# Class 9 Supervised Machine Learning and Tree-Based Models

Dr. Wei Miao

UCL School of Management

October 28, 2026

Section 1

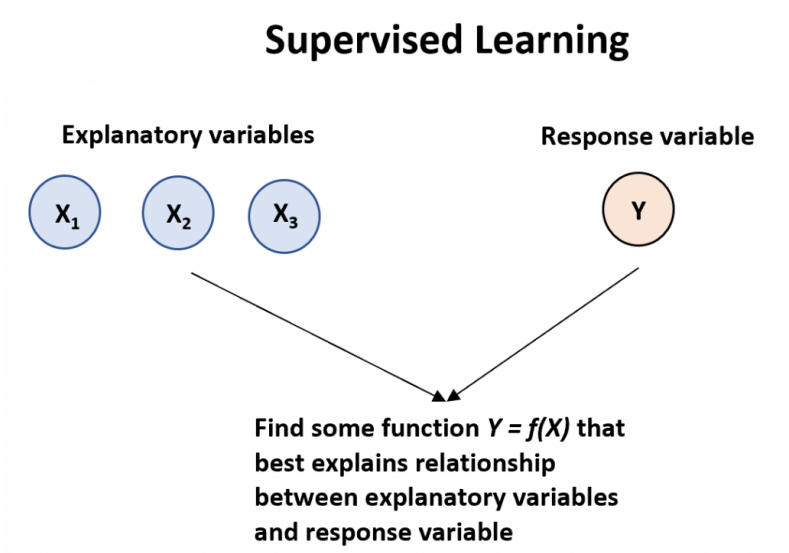
# Supervised Learning

# Learning Objectives

- Understand the fundamentals of supervised learning and its key components
- Distinguish between classification and regression tasks
- Recognize the accuracy-interpretability and bias-variance tradeoffs in machine learning
- Learn how to implement and interpret decision trees
- Understand random forests and their advantages over single decision trees
- Apply cross-validation techniques to mitigate overfitting

# Supervised Learning

- A **supervised learning model** is used when we have one or more **explanatory variables** AND **a response variable** and we would like to learn the **underlying true relationship** between the **explanatory** variables and the **response** variable as accurately as possible.



# Data Generating Process (DGP)

We use the following notations for supervised learning tasks:

$$Y = f(X; \theta) + \epsilon$$

- $Y$ is the **response/outcome/target** variable to be predicted
- $X = (X_1, X_2, ..., X_p)$ are a set of **explanatory variables/features/predictors**
- $f(X; \theta) + \epsilon$ is the true relationship between $X$ and $Y$, or DGP, which is never known to us[1]; $\epsilon$ is the **randomness term** or **error term**
- $\theta$ represents the set of **parameters** to be learnt from the data

[1] "All models are wrong, but some are useful" – George Box. As business analysts, we need to use the "wrong models" correctly.

# Types of Supervised Learning Algorithms

Depending on the type of the **response variable**, supervised learning tasks can be divided into two groups:

- **Classification tasks** if the outcome is **categorical**
  - Whether a customer responds to marketing offers (e.g., 1 for response, 0 for no response)
  - Whether a customer churns (e.g., 1 for churn, 0 for no churn)
  - Which product a customer purchases (e.g., 1 for product A, 2 for product B, etc.)
- **Regression tasks** if the outcome is **continuous**
  - Customer total spending in each period (e.g., $100, $200, etc.)
  - Demand forecasting such as the daily sales of a product (e.g., 100 units, 120 units, etc.)

