ARTIFICIAL INTELLIGENCE NEEDS REAL INTELLIGENCE

Classification III

Professor: Tianyuan ZHANG
tianyuan.zhang@kedgebs.com

KEDGE BUSINESS SCHOOL

Recap of Previous Session

- Metrics for multi-class classification
  - Compute confusion matrix in multi-class case
    - Select a class → Positive class
    - All other classes → Negative class

	Target Cat	Target Dog	Target Monkey
Predict Cat	3	1	1
Predict Dog	1	4	2
Predict Monkey	2	1	5

	Target Not Monkey	Target Monkey
Predict Not Monkey	TN 9	FN 3
Predict Monkey	FP 3	TP 5

	Target Not Dog	Target Dog
Predict Not Dog	TN 11	FN 2
Predict Dog	FP 3	TP 4

	Target Not Cat	Target Cat
Predict Not Cat	TN 12	FN 3
Predict Cat	FP 2	TP 3

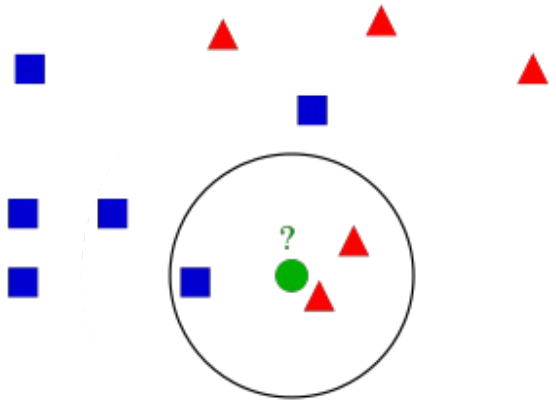
- Calculate class-wise metrics based on the above confusion matrix

Recap of Previous Session

- Metrics for multi-class classification
  - Class-wise metrics
    - Precision, recall, F1-score
    - ROC curve, AUC
  - Overall performance metrics can be represented by the average of class-wise metrics:
    - Macro Average: unweighted
    - Weighted Average: use support as weight
      - Support is the number of actual occurrences of the class in the dataset
  - Accuracy is not class-wise metric
    - Respect to the definition: The proportion of correct predictions to all predictions made

Recap of Previous Session

- K-Nearest Neighbors
  - Classify a data points based on the classes of its nearest neighbors
  - K is the number of nearest neighbors to be considered



Class A *Class B* *To be classified*

*If **K = 3**, consider 3 nearest neighbors:*

- *One is class A*
- *Two are class B*

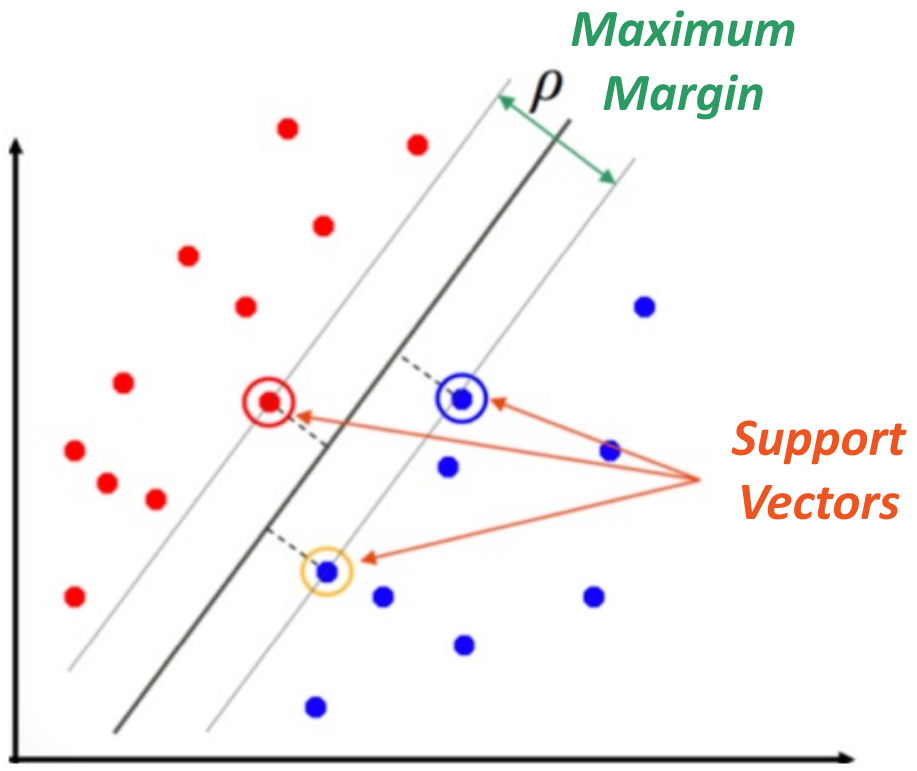
The prediction is class B

Recap of Previous Session

- K-Nearest Neighbors
  - A voting system
    - The value of K
      - A large K smoothen the decision boundaries, which might lead to under-fitting
      - A small K increase the complexity of model, which might lead to over-fitting
    - Identify the nearest neighbors
      - Calculate distance between the data point and its neighbors
        - Perform feature scaling before calculating distances
      - Ranking the neighbors based on distance and find the nearest ones
    - Generate prediction
      - Neighbors are equally important → A simple majority voting
      - The closer one is more important → Weight neighbors by the inverse of their distances

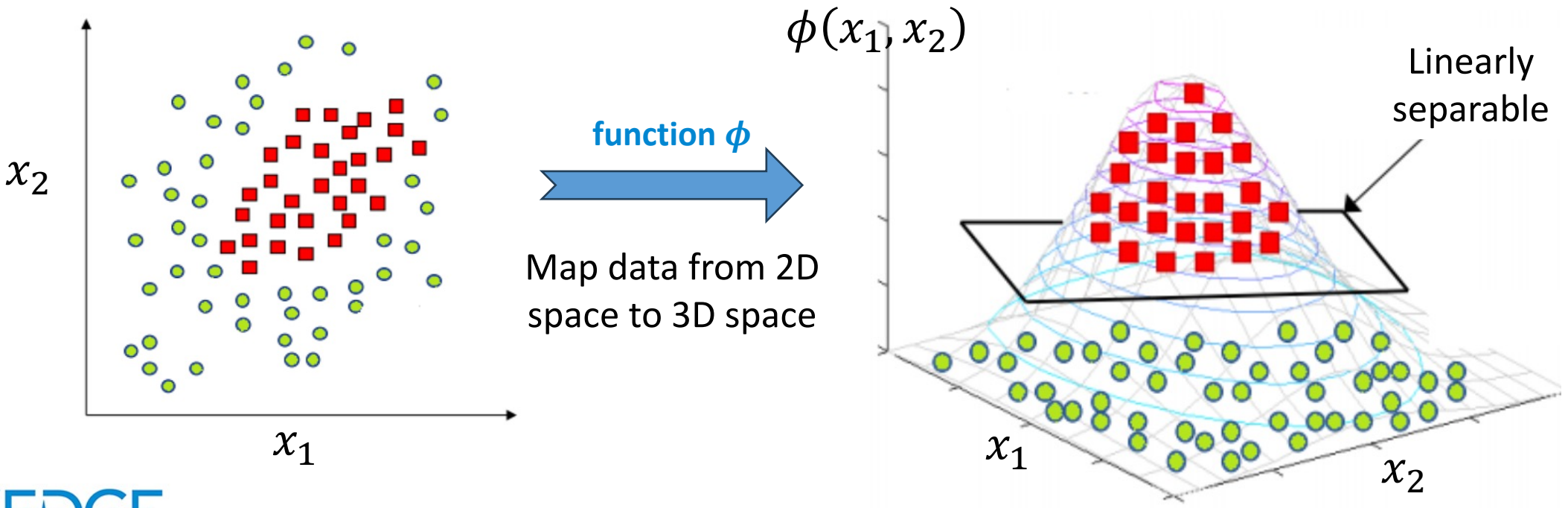
Recap of Previous Session

- Support Vector Machine
  - Finds a hyperplane that divides the feature space into different parts, each represents a category
  - Select the hyperplane that separates the data points with the maximum margin
    - Data points closet to the hyperplane are support vectors
    - Margin ρ of a hyperplane is the sum of the distances between support vectors and the hyperplane.



