

Tree-Based Models & Ensemble Models

Qing-Yuan Jiang

The school of Computer Science and Engineering, NJUST

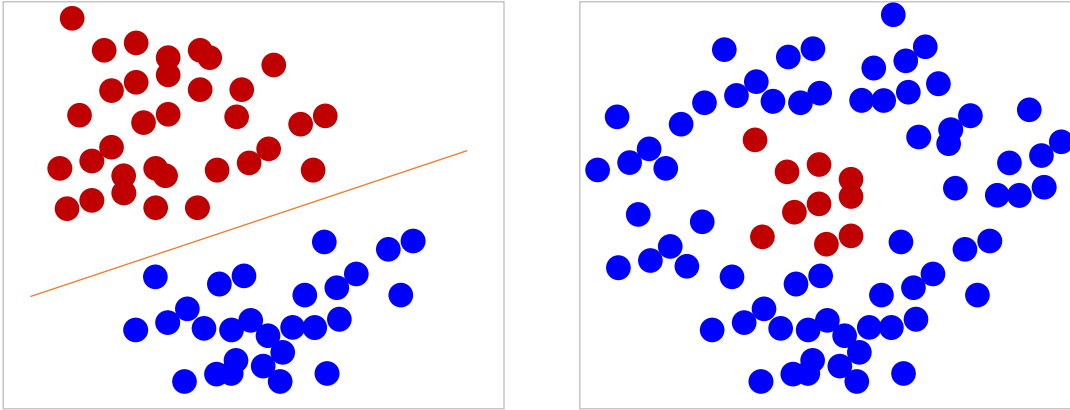
2026.10.08

Review of Previous Class

- What is the basic idea of a linear model?
- What do the weights and bias represent?
- What is the geometric interpretation of a linear model?
- How do we learn a linear regression model?
- Why do we use squared error for linear regression?
- How does logistic regression turn a linear score into a probability?
- Why do we use cross-entropy instead of squared error?
- How are linear regression and logistic regression connected?

Where Linear Models Struggle?

- Can a Linear Model Solve Every Classification Problem?

