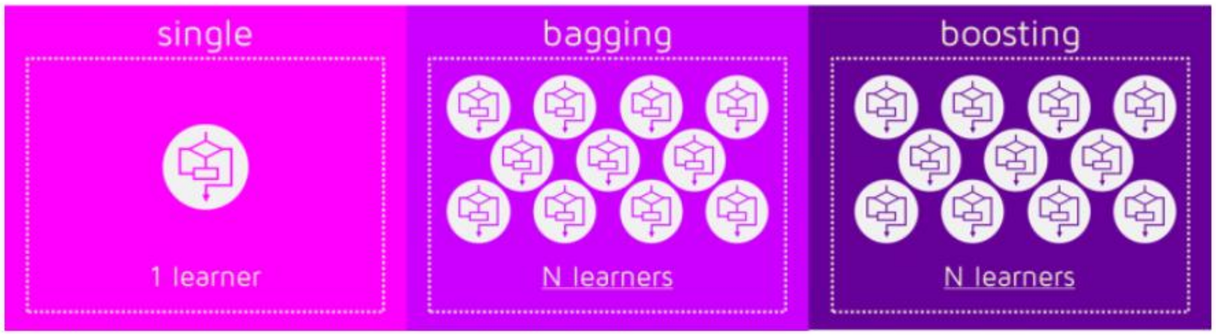
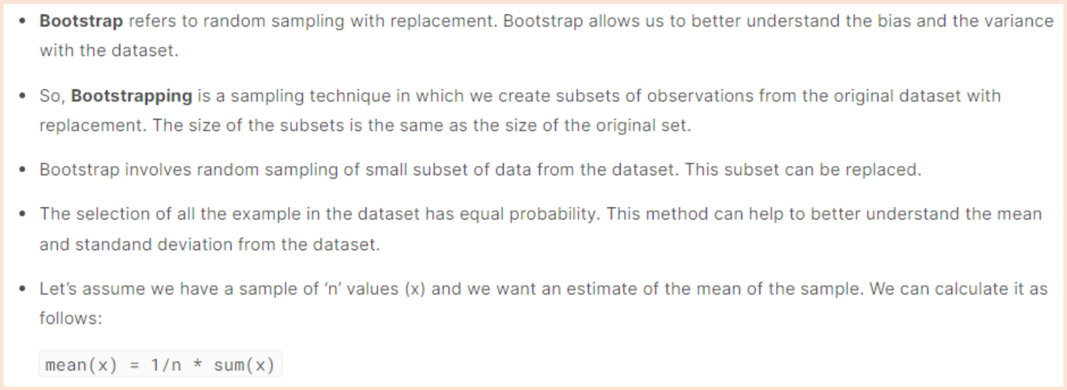
- **Bagging** and **boosting** are both ensemble learning methods in machine learning.
- **Bagging** and **boosting** are similar in that they are both ensemble techniques, where a set of weak learners are combined to create a strong learner that obtains better performance than a single one.
- Ensemble learning helps to improve machine learning model performance by combining several models. This approach allows the production of better predictive performance compared to a single model.
- The basic idea behind ensemble learning is to learn a set of classifiers (experts) and to allow them to vote. This diversification in Machine Learning is achieved by a technique called **ensemble learning**. The idea here is to train multiple models, each with the objective to predict or classify a set of results.
- **Bagging** and **boosting** are two types of ensemble learning techniques. These two decrease the variance of single estimate as they combine several estimates from different models. So the result may be a model with higher stability.
- The main causes of error in learning are due to **noise, bias and variance**. Ensemble helps to minimize these factors. By using ensemble methods, we're able to increase the stability of the final model and reduce the errors mentioned previously.
- **Bagging helps to decrease the model's variance.**
- **Boosting helps to decrease the model's bias.**
- These methods are designed to improve the stability and the accuracy of Machine Learning algorithms. Combinations of multiple classifiers decrease variance, especially in the case of unstable classifiers, and may produce a more reliable classification than a single classifier.

- To use Bagging or Boosting you must select a base learner algorithm. For example, if we choose a classification tree, Bagging and Boosting would consist of a pool of trees as big as we want as shown in the following diagram:



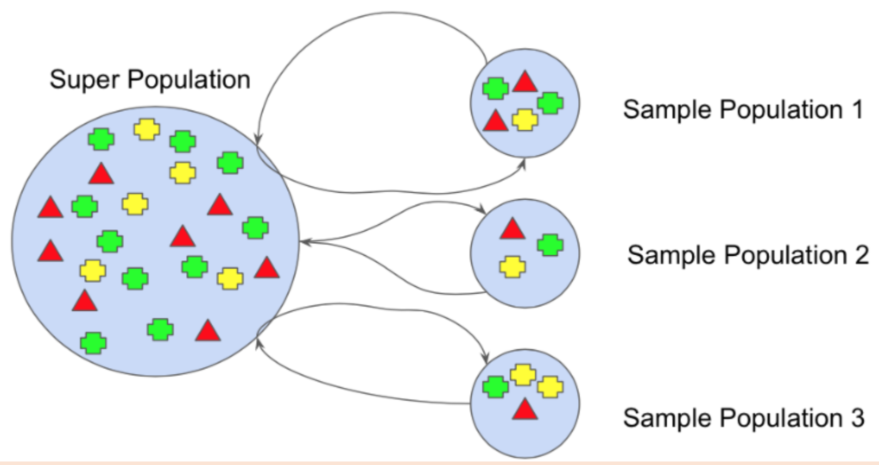
Bootstrapping



- **Bootstrap** refers to random sampling with replacement. Bootstrap allows us to better understand the bias and the variance with the dataset.
- So, **Bootstrapping** is a sampling technique in which we create subsets of observations from the original dataset with replacement. The size of the subsets is the same as the size of the original set.
- Bootstrap involves random sampling of small subset of data from the dataset. This subset can be replaced.
- The selection of all the example in the dataset has equal probability. This method can help to better understand the mean and standand deviation from the dataset.
- Let's assume we have a sample of 'n' values (x) and we want an estimate of the mean of the sample. We can calculate it as follows:

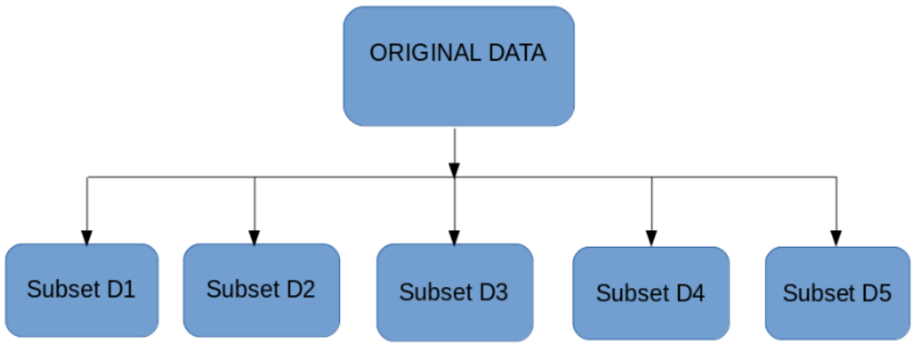
  `mean(x) = 1/n * sum(x)`

- Bootstrapping can be represented diagrammatically as follows:



