Comparing the Performance of AdaBoost, Gradient Boosting, and Logistic Regression for Multiclass Imbalanced Data

Sharmeen Binti Syazwan Lai, Nur Huda Nabihan Binti Md Shahri*, Mazni Binti Mohamad, Hezlin Aryani Binti Abdul Rahman, Adzhar Bin Rambli

EEE598 students: Pooja Yadav, Praveen Paidi, Furqan Al Idrees Dastagir

Our project is based on [1], comparing the performance of different methods while dealing with multiclass imbalanced datasets.

Abstract

An imbalanced data problem occurs in the absence of a good class distribution between classes. One or more majority classes makes up almost the whole data with one or more minority classes [2]. This imbalance in data can cause the classifier to be biased to the majority classes if it believes that the training set is balanced. The aim of our project is to understand the problem of imbalanced dataset and find the best method for dealing with multiclass imbalanced datasets among AdaBoost, Gradient boosting, and Logistic Regression following work in [1]. We obtained gradient boosting performed best among the tested models for highly imbalanced data.

Introduction

In a real-world scenario, we often encounter the imbalance data issue. There are numerous applications where we want to use ML models for anomaly detection, like defect detection in a fabricated chip in VLSI design. The ML model uses defective chips and good chips to learn and predict defects in wafers for future production. The data used for this prediction is highly imbalanced as number of defective chips (per defect) are 5 -10% of total chips. Other cases like perception issues in autonomous vehicles, using near field stereo(recorded) readings of ground path [3]. In this case training data is imbalanced, as observations of ground path are far more than known obstacles and unknown obstacles. The minority class in such scenarios always contains vital information that is crucial for model training and prediction. If we use standard classification algorithms that are designed for the balanced training set, the obtained classifier is biased because classifier is always more intended towards majority class and may not be useful for the application. Therefore, it is crucial to find the best way to deal with imbalanced datasets and emphasize finding a solution for an accurate prediction of minority cases.

[4] [5] [6] [7] [8] accentuate on understanding imbalanced data and finding a solution for an accurate prediction of the minority cases. These works use different methods of oversampling, under sampling, bagging, boosting, Support Vector Machine (SVM) and k-Nearest Neighbors (k-NN) approach to deal with imbalanced data. The data sampling techniques used in previous works may affect the accuracy of prediction by inoculating bias and may miss crucial data information if all the samples in the data are essential. [1] investigates performance comparison of best methods to deal with multiclass imbalanced dataset.

Problem setup

In our project we are comparing performance of boosting algorithms AdaBoost, Gradient Boosting and Logistic regression for multiclass imbalanced datasets Glass, Ecoli, and Wifi Localization extracted from UCI Repository [9]. Number of classes in our dataset are in the range of $[4, 8]$ with minority percentage of observation belonging to each class varying from 1%, 4%, 8%, 12% and more. We also used one dataset (wifi localization) where the dataset is balanced across all the four classes. We will compare the results of these three methods using five performance measures, namely sensitivity, specificity, precision, F1-score, and g-mean. In the end, based on our comparisons we will obtain the best method for each data imbalance ratio.

We used Python 3.10 and ASU High Performance Computing – Agave Cluster to train our model using above algorithms and to do predictions.

Main Contributions

We implemented AdaBoost, Gradient Boosting and Logistic regression algorithms in project. The original paper implemented Adaptive Boosting, Extreme Gradient Boosting (XGBoost) and Logistic Regression methods but we wanted to understand and implement gradient boosting in our project before jumping to extreme gradient boosting.

A. Adaptive Boosting (AdaBoost)

AdaBoost is the boosting technique introduced by Freund and Schapire [10]. AdaBoost uses weak classifiers to build a stronger and more stable classifier. It starts with equal weight for each observation and then based on prediction done by weak learner, greater weights are assigned to observations that are misclassified, iteratively. In final step, output of the weak learners is combined to make a strong learner.

Pseudo Algorithm: Adaboost

Initialize each weight, $D_1(i) = 1/k$.

For t = 1…T: (Where T is hyper parameter, based on amount of accuracy required).

a) Decision tree classifier fit based on index calculation.
b) Calculate total error $\varepsilon_t = \sum D_t(i) * \delta_i$ where δ_i is -1 if $h_t(x_i) = y_i$ and 0 otherwise.
c) Then calculate the performance of the stump. $\alpha_t = 0.5 * \ln(\frac{1-\varepsilon_t}{\varepsilon_t})$
d) Zt = normalization constant such that $\sum D_t(i) = 1$ and update weights as $(D_{t+1}(i)) = D_t(i) * e^{\pm\alpha_t} / Z_t$
e) $h_t(x_i)$ will be updated in each iteration picking misclassified more no. of times and some correct ones.

For Binary class classification, the predicted classifier is given by $H(x)=\text{sign}\left(\sum_{t=1}^{T} \alpha_t . h_t(x_i)\right)$

For Multiclass classification, $h_t(x_i)$ is one hot encoded and the predicted classifier is given by

$H(x)=\text{argmax}\left(\sum_{t=1}^{T} \alpha_t . h_t(x_i)\right)$

B. Gradient Boosting

Grading boosting introduced by Jerome Friedman [11], like other boosting methods combines weak "learners" into a single strong learner in an iterative fashion. Gradient boosting looks at the difference between its current approximation, and the known correct target vector, which is called the residual loss. It then trains a weak model that maps the feature vector to that residual loss vector with the objective of minimizing the loss. We used decision trees as the weak learners in our model and used square cost function to minimize each classifier loss. Adding a loss predicted by a weak model to an existing model's approximation pushes the model toward the correct target. The process is repeated until the loss is reduced below a certain threshold or a specified number of trees is reached.

Pseudo Algorithm: Gradient Boosting

1. Data Inputs: $\{(x_i, y_i)\}_{i=1}^{n}$
2. Initial parameter: $\hat{\theta} = \widehat{\theta_0}$
3. For each iteration t: 1 to M, repeat:
   a) $\nabla L_\theta(\hat{\theta}) = \left[\frac{\partial L(y, f(x,\theta))}{\partial \theta}\right]_{\theta=\hat{\theta}}$
   b) $\hat{\theta}_t \leftarrow -\alpha \nabla L_\theta(\hat{\theta})$
   c) $\hat{\theta} \leftarrow \hat{\theta} + \hat{\theta}_t$
   d) $\hat{f}(x) = f(x, \hat{\theta})$

   where, $f(x, \theta), \theta \in \mathbb{R}^d$

   L_θ is a differentiable loss function

   $\hat{f}(x) = \underset{f(x)}{\text{argmin}}\, \mathbb{E}_{x,y}[L(y, f(x))]$

C. Logistic Regresssion

Logistic Regression is applied when the response variable is binary. Theory behind logistic regression is very similar to linear regression [12]. Logistic regression model [13] do not fit a straight line, instead fit observations on a sigmoid curve. Just like in the sigmoid curve, y-axis in our model range from 0 to 1. This helps us classify our data into two different categories. This model made up of one or more independent variable (continuous or categorical) and one categorical dependent variable.

If 'p' is the probability of success which is transformed to form a linear equation of an exponential function form.

$p = \exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots\ldots + \beta_n x_n)$

here ‘p’ is expressed in terms of odds,

$Odds = \frac{p}{(1-p)} = \exp(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots\ldots + \beta_n x_n)$

Taking natural log on both sides we get,

$\ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots\ldots + \beta_n x_n$

where,

β_0 = constant

β_n = Coefficient of the independent variable x_n

Experimental Results

Dataset description

[1] performed a Monte Carlo simulation study to generate imbalanced datasets having different imbalance ratios. Due to the setup complexity, we didn’t perform Monte Carlo simulation. Instead, we implemented our algorithms on real dataset from UCI Repository [9] and studied their behavior at the granularity of each class.

We collected the dataset from UCI Repository [9] and pre-processed the data. The ecoli dataset has 8 classes with class distribution as {‘cp’,’cm’,’pp’,’imU’,’om’,’omL’,’imL’,’imS’}, wifi dataset consists of four class distributions {1,2,3,4}, glass dataset has 6 classes {1,2,3,4,6,7}. These data and names of the classes are collected and made comma separated files(csv) with appropriate mapping. From the obtained files, we performed statistical analysis to gain the insight of the data. We obtained the total number of classes for each dataset and number of observations for each class. We had number of classes in the range of $[4, 8]$ with minority percentage of observation belonging to each class being less than 1% as shown in Fig.1. The imbalance ratio in real dataset varied from 1%, 4%, 8%, 12% as shown in Fig.2. We also used one real dataset [wifi localization] where the dataset is balanced across all the four classes as shown in Fig.3.

