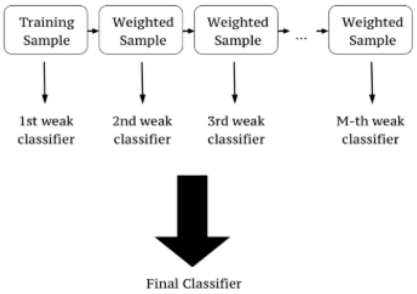
1. What is Boosting?

Boosting

Boosting is an ensemble approach(meaning it involves several trees) that starts from a weaker decision and keeps on building the models such that the final prediction is the weighted sum of all the weaker decision-makers. The weights are assigned based on the performance of an individual tree.



Ensemble parameters are calculated in **stagewise way** which means that while calculating the subsequent weight, the learning from the previous tree is considered as well.

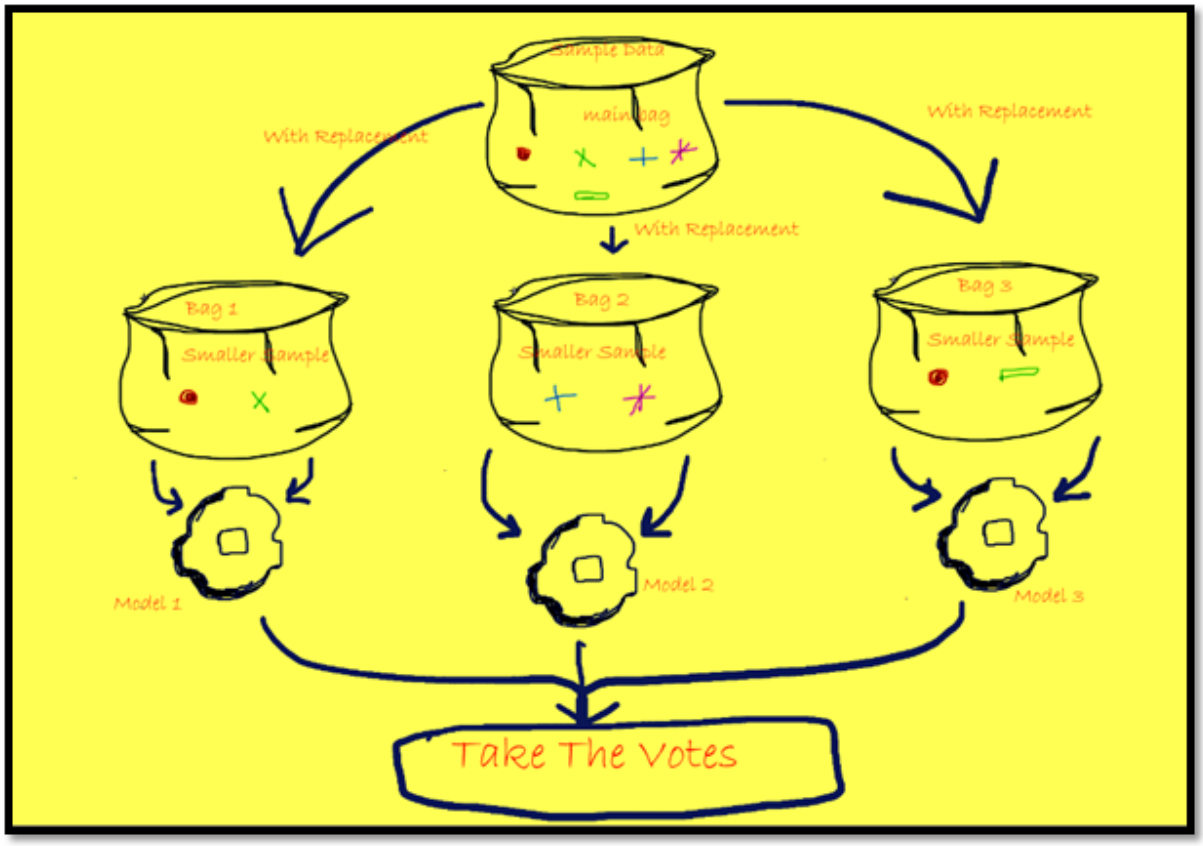
2. How do boosting and bagging differ?

Bagging: It is the method to decrease the variance of model by generating additional data for training from your original data set using combinations with repetitions to produce multisets of the same size as your original data.

Boosting: It helps to calculate the predict the target variables using different models and then average the result(may be using a weighted average approach).

