Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/discussion/35336#198462

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

1. https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25 (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=qxXRKVompI8 (https://www.youtube.com/watch?v=qxXRKVompI8)

1.3. Real-world/Business objectives and constraints.

- No low-latency requirement.
- Interpretability is important.
- Errors can be very costly.

- Probability of a data-point belonging to each class is needed.

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

ID,Text
0||Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity revealed. Previous work has shown that CDK10 silencing increases ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The precise mechanisms by which CDK10 modulates ETS2 activity, and more generally the functions of CDK10, remain elusive. Here we

demonstrate that CDK10 is a cyclin-dependent kinase by identifying cyclin M as an activating cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR syndrome, a human developmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malformations. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and in conferring tamoxifen resistance to breast cancer cells. CDK10/cyclin M phosphorylates ETS2 in vitro, and in cells it positively controls ETS2 degradation by the proteasome. ETS2 protein levels are increased in cells derived from a STAR patient, and this increase is attributable to decreased cyclin M levels. Altogether, our results reveal an additional regulatory mechanism for ETS2, which plays key roles in cancer and development. They also shed light on the molecular mechanisms underlying STAR syndrome.Cyclin-dependent kinases (CDKs) play a pivotal role in the control of a number of fundamental cellular processes (1). The human genome contains 21 genes encoding proteins that can be considered as members of the CDK family owing to their sequence similarity with bona fide CDKs, those known to be activated by cyclins (2). Although discovered almost 20 y ago (3, 4), CDK10 remains one of the two CDKs without an identified cyclin partner. This knowledge gap has largely impeded the exploration of its biological functions. CDK10 can act as a positive cell cycle regulator in some cells (5, 6) or as a tumor suppressor in others (7, 8). CDK10 interacts with the ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity through an unknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK pathway, and induces resistance of MCF7 cells to tamoxifen (6). ...

2.2. Mapping the real-world problem to an ML problem

2.2.1. Type of Machine Learning Problem

```
        There are nine different classes a genetic mutation can be classified into => M
ulti class 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 Learing Objectives and Constraints

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

Constraints:

* Interpretability * Class probabilities are needed. * Penalize the errors in class probabilites => Metric is Log-loss. * No Latency constraints.

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

```
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

```
data = pd.read_csv('training/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']
```

	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

```
# note the seprator in this file
data_text =pd.read_csv("training/training_text",sep="\|\|",engine="python",names=["ID","TEXT"],skiprows=
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']
```

	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...

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

	ID	Gene	Variation	Class	TEXT
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...

```
result[result.isnull().any(axis=1)]
```

	ID	Gene	Variation	Class	TEXT
1109	1109	FANCA	S1088F	1	NaN
1277	1277	ARID5B	Truncating Mutations	1	NaN
1407	1407	FGFR3	K508M	6	NaN
1639	1639	FLT1	Amplification	6	NaN
2755	2755	BRAF	G596C	7	NaN

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

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

	ID	Gene	Variation	Class	TEXT
1109	1109	FANCA	S1088F	1	FANCA S1088F

Feature Engineering

Feature engineering reference - https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/ (https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/)

```
#Variation share
result['Variation_Share'] = result.apply(lambda r: sum([1 for w in r['Variation'].split(' ') if w in r[

#Text word count
result["Text_Word_Count"] = result["TEXT"].apply(lambda x: len(x.split()))

#Text Character count
result['Text_Character_Count'] = result['TEXT'].apply(lambda x: len(str(x)))

#Text average length
result['Text_Avg_length'] = result['Text_Character_Count'] / result['Text_Word_Count']

#Gene Character count
result['Gene_Character_Count'] = result['Gene'].apply(lambda x: len(str(x)))

#Variation_Character_Count
result['Variation_Character_Count'] = result['Variation'].apply(lambda x: len(str(x)))

#Var word count
result["Var_Word_Count"] = result["Variation"].apply(lambda x: len(x.split()))

#Var average length
result['var_Avg_length'] = result['Variation_Character_Count'] / result['Var_Word_Count']

#Stopword text count
from nltk.corpus import stopwords
stop = stopwords.words('english')

result['stopwordst'] = result['TEXT'].apply(lambda x: len([x for x in x.split() if x in stop]))

#Speacial character count
result['hastags'] = result['TEXT'].apply(lambda x: len([x for x in x.split() if x.startswith('#')]))

#Numerics count
```

```
result['numerics'] = result['TEXT'].apply(lambda x: len([x for x in x.split() if x.isdigit()]))

#Uppercase word count
result['upper'] = result['TEXT'].apply(lambda x: len([x for x in x.split() if x.isupper()]))

# label encoder
from sklearn import preprocessing
for c in result.columns:
    if result[c].dtype == 'object':
        if c == 'Gene':
            lbl = preprocessing.LabelEncoder()
            result[c+'_lbl_enc'] = lbl.fit_transform(result[c].values)
        elif c == 'Variation':
            lbl = preprocessing.LabelEncoder()
            result[c+'_lbl_enc'] = lbl.fit_transform(result[c].values)
```

```
result.head(2)
```

	ID	Gene	Variation	Class	TEXT	Variation_Share	Text_Word_Count	Text_Character_Count	Text_A
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	1	6089	39765	6.53062
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...	1	5722	36831	6.43673

3.1.3. Preprocessing of text

```
# 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
```

```
#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")
```

```
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
Time took for preprocessing the text : 121.0215206174142 seconds
```

```
#Creating dummies for Gene_lbl_enc & Variation_lbl_enc
result = pd.concat([result, pd.get_dummies(result['Gene_lbl_enc']).\
                        rename(columns=lambda x: 'Gene_lbl_enc_' + str(x))], axis=1)

result = pd.concat([result, pd.get_dummies(result['Variation_lbl_enc']).\
                        rename(columns=lambda x: 'Variation_lbl_enc_' + str(x))], axis=1)
```

```
result.head(2)
```

	ID	Gene	Variation	Class	TEXT	Variation_Share	Text_Word_Count	Text_Character_Count	Text_A
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	1	6089	39765	6.53062
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...	1	5722	36831	6.43673

2 rows × 3279 columns

Checking feature importance of only newly created variables using Random Forest

```
temp_y=result[['Class']]
temp_train=result.drop(['ID','Gene','Variation','Class','TEXT','Gene_lbl_enc','Variation_lbl_enc'],axis=
```

```
## Import the random forest model.
from sklearn.ensemble import RandomForestClassifier
## This line instantiates the model.
rf = RandomForestClassifier()
## Fit the model on your training data.
rf.fit(temp_train, temp_y)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

```
import pandas as pd
feature_importances = pd.DataFrame(rf.feature_importances_,
                                   index = temp_train.columns,
                                    columns=['importance']).sort_values('importance',ascending=False)
```

```
#Feature importance of newly created variables
feature_importances[:15]
```

	importance
stopwordst	0.053990
numerics	0.053875
Text_Word_Count	0.052921
Text_Avg_length	0.051516
upper	0.051413
Text_Character_Count	0.049281
var_Avg_length	0.030022
Variation_Character_Count	0.028296
Gene_Character_Count	0.023299
Variation_Share	0.018827
Gene_lbl_enc_31	0.015835
hastags	0.015086
Gene_lbl_enc_252	0.013824
Gene_lbl_enc_196	0.013398
Variation_lbl_enc_2629	0.012533

3.1.4. Test, Train and Cross Validation Split

3.1.4.1. Splitting data into train, test and cross validation (64:20:16)

```
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' [stra
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 varai
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

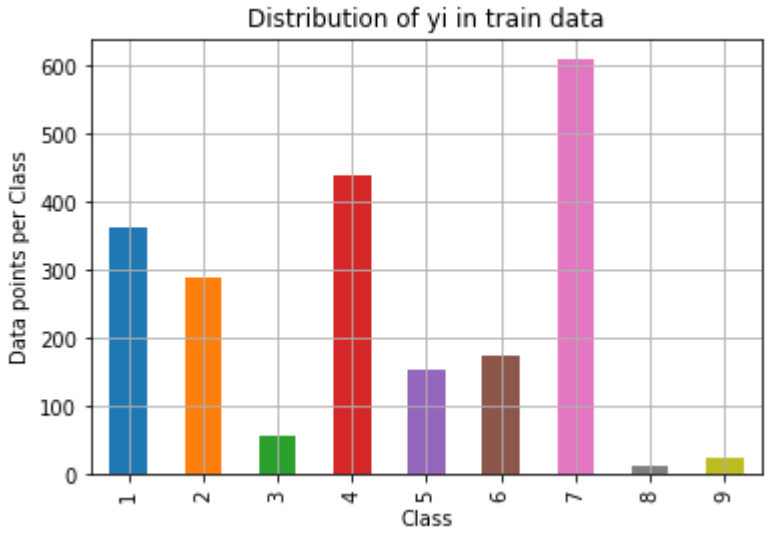
```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

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

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

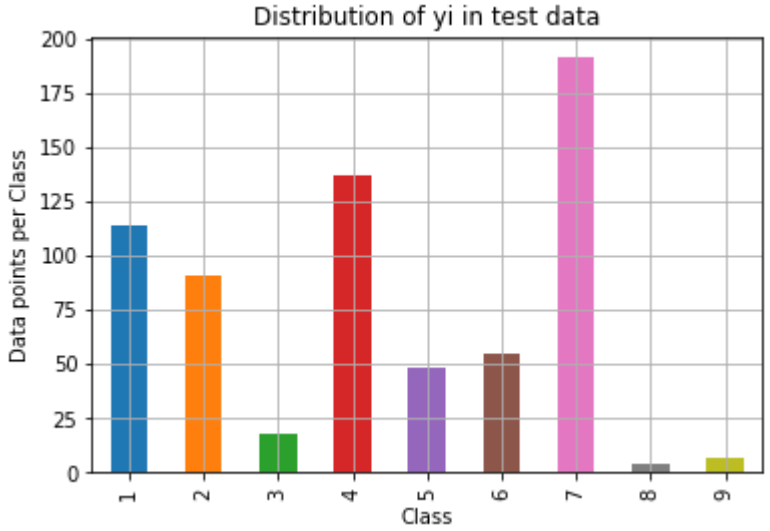
```
# it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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(

