Computational Finance – Lesson 10

06/12/2024

ginevra.angelini@polimi.it

Agenda

- In the previous lecture, we explored the general machine learning pipeline, focusing on clustering algorithms

- In today's lesson we are going to focus on the following supervised learning algorithms:

    i. Random Forest & XgBoost
    ii. Ridge & LASSO regression

- Then we will Implement hands-on examples to see these algorithms in action

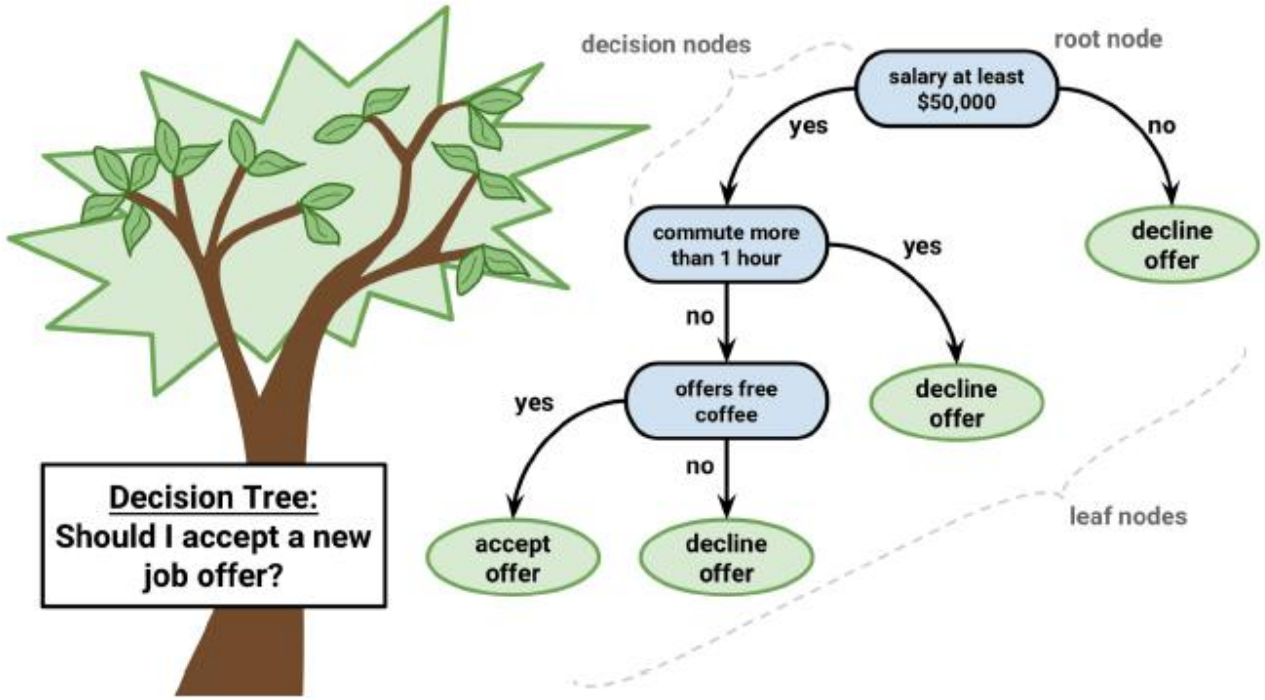
Random Forests & XgBoosting

Decision Trees and Random Forest

- Decision trees are a non-parametric **supervised learning method** for **classification or regression** models in the form of a **tree structure.**

- A decision tree lets you predict the value of a target variable by following the decisions in the tree from the root (beginning) down to a leaf node.

- A tree consists of branching conditions where the value of a predictor is compared to a trained weight. The number of branches and the values of weights are determined in the training process.



Classification Trees

- Either Classification and Regression Tree use the CART algorithm (Classification and Regression Trees)
- CART algorithm first splits the training set in two subsets using a single feature k and a threshold t_k

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

$$\text{where} \begin{cases} G_{\text{left/right}} \text{ measures the impurity of the left/right subset,} \\ m_{\text{left/right}} \text{ is the number of instances in the left/right subset.} \end{cases}$$

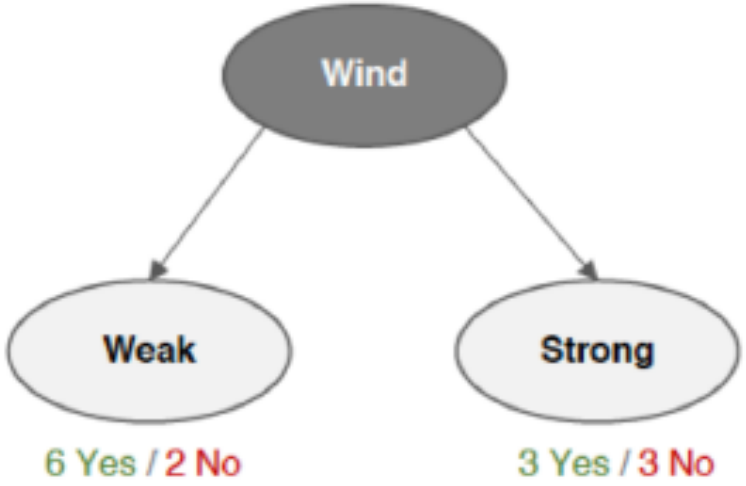
- Once it has successfully split the training set in two, it splits the subsets using the same logic, then the subsets and so on, recursively. It stops recursing once it reaches the maximum depth parameter, or if it cannot find a split that will reduce impurity.