Recap of Previous Session

- Support Vector Machine
  - The kernel trick
    - Convert nonlinearly separable low-dimensional space to a linearly separable high-dimensional space



Outline

- **Decision Tree**
- Hyper-parameter Tuning
- Cross-validation

Decision Tree

- A decision tree is a model composed of a collection of "questions" organized hierarchically in the shape of a tree.
- Components of a decision Tree:
  - Nodes
    - ***Root node***
    - Leaf node
    - Decision node
  - Branches

The decision tree starts from a ***root node***.

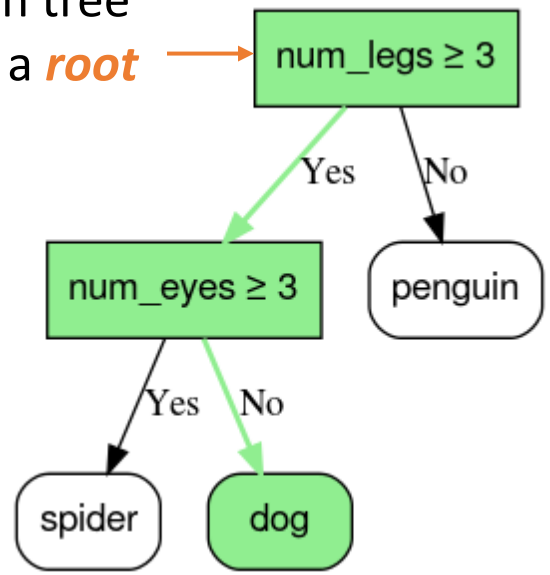


Image source: https://developers.google.com/machine-learning/decision-forests/decision-trees

Decision Tree

- A decision tree is a model composed of a collection of "questions" organized hierarchically in the shape of a tree.
- Components of a decision Tree:
  - Nodes
    - Root node
    - ***Leaf node***
    - Decision node
  - Branches

Leaf nodes are the terminal nodes that represent a class.

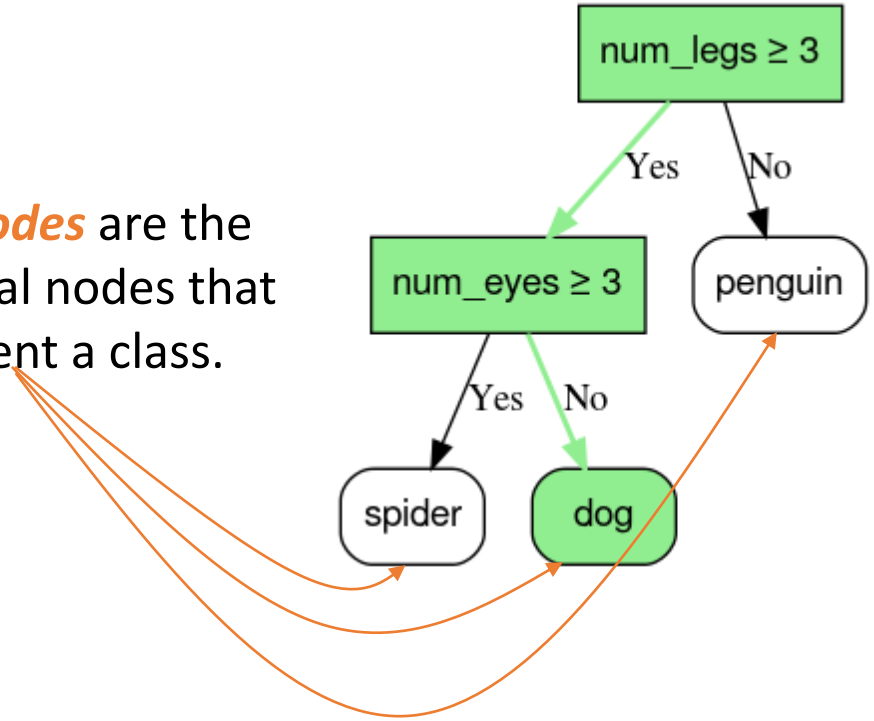


Image source: https://developers.google.com/machine-learning/decision-forests/decision-trees

Decision Tree

- A decision tree is a model composed of a collection of conditions organized hierarchically in the shape of a tree.
- Components of a decision Tree:
  - Nodes
    - Root node
    - Leaf node
    - ***Decision node***
  - Branches

Decision nodes split the data by checking conditions.

- Include root node

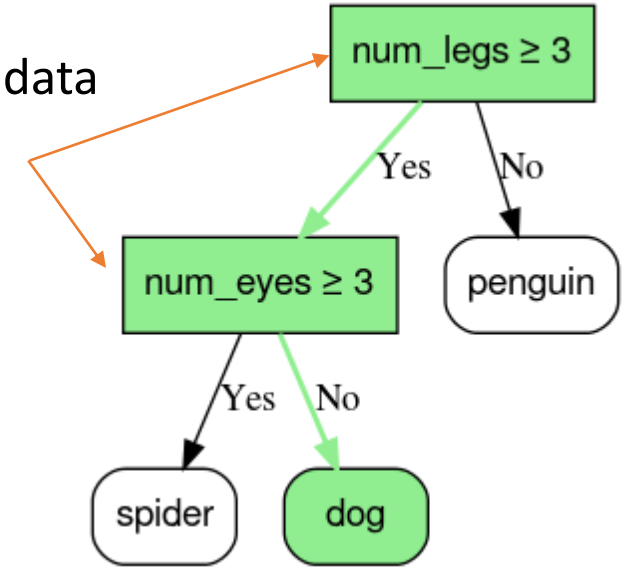


Image source: https://developers.google.com/machine-learning/decision-forests/decision-trees

Decision Tree

- A decision tree is a model composed of a collection of ***conditions*** organized hierarchically in the shape of a tree.
- Components of a decision Tree:
  - Nodes
    - Root node
    - Leaf node
    - Decision node
  - Branches

Each decision node contains a ***condition*** to check.
The common form of a condition is:

- ***Feature $\geq$ Threshold ?***

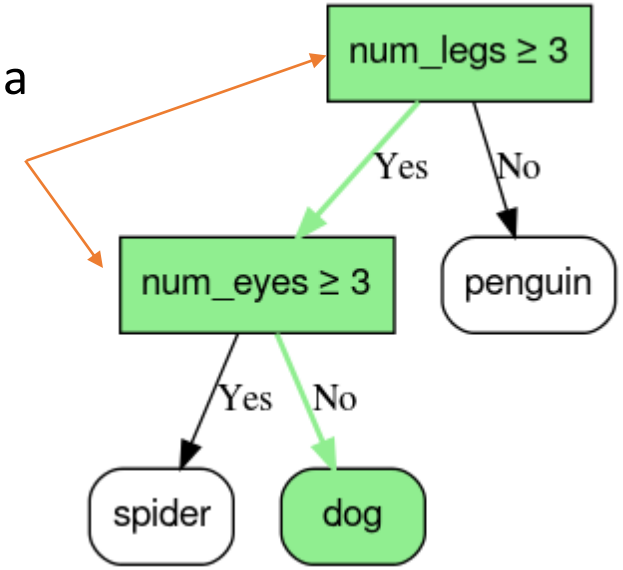


Image source: https://developers.google.com/machine-learning/decision-forests/decision-trees

Decision Tree

- A decision tree is a model composed of a collection of conditions organized hierarchically in the shape of a tree.
- Components of a decision Tree:
  - Nodes
    - Root node
    - Leaf node
    - Decision node
  - ***Branches***

Branches are paths / edges between different nodes

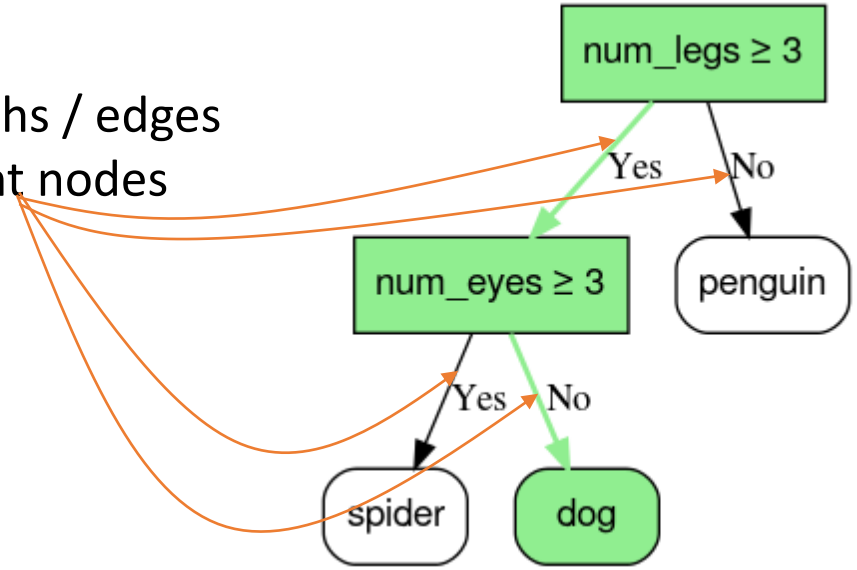


Image source: https://developers.google.com/machine-learning/decision-forests/decision-trees

