# → PERSONALIZED CANCER DIAGNOSIS

## **Problem Statement:**

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

# **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.

```
1 import pandas as pd
 2 import matplotlib.pyplot as plt
 3 import time
 4 import warnings
 5 import numpy as np
 6 warnings.filterwarnings("ignore")
 7 from sklearn.metrics import confusion_matrix
 8 from sklearn import metrics
 9 from sklearn.metrics import roc_curve, auc
10 from nltk.stem.porter import PorterStemmer
11 import re
12 import string
13 from nltk.corpus import stopwords
14 from nltk.stem import PorterStemmer
15 from nltk.stem.wordnet import WordNetLemmatizer
16 from gensim.models import Word2Vec
17 from gensim.models import KeyedVectors
18 import pickle
19 from tqdm import tqdm
20 import os
 1 from google.colab import drive
 3 drive.mount('/content/drive')
   Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pf
     Enter your authorization code:
```

## ▼ READ THE TRAINING VARIANTS DATA

Mounted at /content/drive

```
2 print('Number of data points : ', data.shape[0])
3 print('Number of features : ', data.shape[1])
4 print('Features : ', data.columns.values)
5 data.head()
```

Number of data points : 3321
Number of features : 4

Features : ['ID' 'Gene' 'Variation' 'Class'] TD Gene Variation Class 0 FAM58A Truncating Mutations CBL W802\* 2 2 CBL Q249E 3 **CBL** N454D 3 3 L399V **CBL** 4

training/training\_variants is a comma separated file containing the description of the genetic mutations used for training. Fields

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

# ▼ READ THE TEXT DATA

```
1 data_text =pd.read_csv("/content/drive/My Drive/Colab Notebooks/Personalized Cancer Diagnosis/training_text",sep="
2 print('Number of data points : ', data_text.shape[0])
3 print('Number of features : ', data_text.shape[1])
4 print('Features : ', data_text.columns.values)
5 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...
               Abstract Background Non-small cell lung canc...
     1
         1
         2
     2
               Abstract Background Non-small cell lung canc...
            Recent evidence has demonstrated that acquired...
     3
            Oncogenic mutations in the monomeric Casitas B...
1 import nltk
2 nltk.download('stopwords')
   [nltk_data] Downloading package stopwords to /root/nltk_data...
    [nltk_data]
                  Unzipping corpora/stopwords.zip.
    True
```

## ▼ APPLY NLP PREPROCESSING TASK

1 stop\_words = set(stopwords.words('english'))

```
1 def nlp preprocessing(total text,index,column):
2 if type(total text) is not int:
3
      string = ""
4
      # REPLACE EVERY SPECIAL CHARACTER WITH THE SPACE
5
      total_text=re.sub('[^a-zA-Z0-9\n]',' ',total_text)
6
7
      # REPLACE MULTIPLE SPACES WITH SINGLE SPACE
8
9
      total_text=re.sub('\s+',' ', total_text)
10
11
      # CONVERT ALL THE CHARACTER TO LOWER CASE
12
     total text=total text.lower()
13
14
     for word in total_text.split() :
        if not word in stop words: # IF THE WORD IS NOT STOP WORD THEN RETAIN THAT WORD AND ASSINGN IN STRING VAF
15
          string+=word + " "
16
17
      data_text[column][index] = string
1 start time = time.clock()
3 for index,row in data_text.iterrows():
   if type(row['TEXT']) is str:
      nlp_preprocessing(row['TEXT'],index,'TEXT')
6
    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
Time: 29.239386999999997 seconds
```

8 print('Time : ',time.clock() - start\_time, "seconds")

```
1 # MERGE THE DATA (GENE AND VARIATIONS) & TEXT DATA BASED ON THE ID
2 final_data = pd.merge(data,data_text,on='ID',how='left')
3
4 final_data.head()
```

₽		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

```
1 final_data[final_data.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

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

```
1 from sklearn.model_selection import train_test_split
2 from sklearn.neighbors import KNeighborsClassifier
3 from sklearn.metrics import accuracy_score
4 from sklearn.model_selection import cross_val_score
5 from collections import Counter
6 from sklearn.metrics import confusion_matrix
7 from sklearn.metrics import classification_report
8
9
10 Y_Class=final_data['Class'].values
11 final_data.Gene=final_data.Gene.str.replace('\s+','_')
12 final_data.Variation=final_data.Variation.str.replace('\s+','_')
13
14 # SPLIT THE DATA INTO TEST,TRAIN AND CV
15 X_1,X_Test,Y_1,Y_Test=train_test_split(final_data,Y_Class,stratify=Y_Class,test_size=0.2)
16 X_Train,X_CV,Y_Train,Y_CV=train_test_split(X_1,Y_1,test_size=0.2)
```

```
1 print('Number of data points in train data:', X_Train.shape[0])
2 print('Number of data points in test data:', X_Train.shape[0])
3 print('Number of data points in cross validation data:', X_CV.shape[0])
```

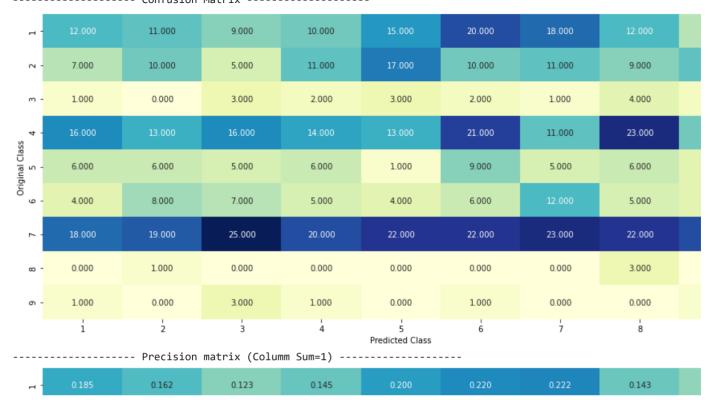
Number of data points in train data: 2124
Number of data points in test data: 2124
Number of data points in cross validation data: 532

```
1 def plot_confusion_matrix(test_y,predict_y):
2    C = confusion_matrix(test_y,predict_y)
4
    A=(((C.T)/(C.sum(axis=1))).T)
5
    B = (C/C.sum(axis=0))
7
   labels = [1,2,3,4,5,6,7,8,9]
8
9
    print("-"*20,"Confusion Matrix","-"*20)
10
11
    plt.figure(figsize = (20,7))
12
   sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
13
   plt.xlabel('Predicted Class')
14
    plt.ylabel('Original Class')
15
16
    plt.show()
17
    print("-"*20, "Precision matrix (Column Sum=1)", "-"*20)
18
19
   plt.figure(figsize=(20,7))
20
   sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
21
   plt.xlabel('Predicted Class')
22
    plt.ylabel('Original Class')
23
    plt.show()
24
   print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
25
26 plt.figure(figsize=(20,7))
27
   sns.heatmap(A, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
28
    plt.xlabel('Predicted Class')
29
    plt.ylabel('Original Class')
30 plt.show()
```

```
1 from sklearn.metrics import log_loss
2 import seaborn as sns
3 X_Test_len = X_Test.shape[0]
```

```
4 X_CV_len = X_CV.shape[0]
 5
 6 Y_CV_Predicted = np.zeros((X_CV_len,9))
 8 for i in range(X_CV_len):
9 rand_probs=np.random.rand(1,9)
10 Y CV Predicted[i] =((rand probs/sum(sum(rand probs)))[0])
11 print("Log Loss on CV using Random Model",log_loss(Y_CV, Y_CV_Predicted,eps=1e-15))
12
13
14 Y_Test_Predicted = np.zeros((X_Test_len,9))
15
16 for i in range(X_Test_len ):
17 rand_probs=np.random.rand(1,9)
18 Y_Test_Predicted[i] =((rand_probs/sum(sum(rand_probs)))[0])
19 print("Log Loss on CV using Random Model",log_loss(Y_Test, Y_Test_Predicted,eps=1e-15))
20
21 predicted_y =np.argmax(Y_Test_Predicted, axis=1)
22 plot_confusion_matrix(Y_Test, predicted_y+1)
```

Log Loss on CV using Random Model 2.4338510975006904 Log Loss on CV using Random Model 2.4719389345050624



# **→ UNIVARIATE ANALYSIS**

```
U.UZZ
1 def get_gene_variation_feature_dic(alpha,feature,df):
2
3
    value_count=df[feature].value_counts()
    print("Value Count :", value_count)
    gene_var = dict()
6
7
    for i,denominator in value_count.items():
8
      vec=[]
9
10
      for k in range(1,10):
11
        class_count=df.loc[(df['Class'] == k) & (df[feature] == i)]
12
        vec.append((class_count.shape[0] + alpha*10)/ (denominator + 90*alpha))
                                                                                   #Laplace Smoothing
13
      gene_var[i]=vec
14
    return gene_var
```

```
Predicted Class
1 def get_gene_variation_features(alpha,feature,df):
3
    gv_dict=get_gene_variation_feature_dic(alpha,feature,df)
4
5
    value_count=df[feature].value_counts()
6
7
    gv_fea=[]
    print("DF Iteration_rows", df.iterrows())
8
9
    for index,row in df.iterrows():
10
      if row[feature] in dict(value_count).keys():
        gv_fea.append(gv_dict[row[feature]])
11
12
13
        gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
14
    return gv_fea
```

# ▼ Univariate Analysis on Gene Features

```
1 unique_genes=X_Train['Gene'].value_counts()
3 print("Number of Unique Genes :",unique genes.shape[0])
5 print(unique_genes.head(10))
   Number of Unique Genes: 231
С⇒
    BRCA1
              159
    TP53
              103
    EGFR
               93
    BRCA2
               86
    PTEN
               84
    KIT
               72
    BRAF
               61
    ERBB2
               49
    ΔΙΚ
               45
    PDGFRA
               38
    Name: Gene, dtype: int64
```

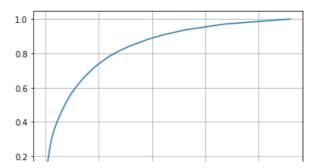
Looking at the count, looks like there are 236 different categories of Gene thats are in Training Data

## Distribution are as follows

```
1 s = sum(unique_genes.values);
2 h = unique_genes.values/s;
3 plt.plot(h, label="Histrogram of Genes")
4 plt.xlabel('Index of a Gene')
5 plt.ylabel('Number of Occurances')
6 plt.legend()
7 plt.grid()
8 plt.show()
```

```
0.07 Histrogram of Genes
0.06 0.05 0.04 0.03 0.02 0.01 0.00 150 200 Index of a Gene
```

```
1 c = np.cumsum(h)
2 plt.plot(c,label='Cumulative distribution of Genes')
3 plt.grid()
4 plt.legend()
5 plt.show()
```



There are 2 ways we can featurize this variable.

- 1. One Hot Encoding
- 2. Response Coding

We will choose the appropriate featurization based on the ML model we use.

# → BAG OF WORDS VECTORIZATION TECHNIQUE

# ▼ Response Coding Method on Gene Feature

```
1 alpha = 1
2
3 X_Train_gene_Feature_responsecoding = np.array(get_gene_variation_features(alpha,"Gene",X_Train))
4 print("Train Gene Feature :",X_Train_gene_Feature_responsecoding.shape)
5
6 print("="*100)
7
8 X_Test_gen_Feature_responsecoding = np.array(get_gene_variation_features(alpha,"Gene",X_Test))
9 print("Test Gene Feature :",X_Test_gen_Feature_responsecoding.shape)
10
11 print("="*100)
12
13 X_CV_gene_Feature_responsecoding = np.array(get_gene_variation_features(alpha,"Gene",X_CV))
14 print("CV Gene Feature :",X_CV_gene_Feature_responsecoding.shape)
15
16 print("="*100)
```