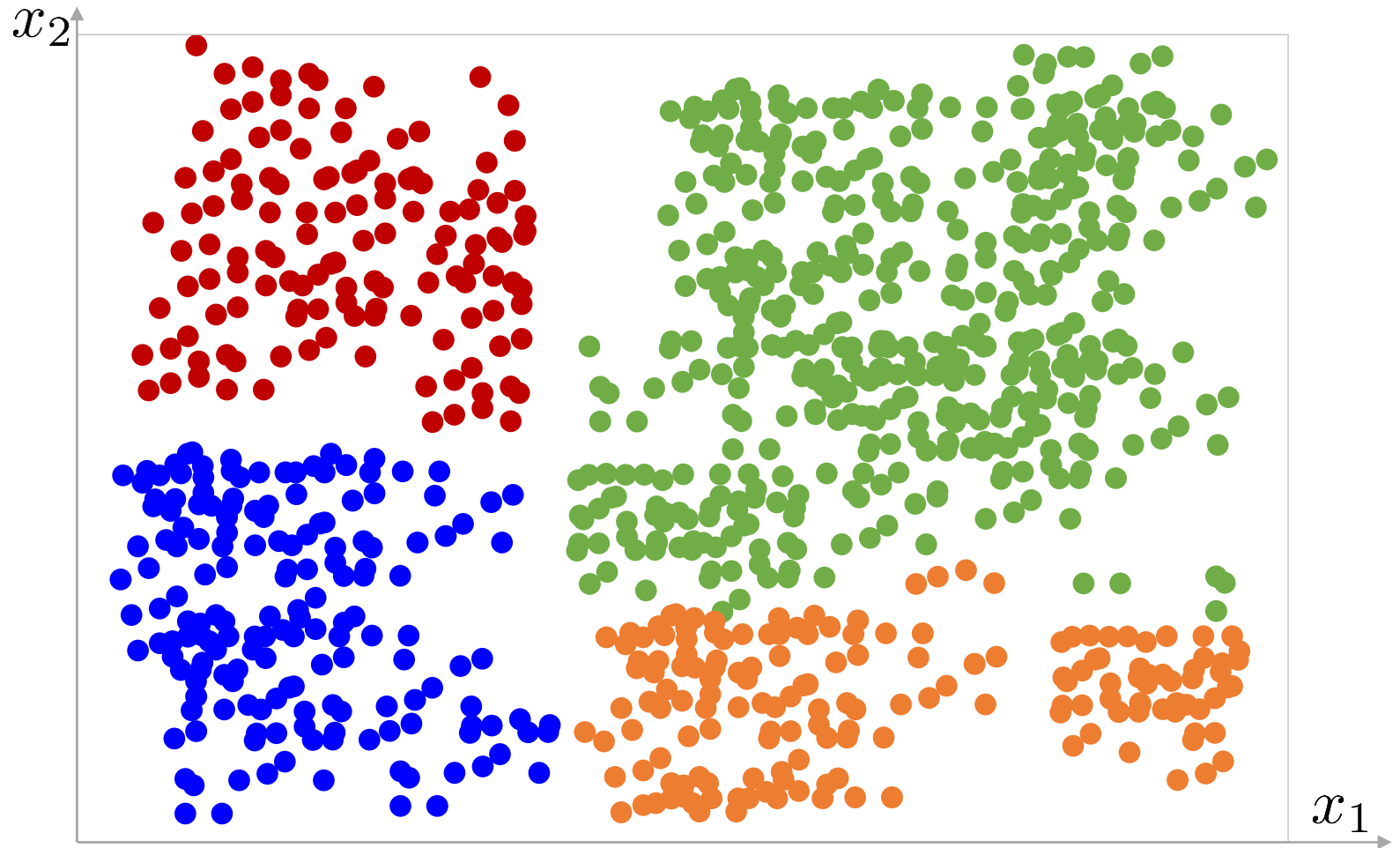


- What if the decision boundary is highly **nonlinear**?

We need a model that can adapt its structure to the data.

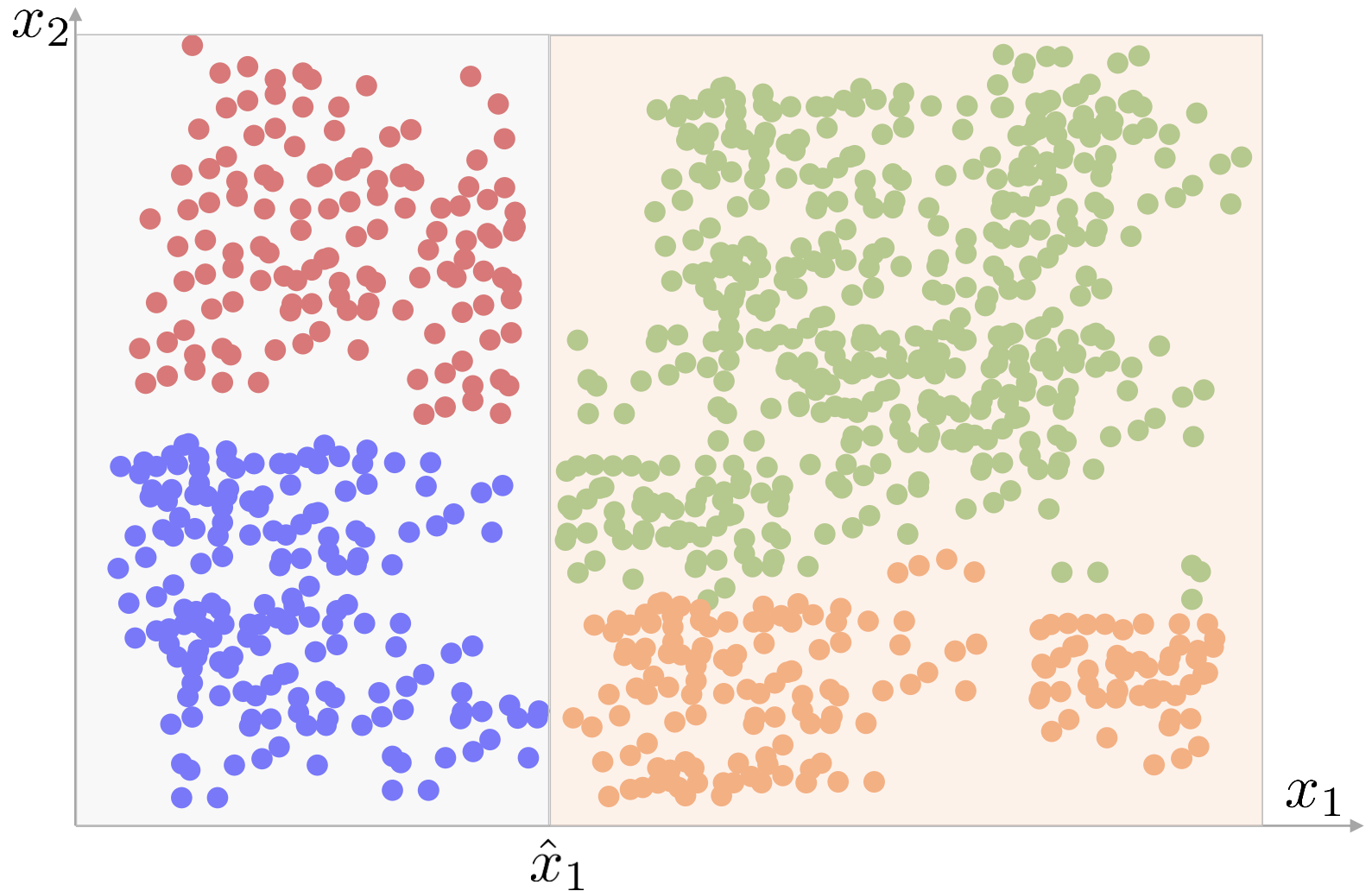
A Different Idea: Divide and Conquer

- How to build a classifier for this dataset?



