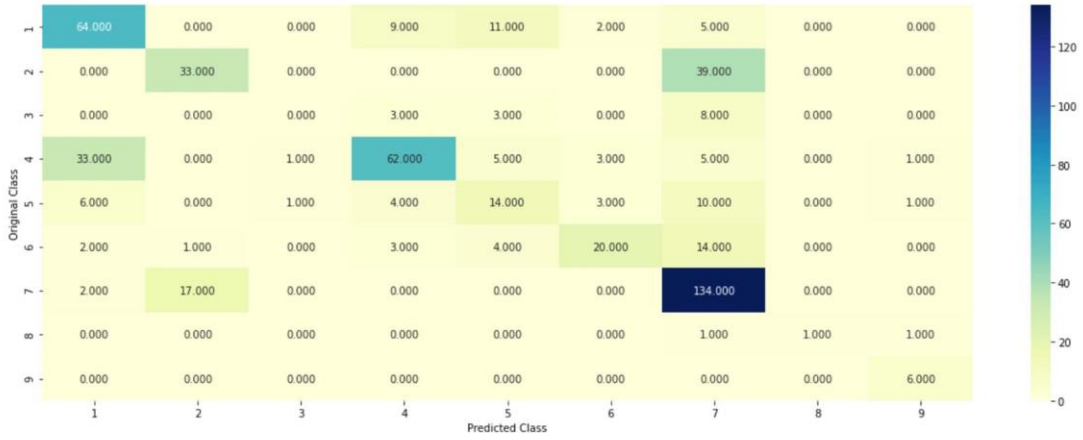

```
For values of best alpha =  0.001 The train log loss is: 0.4399633030131071
For values of best alpha =  0.001 The cross validation log loss is: 1.1878970789307897
For values of best alpha =  0.001 The test log loss is: 1.172029085937199
```

4.2.2. Using the best Hyper parameter found

[43]:

```
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 probabilites we use log-probability estimates
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)- cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))

#Variables that will be used in the end to make comparison table of models
nb_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)- cv_y))/cv_y.shape[0])*100
```

```
Log Loss : 1.1878970789307897
Number of missclassified point : 0.37218045112781956
```



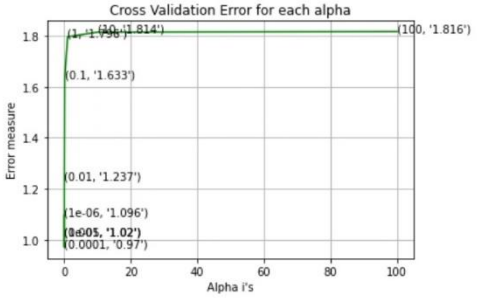
4.3 Logistic Regression with Balanced Data

4.3.1. Hyper parameter tuning

[44]:

```
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 probabilites 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.cl
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, label
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.clas

# Variables that will be used in the end to make comparison table of all models
lr_balance_train = log_loss(y_train, sig_clf.predict_proba(train_x_onehotCoding), labels=clf.classes_, eps=1e-15)
lr_balance_cv = log_loss(y_cv, sig_clf.predict_proba(cv_x_onehotCoding), labels=clf.classes_, eps=1e-15)
lr_balance_test = log_loss(y_test, sig_clf.predict_proba(test_x_onehotCoding), labels=clf.classes_, eps=1e-15)
```

```
for alpha = 1e-06
Log Loss : 1.0961592402537237
for alpha = 1e-05
Log Loss : 1.0195120920942313
for alpha = 0.0001
Log Loss : 0.9701157827989754
for alpha = 0.001
Log Loss : 1.020167688952177
for alpha = 0.01
Log Loss : 1.2373241256680647
for alpha = 0.1
Log Loss : 1.6328837871580215
for alpha = 1
Log Loss : 1.795735442175563
for alpha = 10
Log Loss : 1.8140065085040373
for alpha = 100
Log Loss : 1.8160949699094917
```




```
For values of best alpha =  0.0001 The train log loss is: 0.39631569296579755
```

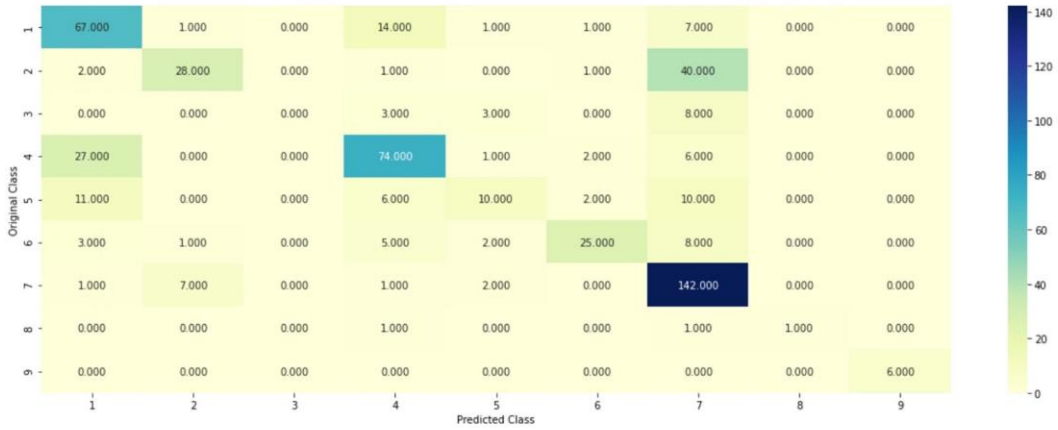
```
For values of best alpha =  0.0001 The cross validation log loss is: 0.9701157827989754
For values of best alpha =  0.0001 The test log loss is: 0.9745579343011249
```

4.3.2. Using the best Hyper parameter found

[45]:

```
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)

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

# Variables that will be used in the end to make comparison table of models
lr_balance_misclassified = (np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)- cv_y))/cv_y.shape[0])*100
```

```
Log loss : 0.9701157827989754
Number of mis-classified points : 0.33646616541353386
```



5. Results

[46]:

```
# Creating table using PrettyTable library
from prettytable import PrettyTable

names =['Naive Bayes','LR With Class Balancing']
train_loss = [nb_train,lr_balance_train]
cv_loss = [nb_cv,lr_balance_cv]
test_loss = [nb_test,lr_balance_test]
misclassified = [nb_misclassified,lr_balance_misclassified]
numbering = [1,2]

ptable = PrettyTable()
ptable.add_column("S.NO.",numbering)
ptable.add_column("MODEL",names)
ptable.add_column("Train_loss",train_loss)
ptable.add_column("CV_loss",cv_loss)
ptable.add_column("Test_loss",test_loss)
ptable.add_column("Misclassified(%)",misclassified)

print(ptable)
```

```
+-------+----------------------+--------------------+--------------------+--------------------+--------------------+
| S.NO. |        MODEL         |     Train_loss     |      CV_loss       |     Test_loss      |  Misclassified(%)  |
+-------+----------------------+--------------------+--------------------+--------------------+--------------------+
|   1   |      Naive Bayes     |  0.4399633030131071 | 1.1878970789307897 | 1.172029085937199  | 37.21804511278196  |
|   2   | LR With Class Balancing | 0.39631569296579755 | 0.9701157827989754 | 0.9745579343011249 | 33.64661654135339  |
+-------+----------------------+--------------------+--------------------+--------------------+--------------------+
```

7 REFERENCES

New Cancer Drug -
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/#488514a36b25

The Different Types of Mutation -
https://www.youtube.com/watch?v=qxXRKVompI8