```
Value Count : BRCA1
                   159
TP53
      103
EGFR
        93
BRCA2
        86
PTEN
        84
ARID1B
        1
WHSC1
         1
ATRX
         1
CDK8
         1
CCND2
         1
Name: Gene, Length: 231, dtype: int64
DF Iteration rows <generator object DataFrame.iterrows at 0x7fd82baa7308>
Train Gene Feature : (2124, 9)
______
Value Count : BRCA1
TP53
       34
EGFR
        31
PTEN
        25
BRAF
        19
MPI
        1
CDK12
        1
FAM58A
        1
ETV6
        1
Name: Gene, Length: 162, dtype: int64
```

# ▼ One Hot Encoding Method on Gene feature

```
VALUE COUIT . DIVENT
 1 from sklearn.feature_extraction.text import CountVectorizer
 2 gene_vectorizer=CountVectorizer()
3
4 X Train gene Feature onehotEncoding=gene vectorizer.fit transform(X Train["Gene"])
 5 print(" Train Gene Feature : ",X Train gene Feature onehotEncoding.shape)
7 print("="*100)
9 X Test gene Feature onehotencoding=gene vectorizer.transform(X Test["Gene"])
10 print(" Test Gene Feature : ",X_Test_gene_Feature_onehotencoding.shape)
12 print("="*100)
13
14
15 X CV gene Feature onehotencoding=gene vectorizer.transform(X CV["Gene"])
16 print(" CV Gene Feature :" ,X_CV_gene_Feature_onehotencoding.shape)
17
18 print("="*100)
```

```
Train Gene Feature : (2124, 231)

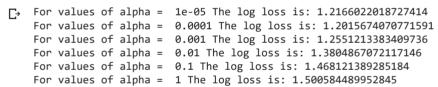
Test Gene Feature : (665, 231)

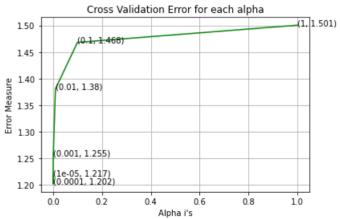
CV Gene Feature : (532, 231)
```

#### APPLY SVM --> SGD CLASSIFIER TO FIND THE BEST HYPERPARAMETER

```
1 from sklearn.linear_model import SGDClassifier
2 from sklearn.calibration import CalibratedClassifierCV
3 alpha = [10 ** x for x in range(-5, 1)]
4
5 cv_log_error=[]
6
```

```
7 for i in alpha:
   clf=SGDClassifier(alpha=i, penalty='12', loss='log', random state=42)
   clf.fit(X Train gene Feature onehotEncoding,Y Train)
10 sig clf=CalibratedClassifierCV(clf,method="sigmoid")
    sig_clf.fit(X_Train_gene_Feature_onehotEncoding,Y_Train)
11
    Predicted Y=sig clf.predict proba(X CV gene Feature onehotencoding)
13
    cv log error.append(log loss(Y CV,Predicted Y,labels=clf.classes ,eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(Y_CV, Predicted_Y, labels=clf.classes_, eps=1e-15
14
15
16 fig,ax = plt.subplots()
17 ax.plot(alpha,cv_log_error,c='g')
19 for i , txt in enumerate(np.round(cv log error,3)):
20 ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error[i]))
21 plt.grid()
22 plt.title("Cross Validation Error for each alpha")
23 plt.xlabel("Alpha i's")
24 plt.ylabel("Error Measure")
25 plt.show()
26
27 best_alpha = np.argmin(cv_log_error)
28 clf = SGDClassifier(alpha=alpha[best alpha], penalty='l2', loss='log', random state=42)
29 clf.fit(X Train gene Feature onehotEncoding, Y Train)
30 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
31 sig_clf.fit(X_Train_gene_Feature_onehotEncoding, Y_Train)
32
33 predict_y = sig_clf.predict_proba(X_Train_gene_Feature_onehotEncoding)
34 print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(Y_Train, predict_y, labe
35 predict_y = sig_clf.predict_proba(X_CV_gene_Feature_onehotencoding)
36 print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(Y_CV, predict
37 predict_y = sig_clf.predict_proba(X_Test_gene_Feature_onehotencoding)
38 print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(Y_Test, predict_y, labels
```





```
For values of best alpha = 0.0001 The train log loss is: 0.9882420694256845

For values of best alpha = 0.0001 The cross validation log loss is: 1.2015674070771591

For values of best alpha = 0.0001 The test log loss is: 1.1981102682844584
```

```
1 print("Q6. How many data points in Test and CV datasets are covered by the ", unique_genes.shape[0], " genes in tr
2
3 test_coverage=X_Test[X_Test['Gene'].isin(list(set(X_Train['Gene'])))].shape[0]
4 cv_coverage=X_CV[X_CV['Gene'].isin(list(set(X_Train['Gene'])))].shape[0]
5
6 print('Ans\n1. In test data',test_coverage, 'out of',X_Test.shape[0], ":",(test_coverage/X_Test.shape[0])*100)
7 print('2. In coose validation data' ov coverage. 'out of', X_CV chang[0].":", (ov coverage/X_CV chang[0])*100)
```

```
/ primic ( ב. in cross validacion data ,cv_coverage, out or ,n_cv.shape[ש], . ,(cv_coverage/n_cv.shape[ש]). אויי
```

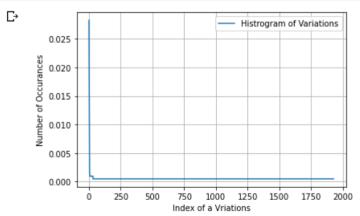
- Q6. How many data points in Test and CV datasets are covered by the 231 genes in train dataset? Ans
  - 1. In test data 645 out of 665 : 96.99248120300751
  - 2. In cross validation data 514 out of 532 : 96.61654135338345

# Univariate Analysis on Variation Features

```
1 unique_variations=X_Train['Variation'].value_counts()
2
3 print("Number of Unique Genes :",unique_variations.shape[0])
4
5 print(unique_variations.head(10))
```

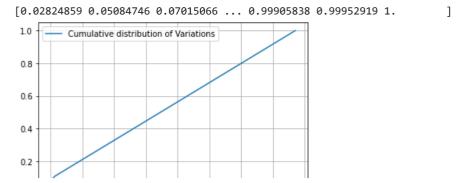
Number of Unique Genes : 1929 Truncating Mutations Deletion Amplification 41 Fusions 18 Overexpression 4 3 061L G12V 3 R170W 2 061H 2 Promoter Hypermethylation Name: Variation, dtype: int64

```
1 s = sum(unique_variations.values);
2 h = unique_variations.values/s;
3 plt.plot(h, label="Histrogram of Variations")
4 plt.xlabel('Index of a Vriations')
5 plt.ylabel('Number of Occurances')
6 plt.legend()
7 plt.grid()
8 plt.show()
```



```
1 c = np.cumsum(h)
2 print(c)
3 plt.plot(c,label='Cumulative distribution of Variations')
4 plt.grid()
5 plt.legend()
6 plt.show()
```

C→



There are 2 ways we can featurize this variable.

- 1. One Hot Encoding
- 2. Response Coding

We will choose the appropriate featurization based on the ML model we use.

# Response Coding Method on Variation Features

```
1 alpha = 1
2
3 X_Train_variation_Feature_responsecoding = np.array(get_gene_variation_features(alpha, "Variation", X_Train))
4 print("Train Variation Feature :", X_Train_variation_Feature_responsecoding.shape)
5
6 print("="*100)
7
8 X_Test_variation_Feature_responsecoding = np.array(get_gene_variation_features(alpha, "Variation", X_Test))
9 print("Test Variation Feature :", X_Test_variation_Feature_responsecoding.shape)
10
11 print("="*100)
12
13 X_CV_variation_Feature_responsecoding = np.array(get_gene_variation_features(alpha, "Variation", X_CV))
14 print("CV Variation Feature :", X_CV_variation_Feature_responsecoding.shape)
15
16 print("="*100)
```

Г⇒

```
Value Count : Truncating_Mutations
                               60
Deletion
Amplification
                   41
Fusions
                   18
Overexpression
                    4
L122R
                    1
T34 A289del
                    1
EWSR1-FLI1 Fusion
                    1
C2060G
                    1
P287A
Name: Variation, Length: 1929, dtype: int64
DF Iteration rows <generator object DataFrame.iterrows at 0x7fd826c7e7d8>
Train Variation Feature : (2124, 9)
______
Value Count : Deletion
                              16
Truncating_Mutations 16
Amplification
                   14
Fusions
                   11
061R
                    2
A1131T
                    1
KEOOD
```

# One Hot Encoding on Variation Feature

```
Name: Variation Length: 611. dtyne: int64

1 from sklearn.feature_extraction.text import CountVectorizer
2 variation_vectorizer=CountVectorizer()

3

4 X_Train_variation_Feature_onehotEncoding=variation_vectorizer.fit_transform(X_Train["Variation"])

5 print(" Train Variation Feature :" ,X_Train_variation_Feature_onehotEncoding.shape)

6

7 print("="*100)

8

9 X_Test_variation_Feature_onehotencoding=variation_vectorizer.transform(X_Test["Variation"])

10 print(" Test Variation Feature :" ,X_Test_variation_Feature_onehotencoding.shape)

11

12 print("="*100)

13

14

15 X_CV_variation_Feature_onehotencoding=variation_vectorizer.transform(X_CV["Variation"])

16 print(" CV Variation Feature :" ,X_CV_variation_Feature_onehotencoding.shape)

17

18 print("="*100)
```

```
Train Variation Feature : (2124, 1956)

-------
Test Variation Feature : (665, 1956)

------
CV Variation Feature : (532, 1956)
```

## ▼ APPLY SVM --> SGD CLASSIFIER TO FIND THE BEST HYPERPARAMETER

```
1 from sklearn.linear_model import SGDClassifier
2 from sklearn.calibration import CalibratedClassifierCV
3 alpha = [10 ** x for x in range(-5, 1)]
4
5 cv_log_error=[]
6
7 for i in alpha :
8 clf=SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
9 clf.fit(X_Train_variation_Feature_onehotEncoding,Y_Train)
10 sig_clf=CalibratedClassifierCV(clf,method="sigmoid")
```

```
11
    sig clf.fit(X Train variation Feature onehotEncoding,Y Train)
12
    Predicted Y=sig clf.predict proba(X CV variation Feature onehotencoding)
13
    cv_log_error.append(log_loss(Y_CV,Predicted_Y,labels=clf.classes_,eps=1e-15))
14
    print('For values of alpha = ', i, "The log loss is:",log loss(Y CV, Predicted Y, labels=clf.classes , eps=1e-15
15
16 fig,ax = plt.subplots()
17 ax.plot(alpha,cv_log_error,c='g')
19 for i , txt in enumerate(np.round(cv_log_error,3)):
20 ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv log error[i]))
21 plt.grid()
22 plt.title("Cross Validation Error for each alpha")
23 plt.xlabel("Alpha i's")
24 plt.ylabel("Error Measure")
25 plt.show()
26
27 best_alpha = np.argmin(cv_log_error)
28 clf = SGDClassifier(alpha=alpha[best alpha], penalty='12', loss='log', random_state=42)
29 clf.fit(X Train variation Feature onehotEncoding, Y Train)
30 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
31 sig clf.fit(X Train variation Feature onehotEncoding, Y Train)
33 predict y = sig clf.predict proba(X Train variation Feature onehotEncoding)
34 print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(Y_Train, predict_y, labe
35 predict_y = sig_clf.predict_proba(X_CV_variation_Feature_onehotencoding)
36 print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(Y_CV, predict
37 predict y = sig clf.predict proba(X Test variation Feature onehotencoding)
38 print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(Y_Test, predict_y, labels
```