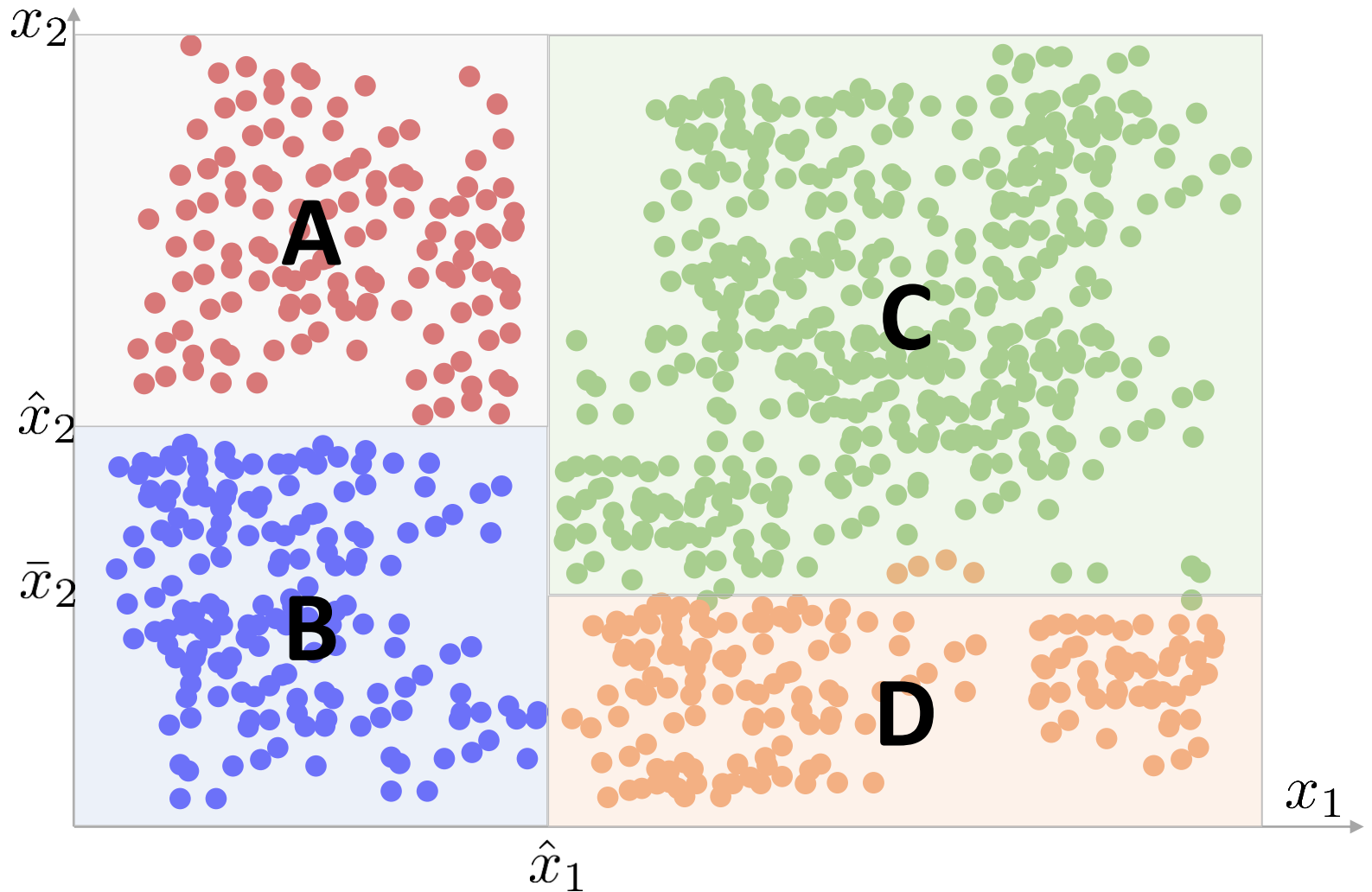
A Different Idea: Divide and Conquer

- How to build a classifier for this dataset?