Bagging

Partitioning of data	Random
Goal to achieve	Minimum variance
Methods used	Random subspace
Functions to combine single model	Weighted average
Example	Random Forest

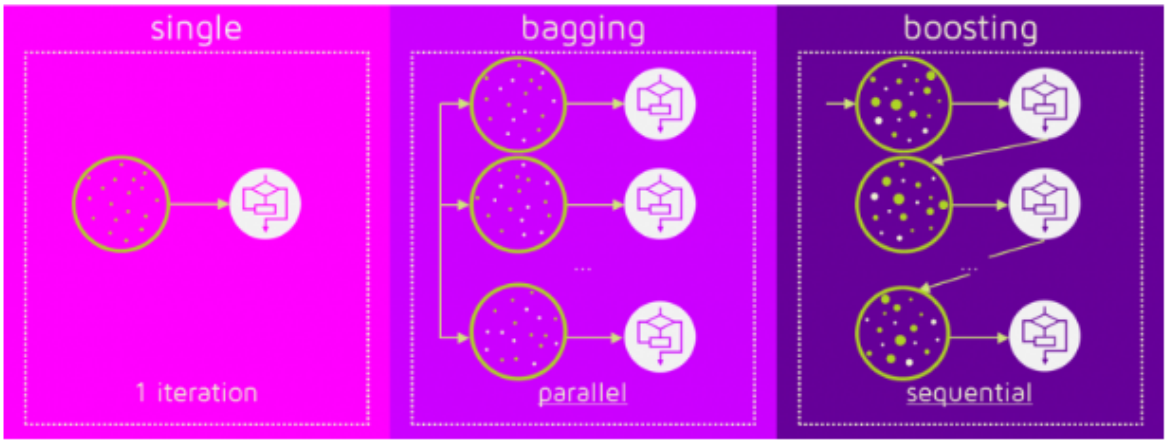


1. What Is Bagging?

- Bagging, which is also known as bootstrap aggregating sits on top of the majority voting principle.
- The samples are bootstrapped each time when the model is trained. When the samples are chosen, they are used to train and validate the predictions. The samples are then replaced back into the training set. The samples are selected at random. This technique is known as bagging.
- To sum up, base classifiers such as decision trees are fitted on random subsets of the original training set. Subsequently, the individual predictions are aggregated (voting or averaging etc). The final results are then used as predictions. It reduces the variance of a black box estimator.

What is Boosting?

The term 'Boosting' refers to a family of algorithms which converts weak learner to strong learners. Boosting is an ensemble method for improving the model predictions of any given learning algorithm. The idea of boosting is to train weak learners sequentially, each trying to correct its predecessor.



Source: Quantdare

2. What Is Boosting?

- The concept of Adaptive Boost revolves around correcting previous classifier mistakes. Each classifier gets trained on the sample set and learns to predict. The misclassification errors are then fed into the next classifier in the chain and are used to correct the mistakes until the final model predicts accurate results.
- This is slightly different than boosting whereby the samples are trained by a classifier and then the errors are used to correct the mistakes.
- The core concept of boosting focuses on those specific training samples that are hard to classify. These training samples are known to be misclassified by the classifiers. These models (classifiers) are known as the weak classifiers.
- When a weak-classifier misclassifies a training sample, the algorithm then uses these very samples to improve the performance of the ensemble.
- The entire training set is fed in the AdaBoost. Eventually, a strong classifier is built that learns from the mistakes of the previous weak learners in the ensemble.

Boosting:

Boosting is another ensemble procedure to make a collection of predictors. In other words, we fit consecutive trees, usually random samples, and at each step, the objective is to solve net error from the prior trees.

If a given input is misclassified by theory, then its weight is increased so that the upcoming hypothesis is more likely to classify it correctly by consolidating the entire set at last converts weak learners into better performing models.

Gradient Boosting is an expansion of the boosting procedure.

> Gradient Boosting = Gradient Descent + Boosting

It utilizes a gradient descent algorithm that can optimize any differentiable loss function. An ensemble of trees is constructed individually, and individual trees are summed successively. The next tree tries to restore the loss (It is the difference between actual and predicted values).

Advantages of using Gradient Boosting methods:

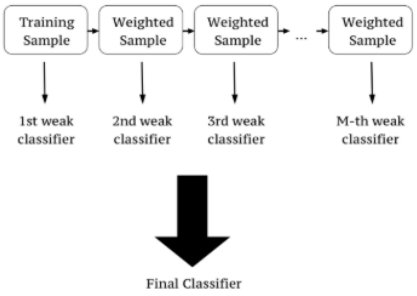
- It supports different loss functions.
- It works well with interactions.