```
For values of alpha = 1e-05 The log loss is: 1.7368916556958085

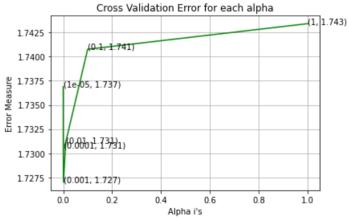
For values of alpha = 0.0001 The log loss is: 1.7305804509088687

For values of alpha = 0.001 The log loss is: 1.7269911213724305

For values of alpha = 0.01 The log loss is: 1.731115111931615

For values of alpha = 0.1 The log loss is: 1.7407613800259418

For values of alpha = 1 The log loss is: 1.7434113605325594
```



```
For values of best alpha = 0.001 The train log loss is: 1.0374898189029886

For values of best alpha = 0.001 The cross validation log loss is: 1.7269911213724305

For values of best alpha = 0.001 The test log loss is: 1.704806332912111
```

```
1 test_coverage=X_Test[X_Test['Variation'].isin(list(set(X_Train['Variation'])))].shape[0]
2 cv_coverage=X_CV[X_CV['Variation'].isin(list(set(X_Train['Variation'])))].shape[0]
3
4 print('1. In test data',test_coverage, 'out of',X_Test.shape[0], ":",(test_coverage/X_Test.shape[0])*100)
5 print('2. In cross validation data',cv_coverage, 'out of ',X_CV.shape[0],":",(cv_coverage/X_CV.shape[0])*100)
```

```
1. In test data 66 out of 665 : 9.924812030075188
2. In cross validation data 59 out of 532 : 11.090225563909774
```

```
1 sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , reverse=True))
2 sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

### ▼ APPLY SVM --> SGD CLASSIFIER TO FIND THE BEST HYPERPARAMETER

```
1 from sklearn.linear model import SGDClassifier
2 from sklearn.calibration import CalibratedClassifierCV
3 \text{ alpha} = [10 ** x \text{ for } x \text{ in range}(-5, 1)]
5 cv_log_error=[]
6
7 for i in alpha:
8 clf=SGDClassifier(alpha=i, penalty='12', loss='log', random_state=42)
9 clf.fit(X_Train_feature_onehotCoding,Y_Train)
10 sig_clf=CalibratedClassifierCV(clf,method="sigmoid")
sig_clf.fit(X_Train_feature_onehotCoding,Y_Train)
12 Predicted Y=sig clf.predict proba(X CV text feature onehotCoding)
13
   cv log error.append(log loss(Y CV,Predicted Y,labels=clf.classes ,eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(Y_CV, Predicted_Y, labels=clf.classes_, eps=1e-15
14
15
16 fig,ax = plt.subplots()
17 ax.plot(alpha,cv_log_error,c='g')
19 for i , txt in enumerate(np.round(cv_log_error,3)):
20 ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv log error[i]))
21 plt.grid()
22 plt.title("Cross Validation Error for each alpha")
23 plt.xlabel("Alpha i's")
24 plt.ylabel("Error Measure")
25 plt.show()
27 best_alpha = np.argmin(cv_log_error)
28 clf = SGDClassifier(alpha=alpha[best_alpha], penalty='12', loss='log', random_state=42)
29 clf.fit(X_Train_feature_onehotCoding, Y_Train)
30 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
31 sig_clf.fit(X_Train_feature_onehotCoding, Y_Train)
32
33 predict_y = sig_clf.predict_proba(X_Train_feature_onehotCoding)
34 print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(Y_Train, predict_y, labe
35 predict y = sig clf.predict proba(X CV text feature onehotCoding)
36 print('For values of best alpha = ', alpha[best alpha], "The cross validation log loss is:",log loss(Y CV, predict
37 predict_y = sig_clf.predict_proba(X_Test_text_feature_onehotCoding)
38 print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(Y_Test, predict_y, labels
```

Г⇒

# Univariate Analysis on Text Features

```
1 def extract dictionary paddle(cls text):
 2
      dictionary = defaultdict(int)
3
      for index, row in cls_text.iterrows():
4
          for word in row['TEXT'].split():
5
               dictionary[word] +=1
      return dictionary
1 import math
2
3 def get_text_responsecoding(df):
      text_feature_responseCoding = np.zeros((df.shape[0],9))
4
      for i in range(0,9):
          row index = 0
6
7
          for index, row in df.iterrows():
8
               sum_prob = 0
9
               for word in row['TEXT'].split():
10
                   sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_dict.get(word,0)+90)))
11
               text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEXT'].split()))
12
               row index += 1
13
      return text_feature_responseCoding
```

# ▼ One Hot Encoding on Text Feature

```
1 text_vectorizer = CountVectorizer(min_df=3)
2 X_Train_feature_onehotCoding = text_vectorizer.fit_transform(X_Train['TEXT'])
3 # getting all the feature names (words)
4 X_Train_text_features= text_vectorizer.get_feature_names()
5
6 # train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of features) vector
7 train_text_fea_counts = X_Train_feature_onehotCoding.sum(axis=0).A1
8
9 # zip(list(text_features),text_fea_counts) will zip a word with its number of times it occured
10 text_fea_dict = dict(zip(list(X_Train_text_features),train_text_fea_counts))
11
12
13 print("Total number of unique words in train data :", len(X_Train_text_features))
```

#### Total number of unique words in train data : 52882

```
1 from collections import defaultdict
 2 dict_list = []
 3 # dict_list =[] contains 9 dictoinaries each corresponds to a class
 4 for i in range(1,10):
      cls_text = X_Train[X_Train['Class']==i]
       # build a word dict based on the words in that class
 7
       dict_list.append(extract_dictionary_paddle(cls_text))
 8
       # append it to dict_list
 9
11 total_dict = extract_dictionary_paddle(X_Train)
12
13
14 confuse array = []
15 for i in X_Train_text_features:
16
    ratios = []
17
      \max_{} val = -1
18
      for j in range(0,9):
           ratios.append((dict_list[j][i]+10 )/(total_dict[i]+90))
```

```
20 confuse_array.append(ratios)
21 confuse_array = np.array(confuse_array)
```

#### Response Coding on Text Feature

```
1 X_Train_text_feature_responseCoding = get_text_responsecoding(X_Train)
2 print("Train Text Feature :", X_Train_text_feature_responseCoding.shape)
3
4 print("="*100)
5
6 X_Test_text_feature_responseCoding = get_text_responsecoding(X_Test)
7 print("Test Text Feature :", X_Test_text_feature_responseCoding.shape)
8
9 print("="*100)
10 X_CV_text_feature_responseCoding = get_text_responsecoding(X_CV)
11 print("CV Text Feature :", X_CV_text_feature_responseCoding.shape)
12
13 print("="*100)
```

Train Text Feature: (2124, 9)

-----
Test Text Feature: (665, 9)

-----
CV Text Feature: (532, 9)

```
1 X_Train_text_feature_responseCoding = (X_Train_text_feature_responseCoding.T/X_Train_text_feature_responseCoding.s  
2 X_Test_text_feature_responseCoding = (X_Test_text_feature_responseCoding.T/X_Test_text_feature_responseCoding.sum(  
3 X_CV_text_feature_responseCoding = (X_CV_text_feature_responseCoding.T/X_CV_text_feature_responseCoding.sum(  
axis=1
```

#### Normalize the features

```
1 from sklearn.preprocessing import normalize
2
3 X_Train_feature_onehotCoding = normalize(X_Train_feature_onehotCoding, axis=0)
4 print("Train Text Feature :", X_Train_feature_onehotCoding.shape)
5
6 print("="*100)
7
8 X_Test_text_feature_onehotCoding = text_vectorizer.transform(X_Test['TEXT'])
9 print("Test Text Feature :", X_Test_text_feature_onehotCoding.shape)
10
11 print("="*100)
12
13 X_Test_text_feature_onehotCoding = normalize(X_Test_text_feature_onehotCoding, axis=0)
14
15 X_CV_text_feature_onehotCoding = text_vectorizer.transform(X_CV['TEXT'])
16 print("CV Text Feature :", X_CV_text_feature_onehotCoding.shape)
17
18 print("="*100)
19
20 X_CV_text_feature_onehotCoding = normalize(X_CV_text_feature_onehotCoding, axis=0)
```

```
Train Text Feature : (2124, 52882)

Test Text Feature : (665, 52882)

CV Text Feature : (532, 52882)
```

```
For values of alpha = 1e-05 The log loss is: 1.2813491064466804
For values of alpha = 0.0001 The log loss is: 1.1531172509122398
For values of alpha = 0.001 The log loss is: 1.1393878819557273
For values of alpha = 0.01 The log loss is: 1.2716465329687872
For values of alpha = 0.1 The log loss is: 1.4686398010742088
For values of alpha = 1 The log loss is: 1.6626389802062658
```

# ▼ BUILDING A MACHINE LEARNING MODELS

```
1 def predict and plot confusionmatrix(X train,Y train,X test,Y test,clf):
2 clf.fit(X train, Y train)
3 sig_clf=CalibratedClassifierCV(clf,method="sigmoid")
4 sig clf.fit(X train,Y train)
    pred_y=sig_clf.predict(X_test)
7
8
    print("Log Loss : ",log_loss(Y_test,sig_clf.predict_proba(X_test)))
9
    print("Number of Misclassified Points :",np.count_nonzero((pred_y - Y_test))/Y_test.shape[0])
10
11
    plot_confusion_matrix(Y_test,pred_y)
    0 004 The heat less less 1 4070462024024027
1 def report_log_loss(X_train,Y_train,X_test,Y_test,clf):
2 clf.fit(X Train, Y train)
3 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
4 sig clf.fit(X train,Y train)
   sig clf probs=sig clf.predict proba(X Test)
6 return log_loss(Y_Test, sig_clf_probs, eps=1e-15)
```

# ▼ Get Feature Names

```
1 def get_impfeature_names_tfidf(indices, text, gene, var, no_features):
      gene_count_vec = TfidfVectorizer()
 3
      var_count_vec = TfidfVectorizer()
 4
      text_count_vec = TfidfVectorizer(min_df=3)
 5
      print ("Hello")
      gene_vec = gene_count_vec.fit(X_Train['Gene'])
 6
 7
      var_vec = var_count_vec.fit(X_Train['Variation'])
 8
      text_vec = text_count_vec.fit(X_Train['TEXT'])
9
10
      fea1_len = len(gene_vec.get_feature_names())
11
      fea2_len = len(var_count_vec.get_feature_names())
12
13
      word_present = 0
14
      for i,v in enumerate(indices):
15
           if (v < fea1_len):</pre>
               word = gene_vec.get_feature_names()[v]
16
17
               yes_no = True if word == gene else False
18
               if yes no:
19
                   word_present += 1
20
                   print(i, "Gene feature [{}] present in test data point [{}]".format(word,yes_no))
21
           elif (v < fea1_len+fea2_len):</pre>
22
               word = var_vec.get_feature_names()[v-(fea1_len)]
23
               yes_no = True if word == var else False
24
               if yes_no:
25
                   word_present += 1
                   print(i, "variation feature [{}] present in test data point [{}]".format(word,yes_no))
26
27
          else:
28
               word = text_vec.get_feature_names()[v-(fea1_len+fea2_len)]
29
               yes_no = True if word in text.split() else False
               if ves 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")
```