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((
```

```
35
36 print('-'*80)
37 my_colors = 'rgbkymc'
38 cv_class_distribution.plot(kind='bar')
39 plt.xlabel('Class')
40 plt.ylabel('Data points per Class')
41 plt.title('Distribution of yi in cross validation data')
42 plt.grid()
43 plt.show()
44
45 # ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
46 # -(train_class_distribution.values): the minus sign will give us in decreasing order
47 sorted_yi = np.argsort(-train_class_distribution.values)
48 for i in sorted_yi:
49     print('Number of data points in class', i+1, ':',cv_class_distribution.values[i], '(', np.round((cv_
50
```

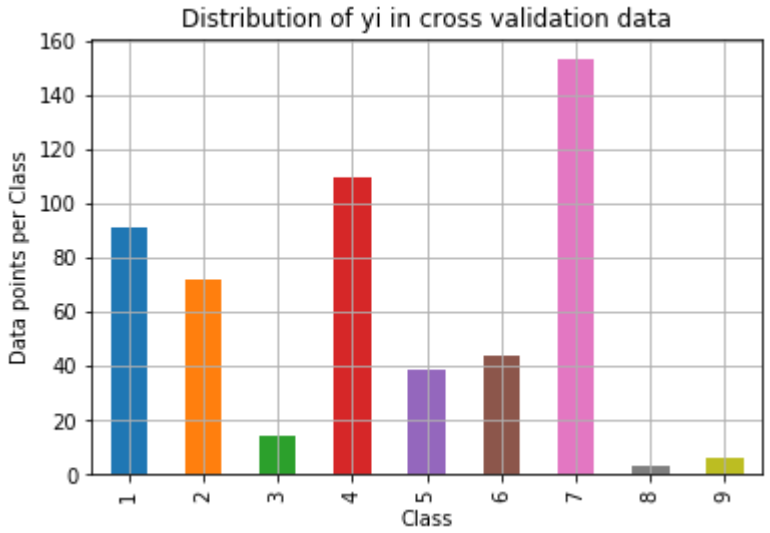



```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
```

```
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298 %)
Number of data points in class 3 : 57 ( 2.684 %)
Number of data points in class 9 : 24 ( 1.13 %)
Number of data points in class 8 : 12 ( 0.565 %)
------------------------------------------------------------------------------
```




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




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

3.2 Prediction using a 'Random' Model

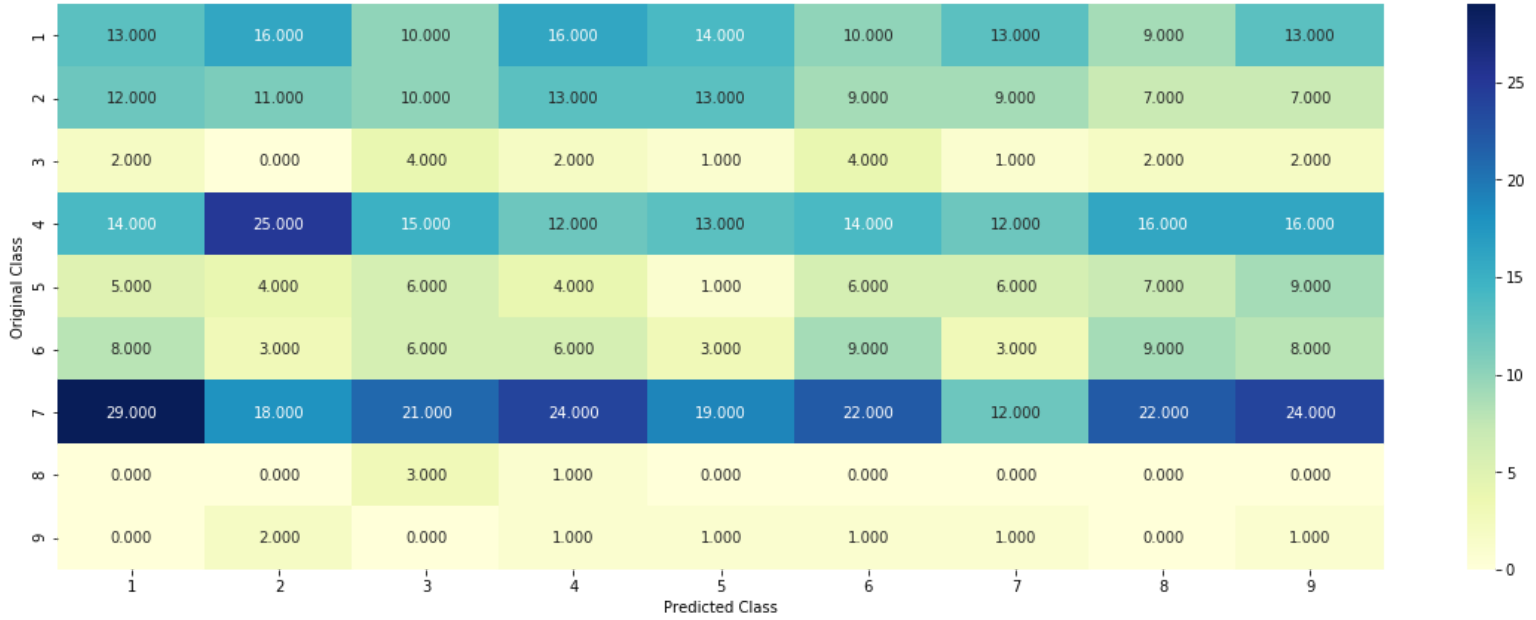
In a 'Random' Model, we generate the NINE class probabilites randomly such that they sum to 1.

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

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

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

    labels = [1,2,3,4,5,6,7,8,9]
    # representing A in heatmap format
    print("-"*20, "Confusion matrix", "-"*20)
    plt.figure(figsize=(20,7))
```

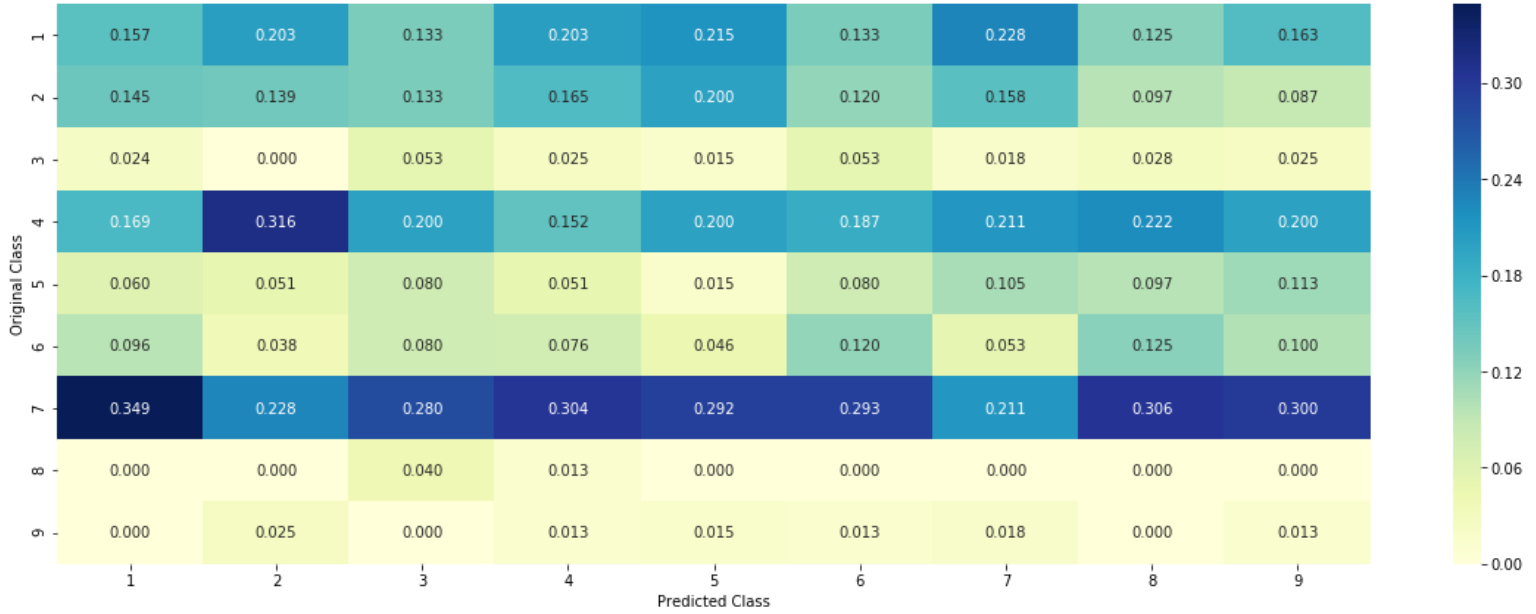
```
    sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

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

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