Bagging

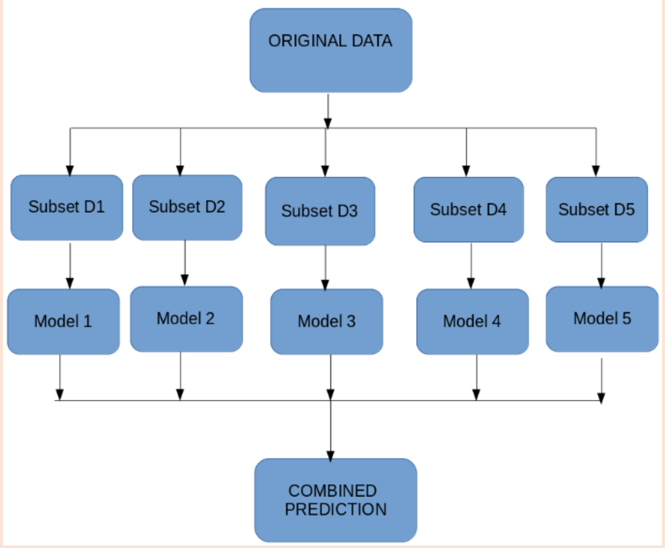
- Bagging (or Bootstrap Aggregation), is a simple and very powerful ensemble method. Bagging is the application of the Bootstrap procedure to a high-variance machine learning algorithm, typically decision trees.
- The idea behind bagging is combining the results of multiple models (for instance, all decision trees) to get a generalized result. Now, bootstrapping comes into picture.
- Bagging (or Bootstrap Aggregating) technique uses these subsets (bags) to get a fair idea of the distribution (complete set). The size of subsets created for bagging may be less than the original set.
- It can be represented as follows:



Bagging works as follows:-

1. Multiple subsets are created from the original dataset, selecting observations with replacement.
2. A base model (weak model) is created on each of these subsets.
3. The models run in parallel and are independent of each other.
4. The final predictions are determined by combining the predictions from all the models.

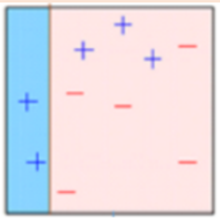
Now, bagging can be represented diagrammatically as follows:



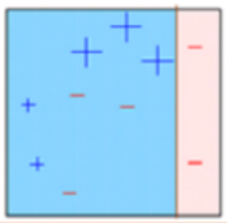
Boosting

- Boosting is a sequential process, where each subsequent model attempts to correct the errors of the previous model. The succeeding models are dependent on the previous model.
- In this technique, learners are learned sequentially with early learners fitting simple models to the data and then analyzing data for errors. In other words, we fit consecutive trees (random sample) and at every step, the goal is to solve for net error from the prior tree.
- When an input is misclassified by a hypothesis, its weight is increased so that next hypothesis is more likely to classify it correctly. By combining the whole set at the end converts weak learners into better performing model.
- Let's understand the way boosting works in the below steps.

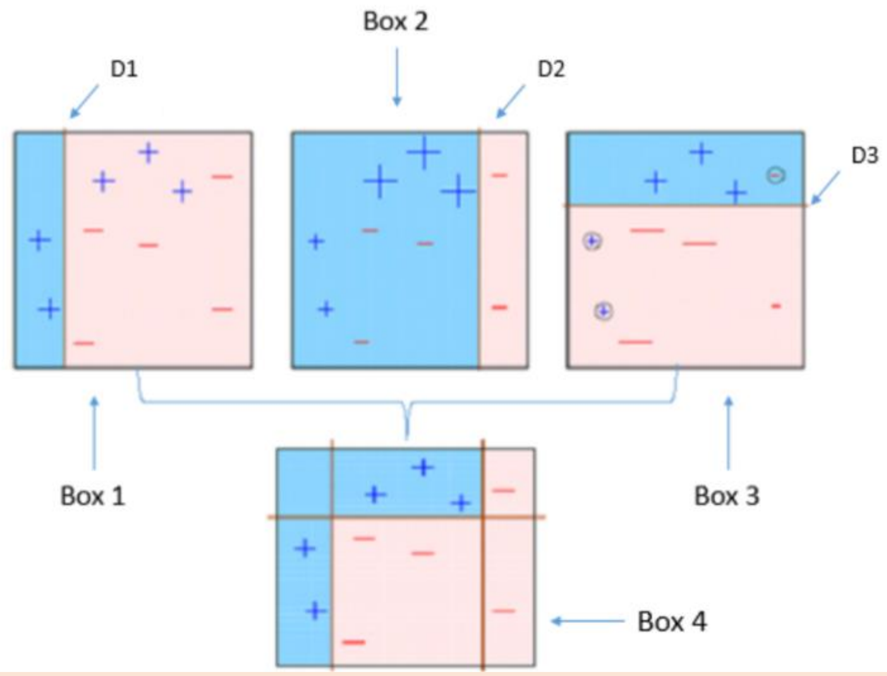
1. A subset is created from the original dataset.
2. Initially, all data points are given equal weights.
3. A base model is created on this subset.
4. This model is used to make predictions on the whole dataset.



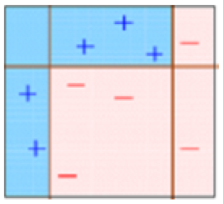
1. Errors are calculated using the actual values and predicted values.
2. The observations which are incorrectly predicted, are given higher weights. (Here, the three misclassified blue-plus points will be given higher weights)
3. Another model is created and predictions are made on the dataset. (This model tries to correct the errors from the previous model)



1. Similarly, multiple models are created, each correcting the errors of the previous model.
2. The final model (strong learner) is the weighted mean of all the models (weak learners).

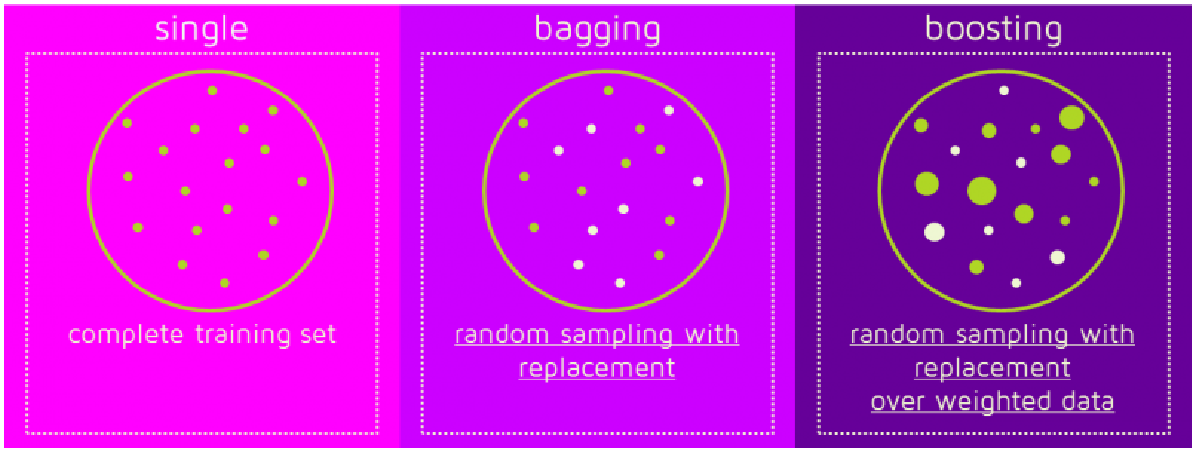


- Thus, the boosting algorithm combines a number of weak learners to form a strong learner.
- The individual models would not perform well on the entire dataset, but they work well for some part of the dataset.
- Thus, each model actually boosts the performance of the ensemble.