# Difference between Supervised and Unsupervised Learning

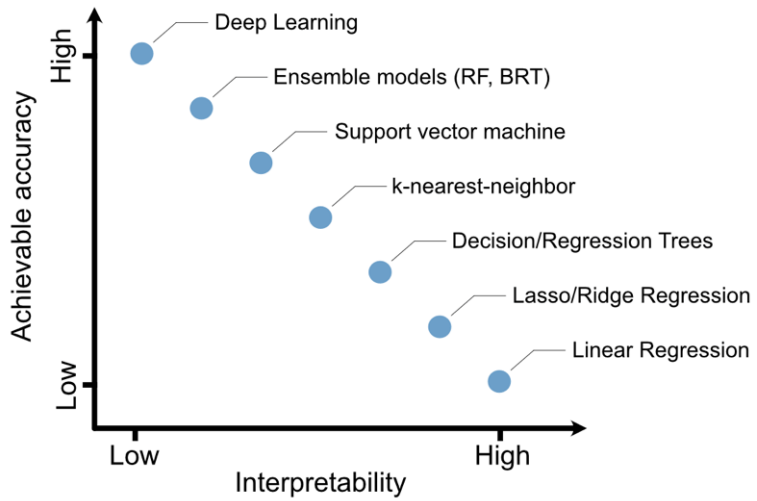
| | **Supervised Learning** | **Unsupervised Learning** |
|---|---|---|
| **Description** | Estimate or predict an output based on one or more inputs. | Find structure and relationships from inputs. No "supervising" output. |
| **Variables** | Explanatory and Response variables | Explanatory variables only |
| **Goal** | (1) predict new values or (2) understand existing relationships between explanatory and response variables | Group observations into clusters based on similarity |
| **Types of algorithms** | (1) Regression and (2) Classification | Clustering |

Section 2

# Fundamental Tradeoffs

# Accuracy-Interpretability Trade-off

- Simpler models are easier to interpret but typically give lower accuracy
- More complex models can give better prediction accuracy but results are harder to interpret

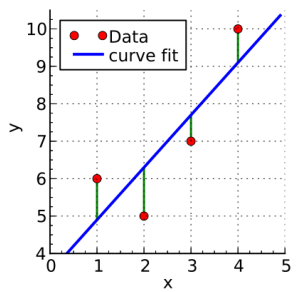


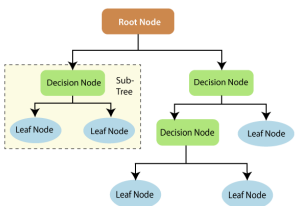

**After-Class Reading**

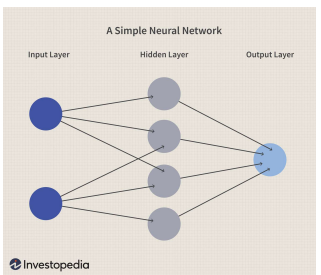
Due to time constraints, we only cover tree-based models in depth. Learn about other ML models in this video.

# Comparison of Classic Supervised Learning Models

- Linear regression class models (easy to interpret, low accuracy)
  - Linear regression coefficients have economic interpretations but prediction accuracy is low
- **Tree-based Models (balance between interpretability and accuracy)**
  - **Decision tree**, **random forest**, and gradient boosting models
- Neural-network based models (hard to interpret, high accuracy)
  - Deep learning only gives estimated weights that have no direct business interpretations



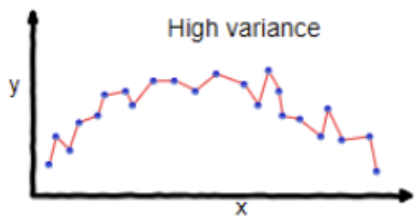



# Bias Error and Variance Error

- After we have trained a machine learning model, we can test the model performance by looking at the errors of predictions.
  - **bias** measures how far off the model's predictions are from the true values on average (systematic error)
  - **variance** measures how much the model's predictions vary when trained on different datasets (sensitivity to training data)

# Overfitting

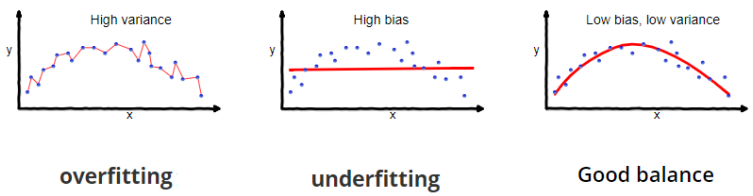
- If a predictive model **learns from one single training dataset too well**, then it may be too rigid and specialised and thus have a higher chance of failing to make predictions for another dataset accurately. This problem is called **overfitting**.
- Overfitting leads to **low bias** but **high variance**. This is not ideal because with supervised learning models, we want to have higher prediction accuracy for new data.




