XG BOOST(Extrem Gradient Boosting)

XGBoost is an [ensemble learning](https://courses.analyticsvidhya.com/courses/ensemble-learning-and-ensemble-learning-techniques?utm_source=blog&utm_medium=an-end-to-end-guide-to-understand-the-math-behind-xgboost) method.

The models that form the ensemble, also known as base learners, could be either from the same learning algorithm or different learning algorithms.

**Bagging and boosting are two widely used ensemble learners**.

### **Bagging**

### **Boosting**

In boosting, the trees are built sequentially such that each subsequent tree aims to reduce the errors of the previous tree. Each tree learns from its predecessors and updates the residual errors. Hence, the tree that grows next in the sequence will learn from an updated version of the residuals

The boosting [ensemble technique](https://courses.analyticsvidhya.com/courses/ensemble-learning-and-ensemble-learning-techniques?utm_source=blog&utm_medium=an-end-to-end-guide-to-understand-the-math-behind-xgboost) consists of three simple steps:

* An initial model F0 is defined to predict the target variable y. This model will be associated with a residual (y – F0)
* A new model h1 is fit to the residuals from the previous step
* Now, F0 and h1 are combined to give F1, the boosted version of F0. The mean squared error from F1 will be lower than that from F0:

Program

'''

The following code is for XGBoost

Created by - ANALYTICS VIDHYA

'''

# importing required libraries

import pandas as pd

from xgboost import XGBClassifier

from sklearn.metrics import accuracy\_score

# read the train and test dataset

train\_data = pd.read\_csv('train-data.csv')

test\_data = pd.read\_csv('test-data.csv')

# shape of the dataset

print('Shape of training data :',train\_data.shape)

print('Shape of testing data :',test\_data.shape)

# Now, we need to predict the missing target variable in the test data

# target variable - Survived

# seperate the independent and target variable on training data

train\_x = train\_data.drop(columns=['Survived'],axis=1)

train\_y = train\_data['Survived']

# seperate the independent and target variable on testing data

test\_x = test\_data.drop(columns=['Survived'],axis=1)

test\_y = test\_data['Survived']

'''

Create the object of the XGBoost model

You can also add other parameters and test your code here

Some parameters are : max\_depth and n\_estimators

Documentation of xgboost:

https://xgboost.readthedocs.io/en/latest/

'''

model = XGBClassifier()

# fit the model with the training data

model.fit(train\_x,train\_y)

# predict the target on the train dataset

predict\_train = model.predict(train\_x)

print('\nTarget on train data',predict\_train)

# Accuray Score on train dataset

accuracy\_train = accuracy\_score(train\_y,predict\_train)

print('\naccuracy\_score on train dataset : ', accuracy\_train)

# predict the target on the test dataset

predict\_test = model.predict(test\_x)

print('\nTarget on test data',predict\_test)

# Accuracy Score on test dataset

accuracy\_test = accuracy\_score(test\_y,predict\_test)

print('\naccuracy\_score on test dataset : ', accuracy\_test)

Both random forest and GBDT build a model consisting of multiple decision trees. The difference is in how the trees are built and combined.

The term “gradient boosting” comes from the idea of “boosting” or improving a single weak model by combining it with a number of other weak models in order to generate a collectively strong model. [**Gradient boosting**](https://developer.nvidia.com/blog/gradient-boosting-decision-trees-xgboost-cuda/) is an extension of boosting where the process of additively generating weak models is formalized as a gradient descent algorithm over an objective function. Gradient boosting sets targeted outcomes for the next model in an effort to minimize errors. Targeted outcomes for each case are based on the gradient of the error (hence the name gradient boosting) with respect to the prediction.