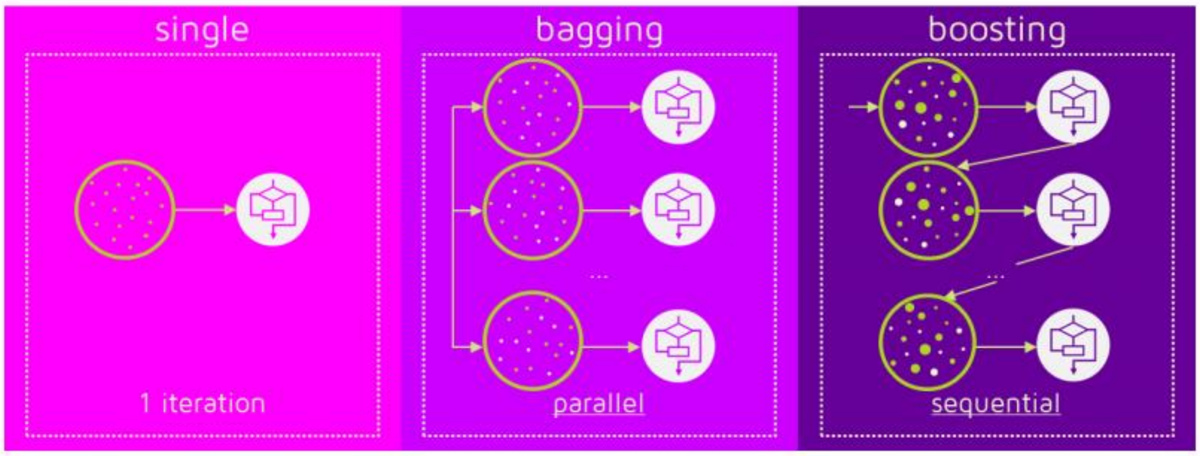
Getting N learners for Bagging and Boosting

- Bagging and Boosting get N learners by generating additional data in the training stage.
- N new training data sets are produced by random sampling with replacement from the original set.
- By sampling with replacement some observations may be repeated in each new training data set.
- In the case of Bagging, any element has the same probability to appear in a new data set.
- However, for Boosting the observations are weighted and therefore some of them will take part in the new sets more often.
- These multiple sets are used to train the same learner algorithm and therefore different classifiers are produced.
- This is represented diagrammatically as follows:


single
complete training set
bagging
random sampling with replacement
boosting
random sampling with replacement over weighted data

Weighted Data Elements

- Now, we know the main difference between the two methods.
- While the training stage is parallel for Bagging (i.e., each model is built independently), Boosting builds the new learner in a sequential way as follows:

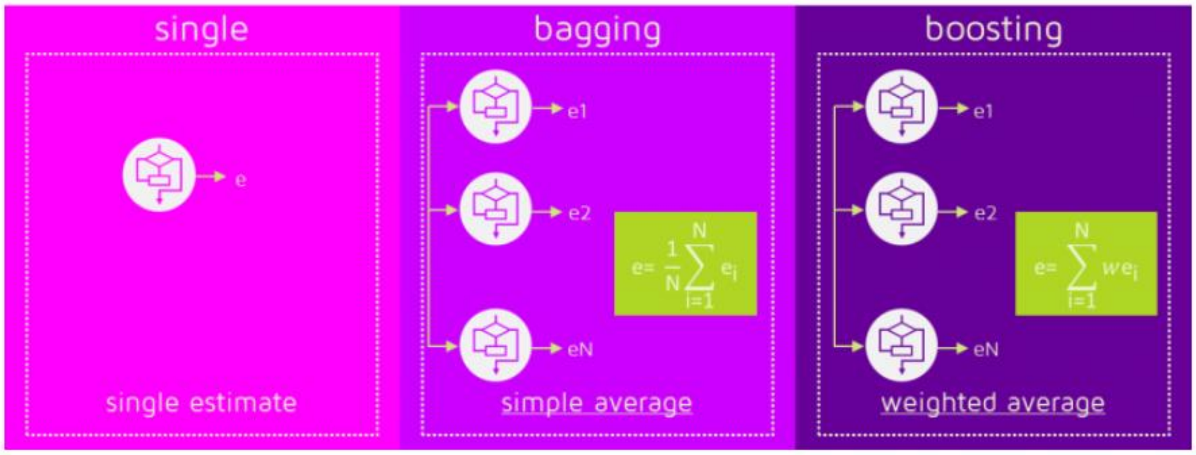


- In Boosting algorithms each classifier is trained on data, taking into account the previous classifiers' success.
- After each training step, the weights are redistributed. Misclassified data increases its weights to emphasise the most difficult cases.
- In this way, subsequent learners will focus on them during their training.

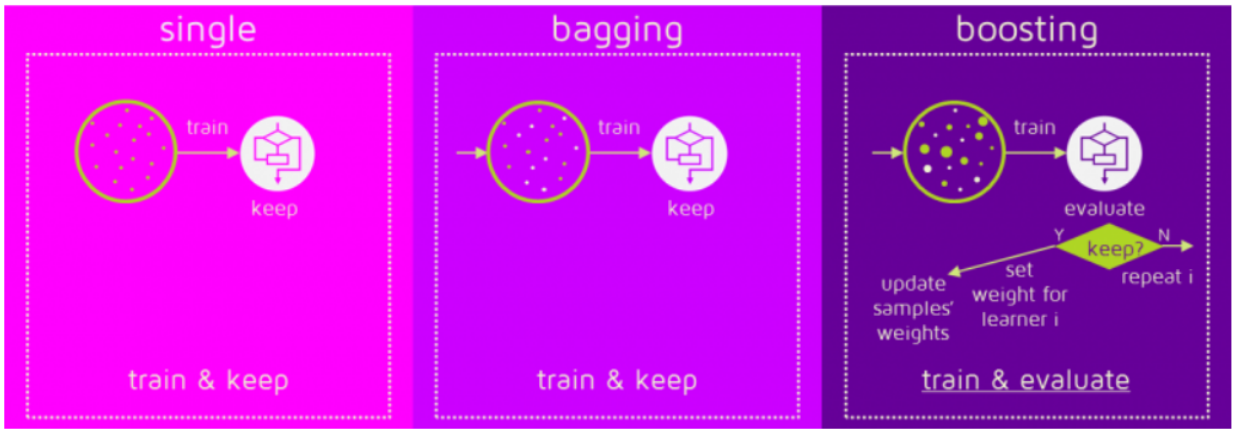
- In Boosting algorithms each classifier is trained on data, taking into account the previous classifiers' success.
- After each training step, the weights are redistributed. Misclassified data increases its weights to emphasise the most difficult cases.
- In this way, subsequent learners will focus on them during their training.

Classification stage in action

- To predict the class of new data we only need to apply the N learners to the new observations.
- In Bagging the result is obtained by averaging the responses of the N learners (or majority vote).
- However, Boosting assigns a second set of weights, this time for the N classifiers, in order to take a weighted average of their estimates.
- This is shown diagrammatically below:



- In the Boosting training stage, the algorithm allocates weights to each resulting model.
- A learner with good a classification result on the training data will be assigned a higher weight than a poor one.
- So when evaluating a new learner, Boosting needs to keep track of learners' errors, too.



- Some of the Boosting techniques include an extra-condition to keep or discard a single learner.
- For example, in AdaBoost, the most renowned, an error less than 50% is required to maintain the model; otherwise, the iteration is repeated until achieving a learner better than a random guess.
- The previous image shows the general process of a Boosting method, but several alternatives exist with different ways to determine the weights to use in the next training step and in the classification stage.