- The CART algorithm is a **greedy algorithm**: it greedily searches for an optimum split at the top level, then repeats the process at each level. It does not check whether or not the split will lead to the lowest possible impurity several levels down.

Decision Trees

- A decision tree is built top-down from a root node. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with **decision nodes and leaf nodes.**

- A decision node has two or more branches. ***Leaf node represents a classification or decision.***



- Let' see how to compare the different ways to split data in a node

- Want to measure the "purity" of the split
  - Pure set (6yes/ 0 No) ⇒ completely certain(100%)
  - Impure (3 yes/ 3 No) ⇒ completely uncertain(50%)

- Splits on all attributes are tested
  - Constructing a decision tree is all about finding attribute that returns the highest information gain or lowest Gini Index (i.e., the most homogeneous branches)

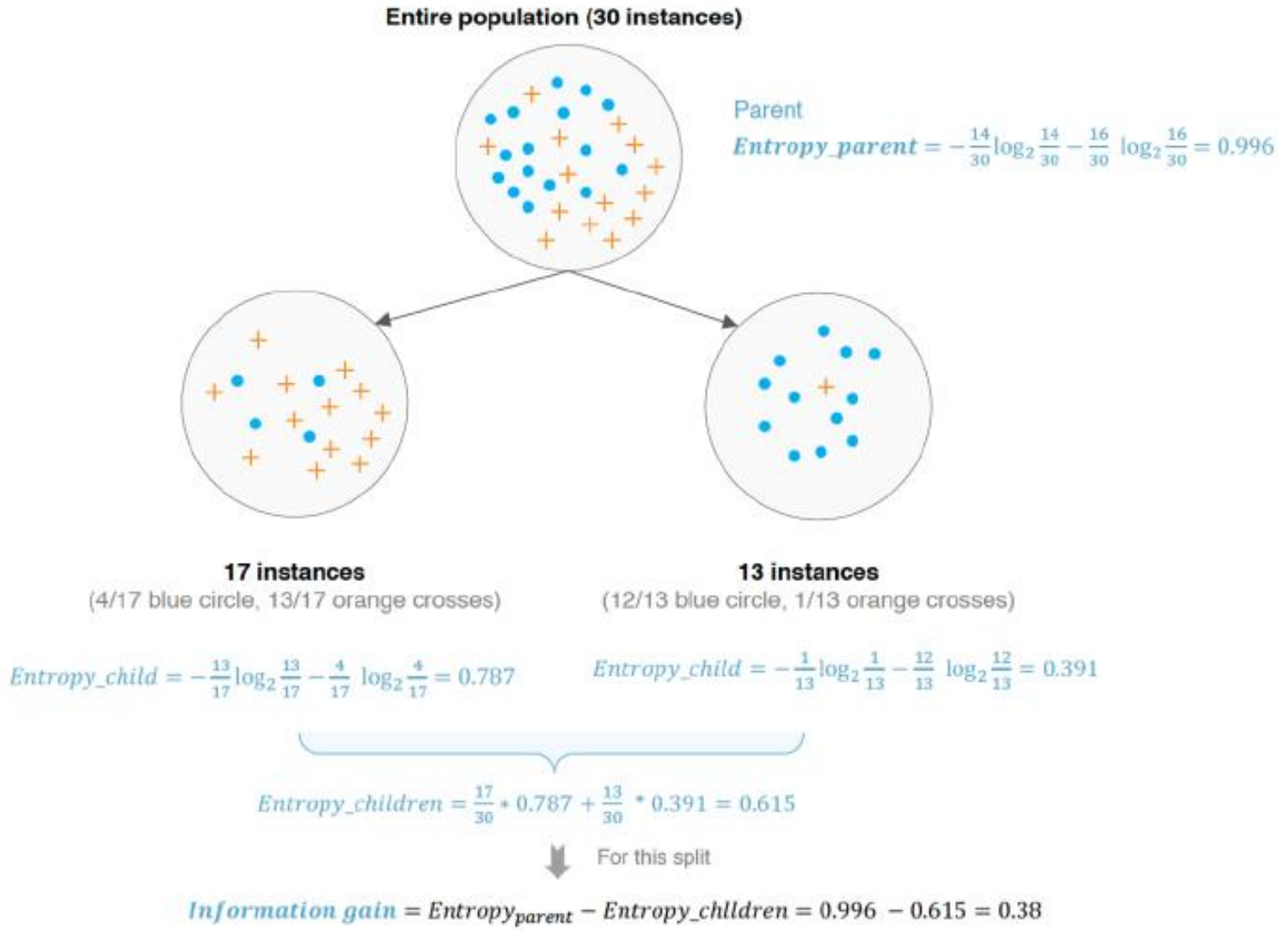
Measures of impurity

- ***Information Gain***, ***Gain Ratio*** and ***Gini Index*** are the most common methods of attribute selection
- *Information Gain*: it increases with the average purity of the subsets. Strategy: choose attribute that gives greatest information gain.
- A reduction of information entropy is often called an information gain.

$$IG = H_{t-1} - H_t$$

- Constructing a decision tree is all about finding attribute that returns the highest information gain (i.e., the most homogeneous branches)
- A decision tree is built top-down from a root node and involves partitioning the data into subsets that contain instances with similar values (homogenous)
- The information gain is based on the decrease in entropy after a dataset is split on an attribute.

