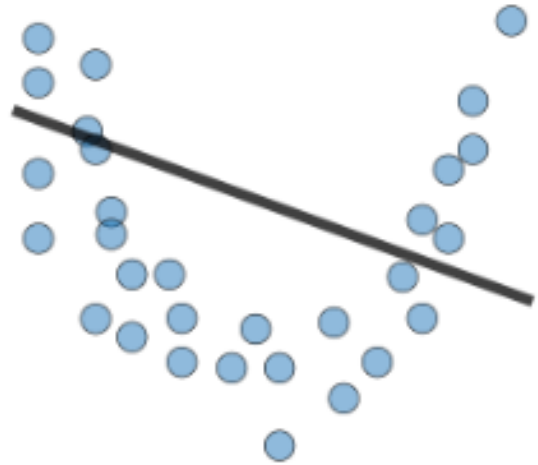
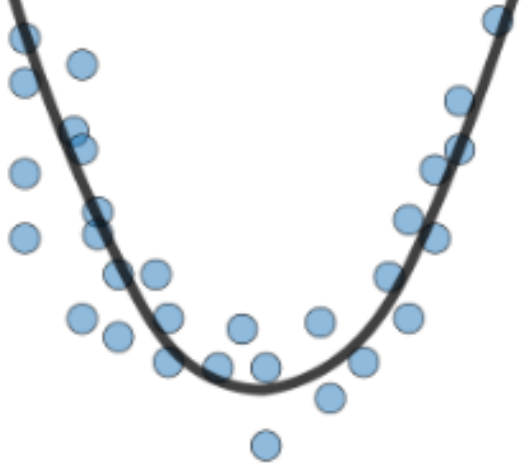
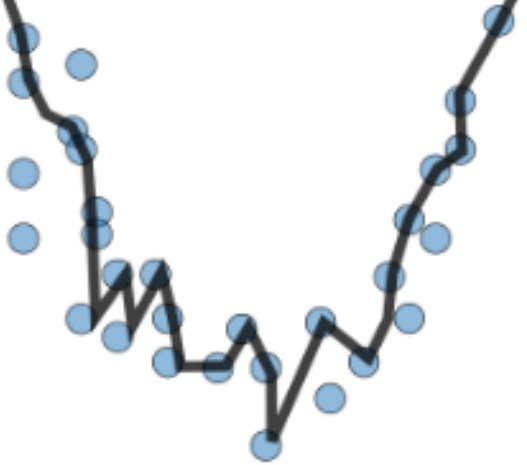
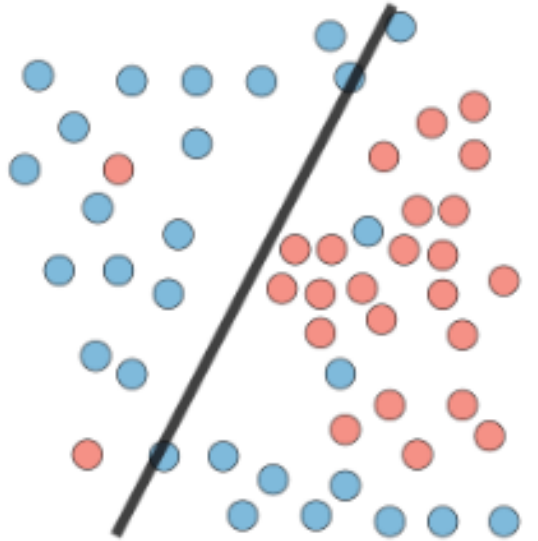
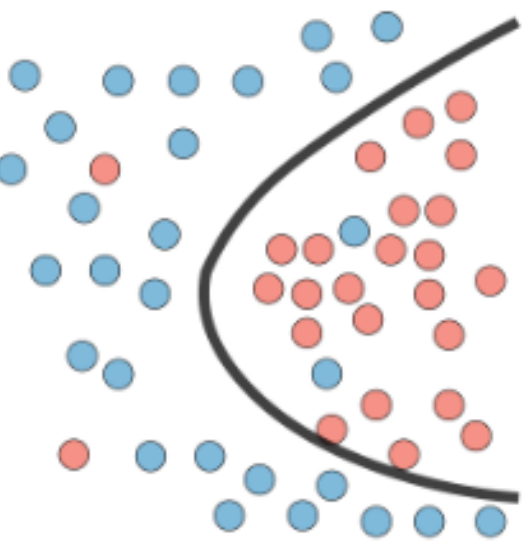
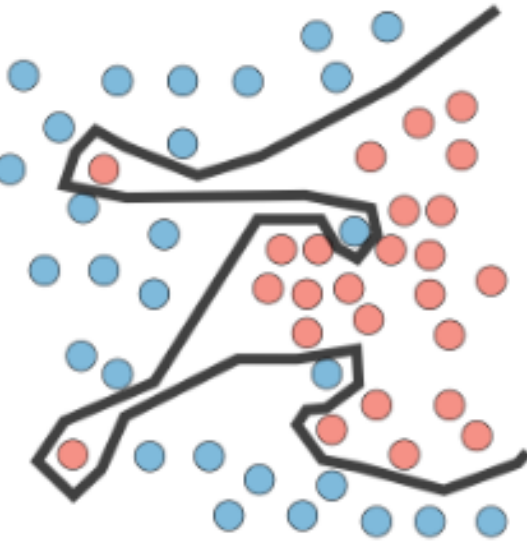