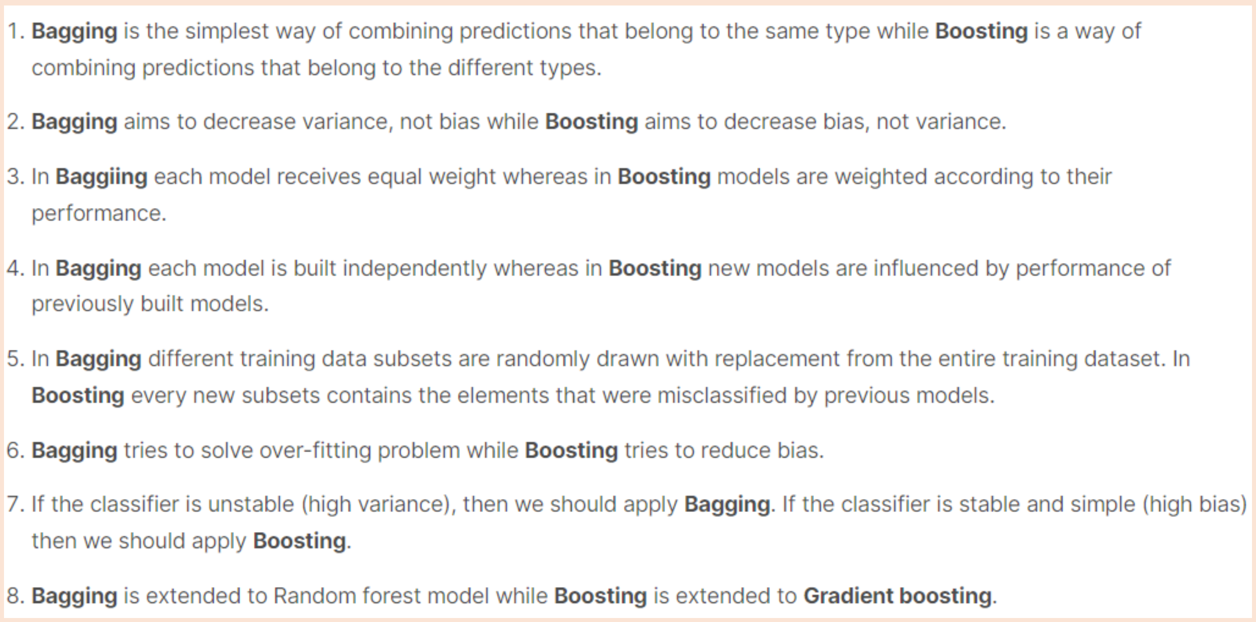
Selecting the best technique- Bagging or Boosting

- Now, the question may come to our mind - whether to select Bagging or Boosting for a particular problem.
- 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 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.

Similarities between Bagging and Boosting

1. Both are ensemble methods to get N learners from 1 learner.
2. Both generate several training data sets by random sampling.
3. Both make the final decision by averaging the N learners (or taking the majority of them i.e Majority Voting).
4. Both are good at reducing variance and provide higher stability.

Differences between Bagging and Boosting



1. **Bagging** is the simplest way of combining predictions that belong to the same type while **Boosting** is a way of combining predictions that belong to the different types.
2. **Bagging** aims to decrease variance, not bias while **Boosting** aims to decrease bias, not variance.
3. In **Baggiing** each model receives equal weight whereas in **Boosting** models are weighted according to their performance.
4. In **Bagging** each model is built independently whereas in **Boosting** new models are influenced by performance of previously built models.
5. In **Bagging** different training data subsets are randomly drawn with replacement from the entire training dataset. In **Boosting** every new subsets contains the elements that were misclassified by previous models.
6. **Bagging** tries to solve over-fitting problem while **Boosting** tries to reduce bias.
7. If the classifier is unstable (high variance), then we should apply **Bagging**. If the classifier is stable and simple (high bias) then we should apply **Boosting**.
8. **Bagging** is extended to Random forest model while **Boosting** is extended to **Gradient boosting**.

AdaBoost Algorithm

There are mainly 3 types of boosting algorithms:

- AdaBoost algorithm in Machine Learning
- Gradient descent algorithm
- Xtreme gradient descent algorithm

Learning Objectives

- To understand what the AdaBoost algorithm is and how it works.
- To understand what stumps are.
- To find out how boosting algorithms help increase the accuracy of ML models.

- **What Is the AdaBoost Algorithm?**
- **Understanding the Working of the AdaBoost Algorithm**

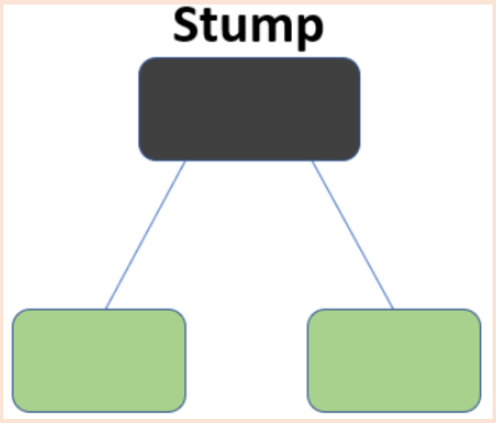
What Is the AdaBoost Algorithm?

There are many machine learning algorithms to choose from for your problem statements. One of these algorithms for predictive modeling is called AdaBoost.

AdaBoost, also called Adaptive Boosting, is a technique in Machine Learning used as an Ensemble Method.

The most common estimator used with AdaBoost is decision trees with one level which means Decision trees with only 1 split.

These trees are also called ***Decision Stumps***.

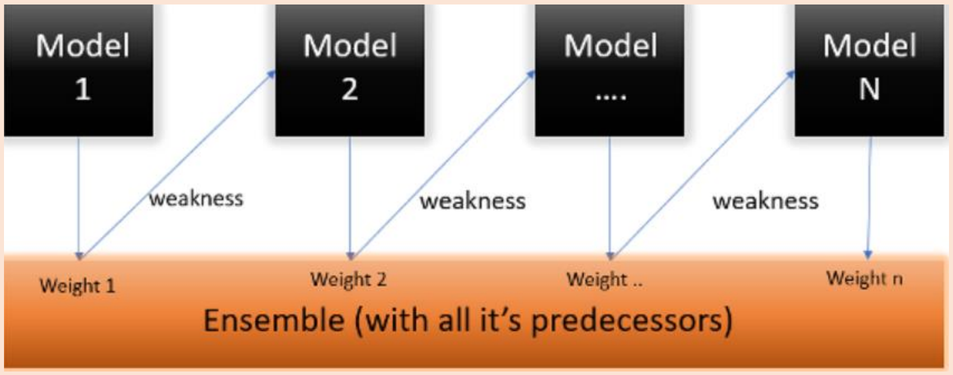


What this algorithm does is that it builds a model and gives equal weights to all the data points.

It then assigns higher weights to points that are wrongly classified.

Now all the points with higher weights are given more importance in the next model.

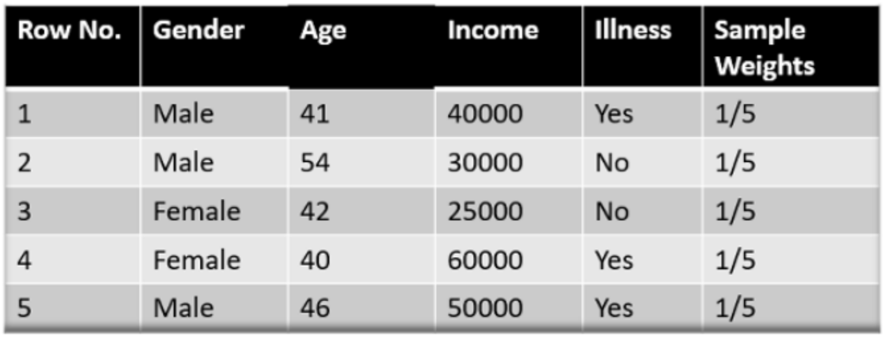
It will keep training models until and unless a lower error is received.



There is another ensemble learning algorithm called the **Gradient Boosting Algorithm**.

In this algorithm, we try to reduce the error instead of weights, as in AdaBoost.