```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

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

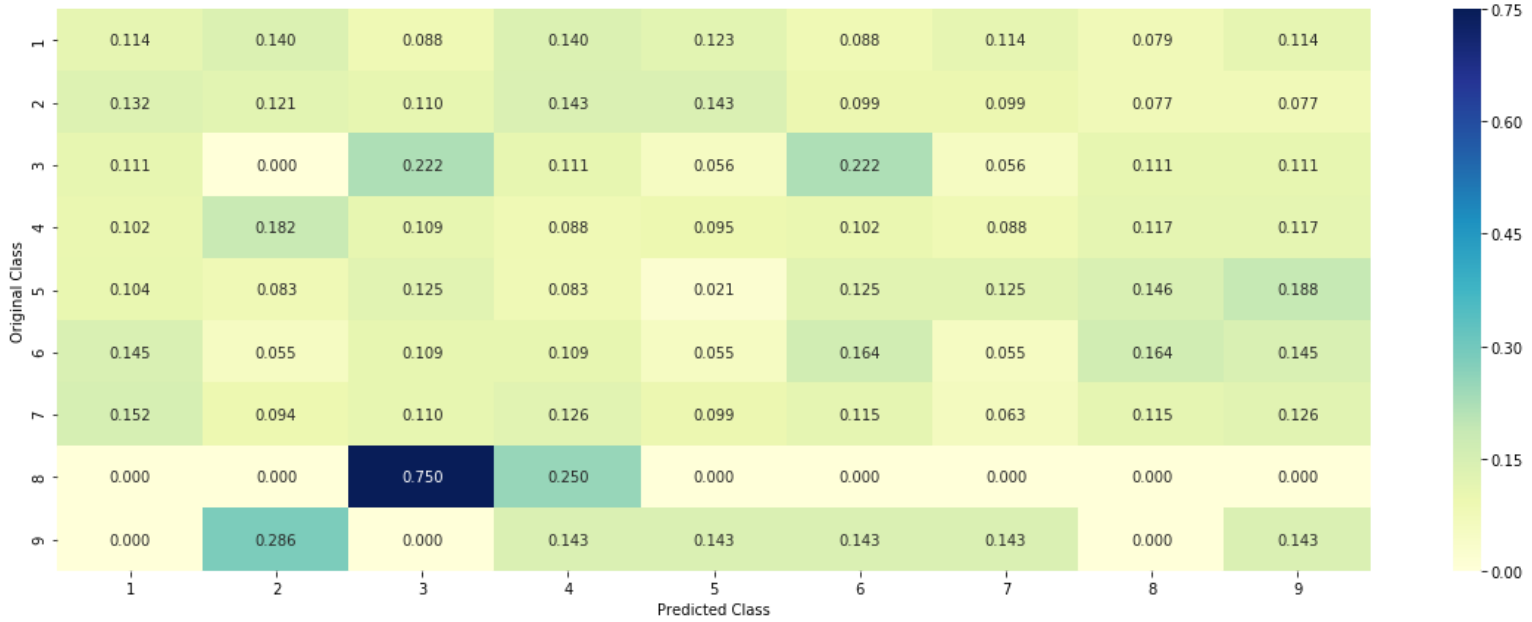
```
Log loss on Cross Validation Data using Random Model 2.503089399222508
Log loss on Test Data using Random Model 2.4987970579719736
-------------------- Confusion matrix --------------------
```



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



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



3.3 Univariate Analysis

```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feature in train data dataframe
# build a vector (1*9) , the first element = (number of times it occured in class1 + 10*alpha / number o
# 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 varaition Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    # value_count: it contains a dict like
    # print(train_df['Gene'].value_counts())
    # output:
    #        {BRCA1      174
    #         TP53       106
    #         EGFR        86
    #         BRCA2       75
    #         PTEN        69
    #         KIT         61
    #         BRAF        60
    #         ERBB2       47
    #         PDGFRA      46
    #         ...}
    # print(train_df['Variation'].value_counts())
    # output:
```

```
    # {
    # Truncating_Mutations                    63
    # Deletion                                43
    # Amplification                           43
    # Fusions                                 22
    # Overexpression                           3
    # E17K                                     3
    # Q61L                                     3
    # S222D                                    2
    # P130S                                    2
    # ...
    # }
    value_count = train_df[feature].value_counts()

    # gv_dict : Gene Variation Dict, which contains the probability array for each gene/variation
    gv_dict = dict()

    # denominator will contain the number of time that particular feature occured in whole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to perticular class
        # vec is 9 diamensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=='BRCA1')])
            #         ID   Gene             Variation  Class
            # 2470  2470  BRCA1               S1715C      1
            # 2486  2486  BRCA1               S1841R      1
            # 2614  2614  BRCA1                  M1R      1
            # 2432  2432  BRCA1               L1657P      1
            # 2567  2567  BRCA1               T1685A      1
            # 2583  2583  BRCA1               E1660G      1
            # 2634  2634  BRCA1               W1718L      1
            # cls_cnt.shape[0] will return the number of rows

            cls_cnt = train_df.loc[(train_df['Class']==k) & (train_df[feature]==i)]
```

```
70
71              # cls_cnt.shape[0](numerator) will contain the number of time that particular feature occur
72              vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))
73
74          # we are adding the gene/variation to the dict as key and vec as value
75          gv_dict[i]=vec
76      return gv_dict
77
78  # Get Gene variation feature
79  def get_gv_feature(alpha, feature, df):
80      # print(gv_dict)
81      #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177, 0.1363636363636363
82      #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.2704081632653061
83      #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177, 0.06818181
84      #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.078787878787878
85      #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.465408805031446
86      #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.07284768211920529
87      #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.0733333333333333
88      #      ...
89      #     }
90      gv_dict = get_gv_fea_dict(alpha, feature, df)
91      # value_count is similar in get_gv_fea_dict
92      value_count = train_df[feature].value_counts()
93
94      # gv_fea: Gene_variation feature, it will contain the feature for each feature value in the data
95      gv_fea = []
96      # for every feature values in the given data frame we will check if it is there in the train data th
97      # 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
98      for index, row in df.iterrows():
99          if row[feature] in dict(value_count).keys():
100             gv_fea.append(gv_dict[row[feature]])
101         else:
102             gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
103 #             gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
104     return gv_fea
```

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

- (numerator + 10*alpha) / (denominator + 90*alpha)

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

Q2. How many categories are there and How they are distributed?

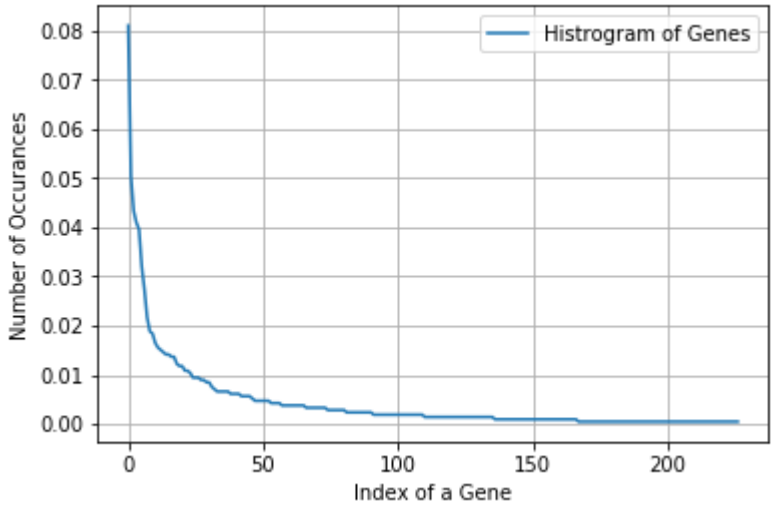
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 227
BRCA1     172
TP53      107
EGFR       92
BRCA2      87
PTEN       84
KIT        68
BRAF       58
ERBB2      46
PDGFRA     40
ALK        39
Name: Gene, dtype: int64
```

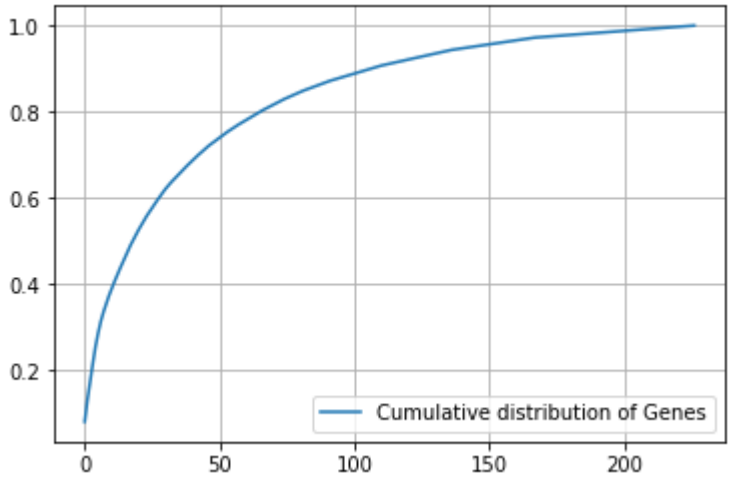
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the train data, and th
```

```
Ans: There are 227 different categories of genes in the train data, and they are distibuted as follows
```

```
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histrogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()

```



```
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 appropriate featurization based on the ML model we use. For this problem of multi-class classification with categorical features, one-hot encoding is better for Logistic regression while response coding is better for Random Forests.

```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
```

```
print("train_gene_feature_responseCoding is converted feature using respone coding method. The shape of
```

```
train_gene_feature_responseCoding is converted feature using respone coding method. The shape of gene feature: (2124, 9)
```

```
# 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'])
```

```
train_df['Gene'].head()
```

```
2065      SOX9
2327      JAK2
953     PDGFRB
1028      TSC2
1794        AR
Name: Gene, dtype: object
```

```
1 gene_vectorizer.get_feature_names()
```

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid2',
 'arid5b',
 'asxl1',
 'asxl2',
 'atm',
 'atrx',
 'aurka',
 'aurkb',
 'axl',
 'b2m',
```

```
1 print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature: (2124, 226)
```

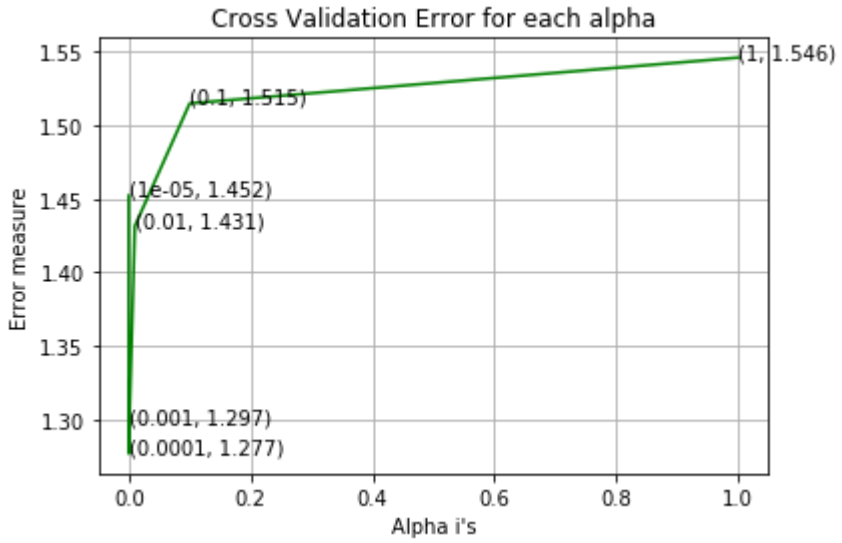
Q4. How good is this gene feature in predicting y_i?