Problem with Decision Trees

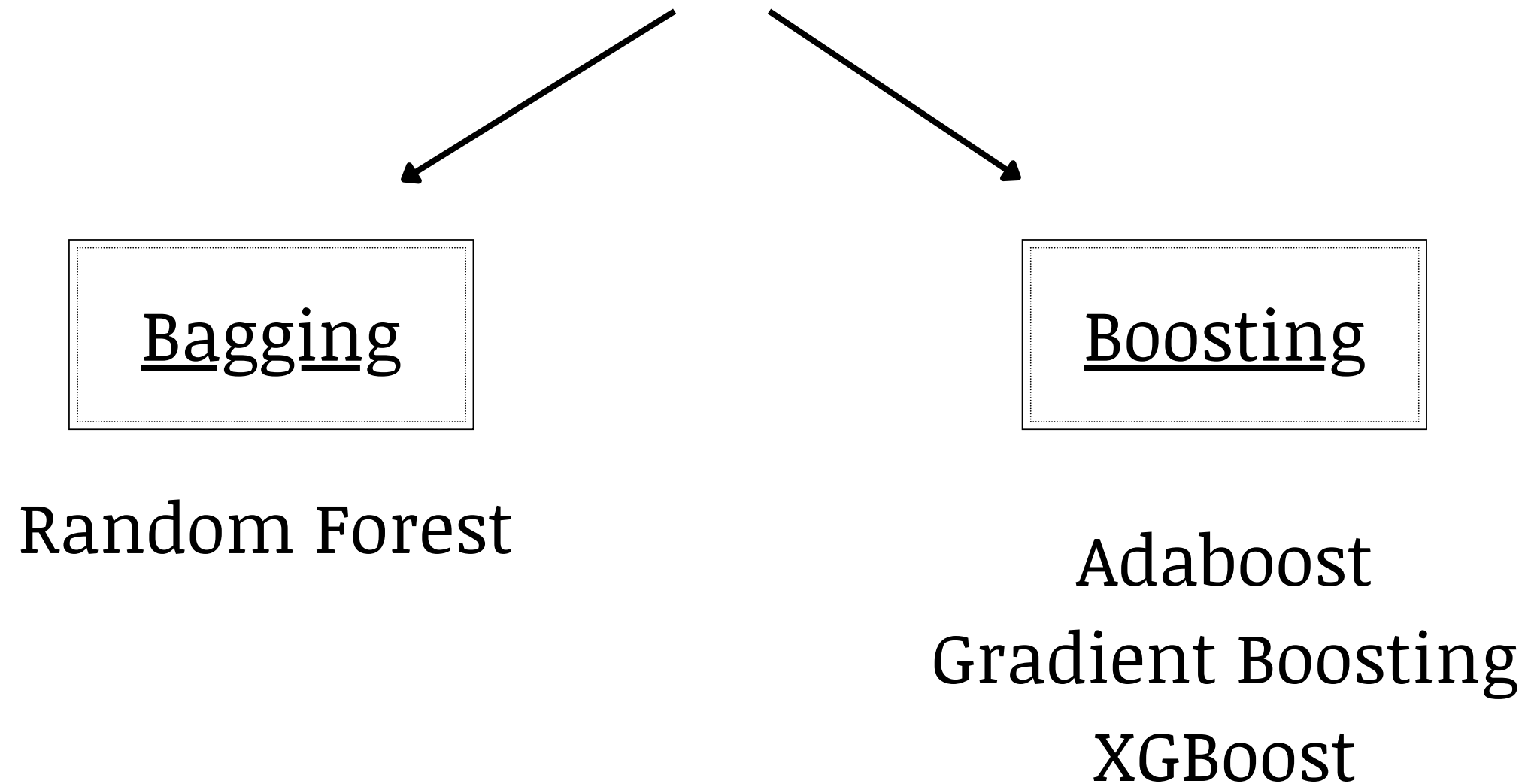
The most significant disadvantage of Decision Trees is that they are prone to overfitting. Decision Trees overfit because you can end up with a leaf node for every single target value in your training data.