Step 1 – The Image shown below is the actual representation of our dataset. Since the target column is binary, it is a classification problem. First of all, these data points will be assigned some weights. Initially, all the weights will be equal.



Row No.	Gender	Age	Income	Illness	Sample Weights
1	Male	41	40000	Yes	1/5
2	Male	54	30000	No	1/5
3	Female	42	25000	No	1/5
4	Female	40	60000	Yes	1/5
5	Male	46	50000	Yes	1/5

The formula to calculate the sample weights is:

$$w(x_i, y_i) = \frac{1}{N}, \; i = 1, 2, \ldots . n$$

Where N is the total number of data points

Here since we have 5 data points, the sample weights assigned will be 1/5.

Step 2 – We start by seeing how well "*Gender*" classifies the samples and will see how the variables (Age, Income) classify the samples.

We'll create a decision stump for each of the features and then calculate the ***Gini Index*** of each tree. The tree with the lowest Gini Index will be our first stump.

Here in our dataset, let's say ***Gender*** has the lowest gini index, so it will be our first stump.

Step 3 – We'll now calculate the "**Amount of Say**" or "**Importance**" or "**Influence**" for this classifier in classifying the data points using this formula:

$$\frac{1}{2}\log\frac{1 - Total\ Error}{Total\ Error}$$

The total error is nothing but the summation of all the sample weights of misclassified data points.

Here in our dataset, let's assume there is 1 wrong output, so our total error will be 1/5, and the alpha (performance of the stump) will be:

$$Performance\ of\ the\ stump\ =\ \frac{1}{2}\log_e\left(\frac{1-Total\ Error}{Total\ Error}\right)$$

$$\alpha\ =\ \frac{1}{2}\log_e\left(\frac{1-\frac{1}{5}}{\frac{1}{5}}\right)$$

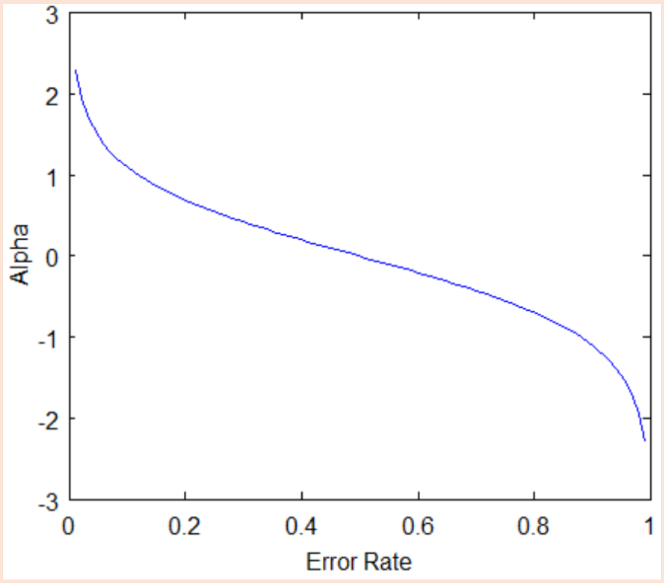
$$\alpha\ =\ \frac{1}{2}\log_e\left(\frac{0.8}{0.2}\right)$$

$$\alpha\ =\ \frac{1}{2}\log_e(4)\ =\ \frac{1}{2}*(1.38)$$

$$\alpha\ =\ 0.69$$

Note: Total error will always be between 0 and 1.

0 Indicates perfect stump, and 1 indicates horrible stump.



From the graph above, we can see that when there is no misclassification, then we have no error (Total Error = 0), so the "amount of say (alpha)" will be a large number.

When the classifier predicts half right and half wrong, then the Total Error = 0.5, and the importance (amount of say) of the classifier will be 0.

If all the samples have been incorrectly classified, then the error will be very high (approx. to 1), and hence our alpha value will be a negative integer.

Step 4 – You must be wondering why it is necessary to calculate the TE and performance of a stump. Well, the answer is very simple, we need to update the weights because if the same weights are applied to the next model, then the output received will be the same as what was received in the first model.

The wrong predictions will be given more weight, whereas the correct predictions weights will be decreased. Now when we build our next model after updating the weights, more preference will be given to the points with higher weights.

After finding the importance of the classifier and total error, we need to finally update the weights, and for this, we use the following formula:

$$New\ sample\ weight\ =\ old\ weight\ *\ e^{\pm Amount\ of\ say\ (\alpha)}$$

The amount of, say (alpha) will be ***negative*** when the sample is **correctly classified**.

The amount of, say (alpha) will be ***positive*** when the sample is **miss-classified.**

There are four correctly classified samples and 1 wrong. Here, the ***sample weight*** of that datapoint is *1/5,* and the ***amount of say/performance of the stump*** of ***Gender*** is *0.69.*

New weights for *correctly classified* samples are:

$$New\ sample\ weight = \frac{1}{5} * \exp(-0.69)$$

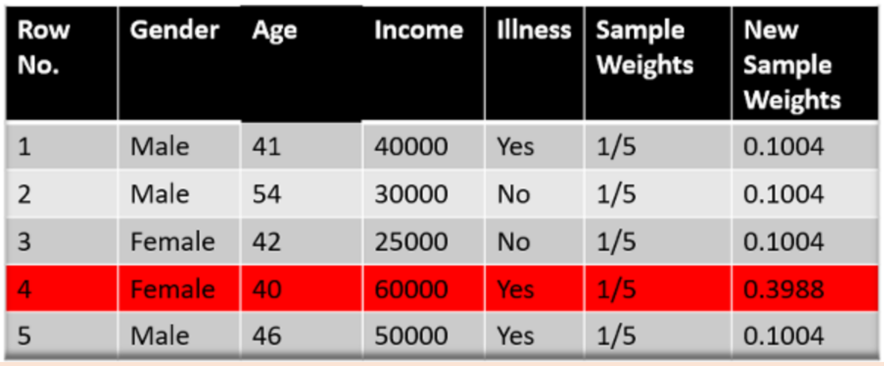
$$New\ sample\ weight = 0.2 * 0.502 = 0.1004$$

For *wrongly classified* samples, the updated weights will be:

$$New\ sample\ weight = \frac{1}{5} * \exp(0.69)$$

$$New\ sample\ weight = 0.2 * 1.994 = 0.3988$$

Note: See the sign of alpha when I am putting the values, the **alpha is negative** when the data point is correctly classified, and this *decreases the sample weight* from 0.2 to 0.1004. It is **positive** when there is **misclassification**, and this will *increase the sample weight* from 0.2 to 0.3988



Row No.	Gender	Age	Income	Illness	Sample Weights	New Sample Weights
1	Male	41	40000	Yes	1/5	0.1004
2	Male	54	30000	No	1/5	0.1004
3	Female	42	25000	No	1/5	0.1004
4	Female	40	60000	Yes	1/5	0.3988
5	Male	46	50000	Yes	1/5	0.1004

We know that the total sum of the sample weights must be equal to 1, but here if we sum up all the new sample weights, we will get 0.8004. To bring this sum equal to 1, we will normalize these weights by dividing all the weights by the total sum of updated weights, which is 0.8004. So, after normalizing the sample weights, we get this dataset, and now the sum is equal to 1.

Row No.	Gender	Age	Income	Illness	Sample Weights	New Sample Weights
1	Male	41	40000	Yes	1/5	0.1004/0.8004 =0.1254
2	Male	54	30000	No	1/5	0.1004/0.8004 =0.1254
3	Female	42	25000	No	1/5	0.1004/0.8004 =0.1254
4	Female	40	60000	Yes	1/5	0.3988/0.8004 =0.4982
5	Male	46	50000	Yes	1/5	0.1004/0.8004 =0.1254

Step 5 – Now, we need to make a new dataset to see if the errors decreased or not. For this, we will remove the “sample weights” and “new sample weights” columns and then, based on the “new sample weights,” divide our data points into buckets.