Fig.1. Ecoli dataset

Fig.2. Glass dataset

Fig.3. Wifi dataset

ML model

We started with generating synthetic dataset using the inbuilt module of sklearn library – make_classification and generated X $\in \mathbb{R}^{n \times d}$ $where\ n = 1000, d = 20$ and $y \in [-1, 1]$ for each observation. We implemented all the three algorithms i.e., Adaptive Boosting, Gradient Boosting, and Logistic Regression to perform binary classification and used the above synthetic data to validate the functionality and performance.

Later, we extend these algorithms to perform multi-class classification with number of classes $\in N\ and\ n \geq 2\ i.e.\ 2, 3, 4, 5, 6 \ldots.$.

For **Adaboost**, we used DecisionTreeClassifier from sklearn.tree for tree formation and predicted the classifiers for training and test samples. For each weak leaner, performance (α) and error were calculated. For each classified records, we updated the new weights with performance value being negative and positive for correctly classified and misclassified respectively. The learning rate(lr) is hyper parameter and 0.5 is used for this. This was chosen to train the data bit slower by decreasing contribution of each classifier.

New sample weight = sample weight $* e^{\pm lr(\alpha)}$

The number of estimators is hyper parameter, we did not focus on tuning the hyper parameters much here. We have chosen max depth of tree as neither underfitting nor overfitting by the graphs. The predicted values of each weak base learner are collected and cumulated to form strong learner. We used one hot encoding for prediction of final prediction value and performance metrices reported in Table 1. The training loss with n-estimators for all classes is plotted with 3 datasets in Fig.4, Fig.5 and Fig.6.

Fig.4. Loss plots for Ecoli dataset

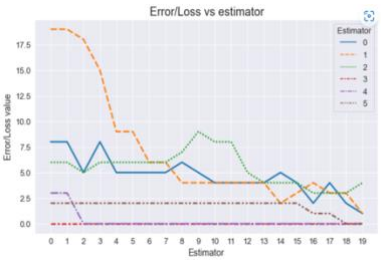

Fig.5. Loss plots for Glass dataset

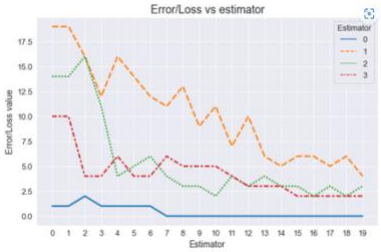

Fig.6. Loss plots for Wifi dataset

For **Gradient Boosting**, we used the One-Hot Encoding to encode the y-values. The dataset was divided in 80-20 train and test ratio. For train data a decision tree is created using DecisionTreeRegressor class of sklearn library. We used the squared loss to compute the difference between predicted values and true values. A new target is calculated using the loss. The new target is used to train the decision tree as a next estimator. For our model, we used the n_estimators, lr and max_depth as the hyperparameters.

Once we have the weak learners trained for each estimator the final value is predicted as:

$$y_{pred} = lr * estimator_1 + lr * estimator_2 + \cdots + lr * estimator_n$$

We performed an argmax operation to identify the class for our prediction as the index of maximum value. Fig. 7, Fig. 8 and Fig. 9 shows the error or loss values per estimator for gradient boosting weak learners and performance metrics obtained are shown in Table 1.

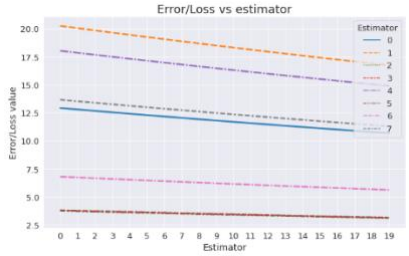

Fig.7. Loss plots for Ecoli dataset

Fig.8. Loss plots for Glass dataset

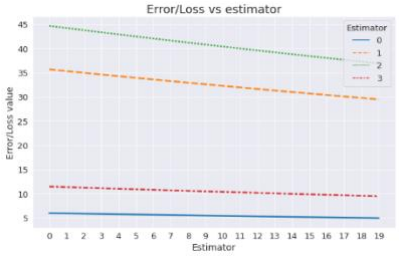

Fig.9. Loss plots for Wifi dataset

We were not able to implement **logistic regression** from scratch. Our team partner assigned with logistic regression didn't do his part, so we used inbuilt sklearn logistic regression function directly with iterations as a choice to all three datasets to obtain plots shown in Fig.10, Fig.11 and Fig.12, and performance parameters for comparison in Table 1.

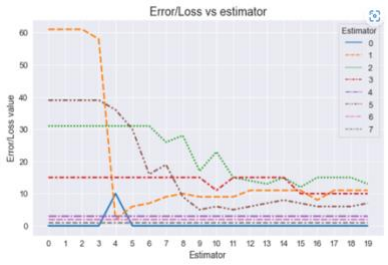

Fig.10. Loss plots for Ecoli dataset

Fig.11. Loss plots for Glass dataset

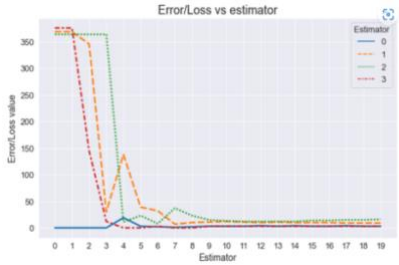

Fig.12. Loss plots for Wifi dataset

Team Contribution

a. **Pooja Yadav** worked on gradient boosting code implementation, performance comparison of all models and writing the report.
b. **Praveen Paidi** worked on Adaboost and Logistic regression code implementation, processed the datasets obtained from UCI repository [2] and assisted in report writing.
c. **Furqan Al Idrees Dastagir** had to work on logistic regression but didn't do his part (No contribution)

Table1. The performance matrices of each model and comparison among the models

Model	Performance Metrics	Ecoli (Highly imbalanced)		Glass (Moderately imbalanced)		Wifi (balanced)		Comparison
		Train	Test	Train	Test	Train	Test	
Gradient boosting	Sensitivity	0.826	0.726	0.850	0.686	0.983	0.971	All three methods show consistent high specificity rates but the sensitivity rate on test data is low for imbalanced data. Gradient boosting performed better for highly imbalanced data compared to moderately imbalanced data.
	Specificity	**0.984**	**0.960**	**0.942**	**0.894**	**0.993**	**0.990**	
	Precision	0.930	0.768	0.866	0.763	0.985	0.972	
	F1-score	0.863	0.733	0.850	0.693	0.983	0.971	
	g-mean	0.895	0.819	0.886	0.764	0.989	0.981	
Adaboost	Sensitivity	0.970	0.716	0.982	0.850	0.994	0.978	Test accuracy for all datasets is bit lower than training accuracy in showing overfitting which further needs hyper parameter tuning. Adaboost performed better on moderately imbalanced data compared to highly imbalanced data.
	Specificity	**0.992**	**0.967**	**0.995**	**0.938**	**0.998**	**0.992**	
	Precision	0.982	0.744	0.993	0.838	0.994	0.977	
	F1-score	0.976	0.720	0.966	0.831	0.994	0.977	
	g-mean	0.981	0.813	0.988	0.893	0.996	0.985	
Logistic regression	Sensitivity	0.669	0.673	0.527	0.680	0.978	0.969	Logistic regression performed better in balanced case and worsened with increase in imbalance. We might improve performance further by increasing the number of iterations.
	Specificity	**0.941**	**0.957**	**0.916**	**0.882**	**0.993**	**0.989**	
	Precision	0.829	0.856	0.777	0.587	0.979	0.968	
	F1-score	0.705	0.709	0.550	0.608	0.979	0.969	
	g-mean	0.772	0.779	0.614	0.692	0.986	0.979	

Conclusion

Pros:

1. In our experimental results we analyzed that all the models showed consistent high specificity rates classify imbalanced data.
2. Boosting models investigated are robust to classify imbalanced data compared to logistic regression.
3. Gradient boosting seems to perform well for datasets that are severely imbalanced while Adaboost seems to perform better for moderately imbalanced data.

Cons:

1. All three methods had low sensitivity rate for imbalanced data.
2. Boosting models have overfitting issue as their performance on training set is better than on test set. Adaboost showed higher overfitting.
3. Moreover, the study did not find one general best method for imbalanced datasets, as each minority percentage has a unique best-performing method.

Through our project, we comprehended the theory and statistical concept of boosting algorithms for multiclassification and imbalanced data. We were able to similar results as original paper with few limitations.

Work in [1] can be further improved by investigating more methods and coming up with a state-of-the-art method which can be considered as generalized best method for imbalanced dataset instead of minority percentage dependence. Hyper parameter tuning can further improve classification.

