|  |  |
| --- | --- |
| Gradient Boosting   * **Gradient boosting** is a technique for building an [ensemble](https://blog.statsbot.co/ensemble-learning-d1dcd548e936) of weak models such that the predictions of the ensemble minimize a loss function. * gradient boosting descends the gradient by introducing new models. * Both algorithms descend the gradient of a differentiable loss function. | Gradient Descent   * **Gradient descent** is an algorithm for finding a set of parameters that optimizes a loss function. Given a loss function *f*(*x*,*ϕ*), where *x* is an n-dimensional vector and *ϕ* is a set of parameters, gradient descent operates by computing the gradient of *f* with respect to *ϕ*. * It then "descends" the gradient by nudging the parameters in the opposite direction of the gradient. This process is repeated for different points in the space of inputs (i.e. different *x*s) until a minimum of *f* is found. * Gradient descent "descends" the gradient by introducing changes to parameters * Both algorithms descend the gradient of a differentiable loss function. |

|  |  |
| --- | --- |
| **Gradient Boosting** | **AdaBoost** |
| * This approach trains learners based upon minimising the loss function of a learner (i.e., training on the residuals of the model) * Weak learners are decision trees constructed in a greedy manner with split points based on purity scores (i.e., Gini, minimise loss). Thus, larger trees can be used with around 4 to 8 levels. Learners should still remain weak and so they should be constrained (i.e., the maximum number of layers, nodes, splits, leaf nodes) * All the learners have equal weights in the case of gradient boosting. The weight is usually set as the learning rate which is small in magnitude. * In Gradient Boosting, ‘shortcomings’ (of existing weak learners) are identified by **gradients**. | * This method focuses on training upon misclassified observations. Alters the distribution of the training dataset to increase weights on sample observations that are difficult to classify. * The weak learners incase of adaptive boosting are a very basic form of decision tree known as stumps. * The final prediction is based on a majority vote of the weak learners’ predictions weighted by their individual accuracy. * In Adaboost, ‘shortcomings’ are identified by **high-weight data points**. |

|  |  |
| --- | --- |
| **GBM (Gradient Boosting Machine)** | **XGBoost (Extreme Gradient Boosting)** |
| * The gradient is used to minimize the loss function (error - difference between the actual values and predicted values). * It is basically the partial derivative of the loss function, so it describes the steepness of our error function. * In each round of training, the weak learner is built and its predicted values are compared to the actual values. * The distance or difference between the prediction and reality represents the error rate of our model. * Take the derivative (gradient) of the Loss Function (error) of each parameter. Calculate the Step Size and Learning Rate and calculate new parameters based on that. * In this way, you will create a new Weak Learner. Keep repeating the steps (descending the gradient) and keep generating the new learners until Step Size is very small or maximum number of steps are completed. * By using gradient descent and updating our predictions based on a learning rate (the “step size” with which we descend the gradient), we can find the values where loss function is minimum. * So, we are basically updating the predictions such that the sum of our residuals is close to 0 (or minimum) and predicted values are sufficiently close to actual values. | * XGBoost stands for **Extreme Gradient Boosting**. XGBoost is a specific **implementation of the Gradient Boosting** method which delivers more accurate approximations by using the strengths of **second order derivative of the loss function**, **L1 and L2 regularization** and **parallel computing**. * XGBoost is more **regularized form of Gradient Boosting**. XGBoost uses advanced regularization (L1 & L2), which improves model generalization capabilities. * XGBoost delivers high performance as compared to Gradient Boosting. Its training is very fast and can be **parallelized / distributed across clusters**. * XGBoost computes second-order gradients, i.e. **second partial derivatives of the loss function**, which provides more information about the direction of gradients and how to get to the minimum of our loss function. * XGBoost also **handles missing values** in the dataset. So, in data wrangling, you may or may not do a separate treatment for the missing values, because XGBoost is capable of handling missing values internally.   Features of XGBoost are:   * Clever Penalisation of Trees * A Proportional shrinking of leaf nodes * [Newton Boosting](https://en.wikipedia.org/wiki/Newton%27s_method_in_optimization) * Extra Randomisation Parameter * In XGBoost the trees can have a varying number of terminal nodes and left weights of the trees that are calculated with less evidence is shrunk more heavily. * Newton Boosting uses Newton-Raphson method of approximations which provides a direct route to the minima than gradient descent. * The extra randomisation parameter can be used to reduce the correlation between the trees, the lesser the correlation among classifiers, the better our ensemble of classifiers. * Generally, XGBoost is faster than gradient boosting but gradient boosting has a wide range of application |

**How is Gradient Boosting interpreted as an optimisation problem?**

* We take up a weak learnerand at each step, we add another weak learner to increase the performance and build a strong learner. This reduces the loss of the loss function.
* We iteratively add each model and compute the loss. The loss represents the error residuals(the difference between actual value and predicted value) and using this loss value the predictions are updated to minimise the residuals.