There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a proper ML model using just this feature. In this case, we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

```
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_mo
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=N
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
# 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_

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, pred
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_
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, predic

```

```
For values of alpha =  1e-05 The log loss is: 1.4522395212582269
For values of alpha =  0.0001 The log loss is: 1.2765937139984063
For values of alpha =  0.001 The log loss is: 1.2973227647270822
For values of alpha =  0.01 The log loss is: 1.4314445419857231
For values of alpha =  0.1 The log loss is: 1.5149362420219377
For values of alpha =  1 The log loss is: 1.5461036406063464
```




```
For values of best alpha =  0.0001 The train log loss is: 1.0441095122666129
For values of best alpha =  0.0001 The cross validation log loss is: 1.2765937139984063
For values of best alpha =  0.0001 The test log loss is: 1.223038923188658
```

Q5. Is the Gene feature stable across all the data sets (Test, Train, Cross validation)?

Ans. Yes, it is. Otherwise, the CV and Test errors would be significantly more than train error.

```
1 print("Q6. How many data points in Test and CV datasets are covered by the ", unique_genes.shape[0], " g
2
3 test_coverage=test_df[test_df['Gene'].isin(list(set(train_df['Gene'])))].shape[0]
4 cv_coverage=cv_df[cv_df['Gene'].isin(list(set(train_df['Gene'])))].shape[0]
5
6 print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/test_df.shape
7 print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.shape
```

```
Q6. How many data points in Test and CV datasets are covered by the  227  genes in train dataset?
Ans
1. In test data 638 out of 665 : 95.93984962406014
2. In cross validation data 513 out of  532 : 96.42857142857143
```

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?

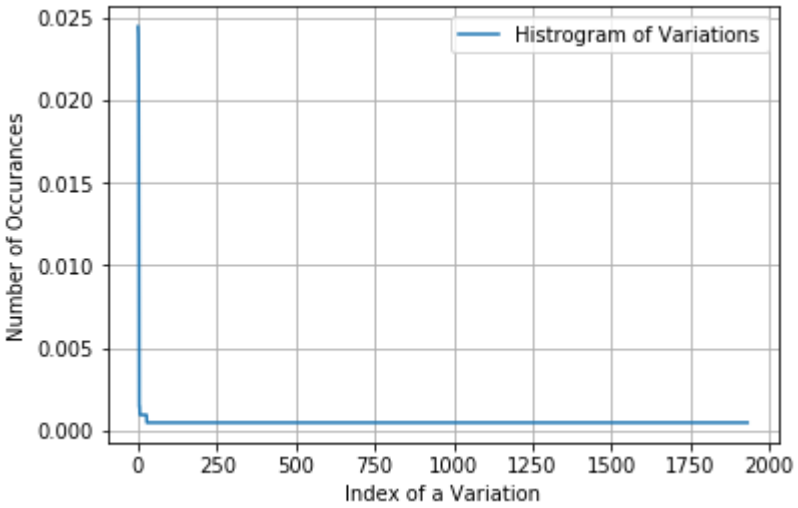
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1932
Deletion               52
Truncating_Mutations   51
Amplification          45
Fusions                22
Overexpression          3
Q61L                    3
C618R                   2
TMPRSS2-ETV1_Fusion     2
G67R                    2
G12C                    2
Name: Variation, dtype: int64
```

```
print("Ans: There are", unique_variations.shape[0] ,"different categories of variations in the train da
```

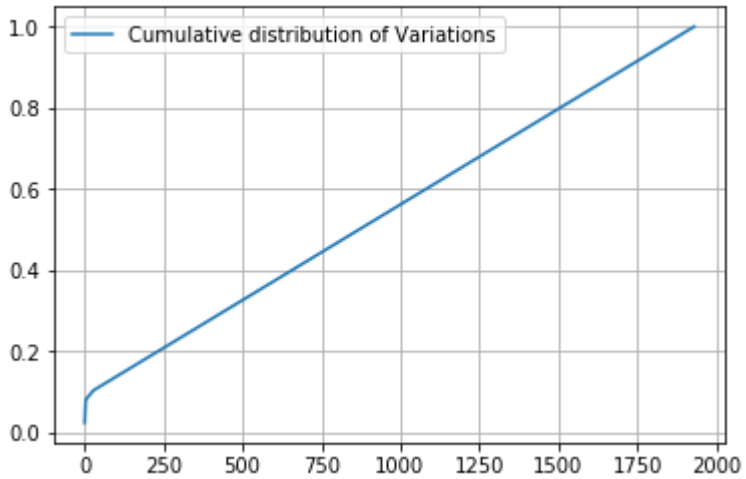
```
Ans: There are 1932 different categories of variations in the train data, and they are distibuted as follows
```

```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



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

```
[0.02448211 0.04849341 0.06967985 ... 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 both these methods to featurize the Variation Feature

```
# 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))
```

```
print("train_variation_feature_responseCoding is a converted feature using the response coding method. 
```

```
train_variation_feature_responseCoding is a converted feature using the response coding method. The shape of Variation feat
ure: (2124, 9)
```

```
# 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'])
```

```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. Tl
```

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Variation featu
re: (2124, 1959)
```

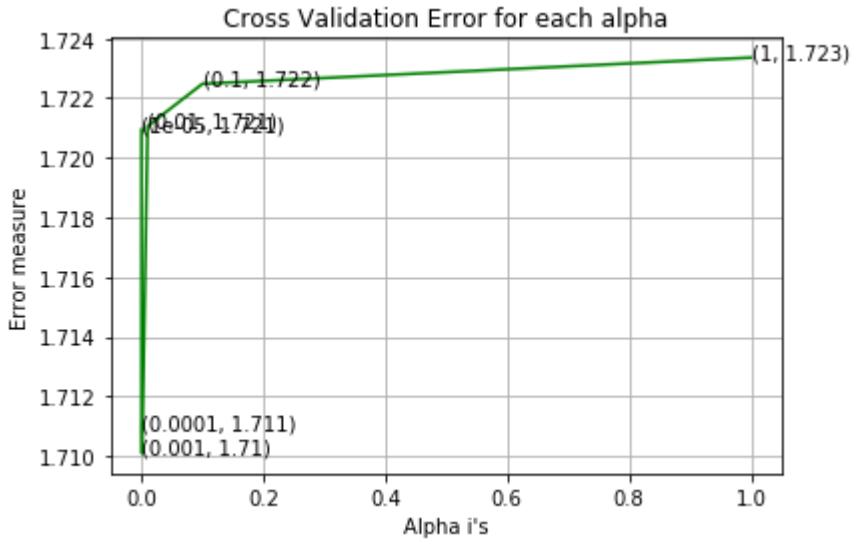
Q10. How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_mo
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=N
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
# 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_

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, pred
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_
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, predic

```

```
For values of alpha =  1e-05 The log loss is: 1.720943581527042
For values of alpha =  0.0001 The log loss is: 1.7109135510927833
For values of alpha =  0.001 The log loss is: 1.7100790201478915
For values of alpha =  0.01 The log loss is: 1.7210571125209235
For values of alpha =  0.1 The log loss is: 1.7224794460396793
For values of alpha =  1 The log loss is: 1.7233578731600068
```




```
For values of best alpha =  0.001 The train log loss is: 1.042227051115231
For values of best alpha =  0.001 The cross validation log loss is: 1.7100790201478915
For values of best alpha =  0.001 The test log loss is: 1.7012039305187143
```

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.

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

```
Q12. How many data points are covered by total  1932  genes in test and cross validation data sets?
Ans
1. In test data 74 out of 665 : 11.12781954887218
2. In cross validation data 54 out of  532 : 10.150375939849624
```

3.2.3 Univariate Analysis on Text Feature

1. How many unique words are present in train data?

2. How are word frequencies distributed?
3. How to featurize text field?
4. Is the text feature useful in predicitng y_i?
5. Is the text feature stable across train, test and CV datasets?

```
# 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
```

```
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
```

```
# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = CountVectorizer()
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) v
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 : 134512
```

```
# Collecting all the genes and variations in a single list
corpus = []
for word in data['Gene'].values:
    corpus.append(word)
for word in data['Variation'].values:
    corpus.append(word)
```

```
# Training TfidfVectorizer on the 'corpus' list
text1 = TfidfVectorizer()
text2 = text1.fit_transform(corpus)
text1_features = text1.get_feature_names()

# Transforming the train_df['TEXT']
train_text = text1.transform(train_df['TEXT'])

# Transforming the test_df['TEXT']
test_text = text1.transform(test_df['TEXT'])

# Transforming the cv_df['TEXT']
cv_text = text1.transform(cv_df['TEXT'])

# Normalizing the train_text
train_text = normalize(train_text,axis=0)

# Normalizing the test_text
test_text = normalize(test_text,axis=0)

# Normalizing the cv_text
cv_text = normalize(cv_text,axis=0)
```

```
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = train_df[train_df['Class']==i]
    # build a word dict based on the words in that class
    dict_list.append(extract_dictionary_paddle(cls_text))
    # append it to dict_list

# dict_list[i] is build on i'th  class text data
# total_dict is buid on whole training text data
total_dict = extract_dictionary_paddle(train_df)


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)
```

```
#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)
```

```
# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.T/train_text_feature_responseCod
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_feature_responseCoding
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_responseCoding.sum(a
```