References

[1] Lai, S. B. S., Shahri, N. H. N. B. M., Mohamad, M. B., Rahman, H. A. B. A., & Rambli, A. B., "Comparing the Performance of AdaBoost, XGBoost, and Logistic Regression for Imbalanced Data", Mathematics and Statistics 9(3): 379-385, 202, DOI: 10.13189/ms.2021.090320.

[2] A. Sonak and R. A. Patankar, "A Survey on Methods to Handle Imbalance Dataset," Int. J. Comput. Sci. Mob. Comput., 2015

[3] Procopio, M. J., Mulligan, J., & Grudic, G., "Coping with imbalanced training data for improved terrain prediction in autonomous outdoor robot navigation", 2010 IEEE International Conference on Robotics and Automation (pp. 518-525).

[4] V. Ganganwar, "An overview of classification algorithms for imbalanced datasets," *Int. J. Emerg. Technol. Adv. Eng.*, 2012.

[5] B. W. Yap, K. A. Rani, H. A. Abd Rahman, S. Fong, Z. Khairudin, and N. N. Abdullah, "An application of oversampling, undersampling, bagging and boosting in handling imbalanced datasets," in *Lecture Notes in Electrical Engineering*, 2014, doi: 10.1007/978-981-4585-1 8-7_2.

[6] Y. B. Wah, H. A. A. Rahman, H. He, and A. Bulgiba, "Handling imbalanced dataset using SVM and k-NN approach," in *AIP Conference Proceedings*, 2016, doi: 10.1063/1.4954536.

[7] B. Krawczyk, "Learning from imbalanced data: open challenges and future directions," *Progress in Artificial Intelligence*. 2016, doi: 10.1007/s13748-016-0094-0.

[8] J. Song, X. Lu, and X. Wu, "An improved adaboost algorithm for unbalanced classification data", 6th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2009, 2009, doi: 10.1109/FSKD.2009.60 8.

[9] UCI Repository: https://archive.ics.uci.edu/ml/datasets.php.

[10] R. E. Schapire, "A brief introduction to boosting", IJCAI International Joint Conference on Artificial Intelligence, 1999.

[11] JH Friedman , "Greedy function approximation: a gradient boosting machine", Annals of statistics, 2001 – JSTOR.

[12] https://towardsdatascience.com/linear-regression-explained-d0a1068accb9

[13] https://towardsdatascience.com/logistic-regression-explained-9ee73cede081

Value of concepts learned in course

We were able to develop and implement codes from scratch based on our learning in class. We could relate the statistical part of algorithm learnt from class and homework. Logistic Regression was completely learnt in class and directly emulated on real datasets here. Anomaly detection, measure of accuracy, sensitivity, specificity by the means of confusion matrix (true positive, true negative, False positive, False negative) are directly applied in calculation of performance of models and comparison among them. Boosting was although not much stressed in curriculum, the bagging and boosting concepts are relatable to concepts of K-means clustering. One hot encoding used for logistic classification and gradient descent is helpful in implementing here for Multiclass classification.

Appendix

Code Adaboost

Ecoli dataset:

```
import pandas as pd
from pandas import read_csv
from collections import Counter
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import math

#data file input directory
filename = r'C:\Users\praveen\project_adaboost\ecoli.csv'
df = read_csv(filename, header=None)

# data forming, forming test and train data
x = df
train, test = train_test_split(x, test_size = 0.2)   #0.295
Y_train=train[7]
Y_test=test[7]
X_train=train.drop(7, axis = 1)
X_test=test.drop(7, axis = 1)
X_train=np.array(X_train)
Y_train=np.array(Y_train)
Y_test=np.array(Y_test)


# one hot encoding of y train for multiclass
targets1=[]
nb_classes = 8
targets = np.array(Y_train).reshape(-1)
for i in range(len(Y_train)):
    if Y_train[i]=='cp':
        targets1.append(0)
    if Y_train[i]=='im':
        targets1.append(1)
    if Y_train[i]=='imU':
        targets1.append(2)
    if Y_train[i]=='om':
        targets1.append(3)
    if Y_train[i]=='omL':
        targets1.append(4)
    if Y_train[i]=='pp':
        targets1.append(5)
    if Y_train[i]=='imS':
        targets1.append(6)
    if Y_train[i]=='imL':
        targets1.append(7)
y= np.eye(nb_classes)[targets1]

# one hot encoding of y test for multiclass
targets1=[]
targets = np.array(Y_test).reshape(-1)
for i in range(len(Y_test)):
```

```
    if Y_test[i]=='cp':
        targets1.append(0)
    if Y_test[i]=='im':
        targets1.append(1)
    if Y_test[i]=='imU':
        targets1.append(2)
    if Y_test[i]=='om':
        targets1.append(3)
    if Y_test[i]=='omL':
        targets1.append(4)
    if Y_test[i]=='pp':
        targets1.append(5)
    if Y_test[i]=='imS':
        targets1.append(6)
    if Y_test[i]=='imL':
        targets1.append(7)
y_test= np.eye(nb_classes)[targets1]
labels_train=np.argmax(y,axis=1)
labels_test=np.argmax(y_test,axis=1)
(dim1,dim2)=(X_train.shape)
(dimt1,dimt2)=(X_test.shape)


# In[561]:


def init_plot_settings():
  # Visualization Reference: https://towardsdatascience.com/a-simple-guide-to-beautiful-visualizations-in-python-
f564e6b9d392
  sns.set_style('darkgrid') # darkgrid, white grid, dark, white and ticks
  plt.rc('axes', titlesize=18)     # fontsize of the axes title
  plt.rc('axes', labelsize=14)    # fontsize of the x and y labels
  plt.rc('xtick', labelsize=13)    # fontsize of the tick labels
  plt.rc('ytick', labelsize=13)    # fontsize of the tick labels
  plt.rc('legend', fontsize=13)    # legend fontsize
  plt.rc('font', size=13)          # controls default text sizes


# In[562]:


#error calculation for each and every class
def error_calc(ypred,labels_train,m):
    con=[]
    for i in range(8):
        error=0
        searchval = i
        ii = np.where(labels_train == searchval)[0]
        for i  in range(len(ii)):
            a=ii[i]
            if ypred[a]!=labels_train[a]:
                error=error+1
        con.append(error)
    return con


# In[563]:
```

```
alphas = []
alp=[]
training_errors = []
prediction_errors = []
prediction_testerrors = []
init_plot_settings()

Iterations = 20                    # Number of iterations is a hyperparameter
ERROR_TRAIN=np.zeros((8,Iterations))
Y_PRED=np.zeros((dim1,8))
Y_PRED_TEST=np.zeros((dimt1,8))
y_interim=np.zeros(dim1,dtype=int)

# Set weights initially
w_i = np.ones(len(labels_train)) * 1 / len(labels_train)

#Adaboost Algorithm
for m in range(Iterations):

    # Fit weak classifier and predict labels
    clf = DecisionTreeClassifier(max_depth =4 )     # Decision Tree depth is a hyper parameter
    clf.fit(X_train, labels_train, sample_weight = w_i)

    y_pred = clf.predict(X_train)          #  Predicting training values
    y_pred_test=clf.predict(X_test)        # Prediciting Testing Values

    Indicator=[]
    #Indicator function for calcualtion of error
    for i in range(len(labels_train)):
        if labels_train[i]!=(y_pred[i]):
            Indicator.append(1)
        else:
            Indicator.append(0)
    Indicator=np.array(Indicator)          # Indicator function

    #Total error Calculation
    error_m=np.sum(np.multiply(w_i,Indicator))/np.sum(w_i)

    #For training Errors graph
    training_errors.append(error_m)

    #alpha Calculation
    alpha_m =np.log((1 - error_m) / error_m)
    alphas.append(alpha_m)

    #Updation of weights  and learning rate is hyper parameter
    w_i=np.multiply(w_i,np.exp(np.multiply(0.5*alpha_m,Indicator)))   # WeightIndicator

    # one hot encoding for predicated label of training
    targets = np.array(y_pred).reshape(-1)
    p_train= np.eye(nb_classes)[targets]

    Y_int=p_train*alpha_m
    Y_PRED=Y_int+Y_PRED
    y_pred_interim=(np.argmax((Y_PRED),axis=1))

    err=0
```

```
    for i in range(len(y_pred)):
        if y_pred_interim[i]!=labels_train[i]:
            err=err+1
    prediction_errors.append(err)

    ERROR_TRAIN[:,m]=error_calc(y_pred_interim,labels_train,m)

    #one hot encoding for testing
    nb_classes = 8
    targets = np.array(y_pred_test).reshape(-1)
    p_test= np.eye(nb_classes)[targets]

    Y_int=p_test*alpha_m
    Y_PRED_TEST=Y_int+Y_PRED_TEST
    y_pred_test_interim=(np.argmax((Y_PRED_TEST),axis=1))

    err=0
    for i in range(len(y_pred_test_interim)):
        if y_pred_test_interim[i]!=labels_test[i]:
            err=err+1
    prediction_testerrors.append(err)

#final predicition
y_pred_train=(np.argmax((Y_PRED),axis=1))
y_pred_test=(np.argmax((Y_PRED_TEST),axis=1))

#PLOTTING
plt.figure(figsize=(10, 6))
plt.plot(prediction_errors,label='Training error')
plt.xlabel('Estimators')
plt.ylabel('Training Error')
plt.title('Training Error vs Number of Estimators')
plt.legend()

plt.figure(figsize=(10, 6))
ax = sns.lineplot(data=ERROR_TRAIN.T[:,:], linewidth=2.5)
ax.set(xlabel='Estimator', ylabel='Error/Loss value', title='Error/Loss vs estimator',xticks=[i for i in
range(ERROR_TRAIN.T.shape[0])])
ax.legend(title='Estimator', title_fontsize = 13, loc=1)
plt.show()


# In[616]:


from sklearn.metrics import confusion_matrix

targets = np.array(y_pred_train).reshape(-1)
Y_PRED= np.eye(nb_classes)[targets]

targets = np.array(y_pred_test).reshape(-1)
Y_PRED_TEST= np.eye(nb_classes)[targets]

def compute_performance(y, y_pred):

    mean_sensitivity=[]
    mean_specificity=[]
    mean_precision=[]
```

```
    mean_f1_score=[]
    g_mean_avg=[]
    total_error = 0

    for i, y_val in enumerate(y):
        if not np.array_equal(y_val,y_pred[i]):
          total_error += 1

    acc_score = 1 - total_error/y.shape[0]

    acc = []
    cf = []
    precision = []
    recall = []


    for i in range(8):
#         print('Class_'+str(i+1))
        tn, fp, fn, tp = confusion_matrix(y[:,i], y_pred[:,i]).ravel()
        sensitivity = tp / (tp+fn)
        mean_sensitivity.append(sensitivity)
        specificity = tn / (tn+fp)
        mean_specificity.append(specificity)
        precision = tp / (tp + fp)
        mean_precision.append(precision)
        f1_score = 2*tp / (2*tp + fp + fn)
        mean_f1_score.append(f1_score)
        g_mean = (sensitivity * specificity) ** (1/2)
        g_mean_avg.append(g_mean)

        #for each invidual class performances
#         print(f"TP: {tp}, FP: {fp}, TN: {tn}, FN: {fn}")
#         print(f"Sensitivity : {sensitivity}")
#         print(f"Specificity : {specificity}")
#         print(f"Precision : {precision}")
#         print(f"F1-score : {f1_score}")
#         print(f"g-mean : {g_mean}")


    mean_sensitivity = [num for num in mean_sensitivity if num<len(labels_train)]
    mean_specificity = [num for num in mean_specificity if num<len(labels_train)]
    mean_precision = [num for num in mean_precision if num<len(labels_train)]
    mean_f1_score = [num for num in mean_f1_score if num<len(labels_train)]
    g_mean_avg = [num for num in g_mean_avg if num<len(labels_train)]


    sensitivity=sum(mean_sensitivity)/len(mean_sensitivity)
    specificity=sum(mean_specificity)/len(mean_specificity)
    precision=sum(mean_precision)/len(mean_precision)
    F1_score=sum(mean_f1_score)/len(mean_f1_score)
    g_mean=sum(g_mean_avg)/len(g_mean_avg)

    print(f"sensitivity : {sensitivity}")
    print(f"specificity : {specificity}")
    print(f"precision   : {precision}")
    print(f"F1_score : {F1_score}")
    print(f"G_mean : {g_mean}")
```

