Gradient Boosting Classification:

What is gradient boosting? - Gradient boosting is one of the variants of ensemble methods where you create multiple weak models and combine them to get better performance as a whole. \*Ensemble methods are techniques that aim at improving the accuracy of results in models by combining multiple models instead of using a single model.

***When do we use GB***: Gradient Boosting Algorithm is generally used when we want to decrease the Bias error. GB can be used for both classification and regression.

***Why do we use GB***: Gradient Boosting Algorithm is generally used when we want to decrease the Bias error

What is ***Tree Ensemble Method*** :

It is know that having a single decision tree is said to rarely generalize well to new data. So, one thing we can do is to combine the predictions made by many such decision trees to make increase the prediction accuracy. Hence, considering many such decision trees to make a prediction reduces the variance while it maintains the low bias a single decision tree has. ***So, combining trees is know as Tree Ensemble Method***. One of the main tasks to be done before constructing an ensemble of trees is the process of ***Sampling with replacement***. In short, sampling with replacements helps us to create more training datasets from the given dataset.

***XGBoost*** :

As mentioned above, one of the most commonly used algorithm to implement tree ensemble is ***XGBoost***. It runs quickly, the open source implementations are easily used, has also been used very successfully to win many machine learning competitions as well as in many commercial applications.

***XGBoost Classification*** :

XGBoost works similar to the Random Forest Algorithm(Prior knowledge of random forest is required), but after each iteration, the Algorithm makes it more likely to select the data that are incorrectly classified.

For Example:

Next page

Given below is a data of cats and dogs with their features :

|  |  |  |  |
| --- | --- | --- | --- |
| Ear Shape | Face Shape | Whiskers | Cat |
| Pointy | Round | Present | Yes |
| Floppy | Round | Absent | No |
| Floppy | Round | Absent | No |
| Pointy | Round | Present | Yes |
| Pointy | Not Round | Present | Yes |
| Floppy | Round | Absent | No |
| Floppy | Round | Present | Yes |
| Pointy | Not Round | Absent | No |
| Pointy | Not Round | Absent | No |
| Pointy | Not Round | Present | Yes |

Below is the Constructed Decision tree for above dataset:

Not Round

Round

Not Cat

Present

Absent

Cat

Not Cat

Ear Shape

Whiskers

Result:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ear Shape | Face Shape | Whiskers | Cat | Prediction |
| Pointy | Round | Present | Yes | Correct |
| Floppy | Round | Absent | No | Incorrect |
| Floppy | Round | Absent | No | Correct |
| Pointy | Round | Present | Yes | Correct |
| Pointy | Not Round | Present | Yes | Correct |
| Floppy | Round | Absent | No | Incorrect |
| Floppy | Round | Present | Yes | Correct |
| Pointy | Not Round | Absent | No | Incorrect |
| Pointy | Not Round | Absent | No | Correct |
| Pointy | Not Round | Present | Yes | Correct |

In the above Result we can see that there are 3 incorrect predictions. Hence, in the next iteration , the incorrectly predicted data will be have an higher probability of getting selected.

Code :

from xgboost import XGBClassifier

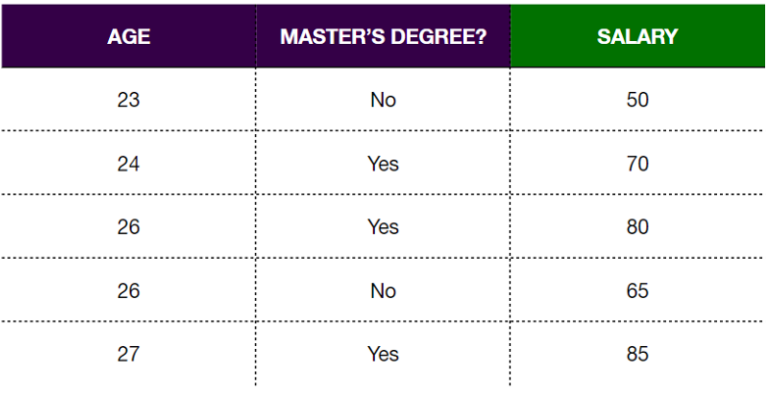
xgbooster = XGBClassifier()

model.fit(y\_Train,y\_Train)

y\_pred = model.predit(X\_test)

***XGBoost Regression***:

Lets take the given below dataset:



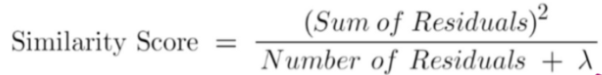
***XGBoost classification and XGBoost regression does not follow same procedure***.

1.We start with an initial prediction. Here we start with the mean = (50+70+80+65+85)/5 =70

Residual = observed – predicted



2.Start building the XGBoost tree. Before building the tree one should know about similarity score



λ (lambda) – Regularization Parameter

-20, 0, 10, -5, 15

Similarity Score = 0 (sum of residuals is 0)

Frist we try split the root node further with criteria ***Masters degree***

-20, 0, 10, -5, 15

Similarity Score – 208.33

Similarity Score – 156.25

Master’s degree - YES

Master’s degree - NO

-20, -5

0, 10, 15

We introduce you to a new term called as Gain. Gain is given by:



Gain for the above tree split = 364.58

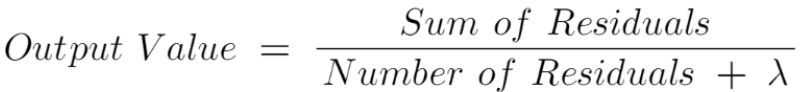
Just like how we split the ***Root***  based on “*If the person has done master degree*”, we can split with ***Age*** also.

When we take feature like age which is real number, there can be several possible criteria which we can use to split. For ex: Age < 25 , Age > 23, Age > 26 etc. We compute gain for all the cases and compare.

***Higher gain value is preferred***

After comparison, we go with master degree at root node as no age criteria could beat the ***gain*** of Master’s. We repeat this with the further nodes until no more criteria left.

Finally, the output can be calculated by:



Code:

from xgboost import XGBRegressor

xgbooster = XGBRegressor ()

model.fit(y\_Train,y\_Train)

y\_pred = model.predit(X\_test)

***Parameters in XGBoost :***

The parameters of XGBoost ar broadly classified into 3 types :

-> ***General parameters*** relate to which booster we are using to do boosting, commonly tree or linear model

-> ***Booster parameters*** depend on which booster you have chosen

-> ***Learning task parameters*** decide on the learning scenario. For example, regression tasks may use different parameters with ranking tasks.

There are a lot of parameters and they pretty much self explanatory. You can refer them [here](https://xgboost.readthedocs.io/en/stable/parameter.html).

***CATBoost:***

CATBoost is an algorithm for gradient boosting on decision trees. It is developed by Yandex researchers and engineers, and is used for search, recommendation systems, personal assistant, self-driving cars, weather prediction and many other tasks at Yandex and in other companies, including CERN, Cloudflare, Careem taxi (Subsidiary of Uber).

CATBoost has a wide range of options. It can be used in Regression, Classification, Ranking, Recommendation system etc.

A prediction model F obtained after several steps of boosting relies on the targets of all training examples . It is statistically proven that this causes a ***“Prediction Shift”***. A prediction shift happens when there is a shift between the predicted model trained on the training samples and the distribution of the test samples. This often occurs as the predicted model is trained using all training samples and not all training samples are fully representative of the test samples. *Prediction Shift may lead to overfitting in some cases*.

Catboost introduces two critical algorithmic advances over other gradient boosting algorithms– the implementation of ordered boosting, a permutation-driven alternative to the classic algorithm, and an innovative algorithm for processing categorical features.

CatBoost uses Symmetric trees because it more flexible towards parameter changes. Symmetric trees are less prone to overfitting.

***Ordered Boosting:***

Ordered boosting works by creating random permutations of the training data and splitting them into two parts: a learn part and a test part. The learn part is used to train a model, while the test part is used to calculate the residuals or errors for each data instance. [The residuals are then used as new target values for the next model](https://stackoverflow.com/questions/65479463/understanding-catboost-ordered-boosting" \t "/home/meghadharsan/Documents\\x/_blank). This process is repeated for each permutation until all models are trained.

[The idea behind ordered boosting is to avoid using the same data instances for both training and calculating residuals, which can cause overfitting due to prediction shift](https://neptune.ai/blog/when-to-choose-catboost-over-xgboost-or-lightgbm" \t "/home/meghadharsan/Documents\\x/_blank). By using different subsets of data for each model, ordered boosting simulates how a model would perform on unseen data.

***Parameters:***

There are more than 50 parameters available which can be used for training the mode. You find them all [here](https://catboost.ai/en/docs/references/training-parameters/).