## ▼ STACKING THE THREE TYPES OF FEATURES.

```
1 from imblearn.over sampling import SMOTE
 2 from collections import Counter
 3 from scipy.sparse import hstack
 4 from sklearn.multiclass import OneVsRestClassifier
 5 from sklearn.svm import SVC
 6 from collections import Counter, defaultdict
 7 from sklearn.calibration import CalibratedClassifierCV
 8 from sklearn.naive_bayes import MultinomialNB
 9 from sklearn.naive_bayes import GaussianNB
10 from sklearn.model selection import train_test_split
11 from sklearn.model selection import GridSearchCV
12 import math
13 from sklearn.metrics import normalized_mutual_info_score
14 from sklearn.ensemble import RandomForestClassifier
15 warnings.filterwarnings("ignore")
16
17 from mlxtend.classifier import StackingClassifier
18
19 from sklearn import model_selection
20 from sklearn.linear model import LogisticRegression
21 from sklearn.linear model import SGDClassifier
22 from sklearn.calibration import CalibratedClassifierCV
23 from sklearn.preprocessing import normalize
24 from collections import defaultdict
25 import math
26 from sklearn.metrics import log loss
27 import seaborn as sns
```

```
1 X_Train_gene_var_onehotcoding = hstack((X_Train_gene_Feature_onehotEncoding,X_Train_variation_Feature_onehotEncodi
2 X Test gene var onehotcoding = hstack((X Test gene Feature onehotencoding, X Test variation Feature onehotencoding
3 X_CV_gene_var_onehotcoding = hstack((X_CV_gene_Feature_onehotencoding,X_CV_variation_Feature_onehotencoding))
5 X_Train_onehotCoding = hstack((X_Train_gene_var_onehotcoding,X_Train_feature_onehotCoding)).tocsr()
6 print("Train One Hot Encoding :", X_Train_onehotCoding.shape)
7 print("="*100)
8
9 X_Test_onehotcoding = hstack((X_Test_gene_var_onehotcoding,X_Test_text_feature_onehotCoding))
10 print("Test One Hot Encoding :", X_Test_onehotcoding.shape)
11 print("="*100)
12
13 X_CV_onehotcoding = hstack((X_CV_gene_var_onehotcoding,X_CV_text_feature_onehotCoding))
14 print("CV One Hot Encoding :", X_CV_onehotcoding.shape)
15 print("="*100)
16
17
18 X Train gene var responseCoding = np.hstack((X Train gene Feature responsecoding, X Train variation Feature respons
19 X_Test_gene_var_responseCoding = np.hstack((X_Test_gen_Feature_responsecoding,X_Test_variation_Feature_responsecod
20 X_CV_gene_var_responseCoding = np.hstack((X_CV_gene_Feature_responsecoding,X_CV_variation_Feature_responsecoding))
21
22 X_Train_responseCoding = np.hstack((X_Train_gene_var_responseCoding, X_Train_text_feature_responseCoding))
23 print("Train Response Coding :", X_Train_responseCoding.shape)
24 print("="*100)
25
26 X Test_responseCoding = np.hstack((X Test gene var_responseCoding, X_Test_text feature responseCoding))
```

```
27 print("Test Response Coding :", X_Test_responseCoding.shape)
28 print("="*100)
29 X_CV_responseCoding = np.hstack((X_CV_gene_var_responseCoding, X_CV_text_feature_responseCoding))
30 print("CV Response Coding :", X_CV_responseCoding.shape)
31 print("="*100)
```

С→

# **▼** BASE LINE MODEL

## ▼ NAIVE BAYES ALGORITHM

```
1 def Naive_Bayes_Algo(X_Train,Y_Train,X_CV,Y_CV,X_Test,Y_Test):
 2
    3
4
    cv_log_error=[]
5
   for i in alpha:
6
7
     print("for alpha =", i)
8
      clf = MultinomialNB(alpha=i)
9
      clf.fit(X_Train_onehotCoding,Y_Train)
10
      sig clf = CalibratedClassifierCV(clf, method="sigmoid")
11
      sig_clf.fit(X_Train_onehotCoding, Y_Train)
      sig clf probs = sig clf.predict proba(X CV onehotcoding)
13
      cv log error.append(log loss(Y CV, sig clf probs, labels=clf.classes , eps=1e-15))
      print("Log Loss :",log_loss(Y_CV, sig_clf_probs))
14
15
16
   fig, ax = plt.subplots()
17
    ax.plot(np.log10(alpha), cv_log_error,c='g')
18
   for i , txt in enumerate(np.round(cv_log_error,3)):
19
     ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv_log_error[i]))
20
    plt.grid()
21
    plt.xticks(np.log10(alpha))
    plt.title("Cross Validation Error for each alpha")
23
    plt.xlabel("Alpha i's")
24
   plt.ylabel("Error measure")
   plt.show()
26
27
    best_alpha = np.argmin(cv_log_error)
28
    clf = MultinomialNB(alpha=alpha[best_alpha])
29
    clf.fit(X_Train_onehotCoding, Y_Train)
   sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
30
31
    sig_clf.fit(X_Train_onehotCoding, Y_Train)
32
33
    predict_y = sig_clf.predict_proba(X_Train_onehotCoding)
34
    print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(Y_Train, predict_y, la
35
36
    predict_y = sig_clf.predict_proba(X_CV_onehotcoding)
37
    print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(Y_CV, predi
```

```
predict_y = sig_clf.predict_proba(X_Test_onehotcoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(Y_Test, predict_y, labe
return alpha,best_alpha
return alpha,best_alpha
```

1 alpha, best alpha = Naive Bayes Algo(X Train, Y Train, X CV, Y CV, X Test, Y Test)

С→

```
1 clf = MultinomialNB(alpha=alpha[best_alpha])
2 clf.fit(X_Train_onehotCoding, Y_Train)
3 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
4 sig_clf.fit(X_Train_onehotCoding, Y_Train)
5 sig_clf_probs = sig_clf.predict_proba(X_CV_onehotcoding)
6
7 print("Log Loss :",log_loss(Y_CV, sig_clf_probs))
8
9 print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(X_CV_onehotcoding) - Y_CV))/Y_CV.shape[
10 plot_confusion_matrix(Y_CV, sig_clf.predict(X_CV_onehotcoding.toarray()))
```

# ▼ K-NEAREST NEIGHBORS ALGORITHM

```
1 alpha = [5, 11, 15, 21, 31, 41, 51, 99]
2 cv_log_error_array = []
3 for i in alpha:
      print("for alpha =", i)
      clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
6
7
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
   sig_clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
9
   sig_clf_probs = sig_clf.predict_proba(X_CV_onehotcoding_tfidf)
      cv_log_error_array.append(log_loss(Y_CV, sig_clf_probs, labels=clf.classes_, eps=1e-15))
10
11
      # to avoid rounding error while multiplying probabilites we use log-probability estimates
12
      print("Log Loss :",log_loss(Y_CV, sig_clf_probs))
13
14 fig, ax = plt.subplots()
15 ax.plot(alpha, cv_log_error_array,c='g')
16 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]))
18 plt.grid()
19 plt.title("Cross Validation Error for each alpha")
20 nl+ vlahal/"Alnha i'c")
```

```
plt.Nlabel("Error measure")

plt.ylabel("Error measure")

plt.show()

best_alpha = np.argmin(cv_log_error_array)

clf.fit(X_Train_onehotcoding_tfidf, Y_Train)

sig_clf = CalibratedClassifier(V(clf, method="sigmoid"))

sig_clf.fit(X_Train_onehotcoding_tfidf, Y_Train)

predict_y = sig_clf.predict_proba(X_Train_onehotcoding_tfidf)

print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(Y_Train, predict_y, labe)

predict_y = sig_clf.predict_proba(X_CV_onehotcoding_tfidf)

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

predict_y = sig_clf.predict_proba(X_Test_onehotcoding_tfidf)

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

print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(Y_Test, predict_y, labels)
```

₽

# ▼ LOGISTIC REGRESSION (WITH CLASS BALANCING)

```
1 def Logistic_Regression_Algo(X_Train,Y_Train,X_CV,Y_CV,X_Test,Y_Test):
2
3    alpha = [10 ** x for x in range(-6, 3)]
4    cv_log_error=[]
5
6    for i in alpha:
7        print("for alpha =", i)
8        clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='12', loss='log', random_state=42)
9        clf.fit(X_Train_onehotCoding,Y_Train)
10        sig clf = CalibratedClassifierCV(clf. method="sigmoid")
```

```
sig_clf.fit(X_Train_onehotCoding, Y_Train)
11
      sig clf probs = sig clf.predict proba(X CV onehotcoding)
12
13
      cv log error.append(log loss(Y CV, sig clf probs, labels=clf.classes , eps=1e-15))
14
      print("Log Loss :",log_loss(Y_CV, sig_clf_probs))
15
16
   fig, ax = plt.subplots()
17
    ax.plot(np.log10(alpha), cv_log_error,c='g')
   for i , txt in enumerate(np.round(cv_log_error,3)):
18
19
     ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv_log_error[i]))
20
   plt.grid()
   plt.xticks(np.log10(alpha))
21
    plt.title("Cross Validation Error for each alpha")
22
    plt.xlabel("Alpha i's")
23
   plt.ylabel("Error measure")
24
25
   plt.show()
26
27
   best_alpha = np.argmin(cv_log_error)
    clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='12', loss='log', random_state=42)
28
    clf.fit(X Train onehotCoding, Y Train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
30
31
    sig clf.fit(X Train onehotCoding, Y Train)
33
    predict_y = sig_clf.predict_proba(X_Train_onehotCoding)
34
    print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(Y_Train, predict_y, la
35
36
    predict y = sig clf.predict_proba(X CV_onehotcoding)
37
    print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(Y_CV, predi
38
    predict_y = sig_clf.predict_proba(X_Test_onehotcoding)
39
    print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(Y_Test, predict_y, labe
40
41
42
    return alpha, best_alpha
```

```
1 alpha,best_alpha = Logistic_Regression_Algo(X_Train,Y_Train,X_CV,Y_CV,X_Test,Y_Test)
```

 $\Box$ 

```
1 clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='12', loss='log', random_state=42)
2 predict_and_plot_confusionmatrix(X_Train_onehotCoding, Y_Train, X_CV_onehotcoding, Y_CV, clf)
```

## ▼ LOGISTIC REGRESSION WITHOUT CLASS BALANCING

```
1 def Logistic Regression Algo WithoutClassBalancing(X Train, Y Train, X CV, Y CV, X Test, Y Test):
2
3
   alpha = [10 ** x for x in range(-6, 3)]
4 cv log error=[]
5
6 for i in alpha:
      print("for alpha =", i)
7
      clf = SGDClassifier(alpha=i, penalty='12', loss='log', random state=42)
8
    clf.fit(X_Train_onehotCoding,Y_Train)
9
10
    sig clf = CalibratedClassifierCV(clf, method="sigmoid")
      sig clf.fit(X Train onehotCoding, Y Train)
11
      sig clf probs = sig clf.predict proba(X CV onehotcoding)
12
      cv log error.append(log loss(Y CV, sig clf probs, labels=clf.classes , eps=1e-15))
13
14
      print("Log Loss :",log_loss(Y_CV, sig_clf_probs))
15
16 fig, ax = plt.subplots()
   ax.plot(np.log10(alpha), cv_log_error,c='g')
17
18
   for i , txt in enumerate(np.round(cv_log_error,3)):
19
     ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv_log_error[i]))
20
    plt.grid()
    plt.xticks(np.log10(alpha))
21
    plt.title("Cross Validation Error for each alpha")
   plt.xlabel("Alpha i's")
   plt.ylabel("Error measure")
25
   plt.show()
26
    best alpha = np.argmin(cv log error)
27
   clf = SGDClassifier(alpha=alpha[best alpha], penalty='12', loss='log', random state=42)
28
   clf.fit(X Train onehotCoding, Y Train)
   sig clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X Train onehotCoding, Y Train)
31
32
33
    predict y = sig clf.predict_proba(X_Train_onehotCoding)
34
    print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(Y_Train, predict_y, la
35
    predict y = sig clf.predict proba(X CV onehotcoding)
    print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(Y_CV, predi
37
38
39
    predict_y = sig_clf.predict_proba(X_Test_onehotcoding)
40
    print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(Y_Test, predict_y, labe
41
42
    return alpha, best_alpha
```

1 alpha, best alpha = Logistic Regression Algo WithoutClassBalancing(X Train, Y Train, X CV, Y CV, X Test, Y Test)