Boosting

Partitioning of data	Higher vote to misclassified samples
Goal to achieve	Increase accuracy
Methods used	Gradient descent
Functions to combine single model	Weighted majority vote
Example	Ada Boost

Boosting

Boosting is an ensemble approach(meaning it involves several trees) that starts from a weaker decision and keeps on building the models such that the final prediction is the weighted sum of all the weaker decision-makers. The weights are assigned based on the performance of an individual tree.



Ensemble parameters are calculated in **stagewise way** which means that while calculating the subsequent weight, the learning from the previous tree is considered as well.

Bagging

Bagging is used when our objective is to reduce the variance of a decision tree. Here the concept is to create a few subsets of data from the training sample, which is chosen randomly with replacement. Now each collection of subset data is used to prepare their decision trees thus, we end up with an ensemble of various models. The average of all the assumptions from numerous tress is used, which is more powerful than a single decision tree.

Random Forest is an expansion over bagging. It takes one additional step to predict a random subset of data. It also makes the random selection of features rather than using all features to develop trees. When we have numerous random trees, it is called the Random Forest.

These are the following steps which are taken to implement a Random forest:

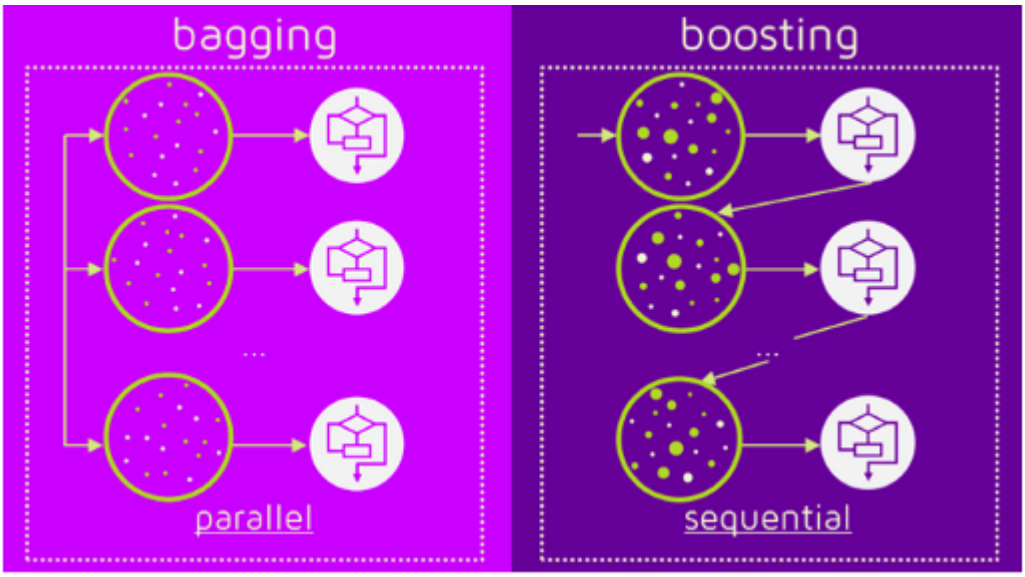
- Let us consider **X** observations **Y** features in the training data set. First, a model from the training data set is taken randomly with substitution.
- The tree is developed to the largest.
- The given steps are repeated, and prediction is given, which is based on the collection of predictions from n number of trees.

Advantages of using Random Forest technique:

- It manages a higher dimension data set very well.
- It manages missing quantities and keeps accuracy for missing data.

Disadvantages of using Random Forest technique:

Since the last prediction depends on the mean predictions from subset trees, it won't give precise value for the regression model.



3. Difference Between Bagging And Boosting

1. Bagging technique can be an effective approach to reduce the variance of a model, to prevent over-fitting and to increase the accuracy of unstable models.
2. On the other hand, Boosting enables us to implement a strong model by combining a number of weak models together.
3. In contrast to bagging, samples drawn from the training dataset are not replaced back into the training set during the boosting exercise.
4. If you analyse the decision boundaries, known as stumps, that are computed by the Adaptive Boosting algorithm when compared with the Decision trees, you will note that the decision boundaries computed by AdaBoost can be very sophisticated.
5. Although this can help us implement a strong predictive model, the ensemble learning increases the computational complexity compared to individual classifiers.

