

Ensemble Methods

Week 2 - Boosting

Discussion Questions

1. What are the key concepts of Boosting?
2. What is the difference between bagging and boosting?
3. What are the different types of boosting algorithms?
4. What are some important hyperparameters of boosting algorithms?

What are the key concepts of Boosting?

- It builds/trains weak learners sequentially.
- Weights of each misclassified point (red cross samples) are increased and weights for each correctly classified (green ticked samples) are decreased for the subsequent learner.
- The misclassified points of the previous model are given more weightage by the next model.
- The same process will be repeated until the stopping criteria are reached.
- The final prediction is decided by taking the weighted average or voting.



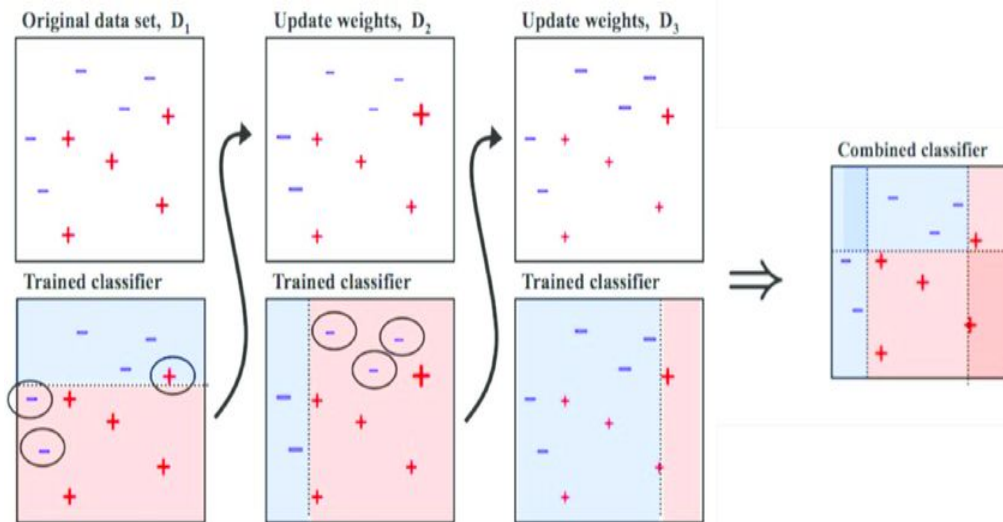
Source: [https://en.wikipedia.org/wiki/Boosting_\(machine_learning\)#/media/File:Ensemble_Boosting.svg](https://en.wikipedia.org/wiki/Boosting_(machine_learning)#/media/File:Ensemble_Boosting.svg)

What is the difference between bagging and boosting?

Bagging	Boosting
Parallel Training - All the weak learners are built in parallel i.e. independent of each other.	Sequential Training - Successive weak learners to improve the accuracy from the prior learners
Equal weightage - Each weak learner has equal weight in the final prediction.	Weighted average - More weight to those weak learners with better performance
Independent samples - Samples are drawn from the original dataset with replacement to train each weak learner	Dependent samples - Subsequent samples have more of those observations which had relatively higher errors in previous weak learners
Can help reduce the variance of the model	Can help reduce the bias of the model
Example: Bagging Classifier, Random Forest	Example: AdaBoost, Gradient Boosting Classifier

Different types of boosting algorithms - AdaBoost

1. Initially, it assigns equal sample weights for each sample.
2. We bootstrap the samples as per the weights assigned and build a weak learner on that sample
3. Once the weak learner is built, AdaBoost measures the importance of a weak learner based on the error made by that weak learner.
4. New sample weight according to the correct and incorrect predictions are assigned to each sample and a new bootstrapped dataset is created with the odds of each sample being chosen based on their new sample weights
5. The process is repeated n number of times.
6. The final prediction is a weighted majority vote/average of all the weak learners



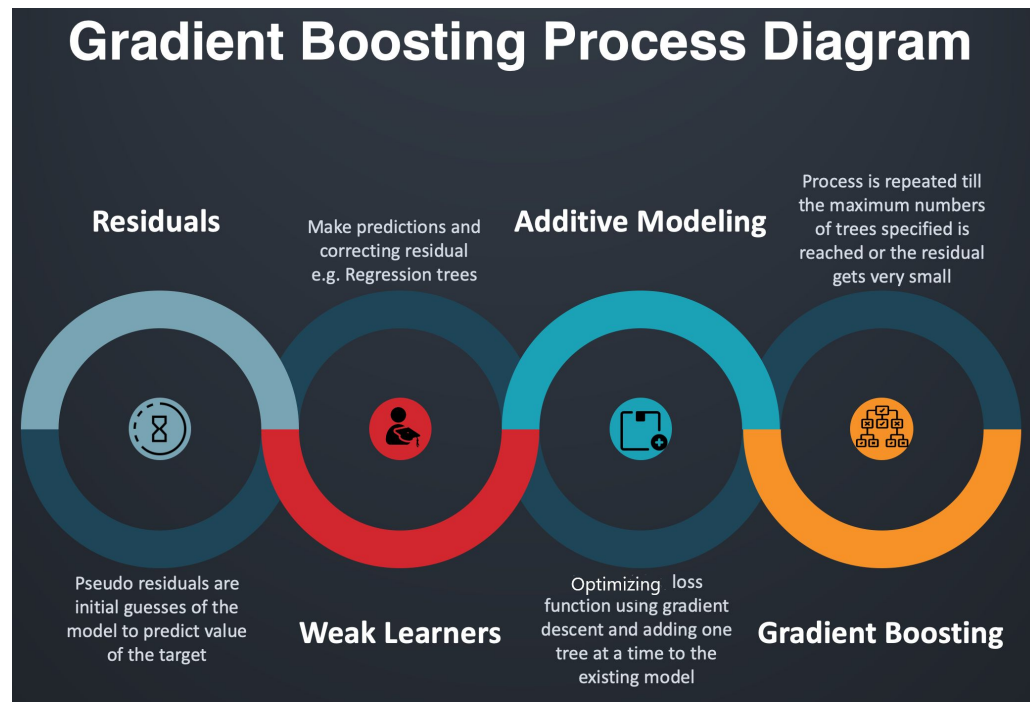
Source: https://www.researchgate.net/figure/Training-of-an-AdaBoost-classifier-The-first-classifier-trains-on-unweighted-data-then_fig3_306054843