In[617]:

```
print('The training performances are as following')
compute_performance(y, Y_PRED)
print('The testing performances are as following')
compute_performance(y_test,Y_PRED_TEST)
```

Glass dataset

```
import pandas as pd
from pandas import read_csv
from collections import Counter
import numpy as np
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import math

filename = r'C:\Users\praveen\project_adaboost\glass.csv'
df = read_csv(filename, header=None)

# data forming
x = df
train, test = train_test_split(x, test_size = 0.2)
Y_train=train[9]
Y_test=test[9]
X_train=train.drop(9, axis = 1)
X_test=test.drop(9, axis = 1)
X_train=np.array(X_train)
X_test=np.array(X_test)
Y_train=np.array(Y_train)
Y_test=np.array(Y_test)

for i in range(len(Y_train)):
    if Y_train[i]<4:
        Y_train[i]=Y_train[i]-1
    if Y_train[i]>4:
        Y_train[i]=Y_train[i]-2

for i in range(len(Y_test)):
    if Y_test[i]<4:
        Y_test[i]=Y_test[i]-1
    if Y_test[i]>4:
        Y_test[i]=Y_test[i]-2

# one hot encoding of y train
nb_classes = 6
targets = np.array(Y_train).reshape(-1)
y= np.eye(nb_classes)[targets]
labels_train=np.argmax(y,axis=1)

# one hot encoding of y train
targets = np.array(Y_test).reshape(-1)
y_test= np.eye(nb_classes)[targets]
labels_test=np.argmax(y_test,axis=1)
```

```
(dimt1,)=(labels_test.shape)
(dim1,)=(labels_train.shape)


# In[53]:


def init_plot_settings():
  # Visualization Reference: https://towardsdatascience.com/a-simple-guide-to-beautiful-visualizations-in-python-f564e6b9d392
  sns.set_style('darkgrid') # darkgrid, white grid, dark, white and ticks
  plt.rc('axes', titlesize=18)     # fontsize of the axes title
  plt.rc('axes', labelsize=14)    # fontsize of the x and y labels
  plt.rc('xtick', labelsize=13)    # fontsize of the tick labels
  plt.rc('ytick', labelsize=13)    # fontsize of the tick labels
  plt.rc('legend', fontsize=13)    # legend fontsize
  plt.rc('font', size=13)          # controls default text sizes

#error calculation for each and every class
def error_calc(ypred,labels_train,m):
    con=[]
    for i in range(6):
        error=0
        searchval = i
        ii = np.where(labels_train == searchval)[0]
        for i  in range(len(ii)):
            a=ii[i]
            if ypred[a]!=labels_train[a]:
                error=error+1
        con.append(error)
    return con


# In[60]:


alphas = []
alp=[]
training_errors = []
prediction_errors = []
prediction_testerrors = []
init_plot_settings()

Iterations = 20                        # Number of iterations is a hyperparameter
ERROR_TRAIN=np.zeros((6,Iterations))
Y_PRED=np.zeros((dim1,6))
Y_PRED_TEST=np.zeros((dimt1,6))
y_interim=np.zeros(dim1,dtype=int)

# Set weights initially
w_i = np.ones(len(labels_train)) * 1 / len(labels_train)

#Adaboost Algorithm
for m in range(Iterations):

    # Fit weak classifier and predict labels
    clf = DecisionTreeClassifier(max_depth =4 )     # Decision Tree depth is a hyper parameter
    clf.fit(X_train, labels_train, sample_weight = w_i)
```

```
y_pred = clf.predict(X_train)          #  Predicting training values
y_pred_test=clf.predict(X_test)        # Prediciting Testing Values

Indicator=[]
#Indicator function for calcualtion of error
for i in range(len(labels_train)):
   if labels_train[i]!=(y_pred[i]):
      Indicator.append(1)
   else:
      Indicator.append(0)
Indicator=np.array(Indicator)          # Indicator function

#Total error Calculation
error_m=np.sum(np.multiply(w_i,Indicator))/np.sum(w_i)

#For training Errors graph
training_errors.append(error_m)

#alpha Calculation
alpha_m =np.log((1 - error_m) / error_m)
alphas.append(alpha_m)

WeightIndicator=[]
for i in range(len(labels_train)):
   if Indicator[i]==1:
      WeightIndicator.append(1)
   else:
      WeightIndicator.append(-1)

#Updation of weights  and learning rate is hyper parameter
w_i=np.multiply(w_i,np.exp(np.multiply(0.25*alpha_m,Indicator)))   # WeightIndicator

# one hot encoding for predicated label of training
targets = np.array(y_pred).reshape(-1)
p_train= np.eye(nb_classes)[targets]

Y_int=p_train*alpha_m
Y_PRED=Y_int+Y_PRED
y_pred_interim=(np.argmax((Y_PRED),axis=1))

err=0
for i in range(len(y_pred)):
   if y_pred_interim[i]!=labels_train[i]:
      err=err+1
prediction_errors.append(err)

ERROR_TRAIN[:,m]=error_calc(y_pred_interim,labels_train,m)

#one hot encoding for testing
targets = np.array(y_pred_test).reshape(-1)
p_test= np.eye(nb_classes)[targets]

Y_int=p_test*alpha_m
Y_PRED_TEST=Y_int+Y_PRED_TEST
y_pred_test_interim=(np.argmax((Y_PRED_TEST),axis=1))

err=0
```

```
    for i in range(len(y_pred_test_interim)):
        if y_pred_test_interim[i]!=labels_test[i]:
            err=err+1
    prediction_testerrors.append(err)

#final predicition
y_pred_train=(np.argmax((Y_PRED),axis=1))
y_pred_test=(np.argmax((Y_PRED_TEST),axis=1))


#PLOTTING
plt.figure(figsize=(10, 6))
plt.plot(prediction_errors,label='Training error')
plt.xlabel('Estimators')
plt.ylabel('Training Error')
plt.title('Training Error vs Number of Estimators')
plt.legend()


plt.figure(figsize=(10, 6))
ax = sns.lineplot(data=ERROR_TRAIN.T[:,:], linewidth=2.5)
ax.set(xlabel='Estimator', ylabel='Error/Loss value', title='Error/Loss vs estimator',xticks=[i for i in
range(ERROR_TRAIN.T.shape[0])])
ax.legend(title='Estimator', title_fontsize = 13, loc=1)
plt.show()


# In[69]:


from sklearn.metrics import confusion_matrix

targets = np.array(y_pred_train).reshape(-1)
Y_PRED= np.eye(nb_classes)[targets]

targets = np.array(y_pred_test).reshape(-1)
Y_PRED_TEST= np.eye(nb_classes)[targets]

def compute_performance(y, y_pred):

    mean_sensitivity=[]
    mean_specificity=[]
    mean_precision=[]
    mean_f1_score=[]
    g_mean_avg=[]
    total_error = 0

    for i, y_val in enumerate(y):
        if not np.array_equal(y_val,y_pred[i]):
            total_error += 1

    acc_score = 1 - total_error/y.shape[0]

    acc = []
    cf = []
    precision = []
    recall = []
```

```
    for i in range(6):
#         print('Class_'+str(i+1))
        tn, fp, fn, tp = confusion_matrix(y[:,i], y_pred[:,i]).ravel()
        sensitivity = tp / (tp+fn)
        mean_sensitivity.append(sensitivity)
        specificity = tn / (tn+fp)
        mean_specificity.append(specificity)
        precision = tp / (tp + fp)
        mean_precision.append(precision)
        f1_score = 2*tp / (2*tp + fp + fn)
        mean_f1_score.append(f1_score)
        g_mean = (sensitivity * specificity) ** (1/2)
        g_mean_avg.append(g_mean)

        #for each invidual class performances
#         print(f"TP: {tp}, FP: {fp}, TN: {tn}, FN: {fn}")
#         print(f"Sensitivity : {sensitivity}")
#         print(f"Specificity : {specificity}")
#         print(f"Precision : {precision}")
#         print(f"F1-score : {f1_score}")
#         print(f"g-mean : {g_mean}")


    mean_sensitivity = [num for num in mean_sensitivity if num<len(labels_train)]
    mean_specificity = [num for num in mean_specificity if num<len(labels_train)]
    mean_precision = [num for num in mean_precision if num<len(labels_train)]
    mean_f1_score = [num for num in mean_f1_score if num<len(labels_train)]
    g_mean_avg = [num for num in g_mean_avg if num<len(labels_train)]


    sensitivity=sum(mean_sensitivity)/len(mean_sensitivity)
    specificity=sum(mean_specificity)/len(mean_specificity)
    precision=sum(mean_precision)/len(mean_precision)
    F1_score=sum(mean_f1_score)/len(mean_f1_score)
    g_mean=sum(g_mean_avg)/len(g_mean_avg)

    print(f"sensitivity : {sensitivity}")
    print(f"specificity : {specificity}")
    print(f"precision   : {precision}")
    print(f"F1_score : {F1_score}")
    print(f"G_mean : {g_mean}")




# In[70]:


print('The following are the training dataset performances')
compute_performance(y, Y_PRED)
print('The following are the testing dataset performances')
compute_performance(y_test,Y_PRED_TEST)
```