С⇒

```
1 clf = SGDClassifier( alpha=alpha[best_alpha], penalty='12', loss='log', random_state=42)
2 predict_and_plot_confusionmatrix(X_Train_onehotCoding, Y_Train, X_CV_onehotcoding, Y_CV, clf)
```

# ▼ LINEAR SUPPORT VECTOR MACHINE

```
1 def LinearSVM_Algo(X_Train,Y_Train,X_CV,Y_CV,X_Test,Y_Test):
2
    alpha = [10 ** x for x in range(-6, 3)]
3
   cv_log_error=[]
6
   for i in alpha:
7
     print("for alpha =", i)
      clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='12', loss='hinge', random_state=42)
8
    clf.fit(X Train onehotCoding,Y_Train)
      sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
10
11
      sig_clf.fit(X_Train_onehotCoding, Y_Train)
12
      sig clf probs = sig clf.predict proba(X CV onehotcoding)
13
      cv_log_error.append(log_loss(Y_CV, sig_clf_probs, labels=clf.classes_, eps=1e-15))
      print("Log Loss :",log_loss(Y_CV, sig_clf_probs))
14
15
16
   fig, ax = plt.subplots()
17
    ax.plot(np.log10(alpha), cv_log_error,c='g')
18
    for i , txt in enumerate(np.round(cv log error,3)):
19
     ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv_log_error[i]))
20
    plt.grid()
21
   plt.xticks(np.log10(alpha))
   plt.title("Cross Validation Error for each alpha")
    plt.xlabel("Alpha i's")
24
    plt.ylabel("Error measure")
    plt.show()
26
27
   best_alpha = np.argmin(cv_log_error)
   clf = SGDClassifier(class weight='balanced', alpha=alpha[best_alpha], penalty='12', loss='hinge', random_state=4
   clf.fit(X_Train_onehotCoding, Y_Train)
    sig clf = CalibratedClassifierCV(clf, method="sigmoid")
30
31
    sig clf.fit(X Train onehotCoding, Y Train)
32
33
    predict_y = sig_clf.predict_proba(X_Train_onehotCoding)
    print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(Y_Train, predict_y, la
35
36
    predict_y = sig_clf.predict_proba(X_CV_onehotcoding)
37
    print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(Y_CV, predi
38
39
    predict_y = sig_clf.predict_proba(X_Test_onehotcoding)
40
    print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(Y_Test, predict_y, labe
41
42
   return alpha,best_alpha
```

```
1 alpha,best_alpha = LinearSVM_Algo(X_Train,Y_Train,X_CV,Y_CV,X_Test,Y_Test)
```

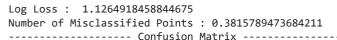
C→

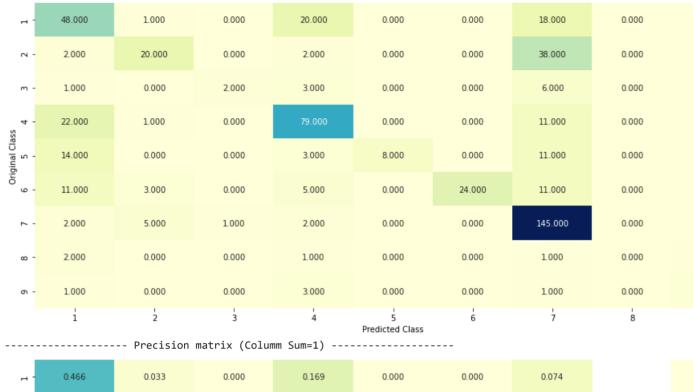
#### ▼ RANDOM FOREST CLASSIFIER

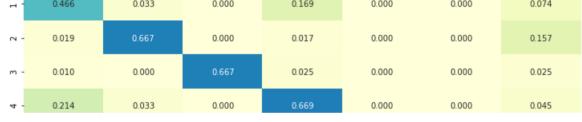
```
1 def Random Forest Algo(X Train, Y Train, X CV, Y CV, X Test, Y Test):
2 alpha = [100,200,500,1000,2000]
3 max_depth = [5, 10]
4 cv_log_error = []
5
   for i in alpha:
6
         for j in max_depth:
 7
             print("for n_estimators =", i,"and max depth = ", j)
8
            clf = RandomForestClassifier(n estimators=i, criterion='gini', max_depth=j, random state=42, n jobs=-1)
9
            clf.fit(X_Train_onehotCoding, Y_Train)
10
            sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
11
            sig_clf.fit(X_Train_onehotCoding, Y_Train)
            sig_clf_probs = sig_clf.predict_proba(X_CV_onehotcoding)
12
             cv_log_error.append(log_loss(Y_CV, sig_clf_probs, labels=clf.classes_, eps=1e-15))
13
14
             print("Log Loss :",log_loss(Y_CV, sig_clf_probs))
15
   best_alpha = np.argmin(cv_log_error)
   clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_alpha/2)]
16
    clf.fit(X_Train_onehotCoding, Y_Train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
18
19
    sig_clf.fit(X_Train_onehotCoding, Y_Train)
20
21
    predict_y = sig_clf.predict_proba(X_Train_onehotCoding)
    print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log loss is:",log_loss(Y_Train, pr
22
23
    predict y = sig clf.predict proba(X CV onehotcoding)
   print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validation log loss is:",log_loss(
    predict_y = sig_clf.predict_proba(X_Test_onehotcoding)
    print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss is:",log_loss(Y_Test, prec
    return alpha ,best alpha,max depth
```

₽

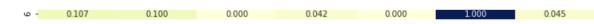
1 clf = RandomForestClassifier(n\_estimators=alpha[int(best\_alpha/2)], criterion='gini', max\_depth=max\_depth[int(best 2 predict\_and\_plot\_confusionmatrix(X\_Train\_onehotCoding, Y\_Train, X\_CV\_onehotcoding, Y\_CV, clf)







# ▼ TF-IDF FEATURIZATION TECHNIQUE



## ▼ TF-IDF on Gene Features

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
 2 gene_vectorizer=TfidfVectorizer()
3
 4 X_Train_gene_Feature_onehotencoding_tfidf=gene_vectorizer.fit_transform(X_Train["Gene"])
 5 print(" Train Gene Feature : " ,X_Train_gene_Feature_onehotencoding_tfidf.shape)
6
7 print("="*100)
 9 X_Test_gene_Feature_onehotencoding_tfidf=gene_vectorizer.transform(X_Test["Gene"])
10 print(" Test Gene Feature :" ,X_Test_gene_Feature_onehotencoding_tfidf.shape)
12 print("="*100)
13
14
15 X_CV_gene_Feature_onehotencoding_tfidf=gene_vectorizer.transform(X_CV["Gene"])
16 print(" CV Gene Feature :" ,X_CV_gene_Feature_onehotencoding_tfidf.shape)
17
18 print("="*100)
```

```
Train Gene Feature : (2124, 225)

Test Gene Feature : (665, 225)

CV Gene Feature : (532, 225)
```

# ▼ TF-IDF on Variation Features

```
1 variation_vectorizer=TfidfVectorizer()
2
3 X_Train_variation_Feature_onehotencoding_tfidf=variation_vectorizer.fit_transform(X_Train["Variation"])
4 print(" Train Variation Feature :" ,X_Train_variation_Feature_onehotencoding_tfidf.shape)
5
6 print("="*100)
7
8 X_Test_variation_Feature_onehotencoding_tfidf=variation_vectorizer.transform(X_Test["Variation"])
9 print(" Test Variation Feature :" ,X_Test_variation_Feature_onehotencoding_tfidf.shape)
10
11 print("="*100)
12
13
14 X_CV_variation_Feature_onehotencoding_tfidf=variation_vectorizer.transform(X_CV["Variation"])
15 print(" CV Variation Feature :" ,X_CV_variation_Feature_onehotencoding_tfidf.shape)
16
17 print("="*100)
```

# ▼ TF-IDF on Text Features

```
1 text_vectorizer = TfidfVectorizer(min_df=5,ngram_range=(1,4),max_features=3000)
2 X_Train_feature_onehotencoding_tfidf = text_vectorizer.fit_transform(X_Train['TEXT'])
3 # getting all the feature names (words)
```

```
1 from sklearn.preprocessing import normalize
 3 X_Train_feature_onehotencoding_tfidf = normalize(X_Train_feature_onehotencoding_tfidf, axis=0)
4 print("Train Text Feature :", X Train feature onehotencoding tfidf.shape)
 5
 6 print("="*100)
8 X_Test_text_feature_onehotencoding_tfidf = text_vectorizer.transform(X_Test['TEXT'])
9 print("Test Text Feature :", X Test text feature onehotencoding tfidf.shape)
10
11 print("="*100)
12
13 X_Test_text_feature_onehotencoding_tfidf = normalize(X_Test_text_feature_onehotencoding_tfidf, axis=0)
15 X_CV_text_feature_onehotencoding_tfidf = text_vectorizer.transform(X_CV['TEXT'])
16 print("CV Text Feature :", X_CV_text_feature_onehotencoding_tfidf.shape)
17
18 print("="*100)
20 V CV tout footune anabatamandina tfidf
                                            nammalian/V CV taut footoms anabatamanding third avia O
```

```
20 x_tv_text_reature_onenotencoaing_triar = normalize(x_tv_text_reature_onenotencoaing_triar, axis=0)
```

```
Train Text Feature : (2124, 3000)
 ______
 Test Text Feature: (665, 3000)
 _____
 CV Text Feature : (532, 3000)
 ______
```

# ▼ STACKING THE THREE TYPES OF FEATURES.

```
1 X_Train_gene_var_onehotencoding_tfidf = hstack((X_Train_gene_Feature_onehotencoding_tfidf,X_Train_variation_Featur
  2 X_Test_gene_var_onehotencoding_tfidf = hstack((X_Test_gene_Feature_onehotencoding_tfidf,X_Test_variation_Feature_
  3 X CV gene var onehotencoding tfidf = hstack((X CV gene Feature onehotencoding tfidf,X CV variation Feature onehote
  5 X Train onehotcoding tfidf = hstack((X Train gene var onehotencoding tfidf, X Train feature onehotencoding tfidf)).
  6 print("Train One Hot Encoding :", X Train onehotcoding tfidf.shape)
  7 print("="*100)
  8
  9 X Test onehotcoding tfidf = hstack((X Test gene var onehotencoding tfidf, X Test text feature onehotencoding tfidf)
10 print("Test One Hot Encoding :", X_Test_onehotcoding_tfidf.shape)
11 print("="*100)
12
\label{eq:cv_onehot} \begin{tabular}{ll} 13 & X\_CV\_onehotcoding\_tfidf = hstack((X\_CV\_gene\_var\_onehotencoding\_tfidf, X\_CV\_text\_feature\_onehotencoding\_tfidf)). tocs & A_CV\_onehotcoding\_tfidf = hstack((X_CV\_gene\_var\_onehotencoding\_tfidf, X_CV\_text\_feature\_onehotencoding\_tfidf)). \\ \begin{tabular}{ll} 13 & X\_CV\_onehotcoding\_tfidf = hstack((X_CV\_gene\_var\_onehotencoding\_tfidf, X_CV\_text\_feature\_onehotencoding\_tfidf)). \\ \begin{tabular}{ll} 13 & X\_CV\_onehotcoding\_tfidf = hstack((X_CV\_gene\_var\_onehotencoding\_tfidf, X_CV\_text\_feature\_onehotencoding\_tfidf)). \\ \begin{tabular}{ll} 13 & X\_CV\_onehotcoding\_tfidf = hstack((X_CV\_gene\_var\_onehotencoding\_tfidf, X_CV\_text\_feature\_onehotencoding\_tfidf)). \\ \begin{tabular}{ll} 13 & X\_CV\_onehotcoding\_tfidf, X_CV\_text\_feature\_onehotencoding\_tfidf)). \\ \begin{tabular}{ll} 13 & X\_CV\_onehotcoding\_tfidf, X_CV\_text\_feature\_onehotencoding\_tfidf). \\ \begin{tabular}{ll} 13 & X\_CV\_onehotcoding\_tfidf). \\ \begin{tabular}{ll}
14 print("CV One Hot Encoding :", X_CV onehotcoding tfidf.shape)
15 print("="*100)
 □ Train One Hot Encoding : (2124, 5198)
            ______
            Test One Hot Encoding : (665, 5198)
            ______
```

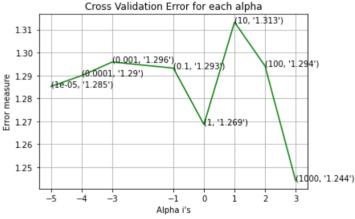
```
CV One Hot Encoding : (532, 5198)
```

# ▼ NAIVE BAYES ALGORITHM