Differences Between Bagging and Boosting –

S.NO	BAGGING	BOOSTING
1.	Simplest way of combining predictions that belong to the same type.	A way of combining predictions that belong to the different types.
2.	Aim to decrease variance, not bias.	Aim to decrease bias, not variance.
3.	Each model receives equal weight.	Models are weighted according to their performance.
4.	Each model is built independently.	New models are influenced by performance of previously built models.
5.	Different training data subsets are randomly drawn with replacement from the entire training dataset.	Every new subsets contains the elements that were misclassified by previous models.
6.	Bagging tries to solve over-fitting problem.	Boosting tries to reduce bias.
7.	If the classifier is unstable (high variance), then apply bagging.	If the classifier is stable and simple (high bias) the apply boosting.
8.	Random forest.	Gradient boosting.

Bagging:

1. **parallel** ensemble: each model is built independently
2. aim to **decrease variance**, not bias
3. suitable for high variance low bias models (complex models)
4. an example of a tree based method is **random forest**, which develop fully grown trees (note that RF modifies the grown procedure to reduce the correlation between trees)

Boosting:

1. **sequential** ensemble: try to add new models that do well where previous models lack
2. aim to **decrease bias**, not variance
3. suitable for low variance high bias models
4. an example of a tree based method is **gradient boosting**

Bagging:
1. resamples training data to get M subsets (bootstrapping);
2. trains M classifiers(same algorithm) based on M datasets(different samples);
3. final classifier combines M outputs by voting;
samples weight equally;
classifiers weight equally;
decreases error by decreasing the variance
Boosting: here focus on adaboost algorithm
1. start with equal weight for all samples in the first round;
2. in the following M-1 rounds, increase weights of samples which are misclassified in last round, decrease weights of samples correctly classified in last round
3. using a weighted voting, final classifier combines multiple classifiers from previous rounds, and give larger weights to classifiers with less misclassifications.
step-wise reweights samples; weights for each round based on results from last round
re-weight samples(boosting) instead of resampling(bagging).

> These two are the most important terms describing the ensemble (combination) of various models into one more effective model.
>
> - **Bagging** to decrease the model's variance.
> - **Boosting** to decreasing the model's bias.
>
> Most of the errors from a model's learning are from three main factors: variance, noise, and bias. By using ensemble methods, we're able to increase the stability of the final model and reduce the errors mentioned previously. By combining many models, we're able to (mostly) reduce the variance, even when they are individually not great, as we won't suffer from random errors from a single source.

Which is the best, Bagging or Boosting?

There's not an outright winner; it depends on the data, the simulation and the circumstances. Bagging and Boosting decrease the variance of your single estimate as they combine several estimates from different models. So the result may be a model with **higher stability.**

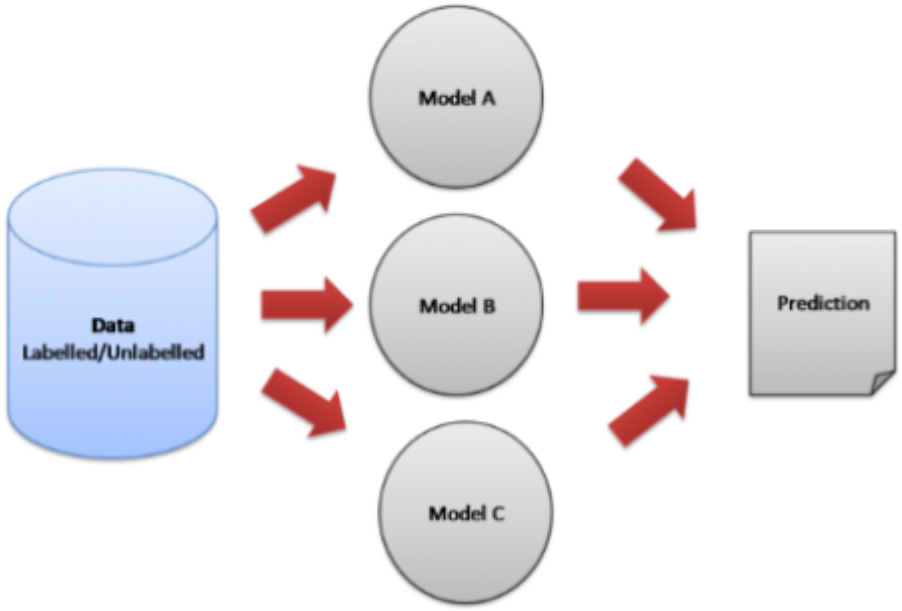
If the problem is that the single model gets a very low performance, Bagging will rarely get a **better bias.** However, Boosting could generate a combined model with lower errors as it optimises the advantages and reduces the pitfalls of the single model.