Some important hyperparameters of AdaBoost (SKlearn)

- **base_estimator:** The base estimator from which the boosted ensemble is built. By default, the base estimator is a decision tree with `max_depth=1`
- **n_estimators:** The maximum number of estimators at which boosting is terminated. The default value is 50.
- **learning_rate:** The learning rate shrinks the contribution of each classifier by `learning_rate`. There is a trade-off between `learning_rate` and `n_estimators`.

Different types of boosting algorithms - Gradient Boost

1. A weak learner is built on a subset of the original data.
2. Errors are calculated after predictions by the weak learner by comparing the predictions and actual values.
3. The main difference between AdaBoost and Gradient Boosting is that AdaBoost changes the weights of the sample after prediction by each weak learner while Gradient Boosting fits the next weak learner to the residuals of the previous weak learner.
4. The residuals become the target variable for the new weak learner. The main aim is to minimize the residuals i.e. the error made by the previous weak learner or regularizing the loss function
5. The same process is repeated until there is no reduction in the residuals, or the number of estimators is reached.



Source: <https://dzone.com/articles/xgboost-a-deep-dive-into-boosting>

Some important hyperparameters of GBM (SKlearn)

- **Init:** An estimator object that is used to compute the initial predictions. If 'zero', the initial raw predictions are set to zero. By default, a DummyEstimator predicting the classes prior is used.
- **Learning_rate:** *The* learning rate shrinks the contribution of each tree by learning_rate. There is a trade-off between learning_rate and n_estimators.**default=0.1**
- **N_estimators:** The number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance, **default=100**
- **Subsample:** The fraction of samples to be used for fitting the individual base learner,**default=1.0**
- **max_features:** The number of features to consider when looking for the best split, **default=None**

Different types of boosting algorithms - XGBoost

- XGBoost builds upon the idea of Gradient Boosting with some modifications to have high performance and faster implementation than gradient boosting and Adaboost algorithm.
- It has a variety of in-built features like handling missing values and different hyperparameters to reduce overfitting and improve overall performance.
- Steps involved in XGBoost-
 1. XGBoost sets the initial prediction for all observations as the mean of the target values for regression and 0.5 for binary classification problems.
 2. Then it calculates the residual for each observation.
 3. It builds a tree to predict the residuals and
 4. The main difference in terms of building a tree between xgboost and gradient boosting is how they decide the best split at each level. While gradient boosting traditional splitting criterion like Gini or entropy, xgboost uses gain calculated from similarity score to find the best split which is directly
 5. After a tree is built, it calculates the output value for each leaf node.
 6. The same process from steps 2-4 is repeated until there is no reduction in the residuals, or the number of estimators is reached.

Some important hyperparameters of XGBoost

- **Learning_rate / eta:** Gradient boosted decision trees are quick to learn and overfit training data. One effective way to slow down learning in the model is to use a learning rate, also called shrinkage.
- **gamma:** A node is split only when the resulting split gives a positive reduction in the loss function. Gamma specifies the minimum loss reduction required to make a split. Higher the gamma value, the lesser the chances of overfitting
- **scale_pos_weight:** Control the balance of positive and negative weights. It can have any positive value as input. It helps in imbalanced classification problems.

Some important hyperparameters of XGBoost

XGBoost has hyperparameters to select a subset of columns while building an individual tree, level in a tree, and node split in a level in a tree:


- **colsample_bytree:** This denotes the ratio of columns to be used when constructing each tree. It takes input as float values between (0,1] i.e. between greater than 0% of features and less than or equal to 100% of features.
- **colsample_bylevel:** This denotes the ratio of columns for each new level of a tree. If `colsample_bytree` is used, then columns are subsampled from the set of columns chosen for the current tree.
- **colsample_bynode:** This denotes the ratio of columns to be used for each node i.e. for each split. If `colsample_bylevel` is used, then columns are subsampled from the set of columns chosen for the current level.

Note: Three parameters defined above works cumulatively, for example, if we have total 32 columns and if `{'colsample_bytree':0.5, 'colsample_bylevel':0.5, 'colsample_bynode':0.5}`, then we leave only 4 columns to choose from at each split.

APPENDIX

How a tree is built in XGBoost?

1. At each level, XGBoost calculates a **similarity score(ss)** for the node and the possible two leaves.

SS for Regression 

$$\frac{(\sum Residuals)^2}{Number\ of\ Samples + \lambda}$$

SS for Classification 

$$\frac{(\sum Residuals)^2}{\sum [Probability \times (1 - Probability)] + \lambda}$$

2. The formula for similarity score is derived directly from the loss function and helps to find the optimal point which minimizes the loss function, Where λ is a regularization parameter in loss function of XGBoost
3. Gain is calculated for each possible split, where the gain is calculated as **Left leaf ss + Right leaf ss - Root node ss**
4. The split which maximizes the gain is considered as the best split and is chosen to grow a tree.

greatlearning
Power Ahead

Happy Learning !