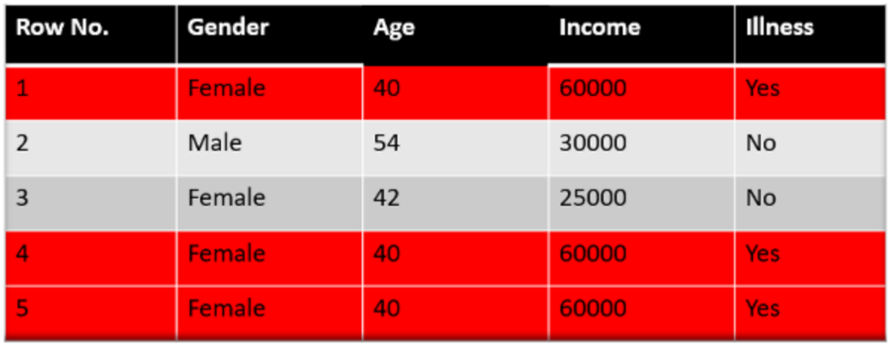
Row No.	Gender	Age	Income	Illness	New Sample Weights	Buckets
1	Male	41	40000	Yes	0.1004/0.8004= 0.1254	0 to 0.1254
2	Male	54	30000	No	0.1004/0.8004= 0.1254	0.1254 to 0.2508
3	Female	42	25000	No	0.1004/0.8004= 0.1254	0.2508 to 0.3762
4	Female	40	60000	Yes	0.3988/0.8004= 0.4982	0.3762 to 0.8744
5	Male	46	50000	Yes	0.1004/0.8004= 0.1254	0.8744 to 0.9998

Step 6 – We are almost done. Now, what the algorithm does is selects random numbers from 0-1. Since incorrectly classified records have higher sample weights, the probability of selecting those records is very high.

Suppose the 5 random numbers our algorithm take is 0.38,0.26,0.98,0.40,0.55.

Now we will see where these random numbers fall in the bucket, and according to it, we’ll make our new dataset shown below.

Row No.	Gender	Age	Income	Illness
1	Female	40	60000	Yes
2	Male	54	30000	No
3	Female	42	25000	No



Row No.	Gender	Age	Income	Illness
1	Female	40	60000	Yes
2	Male	54	30000	No
3	Female	42	25000	No
4	Female	40	60000	Yes
5	Female	40	60000	Yes

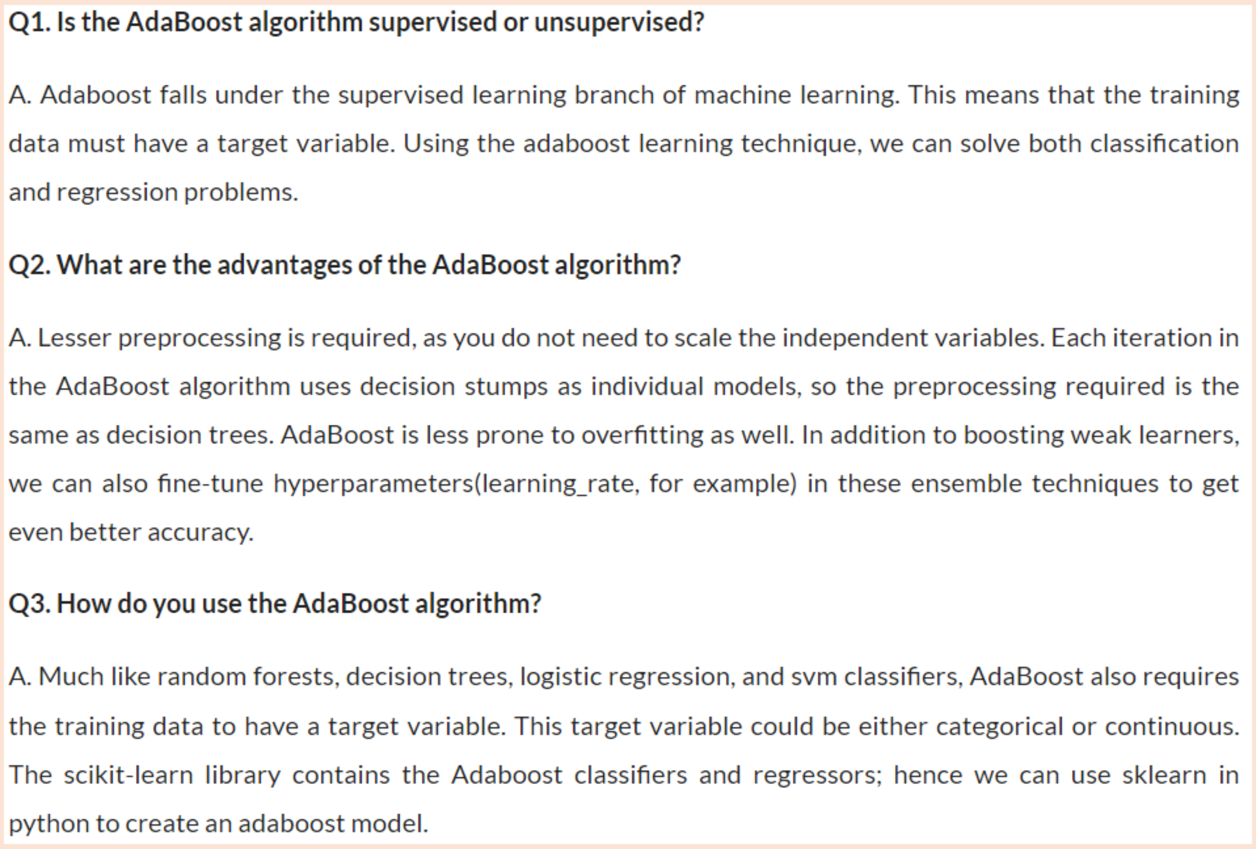
This comes out to be our new dataset, and we see the data point, which was wrongly classified, has been selected 3 times because it has a higher weight.

Step 9 – Now this act as our new dataset, and we need to repeat all the above steps i.e.

1. Assign ***equal weights*** to all the data points.
2. Find the stump that does the ***best job classifying*** the new collection of samples by finding their Gini Index and selecting the one with the lowest Gini index.
3. Calculate the ***"Amount of Say"*** and ***"Total error"*** to update the previous sample weights.
4. Normalize the new sample weights.

Iterate through these steps until and unless a low training error is achieved.

Suppose, with respect to our dataset, we have constructed 3 decision trees (DT1, DT2, DT3) in a ***sequential manner.*** If we send our **test data** now, it will pass through all the decision trees, and finally, we will see which class has the majority, and based on that, we will do predictions
for our test dataset.



Q1. Is the AdaBoost algorithm supervised or unsupervised?

A. Adaboost falls under the supervised learning branch of machine learning. This means that the training data must have a target variable. Using the adaboost learning technique, we can solve both classification and regression problems.

Q2. What are the advantages of the AdaBoost algorithm?

A. Lesser preprocessing is required, as you do not need to scale the independent variables. Each iteration in the AdaBoost algorithm uses decision stumps as individual models, so the preprocessing required is the same as decision trees. AdaBoost is less prone to overfitting as well. In addition to boosting weak learners, we can also fine-tune hyperparameters(learning_rate, for example) in these ensemble techniques to get even better accuracy.

Q3. How do you use the AdaBoost algorithm?