By contrast, if the difficulty of the single model is **over-fitting**, then Bagging is the best option. Boosting for its part doesn't help to avoid over-fitting; in fact, this technique is faced with this problem itself. For this reason, Bagging is effective more often than Boosting.

Bagging	Boosting
Various training data subsets are randomly drawn with replacement from the whole training dataset.	Each new subset contains the components that were misclassified by previous models.
Bagging attempts to tackle the over-fitting issue.	Boosting tries to reduce bias.
If the classifier is unstable (high variance), then we need to apply bagging.	If the classifier is steady and straightforward (high bias), then we need to apply boosting.
Every model receives an equal weight.	Models are weighted by their performance.
Objective to decrease variance, not bias.	Objective to decrease bias, not variance.
It is the easiest way of connecting predictions that belong to the same type.	It is a way of connecting predictions that belong to the different types.
Every model is constructed independently.	New models are affected by the performance of the previously developed model.

Bagging and boosting are commonly used terms by various data enthusiasts around the world. But what exactly bagging and boosting mean and how does it help the data science world. From this post you will learn about bagging, boosting and how they are used.

Both Bagging and boosting are part of a series of statistical techniques called ensemble methods.



Ensemble model

To recap in short, **Bagging** and **Boosting** are normally used inside one algorithm, while **Stacking** is usually used to summarize several results from different algorithms.

- **Bagging**: **Bootstrap** subsets of features and samples to get several predictions and average(or other ways) the results, for example, `Random Forest`, which eliminate variance and does not have overfitting issue.
- **Boosting**: The difference from **Bagging** is that later model is trying to learn the error made by previous one, for example `GBM` and `XGBoost`, which eliminate the variance but have overfitting issue.
- **Stacking**: Normally used in competitions, when one uses multiple algorithms to train on the same data set and average(max, min or other combinations) the result in order to get a higher accuracy of prediction.

All three are so-called "meta-algorithms": approaches to combine several machine learning techniques into one predictive model in order to decrease the variance (*bagging*), bias (*boosting*) or improving the predictive force (*stacking* alias *ensemble*).

Every algorithm consists of two steps:

1. Producing a distribution of simple ML models on subsets of the original data.
2. Combining the distribution into one "aggregated" model.

Here is a short description of all three methods:

1. **Bagging** (stands for **B**ootstrap **Agg**regat**ing**) is a way to decrease the variance of your prediction by generating additional data for training from your original dataset using combinations with repetitions to produce multisets of the same cardinality/size as your original data. By increasing the size of your training set you can't improve the model predictive force, but just decrease the variance, narrowly tuning the prediction to expected outcome.
2. **Boosting** is a two-step approach, where one first uses subsets of the original data to produce a series of averagely performing models and then "boosts" their performance by combining them together using a particular cost function (=majority vote). Unlike bagging, in the classical boosting the subset creation is not random and depends upon the performance of the previous models: every new subsets contains the elements that were (likely to be) misclassified by previous models.
3. **Stacking** is a similar to boosting: you also apply several models to your original data. The difference here is, however, that you don't have just an empirical formula for your weight function, rather you introduce a meta-level and use another model/approach to estimate the input together with outputs of every model to estimate the weights or, in other words, to determine what models perform well and what badly given these input data.

3. What are week and strong classifiers?

A 'weak' learner (classifer, predictor, etc) is just one which performs relatively poorly--its accuracy is above chance, but just barely. There is often, but not always, the added implication that it is computationally simple. Weak learner also suggests that many instances of the algorithm are being pooled (via boosting, bagging, etc) together into to create a "strong" ensemble classifier.

Need Of Weak Learners

Weak learner is a learner that no matter what the distribution over the training data is will always do better than chance, when it tries to label the data. Doing better than chance means we are always going to have an error rate which is less than 1/2.

This means that the learner algorithm is always going to learn something, not always completely accurate i.e., it is weak and poorwhen it comes to learning the relationships between X X (inputs) and Y Y (target).

This eventually improves the weak learners and converts them to strong learners.

Why do people always use trees for gradient boosting? Because trees are very improved by this procedure. Boosting preserve some good things about trees and make them competitive with Neural Networks You can use another weak learners too. But remember you should have fast way to pull one of your weak learner to predict generalized residuals. Because trees are rather well-understanable thing you can create rather various schemas of boosting

4. Why are trees deemed fit for boosting?

Weak classifier - why tree?

First what is a weak classifier? **Weak classifier** - *slightly better* than random guessing.