```
# 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)
```

```
#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()))
```

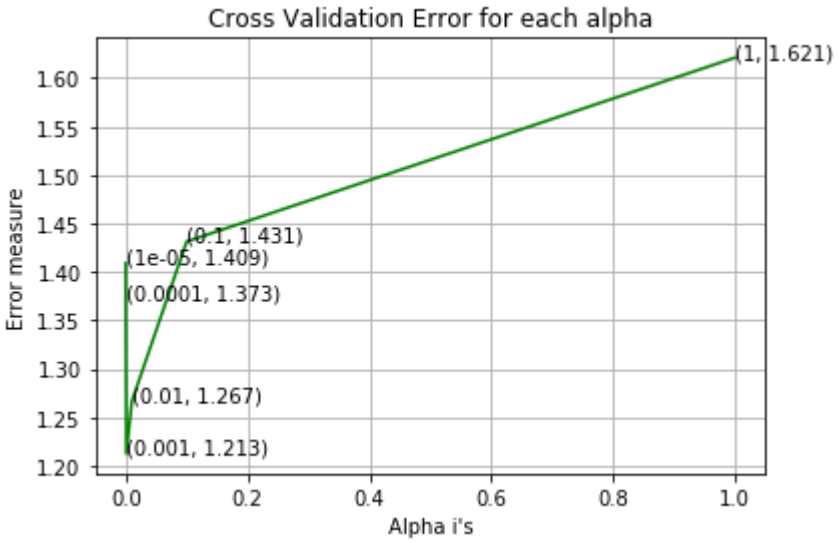
```
# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({1: 46690, 2: 19055, 3: 8600, 4: 7286, 5: 4351, 6: 4332, 8: 2740, 7: 2498, 9: 2351, 10: 2004, 12: 1537, 11: 1486,
15: 1095, 14: 1095, 16: 1040, 13: 928, 18: 892, 17: 738, 20: 706, 19: 623, 21: 557, 22: 540, 48: 536, 24: 526, 30: 483, 2
3: 474, 25: 454, 26: 435, 32: 419, 28: 402, 27: 367, 33: 319, 36: 295, 51: 292, 29: 288, 34: 287, 31: 285, 40: 265, 35: 24
3, 38: 231, 42: 226, 37: 223, 44: 208, 45: 201, 39: 201, 50: 195, 41: 194, 54: 189, 49: 184, 52: 182, 46: 176, 55: 167, 5
7: 161, 43: 161, 64: 151, 60: 147, 56: 143, 53: 141, 47: 139, 58: 138, 63: 125, 62: 120, 72: 115, 65: 114, 61: 114, 59: 11
3, 66: 112, 69: 110, 68: 109, 70: 104, 96: 103, 80: 96, 67: 95, 71: 91, 78: 89, 83: 86, 73: 84, 84: 82, 75: 81, 90: 80, 10
2: 77, 74: 77, 86: 75, 76: 74, 79: 72, 100: 70, 77: 70, 98: 69, 87: 68, 89: 67, 91: 66, 85: 66, 88: 65, 112: 64, 93: 64, 8
1: 64, 99: 61, 92: 60, 144: 59, 103: 58, 82: 58, 108: 56, 113: 53, 105: 53, 94: 53, 107: 50, 114: 49, 128: 48, 109: 48, 9
5: 48, 120: 47, 101: 47, 106: 45, 147: 44, 126: 44, 119: 44, 104: 44, 118: 43, 110: 43, 131: 42, 97: 42, 152: 41, 136: 41,
135: 40, 134: 40, 130: 40, 125: 40, 115: 40, 140: 39, 132: 39, 111: 39, 133: 38, 148: 37, 146: 37, 139: 37, 123: 37, 121:
37, 164: 36, 143: 36, 116: 36, 155: 35, 141: 35, 127: 35, 117: 35, 149: 34, 153: 33, 138: 33, 185: 32, 184: 32, 122: 32, 1
51: 30, 145: 30, 129: 30, 181: 29, 174: 29, 150: 29, 124: 29, 179: 28, 170: 28, 216: 27, 210: 27, 187: 27, 183: 27, 175: 2
7, 162: 27, 171: 26, 160: 26, 156: 26, 142: 26, 223: 25, 206: 25, 200: 25, 182: 25, 177: 25, 167: 25, 178: 24, 165: 24, 15
4: 24, 228: 23, 195: 23, 186: 23, 180: 23, 176: 23, 245: 22, 235: 22, 205: 22, 203: 22, 168: 22, 157: 22, 240: 21, 202: 2
1, 201: 21, 199: 21, 198: 21, 172: 21, 166: 21, 161: 21, 158: 21, 288: 20, 268: 20, 254: 20, 233: 20, 222: 20, 215: 20, 21
1: 20, 192: 20, 191: 20, 188: 20, 274: 19, 250: 19, 249: 19, 217: 19, 214: 19, 212: 19, 209: 19, 207: 19, 159: 19, 277: 1
8, 260: 18, 258: 18, 255: 18, 234: 18, 221: 18, 220: 18, 194: 18, 190: 18, 163: 18, 395: 17, 264: 17, 262: 17, 241: 17, 23
9: 17, 208: 17, 196: 17, 169: 17, 137: 17, 285: 16, 282: 16, 237: 16, 231: 16, 224: 16, 193: 16, 400: 15, 267: 15, 266: 1
5, 252: 15, 246: 15, 244: 15, 242: 15, 230: 15, 225: 15, 219: 15, 189: 15, 173: 15, 371: 14, 298: 14, 294: 14, 291: 14, 27
6: 14, 269: 14, 248: 14, 236: 14, 227: 14, 226: 14, 204: 14, 394: 13, 375: 13, 342: 13, 328: 13, 317: 13, 314: 13, 305: 1
3, 275: 13, 259: 13, 256: 13, 232: 13, 354: 12, 345: 12, 340: 12, 325: 12, 324: 12, 311: 12, 303: 12, 299: 12, 281: 12, 27
```

```
# Train a Logistic regression+Calibration model using text features whicha re 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_mo
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=N
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
# 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_

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, pred
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_
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, predic

```

```
For values of alpha =  1e-05 The log loss is: 1.409032667418594
For values of alpha =  0.0001 The log loss is: 1.373092662164093
For values of alpha =  0.001 The log loss is: 1.212596575674792
For values of alpha =  0.01 The log loss is: 1.2665052982158738
For values of alpha =  0.1 The log loss is: 1.4314377504358973
For values of alpha =  1 The log loss is: 1.6211856455279456
```




```
For values of best alpha =  0.001 The train log loss is: 0.7909988562287417
For values of best alpha =  0.001 The cross validation log loss is: 1.212596575674792
For values of best alpha =  0.001 The test log loss is: 1.226921695988835
```

Q. Is the Text feature stable across all the data sets (Test, Train, Cross validation)?

Ans. Yes, it seems like!

```
def get_intersec_text(df):
    df_text_vec = CountVectorizer()
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

    df_text_fea_counts = df_text_fea.sum(axis=0).A1
    df_text_fea_dict = dict(zip(list(df_text_features),df_text_fea_counts))
    len1 = len(set(df_text_features))
    len2 = len(set(train_text_features) & set(df_text_features))
    return len1,len2
```

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

```
79.069 % of word of test data appeared in train data
79.153 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

```
#Data preparation for ML models.

#Misc. functionns for ML models


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

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

```
# 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()

    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")
```

Keeping top 15 created variables based on feature importance of Random Forest

```
#Keeping separate dataset for newly created feaures
train_df_other_feat=train_df[["stopwordst", "numerics", "Text_Word_Count",  "Text_Avg_length",  "upper",
test_df_other_feat=test_df[["stopwordst",   "numerics", "Text_Word_Count",  "Text_Avg_length",  "upper",
cv_df_other_feat=cv_df[["stopwordst",   "numerics", "Text_Word_Count",  "Text_Avg_length",  "upper",
```

```
train_df_other_feat.head(5)
```

	stopwordst	numerics	Text_Word_Count	Text_Avg_length	upper	Text_Character_Count	var_Avg_leng
2065	2609	84	7461	6.129875	812	45735	4.0
2327	6524	383	22515	6.917877	1968	155756	7.5
953	1468	73	4217	6.782073	191	28600	5.0
1028	1612	79	5158	6.701047	577	34564	6.0
1794	7504	510	25294	6.767573	1711	171179	13.0

```
from sklearn.preprocessing import MinMaxScaler
sc=MinMaxScaler()

train_df_other_feat=sc.fit_transform(train_df_other_feat)
test_df_other_feat=sc.transform(test_df_other_feat)
cv_df_other_feat=sc.transform(cv_df_other_feat)
```

Stacking the three types of features

```
# 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_onehotCod
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']))
```

```
train_x_onehotCoding=hstack((train_x_onehotCoding,train_df_other_feat)).tocsr()
test_x_onehotCoding = hstack((test_x_onehotCoding, test_df_other_feat)).tocsr()
cv_x_onehotCoding = hstack((cv_x_onehotCoding, cv_df_other_feat)).tocsr()
```

```
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.shap
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 136712)
(number of data points * number of features) in test data =  (665, 136712)
(number of data points * number of features) in cross validation data = (532, 136712)
```

4.3. Logistic Regression

4.3.1. With Class balancing

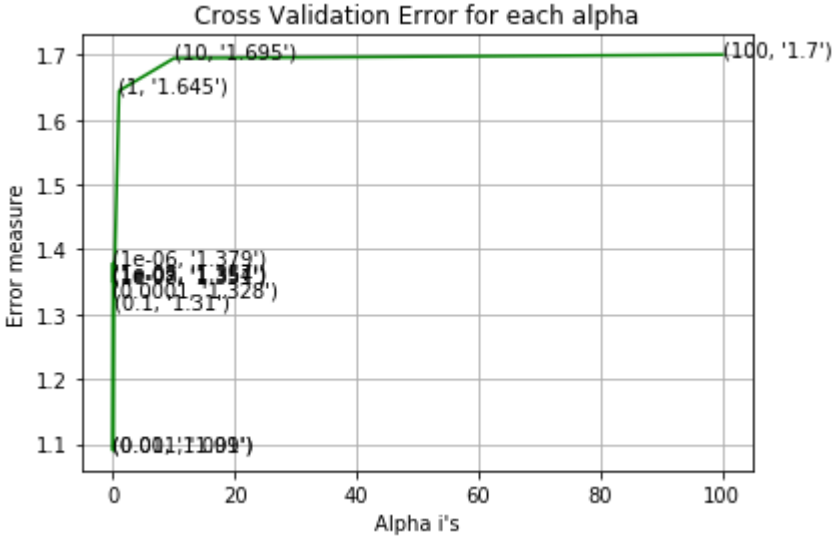
4.3.1.1. Hyper paramter tuning