```
1 alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
 2 cv_log_error_array = []
 3 for i in alpha:
     print("for alpha =", i)
      clf = MultinomialNB(alpha=i)
      clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
 6
      sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
      sig_clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
9
      sig clf probs = sig clf.predict proba(X CV onehotcoding tfidf)
      cv_log_error_array.append(log_loss(Y_CV, sig_clf_probs, labels=clf.classes_, eps=1e-15))
10
      # to avoid rounding error while multiplying probabilites we use log-probability estimates
12
      print("Log Loss :",log_loss(Y_CV, sig_clf_probs))
14 fig, ax = plt.subplots()
15 ax.plot(np.log10(alpha), cv_log_error_array,c='g')
16 for i, txt in enumerate(np.round(cv_log_error_array,3)):
17
      ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv_log_error_array[i]))
18 plt.grid()
19 plt.xticks(np.log10(alpha))
20 plt.title("Cross Validation Error for each alpha")
21 plt.xlabel("Alpha i's")
22 plt.ylabel("Error measure")
23 plt.show()
```

```
26 best_alpha = np.argmin(cv_log_error_array)
27 clf = MultinomialNB(alpha=alpha[best_alpha])
28 clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
29 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
30 sig_clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
31
32
33 predict_y = sig_clf.predict_proba(X_Train_onehotcoding_tfidf)
34 print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(Y_Train, predict_y, labe print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(Y_CV, predict_y predict_y = sig_clf.predict_proba(X_CV_onehotcoding_tfidf)
36 print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(Y_CV, predict_y predict_y = sig_clf.predict_proba(X_Test_onehotcoding_tfidf)
38 print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(Y_Test, predict_y, labels)
```

```
for alpha = 1e-05
 Log Loss: 1.2852242745511553
 for alpha = 0.0001
Log Loss: 1.2899281532020483
 for alpha = 0.001
 Log Loss: 1.2958675085806037
 for alpha = 0.1
 Log Loss: 1.293202683703183
 for alpha = 1
 Log Loss: 1.2686115338014161
 for alpha = 10
 Log Loss: 1.3132818674896714
 for alpha = 100
 Log Loss: 1.2940842269235393
 for alpha = 1000
 Log Loss: 1.2440531436451192
```



For values of best alpha = 1000 The train log loss is: 0.8125492088099504 For values of best alpha = 1000 The cross validation log loss is: 1.2440531436451192 For values of best alpha = 1000 The test log loss is: 1.2117067312528402

```
1 clf = MultinomialNB(alpha=alpha[best_alpha])
2 clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
3 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
4 sig_clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
5 sig_clf_probs = sig_clf.predict_proba(X_CV_onehotcoding_tfidf)
6
7 print("Log Loss :",log_loss(Y_CV, sig_clf_probs))
8
9 print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(X_CV_onehotcoding_tfidf) - Y_CV))/Y_CV.
10 plot_confusion_matrix(Y_CV, sig_clf.predict(X_CV_onehotcoding_tfidf.toarray()))
```

С→

Log Loss: 1.2000466448985152

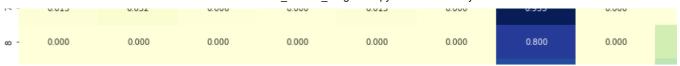
Number of missclassified point : 0.42857142857142855

----- Confusion Matrix ------40.000 1.000 0.000 12.000 1.000 31.000 0.000 2.000 2.000 16.000 0.000 1.000 0.000 0.000 43.000 0.000 1.000 0.000 2.000 3.000 2.000 0.000 4.000 0.000 17.000 1.000 60.000 8.000 26.000 0.000 0.000 1.000 Original Class 11.000 9.000 10.000 0.000 0.000 3.000 3.000 0.000 28.000 13.000 9 000 0.000 0.000 3 000 1 000 0.000 2.000 5.000 1.000 0.000 2.000 0.000 145.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 4.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 6.000 0.000 7 ż 1 6 8 Predicted Class Precision matrix (Columm Sum=1) -----0.146 0.077 0.030 0.043 0.000 0.110 0.025 0.696 0.000 0.012 0.000 0.000 0.153 0.012 0.000 0.037 0.077 0.000 0.014 0.308 0.093 0.210 0.043 0.000 0.030 Original Class 0.123 0.000 0.000 0.037 0.091 0.032 Ŋ 0.111 0.000 0.000 0.037 0.038 0.848 0.046 0.025 0.217 0.333 0.000 0.077 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.014 0.000 0.000 0.000 0.000 0.021 0.000 0.000 8 6 Predicted Class Recall matrix (Row sum=1) 0.011 0.000 0.138 0.023 0.011 0.356 0.000 0.032 0.258 0.000 0.016 0.000 0.000 0.694 0.000 0.250 0.083 0.000 0.167 0.167 0.000 0.333 0.000 0.150 0.009 0.000 0.071 0.009 0.230 0.000 Original Class 0.000 0.083 0.306 0.083 0.250 0.278 0.000 0.000 0.167 0.000 0.000 0.056 0.019 0.241 0.000

0.000

0.013

0.013



i realessa elass

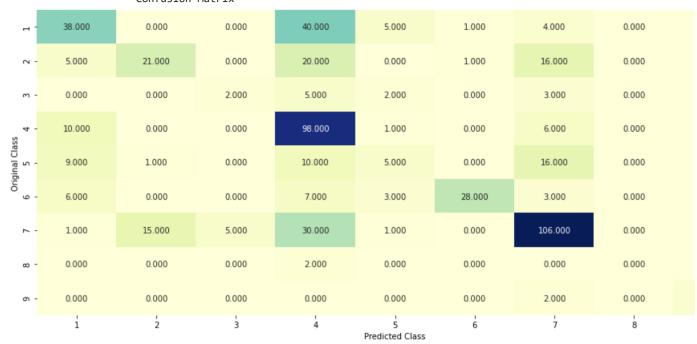
## ▼ K-NEAREST NEIGHBORS ALGORITHM

```
1 alpha = [5, 11, 15, 21, 31, 41, 51, 99]
2 cv_log_error_array = []
3 for i in alpha:
      print("for alpha =", i)
      clf = KNeighborsClassifier(n_neighbors=i)
      clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
6
7
      sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
      sig_clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
8
9
      sig_clf_probs = sig_clf.predict_proba(X_CV_onehotcoding_tfidf)
10
      cv log error array.append(log loss(Y CV, sig clf probs, labels=clf.classes , eps=1e-15))
      # to avoid rounding error while multiplying probabilites we use log-probability estimates
11
12
      print("Log Loss :",log loss(Y CV, sig clf probs))
13
14 fig, ax = plt.subplots()
15 ax.plot(alpha, cv log error array,c='g')
16 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]))
18 plt.grid()
19 plt.title("Cross Validation Error for each alpha")
20 plt.xlabel("Alpha i's")
21 plt.ylabel("Error measure")
22 plt.show()
23
24
25 best alpha = np.argmin(cv log error array)
26 clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
27 clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
28 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
29 sig_clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
31 predict_y = sig_clf.predict_proba(X_Train_onehotcoding_tfidf)
32 print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(Y_Train, predict_y, labe
33 predict_y = sig_clf.predict_proba(X_CV_onehotcoding_tfidf)
34 print('For values of best alpha = ', alpha[best alpha], "The cross validation log loss is:",log_loss(Y_CV, predict
35 predict_y = sig_clf.predict_proba(X_Test_onehotcoding_tfidf)
36 print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(Y_Test, predict_y, labels
```

Г⇒

```
for alpha = 5
   Log Loss : 1.260740792425099
   for alpha = 11
   Log Loss : 1.3267641338744462
   for alpha = 15
   Log Loss: 1.3556204961769065
   for alpha = 21
   Log Loss : 1.386854135603434
   for alpha = 31
   Log Loss: 1.4147264394638535
   for alpha = 41
   Log Loss : 1.4384943533717045
   for alpha = 51
   Log Loss: 1.4309811480079746
   for alpha = 99
   100 1000 1 4646110220471040
1 clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
2 predict_and_plot_confusionmatrix(X_Train_onehotcoding_tfidf, Y_Train, X_CV_onehotcoding_tfidf, Y_CV, clf)
```

С>



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

```
1 clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
2 clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
3 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
4 sig_clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
5
6 test_point_index = 1
7 predicted_cls = sig_clf.predict(X_Test_onehotcoding_tfidf[0].reshape(1,-1))
8 print("Predicted Class :", predicted_cls[0])
9 print("Actual Class :", Y_Test[test_point_index])
10 neighbors = clf.kneighbors(X_Test_onehotcoding_tfidf[test_point_index].reshape(1, -1), alpha[best_alpha])
11 print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to classes",Y_Train[neighbors[1][0]]
12 print("Fequency of nearest points :",Counter(Y_Train[neighbors[1][0]]))
```

```
Predicted Class : 7
   Actual Class : 4
```

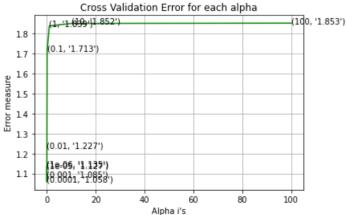
The 5 nearest neighbours of the test points belongs to classes [3 4 1 4 1] Fequency of nearest points : Counter({4: 2, 1: 2, 3: 1})

## LOGISTIC REGRESSION WITH CLASS BALANCING

```
1 alpha = [10 ** x for x in range(-6, 3)]
2 cv_log_error_array = []
3 for i in alpha:
      print("for alpha =", i)
5
      clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='12', loss='log', random_state=42)
6
      clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
7
      sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
      sig_clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
9
      sig_clf_probs = sig_clf.predict_proba(X_CV_onehotcoding_tfidf)
10
      cv_log_error_array.append(log_loss(Y_CV, sig_clf_probs, labels=clf.classes_, eps=1e-15))
      # to avoid rounding error while multiplying probabilites we use log-probability estimates
11
12
      print("Log Loss :",log_loss(Y_CV, sig_clf_probs))
13
14 fig, ax = plt.subplots()
15 ax.plot(alpha, cv_log_error_array,c='g')
```

```
16 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]))
18 plt.grid()
19 plt.title("Cross Validation Error for each alpha")
20 plt.xlabel("Alpha i's")
21 plt.ylabel("Error measure")
22 plt.show()
23
25 best_alpha = np.argmin(cv_log_error_array)
26 clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='12', loss='log', random_state=42)
27 clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
28 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
29 sig_clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
30
31 predict_y = sig_clf.predict_proba(X_Train_onehotcoding_tfidf)
32 print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(Y_Train, predict_y, labe
33 predict y = sig clf.predict proba(X CV onehotcoding tfidf)
34 print('For values of best alpha = ', alpha[best alpha], "The cross validation log loss is:",log_loss(Y_CV, predict
35 predict_y = sig_clf.predict_proba(X_Test_onehotcoding_tfidf)
36 print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(Y_Test, predict_y, labels
```