Any algorithm could have been used as a base for the boosting technique, but the reason for choosing trees are:

Pro's

- computational scalability,
- handles missing values,
- robust to outliers,
- does not require feature scaling,
- can deal with irrelevant inputs,
- interpretable (if small),
- handles mixed predictors as well (quantitive and qualitative)

Con's

- inability to extract a linear combination of features
- high variance leading to a small computational power

And that's where boosting comes into the picture. It minimises the variance by taking into consideration the results from various trees.

In every machine learning model, the training objective is a sum of a loss function L and regularisation Ω:

$$obj = L + \Omega$$

The loss function controls the predictive power of an algorithm and the regularisation term controls its simplicity.

There are several algorithms which use boosting. A few are discussed here.

5. Explain the step by step implementation of ADA Boost.

Ada Boost (Adaptive Boosting)

The steps to implement the Ada Boost algorithm using the decision trees are as follows:

Algorithm:

Assume that the number of training samples is denoted by N, and the number of iterations (created trees) is M. Notice that possible class outputs are $Y = \{-1, 1\}$

1. Initialize the observation weights $\omega_i = \frac{1}{N}$ where $i = 1, 2, \dots, N$ for all the samples.
2. For $m = 1$ to M:
   - fit a classifier $G_m(x)$ to the training data using weights ω_i,
   - compute $err_m = \frac{\sum_{i=1}^{N} \omega_i I(y_i \neq G_m(x))}{\sum_{i=1}^{N} \omega_i}$,
   - compute $\alpha_m = \frac{1}{2}\log(\frac{(1-err_m)}{err_m})$. This is the contribution of that tree to the final result.
   - calculate the new weights using the formula:

     $\omega_i \leftarrow \omega_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x)]$, where $i = 1, 2, \dots, N$
3. Normalize the new sample weights so that their sum is 1.
4. Construct the next tree using the new weights

3. At the end, compare the summation of results from all the trees and the final result is either the one with the highest sum(for regression) or it is the class which has the most weighted voted average(for classification).

   Output $G_m(x) = argmax[\sum_{m=1}^{M} \alpha_m G_m(x)]$ (Regression)

   Output $G_m(x) = sigm[\sum_{m-1}^{M} \alpha_m G_m(x)]$ (Classification)

6. What are pseudo residuals?

Gradient Boosted Trees

Gradient Boosted Trees use decision trees as estimators. It can work with different loss functions (regression, classification, risk modelling etc.), evaluate it's gradient and approximates it with a simple tree (stage-wisely, that minimizes the overall error).

AdaBoost is a special case of Gradient Boosted Tree that uses exponential loss function.

The Algorithm:

- Calculate the average of the label column as initially this average shall minimise the total error.
- Calculate the pseudo residuals.

```
Pseudo residual= actual label- the predicted result (which is average in the first iteration)
```

Mathematically,

derivative of the pseudo residual=$(\frac{\delta L(y_i,f(x_i))}{\delta(f(x_i))})$

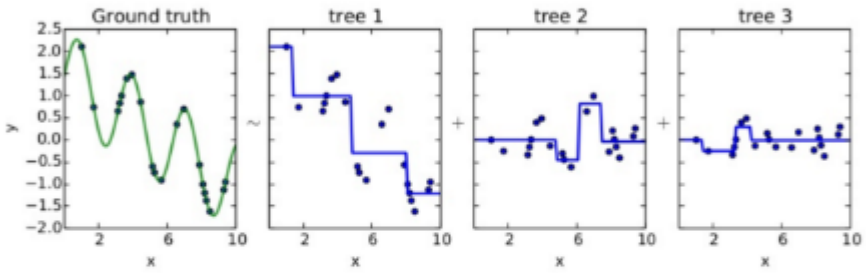
where, L is the loss function.

```
Here, the gradient of the error term is getting calculated as the goal is to minimize the error.
Hence the name gradient boosted trees
```

7. Explain the step by step implementation of Gradient boosted trees.

Gradient Boosting

Just like AdaBoost, Gradient Boosting works by sequentially adding predictors to an ensemble, each one correcting its predecessor. However, instead of changing the weights for every incorrect classified observation at every iteration like AdaBoost, Gradient Boosting method tries to fit the new predictor to the residual errors made by the previous predictor.



Source: Quora

GBM uses Gradient Descent to find the shortcomings in the previous learner's predictions. GBM algorithm can be given by following steps.

- Fit a model to the data, $F1(x) = y$
- Fit a model to the residuals, $h1(x) = y - F1(x)$
- Create a new model, $F2(x) = F1(x) + h1(x)$
- By combining weak learner after weak learner, our final model is able to account for a lot of the error from the original model and reduces this error over time.