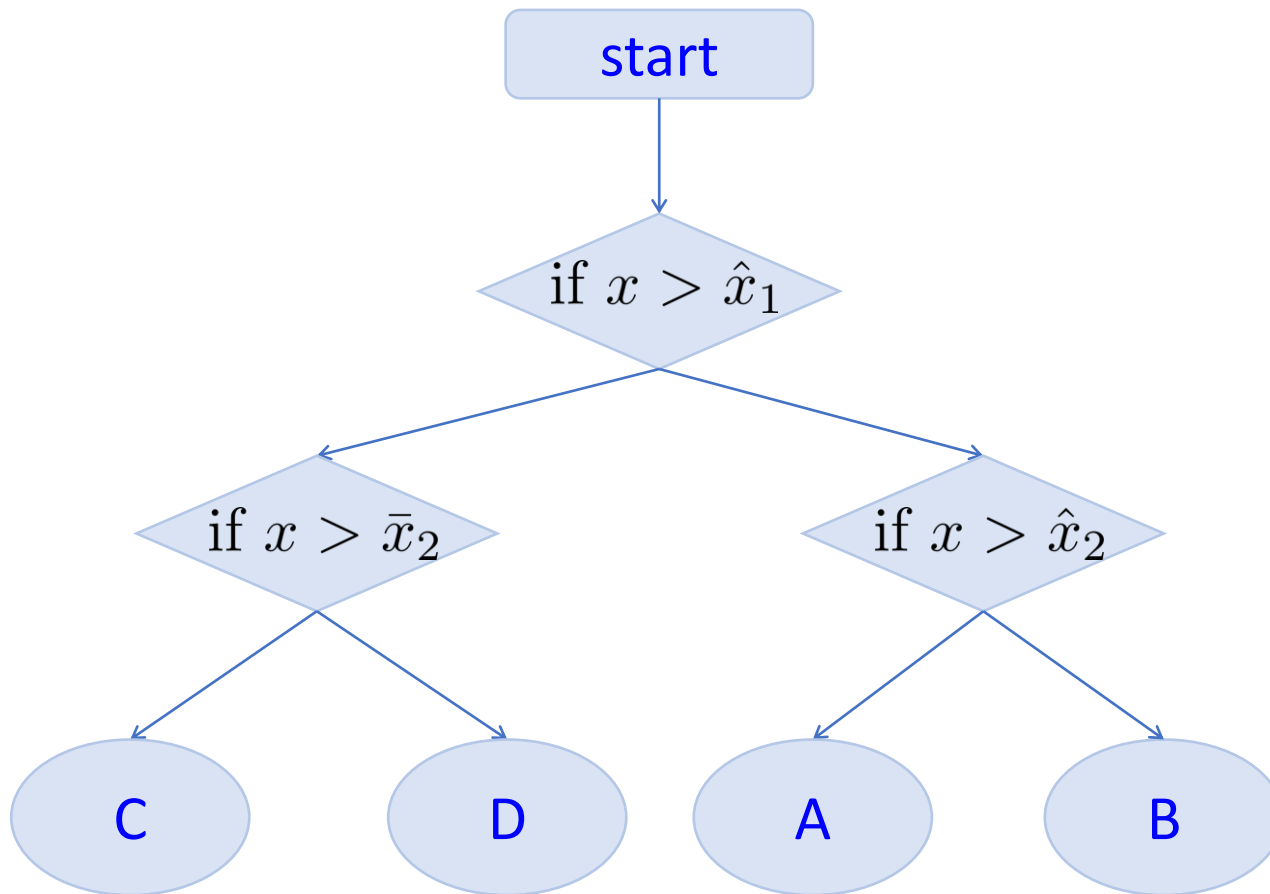
A Different Idea: Divide and Conquer

- How to build a classifier for this dataset?

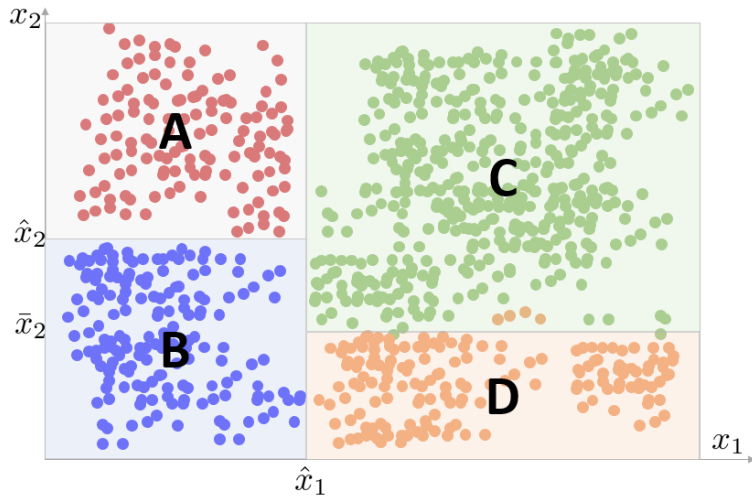


A Different Idea: Divide and Conquer

- How to implement this classifier?
 - Hint: this classifier contains $\{\hat{x}_1, \hat{x}_2, \bar{x}_2\}$.