In fact, that is the default parameter setting for Decision Tree Classifier / Regressor in sklearn. Though you can tune some hyperparameters like `max_features`, `min_samples_split`, `min_samples_leaf`, and `max_depth`, it is because of this disadvantage that simple Decision Trees are often traded for more complex algorithms : ensemble methods.

	Underfitting	Just right	Overfitting
Symptoms	<ul style="list-style-type: none">• High training error• Training error close to test error• High bias	<ul style="list-style-type: none">• Training error slightly lower than test error	<ul style="list-style-type: none">• Very low training error• Training error much lower than test error• High variance
Regression illustration			
Classification illustration			

Ensembling Methods

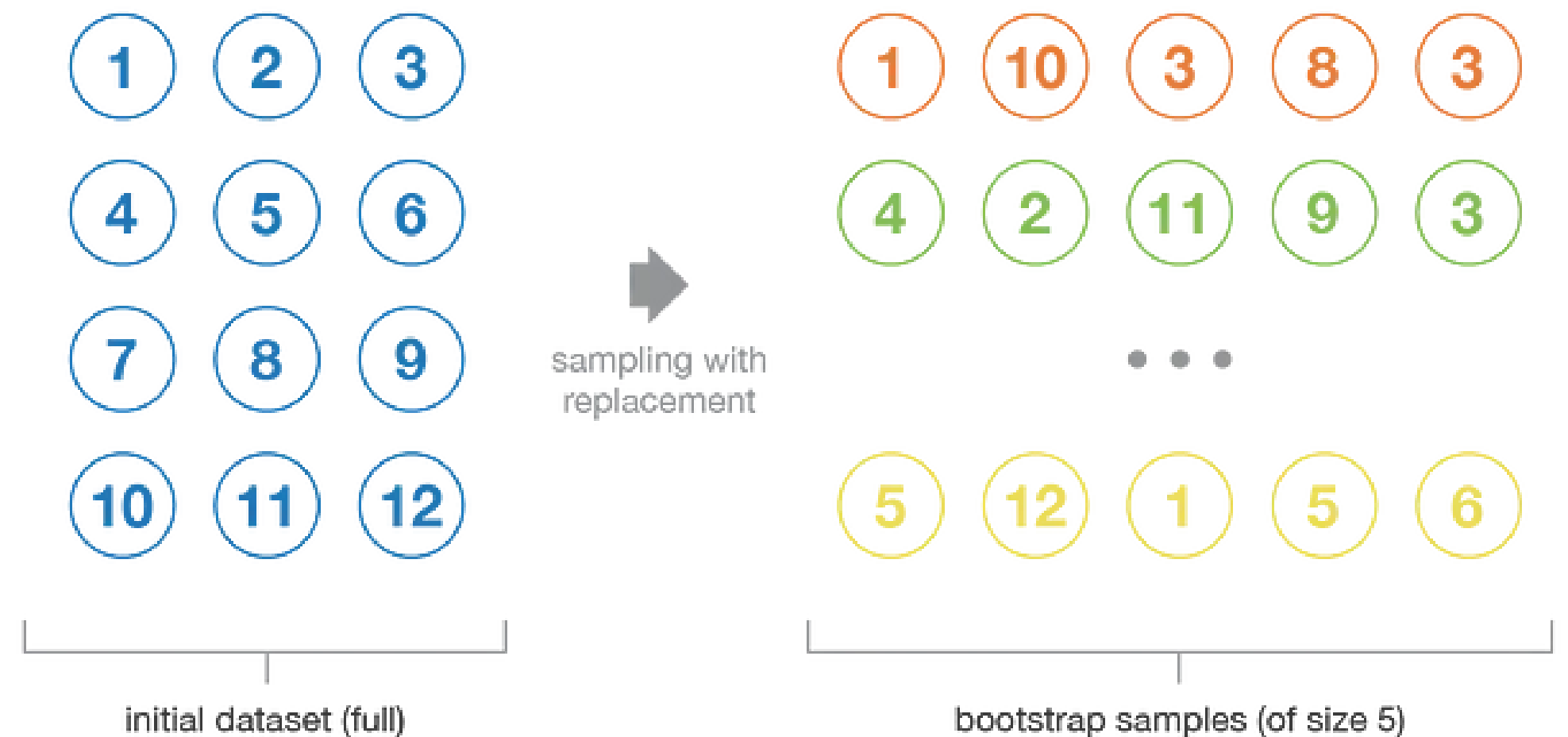
Ensemble learning is a machine learning technique where multiple models (often called “weak learners”) are trained to solve the same problem and combined to get better results. The main hypothesis is that when weak models are correctly combined we can obtain more accurate and/or robust models.