Wifi dataset

```
import pandas as pd
from pandas import read_csv
```

```
from collections import Counter
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as snss
import math

filename = r'C:\Users\praveen\project_adaboost\wifi.csv'
df = read_csv(filename, header=None)
# data forming
x = df
train, test = train_test_split(x, test_size = 0.2)

Y_train=train[7]
Y_test=test[7]
X_train=train.drop(7, axis = 1)
X_test=test.drop(7, axis = 1)
X_train=np.array(X_train)
X_test=np.array(X_test)
Y_train=np.array(Y_train)
Y_train=Y_train-1
Y_test=np.array(Y_test)
Y_test=Y_test-1

# one hot encoding of y train
nb_classes = 4
targets = np.array(Y_train).reshape(-1)
y= np.eye(nb_classes)[targets]
labels_train=np.argmax(y,axis=1)

# one hot encoding of y train
nb_classes = 4
targets = np.array(Y_test).reshape(-1)
y_test= np.eye(nb_classes)[targets]
labels_test=np.argmax(y_test,axis=1)
(X_train.shape)
(dimt1,)=(labels_test.shape)
(dim1,)=(labels_train.shape)


# In[21]:


def init_plot_settings():
  # Visualization Reference: https://towardsdatascience.com/a-simple-guide-to-beautiful-visualizations-in-python-
f564e6b9d392
  sns.set_style('darkgrid') # darkgrid, white grid, dark, white and ticks
  plt.rc('axes', titlesize=18)     # fontsize of the axes title
  plt.rc('axes', labelsize=14)    # fontsize of the x and y labels
  plt.rc('xtick', labelsize=13)    # fontsize of the tick labels
  plt.rc('ytick', labelsize=13)    # fontsize of the tick labels
  plt.rc('legend', fontsize=13)    # legend fontsize
  plt.rc('font', size=13)          # controls default text sizes

#error calculation for each and every class
def error_calc(ypred,labels_train,m):
    con=[]
```

```
    for i in range(4):
        error=0
        searchval = i
        ii = np.where(labels_train == searchval)[0]
        for i  in range(len(ii)):
            a=ii[i]
            if ypred[a]!=labels_train[a]:
                error=error+1
        con.append(error)
    return con


# In[29]:


alphas = []
alp=[]
training_errors = []
prediction_errors = []
prediction_testerrors = []
init_plot_settings()

Iterations = 20                        # Number of iterations is a hyperparameter
ERROR_TRAIN=np.zeros((4,Iterations))
Y_PRED=np.zeros((dim1,4))
Y_PRED_TEST=np.zeros((dimt1,4))
y_interim=np.zeros(dim1,dtype=int)

# Set weights initially
w_i = np.ones(len(labels_train)) * 1 / len(labels_train)

#Adaboost Algorithm
for m in range(Iterations):

    # Fit weak classifier and predict labels
    clf = DecisionTreeClassifier(max_depth =3 )     # Decision Tree depth is a hyper parameter
    clf.fit(X_train, labels_train, sample_weight = w_i)

    y_pred = clf.predict(X_train)          #  Predicting training values
    y_pred_test=clf.predict(X_test)        # Prediciting Testing Values

    Indicator=[]
    #Indicator function for calcualtion of error
    for i in range(len(labels_train)):
        if labels_train[i]!=(y_pred[i]):
            Indicator.append(1)
        else:
            Indicator.append(0)
    Indicator=np.array(Indicator)          # Indicator function

    #Total error Calculation
    error_m=np.sum(np.multiply(w_i,Indicator))/np.sum(w_i)

    #For training Errors graph
    training_errors.append(error_m)

    #alpha Calculation
    alpha_m =np.log((1 - error_m) / error_m)
```

```
    alphas.append(alpha_m)

    WeightIndicator=[]
    for i in range(len(labels_train)):
        if Indicator[i]==1:
            WeightIndicator.append(1)
        else:
            WeightIndicator.append(-1)

    #Updation of weights  and learning rate is hyper parameter
    w_i=np.multiply(w_i,np.exp(np.multiply(0.5*alpha_m,Indicator)))   # WeightIndicator

    # one hot encoding for predicated label of training
    nb_classes = 4
    targets = np.array(y_pred).reshape(-1)
    p_train= np.eye(nb_classes)[targets]

    Y_int=p_train*alpha_m
    Y_PRED=Y_int+Y_PRED
    y_pred_interim=(np.argmax((Y_PRED),axis=1))

    err=0
    for i in range(len(y_pred)):
        if y_pred_interim[i]!=labels_train[i]:
            err=err+1
    prediction_errors.append(err)

    ERROR_TRAIN[:,m]=error_calc(y_pred_interim,labels_train,m)

    #one hot encoding for testing
    nb_classes = 4
    targets = np.array(y_pred_test).reshape(-1)
    p_test= np.eye(nb_classes)[targets]

    Y_int=p_test*alpha_m
    Y_PRED_TEST=Y_int+Y_PRED_TEST
    y_pred_test_interim=(np.argmax((Y_PRED_TEST),axis=1))

    err=0
    for i in range(len(y_pred_test_interim)):
        if y_pred_test_interim[i]!=labels_test[i]:
            err=err+1
    prediction_testerrors.append(err)

#final predicition
y_pred_train=(np.argmax((Y_PRED),axis=1))
y_pred_test=(np.argmax((Y_PRED_TEST),axis=1))


#PLOTTING
plt.figure(figsize=(10, 6))
plt.plot(prediction_errors,label='Training error')
plt.xlabel('Estimators')
plt.ylabel('Training Error')
plt.title('Training Error vs Number of Estimators')
plt.legend()
```

```
plt.figure(figsize=(10, 6))
ax = sns.lineplot(data=ERROR_TRAIN.T[:,:], linewidth=2.5)
ax.set(xlabel='Estimator', ylabel='Error/Loss value', title='Error/Loss vs estimator',xticks=[i for i in range(ERROR_TRAIN.T.shape[0])])
ax.legend(title='Estimator', title_fontsize = 13, loc=1)
plt.show()


# In[41]:


from sklearn.metrics import confusion_matrix

targets = np.array(y_pred_train).reshape(-1)
Y_PRED= np.eye(nb_classes)[targets]

targets = np.array(y_pred_test).reshape(-1)
Y_PRED_TEST= np.eye(nb_classes)[targets]

def compute_performance(y, y_pred):

    mean_sensitivity=[]
    mean_specificity=[]
    mean_precision=[]
    mean_f1_score=[]
    g_mean_avg=[]
    total_error = 0

    for i, y_val in enumerate(y):
        if not np.array_equal(y_val,y_pred[i]):
          total_error += 1

    acc_score = 1 - total_error/y.shape[0]

    acc = []
    cf = []
    precision = []
    recall = []


    for i in range(4):
#         print('Class_'+str(i+1))
        tn, fp, fn, tp = confusion_matrix(y[:,i], y_pred[:,i]).ravel()
        sensitivity = tp / (tp+fn)
        mean_sensitivity.append(sensitivity)
        specificity = tn / (tn+fp)
        mean_specificity.append(specificity)
        precision = tp / (tp + fp)
        mean_precision.append(precision)
        f1_score = 2*tp / (2*tp + fp + fn)
        mean_f1_score.append(f1_score)
        g_mean = (sensitivity * specificity) ** (1/2)
        g_mean_avg.append(g_mean)

        #for each invidual class performances
#         print(f"TP: {tp}, FP: {fp}, TN: {tn}, FN: {fn}")
#         print(f"Sensitivity : {sensitivity}")
#         print(f"Specificity : {specificity}")
```

```
#       print(f"Precision : {precision}")
#       print(f"F1-score : {f1_score}")
#       print(f"g-mean : {g_mean}")


    mean_sensitivity = [num for num in mean_sensitivity if num<len(labels_train)]
    mean_specificity = [num for num in mean_specificity if num<len(labels_train)]
    mean_precision = [num for num in mean_precision if num<len(labels_train)]
    mean_f1_score = [num for num in mean_f1_score if num<len(labels_train)]
    g_mean_avg = [num for num in g_mean_avg if num<len(labels_train)]


    sensitivity=sum(mean_sensitivity)/len(mean_sensitivity)
    specificity=sum(mean_specificity)/len(mean_specificity)
    precision=sum(mean_precision)/len(mean_precision)
    F1_score=sum(mean_f1_score)/len(mean_f1_score)
    g_mean=sum(g_mean_avg)/len(g_mean_avg)

    print(f"sensitivity : {sensitivity}")
    print(f"specificity : {specificity}")
    print(f"precision   : {precision}")
    print(f"F1_score : {F1_score}")
    print(f"G_mean : {g_mean}")




# In[42]:


print('The following are the training dataset performances')
compute_performance(y, Y_PRED)
print('The following are the testing dataset performances')
compute_performance(y_test,Y_PRED_TEST)
```