Gradient Boosted Trees

Gradient Boosted Trees use decision trees as estimators. It can work with different loss functions (regression, classification, risk modelling etc.), evaluate it's gradient and approximates it with a simple tree (stage-wisely, that minimizes the overall error).

AdaBoost is a special case of Gradient Boosted Tree that uses exponential loss function.

The Algorithm:

- Calculate the average of the label column as initially this average shall minimise the total error.
- Calculate the pseudo residuals.

```
Pseudo residual= actual label- the predicted result (which is average in the first iteration)
```

Mathematically,

derivative of the pseudo residual=$(\frac{\delta L(y_i, f(x_i))}{\delta(f(x_i))})$

where, L is the loss function.

```
Here, the gradient of the error term is getting calculated as the goal is to minimize the error.
Hence the name gradient boosted trees
```

- create a tree to predict the pseudo residuals instead of a tree to predict for the actual column values.
- new result= previous result+learning rate* residual

Mathematically, $F_1(x) = F_0(x) + \nu \sum \gamma$

where ν is the learning rate and γ is the residual

Repeat these steps until the residual stops decreasing

8. Explain the step by step implementation of XGBoost Algorithm.

XGBoost

XGBoost improves the gradient boosting method even further.

> **XGBoost** (*extreme gradient boosting*) regularises data better than normal gradient boosted Trees.

It was developed by Tianqi Chen in C++ but now has interfaces for Python, R, Julia.

XGBoost's objective function is the sum of loss function evaluated over all the predictions and a regularisation function for all predictors (j trees). In the formula f_j means a prediction coming from the j^th tree.

$$obj(\theta) = \sum_{i}^{n} l(y_i - \hat{y_i}) + \sum_{j=1}^{j} \Omega(f_j)$$

Loss function depends on the task being performed (classification, regression, etc.) and a regularization term is described by the following equation:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} \omega_j^2$$

First part (γT) is responsible for controlling the overall number of created leaves, and the second term ($\frac{1}{2}\lambda \sum_{j=1}^{T} \omega_j^2$) watches over the scores.

Mathematics Involved Unlike the other tree-building algorithms, XGBoost doesn't use entropy or Gini indices. Instead, it utilises gradient (the error term) and hessian for creating the trees. Hessian for a Regression problem is the *number of residuals* and for a classification problem. Mathematically, Hessian is a second order derivative of the loss at the current estimate given as:

$$h_m(x) = \frac{\partial^2 L(Y, f(x))}{\partial f(x)^2}_{f(x)=f^{(m-1)}(x).}$$

where **L** is the loss function.

- Initialise the tree with only one leaf.
- compute the similarity using the formula

$$Similarity = \frac{Gradient^2}{hessian + \lambda}$$

  Where λ is the regularisation term.
- Now for splitting data into a tree form, calculate

$$Gain = leftsimilarity + rightsimilarity - similarityforroot$$

- For tree pruning, the parameter γ is used. The algorithm starts from the lowest level of the tree and then starts pruning based on the value of γ.

If $Gain - \gamma < 0$, remove that branch. Else, keep the branch

- Learning is done using the equation

$$NewValue = oldValue + \eta * prediction$$

where η is the learning rate

9. What are the advantages of XGBoost?

XGBoost stands for e**X**treme **G**radient **B**oosting. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. Gradient boosting machines are generally very slow in implementation because of sequential model training. Hence, they are not very scalable. Thus, XGBoost is focused on computational speed and model performance. XGBoost provides:

- **Parallelization** of tree construction using all of your CPU cores during training.
- **Distributed Computing** for training very large models using a cluster of machines.
- **Out-of-Core Computing** for very large datasets that don't fit into memory.
- **Cache Optimization** of data structures and algorithm to make the best use of hardware.

Mathematics Involved Unlike the other tree-building algorithms, XGBoost doesn't use entropy or Gini indices. Instead, it utilises gradient (the error term) and hessian for creating the trees. Hessian for a Regression problem is the *number of residuals* and for a classification problem. Mathematically, Hessian is a second order derivative of the loss at the current estimate given as:

$$h_m(x) = \frac{\partial^2 L(Y, f(x))}{\partial f(x)^2}_{f(x)=f^{(m-1)}(x).}$$

where **L** is the loss function.

- Initialise the tree with only one leaf.
- compute the similarity using the formula

$$Similarity = \frac{Gradient^2}{hessian + \lambda}$$

  Where λ is the regularisation term.
- Now for splitting data into a tree form, calculate

$$Gain = leftsimilarity + rightsimilarity - similarityforroot$$

- For tree pruning, the parameter γ is used. The algorithm starts from the lowest level of the tree and then starts pruning based on the value of γ.