From Regions to Predictions



- The two-dimensional feature space is partitioned into multiple **rectangular regions**.
 - Each region corresponds to a prediction.
-
- For classification task:
 - Each region $\rightarrow$ one class
 - For regression task:
 - Each region $\rightarrow$ one numerical value

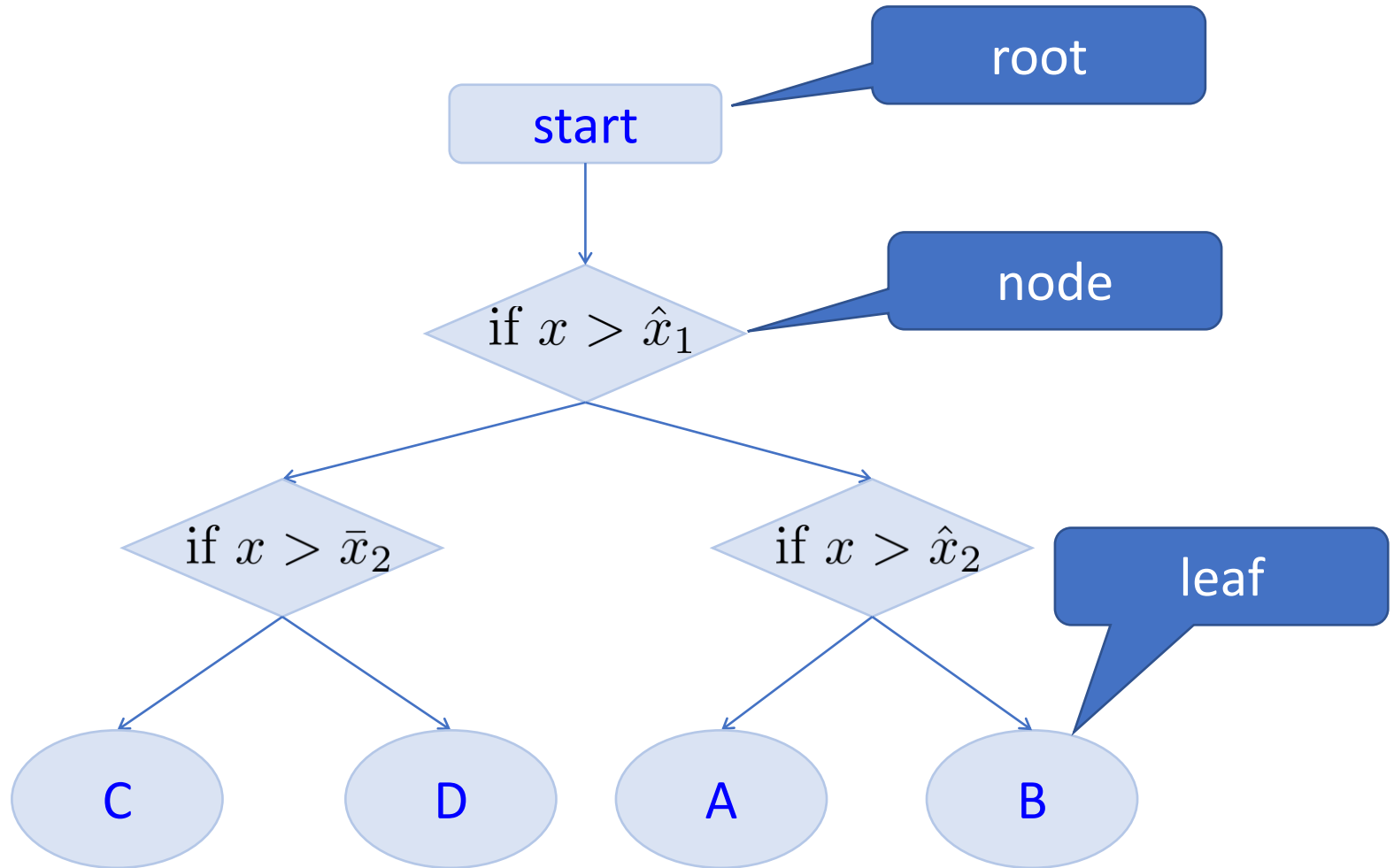
Why Trees?

- Tree-based models provide several attractive properties:
 - Nonlinear decision boundaries
 - Automatic feature selection
 - Little preprocessing
 - Interpretable decision rules
 - Applicable to classification and regression

Key idea: recursively partition the feature space.