Code gradient boosting

```
from google.colab import drive
drive.mount('/content/drive')
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
from sklearn.metrics import classification_report

def read_data(file):
  '''
  Function to read data
  Data source:
```

```
  '''
  df = pd.read_csv(file, header=None)
  num_col = len(df. columns) - 1
  X = df.iloc[:, 0:num_col]
  y = df.iloc[:,num_col]
  y = y.values.reshape(-1,1)
  encoder = OneHotEncoder(sparse=False)
  y_OneHot = encoder.fit_transform(y)
  return X, y_OneHot

def data_standardization(X):
  '''
  Function to standardize the data using StandardScaler
  X: input data
  '''
  sc = StandardScaler()
  X = sc.fit_transform(X)
  return X

def init_plot_settings():
  # Visualization Reference: https://towardsdatascience.com/a-simple-guide-to-beautiful-visualizations-in-python-f564e6b9d392
  sns.set_style('darkgrid') # darkgrid, white grid, dark, white and ticks
  plt.rc('axes', titlesize=18)     # fontsize of the axes title
  plt.rc('axes', labelsize=14)    # fontsize of the x and y labels
  plt.rc('xtick', labelsize=13)    # fontsize of the tick labels
  plt.rc('ytick', labelsize=13)    # fontsize of the tick labels
  plt.rc('legend', fontsize=13)    # legend fontsize
  plt.rc('font', size=13)          # controls default text sizes

def plot_loss(error):
  error = np.array(error)
  plt.figure(figsize=(10, 6))
  ax = sns.lineplot(data=error[:,:], linewidth=2.5)
  ax.set(xlabel='Estimator', ylabel='Error/Loss value', title='Error/Loss vs estimator',xticks=[i for i in range(error.shape[0])])
  ax.legend(title='Estimator', title_fontsize = 13, loc=1)
  plt.show()

def Gradient_Boosting(X_train, y_train, n_estimator = 10, max_depth = 2, lr = 0.01):
  gradient_boosting_estimator = []
  y_train_org = y_train.copy()
  train_error = []
  for i in range(n_estimator):
    clf = DecisionTreeRegressor(max_depth=max_depth, random_state=1)
    model = clf.fit(X_train, y_train)

    y_pred = model.predict(X_train)
    loss_grad = y_train - y_pred
    train_error.append(sum(abs(loss_grad)))

    y_train = y_train - lr * loss_grad

    gradient_boosting_estimator.append((lr, model))


  plot_loss(train_error)
```

```
  return gradient_boosting_estimator

def predict(X, estimator, classes):
  pred_vals = np.zeros((X.shape[0], classes))
  for val in estimator:
    pred_vals += val[0] * val[1].predict(X)

  y_pred = np.zeros((X.shape[0], classes))

  for i, pred in enumerate(pred_vals):
    max_idx = np.argmax(pred)
    y_pred[i][max_idx] = 1

  return y_pred

def Decision_tree(X, y):
  clf = DecisionTreeClassifier(max_depth = 1)
  clf.fit(X,y)
  return clf

def compute_preformance(y, y_pred, classes):

  total_error = 0

  for i, y_val in enumerate(y):
    if not np.array_equal(y_val,y_pred[i]):
      total_error += 1

  acc_score = 1 - total_error/y.shape[0]

  acc = []
  cf = []
  precision = []
  recall = []

  target_names = ['Class'+str(i+1) for i in range(classes)]

  for i in range(y.shape[1]):
    print('Class_'+str(i+1))
    tn, fp, fn, tp = confusion_matrix(y[:,i], y_pred[:,i]).ravel()
    sensitivity = tp / (tp+fn)
    specificity = tn / (tn+fp)
    precision = tp / (tp + fp)
    f1_score = 2*tp / (2*tp + fp + fn)
    g_mean = (sensitivity * specificity) ** (1/2)
    print(f"TP: {tp}, FP: {fp}, TN: {tn}, FN: {fn}")
    print(f"Sensitivity : {sensitivity}")
    print(f"Specificity : {specificity}")
    print(f"Precision : {precision}")
    print(f"F1-score : {f1_score}")
    print(f"g-mean : {g_mean}")

file_path = '/content/drive/MyDrive/SML Project/wifi.csv'
X, y = read_data(file_path)
classes = y.shape[1]
init_plot_settings()
```

```
# Data creation
X = data_standardization(X)
# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

gradient_boosting_estimator = Gradient_Boosting(X_train, y_train, n_estimator = 20, max_depth = 4, lr = 0.01)

y_pred_train = predict(X_train, gradient_boosting_estimator, classes)
y_pred_test = predict(X_test, gradient_boosting_estimator, classes)
print(f"Train:")
compute_preformance(y_train, y_pred_train, classes)

print(f"Test:")
compute_preformance(y_test, y_pred_test, classes)



# Decision Tree Predictions
dt = Decision_tree(X_train, y_train)
y_pred_train = dt.predict(X_train)
y_pred_test = dt.predict(X_test)
print(f"Train:")
compute_preformance(y_train, y_pred_train, classes)

print(f"Test:")
compute_preformance(y_test, y_pred_test, classes)
```