A. Much like random forests, decision trees, logistic regression, and svm classifiers, AdaBoost also requires the training data to have a target variable. This target variable could be either categorical or continuous. The scikit-learn library contains the Adaboost classifiers and regressors; hence we can use sklearn in python to create an adaboost model.

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

What is A Random Forest?

Random forest is a commonly-used machine learning algorithm trademarked by Leo Breiman and Adele Cutler, which combines the output of multiple decision trees to reach a single result.

Its ease of use and flexibility have fueled its adoption, as it handles both classification and regression problems.

Decision trees

Since the random forest model is made up of multiple decision trees, it would be helpful to start by describing the decision tree algorithm briefly.

Decision trees start with a basic question, such as, “Should I surf?”

From there, you can ask a series of questions to determine an answer, such as, “Is it a long period swell?” or “Is the wind blowing offshore?”.

These questions make up the decision nodes in the tree, acting as a means to split the data.

Each question helps an individual to arrive at a final decision, which would be denoted by the leaf node.

Observations that fit the criteria will follow the “Yes” branch and those that don’t will follow the alternate path.

Decision trees seek to find the best split to subset the data, and they are typically trained through the Classification and Regression Tree (CART) algorithm.

Metrics, such as Gini impurity, information gain, or mean square error (MSE), can be used to evaluate the quality of the split.

This decision tree is an example of a classification problem, where the class labels are "surf" and "don't surf."

While decision trees are common supervised learning algorithms, they can be prone to problems, such as bias and overfitting.

However, when multiple decision trees form an ensemble in the random forest algorithm, they predict more accurate results, particularly when the individual trees are uncorrelated with each other.

Ensemble methods

Ensemble learning methods are made up of a set of classifiers—e.g. decision trees—and their predictions are aggregated to identify the most popular result.

The most well-known ensemble methods are bagging, also known as bootstrap aggregation, and boosting.

In 1996, Leo Breiman introduced the bagging method; in this method, a random sample of data in a training set is selected with replacement—meaning that the individual data points can be chosen more than once.

After several data samples are generated, these models are then trained independently, and depending on the type of task - i.e. Regression or Classification—the average or majority of those predictions yield a more accurate estimate.

This approach is commonly used to reduce variance within a noisy dataset.

Random Forest Algorithm

The random forest algorithm is an extension of the bagging method as it utilizes both bagging and feature randomness to create an uncorrelated forest of decision trees.

Feature randomness, also known as feature bagging or "the random subspace method"(link resides outside ibm.com) (PDF, 121 KB), generates a random subset of features, which ensures low correlation among decision trees.

This is a key difference between decision trees and random forests.

While decision trees consider all the possible feature splits, random forests only select a subset of those features.
If we go back to the "should I surf?" example, the questions that I may ask to determine the prediction may not be as comprehensive as someone else's set of questions.

By accounting for all the potential variability in the data, we can reduce the risk of overfitting, bias, and overall variance, resulting in more precise predictions.

How it Works?

Random forest algorithms have three main hyperparameters, which need to be set before training. These include node size, the number of trees, and the number of features sampled.

From there, the random forest classifier can be used to solve for regression or classification problems.

The random forest algorithm is made up of a collection of decision trees, and each tree in the ensemble is comprised of a data sample drawn from a training set with replacement, called the bootstrap sample.

Of that training sample, one-third of it is set aside as test data, known as the out-of-bag (oob) sample, which we'll come back to later.

Another instance of randomness is then injected through feature bagging, adding more diversity to the dataset and reducing the correlation among decision trees.

Depending on the type of problem, the determination of the prediction will vary.

For a regression task, the individual decision trees will be averaged, and for a classification task, a majority vote—i.e. the most frequent categorical variable—will yield the predicted class.

Finally, the oob sample is then used for cross-validation, finalizing that prediction.

Benefits and challenges of Random Forest

There are a number of key advantages and challenges that the random forest algorithm presents when used for classification or regression problems. Some of them include:

Key Benefits:

- **Reduced risk of overfitting:** Decision trees run the risk of overfitting as they tend to tightly fit all the samples within training data. However, when there's a robust number of decision trees in a random forest, the classifier won't overfit the model since the averaging of uncorrelated trees lowers the overall variance and prediction error.

- **Provides flexibility:** Since random forest can handle both regression and classification tasks with a high degree of accuracy, it is a popular method among data scientists. Feature bagging also makes the random forest classifier an effective tool for estimating missing values as it maintains accuracy when a portion of the data is missing.

- **Easy to determine feature importance:** Random forest makes it easy to evaluate variable importance, or contribution, to the model. There are a few ways to evaluate feature importance. Gini importance and mean decrease in impurity (MDI) are usually used to measure how much the model's accuracy decreases when a given variable is excluded. However, permutation importance, also known as mean decrease accuracy (MDA), is another importance measure. MDA identifies the average decrease in accuracy by randomly permutating the feature values in oob samples.

Key Challenges

- **Time-consuming process:** Since random forest algorithms can handle large data sets, they can be provide more accurate predictions, but can be slow to process data as they are computing data for each individual decision tree.
- **Requires more resources:** Since random forests process larger data sets, they'll require more resources to store that data.
- **More complex:** The prediction of a single decision tree is easier to interpret when compared to a forest of them.

Random Forest Applications:

The random forest algorithm has been applied across a number of industries, allowing them to make better business decisions. Some use cases include:

- **Finance:** It is a preferred algorithm over others as it reduces time spent on data management and pre-processing tasks. It can be used to evaluate customers with high credit risk, to detect fraud, and option pricing problems.
- **Healthcare:** The random forest algorithm has applications within computational biology (link resides outside ibm.com) (PDF, 737 KB), allowing doctors to tackle problems such as gene expression classification, biomarker discovery, and sequence annotation. As a result, doctors can make estimates around drug responses to specific medications.
- **E-commerce:** It can be used for recommendation engines for cross-sell purposes.

Working of Random Forest Algorithm

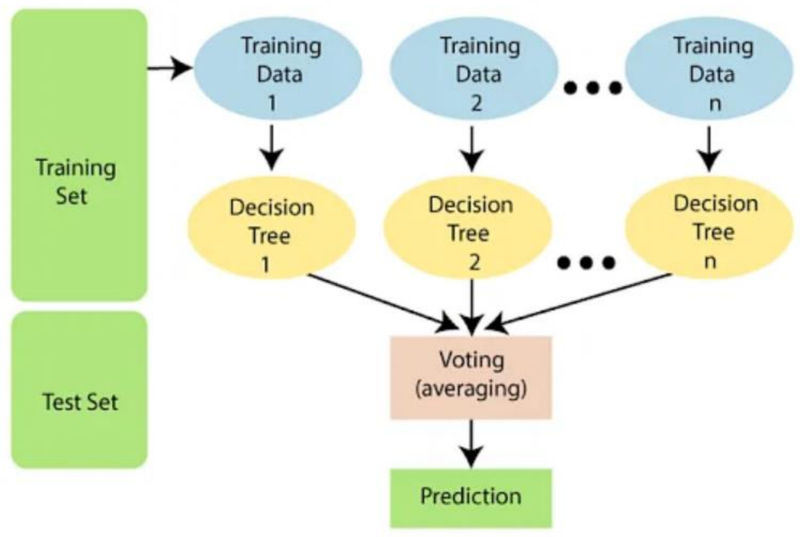


IMAGE COURTESY: javapoint

The following steps explain the working Random Forest Algorithm:

Step 1: Select random samples from a given data or training set.