Decision Tree

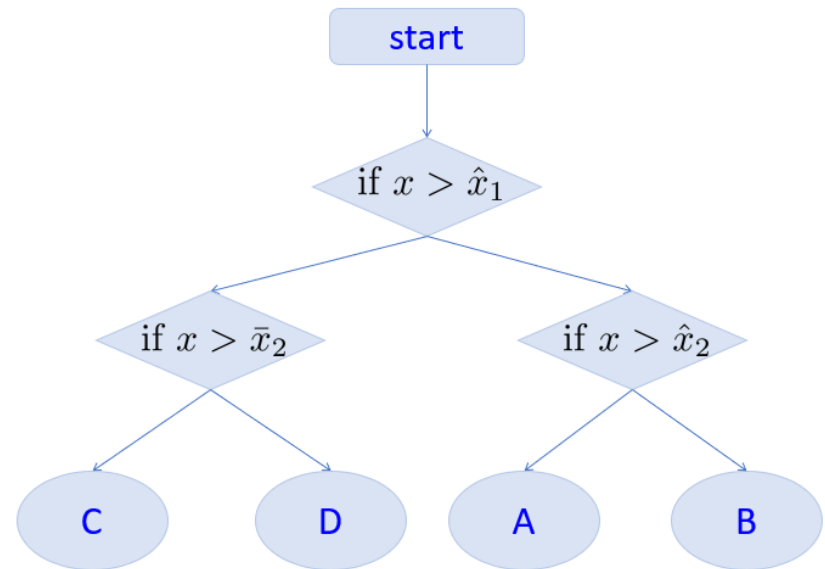
What is a decision tree?



From Tree to Decision Rules

- A Tree Represents a Set of **If-Then** Rules
- Example:

```
def decision_tree(tree, x):  
    if x[0] > tree.x1:  
        if x[1] > tree.x21:  
            return 'C'  
        else:  
            return 'D'  
    elif x[1] > tree.x22:  
        return 'A'  
    else:  
        return 'B'
```



- Each **root-to-leaf** path defines one rule.
- A decision tree is a **collection of local decision rules**.

Which Feature Should We Split On?

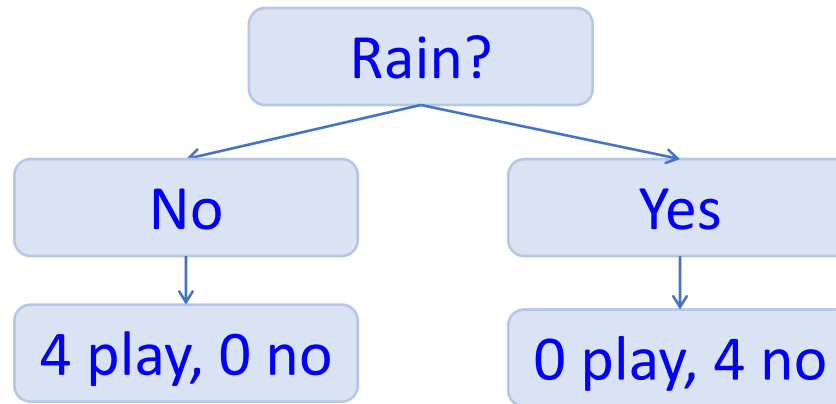
- **Task:** Predict whether to play tennis.
- Two candidate features: Rain and windy

No	Rain?	Windy?	Play Tennis?
1	No	No	Yes
2	No	Yes	Yes
3	Yes	No	No
4	Yes	Yes	No
5	No	No	Yes
6	No	Yes	Yes
7	Yes	No	No
8	Yes	Yes	No

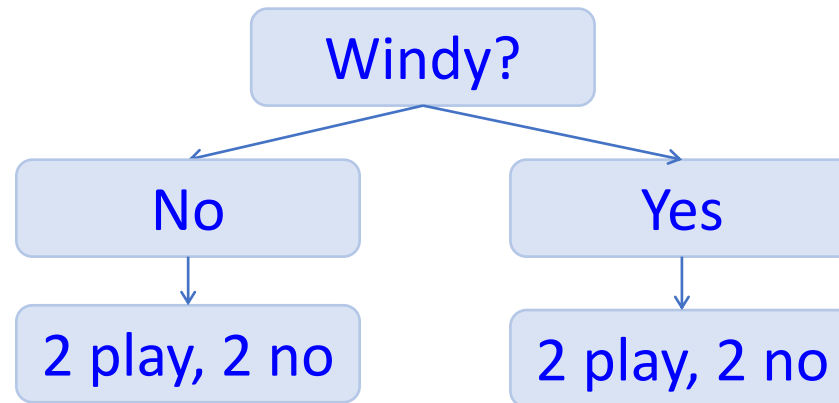
- Which feature should be used for **the first** split?

Which Feature Should We Split On?

- Split by Rain?



- Split by Windy?



- A good split produces **purier child nodes**.

Measuring Node Impurity

- Impurity measures how mixed the labels are within a node.