Bagging

Bagging stands for Bootstrap Aggregating. The idea is to improve the accuracy and stability of models by creating an ensemble of multiple models trained on different subsets of the training data..

Bagging works by generating multiple bootstrap samples from the original training dataset. A bootstrap sample is created by randomly selecting data points from the original dataset with replacement, which means that some samples may be selected multiple times while others may not be selected at all. This process introduces diversity in the training datasets.



Parallel Training : For each bootstrap sample, a separate model is trained using a chosen learning algorithm. These models are often referred to as base models or weak learners. By training each model on a different subset of the data, the models capture different aspects and patterns present in the dataset.

Aggregation : Finally, depending on the task (i.e. regression or classification), an average or a majority of the predictions are taken to compute a more accurate estimate. In the case of regression, an average is taken of all the outputs predicted by the individual classifiers; this is known as soft voting. For classification problems, the class with the highest majority of votes is accepted; this is known as hard voting or majority voting.

Random Forest

It is a bagging technique that uses a high number of decision trees built out of randomly selected sets of features. In addition to that, each tree is only allowed to consider a random subset of features during the tree construction process.

Contrary to the simple decision tree, it is highly uninterpretable but its generally good performance makes it a popular algorithm. The weak learners in case of Random Forest are either shallow or deep decision trees.

```
sklearn.ensemble.RandomForestClassifier(n_estimators=100, criterion='gini',  
max_depth=None, min_samples_split=2, max_features='sqrt', bootstrap=True )
```

`n_estimators` : Number of decision trees.

`criterion` : gini or entropy for class, for regression : squared error or absolute error.

`max_depth` : depth of each tree.

`max_features` : sqrt, log2 or 1 , if there are 10 features, take only $\sqrt{10} = 3$ features.

Boosting

Boosting is another popular ensemble learning technique which focuses on building a sequence of models, where each subsequent model tries to correct the mistakes made by the previous models. Boosting algorithms aim to create a strong learner by combining multiple weak learners.

Different Boosting Techniques :

Adaptive boosting	Gradient boosting
<ul style="list-style-type: none">• High weights are put on errors to improve at the next boosting step• Known as Adaboost	<ul style="list-style-type: none">• Weak learners are trained on residuals• Examples include XGBoost

ADABOOST

It works on the principle of learners growing sequentially. Except for the first, each subsequent learner is grown from previously grown learners.