Decision Tree

- Inference of a decision tree
  - Routing an example from the root node to one of the leaf nodes according to conditions.
- An example
  - An animal with:
    - 4 legs
    - 2 eyes

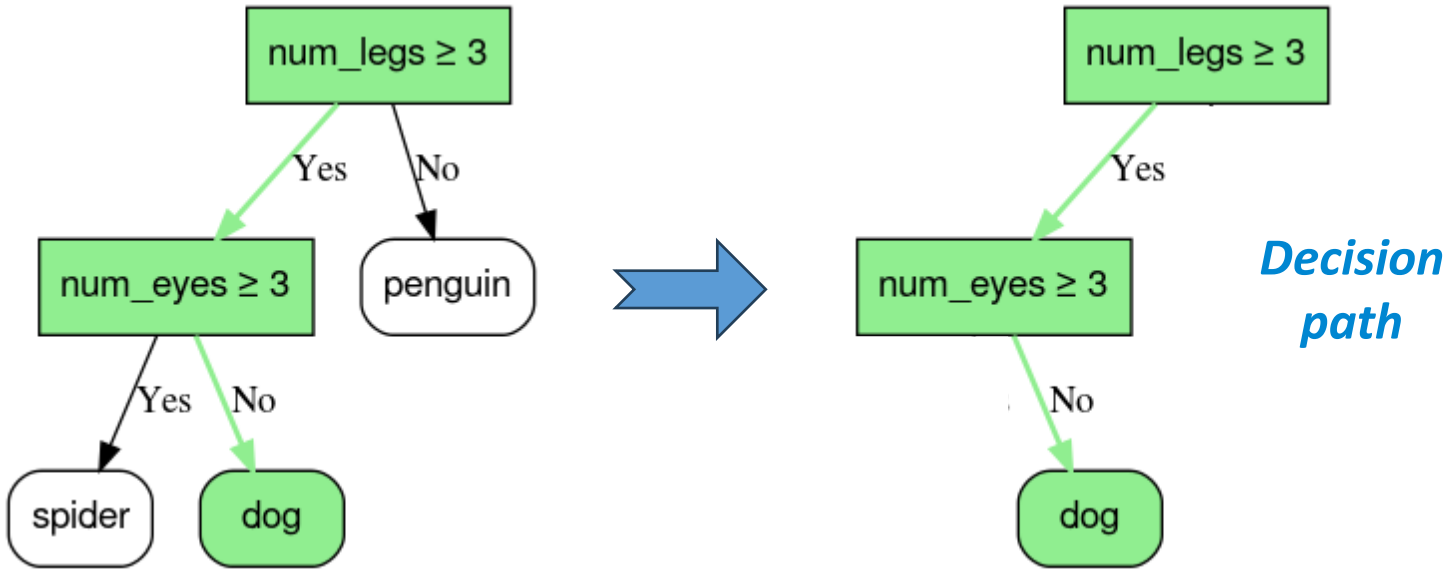


Image source: https://developers.google.com/machine-learning/decision-forests/decision-trees

Decision Tree

- Growing a decision tree → Training
  1. Create a root node
  2. Choose a pair of feature and threshold as the condition of this node
     1. Split the data into to branches
     2. Each branch reaches a child node
  3. Repeat step 2 recursively for each node
  4. Stop until all child nodes became leaf nodes
     1. Cannot find a condition for the child node → Leaf node
     2. Or all data in the child node have same class → Leaf node

Decision Tree

- Growing a decision tree → Training
  - Step 1. Create a root node

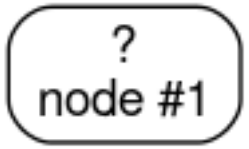


Image source: https://developers.google.com/machine-learning/decision-forests/decision-trees

Decision Tree

- Growing a decision tree → Training
  - Step 1. Create a root node
  - Step 2. Find the condition $x_1 \geq 1$ for root node
    - Split the data into two branches
    - Two child nodes are created

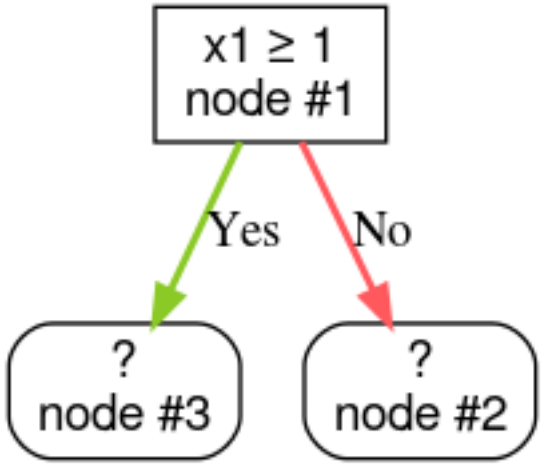


Image source: https://developers.google.com/machine-learning/decision-forests/decision-trees

Decision Tree

- Growing a decision tree → Training
  - Step 1. Create a root node
  - Step 2. Find the condition $x_1 \geq 1$ for root node
  - Step 3. Fail to find the condition for node #2
    - All instances in node #2 belongs to the same class
    - Make node #2 the leaf node

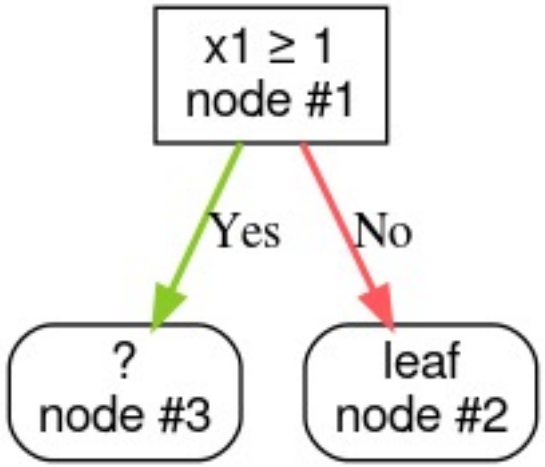


Image source: https://developers.google.com/machine-learning/decision-forests/decision-trees

Decision Tree

- Growing a decision tree → Training
  - Step 1. Create a root node
  - Step 2. Find the condition $x_1 \geq 1$ for root node
  - Step 3. Fail to find the condition for node #2
  - Step 4. Find the condition $x_2 \geq 0.5$ for node #3
    - Split the data into two branches
    - Two child nodes are created

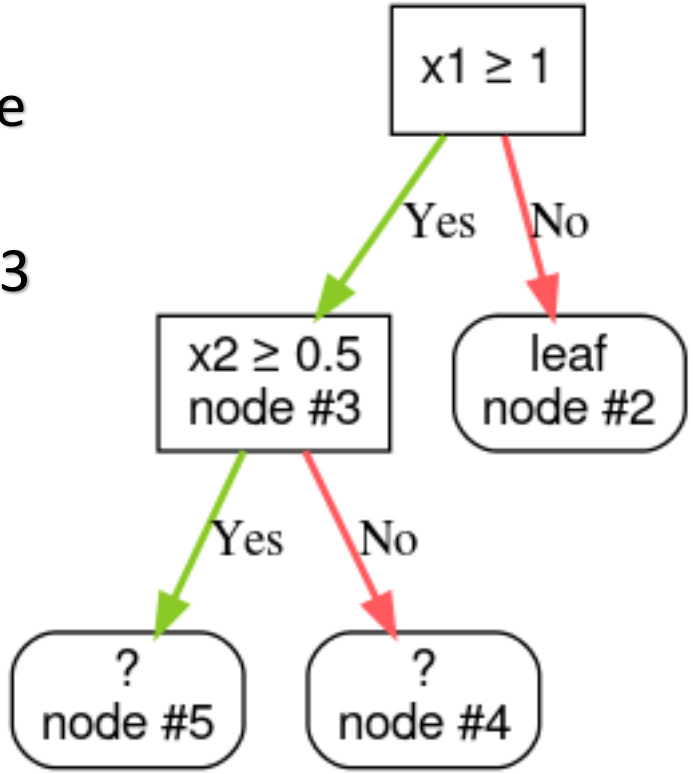


Image source: https://developers.google.com/machine-learning/decision-forests/decision-trees