```

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_mo
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=N
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
# 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-intuiti
#-------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklea
# ------------------------------
# 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(-8, 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_
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, pred
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_
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, predic
```

```
for alpha = 1e-08
```

```
Log Loss : 1.3508842922170041
for alpha = 1e-07
Log Loss : 1.35447047561646
for alpha = 1e-06
Log Loss : 1.3790610180468985
for alpha = 1e-05
Log Loss : 1.3565433625500933
for alpha = 0.0001
Log Loss : 1.3281267828415615
for alpha = 0.001
Log Loss : 1.0897809375901875
for alpha = 0.01
Log Loss : 1.090672560148139
for alpha = 0.1
Log Loss : 1.309603200733344
for alpha = 1
Log Loss : 1.6446133724366971
for alpha = 10
Log Loss : 1.6947507654550469
for alpha = 100
Log Loss : 1.7004145691906267
```

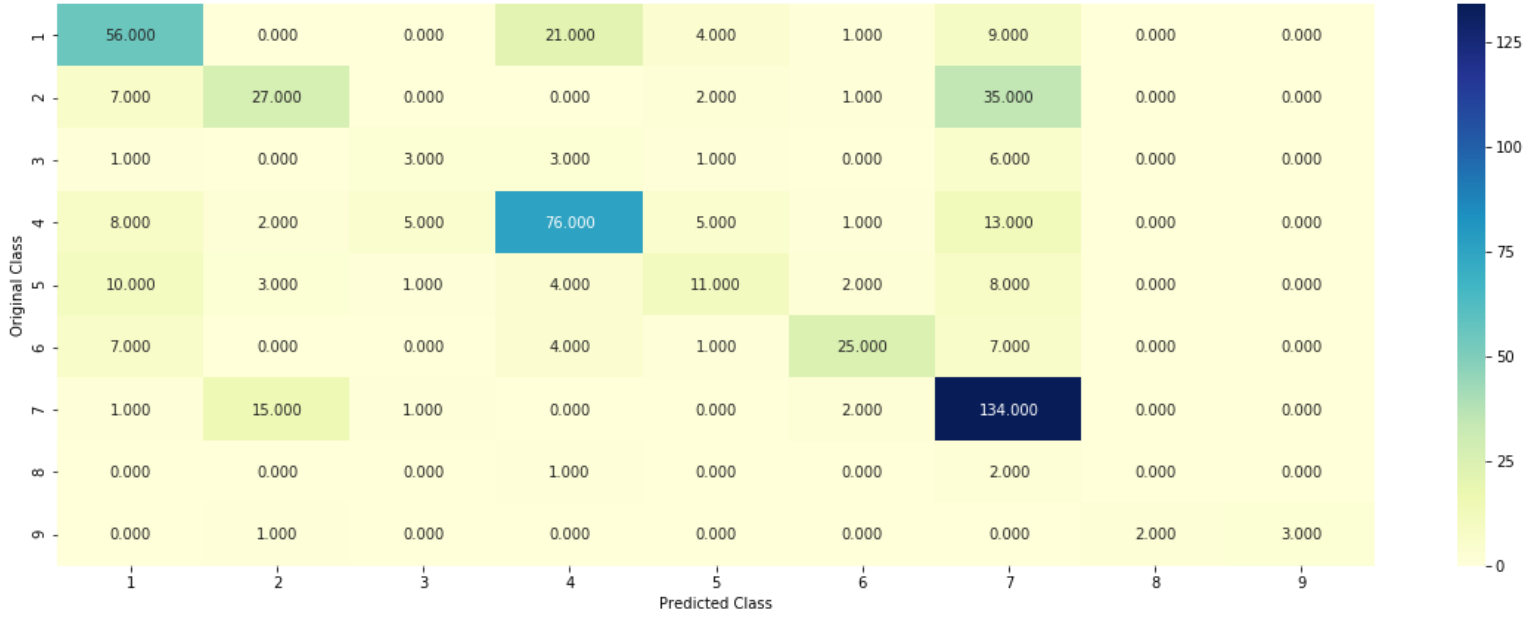



```
For values of best alpha =  0.001 The train log loss is: 0.6604857780958183
For values of best alpha =  0.001 The cross validation log loss is: 1.0897809375901875
For values of best alpha =  0.001 The test log loss is: 1.1094426053084374
```

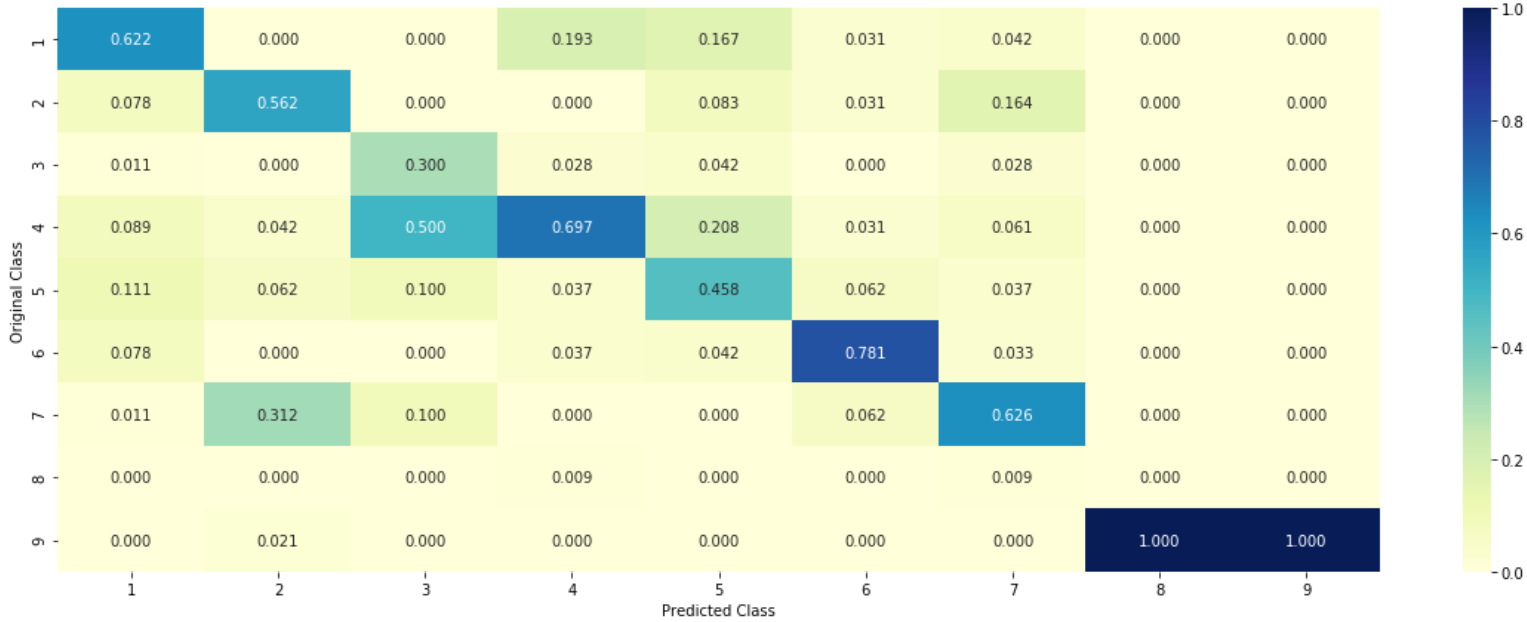
4.3.1.2. Testing the model with best hyper paramters

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_mo
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=N
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
# 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-intuiti
#-------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

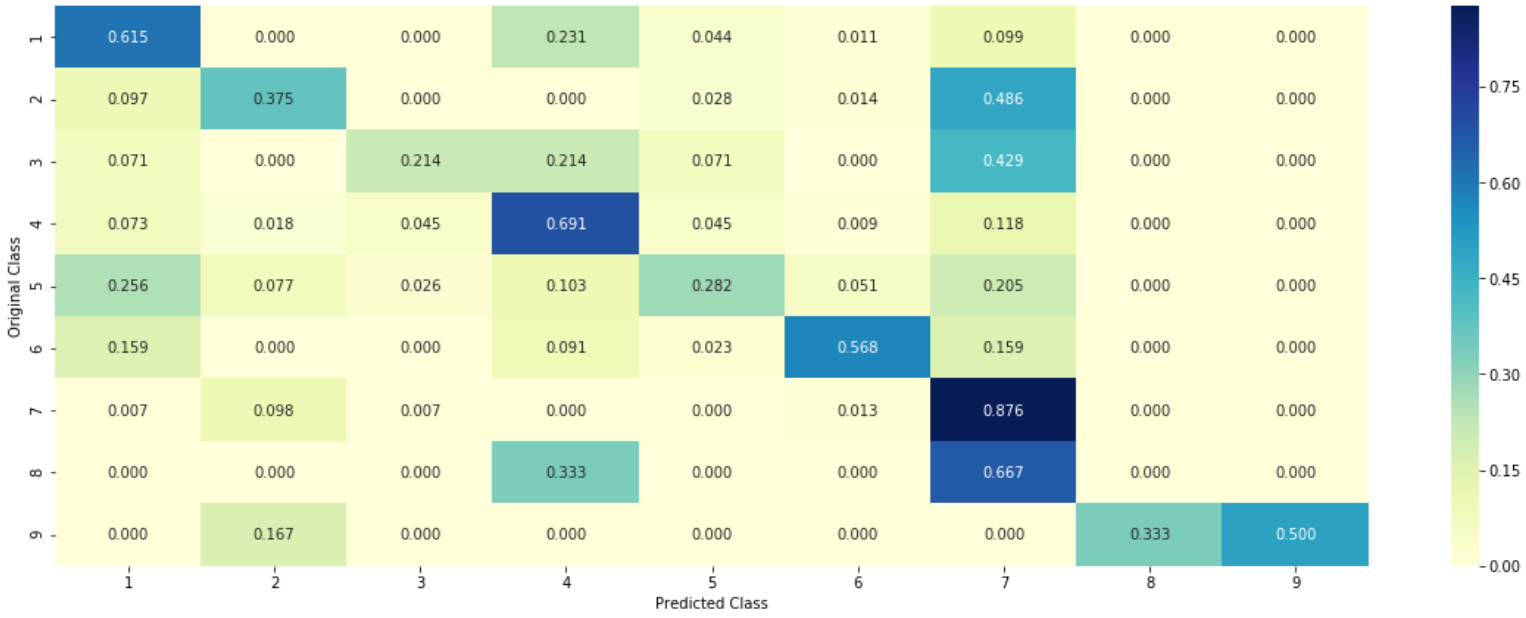
```
Log loss : 1.0897809375901875
Number of mis-classified points : 0.37030075187969924
-------------------- Confusion matrix --------------------
```



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



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



4.3.1.3. Feature Importance