Feature	Class Distribution	Impurity
Rain	4 play, 0 no	Low
Windy	2 play, 2 no	High

- Goal: Find splits that reduce impurity.
- Strategy:
 - Entropy
 - Information gain
 - Gini impurity

Entropy

- Entropy:

$$H(S) = - \sum_{k=1}^K p_k \log p_k$$

- For binary classification:

$$H(S) = -p \log p - (1 - p) \log(1 - p)$$

- Interpretation:

- $H(S) = 0$: completely pure
- High entropy: highly mixed
- Maximum entropy: classes are evenly distributed

- Example:

- Rain: $H(S) = 0$
- Windy: $H(S) = -(0.5 \times \log 0.5 + 0.5 \times \log 0.5) = 1$

Information Gain

- Information gain:

$$IG(S) = H(S) - \sum_v \frac{|S_v|}{|S|} H(S_v)$$

- where:
 - S : parent node
 - S_v : child node after the split
- Interpretation:
 - Information gain measures how much uncertainty is reduced by a split.
- Choose the split with larger information gain.

Gini Impurity

- Gini impurity:

$$G(S) = 1 - \sum_{k=1}^K p_k^2$$

- For binary classification:

$$G(S) = 2p(1 - p)$$

- Interpretation:

- $G(S) = 0$: completely pure
- Large Gini: highly mixed
- Lower Gini: better split

- Example:

- Rain: $G(S) = 0$
- Windy: $G(S) = 1 - (0.5^2 + 0.5^2) = 0.5$

Building a Decision Tree

- Choosing the Best Split:
 - For each candidate split:
 - Partition the current node.
 - Measure the impurity of each child.
 - Compute the weighted impurity.
 - Select the best split.

- Conceptually:

$$(j^*, t^*) = \arg \min_{j, t} \ell_{\text{split}}(j, t)$$

- The best split produces **the largest reduction in impurity**.

Numerical Features Splitting

- For a numerical feature x_i :

$$x_i < t$$

- How to select t ?
- Typical procedure:
 - Sort feature values.
 - Consider candidate thresholds.
 - Evaluate each split.
 - Select the best threshold.

Categorical Features Splitting

- For a categorical feature x_i :

$$x_i \in \{red, blue, black\}$$

- Possible binary splits:
 - Red vs. {Blue, Black}
 - Blue vs. {Red, Black}
 - Black vs. {Red, Blue}
- The treatment of categorical variables depends on the tree implementation.

Recursive Partitioning

The same split procedure applies at every internal node

Step	Operation
1	Choose the best feature and split at the current node.
2	Partition the samples into child nodes.
3	Repeat independently within each child.
4	Assign predictions when a stopping condition is met.

Local decisions build a hierarchical prediction rule.

When Should We Stop?

Should we split until every leaf is pure?

- Maximum tree depth
- Minimum samples per leaf
- Minimum samples required for splitting
- Minimum impurity decrease
- No useful split remains

Tree growth must balance fit and complexity.

A Fully Grown Tree

A deep tree can fit training data extremely well

- Creates very small leaf regions
- Captures training samples in increasingly fine detail
- Can also capture noise and idiosyncrasies

Greater depth $\Rightarrow$ Lower training error

Does better training performance imply better generalization?

Overfitting in Decision Trees

Tree depth controls model complexity

Depth	Training behavior	Generalization
Too shallow	Underfits the data	High bias
Moderate	Captures useful structure	Often a better balance
Too deep	Fits noise and small regions	High variance

Training error falls, but validation error can rise.

Controlling Tree Complexity

Pre-pruning

Stop tree growth early.

Maximum depth

Minimum samples per leaf

Minimum impurity decrease

Post-pruning

Grow a large tree first.

Remove branches that add complexity without enough improvement in fit.

Select complexity using validation performance.

Classification vs. Regression Trees

	Classification tree	Regression tree
Leaf prediction	Most likely class	Mean target value in leaf
Split criterion	Gini impurity or entropy	Squared error
Target	Discrete class label	Numerical value

Classification

$$\hat{y} = \arg \max_k p_k$$

Regression

$$\hat{y} = \frac{1}{|R_m|} \sum_{i \in R_m} y_i$$

The structure is similar. The objective and prediction differ.

Strengths and Weaknesses of a Single Tree

Strengths

- Nonlinear modeling
- Easy to interpret
- Little preprocessing
- Automatic feature selection

Weaknesses

- High variance
- Sensitive to training data
- Can overfit
- Greedy construction

A small data change can produce a substantially different tree.

Can We Do Better Than One Tree?

Train several trees on different versions of the data

Learner	Training set	Prediction
Tree 1	Dataset 1	$f_1(x)$
Tree 2	Dataset 2	$f_2(x)$
Tree M	Dataset M	$f_M(x)$

Combine the predictions into one ensemble predictor.