Decision Tree

- Growing a decision tree → Training
  - Step 1. Create a root node
  - Step 2. Find the condition $x_1 \geq 1$ for root node
  - Step 3. Fail to find the condition for node #2
  - Step 4. Find the condition $x_2 \geq 0.5$ for node #3
  - …
  - …
  - Stop until all child nodes became leaf nodes

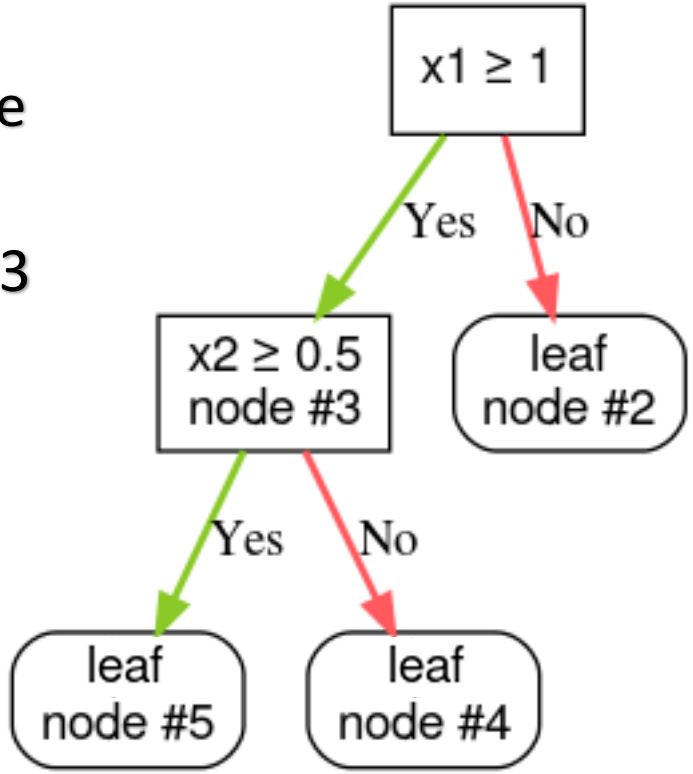
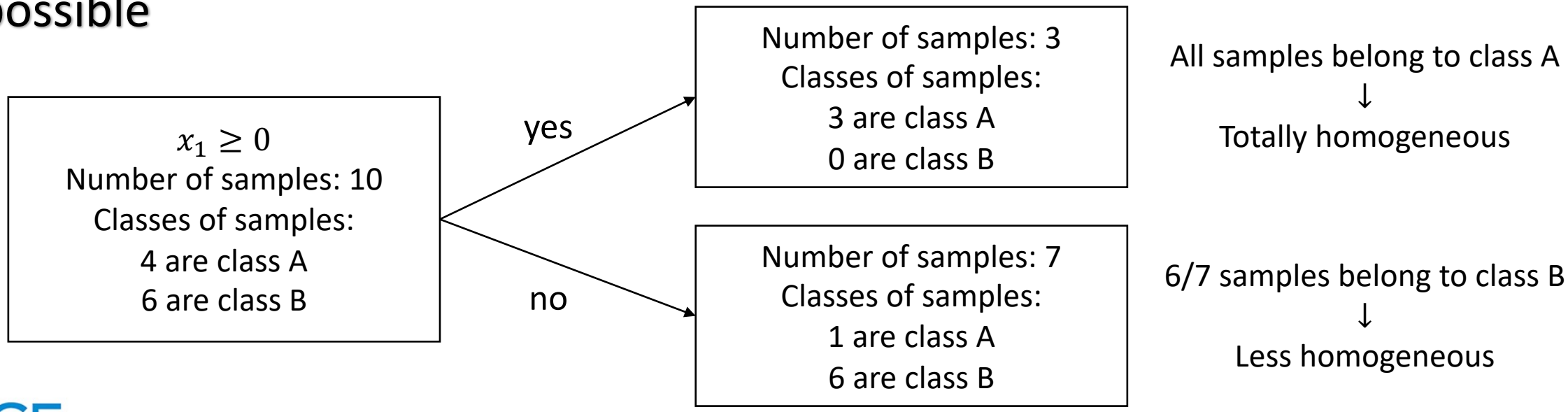


Image source: https://developers.google.com/machine-learning/decision-forests/decision-trees

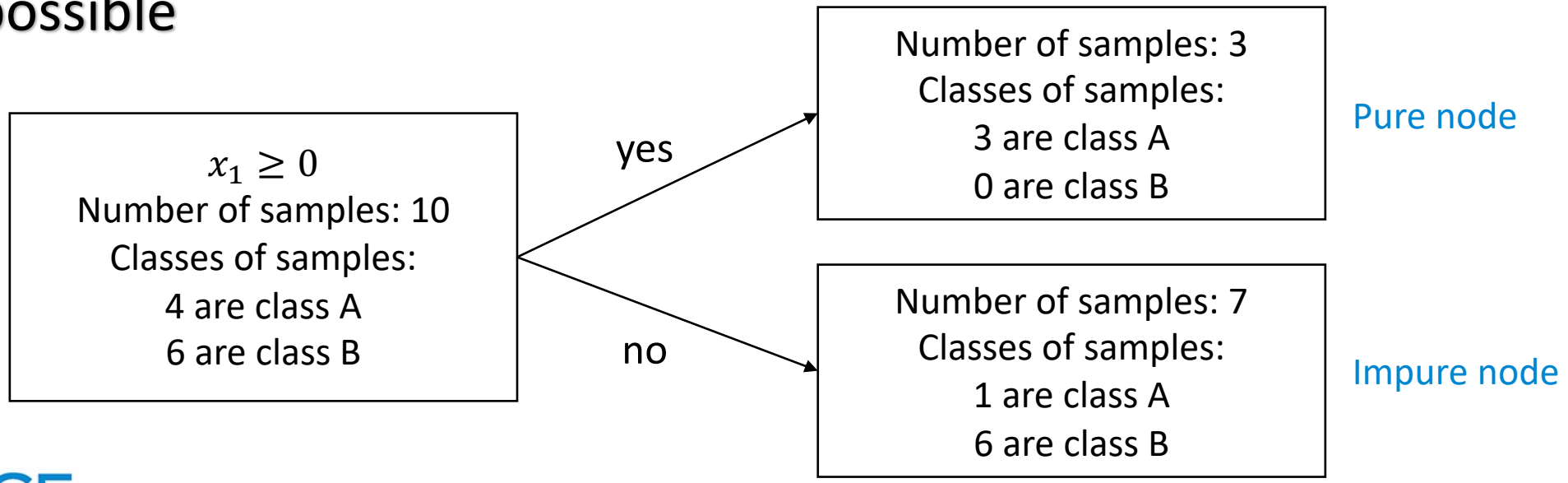
Decision Tree

- How to decide the condition for each decision node?
  - Which feature to choose?
  - Which threshold to use for split?
- Choose the condition that makes the child nodes to be as homogeneous as possible



Decision Tree

- How to decide the condition for each decision node?
  - Which feature to choose?
  - Which threshold to use for split?
- Choose the condition that makes the child nodes to be as homogeneous as possible



Decision Tree

- How to decide the condition for each decision node?
  - Which feature to choose?
  - Which threshold to use for splitting?
- Choose the condition that makes the child nodes to be as homogeneous as possible
  - Choose the condition that maximize the purity of all child nodes
  - Use entropy to measure the purity

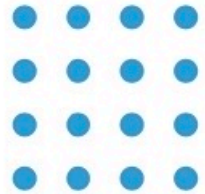
Decision Tree

- Entropy
  - A measure of disorder

High Entropy

- High Disorder
- Messy
- High Randomness;
- High uncertainty
- Low certainty

Increase entropy



Low Entropy

- Well- ordered
- Neat
- Low Randomness
- Low uncertainty
- High certainty

Decision Tree

- Entropy
  - A measure of disorder
  - In binary classification:

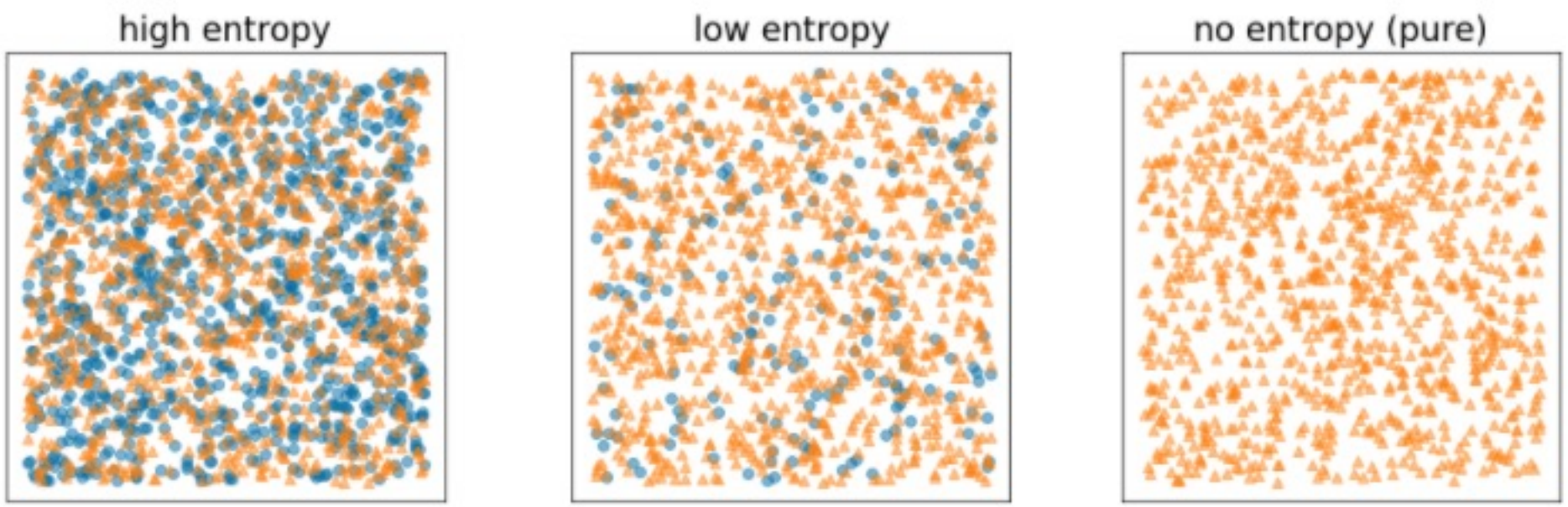
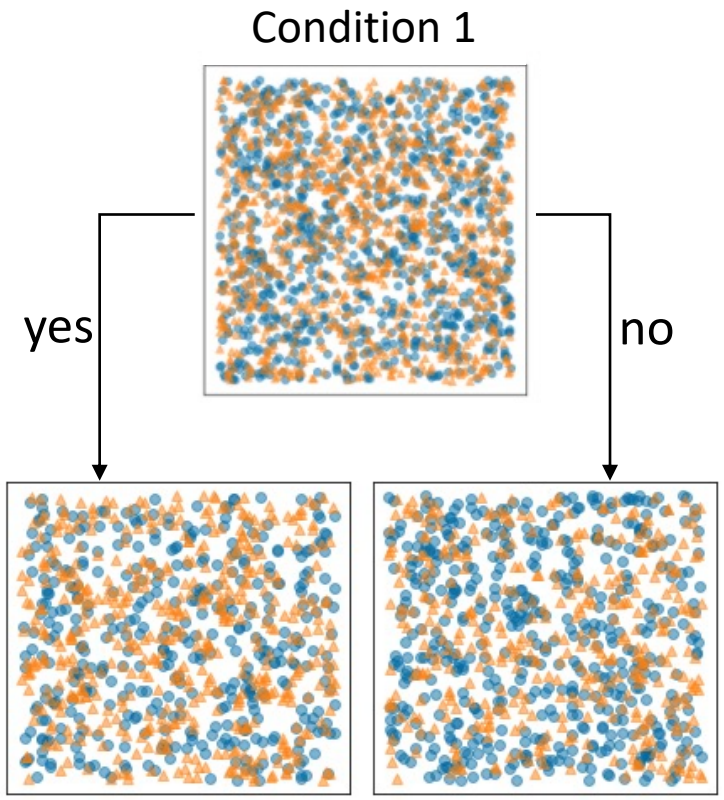


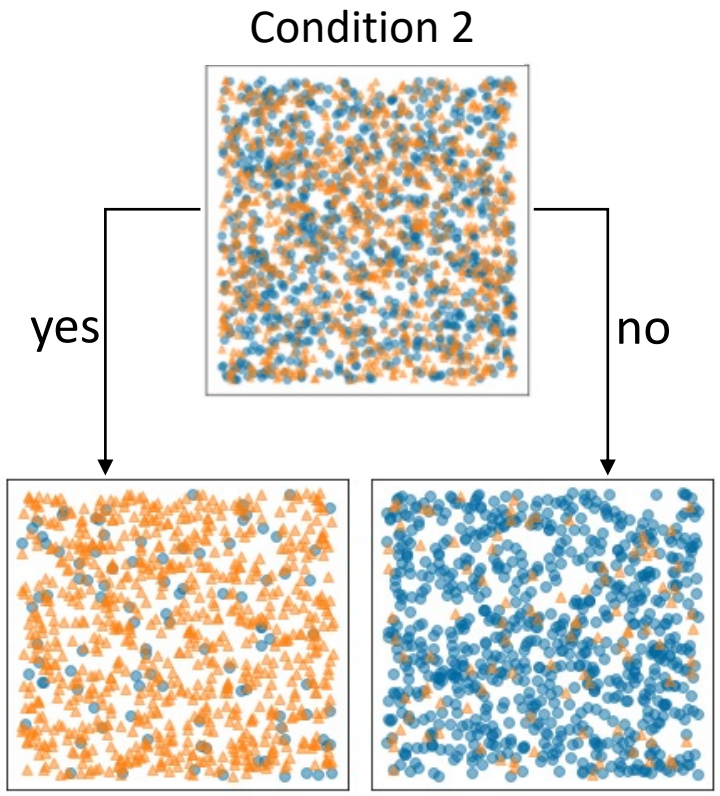
Image source: https://developers.google.com/machine-learning/decision-forests/decision-trees

Decision Tree

- Entropy



A bad condition



A good condition

Image source: https://developers.google.com/machine-learning/decision-forests/decision-trees

Decision Tree

- Entropy
  - A measure of disorder
  - In binary classification: class A & class B

$$\text{Entropy} = -(p_A \log_2 p_A + p_B \log_2 p_B)$$

  - p_A is the probability an instance in the node belongs to class A
    - The proportion of class A in the node
  - p_B is the probability an instance in the node belongs to class B
    - The proportion of class B in the node

Decision Tree

- Entropy
  - A measure of disorder
  - In binary classification: class A & class B

$$\text{Entropy} = -(p_A \log_2 p_A + p_B \log_2 p_B)$$

  - When $p_A = p_B = 0.5$, Entropy = 1

Decision Tree

- Entropy
  - A measure of disorder
  - In binary classification: class A & class B

$$\text{Entropy} = -(p_A \log_2 p_A + p_B \log_2 p_B)$$

$$\downarrow$$

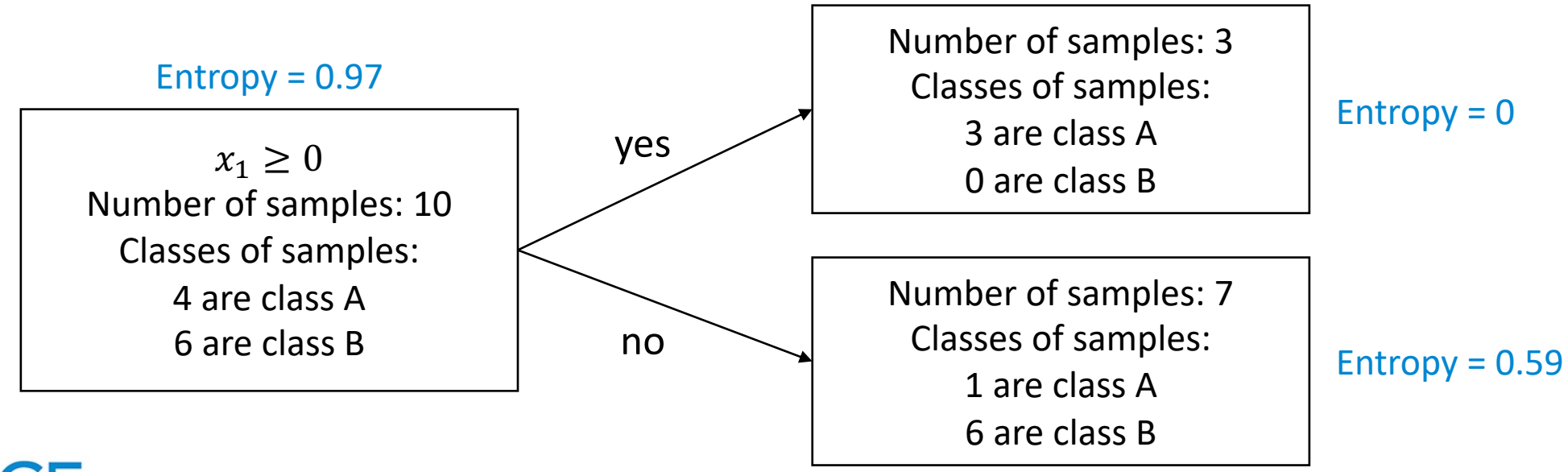
$$\text{Entropy} = -1 \times 0 - 0 \times (-\infty) = 0$$