A general overview of the Adaboost process for a classification task :

- 1. Initialize weights:** Each instance in the training dataset is initially assigned equal weights. *Suppose you have 10 samples in your train dataset. Assign weights of 0.1 to each sample.*
- 2. Train weak model:** A weak model, often referred to as a base or weak learner, is trained on the training data. The weak learner can be any learning algorithm, such as decision trees, logistic regression, or support vector machines. However, decision stumps (trees with only 1 split) are commonly used as weak learners in boosting algorithms. *Suppose you splitted the data using decision tree and gini impurity as the criteria.*
- 3. Evaluate model performance:** The trained weak model's performance is evaluated by measuring its accuracy or error rate on the training data. Instances that were misclassified by the model are given higher weights. *The total error is calculated as $E = \text{sum}(\text{weight of sample} * \text{error in the sample})$. Error is either 1 or 0 in case of classification. Now the classifier chosen in the previous step is assigned a weight, $\alpha = 0.5 * \ln((1-E)/E)$. If error is less, alpha will be high.*

4. Update instance weights: The weights of the training instances are updated based on their classification errors.

*The formula is $\text{new_weights} = \text{old_weights} * e^{(+/- \alpha)} / \text{sum of new_weights}$. In classification tasks, Plus is used when it was wrongly classified, thus increasing weights and so on. Dividing term is to Normalize the weights so that their sum=1.*

5. Create a New Dataset: *Arrange the normalized weights in ascending order and create buckets in the dataset. Now choose n random numbers between 0 and 1. Select the buckets where those numbers lie and make a new dataset from those buckets. There is a high probability for wrong records to get selected several times.*

5. Repeat steps 2-5 on the new dataset and so on for a predefined number of iterations or until a certain condition is met. In each iteration, the subsequent weak models are trained to focus on the instances that were misclassified or had higher weights in the previous iterations.

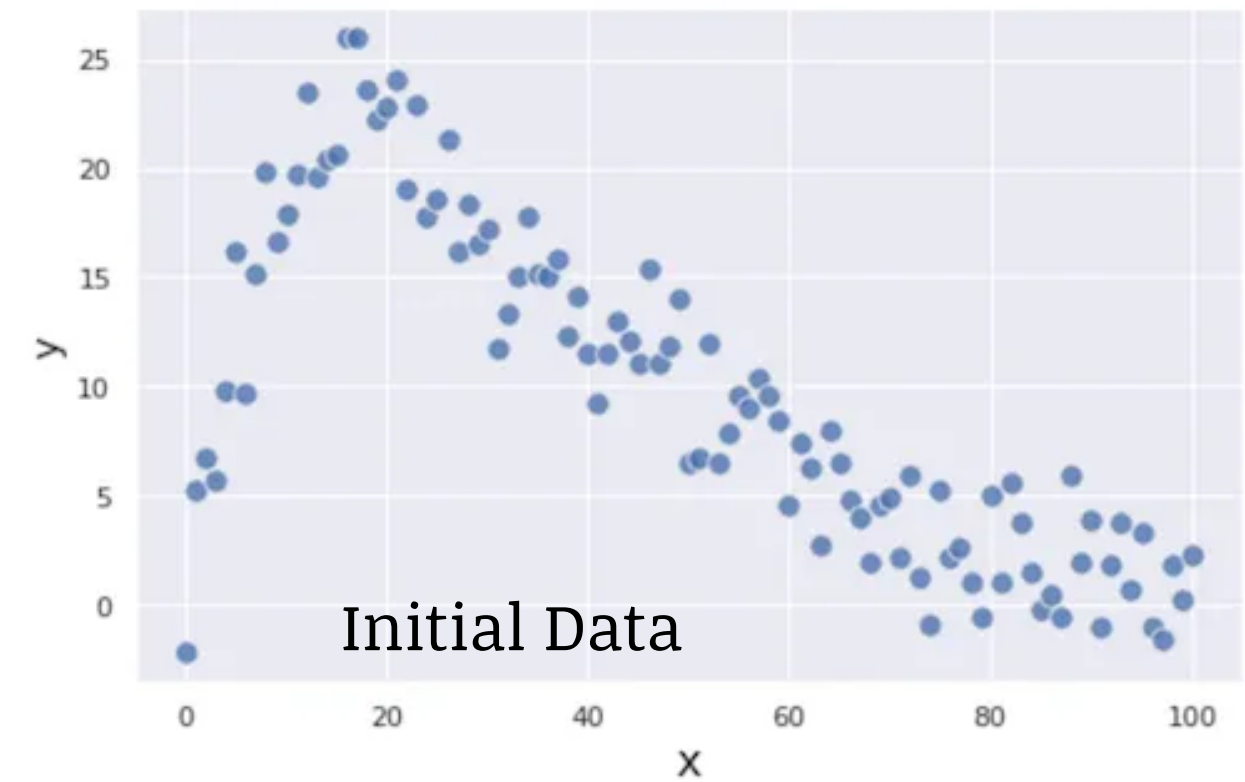
6. Final prediction: The final prediction is made by combining the predictions of all the weak models in the ensemble. The combination can be done through weighted voting, where models with higher weights have more influence on the final prediction.

Gradient Boosting

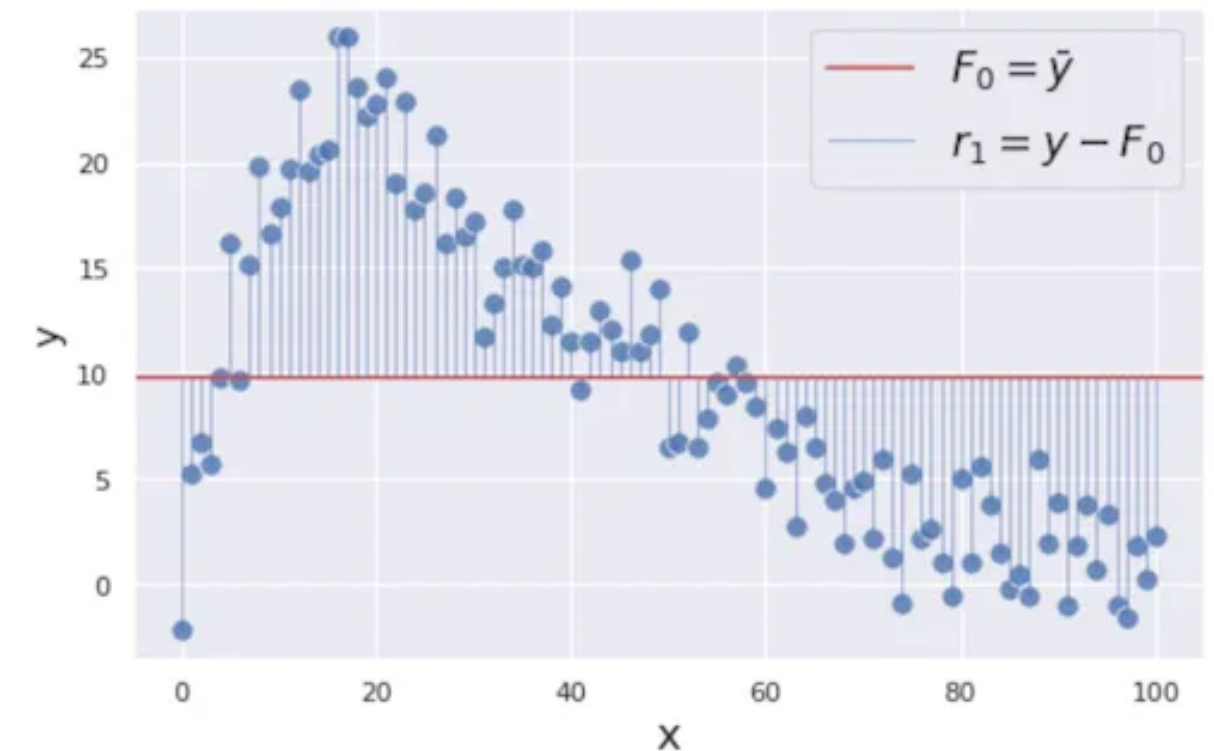
Gradient Boosting is a popular boosting algorithm that builds an ensemble of weak models, typically decision trees, to create a strong predictive model. It works by iteratively training models that minimize a loss function by leveraging gradient descent.

Here's an overview of the Gradient Boosting process:

- 1. Base Model:** The initial prediction for each instance is set to a constant value, typically the *mean of the target variable* in the case of regression or the class probabilities in the case of classification.
- 2. Calculate residual loss (MSE) and Compute the gradient:** The gradient of the loss function with respect to the current predictions is calculated for each training instance. The gradient represents the direction and magnitude of the steepest descent.
- 3. Train a weak model:** A weak model, often a decision tree, is trained to predict the negative gradient (the "residuals") of the loss function. The weak model is fit to the residuals, aiming to capture the patterns that the current ensemble has not yet learned.



Step 1 and 2

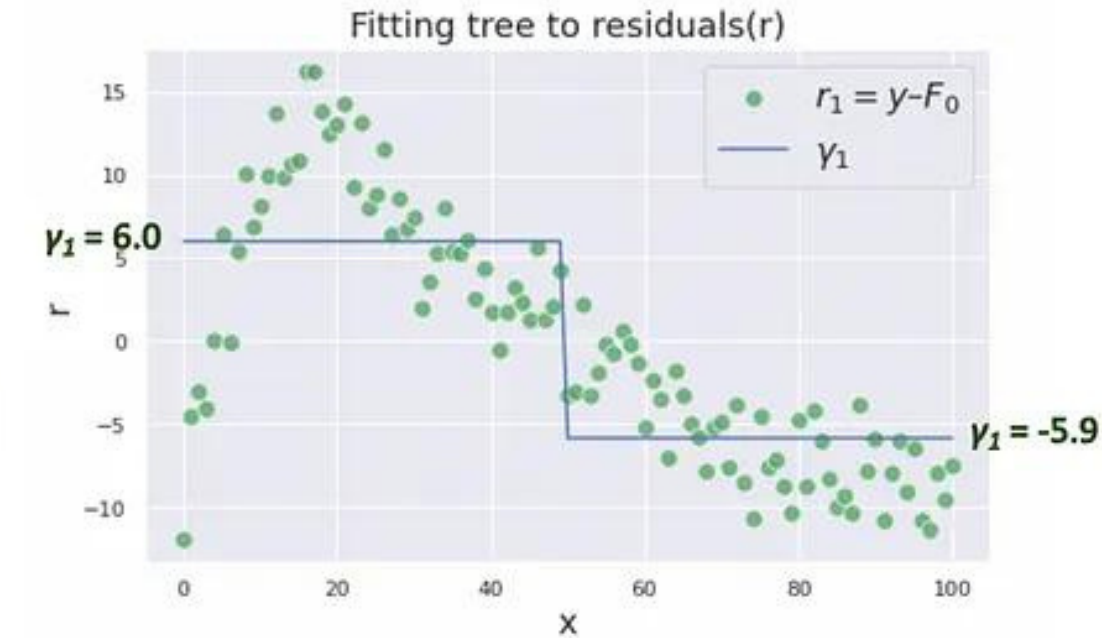
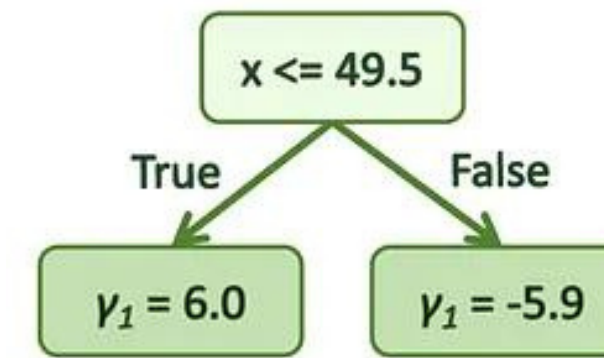


4. Update the predictions: The predictions of the ensemble are updated by adding the predictions of the weak model. after multiplying by a learning rate.

5. Repeat steps 2-4 iteratively for a predefined number of iterations or until a stopping criterion is met. In each iteration, the subsequent weak models are trained to capture the remaining patterns in the data that the ensemble has not yet learned.