```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i], yes_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0]," class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or Not']))
```

4.3.1.3.1. Correctly Classified point

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_i
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_
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0045 0.1905 0.0012 0.0012 0.0047 0.0014 0.7872 0.0076 0.0017]]
Actual Class : 7
--------------------------------------------------
23 Text feature [constitutively] present in test data point [True]
39 Text feature [flt1] present in test data point [True]
79 Text feature [oncogene] present in test data point [True]
80 Text feature [oncogenes] present in test data point [True]
84 Text feature [cysteine] present in test data point [True]
89 Text feature [inhibited] present in test data point [True]
137 Text feature [technology] present in test data point [True]
160 Text feature [dramatic] present in test data point [True]
162 Text feature [gaiix] present in test data point [True]
166 Text feature [ligand] present in test data point [True]
177 Text feature [downstream] present in test data point [True]
181 Text feature [concentrations] present in test data point [True]
182 Text feature [thyroid] present in test data point [True]
187 Text feature [expressing] present in test data point [True]
217 Text feature [activating] present in test data point [True]
241 Text feature [cdnas] present in test data point [True]
250 Text feature [manageable] present in test data point [True]
265 Text feature [axilla] present in test data point [True]
302 Text feature [inhibitor] present in test data point [True]
311 Text feature [cot] present in test data point [True]
313 Text feature [viability] present in test data point [True]
334 Text feature [activation] present in test data point [True]
```

```
352 Text feature [forced] present in test data point [True]
368 Text feature [subcutaneous] present in test data point [True]
371 Text feature [melanocyte] present in test data point [True]
376 Text feature [erk1] present in test data point [True]
388 Text feature [hours] present in test data point [True]
446 Text feature [procure] present in test data point [True]
448 Text feature [doses] present in test data point [True]
480 Text feature [mapk] present in test data point [True]
Out of the top  500  features  30 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_i
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_
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0482 0.2032 0.0108 0.0446 0.071  0.0164 0.5932 0.0078 0.0046]]
Actual Class : 7
--------------------------------------------------
23 Text feature [constitutively] present in test data point [True]
29 Text feature [constitutive] present in test data point [True]
47 Text feature [activated] present in test data point [True]
79 Text feature [oncogene] present in test data point [True]
89 Text feature [inhibited] present in test data point [True]
93 Text feature [transforming] present in test data point [True]
108 Text feature [transform] present in test data point [True]
148 Text feature [receptors] present in test data point [True]
177 Text feature [downstream] present in test data point [True]
210 Text feature [isozyme] present in test data point [True]
217 Text feature [activating] present in test data point [True]
232 Text feature [exchange] present in test data point [True]
326 Text feature [murine] present in test data point [True]
333 Text feature [agar] present in test data point [True]
334 Text feature [activation] present in test data point [True]
Out of the top  500  features  15 are present in query point
```

4.3.2. Without Class balancing

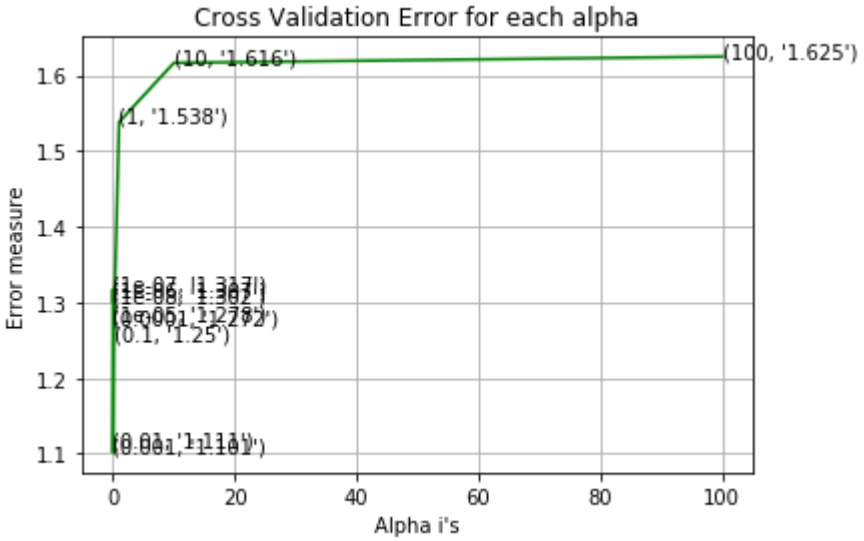
4.3.2.1. Hyper paramter tuning

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_mo
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=N
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
# 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-intuiti
#-------------------------------



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklea
# ----------------------------
# 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(-8, 3)]
cv_log_error_array = []
for i in alpha:
```

```
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_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, pred
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_
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, predic
```

```
for alpha = 1e-08
Log Loss : 1.301898440354458
```

```
for alpha = 1e-07
Log Loss : 1.3171176156919957
for alpha = 1e-06
Log Loss : 1.3074648254683952
for alpha = 1e-05
Log Loss : 1.2778335725673895
for alpha = 0.0001
Log Loss : 1.2718700994639265
for alpha = 0.001
Log Loss : 1.1014973044058862
for alpha = 0.01
Log Loss : 1.1110127984170965
for alpha = 0.1
Log Loss : 1.2501172885189447
for alpha = 1
Log Loss : 1.5380488780721915
for alpha = 10
Log Loss : 1.6162602554168248
for alpha = 100
Log Loss : 1.6245371194726435
```

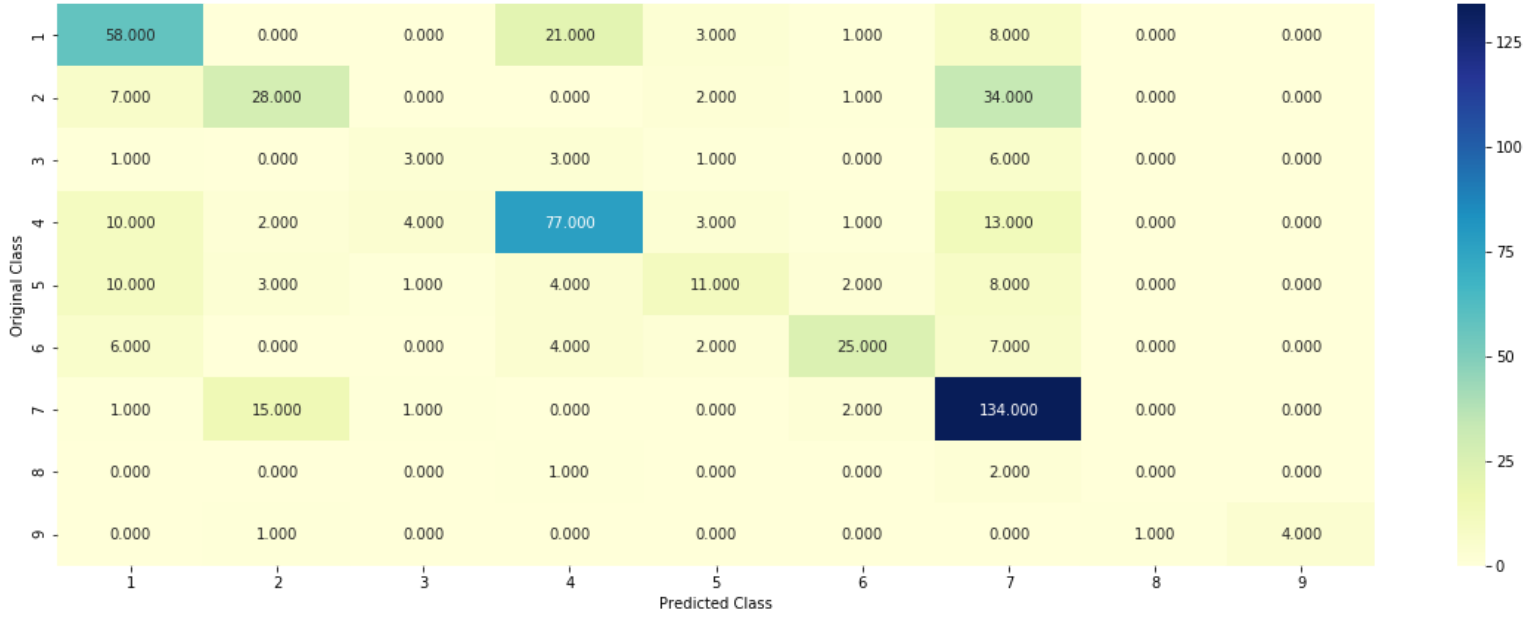



