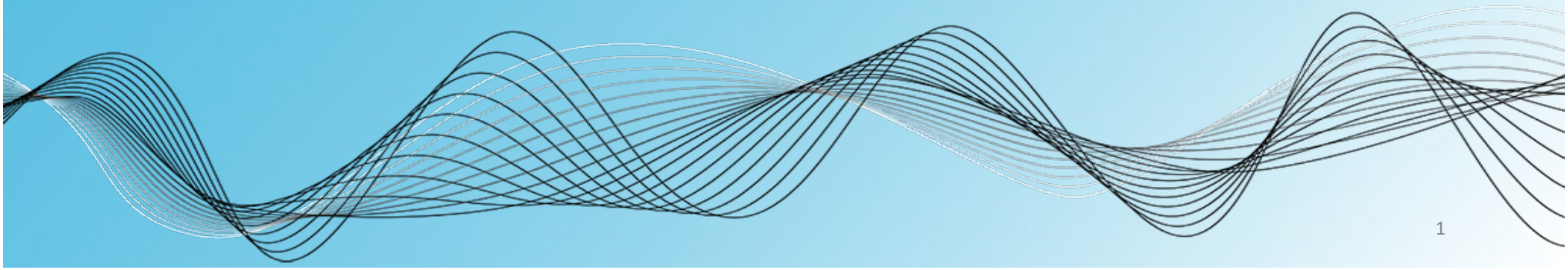
Ensemble Methods



Ensemble Methods

- Imagine that you have received an offer letter from an organization but you are still uncertain about the work culture and ethics of the organization.

How will you finalize your decision whether to join the organization or not?

There can be multiple ways to gather the relevant information like:

a) Checking your LinkedIn network to find if you can get such information (This can be a time taking exercise and yet not reliable)

b) Asking your friend who works in the same domain or in that organization (This can be helpful but still there can be bias associated with that single review)

c) Check websites likes glassdoor or ambitionbox and compare the ratings.(If majority of the reviews are positive then join else not)

Ensemble Methods

- The responses, in third case, would be more generalized and diversified since now you have people with different sets of skills and as it turns out – this is a better approach to get honest ratings than the previous cases we saw.
- With these examples, you can infer that a diverse group of people are likely to make better decisions as compared to individuals.
- Similar is true for a diverse set of models in comparison to single models. **This diversification** in Machine Learning is achieved by a technique called **Ensemble Learning**.

Ensemble Learning

- Ensemble learning helps improve machine learning results by combining several models. This approach allows the production of better predictive performance compared to a single model.
- Basic idea is to learn a set of classifiers (experts) and to allow them to vote.
- The biggest advantage of using ensemble machine learning is that it improves the predictive accuracy significantly.

Simple Ensemble Techniques

There are few ensemble techniques which are simple to implement yet those are very useful which are as follows

- Max Voting
- Averaging
- Weighted Averaging

1) Max Voting

- The max voting method is generally used for classification problems.
- In this technique, multiple models are used to make predictions for each data point.
- The predictions by each model are considered as a **'vote'.** The **predictions** which we get **from the majority of the models** are used as the **final prediction**.

For example, when you asked 5 of your colleagues to provide feedback about the organisation which you are willing to join. we'll assume three of them gave positive feedback while two of them gave it negative feedback. Since the majority gave positive feedback , the final decision will be taken as positive.

We can consider this as taking the mode of all the predictions.

Simple Ensemble Techniques

Results of Max Voting

The result of max voting would be something like this:

Colleague 1	Colleague 2	Colleague 3	Colleague 4	Colleague 5	Final rating
Negative	Positive	Negative	Positive	Positive	Positive

2) Averaging

Similar to the max voting technique, multiple predictions are made for each data point in averaging.

In this method, we take an average of predictions from all the models and use it to make the final prediction. Averaging can be used for **making predictions in regression problems** or while **calculating probabilities for classification problems.**

Suppose in the above example we ask your colleagues to provide ratings out of 5.Then final rating will be average of all the five ratings.

Colleague 1	Colleague 2	Colleague 3	Colleague 4	Colleague 5	Final rating
5	4	5	4	4	4.4

Note : In case of averaging in classification problems predicted probabilities are averaged rather than the predictions and accordingly decision is taken.

Simple Ensemble Techniques

3) Weighted Averaging

- This is an extension of the averaging method.
- All models are assigned different weights defining the importance of each model for prediction.
- For instance, if two of your colleagues are in the same domain (Data science), while others have no prior experience in that domain, then the answers by these two colleagues are given more importance as compared to the other people.

	Colleague 1	Colleague 2	Colleague 3	Colleague 4	Colleague 5	Final rating
Ratings	5	4	5	4	4	4.4
Weights	0.35	0.35	0.1	0.1	0.1	
Weighted ratings	1.75	1.4	0.5	0.4	0.4	4.45

Advanced Ensemble Techniques

Previously we have simple but effective ensemble methods and now we will see the advanced ensemble techniques which are

- Stacking
- Blending
- Bagging

Stacking

Stacking is an ensemble learning technique that uses predictions from multiple models (for example decision tree, knn or svm) to build a new model. This model is used for making predictions on the test set.

Below is a step-wise explanation for a simple stacked ensemble:

Step 1) The train set is split into 10 parts

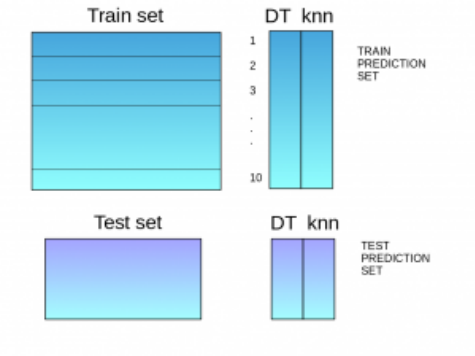
Advanced Ensemble Techniques

Step 2) A base model (suppose a decision tree) is fitted on 9 parts and predictions are made for the 10th part. This is done for each part of the train set.