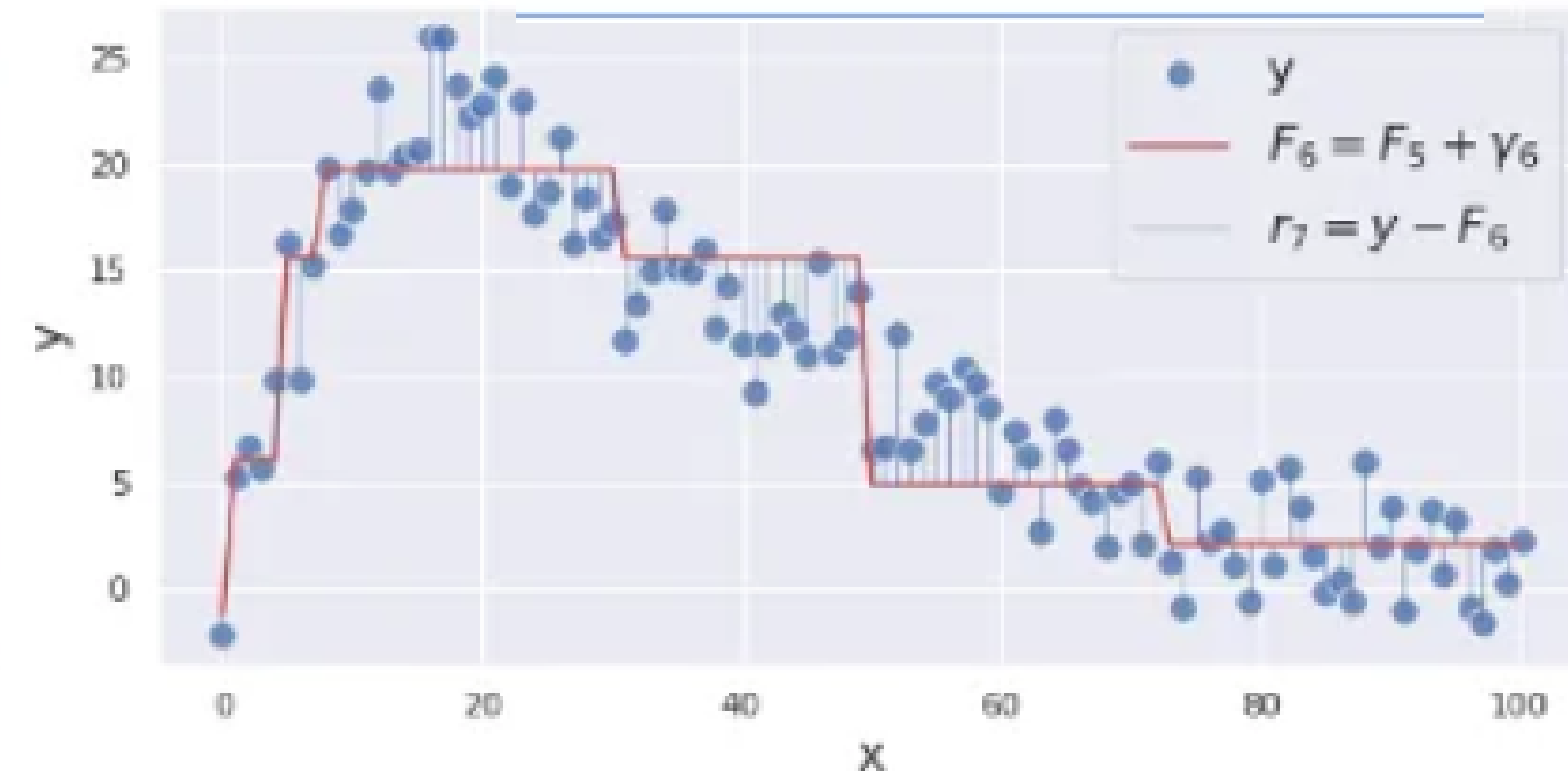
6. Final prediction: The final prediction is made by aggregating the predictions of all the weak models in the ensemble. For regression, the predictions are usually summed, while for classification, they can be combined using weighted voting or averaging.

Step 3



Step 4

$$F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} 1(x \in R_{jm})$$



Common implementations of Gradient Boosting include XGBoost, LightGBM, and CatBoost, which provide optimizations and additional features to enhance the algorithm's

Gradient Boosting Algorithm

1. Initialize model with a constant value:

$$F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^n L(y_i, \gamma)$$

2. for $m = 1$ to M :

2-1. Compute residuals $r_{im} = - \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F_{m-1}(x)} \text{ for } i = 1, \dots, n$

2-2. Train regression tree with features x against r and create terminal node regions R_{jm} for $j = 1, \dots, J_m$

2-3. Compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma) \text{ for } j = 1, \dots, J_m$

2-4. Update the model:

$$F_m(x) = F_{m-1}(x) + v \sum_{j=1}^{J_m} \gamma_{jm} 1(x \in R_{jm})$$

For our case L is the Mean Square Error.