Code logistic regression
Ecoli dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score


# In[48]:


file = pd.read_csv(r'C:\Users\praveen\project_adaboost\ecoli.csv')

df = file.values
train, test = train_test_split(df, test_size = 0.2)
X_train= train[:,0:7]
Y_train = train[:,7]
X_test=test[:,0:7]
Y_test=test[:,7]

# one hot encoding of y train for multiclass
```

```
targets1=[]
nb_classes = 8
targets = np.array(Y_train).reshape(-1)
for i in range(len(Y_train)):
    if Y_train[i]=='cp':
        targets1.append(0)
    if Y_train[i]=='im':
        targets1.append(1)
    if Y_train[i]=='imU':
        targets1.append(2)
    if Y_train[i]=='om':
        targets1.append(3)
    if Y_train[i]=='omL':
        targets1.append(4)
    if Y_train[i]=='pp':
        targets1.append(5)
    if Y_train[i]=='imS':
        targets1.append(6)
    if Y_train[i]=='imL':
        targets1.append(7)
y= np.eye(nb_classes)[targets1]

# one hot encoding of y test for multiclass
targets1=[]
targets = np.array(Y_test).reshape(-1)
for i in range(len(Y_test)):
    if Y_test[i]=='cp':
        targets1.append(0)
    if Y_test[i]=='im':
        targets1.append(1)
    if Y_test[i]=='imU':
        targets1.append(2)
    if Y_test[i]=='om':
        targets1.append(3)
    if Y_test[i]=='omL':
        targets1.append(4)
    if Y_test[i]=='pp':
        targets1.append(5)
    if Y_test[i]=='imS':
        targets1.append(6)
    if Y_test[i]=='imL':
        targets1.append(7)
y_test= np.eye(nb_classes)[targets1]
labels_train=np.argmax(y,axis=1)
lables_test=np.argmax(y_test,axis=1)


# In[49]:


def init_plot_settings():
  # Visualization Reference: https://towardsdatascience.com/a-simple-guide-to-beautiful-visualizations-in-python-f564e6b9d392
  sns.set_style('darkgrid') # darkgrid, white grid, dark, white and ticks
  plt.rc('axes', titlesize=18)     # fontsize of the axes title
  plt.rc('axes', labelsize=14)    # fontsize of the x and y labels
  plt.rc('xtick', labelsize=13)    # fontsize of the tick labels
  plt.rc('ytick', labelsize=13)    # fontsize of the tick labels
```

```
  plt.rc('legend', fontsize=13)    # legend fontsize
  plt.rc('font', size=13)          # controls default text sizes

#error calculation for each and every class
def error_calc(ypred,labels_train,m):
    con=[]
    for i in range(8):
        error=0
        searchval = i
        ii = np.where(labels_train == searchval)[0]
        for i  in range(len(ii)):
            a=ii[i]
            if ypred[a]!=labels_train[a]:
                error=error+1
        con.append(error)
    return con


# In[50]:


init_plot_settings()
trainingerror=[]
ERROR_TRAIN=np.zeros((8,20))
for i in range(20):
    clf = LogisticRegression(max_iter=i)
    clf.fit(X_train,labels_train)
    y_pred_train= clf.predict(X_train)
    y_pred_test=clf.predict(X_test)
    accuracy = accuracy_score(labels_train, y_pred_train)
#     print('Accuracy Score',(1-accuracy)*100)
    trainingerror.append((1-accuracy)*100)

    ERROR_TRAIN[:,i]=error_calc(y_pred_train,labels_train,i)



# In[51]:


plt.figure(figsize=(10, 6))
ax = sns.lineplot(data=ERROR_TRAIN.T[:,:], linewidth=2.5)
ax.set(xlabel='Estimator', ylabel='Error/Loss value', title='Error/Loss vs estimator',xticks=[i for i in
range(ERROR_TRAIN.T.shape[0])])
ax.legend(title='Estimator', title_fontsize = 13, loc=1)
plt.show()


# In[52]:


from sklearn.metrics import confusion_matrix
nb_classes=8
targets = np.array(lables_test).reshape(-1)
y_test= np.eye(nb_classes)[targets]
targets = np.array(y_pred_test).reshape(-1)
y_pred_test= np.eye(nb_classes)[targets]
targets = np.array(y_pred_train).reshape(-1)
```

```
y_pred_train= np.eye(nb_classes)[targets]


# In[53]:


def compute_performance(y, y_pred):

    mean_sensitivity=[]
    mean_specificity=[]
    mean_precision=[]
    mean_f1_score=[]
    g_mean_avg=[]
    total_error = 0

    for i, y_val in enumerate(y):
        if not np.array_equal(y_val,y_pred[i]):
          total_error += 1

    acc_score = 1 - total_error/y.shape[0]

    acc = []
    cf = []
    precision = []
    recall = []


    for i in range(4):
#         print('Class_'+str(i+1))
        tn, fp, fn, tp = confusion_matrix(y[:,i], y_pred[:,i]).ravel()
        sensitivity = tp / (tp+fn)
        mean_sensitivity.append(sensitivity)
        specificity = tn / (tn+fp)
        mean_specificity.append(specificity)
        precision = tp / (tp + fp)
        mean_precision.append(precision)
        f1_score = 2*tp / (2*tp + fp + fn)
        mean_f1_score.append(f1_score)
        g_mean = (sensitivity * specificity) ** (1/2)
        g_mean_avg.append(g_mean)


    sensitivity=sum(mean_sensitivity)/len(mean_sensitivity)
    specificity=sum(mean_specificity)/len(mean_specificity)
    precision=sum(mean_precision)/len(mean_precision)
    F1_score=sum(mean_f1_score)/len(mean_f1_score)
    g_mean=sum(g_mean_avg)/len(g_mean_avg)

    print(f"sensitivity : {sensitivity}")
    print(f"specificity : {specificity}")
    print(f"precision   : {precision}")
    print(f"F1_score : {F1_score}")
    print(f"G_mean : {g_mean}")