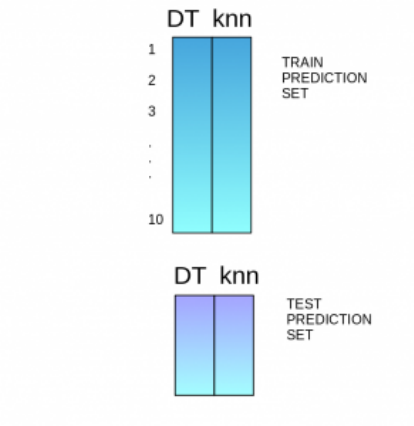
Step 3) The base model (in this case, decision tree) is then fitted on the whole train dataset.
Step 4) Using this model, predictions are made on the test set.

Advanced Ensemble Techniques

Step 5) Steps 2 to 4 are repeated for another base model (say knn) resulting in another set of predictions for the train set and test set.



Step 6) The predictions from the train set are used as features to build a new model.



Step 7) This model is used to make final predictions on the test prediction set.

Advanced Ensemble Techniques

Blending

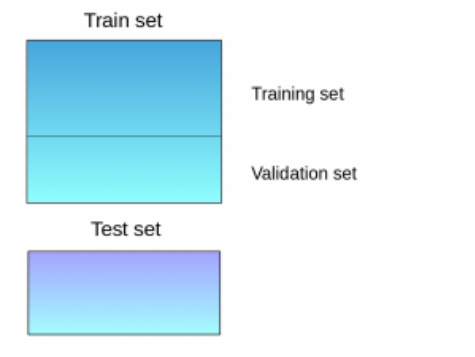
Blending follows the same approach as stacking but uses only a **holdout (validation)** set from the train set to make predictions.

In other words, **unlike stacking, the predictions are made on the holdout set only.**

The holdout set and the predictions are used to build a model which is run on the test set.

Steps of the blending process:

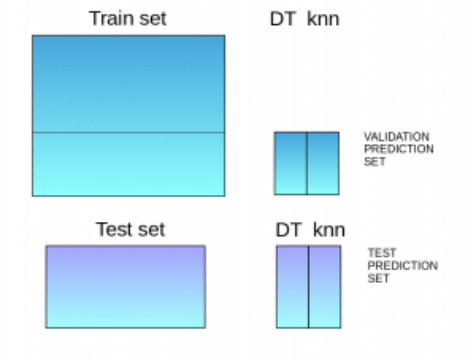
Step 1) The train set is split into training and validation sets



Step 2) Model(s) are fitted on the training set

Advanced Ensemble Techniques

Step 3) The predictions are made on the validation set and the test set.



Step 4) The validation set and its predictions are used as features to build a new model.

Step 5) This model is used to make final predictions on the test and meta-features.

Advanced Ensemble Techniques

Bagging

- The idea behind bagging is combining the results of multiple models (for instance, all decision trees) to get a generalized result.
- But If you create all the models on the same set of data and combine it, will it be useful?
- There is a high chance that these models will give the same result since they are getting the same input.
- So how can we solve this problem?
- One of the techniques is **bootstrapping.**

Bootstrapping

- Bootstrapping is a sampling technique in which we create subsets of observations from the original dataset, **with replacement**.
- The size of columns of the subsets is the same as the size of the original set.

Bagging **(or Bootstrap Aggregating)** technique uses these **subsets (bags) to get a fair idea of the distribution** (complete set).

The size of rows of subsets created for bagging may be less than the original set.

Advanced Ensemble Techniques

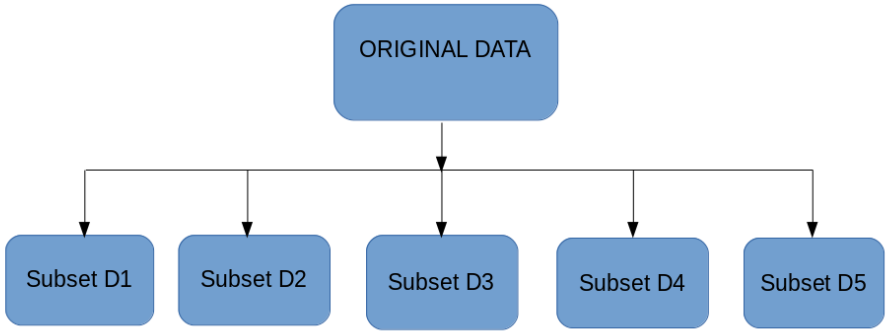
Steps involved in Bagging

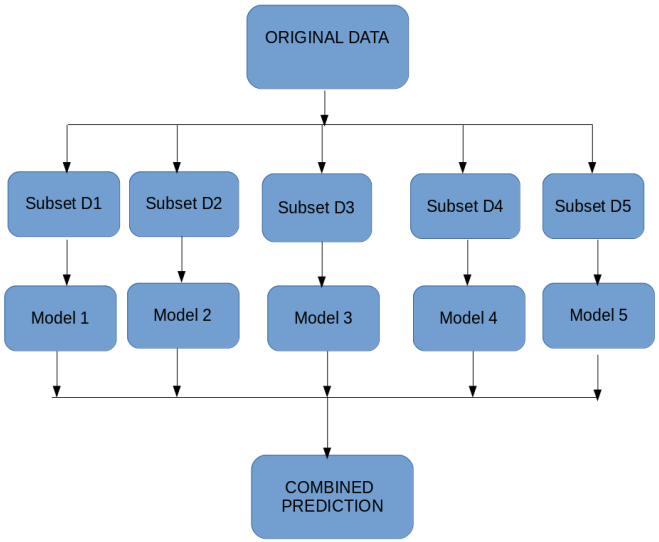
1) Multiple subsets are created from the original dataset, selecting observations with replacement.

2) A base model (weak model) is created on each of these subsets.

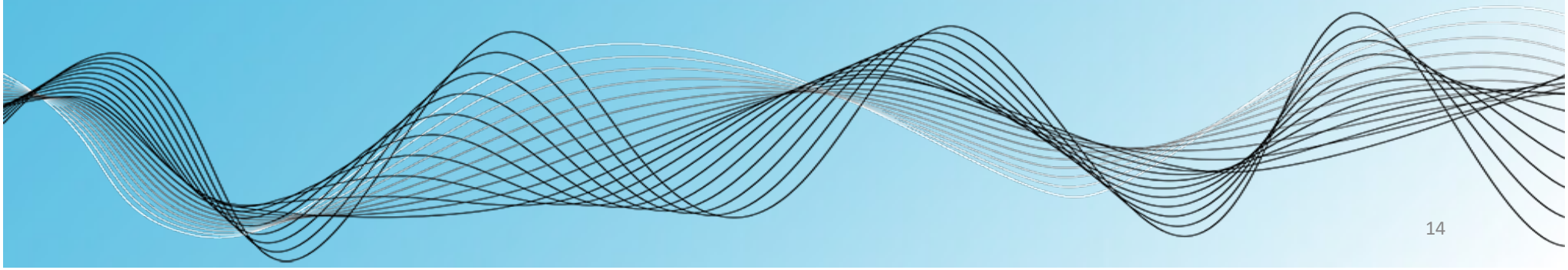
3) The models run in parallel and are independent of each other.

4) The final predictions are determined by combining the predictions from all the models.





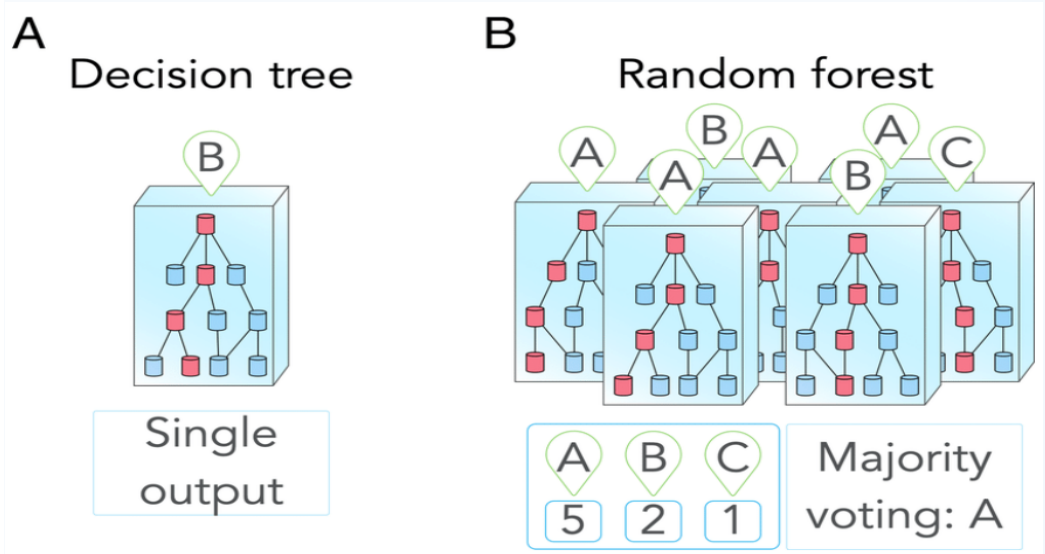
Random Forest



Random Forest

- Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML.
- We know that a forest comprises numerous trees, and the more trees more it will be robust. Similarly, the greater the number of trees in a Random Forest Algorithm, the higher its accuracy and problem-solving ability.
- Random Forest is a classifier that contains several decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset.
- It is based on the concept of ensemble learning which is a process of combining multiple classifiers to solve a complex problem and improve the performance of the model.

Decision Tree vs Random Forest



DECISION TREE VERSUS RANDOM FOREST

DECISION TREE	RANDOM FOREST
A decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility	An ensemble learning method that operates by constructing a multitude of decision trees at training time and outputting the class depending on the individual trees
There is a possibility of overfitting	Reduced risk of overfitting
Gives less accurate results	Gives more accurate results
Simpler and easier to understand, interpret and visualize	Comparatively more complex

Difference in Decision Tree and Individual RF Tree

Are decision trees in Random Forest different from regular decision trees?

- It's easy to get confused by a single decision tree and a decision forest.
- Random Forest looks like a decision forest would be a bunch of single decision trees but It's a bunch of single decision trees but all of the **trees are mixed together randomly instead of separate trees growing individually.**
- When using a regular decision tree, you would input a training dataset with features and labels and it will formulate some set of rules which it will use to make predictions.
- If you entered that same information into a Random Forest algorithm, it will randomly select observations and features to build several decision trees and then average the results.

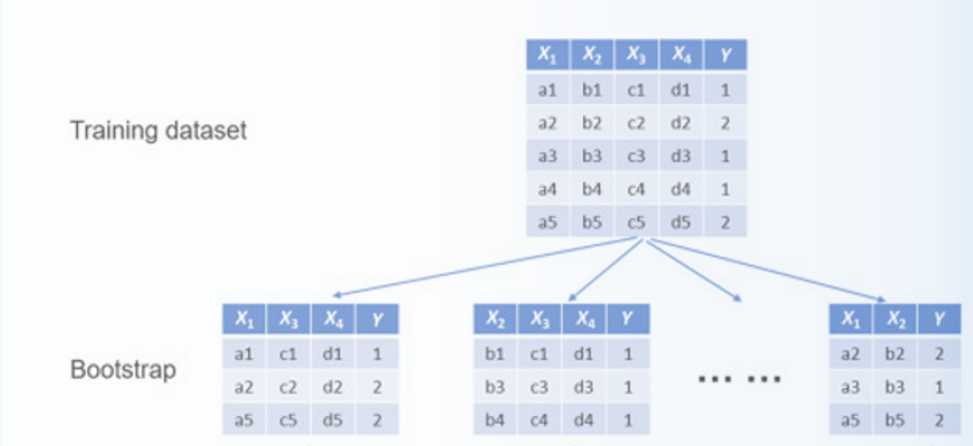
Working of Random Forest

Suppose we have a dataset which consist of **n observations** and **m features**.

Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

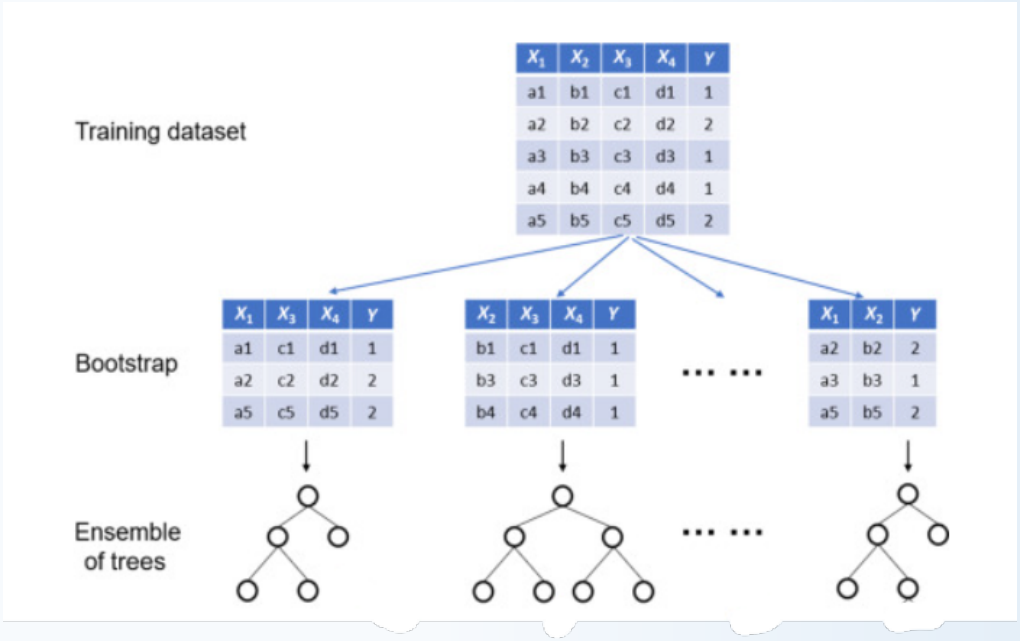
The Working process can be explained in the below steps

Step-1: Select any **k (k<=n) rows** (observations) and **p (p<=m) features** from the training set.



Working of Random Forest

Step-2: Individual decision trees are constructed for each sample.



X_1	X_2	X_3	X_4	Y
a1	b1	c1	d1	1
a2	b2	c2	d2	2
a3	b3	c3	d3	1
a4	b4	c4	d4	1
a5	b5	c5	d5	2

X_1	X_3	X_4	Y
a1	c1	d1	1
a2	c2	d2	2
a5	c5	d5	2

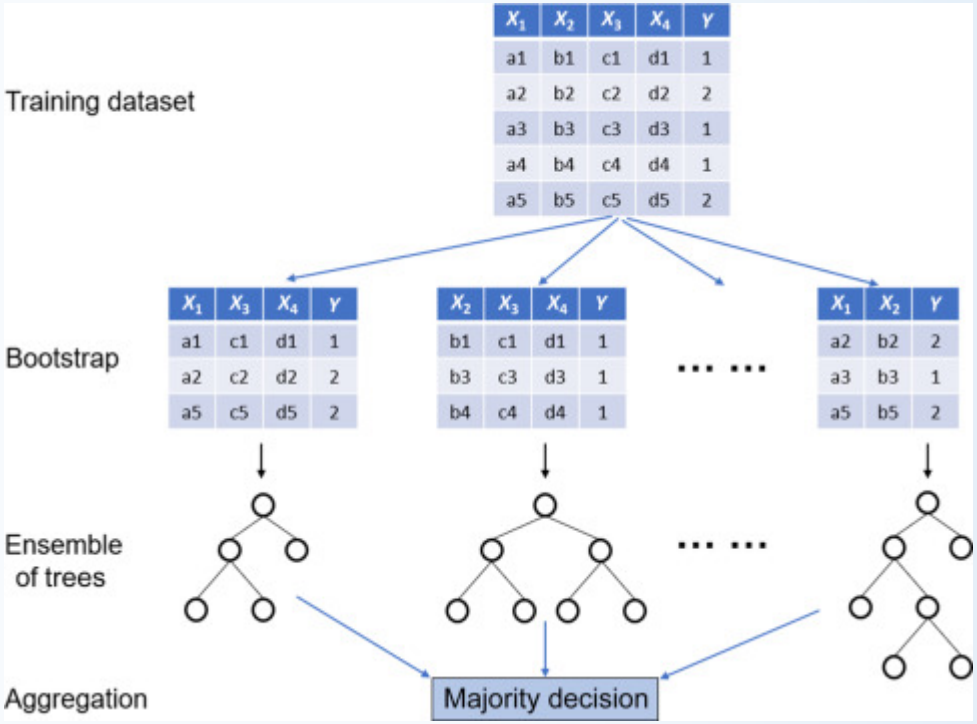
X_2	X_3	X_4	Y
b1	c1	d1	1
b3	c3	d3	1
b4	c4	d4	1

X_1	X_2	Y
a2	b2	2
a3	b3	1
a5	b5	2

Step 3: Each decision tree will generate an output.

Working of Random Forest

Step 4: Final output is considered based on Majority Voting or Averaging for Classification and regression respectively.



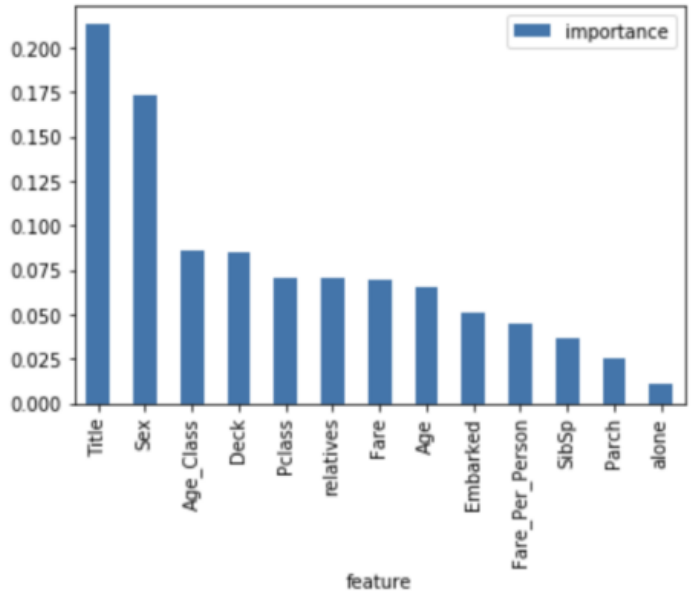
Note : Random Forest Regressor works similar to Random Forest Classifier except for the fact that in case of regression the predictions generated from each tree are averaged rather than max voting.

Random Forest Feature Importance

- Another great quality of the random forest algorithm is that it is very easy to measure the relative importance of each feature on the prediction.
- Sklearn provides a great tool for this that measures a feature's importance by looking at how much the tree nodes that use that feature reduce impurity across all trees in the forest.
- It computes this score automatically for each feature after training and scales the results so the sum of all importance is equal to one.
- By looking at the feature importance you can decide which features to possibly drop because they don't contribute enough (or sometimes nothing at all) to the prediction process.
- This is important because a general rule in machine learning is that the more features you have the more likely your model will suffer from overfitting and vice versa.

Random Forest Feature Importance

feature	importance
Title	0.213
Sex	0.173
Age_Class	0.086
Deck	0.085
Pclass	0.071
relatives	0.070
Fare	0.069
Age	0.065
Embarked	0.051
Fare_Per_Person	0.045
SibSp	0.037
Parch	0.025
alone	0.011



Random Forest Important Parameters

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

Hyperparameters that increases predictive power:

1. n_estimators– number of trees the algorithm builds before averaging the predictions.

2. max_features– maximum number of features random forest considers splitting a node.

3. min_sample_leaf– determines the minimum number of leaves required to split an internal node.

Random Forest Important Parameters

Hyperparameters that increases speed of modelling:

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

2. ***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.

3. ***oob_score*** – OOB means out of the bag. It is a random forest cross-validation method. In this one-third of the sample is not used to train the data instead used to evaluate its performance. These samples are called out of bag samples.

Sklearn Parameters and default values

n_estimators: default = 100

The number of trees in the forest.

criterion : {“gini”, “entropy”, “log_loss”}, default=”gini”

The function to measure the quality of a split.

max_depth : default = None

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than **min_samples_split** samples.

min_samples_split : default = 2

The minimum number of samples required to split an internal node:

min_samples_leaf : default = 1

The minimum number of samples required to be at a leaf node.

Sklearn Parameters and default values

max_features: {“sqrt”, “log2”, None} default=”sqrt”

The number of features to consider when looking for the best split

oob_score: default=False

Whether to use out-of-bag samples to estimate the generalization score.

n_jobs: default=None

The number of jobs to run in parallel. None means 1. -1 means using all processors.

random_state: default=None

Controls both the randomness of the bootstrapping of the samples used when building trees (if bootstrap=True) and the sampling of the features to consider when looking for the best split at each node (if max_features < n_features).

Sklearn Parameters and default values

ccp_alpha: default=0.0

Complexity parameter used for Minimal Cost-Complexity Pruning.

max_samples: default=None

If bootstrap is True, the number of samples to draw from X to train each base estimator.

Important Characteristics of Random Forest

- **Diversity :** Not all attributes/variables/features are considered while making an individual tree, each tree is different.
- **Immune to the curse of dimensionality:** Since each tree does not consider all the features, the feature space is reduced.
- **Parallelization:** Each tree is created independently out of different data and attributes. This means that we can make full use of the CPU to build random forests.
- **Stability :** Stability arises because the result is based on majority voting/ averaging.

Disadvantages of Random Forest

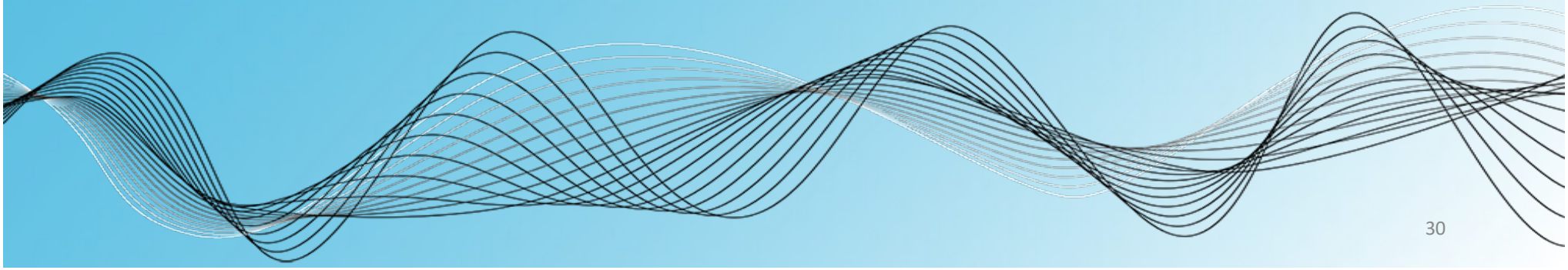
High computation power

- Because random forest uses many decision trees, it can require a lot of memory on larger projects. This can make it slower than some other, more efficient, algorithms.

Prone to overfit

- Sometimes, because this is a decision tree-based method and decision trees often suffer from overfitting, this problem can affect the overall forest.
- This problem is usually prevented by Random Forest by default because it uses random subsets of the features and builds smaller trees with those subsets. This can slow down processing speed but increase accuracy.

Ensemble Methods : Boosting



What is boosting

- Boosting is an ensemble learning method that combines a set of weak learners into strong learners to minimize training errors.
- In boosting, a random sample of data is selected, fitted with a model, and then trained sequentially. That is, each model tries to compensate for the weaknesses of its predecessor.
- Each classifier's weak rules are combined with each iteration to form one strict prediction rule.
- Boosting is an efficient algorithm that converts a weak learner into a strong learner.
- They use the concept of the weak learner and strong learner conversation through the weighted average values and higher votes values for prediction.
- These algorithms use decision stump and margin maximizing classification for processing.

Why we use boosting?

To solve more complex problems, we can not rely on base ML models and we require more advanced techniques.
Suppose that, given a data set of about characteristics of houses, we have to to build a model that can predict if we should purchase the house or not. We will start determining those features which can lead to decision making of purchase.

- Total Carpet area: **Purchase**
- Number of bedroom: **Purchase**
- Distance from airport/railway station: **Don't Purchase**
- Availability of essential facilities (Schools/Hospitals): **Don't purchase**
- Expected return on investment (ROI): **Purchase**

These rules help us whether to purchase the house or not.

However, the prediction would be flawed if we were make decision based on an individual (single) rule.

These rules are called weak learners because these rules are not strong enough to make the decision.

Therefore, to ensure our prediction is more accurate, we can combine the prediction from these weak learners by using the majority rule or weighted average. This makes a strong learner model.

In the above example, we have defined 5 weak learners, and the majority of these give us the prediction that we should purchase the house. Therefore, our final output is to purchase the house.

Boosting vs Bagging

Ensemble learning can be performed in two ways:

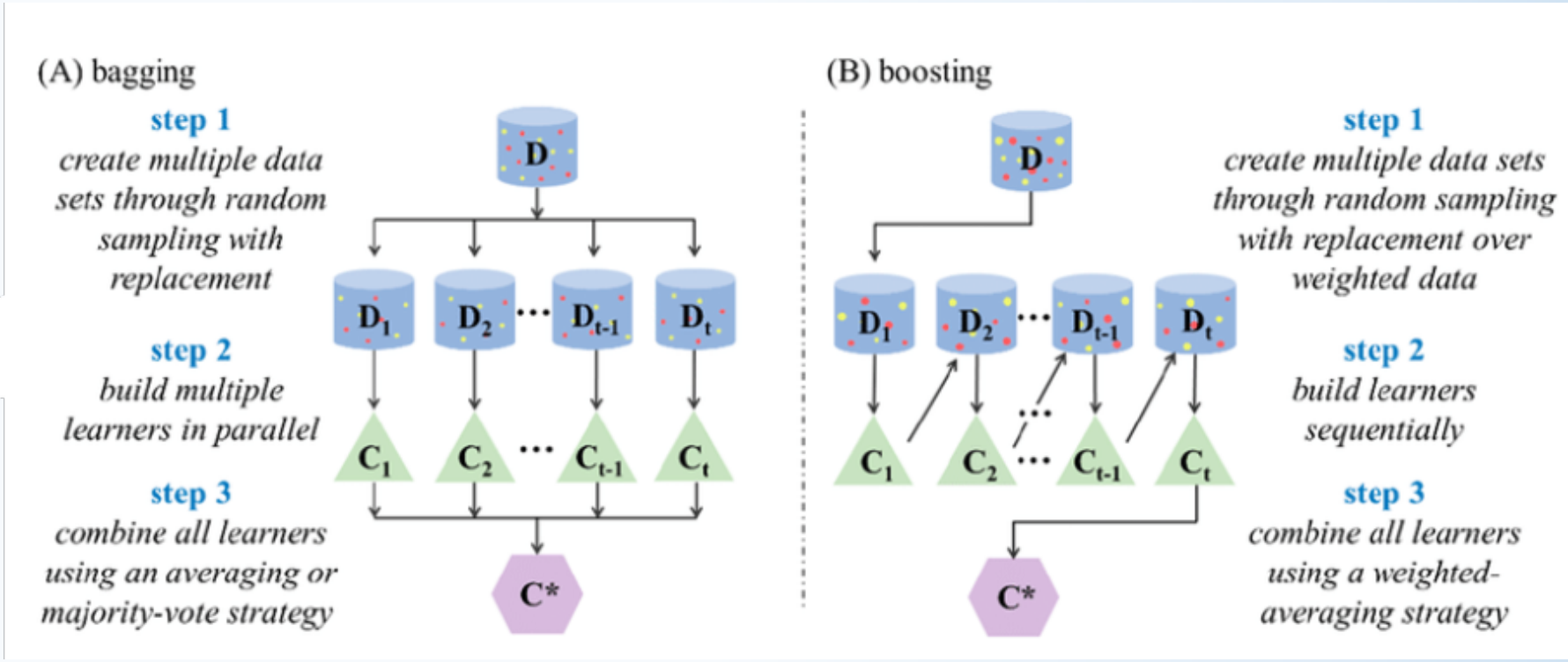
Parallel ensemble

- This is also known as bagging
- Here the weak learners are produced parallelly during the training phase.
- The performance of the model can be increased by parallelly training a number of weak learners on bootstrapped data sets.
- An example of bagging is the Random Forest algorithm.

Sequential ensemble

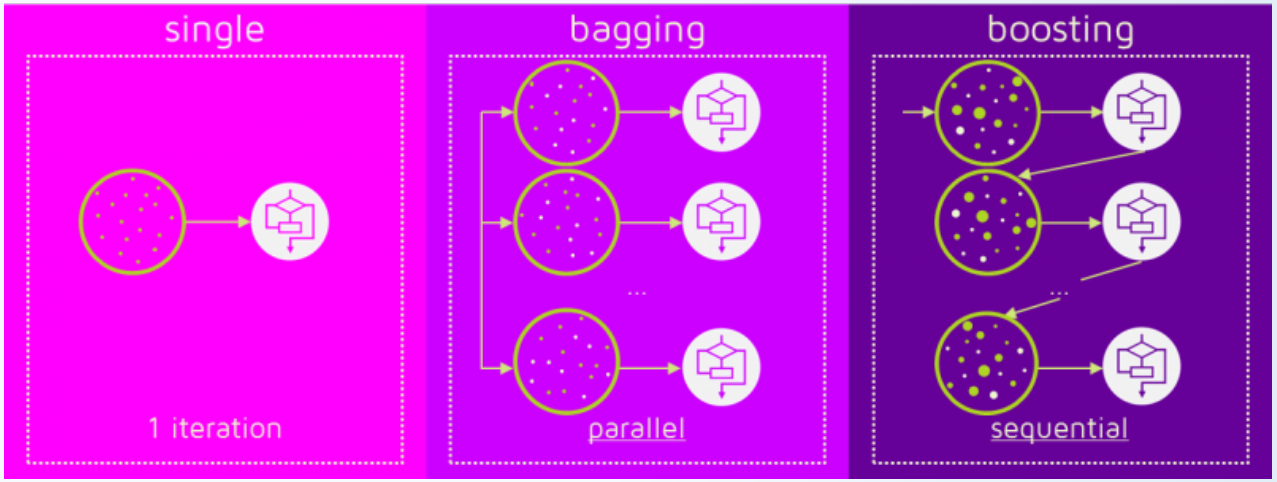
- This is popularly known as boosting,
- Here the weak learners are sequentially produced during the training phase.
- The performance of the model is improved by assigning a higher weightage to the previous, incorrectly classified samples.
- An example of boosting is the AdaBoost algorithm.

Boosting vs Bagging



How boosting Algorithm works

- The basic principle behind the working of the boosting algorithm is to generate multiple weak learners and combine their predictions to form one strong rule.
- These weak rules are generated by applying base Machine Learning algorithms on different distributions of the data set.
- These algorithms generate weak rules for each iteration.
- After multiple iterations, the weak learners are combined to form a strong learner that will predict a more accurate outcome.



How boosting works

Here's how the algorithm works:

Step 1: The base algorithm reads the data and assigns equal weight to each sample observation.

Step 2: False predictions made by the base learner are identified. In the next iteration, these false predictions are assigned to the next base learner with a higher weightage on these incorrect predictions.

Step 3: Repeat step 2 until the algorithm can correctly classify the output.

How boosting works

Finally, it combines the outputs from weak learner and creates a strong learner which eventually improves the prediction power of the model.

Boosting pays higher focus on examples which are mis-classified or have higher errors by preceding weak rules.

Types of Boosting

Boosting methods are focused on iteratively combining weak learners to build a strong learner that can predict more accurate outcomes. As a reminder, a weak learner classifies data slightly better than random guessing.

Boosting algorithms can differ in how they create and aggregate weak learners during the sequential process.

Three popular types of boosting methods include:

- **AdaBoost (Adaptive Boosting)**
- **Gradient Tree Boosting**
- **XGBoost**

Gradient Boost

- Gradient Boosting is also based on sequential ensemble learning. Here the base learners are generated sequentially so that the present base learner is always more effective than the previous one, i.e., and the overall model improves sequentially with each iteration.
- The difference in this boosting type is that the weights for misclassified outcomes are not incremented. Instead, the Gradient Boosting method tries to optimize the loss function of the previous learner by adding a new model that adds weak learners to reduce the loss function.

Gradient Boost

- Start with a simple model
  - Start with a weak learner (e.g., a small decision tree or even a constant mean value).
  - Example: F0(x)=mean(y)
- Compute residuals
  - Calculate the errors (residuals):
  - ri=yi−F0(xi)
  - These residuals represent what's left to learn.
- Fit a new model on residuals
  - Train a new weak learner h1(x)h_1(x)h1(x) to predict these residuals.

Gradient Boost

- Update the model
  - Add the new model's predictions to the previous model:
  - $F1(x)=F0(x)+\eta \cdot h1(x)$
  - Here η (learning rate) controls how much each tree contributes.
- Repeat
  - Compute new residuals and add more trees iteratively:
  - $Fm(x)=Fm-1(x)+\eta \cdot hm(x)$
- Final model
  - The final prediction is the sum of all weak learners.

Gradient Boost

Let's predict **house price** using one feature — area.

Area (sqft)	Price ($1000s)
1000	200
1500	250
2000	300

Step 1 – Initial prediction:
Average price = (200 + 250 + 300)/3 = **250**

Area	Actual	Pred	Residual (Actual - Pred)
1000	200	250	-50
1500	250	250	0
2000	300	250	+50

Gradient Boost

Step 2 – Fit a tree on residuals:

•The tree learns:

- For area < 1500 → predict -50
- For area ≥ 1500 → predict +25 (average of 0 and +50)

Step 3 – Update the model:

$F1(x)=250+0.1\times tree1(x)$

Learning rate $\eta=0.1$:

•For 1000 sqft → New prediction = 250 + 0.1(-50) = 245

•For 1500 sqft → New prediction = 250 + 0.1(0) = 250

•For 2000 sqft → New prediction = 250 + 0.1(50) = 255

- The model slightly moves toward the correct answer.
- Repeat this process for several trees — each one reduces remaining errors.

Gradient Boost - Classifier

- Let's say you're trying to predict whether a customer will buy (1) or not buy (0) a product.

-

Customer	Income	Bought (Y)
A	20k	0
B	30k	0
C	60k	1
D	80k	1

Step 1: Start with a simple guess
The model first makes a **basic prediction**, like "most people didn't buy," so it predicts 0 for everyone.

Customer	True Y	Prediction	Error
A	0	0	✅
B	0	0	✅
C	1	0	❌
D	1	0	❌

Gradient Boost - Classifier

Step 2: Build a small tree to fix those mistakes
Now, GBM builds a small **decision tree** focused on correcting the errors.
That tree learns:
•People with income above 50k are more likely to buy.

Step 3: Update the overall prediction
The model **adds** this new tree's knowledge to the previous one.
Now predictions improve:

Customer	New Prediction	Result
A	still 0	✅
B	still 0	✅
C	now 1	✅
D	now 1	✅

Gradient Boost - Classifier

Step 4: Repeat the process

GBM repeats this process **many times**:

•Compute errors (residuals)

•Train a new tree to fix them

•Add that tree to the model (with a small learning rate so we don't overfit)

Each tree slightly corrects the previous model's mistakes.

"Gradient" means direction of improvement.

GBM uses the gradient of the loss function (i.e., the direction in which the model's predictions should move to reduce errors).

So, each new tree doesn't just guess randomly — it follows the mathematical direction that reduces errors the most.

Gradient Boost

The main idea here is to overcome the errors in the previous learner's predictions. Gradient boosting has three main components:

Loss function: The use of the loss function depends on the type of problem. The advantage of gradient boosting is that there is no need for a new boosting algorithm for each loss function.

Weak learner: In gradient boosting, decision trees are used as a weak learners. A regression tree is used to give true values, which can combine to create correct predictions. Like in the AdaBoost algorithm, small trees with a single split are used, i.e., decision stump. Larger trees are used for large levels

Additive Model: Trees are added one at a time in this model. Existing trees remain the same. During the addition of trees, gradient descent is used to minimize the loss function.

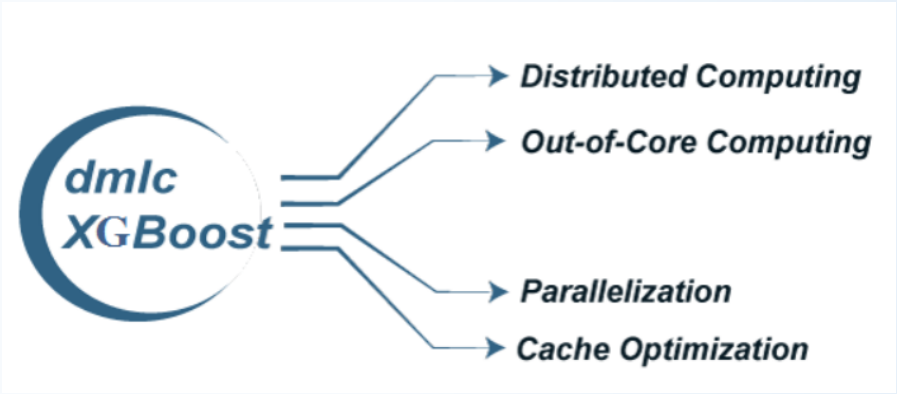
Like AdaBoost, Gradient Boosting can also be used for classification and regression problems.

XGBoost

XGBoost is an advanced gradient boosting method. XGBoost, falls under the Distributed Machine Learning Community (DMLC) category.

The main aim of this algorithm is to increase the speed and efficiency of computation. The Gradient Descent Boosting algorithm computes the output slower since they sequentially analyze the data set. Therefore XGBoost is used to boost or extremely boost the model's performance.

XGBoost is designed to focus on computational speed and model efficiency. The main features provided by XGBoost are:



XGBoost

Parallel Processing: XG Boost provides Parallel Processing for tree construction which uses CPU cores while training.

Cross-Validation: XG Boost enables users to run cross-validation of the boosting process at each iteration, making it easy to get the exact optimum number of boosting iterations in one run.

Cache Optimization: It provides Cache Optimization of the algorithms for higher execution speed.

Distributed Computing: For training large models, XG Boost allows Distributed Computing.

Benefits of Boosting

The boosting method presents many advantages and challenges for classification or regression problems. The benefits of boosting include:

Ease of Implementation: Boosting can be used with several hyper-parameter tuning options to improve fitting. No data preprocessing is required, and boosting algorithms have built-in routines to handle missing data. In Python, the sci-kit-learn library of ensemble methods makes it easy to implement the popular boosting methods, including AdaBoost, XGBoost, etc.

Reduction of bias: Boosting algorithms combine multiple weak learners in a sequential method, iteratively improving upon observations. This approach can help to reduce high bias, commonly seen in shallow decision trees and logistic regression models.

Computational Efficiency: Since boosting algorithms have special features that increase their predictive power during training, it can help reduce dimensionality and increase computational efficiency.

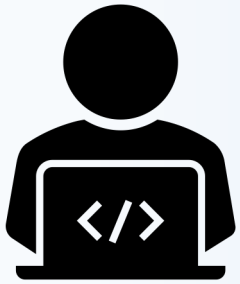
Challenges of Boosting

Overfitting: There's some dispute in the research around whether or not boosting can help reduce overfitting or make it worse. We include it under challenges because in the instances that it does occur, predictions cannot be generalized to new datasets.

Intense computation: Sequential training in boosting is hard to scale up. Since each estimator is built on its predecessors, boosting models can be computationally expensive, although XGBoost seeks to address scalability issues in other boosting methods. Boosting algorithms can be slower to train when compared to bagging, as a large number of parameters can also influence the model's behavior.

Vulnerability to outlier data: Boosting models are vulnerable to outliers or data values that are different from the rest of the dataset. Because each model attempts to correct the faults of its predecessor, outliers can skew results significantly.

Real-time implementation: You might find it challenging to use boosting for real-time implementation because the algorithm is more complex than other processes. Boosting methods have high adaptability, so you can use various model parameters that immediately affect the model's performance.



Keep Learning…… Keep Coding…… Keep going……