```
For values of best alpha =  0.001 The train log loss is: 0.648003444102412
For values of best alpha =  0.001 The cross validation log loss is: 1.1014973044058862
For values of best alpha =  0.001 The test log loss is: 1.1079192403984552
```

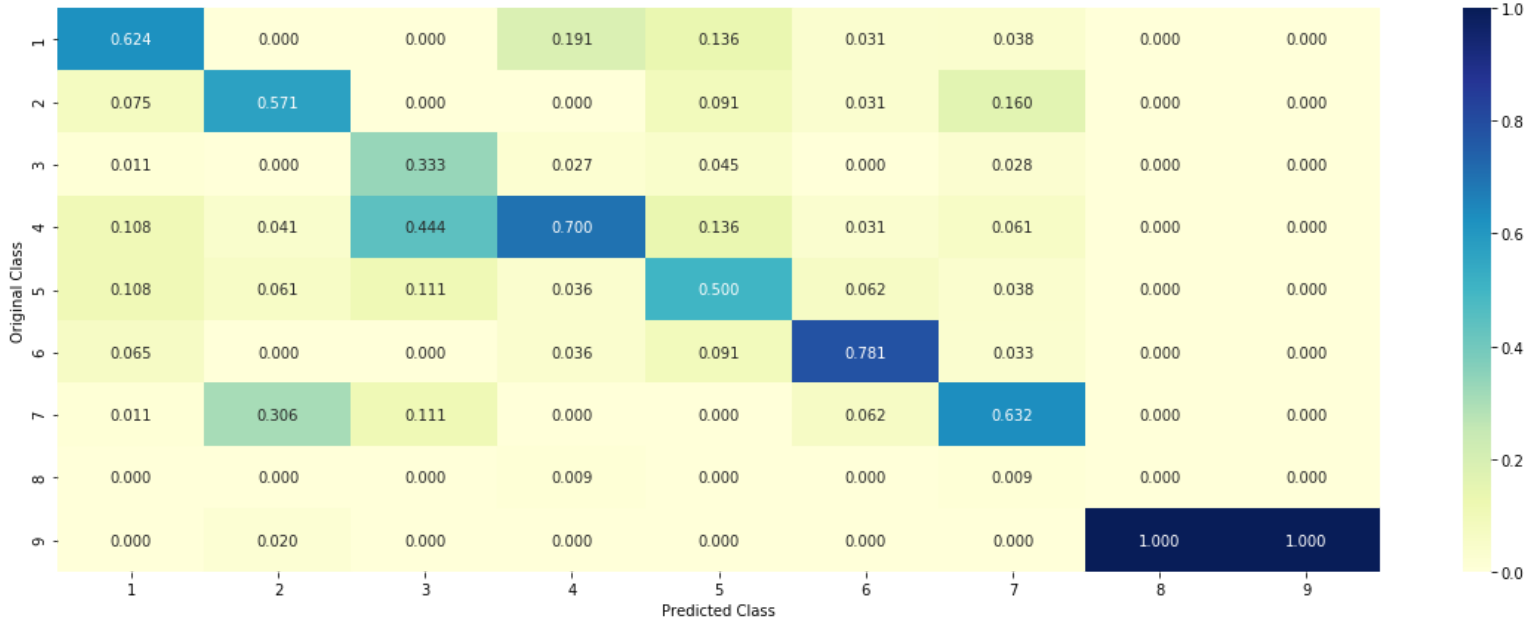
4.3.2.2. Testing model with best hyper parameters

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_mo
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=N
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
# 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)
```

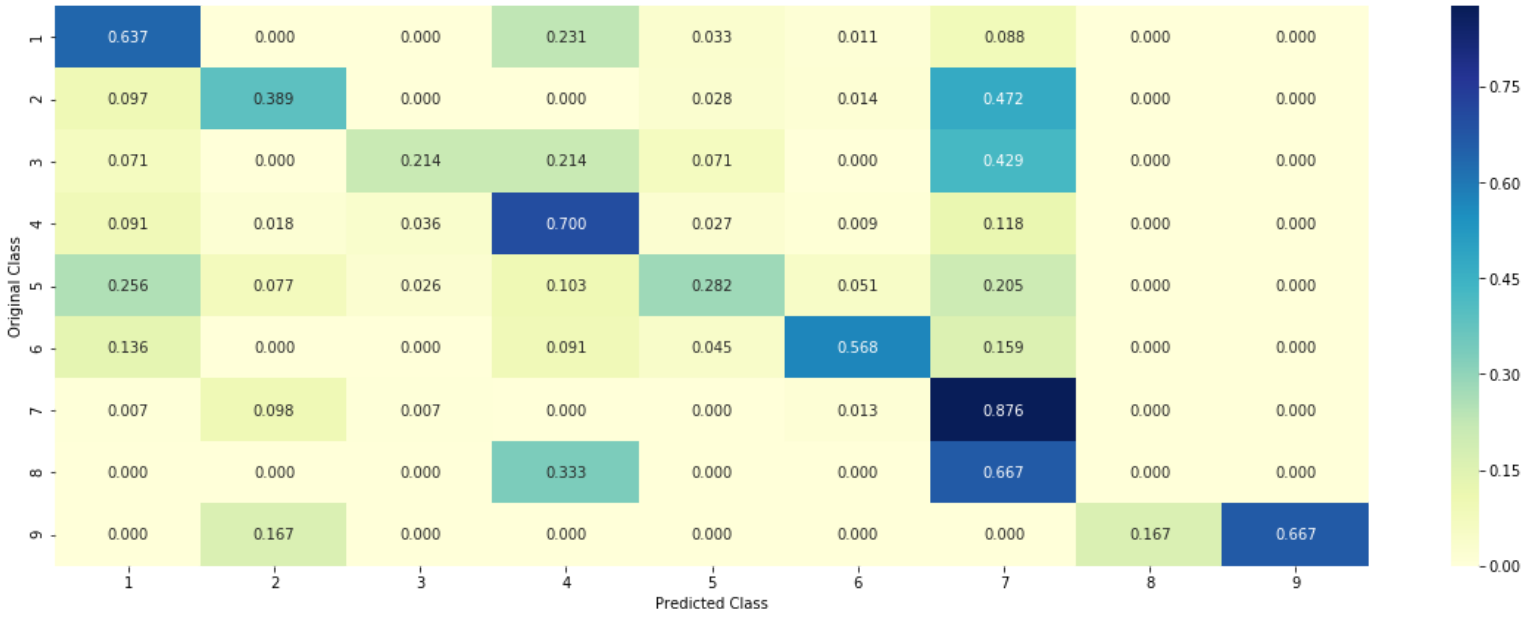
```
Log loss : 1.1014973044058862
Number of mis-classified points : 0.3609022556390977
-------------------- Confusion matrix --------------------
```



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



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



4.3.2.3. Feature Importance, Correctly Classified point

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_i
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_
```

```
Predicted Class : 7
Predicted Class Probabilities: [[5.100e-03 1.255e-01 2.000e-04 1.300e-03 2.300e-03 1.400e-03 8.556e-01
  8.500e-03 1.000e-04]]
Actual Class : 7
--------------------------------------------------
60 Text feature [constitutively] present in test data point [True]
107 Text feature [flt1] present in test data point [True]
124 Text feature [cysteine] present in test data point [True]
157 Text feature [oncogenes] present in test data point [True]
158 Text feature [inhibited] present in test data point [True]
195 Text feature [activating] present in test data point [True]
200 Text feature [ligand] present in test data point [True]
203 Text feature [oncogene] present in test data point [True]
204 Text feature [technology] present in test data point [True]
257 Text feature [gaiix] present in test data point [True]
260 Text feature [concentrations] present in test data point [True]
265 Text feature [downstream] present in test data point [True]
314 Text feature [hki] present in test data point [True]
316 Text feature [dramatic] present in test data point [True]
323 Text feature [expressing] present in test data point [True]
371 Text feature [cdnas] present in test data point [True]
380 Text feature [viability] present in test data point [True]
412 Text feature [thyroid] present in test data point [True]
459 Text feature [activation] present in test data point [True]
461 Text feature [manageable] present in test data point [True]
462 Text feature [ser473] present in test data point [True]
468 Text feature [axilla] present in test data point [True]
```

```
495 Text feature [extracellular] present in test data point [True]
Out of the top  500  features  23 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_i
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_
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0485 0.1851 0.0052 0.0442 0.0617 0.0143 0.6317 0.0072 0.0022]]
Actual Class : 7
--------------------------------------------------
60 Text feature [constitutively] present in test data point [True]
89 Text feature [constitutive] present in test data point [True]
116 Text feature [activated] present in test data point [True]
158 Text feature [inhibited] present in test data point [True]
159 Text feature [transforming] present in test data point [True]
193 Text feature [receptors] present in test data point [True]
195 Text feature [activating] present in test data point [True]
203 Text feature [oncogene] present in test data point [True]
226 Text feature [transform] present in test data point [True]
241 Text feature [isozyme] present in test data point [True]
265 Text feature [downstream] present in test data point [True]
377 Text feature [agar] present in test data point [True]
442 Text feature [interatomic] present in test data point [True]
459 Text feature [activation] present in test data point [True]
Out of the top  500  features  14 are present in query point
```

Summary

```
from prettytable import PrettyTable
x=PrettyTable()
x.field_names = ["Model","Train loss", "CV loss", "Test Loss", "Mis-classified pts"]
x.add_row(["LR(class balancing) with one hot encoding",0.6604,1.0897,1.1094,0.3703])
x.add_row(["LR(without class balancing) with one hot encoding",0.6480,1.1014,1.1079,0.3609])

print(x)
```

```
+-------------------------------------------------+------------+---------+-----------+--------------------+
|                      Model                      | Train loss | CV loss | Test Loss | Mis-classified pts |
+-------------------------------------------------+------------+---------+-----------+--------------------+
|     LR(class balancing) with one hot encoding   |   0.6604   |  1.0897 |   1.1094  |       0.3703       |
| LR(without class balancing) with one hot encoding |   0.648    |  1.1014 |   1.1079  |       0.3609       |
+-------------------------------------------------+------------+---------+-----------+--------------------+
```

In case of BOW LR(without class balancing) performed slightly better.