# In[54]:
```

```
print('The following are the training dataset performances')
compute_performance(y,y_pred_train )
print('The following are the testing dataset performances')
compute_performance(y_test,y_pred_test)
```

Glass dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix


# In[150]:


file = pd.read_csv (r'C:\Users\praveen\project_adaboost\glass.csv')

df = file.values
train, test = train_test_split(df, test_size = 0.2,random_state=1)
X_train= train[:,0:9]
Y_train = train[:,9]
X_test=test[:,0:9]
Y_test=test[:,9]

for i in range(len(Y_train)):
    if Y_train[i]<4:
        Y_train[i]=Y_train[i]-1
    if Y_train[i]>4:
        Y_train[i]=Y_train[i]-2

for i in range(len(Y_test)):
    if Y_test[i]<4:
        Y_test[i]=Y_test[i]-1
    if Y_test[i]>4:
        Y_test[i]=Y_test[i]-2


# In[151]:


def init_plot_settings():
  # Visualization Reference: https://towardsdatascience.com/a-simple-guide-to-beautiful-visualizations-in-python-f564e6b9d392
  sns.set_style('darkgrid') # darkgrid, white grid, dark, white and ticks
  plt.rc('axes', titlesize=18)     # fontsize of the axes title
  plt.rc('axes', labelsize=14)    # fontsize of the x and y labels
  plt.rc('xtick', labelsize=13)    # fontsize of the tick labels
  plt.rc('ytick', labelsize=13)    # fontsize of the tick labels
  plt.rc('legend', fontsize=13)    # legend fontsize
```

```
  plt.rc('font', size=13)          # controls default text sizes

#error calculation for each and every class
def error_calc(ypred,labels_train,m):
    con=[]
    for i in range(6):
        error=0
        searchval = i
        ii = np.where(labels_train == searchval)[0]
        for i  in range(len(ii)):
            a=ii[i]
            if ypred[a]!=labels_train[a]:
                error=error+1
        con.append(error)
    return con


# In[152]:


init_plot_settings()
trainingerror=[]
ERROR_TRAIN=np.zeros((6,20))
for i in range(20):
    clf = LogisticRegression(max_iter=i)
    clf.fit(X_train,Y_train)
    y_pred_train= clf.predict(X_train)
    y_pred_test=clf.predict(X_test)
    accuracy = accuracy_score(Y_train, y_pred_train)
#     print('Accuracy Score',(1-accuracy)*100)
    trainingerror.append((1-accuracy)*100)

    ERROR_TRAIN[:,i]=error_calc(y_pred_train,Y_train,i)



# In[153]:


plt.figure(figsize=(10, 6))
ax = sns.lineplot(data=ERROR_TRAIN.T[:,:], linewidth=2.5)
ax.set(xlabel='Estimator', ylabel='Error/Loss value', title='Error/Loss vs estimator',xticks=[i for i in
range(ERROR_TRAIN.T.shape[0])])
ax.legend(title='Estimator', title_fontsize = 13, loc=1)
plt.show()


# In[154]:


nb_classes=6
targets = np.array(y_pred_test,dtype=int).reshape(-1)
y_pred_test= np.eye(nb_classes)[targets]
targets = np.array(y_pred_train,dtype=int).reshape(-1)
y_pred_train= np.eye(nb_classes)[targets]
targets = np.array(Y_train,dtype=int).reshape(-1)
Y_train= np.eye(nb_classes)[targets]
targets = np.array(Y_test,dtype=int).reshape(-1)
```

```
Y_test= np.eye(nb_classes)[targets]


# In[155]:


def compute_performance(y, y_pred):

    mean_sensitivity=[]
    mean_specificity=[]
    mean_precision=[]
    mean_f1_score=[]
    g_mean_avg=[]
    total_error = 0

    for i, y_val in enumerate(y):
        if not np.array_equal(y_val,y_pred[i]):
          total_error += 1

    acc_score = 1 - total_error/y.shape[0]

    acc = []
    cf = []
    precision = []
    recall = []


    for i in range(6):
#         print('Class_'+str(i+1))
        tn, fp, fn, tp = confusion_matrix(y[:,i], y_pred[:,i]).ravel()
        sensitivity = tp / (tp+fn)
        mean_sensitivity.append(sensitivity)
        specificity = tn / (tn+fp)
        mean_specificity.append(specificity)
        precision = tp / (tp + fp)
        mean_precision.append(precision)
        f1_score = 2*tp / (2*tp + fp + fn)
        mean_f1_score.append(f1_score)
        g_mean = (sensitivity * specificity) ** (1/2)
        g_mean_avg.append(g_mean)

    mean_sensitivity = [num for num in mean_sensitivity if num<len(Y_train)]
    mean_specificity = [num for num in mean_specificity if num<len(Y_train)]
    mean_precision = [num for num in mean_precision if num<len(Y_train)]
    mean_f1_score = [num for num in mean_f1_score if num<len(Y_train)]
    g_mean_avg = [num for num in g_mean_avg if num<len(Y_train)]

    sensitivity=sum(mean_sensitivity)/len(mean_sensitivity)
    specificity=sum(mean_specificity)/len(mean_specificity)
    precision=sum(mean_precision)/len(mean_precision)
    F1_score=sum(mean_f1_score)/len(mean_f1_score)
    g_mean=sum(g_mean_avg)/len(g_mean_avg)

    print(f"sensitivity : {sensitivity}")
    print(f"specificity : {specificity}")
    print(f"precision   : {precision}")
    print(f"F1_score    : {F1_score}")
    print(f"G_mean      : {g_mean}")
```

In[156]:

```
print('The following are the training dataset performances')
compute_performance(Y_train,y_pred_train )
print('The following are the testing dataset performances')
compute_performance(Y_test,y_pred_test)
```

Wifi dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
```

In[154]:

```
file = pd.read_csv (r'C:\Users\praveen\project_adaboost\wifi.csv')
df = file.values
X = df[:,0:7]
y = df[:,7]-1
```

In[155]:

```
def init_plot_settings():
  # Visualization Reference: https://towardsdatascience.com/a-simple-guide-to-beautiful-visualizations-in-python-f564e6b9d392
  sns.set_style('darkgrid') # darkgrid, white grid, dark, white and ticks
  plt.rc('axes', titlesize=18)     # fontsize of the axes title
  plt.rc('axes', labelsize=14)    # fontsize of the x and y labels
  plt.rc('xtick', labelsize=13)    # fontsize of the tick labels
  plt.rc('ytick', labelsize=13)    # fontsize of the tick labels
  plt.rc('legend', fontsize=13)    # legend fontsize
  plt.rc('font', size=13)          # controls default text sizes

#error calculation for each and every class
def error_calc(ypred,labels_train,m):
    con=[]
    for i in range(4):
        error=0
        searchval = i
        ii = np.where(labels_train == searchval)[0]
        for i  in range(len(ii)):
            a=ii[i]
```

```
            if ypred[a]!=labels_train[a]:
                error=error+1
        con.append(error)
    return con


# In[156]:


init_plot_settings()
X_train, X_test, y_train, y_test = train_test_split(X, y)
trainingerror=[]
ERROR_TRAIN=np.zeros((4,20))
for i in range(20):
    clf = LogisticRegression(max_iter=i)
    clf = clf.fit(X,y)
    clf.fit(X_train,y_train)
    y_pred_train= clf.predict(X_train)
    y_pred_test=clf.predict(X_test)
    accuracy = accuracy_score(y_train, y_pred_train)
#     print('Accuracy Score',(1-accuracy)*100)
    trainingerror.append((1-accuracy)*100)

    ERROR_TRAIN[:,i]=error_calc(y_pred_train,y_train,i)



# In[157]:


plt.figure(figsize=(10, 6))
ax = sns.lineplot(data=ERROR_TRAIN.T[:,:], linewidth=2.5)
ax.set(xlabel='Estimator', ylabel='Error/Loss value', title='Error/Loss vs estimator',xticks=[i for i in
range(ERROR_TRAIN.T.shape[0])])
ax.legend(title='Estimator', title_fontsize = 13, loc=1)
plt.show()


# In[158]:


from sklearn.metrics import confusion_matrix
nb_classes=4
targets = np.array(y_test).reshape(-1)
y_test= np.eye(nb_classes)[targets]
nb_classes=4
targets = np.array(y_train).reshape(-1)
y_train= np.eye(nb_classes)[targets]

targets = np.array(y_pred_train).reshape(-1)
y_pred_train= np.eye(nb_classes)[targets]
targets = np.array(y_pred_test).reshape(-1)
y_pred_test= np.eye(nb_classes)[targets]


# In[159]:
```

```
def compute_performance(y, y_pred):

    mean_sensitivity=[]
    mean_specificity=[]
    mean_precision=[]
    mean_f1_score=[]
    g_mean_avg=[]
    total_error = 0

    for i, y_val in enumerate(y):
        if not np.array_equal(y_val,y_pred[i]):
          total_error += 1

    acc_score = 1 - total_error/y.shape[0]

    acc = []
    cf = []
    precision = []
    recall = []


    for i in range(4):
        tn, fp, fn, tp = confusion_matrix(y[:,i], y_pred[:,i]).ravel()
        sensitivity = tp / (tp+fn)
        mean_sensitivity.append(sensitivity)
        specificity = tn / (tn+fp)
        mean_specificity.append(specificity)
        precision = tp / (tp + fp)
        mean_precision.append(precision)
        f1_score = 2*tp / (2*tp + fp + fn)
        mean_f1_score.append(f1_score)
        g_mean = (sensitivity * specificity) ** (1/2)
        g_mean_avg.append(g_mean)


    sensitivity=sum(mean_sensitivity)/len(mean_sensitivity)
    specificity=sum(mean_specificity)/len(mean_specificity)
    precision=sum(mean_precision)/len(mean_precision)
    F1_score=sum(mean_f1_score)/len(mean_f1_score)
    g_mean=sum(g_mean_avg)/len(g_mean_avg)

    print(f"sensitivity : {sensitivity}")
    print(f"specificity : {specificity}")
    print(f"precision   : {precision}")
    print(f"F1_score : {F1_score}")
    print(f"G_mean : {g_mean}")



# In[160]:



print('The following are the training dataset performances')
compute_performance(y_train,y_pred_train )
print('The following are the testing dataset performances')
compute_performance(y_test,y_pred_test)
```