```
r→ for alpha = 1e-06
    Log Loss: 1.1349239214992324
    for alpha = 1e-05
    Log Loss: 1.1270021942239883
    for alpha = 0.0001
   Log Loss: 1.0584224477093065
    for alpha = 0.001
   Log Loss: 1.0848800628677302
    for alpha = 0.01
    Log Loss: 1.2267667992253835
    for alpha = 0.1
    Log Loss: 1.7127788552336967
    for alpha = 1
    Log Loss : 1.8391919496093834
    for alpha = 10
    Log Loss: 1.85154108049996
    for alpha = 100
    Log Loss: 1.8529278523272878
```



```
For values of best alpha = 0.0001 The train log loss is: 0.36081440897296463

For values of best alpha = 0.0001 The cross validation log loss is: 1.0584224477093065

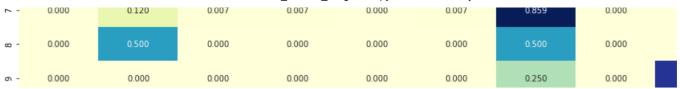
For values of best alpha = 0.0001 The test log loss is: 1.0564367265738994
```

```
1 clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
2 predict_and_plot_confusionmatrix(X_Train_onehotcoding_tfidf, Y_Train, X_CV_onehotcoding_tfidf, Y_CV, clf)
```

 $\Box$ 

Log Loss: 1.0584224477093065 Number of Misclassified Points: 0.34398496240601506

------ Confusion Matrix -----4.000 0.000 22.000 3.000 2.000 0.000 6.000 0.000 38.000 0.000 2.000 4.000 0.000 36.000 0.000 0.000 1.000 3.000 4.000 1.000 1.000 6.000 0.000 11.000 0.000 5.000 0.000 0.000 3.000 1.000 Original Class 10.000 6.000 2.000 1.000 9.000 1.000 7.000 0.000 3.000 27.000 6.000 12 000 1 000 0.000 0.000 0.000 0.000 17.000 1.000 1.000 0.000 1.000 122.000 0.000 0.000 1.000 0.000 0.000 0.000 0.000 1.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 1.000 0.000 ż 1 8 Predicted Class ----- Precision matrix (Columm Sum=1) -----0.179 0.250 0.088 0.011 0.062 0.000 0.194 0.000 0.000 0.016 0.167 0.000 0.033 0.000 0.016 0.042 0.029 0.032 0.118 0.000 0.000 0.125 0.029 0.027 Original Class 0.065 0.031 0.200 0.073 0.417 0.029 0.038 'n 0.129 0.016 0.000 0.024 0.000 0.032 0.000 0.266 0.200 0.008 0.000 0.029 0.000 0.016 0.000 0.000 0.000 0.000 0.005 0.000 0.000 0.000 0.000 0.000 0.000 0.005 Predicted Class Recall matrix (Row sum=1) -----0.040 0.000 0.218 0.059 0.030 0.020 0.000 0.000 0.000 0.025 0.050 0.000 0.000 0.250 0.375 0.000 0.062 0.188 0.062 0.062 0.000 0.108 0.000 0.000 0.804 0.029 0.010 0.049 0.000 Original Class 0.167 0.056 0.028 0.250 0.278 0.028 0.194 0.000 0.245 0.020 0.000 0.061 0.000 0.122 0.000



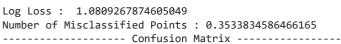
## ▼ LOGISTIC REGRESSION WITHOUT CLASS BALANCING

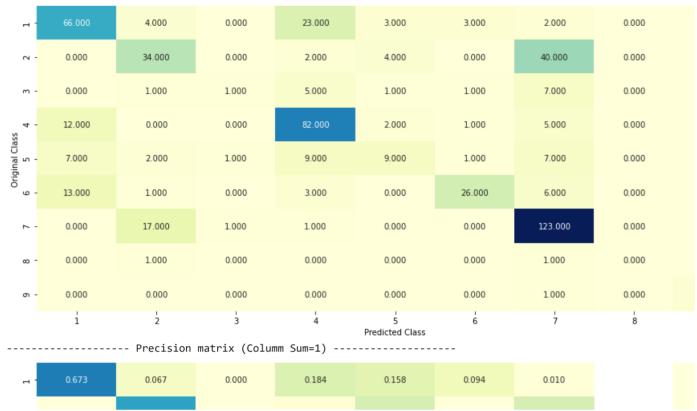
```
1 alpha = [10 ** x for x in range(-6, 3)]
2 cv_log_error_array = []
3 for i in alpha:
      print("for alpha =", i)
      clf = SGDClassifier(alpha=i, penalty='12', loss='log', random_state=42)
5
      clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
6
      sig clf = CalibratedClassifierCV(clf, method="sigmoid")
7
      sig clf.fit(X Train onehotcoding tfidf, Y Train)
8
      sig_clf_probs = sig_clf.predict_proba(X_CV_onehotcoding_tfidf)
9
      cv log error array.append(log loss(Y CV, sig clf probs, labels=clf.classes , eps=1e-15))
10
11
      # to avoid rounding error while multiplying probabilites we use log-probability estimates
12
      print("Log Loss :",log_loss(Y_CV, sig_clf_probs))
13
14 fig, ax = plt.subplots()
15 ax.plot(alpha, cv_log_error_array,c='g')
16 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]))
17
18 plt.grid()
19 plt.title("Cross Validation Error for each alpha")
20 plt.xlabel("Alpha i's")
21 plt.ylabel("Error measure")
22 plt.show()
23
24
25 best_alpha = np.argmin(cv_log_error_array)
26 clf = SGDClassifier(alpha=alpha[best alpha], penalty='12', loss='log', random_state=42)
27 clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
28 sig clf = CalibratedClassifierCV(clf, method="sigmoid")
29 sig_clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
31 predict_y = sig_clf.predict_proba(X_Train_onehotcoding_tfidf)
32 print('For values of best alpha = ', alpha[best alpha], "The train log loss is:",log loss(Y Train, predict y, labe
33 predict_y = sig_clf.predict_proba(X_CV_onehotcoding_tfidf)
34 print('For values of best alpha = ', alpha[best alpha], "The cross validation log loss is:",log_loss(Y_CV, predict
35 predict y = sig clf.predict_proba(X_Test_onehotcoding_tfidf)
36 print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(Y_Test, predict_y, labels
```

```
for alpha = 1e-06
Log Loss : 1.1386593648101406
for alpha = 1e-05
Log Loss: 1.1280770426263258
for alpha = 0.0001
Log Loss: 1.0809267874605049
for alpha = 0.001
Log Loss : 1.1599651555818726
for alpha = 0.01
Log Loss: 1.4711079574295722
for alpha = 0.1
Log Loss : 1.7324620156684516
for alpha = 1
Log Loss: 1.862433088524242
for alpha = 10
Log Loss : 1.8771853566934507
for alpha = 100
Log Loss: 1.8788362002968801
             Cross Validation Error for each alpha
```

```
1 clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
2 predict_and_plot_confusionmatrix(X_Train_onehotcoding_tfidf, Y_Train, X_CV_onehotcoding_tfidf, Y_CV, clf)
```

 $\Box$ 





## **▼ LINEAR SUPPORT VECTOR MACHINE**

```
1 alpha = [10 ** x for x in range(-6, 3)]
2 cv_log_error_array = []
3 for i in alpha:
      print("for alpha =", i)
4
5
      clf = SGDClassifier(class_weight='balanced',alpha=i, penalty='12', loss='hinge', random_state=42)
6
      clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
      sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
7
      sig_clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
8
9
      sig_clf_probs = sig_clf.predict_proba(X_CV_onehotcoding_tfidf)
      cv_log_error_array.append(log_loss(Y_CV, sig_clf_probs, labels=clf.classes_, eps=1e-15))
10
11
      # to avoid rounding error while multiplying probabilites we use log-probability estimates
12
      print("Log Loss :",log_loss(Y_CV, sig_clf_probs))
13
14 fig, ax = plt.subplots()
15 ax.plot(alpha, cv_log_error_array,c='g')
16 for i, txt in enumerate(np.round(cv_log_error_array,3)):
17
      ax.annotate((alpha[i],str(txt)), (alpha[i],cv_log_error_array[i]))
18 plt.grid()
19 plt.title("Cross Validation Error for each alpha")
20 plt.xlabel("Alpha i's")
21 plt.ylabel("Error measure")
22 plt.show()
23
24
25 best_alpha = np.argmin(cv_log_error_array)
26 clf = SGDClassifier(class_weight='balanced',alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
27 clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
28 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
29 sig_clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
```

```
31 predict_y = sig_clf.predict_proba(X_Train_onehotcoding_tfidf)
32 print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(Y_Train, predict_y, labe
33 predict_y = sig_clf.predict_proba(X_CV_onehotcoding_tfidf)
34 print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(Y_CV, predict
35 predict_y = sig_clf.predict_proba(X_Test_onehotcoding_tfidf)
36 print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(Y_Test, predict_y, labels
```

С→

```
1 clf = SGDClassifier(alpha=alpha[best_alpha], penalty='12', loss='hinge', random_state=42,class_weight='balanced')
2 predict_and_plot_confusionmatrix(X_Train_onehotcoding_tfidf, Y_Train,X_CV_onehotcoding_tfidf,Y_CV, clf)
```

## ▼ RANDOM FOREST CLASSIFIER

```
1 alpha = [100,200,500,1000,2000]
 2 max depth = [5, 10]
3 cv_log_error_array = []
4 for i in alpha:
      for j in max_depth:
           print("for n_estimators =", i,"and max depth = ", j)
6
7
           clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42, n_jobs=-1)
8
           clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
9
           sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
           sig_clf.fit(X_Train_onehotcoding_tfidf, Y_Train)
10
11
           sig_clf_probs = sig_clf.predict_proba(X_CV_onehotcoding_tfidf)
           cv_log_error_array.append(log_loss(Y_CV, sig_clf_probs, labels=clf.classes_, eps=1e-15))
12
           print("Log Loss :",log_loss(Y_CV, sig_clf_probs))
13
14
15 '''fig, ax = plt.subplots()
16 features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
17 ax.plot(features, cv_log_error_array,c='g')
18 for i, txt in enumerate(np.round(cv_log_error_array,3)):
      ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)), (features[i],cv_log_error_array[i]))
19
20 plt.grid()
21 plt.title("Cross Validation Error for each alpha")
22 plt.xlabel("Alpha i's")
23 plt.ylabel("Error measure")
24 plt.show()
25 '''
27 host almba - nn anamin(cu log annon annou)
```

```
28 clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_2 clf.fit(X_Train_onehotcoding_tfidf, Y_Train))

30 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")

31 sig_clf.fit(X_Train_onehotcoding_tfidf, Y_Train)

32

33 predict_y = sig_clf.predict_proba(X_Train_onehotcoding_tfidf)

34 print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log loss is:",log_loss(Y_Train, predict_y = sig_clf.predict_proba(X_CV_onehotcoding_tfidf)

36 print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validation log loss is:",log_loss(Y_Train_onehotcoding_tfidf)

37 predict_y = sig_clf.predict_proba(X_Test_onehotcoding_tfidf)

38 print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss is:",log_loss(Y_Test, predict_onehotcoding_tfidf)

38 print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss is:",log_loss(Y_Test, predict_onehotcoding_tfidf)

39 print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss is:",log_loss(Y_Test, predict_onehotcoding_tfidf)

30 sig_clf = RandomForestClassifier(v(clf, method="sigmoid"))

31 sig_clf.fit(X_Train_onehotcoding_tfidf)

32 predict_y = sig_clf.predict_proba(X_Test_onehotcoding_tfidf)

33 predict_y = sig_clf.predict_proba(X_Test_onehotcoding_tfidf)

34 print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss is:",log_loss(Y_Test_onehotcoding_tfidf)