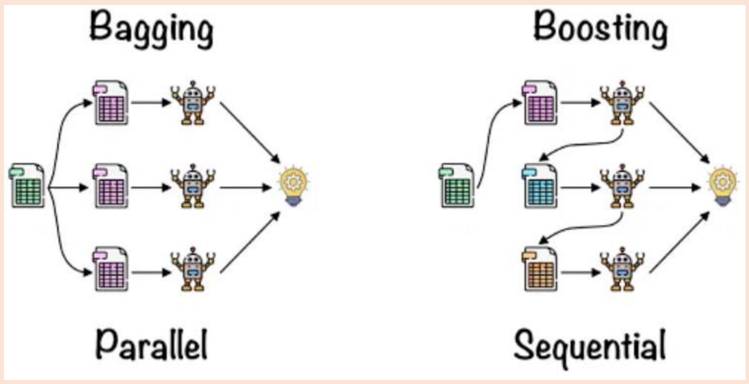
Step 2: This algorithm will construct a decision tree for every training data.

Step 3: Voting will take place by averaging the decision tree.

Step 4: Finally, select the most voted prediction result as the final prediction result.

This combination of multiple models is called Ensemble. Ensemble uses two methods:

1. Bagging: Creating a different training subset from sample training data with replacement is called Bagging. The final output is based on majority voting.
2. Boosting: Combing weak learners into strong learners by creating sequential models such that the final model has the highest accuracy is called Boosting. Example: ADA BOOST, XG BOOST.



Bagging: From the principle mentioned above, we can understand Random forest uses the Bagging code.

Bagging is also known as **Bootstrap Aggregation** used by random forest.

The ***process begins with any original random data***.

After arranging, it is **organized into samples known as Bootstrap Sample**.

This process is known as Bootstrapping.

Further, the models are trained individually, yielding different results known as Aggregation.

In the last step, all the results are combined, and the generated output is based on majority voting.

This step is known as Bagging and is done using an Ensemble Classifier.

Essential Features of Random Forest

- **Miscellany:** Each tree has a unique attribute, variety and features concerning other trees. Not all trees are the same.
- **Immune to the curse of dimensionality:** Since a tree is a conceptual idea, it requires no features to be considered. Hence, the feature space is reduced.
- **Parallelization:** We can fully use the CPU to build random forests since each tree is created autonomously from different data and features.
- **Train-Test split:** In a Random Forest, we don't have to differentiate the data for train and test because the decision tree never sees 30% of the data.

- **Stability:** The final result is based on Bagging, meaning the result is based on majority voting or average.

=21

Decision Trees	**Random Forest**
They usually suffer from the problem of overfitting if it's allowed to grow without any control.	Since they are created from subsets of data and the final output is based on average or majority ranking, the problem of overfitting doesn't happen here.
A single decision tree is comparatively faster in computation.	It is slower.
They use a particular set of rules when a data set with features are taken as input.	Random Forest randomly selects observations, builds a decision tree and then the result is obtained based on majority voting. No formulae are required here.

Why Use a Random Forest Algorithm?

There are a lot of benefits to using Random Forest Algorithm, but one of the main advantages is that it reduces the risk of overfitting and the required training time.

Additionally, it offers a high level of accuracy. Random Forest algorithm runs efficiently in large databases and produces highly accurate predictions by estimating missing data.

Important Hyperparameters

Hyperparameters are used in random forests to either enhance the performance and predictive power of models or to make the model faster.

The following hyperparameters are used to enhance the predictive power:

- **n_estimators:** Number of trees built by the algorithm before averaging the products.
- **max_features:** Maximum number of features random forest uses before considering splitting a node.
- **mini_sample_leaf:** Determines the minimum number of leaves required to split an internal node.

The ***following hyperparameters*** are used to increase the speed of the model:

- **n_jobs:** Conveys to the engine how many processors are allowed to use. If the value is 1, it can use only one processor, but if the value is -1,, there is no limit.

- **random_state:** Controls randomness of the sample. The model will always produce the same results if it has a definite value of random state and if it has been given the same hyperparameters and the same training data.
- **oob_score:** OOB (Out Of the Bag) is a random forest cross-validation method. In this, one-third of the sample is not used to train the data but to evaluate its performance.

Important Terms

There are different ways that the Random Forest algorithm makes data decisions, and consequently, there are some important related terms to know. Some of these terms include:

- **Entropy:** It is a measure of randomness or unpredictability in the data set.
- Information Gain: A measure of the decrease in the entropy after the data set is split is the information gain.
- **Leaf Node:** A leaf node is a node that carries the classification or the decision.
- **Decision Node:** A node that has two or more branches.
- **Root Node:** The root node is the topmost decision node, which is where you have all of your data.

Random Forest:

Every decision tree has high variance, but when we combine all of them together in parallel then the resultant variance is low as each decision tree gets perfectly trained on that particular sample data and hence the output doesn't depend on one decision tree but multiple decision trees.

In the case of a classification problem, the final output is taken by using the majority voting classifier.

In the case of a regression problem, the final output is the mean of all the outputs. This part is Aggregation.

The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.

Random Forest has multiple decision trees as base learning models.

We randomly perform row sampling and feature sampling from the dataset forming sample datasets for every model. This part is called Bootstrap.

Boosting:

Boosting is an ensemble modelling, technique that attempts to build a strong classifier from the number of weak classifiers.

It is done by building a model by using weak models in series.

Firstly, a model is built from the training data.

Then the second model is built which tries to correct the errors present in the first model.

This procedure is continued and models are added until either the complete training data set is predicted correctly or the maximum number of models are added.

Gradient Boosting

Gradient Boosting is a popular boosting algorithm.

In gradient boosting, each predictor corrects its predecessor’s error.

In contrast to Adaboost, the weights of the training instances are not tweaked, instead, each predictor is trained using the residual errors of predecessor as labels.

XGBoost

XGBoost is an implementation of Gradient Boosted decision trees.

XGBoost models majorly dominate in many Kaggle Competitions.

In this algorithm, decision trees are created in sequential form.

Weights play an important role in XGBoost.

Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results.

The weight of variables predicted wrong by the tree is increased and these variables are then fed to the second decision tree.

These individual classifiers/predictors then ensemble to give a strong and more precise model.

It can work on regression, classification, ranking, and user-defined prediction problems.

XGBoost is an optimized distributed gradient boosting library designed for efficient and scalable training of machine learning models.

It is an ensemble learning method that combines the predictions of multiple weak models to produce a stronger prediction.

XGBoost stands for “Extreme Gradient Boosting” and it has become one of the most popular and widely used machine learning algorithms due to its ability to handle large datasets and its ability to achieve state-of-the-art performance in many machine learning tasks such as classification and regression.

One of the key features of XGBoost is its efficient handling of missing values, which allows it to handle real-world data with missing values without requiring significant pre-processing.

Additionally, XGBoost has built-in support for parallel processing, making it possible to train models on large datasets in a reasonable amount of time.

XGBoost can be used in a variety of applications, including Kaggle competitions, recommendation systems, and click-through rate prediction, among others.

It is also highly customizable and allows for fine-tuning of various model parameters to optimize performance.

XgBoost stands for Extreme Gradient Boosting, which was proposed by the researchers at the University of Washington. It is a library written in C++ which optimizes the training for Gradient Boosting.

Bagging:

A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction.

Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it.

Each base classifier is trained in parallel with a training set which is generated by randomly drawing, with replacement, N examples(or data) from the original training dataset, where N is the size of the original training set.

The training set for each of the base classifiers is independent of each other.

Many of the original data may be repeated in the resulting training set while others may be left out.

Bagging reduces overfitting (variance) by averaging or voting, however, this leads to an increase in bias, which is compensated by the reduction in variance though.

How to preprocess your datasets for XGBoost

Apart from basic data cleaning operations, there are some requirements for XGBoost to achieve top performance. Mainly:

- Numeric features should be scaled
- Categorical features should be encoded

https://xgboost.readthedocs.io/en/stable/python/python_api.html