- When $p_A = 1$ and $p_B = 0$, Entropy = 0

Decision Tree

- Entropy
  - In binary classification: class A & class B

$$\text{Entropy} = -(p_A \log_2 p_A + p_B \log_2 p_B)$$



Decision Tree

- How to decide the condition for each decision node?
  - Maximize the reduction of entropy → Maximize the information gain

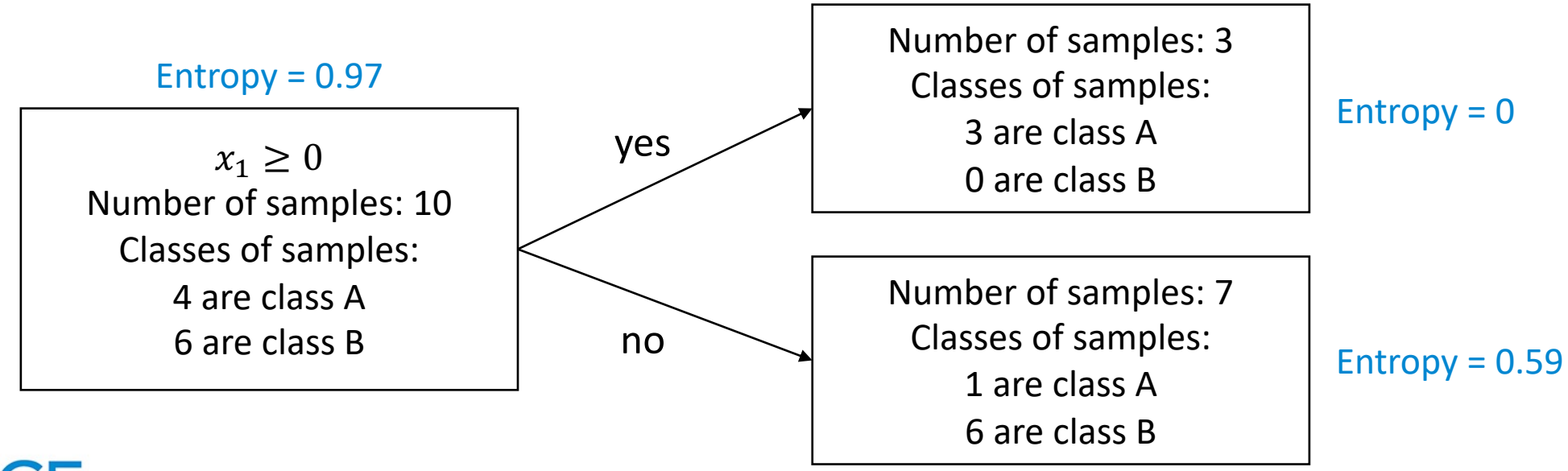
$$\text{Information Gain} = \text{Entropy}_{\text{before}} - \text{Entropy}_{\text{after}}$$

(Weighted) average entropy of child nodes

Decision Tree

- How to decide the condition for each decision node?
  - Maximize the reduction of entropy → Maximize the information gain

$$\text{Information Gain} = 0.97 - \frac{3}{10} \times 0 - \frac{7}{10} \times 0.59 = 0.56$$



Decision Tree

- How to decide the condition for each decision node?
  - Different implementations have different metrics to maximize
    - Information Gain
    - Information Gain Ratio
    - Gini Impurity Index
  - `sklearn.tree.DecisionTreeClassifier`

Parameters:	**criterion : *{"gini", "entropy", "log_loss"}, default="gini"*** The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "log_loss" and "entropy" both for the Shannon information gain, see Mathematical formulation.

Decision Tree

- Decision tree is prone to over-fitting
  - Decision tree classifier can fit super complex decision boundaries by creating a lot of very detailed conditions
  - Growing a decision tree without complexity restriction will lead to a tree with all pure leaf nodes
    - Some of the pure leaf nodes can contains only one instance which is an outlier or noisy data

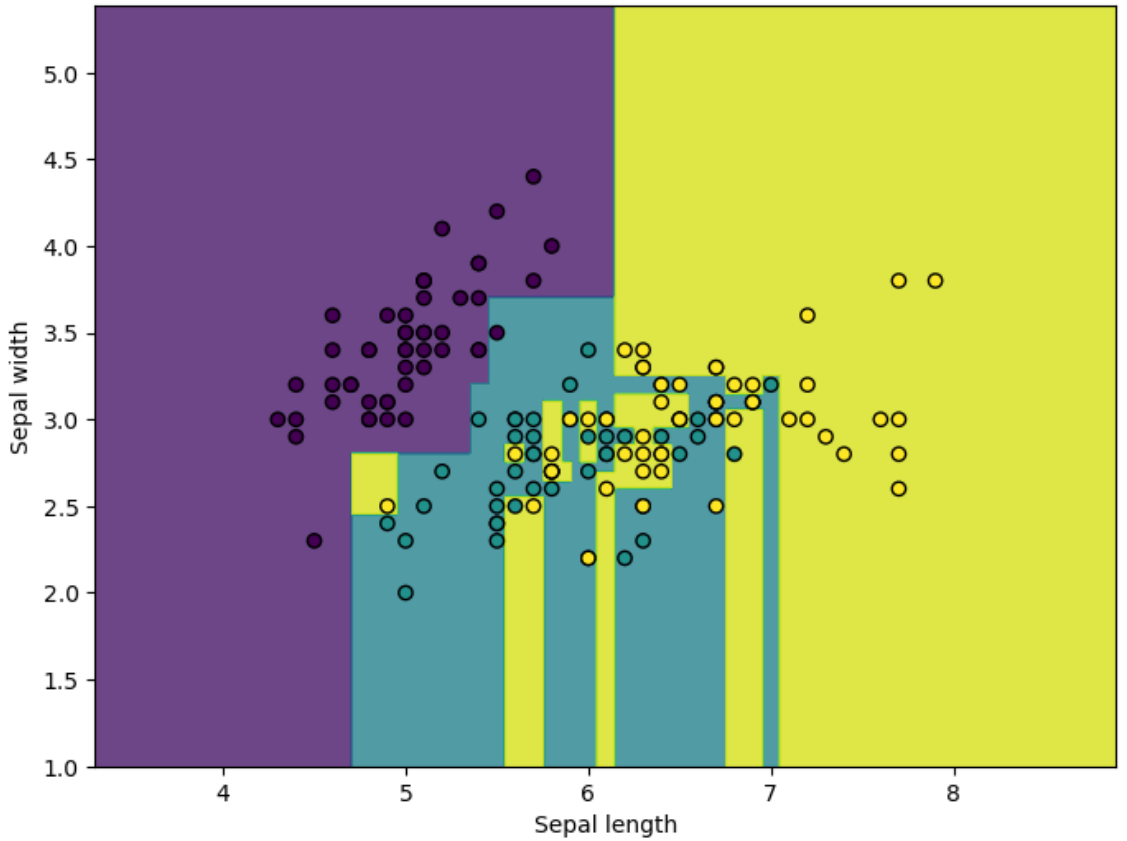
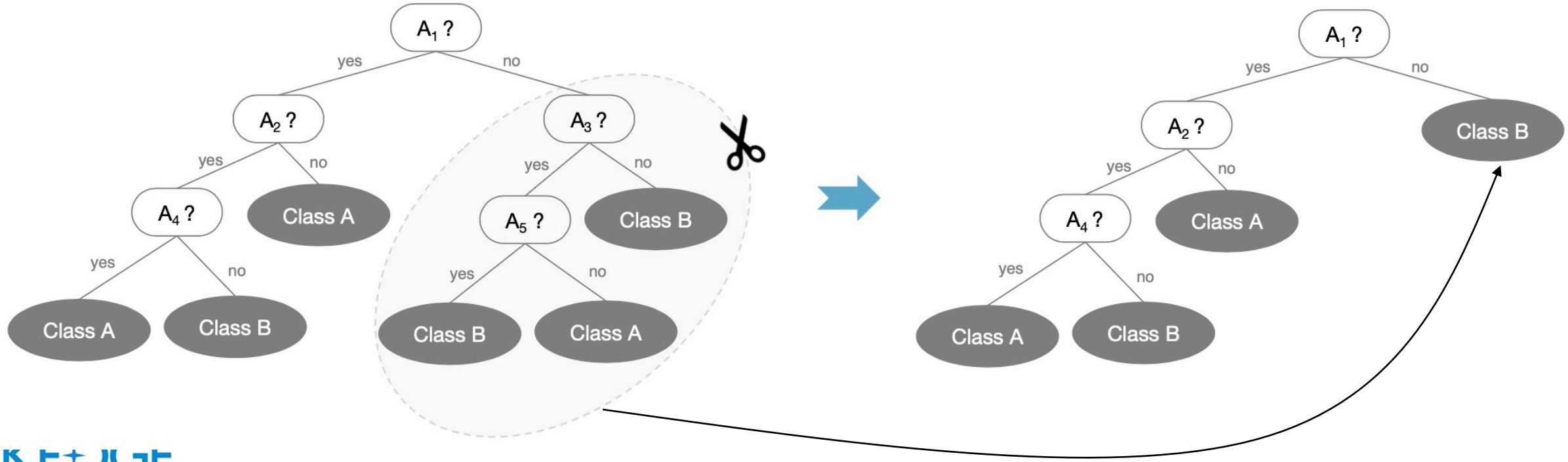


Image source: https://pub.aimind.so/enhancing-decision-trees-with-pre-pruning-a-guide-to-prevent-overfitting-1d81de7e3c98

Decision Tree

- Avoid over-fitting in decision trees by **pruning**
  - Pruning is the process of trimming down a decision tree to reduce complexity and prevent over-fitting.



Decision Tree

- Avoid over-fitting in decision trees by **pruning**
  - Pruning is the process of trimming down a decision tree to reduce complexity and prevent over-fitting.
  - Two strategies of pruning
    - Post-pruning
      1. Grow a complete tree with pure leaf nodes
      2. Remove some branches according to pre-defined criteria
      3. Replace the removed branches by a leaf node
         1. The leaf node is classified as the most frequent class
    - Pre-pruning

Decision Tree

- Avoid over-fitting in decision trees by **pruning**
  - Pruning is the process of trimming down a decision tree to reduce complexity and prevent over-fitting.
  - Two strategies of pruning
    - Post-pruning
    - Pre-pruning
      - Set a series of criteria to stop the growth early
      - Prevent the decision tree from becoming too complex by early stopping
      - `sklearn.tree.DecisionTreeClassifier` adopts the pre-pruning strategy

Decision Tree

- Pre-pruning in `sklearn.tree.DecisionTreeClassifier`
  - `max_depth`
    - maximum depth of a tree
  - `min_samples_split`
    - minimum number of samples required to split
  - `min_samples_leaf`
    - minimum number of samples required to be at a leaf node
  - `max_leaf_node`
    - maximum number of leaf nodes in a tree
  - `min_impurity_decrease`
    - minimum impurity decrease required to split

Outline

- Decision Tree
- **Hyper-parameter Tuning**
- Cross-validation

Hyper-parameter Tuning

- Hyper-parameter
  - The value is used to control models' training process
  - The value is defined before models' training process
  - The value remains unchanged during models' training process
- Parameter
  - The value is estimated by fitting the model to the training dataset

Hyper-parameter Tuning

- For a polynomial regression model
  - $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_n x^n$
  - Parameter
    - $\beta_0, \beta_1, \beta_2, \ldots, \beta_n$
  - Hyper-parameter
    - The degree n of the polynomial

Hyper-parameter Tuning

- Even the training dataset and learning algorithm are the same, different hyper-parameters will lead to different performance
- Hyper-parameter tuning
  - The process of choosing a set of optimal hyper-parameter values for a model
    - Give the best fitness and generalization
    - Avoid under-fitting or over-fitting

Hyper-parameter Tuning

Learning algorithm	Hyper-parameters
Linear regression	`fit_intercept`
Polynomial regression	`degree`, `interaction_only`, `include_bias`
Logistic regression	`penalty`, `tol`, `C`, `fit_intercept` …
K-nearest neighbors	`n_neighbors`, `weights`, `metric` …
Support vector machine	`C`, `kernel`, `tol`, …
Decision tree	`criterion`, `max_depth`, `min_samples_split`, `min_samples_leaf`, `max_leaf_node`, `min_impurity_decrease` …

Hyper-parameter Tuning

Learning algorithm	Hyper-parameters
Linear regression	`fit_intercept`
Polynomial regression	`degree`, `interaction_only`, `include_bias`
Logistic regression	**`penalty`, `tol`, `C`,** `fit_intercept` …
K-nearest neighbors	`n_neighbors`, `weights`, `metric` …
Support vector machine	**`C`,** `kernel`, **`tol`,** …
Decision tree	`criterion`, **`max_depth`, `min_samples_split`, `min_samples_leaf`, `max_leaf_node`, `min_impurity_decrease`** …

Some algorithms have hyper-parameters to penalize the complexity or perform early stopping to avoid over-fitting

Hyper-parameter Tuning

- Different hyper-parameters may interact with each other.
- Hyper-parameter tuning is actually the searching process to find the best combination of hyper-parameter values
- Two searching schemes
  - Grid Search
    - An exhaustive enumeration of all possible combinations

KNN hyper-parameter
K: 3 or 5
weight: uniform or distance

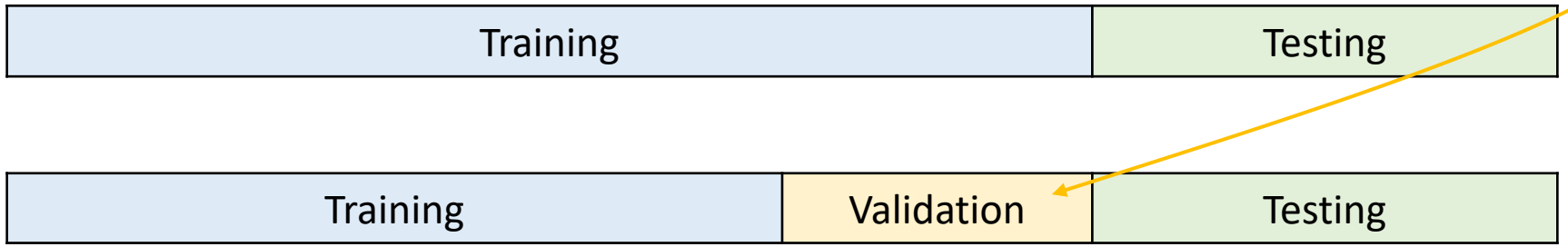
Hyper-parameter	Combination 1	Combination 1	Combination 1	Combination 1
K	3	3	5	5
weight	uniform	distance	uniform	distance

Hyper-parameter Tuning

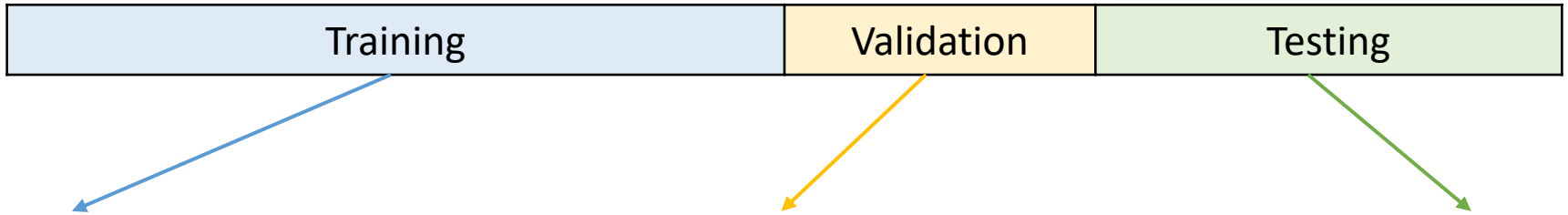
- Different hyper-parameters may interact with each other.
- Hyper-parameter tuning is actually the searching process to find the best combination of hyper-parameter values
- Two searching schemes
  - Grid Search
  - Random Search
    - Search among randomly generated combinations
    - How many combinations to search is defined by user
    - Less time consuming than grid search

Hyper-parameter Tuning

- How to decide which combination is the best one?
  - Train a model with this combination of hyper-parameter values
  - Calculate performance metric using
    - The entire dataset?
    - The training dataset?
    - The testing dataset?



Hyper-parameter Tuning



Train the model with the training dataset and a specific set of hyper-parameter values

Evaluate the model performance with the validation dataset

- Calculate the metric
- Ranking different combinations
- Select the best one

Perform final evaluation with the testing dataset

- Select the optimal hyper-parameters using the validation dataset
  - The choice may depend on a particular split of (training, validation, testing) sets.

Outline

- Decision Tree
- Hyper-parameter Tuning
- **Cross-validation**

Cross-validation

- A model validation technique for assessing how the trained model will generalize to unseen data.
- A resampling method that uses different portions of the dataset to train the model and assess the performance in different iterations
- K-fold cross-validation
  - The original training set is randomly partitioned into k equal sized subsets, often referred to as "folds”
  - In each iteration, a single fold is used as the validation set, the rest folds are used as the training set.

Cross-validation

- K-fold cross-validation

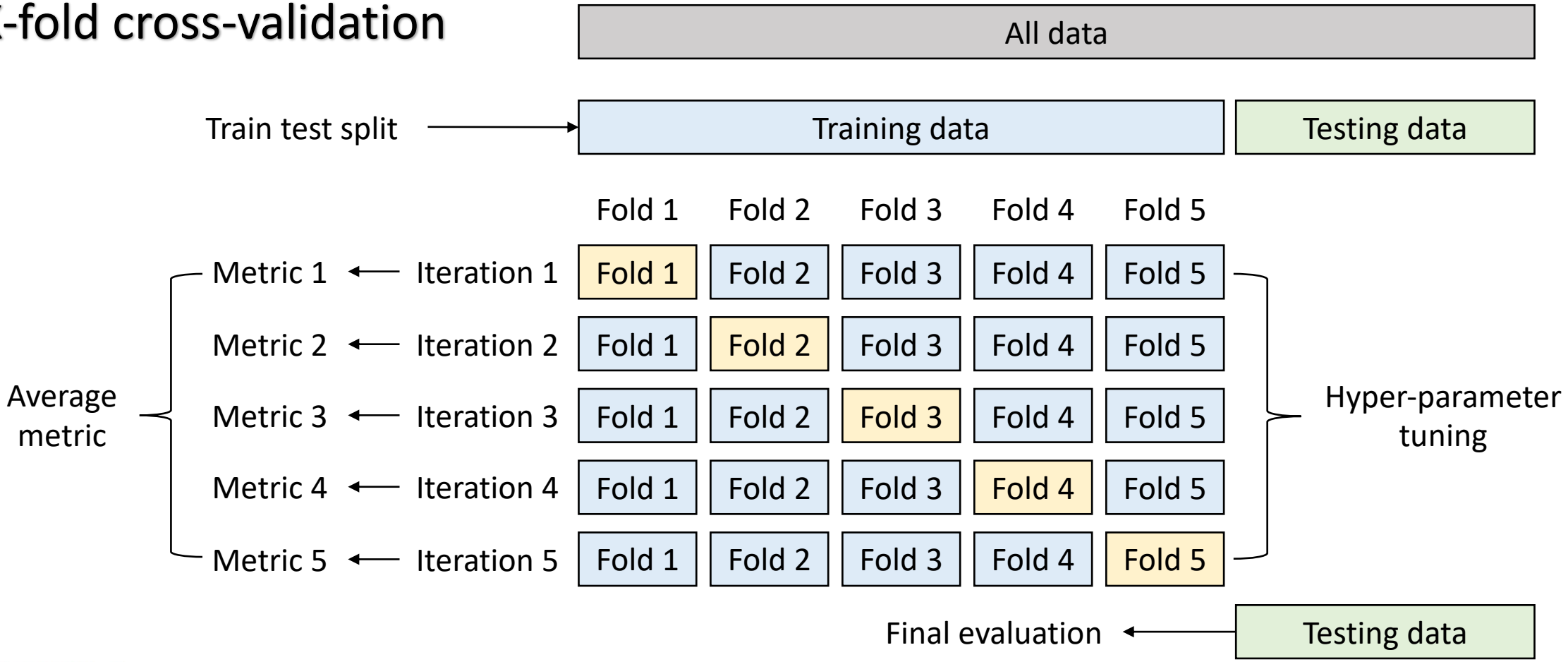


Image source: https://scikit-learn.org/stable/modules/cross_validation.html

Cross-validation

- K-fold cross-validation
- Leave-one-out cross-validation
  - The size of dataset is n, which is small
  - The extreme case of k-fold cross validation, where k = n
    - Each fold only contains one data point
    - In each iteration, leave one out for validation, use the rest for training

Cross-validation

- Best practice

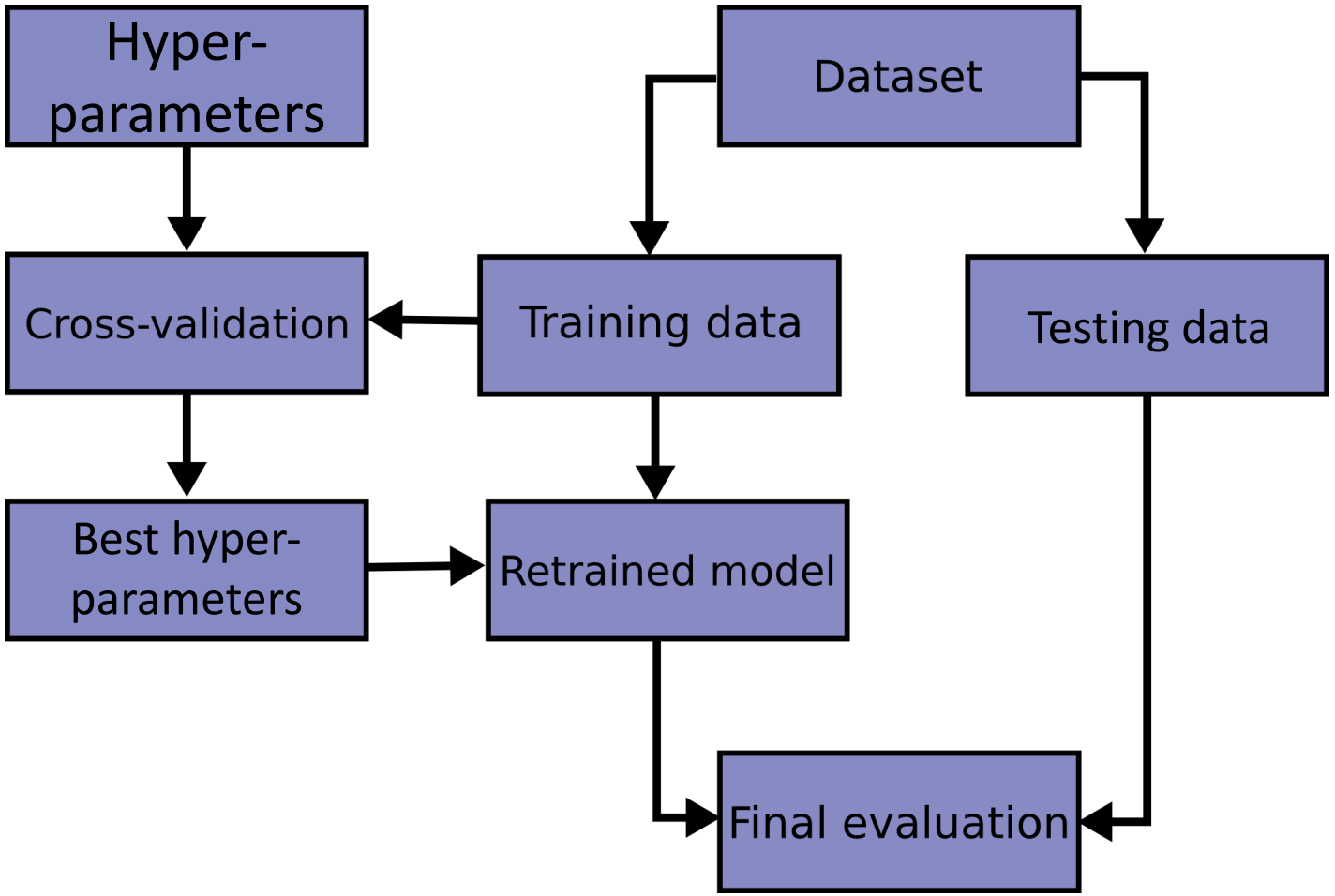


Image source: https://scikit-learn.org/stable/modules/cross_validation.html

Hands-on Exercise

- Exercise 06 Classification III