**overfitting**

# Underfitting

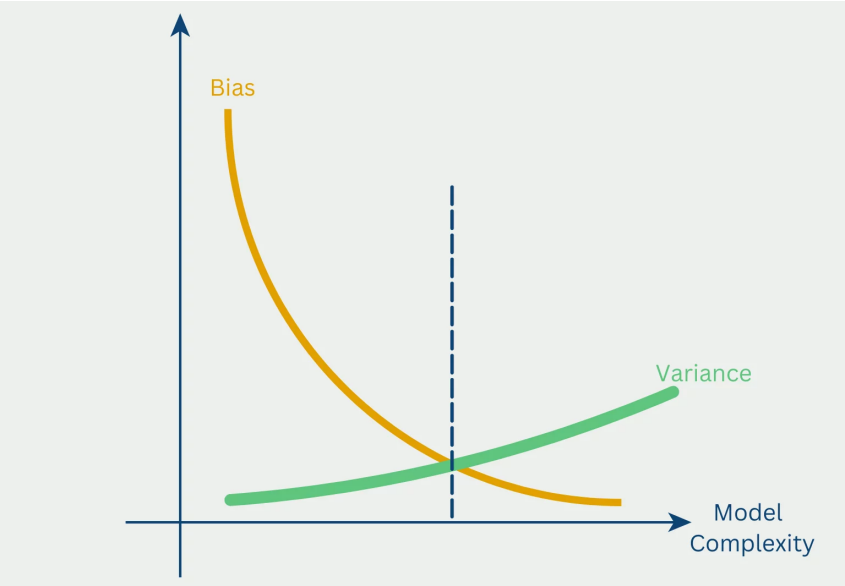
- On another extreme, **underfitting** occurs when a predictive model cannot sufficiently capture the DGP even on the historical training data.
- Underfitting leads to **high bias** but typically **low variance**. An underfitting model performs poorly on both training and test data, which should be avoided by all means.
- To mitigate the underfitting problem, we need to select more suitable or more complex models.




overfitting | underfitting | Good balance

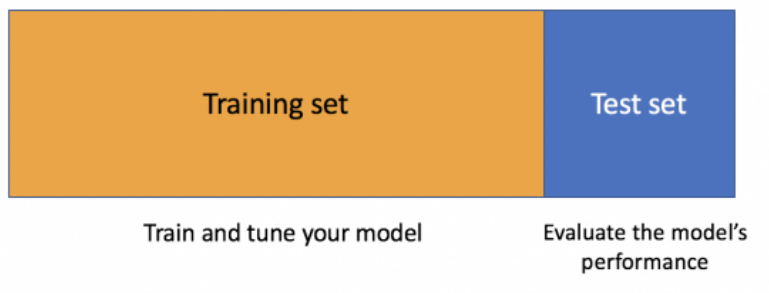
# Bias-Variance Trade-off

- Increasing model complexity (e.g., adding more layers to a neural network or more branches to a decision tree) typically decreases bias but increases variance. The model fits the training data better but becomes more sensitive to it, leading to overfitting.
- Decreasing model complexity (e.g., using a simpler model like linear regression) typically increases bias but decreases variance. The model is more general but may miss underlying patterns in the data, leading to underfitting.
- Hence we face a **bias-variance trade-off** or **bias-variance dilemma**.



# How to Mitigate Overfitting

- To mitigate the overfitting problem, when training predictive models, we need to use the **cross-validation** technique by splitting the full **historical data** into a **training set** and a **test set**.
  - **A training set** (70% - 80% of labelled data)**:** we train the ML model based on the training set.
  - **A test set** (20% - 30% of labelled data)**:** Using the trained ML model from the training data, we can make predictions for the test data. However, we do observe the actual outcomes for the test set, so that we can evaluate the prediction accuracy by comparing the predicted outcomes versus the actual outcomes.




**Note**

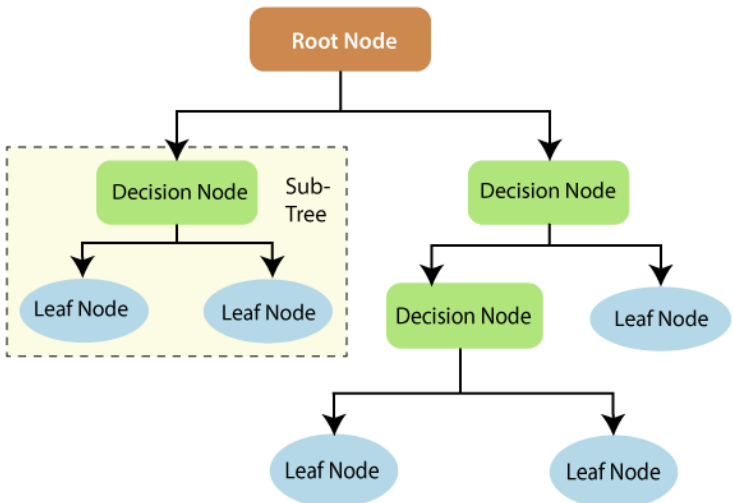
For more complicated models with hyper-parameters such as deep learning models, we may even need to split our data into 3 sets (training, validation, and test sets).

Section 3

# Decision Tree

# Introduction to Decision Tree

- A **decision tree** is a tree-like structure, which can be used for both classification and regression tasks.



# Business Objective: Predict Customer Response to Marketing Offers

- M&S made marketing offers to customers in the data, and the variable `Response` represents whether or not customers responded to the offer in the previous similar marketing campaign.
- **Business objective**: Based on the historical data `data_full`, we want to train a decision tree model to predict the outcome variable `Response` based on `Recency` and `total_spending`.
- **Data collection and cleaning:**

```
pacman::p_load(dplyr, modelsummary)

data_full <- read.csv("../../data/data_full.csv") %>%
    mutate(
        total_spending = MntWines + MntFruits + MntMeatProducts +
            MntFishProducts + MntSweetProducts + MntGoldProds
    )
```

# Implementation of Decision Tree in R

- Package `rpart` provides an efficient implementation of decision trees in R; Package `rpart.plot` provides visualizations of decision trees
  - `formula`: `Response ~ Recency + total_spending` means that we want to predict the outcome variable `Response` based on the explanatory variables `Recency` and `total_spending`. In R, we use `~` to separate the outcome variable and the explanatory variables for all supervised learning tasks.
  - `data`: the training dataset to train the model
  - `method`: "class" for **classification** tasks, "anova" for **regression** tasks

```
# Load the necessary packages
pacman::p_load(rpart, rpart.plot)

# Below example shows how to train a decision tree
tree1 <- rpart(
    formula = Response ~ Recency + total_spending,
    data = data_full,
    method = "class" # classification task; or 'anova' for regression
)

# visualize the tree
rpart.plot(tree1)
```

# How to Measure Split Quality: Classification Tasks

- For classification tasks, the goal is to split the data to create nodes that are as "pure" as possible, meaning they contain instances of a single class.
- Two common metrics are used to measure the quality of a split: Gini Impurity and Information Gain (based on Entropy). Gini impurity is more commonly used in practice due to its computational efficiency.

## Gini Impurity

- Formula: $Gini = 1 - \sum_{i=1}^{C} (p_i)^2$, where $p_i$ is the proportion of samples of class $i$.
- A Gini score of 0 indicates a perfectly pure node. The algorithm seeks splits that minimize the weighted Gini impurity of the child nodes.

# Numeric Example

Let us start with a dataset of 10 customers, from which we observe X = total spending and Y = Response (1 for response, 0 for no response).

Case 1: Purest

- 10 customers responded
- 0 customers did not respond
- Gini = $1 - ((10/10)^2 + (0/10)^2) = 0$

Case 2: In-between

- 7 customers responded
- 3 customers did not respond
- Gini = $1 - ((7/10)^2 + (3/10)^2) = 0.42$

Case 3: Impurest

- 5 customers responded
- 5 customers did not respond
- Gini = $1 - ((5/10)^2 + (5/10)^2) = 0.5$

# Numeric Example: Split

If we split the 10 customers in Case 2 into two child nodes based on total spending at a threshold of 1396:

## Child Node 1 (Left)

- `total_spending < 1396`
- 4 customers total
  - 1 responded
  - 3 did not respond
- **Gini Calculation:**
  - $p_1 = 1/4 = 0.25$
  - $p_0 = 3/4 = 0.75$
  - Gini = $1 - (0.25^2 + 0.75^2) = 0.375$
  - This node is **impure**.

## Child Node 2 (Right)

- `total_spending >= 1396`
- 6 customers total
  - 6 responded
  - 0 did not respond
- **Gini Calculation:**
  - $p_1 = 6/6 = 1$
  - $p_0 = 0/6 = 0$
  - Gini $= 1 - (1^2 + 0^2) = 0$
  - This node is **pure**.

# Numeric Example: Weighted Gini and Gini Gain

The goal is to find the split that results in the lowest weighted Gini impurity.

1. **Calculate Weighted Gini of the Split**
   - Weight (Left) $= 4/10 = 0.4$
   - Weight (Right) $= 6/10 = 0.6$
   - Weighted Gini $= (0.4 \times 0.375) + (0.6 \times 0) = 0.15$
2. **Calculate Gini Gain (The Decision)**
   - Gini Gain = Gini (Parent) - Weighted Gini (Split)
   - Gini Gain $= 0.42 - 0.15 = 0.27$

Since the Gini Gain is positive, impurity was reduced, making this a good split. The `rpart` algorithm repeats this for all possible splits and chooses the one with the **highest Gini Gain**.

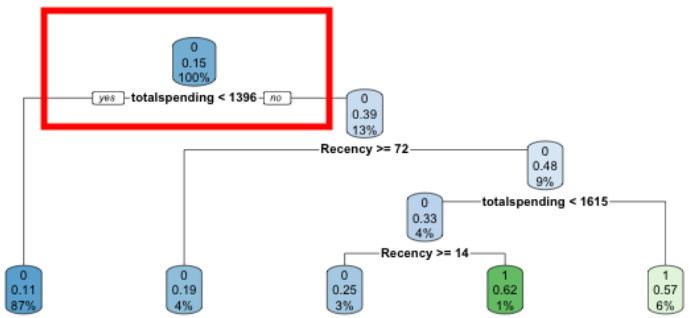
# (Optional) How to Measure Split Quality: Regression Tasks

- For regression tasks, the goal is to split the data to create nodes where the outcome values are as similar as possible.
- The most common metric used to measure the quality of a split is the **Sum of Squared Errors (SSE)**.

## Sum of Squared Errors (SSE)

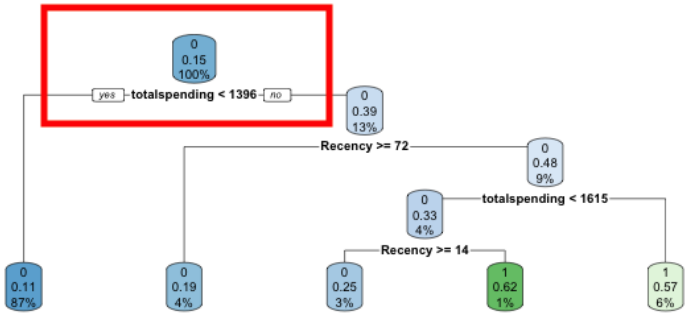
- Measures the total squared difference between the actual values and the mean value of the outcome variable within a node.
- Formula: $SSE = \sum_{i \in \text{node}} (y_i - \bar{y}_{\text{node}})^2$, where $y_i$ is the actual value and $\bar{y}_{\text{node}}$ is the mean value of the outcome in the node.
- The algorithm seeks the split that results in the largest reduction in the total SSE of the child nodes compared to the parent node.

# How Decision Tree Works: Step 1




Step 1. The decision tree (DT) will try to split customers into 2 groups based on each unique value of each variable, and see which split can lead to customers being most different in terms of outcome `Response`.

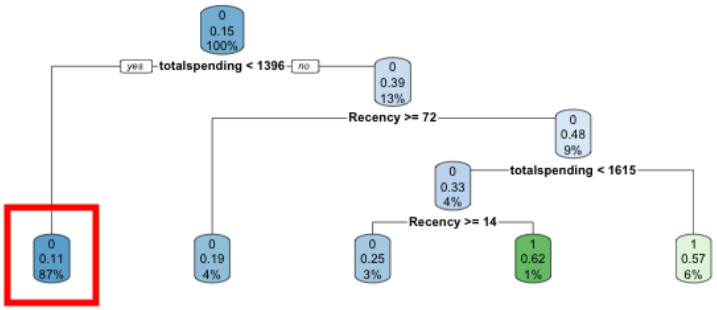
# How Decision Tree Works: Step 1




Step 1. The decision tree (DT) will try to split customers into 2 groups based on each unique value of each variable, and see which split can lead to customers being most different in terms of outcome `Response`.

- After this step, DT finds that total spending is the best variable and 1396 is the best cut-off.
- DT therefore splits customers into 2 groups based on 1396.
- In each node, the 3 numbers are: (1) predicted outcome, (2) predicted probability of outcome being 1, and (3) share of customers in the node

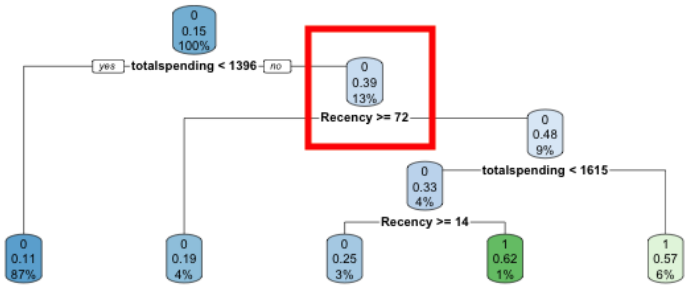
# How Decision Tree Works: Step 2




Step 2. For customers in the left branch (`total_spending` $< 1396$), DT will continue to split based on each unique value of each variable, and see which split can result in the customers being most different in terms of `Response`.

- However, DT couldn't find a cut-off that sufficiently differentiates customers, so DT stops in the left branch.

# How Decision Tree Works: Step 3 ...




Step 3. For customers in the right branch (`total_spending` $>=$ 1396), DT will continue to split based on each unique value of each variable, and see which split can result in the customers being most different in terms of Response.

- After this step, DT finds `Recency` is the best variable and 72 is the best cut-off. DT further splits customers into 2 groups.

Step 4. This process continues until DT determines that there is no need to further split customers.

# How Decision Tree Works: Step 4

- Once the tree is fully grown, we can use the tree to make predictions on new customers.
- For a new customer, we can follow the tree from the root node to the leaf node, and the predicted outcome is the outcome of the leaf node.
- In R, we can use the `predict()` function to make predictions on new customers, which returns the predicted outcome of the new customers. Note that the test data should have the **exact same variable names** as the training data.

```
# Make predictions on the mtcars
prediction_tree1 <- predict(tree1, data = data_test)
```

# Advantages of Decision Trees

- They are very interpretable.
- Making predictions is fast.
- It's easy to understand what variables are important in making the prediction. The internal nodes (splits) are those variables that most largely reduce the Gini Impurity/SSE (criteria for split).

Section 4

# Prediction Accuracy (Optional)

# Classification Tasks

For classification tasks, we can evaluate model performance using:

- **Confusion Matrix**: A table showing true positives, true negatives, false positives, and false negatives

| | **Predicted: No** | **Predicted: Yes** |
|---|---|---|
| **Actual: No** | True Negatives | False Positives |
| **Actual: Yes** | False Negatives | True Positives |

# Classification Tasks

Based on the confusion matrix, we can further compute the following metrics:

- **Accuracy**: The proportion of correct predictions

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Predictions}}$$

- **Precision**: Among predicted positives, how many are actually positive

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- **Recall (Sensitivity)**: Among actual positives, how many are correctly predicted

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- **F1-Score**: Harmonic mean of precision and recall

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

# Regression Tasks

For regression tasks, we can evaluate model performance using:

- **Mean Absolute Error (MAE)**: Average absolute difference between predicted and actual values

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- **Root Mean Square Error (RMSE)**: Square root of average squared differences

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- **Sum of Squared Errors (SSE)**: Total squared difference between predicted and actual values

$$\text{SSE} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- Lower MAE/RMSE/SSE indicate better predictions

**Note**

In R, the `caret` package provides functions to compute these metrics easily.

# Business Metrics as Evaluation Criteria

- In practice, we may also use business metrics such as ROI, profit, or customer lifetime value (CLV) as evaluation criteria for predictive models.
- For example, in targeted marketing campaigns, we may want to evaluate predictive models based on the ROI of the marketing campaign when using the model to select target customers. The intuition is that a better predictive model should lead to a higher ROI for the marketing campaign. We will see an example of this in the case study later.

Section 5

# Random Forest

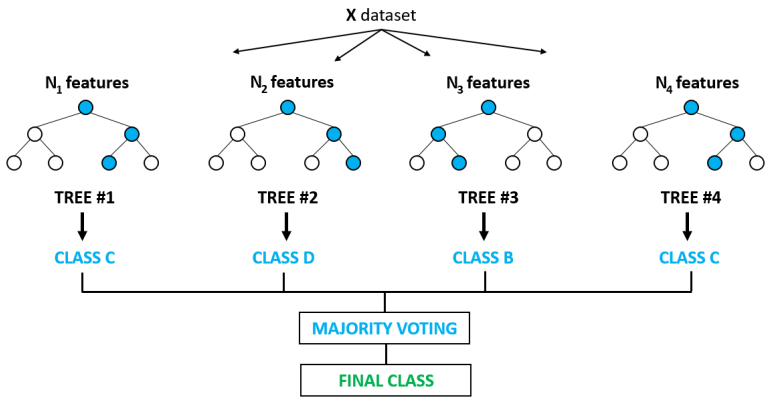
# Disadvantages of Decision Trees

- Single regression trees tend to overfit, resulting in unstable predictions.
- Due to the high variance, single regression trees tend to have poor predictive accuracy.

# Random Forest

- To overcome the overfitting tendency of a single decision tree, random forest has been developed by (Breiman 2001).
  - Instead of using all customers, each tree is grown to a **subsample** of customers instead of all customers (e.g., 70% of training data)
  - Instead of using all features for splitting, each tree is grown to a **subset** of features instead of all features (e.g., 3 out of 5 features)

# Visualization of Random Forest




For a new customer,

- Each tree gives a prediction of the outcome
- Random forest takes the average (for regression tasks) or majority vote (for classification tasks) of all trees' predictions as the final prediction

# Implementation of Random Forest in R

- Package `ranger` provides implementation of random forest in R.
- `ranger()` is the function in the package to train a random forest; refer to its help function for more details.
- The following code shows how to train a random forest consisting of 500 decision trees, where the outcome variable is `Response`, and the predictors are `total_spending` and `Recency`.

```
pacman::p_load(ranger)
randomforest1 <- ranger(
    formula = Response ~ total_spending + Recency, # formula
    data = data_full, # dataset to train the model
    num.trees = 500, # 500 decision trees
    seed = 888, # make sure of replication
    probability = TRUE # to return predicted probabilities
)
```

# Make Predictions from Random Forest

- After we train the predictive model, we can use the `predict()` function to make predictions
  - The 1st argument is the trained model object
  - The 2nd argument is the dataset on which to make predictions

```
# Make predictions on the mtcars
prediction_rf <- predict(randomforest1, data = data_full)

# Because prediction_rf is a list object
# Need to use $ to extract the predicted value as a numeric vector
prediction_rf$predictions
```

# After-Class Reading

- (recommended) Decision tree in R
- (recommended) Random forest in R

Breiman, Leo. 2001. "Random Forests." *Machine Learning* 45 (1): 5–32. https://doi.org/10.1023/A:1010933404324.