Measures of impurity



Measures of impurity

- **Gini Index**: the Gini impurity is defined as: $G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2$, where $p_{i,k}$ is the ratio of class k instances among the training instances in the i^{th} node

- A node's Gini attribute measures its impurity: a node is "pure" (gini=0) if all training instances it applies to belong to the same class. In other words, Gini Index would be zero if perfectly classified.

- To avoid **overfitting** the training data, you need to restrict the Decision Tree's freedom during training
- The hyperparameters of CART used for regularization are the following:

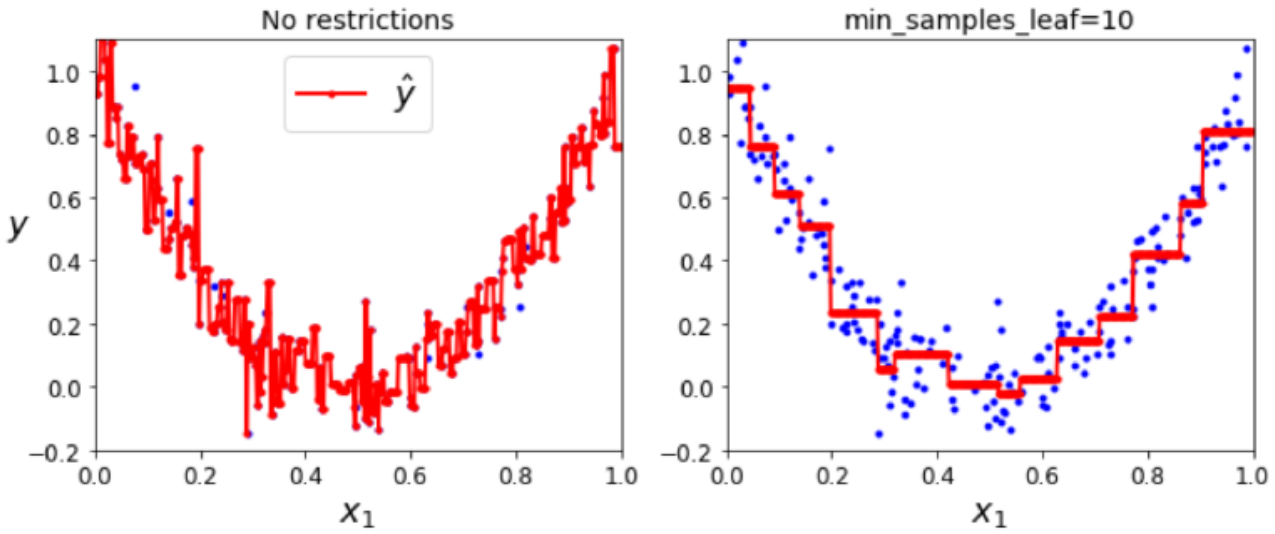
  - max_depth: maximum depth of the Decision Tree
  - min_samples_split: minimum number of samples a node must have before it can be split
  - min_samples_leaf: minimum number of samples a leaf node must have
  - min_weight_fraction_leaf: fraction of the total number of weighted instance
  - max_leaf_nodes: maximum number of leaf nodes
  - max_features: maximum number of features that are evaluated for splitting at each node

- Increasing min_* hyperparameters or reducing max_* hyperparameters will regularize the model.

Regression Trees

- The CART algorithm works mostly the same way as earlier, except that instead of trying to split the training set in a way that minimizes impurity, it now tries to split the training set in a way that **minimizes the MSE**:

$$J(k, t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}} \quad \text{where} \begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} \left(\hat{y}_{\text{node}} - y^{(i)}\right)^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$



Pros & Cons

- Decision Trees are very sensitive to small variations in the training data and may yield very different models even on the same training data.
- Solution:
    1. Perform dimensionality reduction (e.g. PCA) to alleviate variations
    2. Use many trees and make the prediction by averaging them (Random Forest)

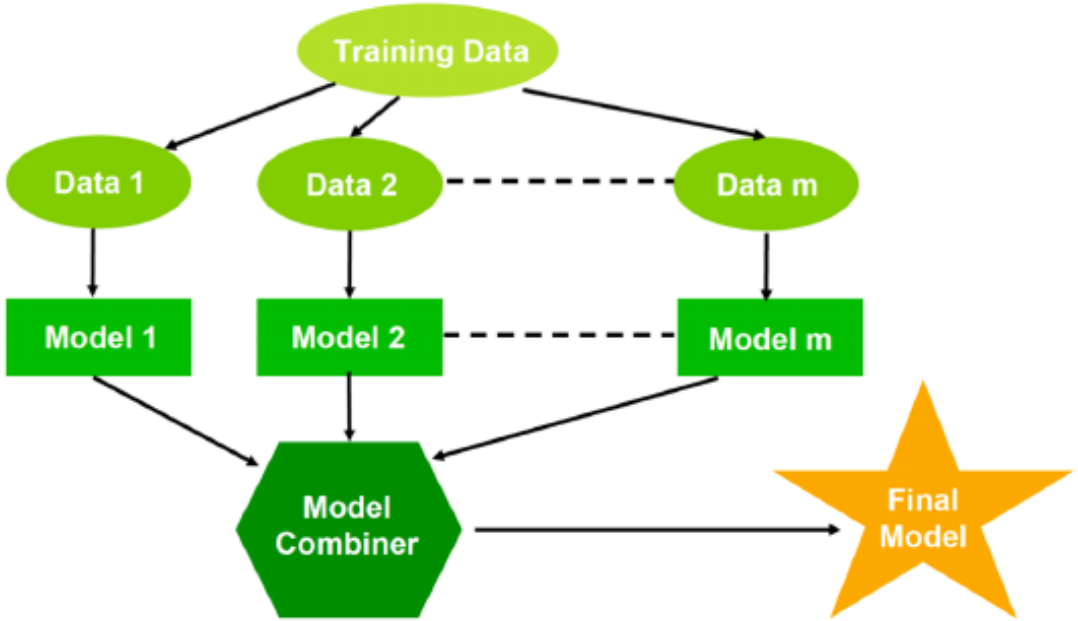
- **Advantages:**
    - Simple to understand and to interpret
    - To build decision tree requires little data preparation (no need feature scaling, not sensitive to outliers)
    - Decision trees are able to handle both continuous and categorical variables
    - Implicitly perform feature selection

- **Disadvantages:**
    - They are prone to over-fitting
    - Decision Tree learner can create biased trees in case of unbalanced data
    - Instability
    - Greedy approach used by Decision tree doesn’t guarantee best solution

Ensemble Learning

- **Ensemble learning** is a machine learning paradigm where **multiple learners** are trained and combined to solve the same problem. By using multiple learners, the generalization ability of an ensemble can be much better than that of a single learner.

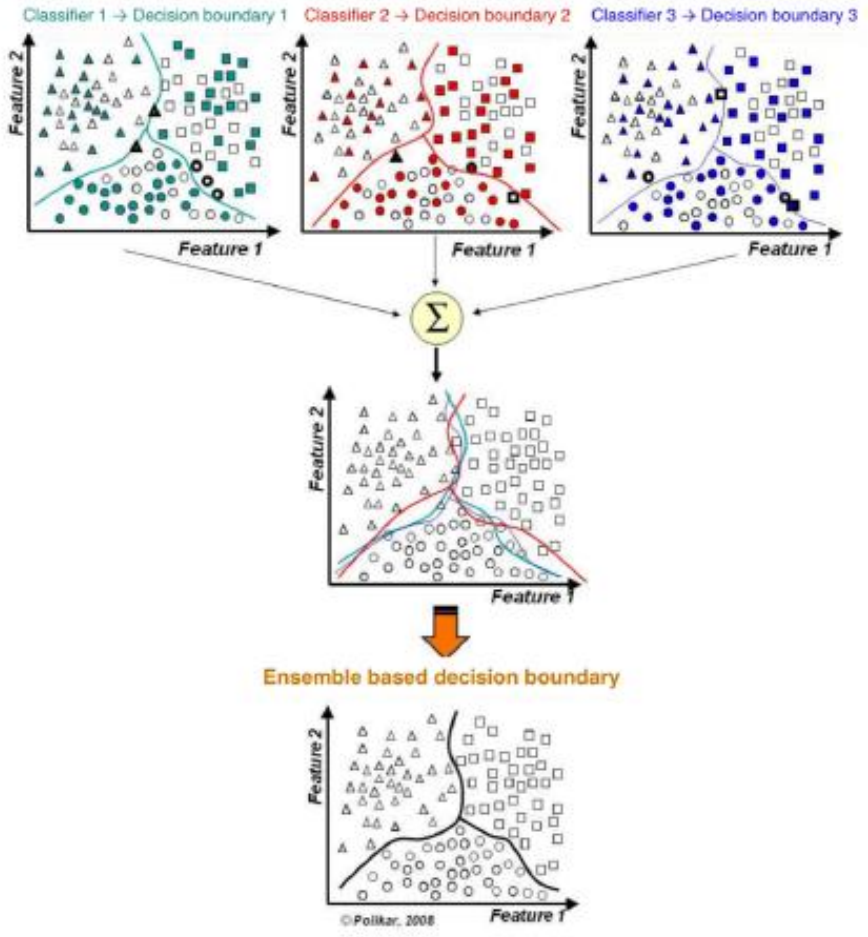
- Suppose you ask a complex question to thousands of random people, then aggregate their answers. In many cases you will find that this aggregated answer is better than an expert's answer. This is called the ***wisdom of the crowd***.



- Similarly, if you aggregate the predictions of a group of predictors (such as classifiers or regressors), you will often get *better predictions* than with the best individual predictor. A group of predictors is called an ensemble; thus, this technique is called Ensemble Learning

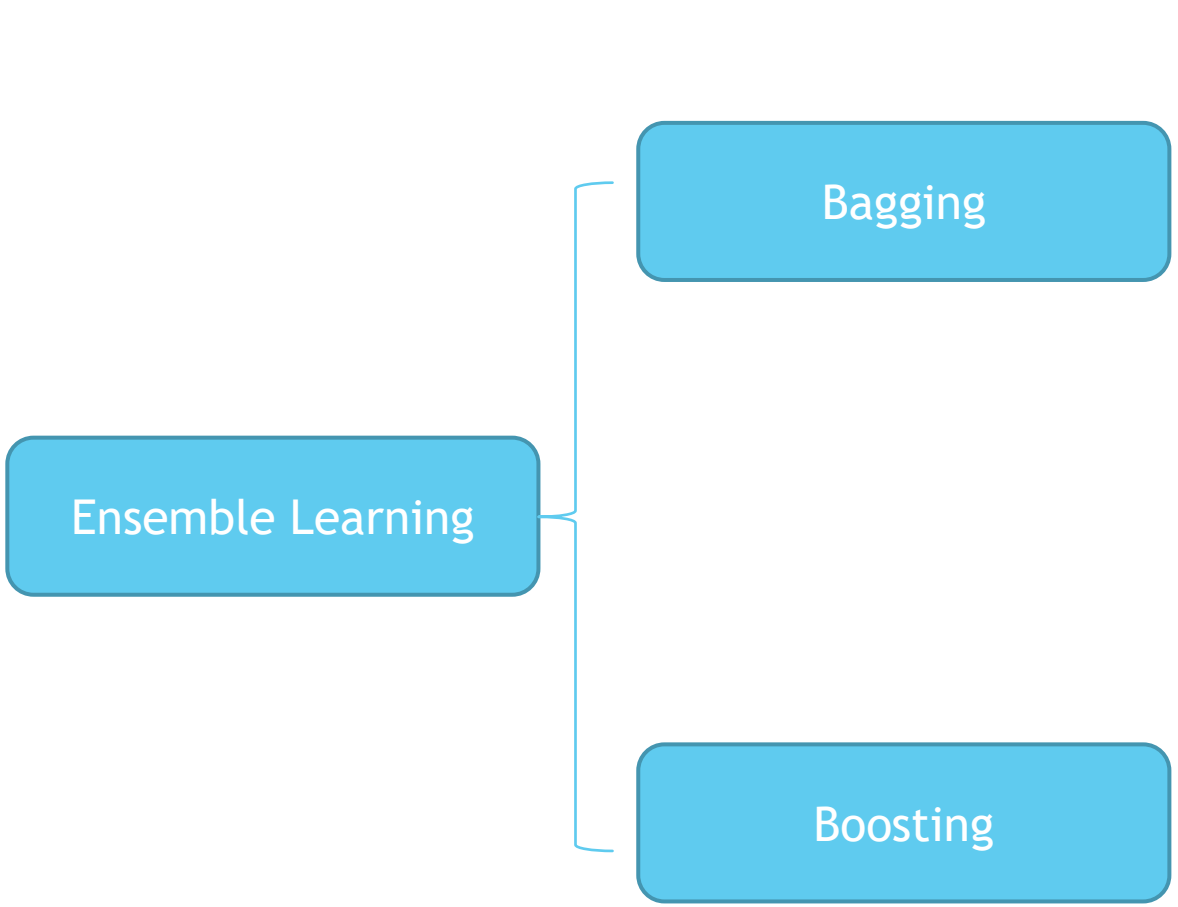
Example Ensemble Learning

- Combining an ensemble of classifiers could reduce classification error and reduce the overall risk of making selection of a particularly poorly performing classifier



- The figure graphically illustrates this concept, where each classifier - trained on a different subset of the available training data - makes different errors (shown as instances with dark borders), but the combination of the (three) classifiers provides the best decision boundary

Types of Ensemble Learning



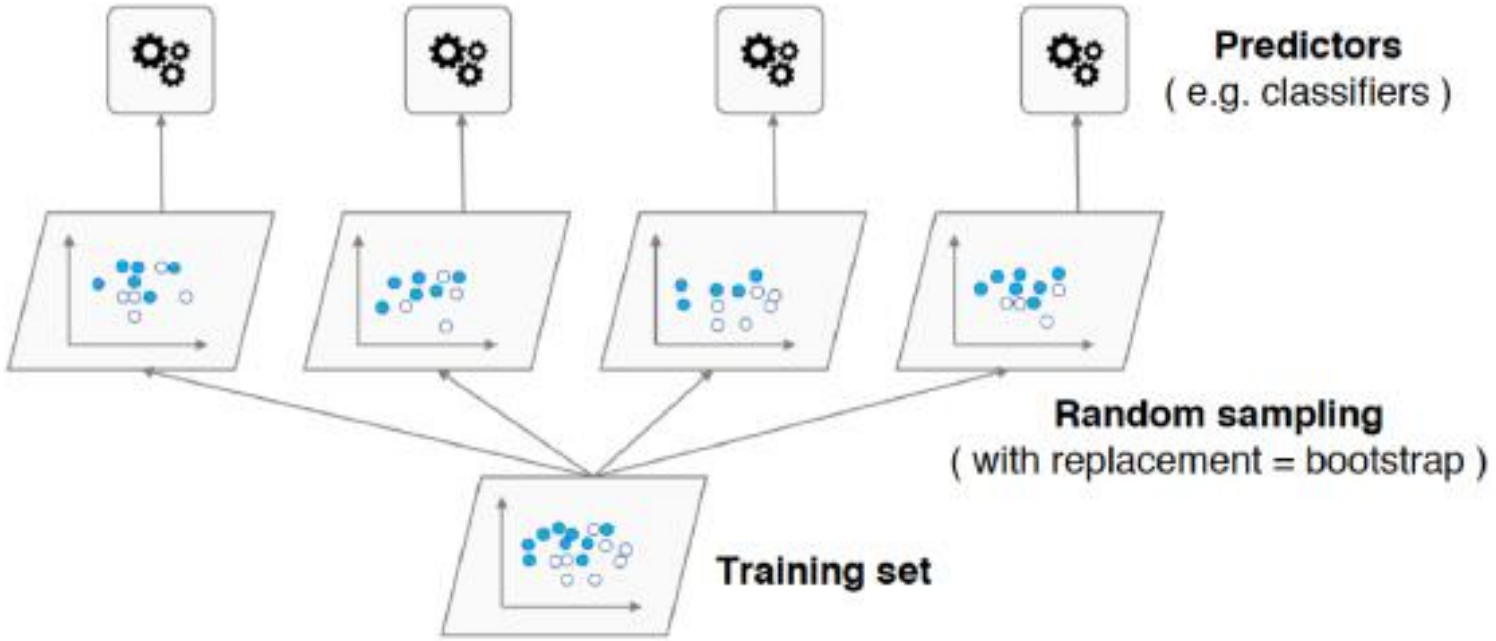
Example: Random Forest
The driving principle is to build **several estimators/learners independently** and then to **average their predictions.** On average, the combined learners is usually better than any of the single base learner because its variance is reduced

Example: Gradient Boosting, AdaBoost, xgBoost
Base **learners are built sequentially** and one tries to reduce the bias of the combined learner. Each, and focus on previous one' **learner is dependent on the previous one**s error. The motivation is to combine several weak models to produce a powerful ensemble.

Bagging

- **Bagging** (short for bootstrap aggregating) is a technique for reducing generalization error by combining several models. The idea is to train several different models separately, then have all of the models vote on the output for test examples (average of the predictions).

- Predictors are trained on different random subsets of the training set. When sampling is performed with replacement, this method is called **bagging.**



How Bagging Works

i. Create random subsets of data by bootstrap sampling different training data subsets are randomly drawn – with replacement – from the entire training dataset. By sampling with replacement some observations may be repeated in each new training data set.

ii. Train base learners on each bootstrap sample separately

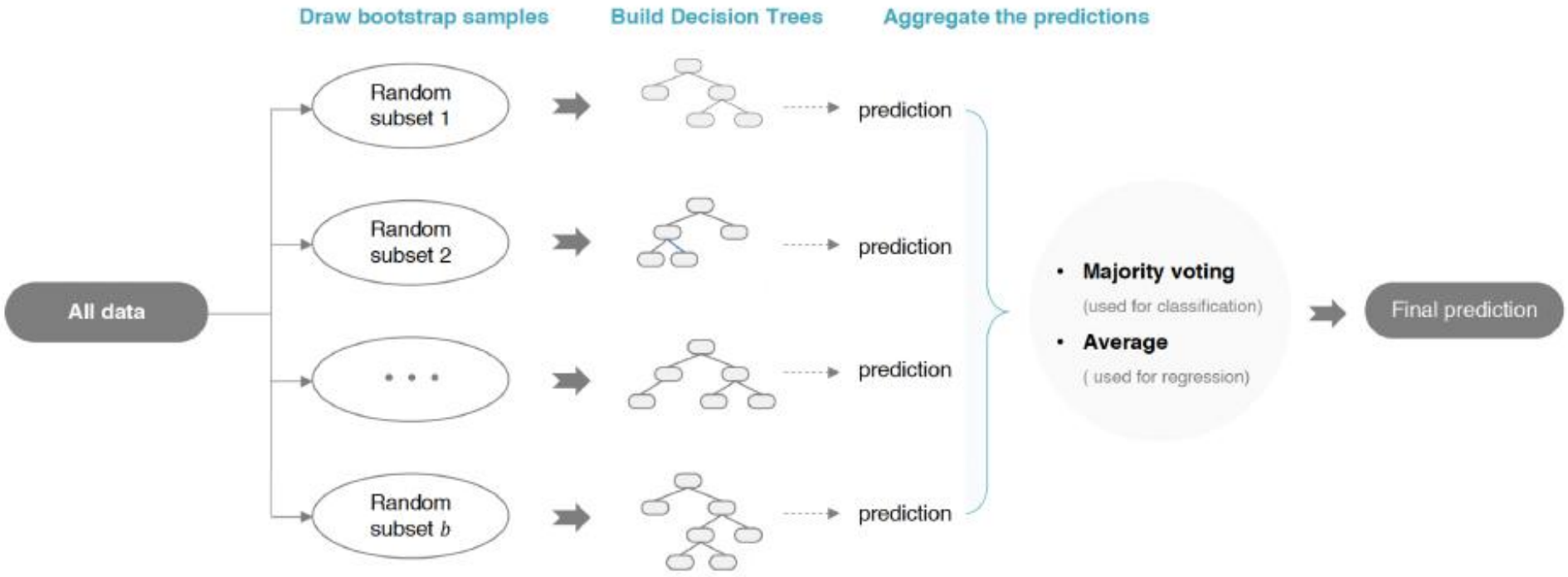
iii. Average predictions from multiple base learners

iv. The final result is to aggregate the outputs of base learners by

- Majority voting for *classification*
- Averaging for *regression*

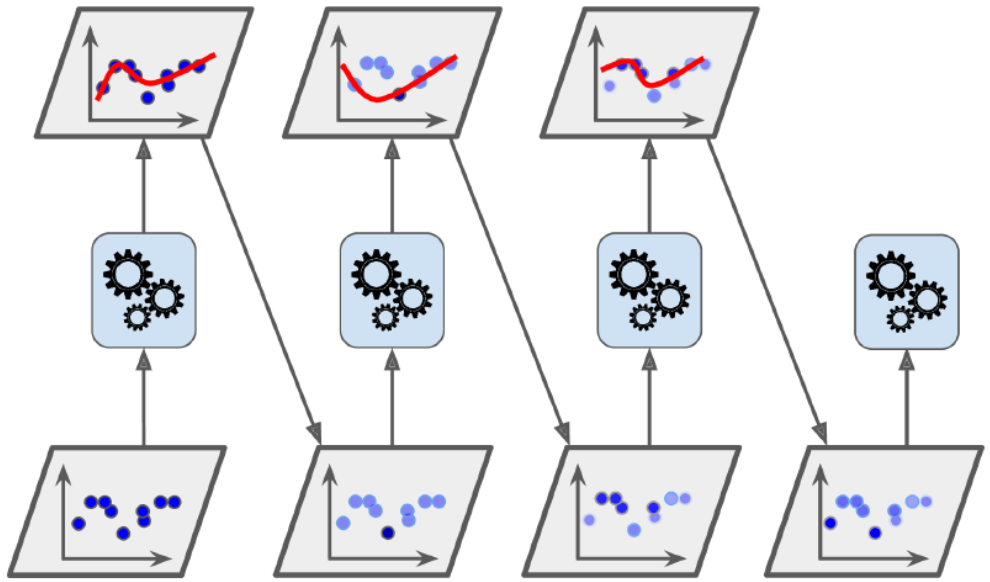
Random Forest

- **Random Forest** = Bagging + Full-grown CART decision Tree
- Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction. It can be used for both classification and regression problems.
- The Random Forest algorithm randomly selects observations and features to build several decision trees and then averages the results.



How Boosting works

- **Boosting** works by sequentially training models, where each model focuses on correcting the errors of the previous ones. At each step, the algorithm assigns more weight to data points that were mis-predicted in earlier iterations, forcing the new model to learn from these mistakes



- **Step-by-step implementation**

1. **Initialize the Model:** Start with a base model $f_1(x)$ that makes an initial prediction
2. **Compute Residuals or Weights:** Evaluate the errors (residuals) for the predictions
3. **Train a New Model on Errors:** Train the next weak learner $f_2(x)$ to focus on minimizing these errors or misclassified points. This model complements the previous one by improving where the first model failed
4. **Update the Combined Model:** Combine the outputs of the current and previous models. Add the new model's contribution with a weight α: $F(x)=F(x)+\alpha f(x)$
   α determines how much the new model's prediction should influence the overall result.
5. **Repeat the Process:** Iterate through steps 2–4 for a predefined number of rounds or until performance stops improving

XgBoost

- XGBoost, short for *Extreme Gradient Boosting*, is a machine learning algorithm designed for efficiency, scalability, and accuracy

- **Key Concepts:**
  - **Gradient Boosting**
    - Core Idea: Combine weak learners (usually decision trees) iteratively to form a strong learner.
    - How it works:
      i. A model predicts the target value.
      ii. The error (residual) is calculated.
      iii. A new model is trained to predict the residuals, and this process is repeated.
      iv. Predictions are updated by adding the contributions of the new model at each step.

  - **Why "*Extreme*"?** XGBoost extends gradient boosting with optimizations:
    - Regularization: Reduces overfitting by penalizing complex models.
    - Parallel Processing: Leverages multi-core systems for faster computations.
    - Sparse Data Handling: Efficiently handles missing or sparse data.
    - Tree Pruning: Ensures optimal tree size by pruning after training

XGBoost

- Think of XGBoost as a system of pipelines:
  1. **Input data** is passed through a sequence of decision trees
  2. Each tree adjusts its splits to "learn" from the errors of the previous ones
  3. The final output combines all tree predictions, weighted by their accuracy

- **How XGBoost Trains Decision Trees**
  1. **Loss Function**
     1. Defines the error between predicted and actual values (e.g., Mean Squared Error for regression)
     2. XGBoost uses gradients (slopes) of the loss function to decide how to adjust predictions
  2. **Additive Training**
     1. Each new tree is trained to minimize the residuals (errors) from the previous iteration
  3. **Split Finding**
     1. XGBoost uses a scoring function to decide the best way to split a node, focusing on maximizing information gain
  4. **Regularization**
     1. Penalizes overly complex trees to avoid overfitting
     2. Controls tree depth, leaf weights, and other parameters

Random Forest vs XGBoost

	Random Forest	XGBoost
Focus	Reducing variance by aggregating predictions from multiple independent trees.	Optimizing performance through gradient boosting and regularization.
Base Algorithm	Bagging (parallel tree building).	Gradient Boosting (sequential tree building
Handling Overfitting	Random feature selection and ensemble averaging.	Regularization techniques (L1, L2).
Strengths	- Robust to noise and overfitting due to averaging	- Can handle missing data.
	- Simpler to understand and implement.	- Highly customizable with hyperparameters (e.g., learning rate, tree depth).
	- Performs well on many datasets with little preprocessing.	- Supports advanced functionalities like monotonic constraints
Weaknesses	- Can struggle with extrapolation for small datasets.	- Sensitive to hyperparameter settings (learning rate, number of estimators).
	- Requires many trees to achieve high accuracy, leading to slower inference	- Computationally expensive for very large datasets
	- May not perform as well as XGBoost on highly imbalanced datasets.	- Can overfit on noisy data if not regularized properly.
Interpretability	Easier to interpret (e.g., variable importance via mean decrease impurity)	Harder to interpret due to sequential tree structure and additive nature.
Handling Imbalanced Data	May require resampling or weighting to handle class imbalance effectively	Performs well with proper objective functions (e.g., log loss for classification).

Ridge & Lasso regression

Linear Regression Models

- **Linear models** are foundational in statistics and machine learning. They model the relationship between a response variable **y** and one or more predictors ($x_1, x_2, \ldots, x_p$) using a linear equation.

- **Key Assumptions of Linear Models:** Traditional linear models are based on several assumptions to ensure validity and interpretability:
  - **Linearity**
    i. The relationship between predictors and the response is linear.
    ii. The response **y** is expressed as $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p + \epsilon$ where ϵ is the error term.
  - **Independence**
    i. Observations are independent of each other.
  - **Homoscedasticity**
    i. The variance of the error term (ϵ\epsilonϵ) is constant across all levels of predictors.
  - **Normality of Errors**
    i. The error term follows a normal distribution.
  - **No Multicollinearity**
    i. Predictors are not highly correlated with each other.
  - **Full Rank**
    i. There is no perfect collinearity among predictors, ensuring unique coefficient estimates.

Linear Regression Models

- **Simple Linear Regression Models:** the relationship between one predictor x and a response

$$y = \beta_0 + \beta_1 x + \epsilon$$

- **Multiple Linear Regression:** Extends simple regression to multiple predictors:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p + \epsilon$$

- **Generalized Linear Models (GLMs):** Extends linear models to accommodate non-normal response variables. Key components:
  - Linear predictor: $\eta = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p$
  - Link function: Links the linear predictor to the expected value of the response variable ($E(y)$)

- Linear models use the **Ordinary Least Squares (OLS)** method to estimate coefficients

$$\hat{\beta} = \arg\min_{\beta} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

This minimizes the sum of squared residuals (differences between observed and predicted values)

Ridge & LASSO Regression Models

- **Ridge and Lasso regression** were introduced to address two critical problems faced in traditional **linear regression** models:

  i. **Overfitting**:
     - When the number of predictors (features) is large relative to the number of observations (high-dimensional data), the model can fit the training data too well, capturing noise instead of the underlying trend. This results in poor performance on new data

  ii. **Multicollinearity**:
     - When predictors are highly correlated, the least squares estimates of the coefficients become unstable, leading to high variance and making the model sensitive to small changes in the data

- Both methods add a **regularization term** to the linear regression loss function, which helps reduce overfitting and improve the interpretability of the model

Ridge Regression

- **Ridge Regression:**
  - **How it Works:** Ridge regression adds an L2-norm penalty (squared magnitude of coefficients) to the loss function of ordinary least squares regression
  - The **Ridge loss function** is:

$$L(\beta) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

  - $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$: Ordinary least squares error (residual sum of squares)
  - $\sum_{j=1}^{p} \beta_j^2$: L2-norm penalty on the coefficients
  - λ: regularization parameter that controls the strength of the penalty

- **Key Characteristics**
  - Shrinks coefficients toward zero but does not force them to be exactly zero
  - Retains all predictors in the model, but with smaller coefficients

LASSO Regression

- **Lasso regression** adds an L1-norm penalty (absolute magnitude of coefficients) to the loss function
  - The **Lasso loss function** is:

$$L(\beta) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

  - $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$: Ordinary least squares error (residual sum of squares)
  - $\sum_{j=1}^{p} |\beta_j|$: L1-norm penalty on the coefficients
  - λ: regularization parameter that controls the strength of the penalty

- **Key Characteristics**
  - Shrinks some coefficients to exactly zero, effectively performing feature selection

Ridge vs LASSO Regression

	Ridge Regression	LASSO Regression
Penalty Type	L2-norm	L1-norm
Feature Selection	Keeps all features but shrinks coefficients	Shrinks some coefficients to zero, removing features
Effect on Coefficients	Reduces magnitude uniformly	Drives some coefficients to zero, creating sparsity
Use Case	When all predictors are potentially important	When some predictors are irrelevant or redundant

	Ridge Regression	LASSO Regression
Advantages	- Prevents overfitting by shrinking coefficients.	- Performs both regularization and feature selection
	- Works well when all predictors contribute to the target.	- Produces interpretable models with fewer predictors.
Drawbacks	- Does not eliminate irrelevant features.	- May struggle when predictors are highly correlated (one variable is chosen arbitrarily).
	- Less interpretable than Lasso for sparse datasets.	- Not suitable for scenarios where all predictors are important.