If $Gain - \gamma < 0$, remove that branch. Else, keep the branch

- Learning is done using the equation

$$NewValue = oldValue + \eta * prediction$$

where η is the learning rate

XGBoost stands for e**X**treme **G**radient **B**oosting. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. Gradient boosting machines are generally very slow in implementation because of sequential model training. Hence, they are not very scalable. Thus, XGBoost is focused on computational speed and model performance. XGBoost provides:

- **Parallelization** of tree construction using all of your CPU cores during training.
- **Distributed Computing** for training very large models using a cluster of machines.
- **Out-of-Core Computing** for very large datasets that don't fit into memory.
- **Cache Optimization** of data structures and algorithm to make the best use of hardware.

Advantages of XGBoost Algorithm in Machine Learning

XGBoost is an efficient and easy to use algorithm which delivers high performance and accuracy as compared to other algorithms. XGBoost is also known as **regularized version of GBM**. Let see some of the advantages of XGBoost algorithm:

1. Regularization: XGBoost has in-built L1 (Lasso Regression) and L2 (Ridge Regression) regularization which prevents the model from overfitting. That is why, XGBoost is also called regularized form of GBM (Gradient Boosting Machine).

While using Scikit Learn libarary, we pass two hyper-parameters (**alpha** and **lambda**) to XGBoost related to regularization. **alpha** is used for L1 regularization and **lambda** is used for L2 regularization.

2. Parallel Processing: XGBoost utilizes the power of parallel processing and that is why it is much faster than GBM. It uses multiple CPU cores to execute the model.

While using Scikit Learn libarary, **nthread** hyper-parameter is used for parallel processing. **nthread** represents number of CPU cores to be used. If you want to use all the available cores, don't mention any value for **nthread** and the algorithm will detect automatically.

3. Handling Missing Values: XGBoost has an in-built capability to handle missing values. When XGBoost encounters a missing value at a node, it tries both the left and right hand split and learns the way leading to higher loss for each node. It then does the same when working on the testing data.

XGBoost

XGBoost improves the gradient boosting method even further.

> **XGBoost** (*extreme gradient boosting*) regularises data better than normal gradient boosted Trees.

It was developed by Tianqi Chen in C++ but now has interfaces for Python, R, Julia.

XGBoost's objective function is the sum of loss function evaluated over all the predictions and a regularisation function for all predictors (j trees). In the formula f_i means a prediction coming from the j^th tree.

$$obj(\theta) = \sum_{i}^{n} l(y_i - \hat{y_i}) + \sum_{j=1}^{j} \Omega(f_j)$$

Loss function depends on the task being performed (classification, regression, etc.) and a regularization term is described by the following equation:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} \omega_j^2$$

First part (γT) is responsible for controlling the overall number of created leaves, and the second term ($\frac{1}{2}\lambda \sum_{j=1}^{T} \omega_j^2$) watches over the scores.

4. Cross Validation: XGBoost allows user to run a cross-validation at each iteration of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run. This is unlike GBM where we have to run a grid-search and only a limited values can be tested.

5. Effective Tree Pruning: A GBM would stop splitting a node when it encounters a negative loss in the split. Thus it is more of a greedy algorithm. XGBoost on the other hand make splits upto the **max_depth** specified and then start pruning the tree backwards and remove splits beyond which there is no positive gain.

For example: There may be a situation where split of negative loss say -4 may be followed by a split of positive loss +13. GBM would stop as it encounters -4. But XGBoost will go deeper and it will see a combined effect of +9 of the split and keep both.

Difference between GBM (Gradient Boosting Machine) and XGBoost (Extreme Gradient Boosting)

The objective of both GBM and XGBoost is to minimize the loss function. Both are used to improve the performance of an algorithm using Ensemble Learning. Both are boosting algorithms. If you are not familiar with bagging and boosting, please go through my previous article on bagging and boosting.

Below is a small introduction and **difference between GBM and XGBoost**.

GBM (Gradient Boosting Machine)

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 (Extreme Gradient Boosting)

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 particularly popular because it has been the winning algorithm in a number of recent **Kaggle competitions**.

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.

	Bagging	Boosting	Stacking
Partitioning of the data into subsets	Random	Giving mis-classified samples higher preference	Various
Goal to achieve	Minimize variance	Increase predictive force	Both
Methods where this is used	Random subspace	Gradient descent	Blending
Function to combine single models	(Weighted) average	Weighted majority vote	Logistic regression

In [ ]: