Table of Contents

Ensembles

Introduction and Motivation for Ensemble Techniques

In machine learning, no single model is perfect. Each algorithm comes with its own strengths and weaknesses: some capture patterns very well but overfit to noise, while others are more stable but may miss finer details. This raises an important question: instead of relying on just one model, can we combine the knowledge of many?

This is the central idea of **ensemble methods**. An ensemble brings together multiple models, often referred to as *base learners*, and combines their predictions. Just as a panel of doctors may provide a more reliable diagnosis than a single physician, ensembles often yield more accurate and robust results than any individual model alone.

Ensemble methods have become a cornerstone of modern machine learning. They underpin many of the best-performing models in practice, from spam detection and fraud analysis to recommendation engines and competition-winning solutions on platforms like Kaggle.

Types of Ensemble Methods

Ensemble learning methods differ in *how* they combine multiple models. The three most widely used approaches are **bagging**, **boosting**, and **stacking**.

- Bagging (Bootstrap Aggregating)
  - Builds several models on different random parts of the data and then combines them
  - Examples: random forest, bagged k-nearest neighbours (kNN)
- Boosting
  - Trains models one after another, each one correcting the mistakes of the previous
  - Examples: AdaBoost, XGBoost, LightGBM
- Stacking (Stacked Generalisation)
  - Combines the predictions from different types of models using another model to decide the final output
  - Examples: Stacked classifiers and regressors where models like logistic regression, support vector machines (SVMs), and neural networks are combined

Ensembles in the Context of Decision Trees

Decision trees are popular base models for ensembles because they are flexible, non-parametric, and capable of capturing complex, non-linear relationships in data. They handle both numerical and categorical variables, work well with missing values, and require little preprocessing, making them versatile building blocks.

At the same time, individual trees tend to overfit and can be highly unstable. This weakness becomes a strength in ensembles: when many diverse trees are combined, their variations cancel out, leading to models that are accurate, robust, and scalable. This combination of simplicity at the base level and power when aggregated makes trees the backbone of many successful ensemble methods.

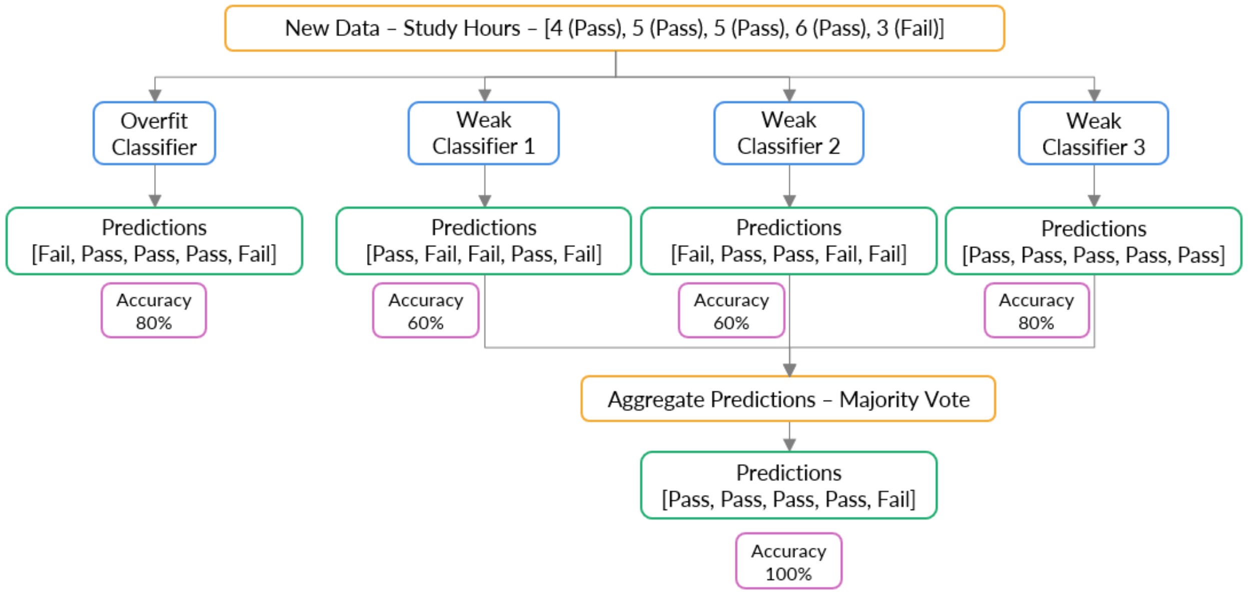
Bagging

The main idea behind bagging, or *bootstrap aggregating*, is that a single model may give an unreliable picture of the data because it learns patterns too closely tied to one sample. By training many models on different random samples, we can reduce this instability and get a more reliable outcome.

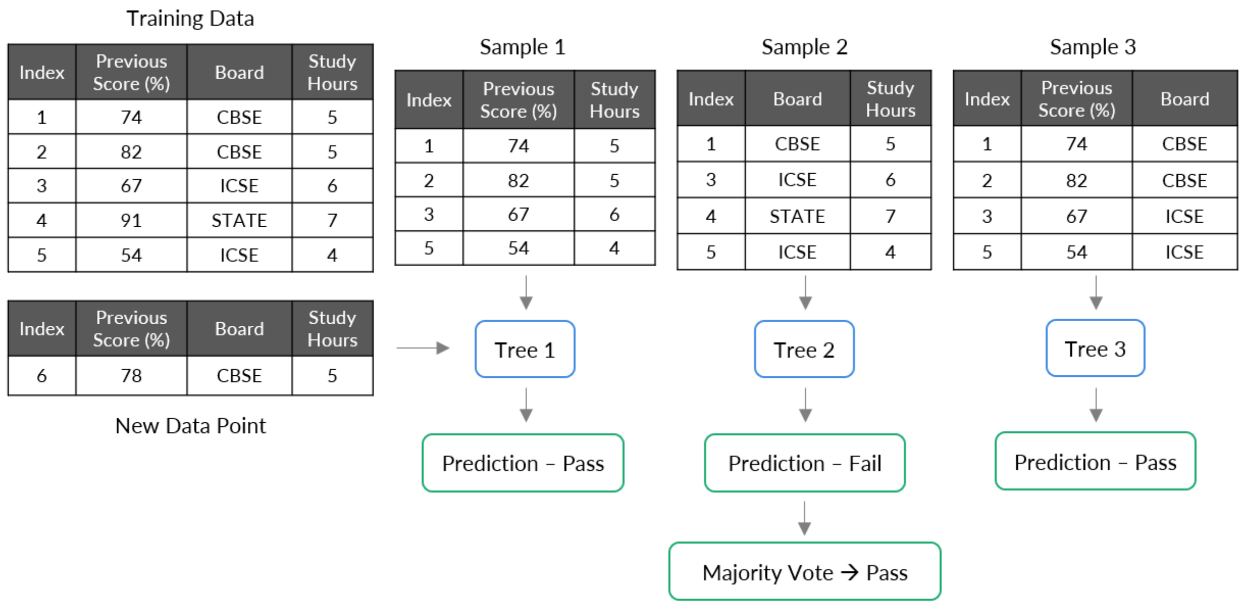
Think of it like asking multiple people to guess the weight of an object. One person might guess too high, another too low, but if we take the average of everyone's guesses, we usually end up very close to the true value. Bagging works in the same way: it averages predictions for regression or takes a majority vote for classification, smoothing out the extremes.

This process reduces variance — the tendency of a single model to fluctuate wildly depending on the training data. By combining many "noisy" predictors, bagging produces a stronger and more stable model that is less likely to overfit and more likely to generalise well to unseen data.

Consider the following as an example of how bagging operates. On the left, we have an overfit classifier which is makes predictions on its own. On the right, we instead have three weak classifiers aggregating their predictions via majority vote. You can see how, although these classifiers are weaker, they outperform the lone classifier.


New Data – Study Hours – [4 (Pass), 5 (Pass), 5 (Pass), 6 (Pass), 3 (Fail)]
Overfit Classifier
Weak Classifier 1
Weak Classifier 2
Weak Classifier 3
Predictions [Fail, Pass, Pass, Pass, Fail]
Predictions [Pass, Fail, Fail, Pass, Fail]
Predictions [Fail, Pass, Pass, Fail, Fail]
Predictions [Pass, Pass, Pass, Pass, Pass]
Accuracy 80%
Accuracy 60%
Accuracy 60%
Accuracy 80%
Aggregate Predictions – Majority Vote
Predictions [Pass, Pass, Pass, Pass, Fail]
Accuracy 100%

Below is an example of how trees work in tandem to detect if a learner is likely to pass or fail in their exams depending on their previous score, their board of education, and the number of hours they study. Note that this is a simplified example and the actual mechanisms would vary depending on the algorithm used.



Training Data

Index	Previous Score (%)	Board	Study Hours
1	74	CBSE	5
2	82	CBSE	5
3	67	ICSE	6
4	91	STATE	7
5	54	ICSE	4

Index	Previous Score (%)	Board	Study Hours
6	78	CBSE	5

New Data Point

Sample 1

Index	Previous Score (%)	Study Hours
1	74	5
2	82	5
3	67	6
5	54	4

Sample 2

Index	Board	Study Hours
1	CBSE	5
3	ICSE	6
4	STATE	7
5	ICSE	4

Sample 3

Index	Previous Score (%)	Board
1	74	CBSE
2	82	CBSE
3	67	ICSE
5	54	ICSE

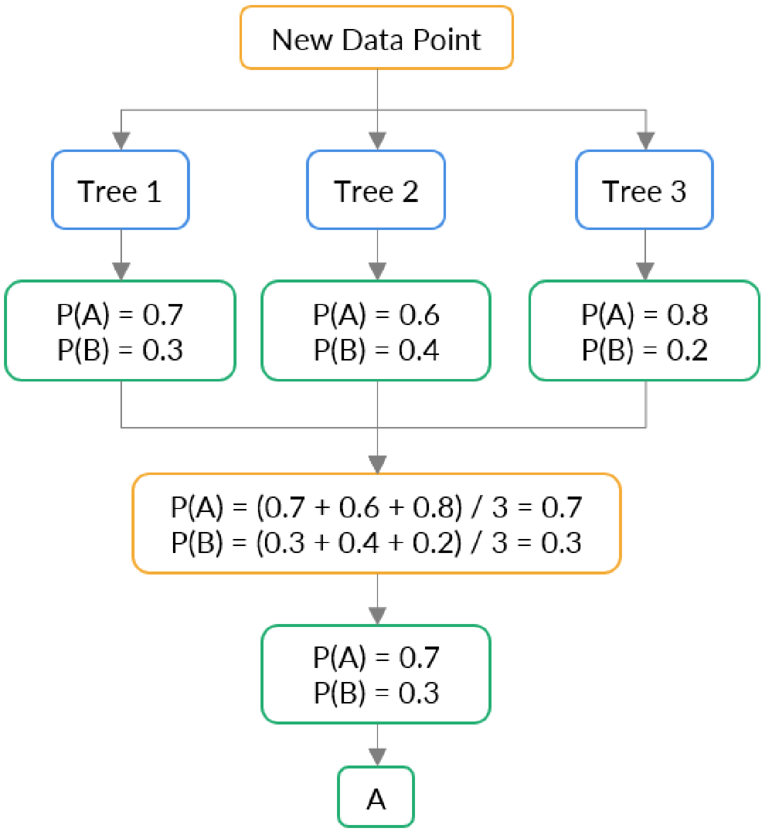
Random Forests

A random forest is an ensemble of decision trees built using the principles of bagging with an extra layer of randomness. Instead of growing just one tree on the full dataset, it grows many trees, each trained on a different random subset of the data created through bootstrapping.

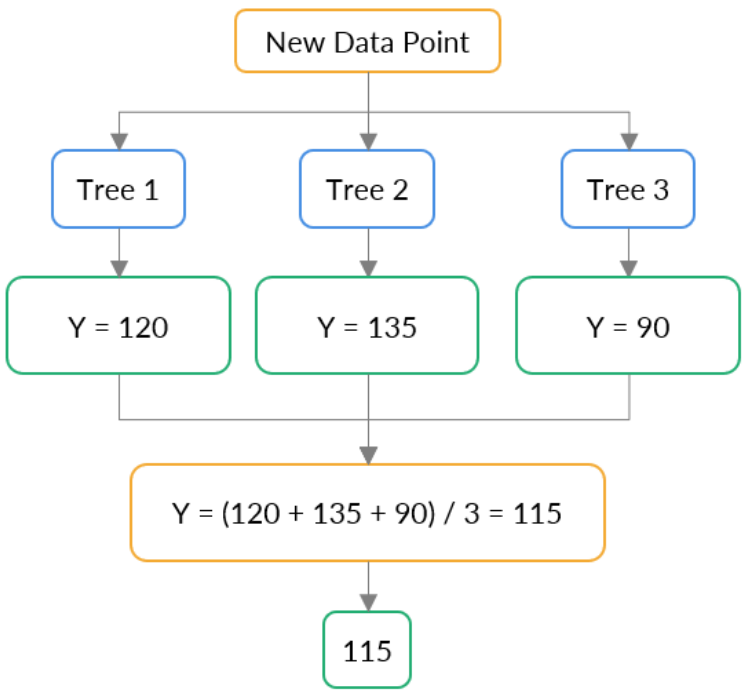
When each tree is trained, it does not consider all available features at every split. Instead, only a **random subset of features** is chosen for each decision point. This ensures the trees are diverse, as different features drive different splits across the forest.

Once all the trees are built, their predictions are combined: by **majority vote** in classification tasks, or by **averaging** in regression tasks. The combination reduces overfitting, improves stability, and leverages the collective wisdom of many varied trees to achieve stronger performance than any single tree alone.

Consider the following example of a single new data point being classified using a random forest. Here, there are two classes: A and B. We can see how the predicted probabilities for the point being classified as belonging to A or B are aggregated using simple averaging. The mechanism is similar for multiclass classification.



Consider the following example for the regression case, showing once again how averages are used to simulate majority vote.



Evaluating Random Forests

Evaluating a random forest follows the same principles as other supervised learning models. Its performance is typically assessed on unseen test data using standard metrics such as **accuracy, precision, recall,** and **F1-score** for classification, or **mean squared error** and **R^2** for regression. These measures help determine how well the model generalises beyond the training set.

One advantage of Random Forests is that they provide additional built-in evaluation tools. The **out-of-bag (OOB) error** is an internal estimate of prediction error, obtained from the samples not included in each tree's bootstrap dataset. This gives a reliable performance estimate without needing a separate validation set.

Another useful feature are the **feature importance scores**, which rank variables by how much they contribute to the predictive power of the model, offering interpretability alongside performance.

Advantages and Disadvantages of Random Forests

In the table below are some of the advantages and disadvantages of random forests:

Advantages of Random Forests	**Disadvantages of Random Forests**

High accuracy and robustness; generalises well to unseen data	Less interpretable than a single decision tree
Handles both numerical and categorical data with minimal preprocessing	Computationally intensive; requires more memory and processing power
Reduces overfitting by combining multiple diverse trees	Slower predictions due to many trees being evaluated
Provides feature importance scores for interpretability	Performs poorly when asked to extrapolate beyond training data
Can handle large datasets and missing values effectively	May be less efficient than simpler models on smaller datasets

Key Hyperparameters of Random Forests

Below are some of the key hyperparameters of random forests. Do note that the availability of these parameters as well as their names in Python would depend on the library used and varies between different implementations.

- **Number of trees:** This determines how many decision trees make up the forest
- **Maximum tree depth:** This defines how deep each tree is allowed to grow before stopping
- **Minimum samples per split:** This sets the smallest number of data points needed to create a split in a node
- **Minimum samples per leaf:** This specifies the least number of data points required in a terminal (leaf) node
- **Number of features considered at each split:** This controls how many features each tree examines when finding the best split
- **Sampling method:** This indicates whether each tree is trained on a random sample of the data, with or without replacement
- **Sample size per tree:** This determines the proportion or count of data used to train each tree
- **Randomness control:** This manages how random sampling and feature selection are handled to ensure diversity among trees
- **Class or sample weighting:** This adjusts the influence of certain classes or samples, which is useful for imbalanced data

- **Parallel processing:** This specifies how many computational resources or processors are used when building the forest

In addition, you may also tune for the parameters of individual decision trees used by the forest.

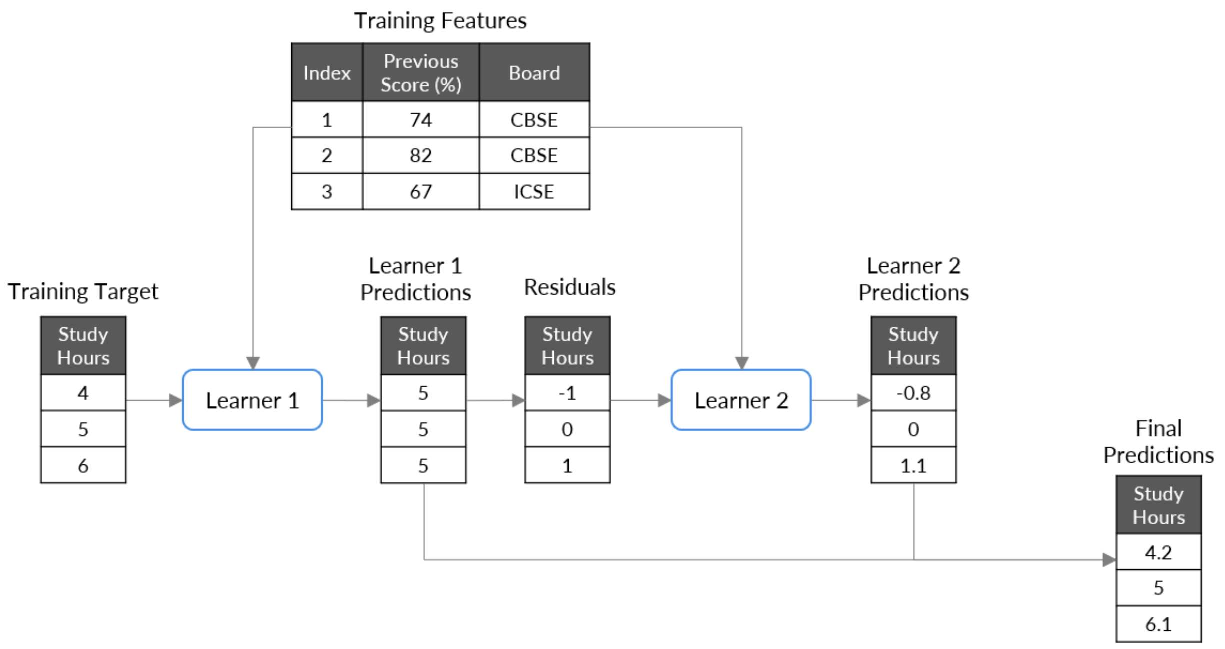
Boosting

Boosting builds on a simple yet powerful idea: instead of training many models independently, as in bagging, it trains them **sequentially,** with each new model learning from the mistakes of the ones before it. The goal is to gradually turn a collection of weak learners into one strong, accurate predictor.

Imagine a classroom where several tutors help a student master a subject. The first tutor explains the basics, but the student still struggles with some topics. The next tutor focuses specifically on those weak areas, and each subsequent tutor continues to refine the student’s understanding based on what’s still not clear. By the end, the student has benefited from many small corrections — each one targeted to earlier errors.

In the same way, boosting gives more attention to data points that earlier models handled poorly. Each model tries to correct previous errors, and their combined effort produces a final model that is far more precise. This process reduces bias — the systematic error from overly simple models — while maintaining low variance. The result is a model that learns deeply from its mistakes, offering high accuracy and strong generalisation across diverse datasets.

Given below is a simple example of how boosting occurs using two learners. The first learner is only predicting the mean of the training target, while the next learner trains to predict the residuals left by this prediction. Note that this is a simplified example not accounting for factors such as the learning rate. The actual mechanisms would vary depending on the algorithm used.



Gradient Boosting

Gradient boosting is an ensemble technique that builds models sequentially, with each new model attempting to correct the errors of the combined models before it. Unlike random forests, where all trees are built independently, gradient boosting creates a chain of models that learn from the residual mistakes of the previous ones.

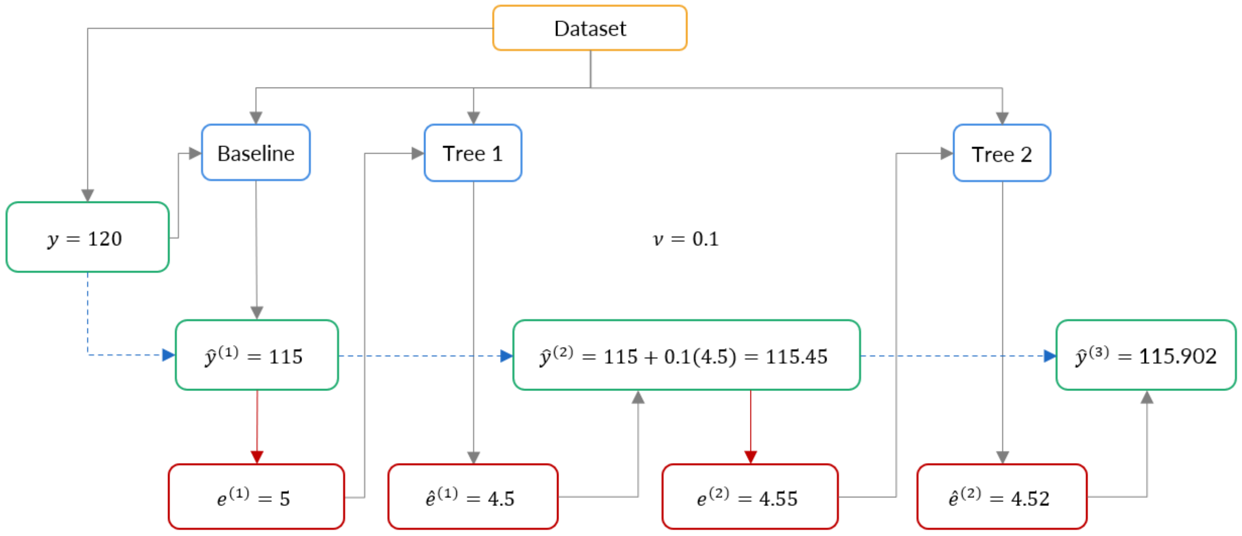
At the start, a simple model makes an initial prediction, denoted as $\hat{y}^{(1)}$. This simple model is the baseline or naïve model and often simply predicts the mean of the data. The difference between the actual and predicted values is the error or residual, $e^{(1)} = y - \hat{y}^{(1)}$.

The next learner is then trained to **predict this residual,** producing $\hat{e}^{(1)}$. The model's prediction is updated as $\hat{y}^{(2)} = \hat{y}^{(1)} + \nu\hat{e}^{(1)}$, where ν (the learning rate or step size) controls how much correction is applied at each step.

After this update, a new residual is computed as $e^{(2)} = y - \hat{y}^{(2)}$, and the process continues for m learners. The final prediction after m iterations is $\hat{y}^{(m)} = \hat{y}^{(m-1)} + \nu\hat{e}^{(m-1)}$.

This gradual, step-by-step improvement ensures that the model learns cautiously and avoids overfitting. Each learner moves the predictions in the direction that most reduces the loss function — this is what the "gradient" in gradient boosting refers to.

Below is an example of how predictions are made for a single new point using simple gradient boosting comprising of a baseline model followed by two tree learners.



The classification case would follow a similar mechanism but would involve probabilities representing the likelihood of each class.

Through this targeted correction process, gradient boosting reduces bias and achieves high accuracy. When tuned properly, it often outperforms other ensemble methods, though it requires careful control to prevent overfitting.

Evaluating Gradient Boosting

Evaluating a boosting model follows the same principles as other supervised learning algorithms. Its performance is assessed on unseen test data using standard metrics such as **accuracy, precision, recall,** and **F1-score** for classification tasks, or **mean squared error** and **R^2** for regression. These metrics reveal how effectively the model generalises beyond the training data.

Another powerful diagnostic are the **feature importances**, which show how much each feature contributes to reducing the overall loss. This allows for a clearer interpretation of which variables drive the model's predictions.

Finally, because boosting models are sensitive to overfitting, techniques such as early stopping—where training halts once validation performance no longer improves—are used to fine-tune performance while maintaining generalisability.

Advantages and Disadvantages of Gradient Boosting

In the table below are some of the advantages and disadvantages of gradient boosting:

Advantages of Gradient Boosting	Disadvantages of Gradient Boosting
Produces highly accurate models by sequentially correcting errors from previous learners	More prone to overfitting if the learning rate is too high or the model is over-trained
Handles both classification and regression problems effectively	Computationally expensive due to sequential training of learners
Works well with weak learners (e.g., shallow trees) to create a strong ensemble	Slower to train compared to parallel methods like Random Forests
Focuses more on difficult-to-predict samples, improving predictive performance	Sensitive to noise and outliers, as these can be repeatedly emphasised during boosting
Can incorporate regularisation techniques (like shrinkage or subsampling) to improve generalisation	Requires careful tuning of hyperparameters such as learning rate, number of estimators, and tree depth

Key Hyperparameters of Gradient Boosting

Below are some of the key hyperparameters commonly used in gradient boosting models. Do note that the availability and exact naming of these parameters can vary across different libraries and implementations.

- **Number of estimators:** Determines how many weak learners (typically decision trees) are added sequentially to build the ensemble
- **Learning rate:** Controls the contribution of each learner to the final model; lower values make learning slower but often yield better generalisation
- **Maximum tree depth:** Defines how complex each individual tree can be, influencing both bias and variance
- **Minimum samples per split:** Sets the minimum number of samples required to split an internal node within each tree
- **Minimum samples per leaf:** Specifies the minimum number of samples that must be present in a leaf node
- **Subsample ratio:** Indicates the fraction of the training data randomly selected for each boosting round, helping reduce overfitting
- **Loss function:** Defines the error metric that the algorithm minimises, such as mean squared error for regression or logistic loss for classification

- **Number of features considered per split:** Controls how many features are randomly selected for evaluating the best split at each node
- **Regularisation parameters:** Include penalties such as L1 or L2 terms to constrain model complexity and improve generalisation
- **Early stopping criteria:** Specifies when training should stop if performance on a validation set stops improving
- **Randomness control:** Manages the reproducibility of results and the random selection of samples or features during training

In addition, you may also tune hyperparameters related to the base learners used within the boosting framework, such as decision tree depth or leaf configuration.

Bagging vs Boosting

Here are some of the differences between bagging and boosting:

Bagging	Boosting
Trains learners independently in **parallel** on random subsets of data	Trains learners **sequentially**, each correcting the errors of the previous ones
Improves predictions by **aggregating** using averages	Improves predictions in a **step-by-step** manner
All learners are given **equal weight** in the final model	Learners may be **weighted** based on their performance
Less prone to overfitting, works well with **moderately strong** learners (full trees)	More prone to overfitting if not tuned, works best with **weak** learners (shallow trees)

Applications of Ensembles

Ensemble methods have found wide application across domains because of their ability to improve prediction accuracy and stability. In finance, they are used for credit scoring, fraud detection, and stock trend prediction, where combining multiple models helps manage uncertainty in data and reduce the risk of overfitting. By blending the outputs of different algorithms, ensemble models provide a more reliable assessment of risk or opportunity than any single model could achieve.

In healthcare, ensembles are used for diagnostic prediction, disease classification, and treatment outcome analysis. Models such as random forests and gradient boosting can process complex and high-dimensional clinical data to identify subtle patterns that might be missed by traditional techniques. They also play an important role in genomics, medical imaging, and patient survival analysis, where accuracy and interpretability are both essential.

Within marketing and customer analytics, ensemble methods power recommendation systems, churn prediction, and customer segmentation. By combining different models trained on demographic, behavioural, and transactional data, organisations can obtain more precise insights into customer behaviour and personalise their strategies effectively.

In the broader field of data science and artificial intelligence, ensembles are fundamental to improving the robustness of predictive systems. They are frequently used in competitions, research, and production environments because they balance bias and variance effectively. Whether applied to classification, regression, or ranking tasks, ensemble techniques have become a cornerstone of modern machine learning practice.