35 print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss is:",log_loss(Y_Test_onehotcoding_tfidf)
```



## **▼ STACK THE MODELS**

```
1 clf1 = SGDClassifier(alpha=0.001, penalty='12', loss='log', class_weight='balanced', random_state=0)
2 clf1.fit(X_Train_onehotcoding_tfidf, Y_Train)
3 sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

1 clf2 = SGDClassifier(alpha=1, penalty='12', loss='hinge', class_weight='balanced', random_state=0)
2 clf2.fit(X_Train_onehotcoding_tfidf, Y_Train)
3 sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")

1 clf3 = MultinomialNB(alpha=0.001)
2 clf3.fit(X_Train_onehotcoding_tfidf, Y_Train)
3 sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")

1 sig_clf1.fit(X_Train_onehotcoding_tfidf, Y_Train)
2 print("Logistic Regression : Log Loss: %0.2f" % (log_loss(Y_CV, sig_clf1.predict_proba(X_CV_onehotcoding_tfidf)))
3 sig_clf2.fit(X_Train_onehotcoding_tfidf, Y_Train)
4 print("Support vector machines : Log Loss: %0.2f" % (log_loss(Y_CV, sig_clf2.predict_proba(X_CV_onehotcoding_tfidf))
```

С→

3 print("-"\*50)

```
1 alpha = [0.0001,0.001,0.01,0.1,1,10]
2 best_alpha = 999
3 for i in alpha:
```

2 print("Naive Bayes : Log Loss: %0.2f" % (log\_loss(Y\_CV, sig\_clf3.predict\_proba(X\_CV\_onehotcoding\_tfidf))))

1 sig clf3.fit(X Train onehotcoding tfidf, Y Train)

С→

```
1 lr = LogisticRegression(C=0.1)
2 sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas=True)
3 sclf.fit(X_Train_onehotcoding_tfidf, Y_Train)
4
5 log_error = log_loss(Y_Train, sclf.predict_proba(X_Train_onehotcoding_tfidf))
6 print("Log loss (train) on the stacking classifier :",log_error)
7
8 log_error = log_loss(Y_CV, sclf.predict_proba(X_CV_onehotcoding_tfidf))
9 print("Log loss (CV) on the stacking classifier :",log_error)
10
11 log_error = log_loss(Y_Test, sclf.predict_proba(X_Test_onehotcoding_tfidf))
12 print("Log loss (test) on the stacking classifier :",log_error)
13
14 print("Number of missclassified point :", np.count_nonzero((sclf.predict(X_Test_onehotcoding_tfidf))
15 plot_confusion_matrix(test_y=Y_Test, predict_y=sclf.predict(X_Test_onehotcoding_tfidf))
```

# ▼ LOGISTIC REGRESSION WITH COUNTVECTORIZER USING BIGRAM

# ▼ CountVectorizer on Gene Features

```
1 from sklearn.feature_extraction.text import CountVectorizer
2
3 count_vectorizer_bigram=CountVectorizer(ngram_range=(1,2))
4
5 X_Train_gene_Feature_onehotencoding_bigram=count_vectorizer_bigram.fit_transform(X_Train["Gene"])
6 print(" Train Gene Feature :" ,X_Train_gene_Feature_onehotencoding_bigram.shape)
7
8 print("="*100)
9
10 X_Test_gene_Feature_onehotencoding_bigram=count_vectorizer_bigram.transform(X_Test["Gene"])
11 print(" Test Gene Feature :" ,X_Test_gene_Feature_onehotencoding_bigram.shape)
12
13 print("="*100)
14
```

```
15
16 X_CV_gene_Feature_onehotencoding_bigram=count_vectorizer_bigram.transform(X_CV["Gene"])
17 print(" CV Gene Feature :" ,X_CV_gene_Feature_onehotencoding_bigram.shape)
18
19 print("="*100)
```

₽

## CountVectorizer on Variation Features

```
1 variation_vectorizer_bigram=CountVectorizer(ngram_range=(1,2))
2
3 X_Train_variation_Feature_onehotencoding_bigram=variation_vectorizer_bigram.fit_transform(X_Train["Variation"])
4 print(" Train Variation Feature :" ,X_Train_variation_Feature_onehotencoding_bigram.shape)
5
6 print("="*100)
7
8 X_Test_variation_Feature_onehotencoding_bigram=variation_vectorizer_bigram.transform(X_Test["Variation"])
9 print(" Test Variation Feature :" ,X_Test_variation_Feature_onehotencoding_bigram.shape)
10
11 print("="*100)
12
13
14 X_CV_variation_Feature_onehotencoding_bigram=variation_vectorizer_bigram.transform(X_CV["Variation"])
15 print(" CV Variation Feature :" ,X_CV_variation_Feature_onehotencoding_bigram.shape)
16
17 print("="*100)
```

С⇒

# ▼ CountVectorizer on Text Features

```
1 from sklearn.preprocessing import normalize
2
3 text_vectorizer_bigram = CountVectorizer(min_df=3,ngram_range=(1,2))
4 X_Train_feature_onehotencoding_bigram = text_vectorizer_bigram.fit_transform(X_Train['TEXT'])
5
6
7 X_Train_feature_onehotencoding_bigram = normalize(X_Train_feature_onehotencoding_bigram, axis=0)
8 print("Train Text Feature :", X_Train_feature_onehotencoding_bigram.shape)
9
10 print("="*100)
11
12 X_Test_text_feature_onehotencoding_bigram = text_vectorizer_bigram.transform(X_Test['TEXT'])
13 print("Test Text Feature :", X_Test_text_feature_onehotencoding_bigram.shape)
14
15 print("="*100)
16
17 X_Test_text_feature_onehotencoding_bigram = normalize(X_Test_text_feature_onehotencoding_bigram, axis=0)
18
```

```
19 X_CV_text_feature_onehotencoding_bigram = text_vectorizer_bigram.transform(X_CV['TEXT'])
20 print("CV Text Feature :", X_CV_text_feature_onehotencoding_bigram.shape)
21
22 print("="*100)
23
24 X_CV_text_feature_onehotencoding_bigram = normalize(X_CV_text_feature_onehotencoding_bigram, axis=0)
```

С→

#### ▼ STACKING THE 3 TYPES OF FEATURES.

```
1 X_Train_gene_var_onehotencoding_bigram = hstack((X_Train_gene_Feature_onehotencoding_bigram,X_Train_variation_Feat
2 X_Test_gene_var_onehotencoding_bigram = hstack((X_Test_gene_Feature_onehotencoding_bigram,X_Test_variation_Featur
3 X_CV_gene_var_onehotencoding_bigram = hstack((X_CV_gene_Feature_onehotencoding_bigram,X_CV_variation_Feature_onehot
4
5 X_Train_onehotcoding_bigram = hstack((X_Train_gene_var_onehotencoding_bigram,X_Train_feature_onehotencoding_bigram
6 print("Train One Hot Encoding :", X_Train_onehotcoding_bigram.shape)
7 print("="*100)
8
9 X_Test_onehotcoding_bigram = hstack((X_Test_gene_var_onehotencoding_bigram,X_Test_text_feature_onehotencoding_bigr
10 print("Test One Hot Encoding :", X_Test_onehotcoding_bigram.shape)
11 print("="*100)
12
13 X_CV_onehotcoding_bigram = hstack((X_CV_gene_var_onehotencoding_bigram,X_CV_text_feature_onehotencoding_bigram)).t
14 print("CV One Hot Encoding :", X_CV_onehotcoding_bigram.shape)
15 print("="*100)
```

₽

#### ▼ LOGISTIC REGRESSION WITH CLASS BALANCING

```
1 alpha = [10 ** x for x in range(-6, 3)]
 2 cv_log_error_array = []
 3 for i in alpha:
      print("for alpha =", i)
      clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='12', loss='log', random_state=42)
 5
 6
      clf.fit(X_Train_onehotcoding_bigram, Y_Train)
 7
      sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
      sig_clf.fit(X_Train_onehotcoding_bigram, Y_Train)
8
9
      sig_clf_probs = sig_clf.predict_proba(X_CV_onehotcoding_bigram)
10
      cv_log_error_array.append(log_loss(Y_CV, sig_clf_probs, labels=clf.classes_, eps=1e-15))
11
      # to avoid rounding error while multiplying probabilites we use log-probability estimates
12
      print("Log Loss :",log_loss(Y_CV, sig_clf_probs))
14 fig, ax = plt.subplots()
15 ax.plot(alpha, cv_log_error_array,c='g')
16 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]))
17
18 plt.grid()
19 plt.title("Cross Validation Error for each alpha")
```

```
20 plt.xlabel("Alpha i's")
21 plt.ylabel("Error measure")
22 plt.show()
23
24
25 best_alpha = np.argmin(cv_log_error_array)
26 clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='12', loss='log', random_state=42)
27 clf.fit(X_Train_onehotcoding_bigram, Y_Train)
28 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
29 sig_clf.fit(X_Train_onehotcoding_bigram, Y_Train)
30
31 predict y = sig clf.predict_proba(X_Train_onehotcoding_bigram)
32 print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(Y_Train, predict_y, labe
33 predict_y = sig_clf.predict_proba(X_CV_onehotcoding_bigram)
34 print('For values of best alpha = ', alpha[best alpha], "The cross validation log loss is:",log_loss(Y_CV, predict
35 predict y = sig clf.predict proba(X Test onehotcoding bigram)
36 print('For values of best alpha = ', alpha[best alpha], "The test log loss is:",log loss(Y Test, predict y, labels
```

Г⇒

```
1 clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='12', loss='log', random_state=42)
2 predict_and_plot_confusionmatrix(X_Train_onehotcoding_bigram, Y_Train, X_CV_onehotcoding_bigram, Y_CV, clf)
```

### ▼ LOGISTIC REGRESSION WITHOUT CLASS BALANCING

```
1 alpha = [10 ** x for x in range(-6, 3)]
2 cv_log_error_array = []
3 for i in alpha:
      print("for alpha =", i)
      clf = SGDClassifier(alpha=i, penalty='12', loss='log', random_state=42)
      clf.fit(X Train onehotcoding bigram, Y Train)
7
      sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
8
      sig_clf.fit(X_Train_onehotcoding_bigram, Y_Train)
      sig clf probs = sig clf.predict proba(X CV onehotcoding bigram)
9
    cv_log_error_array.append(log_loss(Y_CV, sig_clf_probs, labels=clf.classes_, eps=1e-15))
10
11
      # to avoid rounding error while multiplying probabilites we use log-probability estimates
12
      print("Log Loss :",log loss(Y CV, sig clf probs))
13
14 fig, ax = plt.subplots()
15 ax.plot(alpha, cv log error array,c='g')
16 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]))
18 plt.grid()
19 plt.title("Cross Validation Error for each alpha")
20 plt.xlabel("Alpha i's")
21 plt.ylabel("Error measure")
22 plt.show()
23
24
25 best alpha = np.argmin(cv log error array)
26 clf = SGDClassifier(alpha=alpha[best_alpha], penalty='12', loss='log', random_state=42)
27 clf.fit(X_Train_onehotcoding_bigram, Y_Train)
28 sig clf = CalibratedClassifierCV(clf, method="sigmoid")
29 sig_clf.fit(X_Train_onehotcoding_bigram, Y_Train)
31 predict_y = sig_clf.predict_proba(X_Train_onehotcoding_bigram)
32 print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(Y_Train, predict_y, labe
33 predict_y = sig_clf.predict_proba(X_CV_onehotcoding_bigram)
34 print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss(Y_CV, predict
35 predict_y = sig_clf.predict_proba(X_Test_onehotcoding_bigram)
36 print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(Y_Test, predict_y, labels
```

 $\Box$ 

```
1 clf = SGDClassifier( alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
2 predict_and_plot_confusionmatrix(X_Train_onehotcoding_bigram, Y_Train, X_CV_onehotcoding_bigram, Y_CV, clf)
```