Can multiple imperfect learners produce a better predictor?

What Is Ensemble Learning?

Combine multiple learners into a stronger predictor

$$f(x) = \mathcal{A}(f_1(x), f_2(x), \dots, f_M(x))$$

Classification

Majority vote

Regression

$$\hat{y} = \frac{1}{M} \sum_{m=1}^M f_m(x)$$

Aggregation combines the predictions of individual models.

Why Can Multiple Models Help?

Suppose each model has an individual prediction error

$$f_m(x) = f(x) + \varepsilon_m$$

$$\bar{f}(x) = f(x) + \frac{1}{M} \sum_{m=1}^M \varepsilon_m$$

If the errors are not perfectly correlated, averaging can reduce variance.

The ensemble needs both accuracy and diversity.

Bagging & Random Forest

Bagging: Bootstrap Aggregating

Basic procedure

- Draw multiple bootstrap datasets from the original data.
- Train one learner on each bootstrap dataset.
- Aggregate the predictions of all learners.

$$\{\mathcal{D}_1, \dots, \mathcal{D}_M\} \Rightarrow \{\mathcal{T}_1, \dots, \mathcal{T}_M\}$$

Learners train independently on different resampled datasets.

Bootstrap Sampling

Sample N examples with replacement from N examples

Original data	One possible bootstrap sample
1, 2, 3, 4, 5	2, 2, 4, 5, 5

- Some examples appear multiple times.
- Some examples are not selected.
- Each learner sees a different training set.

Bootstrap sampling creates diversity among learners.

Bagging for Classification

$$\hat{y} = \text{mode}\{\mathcal{T}_1(x), \mathcal{T}_2(x), \dots, \mathcal{T}_m(x), \}$$

Tree	1	2	3	4	5
Prediction	Cat	Dog	Cat	Cat	Dog

Final prediction: Cat

3 votes for Cat, 2 votes for Dog

Majority voting combines individual predictions.

Bagging for Regression

$$\hat{y} = \frac{1}{M} \sum_{m=1}^M \mathcal{T}_m(x)$$

Tree	1	2	3	4	5
Prediction	1.2	1.7	1.4	1.5	1.3

Final prediction: $(1.2 + 1.7 + 1.4 + 1.5 + 1.3) / 5 = 1.42$

Averaging smooths individual prediction fluctuations.

From Bagging to Random Forest

Bootstrap datasets alone may not diversify every split

- A strong feature may dominate the split selection.
- Many trees can repeatedly select the same feature.
- The trees then have correlated predictions.
- Highly correlated trees benefit less from averaging.

Add randomness to feature selection.

Random Forest

Bagging with random feature selection

- Draw a bootstrap sample for each tree.
- At each split, randomly select a subset of features.
- Find the best split among the selected features.
- Grow the tree, usually without pruning.
- Aggregate predictions across trees.

Randomize both training samples and candidate features.

Random Forest: A Complete Picture

Stage	Tree 1	Tree 2	Tree M
Sample data	Bootstrap D_1	Bootstrap D_2	Bootstrap D_m
At each split	Random features	Random features	Random features
Train	T_1	T_2	T_m
Aggregate	Vote or average across all trees		

Two sources of randomness:

Random samples and random candidate features

Diversity makes aggregation more effective.

Out-of-Bag Evaluation

Some examples are absent from each bootstrap sample

$$p(\text{not selected}) = \left(1 - \frac{1}{N}\right)^n$$

- These examples are out-of-bag (OOB) for that tree.
- For each example, aggregate only its OOB trees.
- Compare the OOB prediction with the true target.

OOB predictions estimate generalization without a separate split.

Decision Tree vs. Random Forest

Property	Decision tree	Random forest
Number of trees	One	Many
Variance	High	Lower
Interpretability	High	Lower
Training cost	Low	Higher
Prediction cost	Low	Higher
Overfitting	More sensitive	Usually less sensitive

Random forests reduce variance through diversity and aggregation.

Boosting

Bagging vs. Boosting

Bagging

- Train learners independently.
- Different bootstrap samples
- Aggregate by vote or average

Boosting

- Train learners sequentially.
- Each learner improves the current ensemble.
- Combine their contributions

Bagging averages learners. Boosting adds corrective learners.

The Basic Idea of Boosting

Start with a simple initial model $f_0(x)$

$$f_m(x) = f_{m-1}(x) + \alpha_m h_m(x)$$

- $h_m(x)$: the new weak learner
- α_m : the contribution of this learner
- Repeat to build an additive ensemble.

A strong predictor grows by adding weak learners.

AdaBoost: Focus on Difficult Samples

$$\text{Initially: } w_i = \frac{1}{N}$$

- Train a weak learner using the current sample weights.
- Identify the misclassified examples.
- Increase their weights relative to correct examples.
- Train the next learner using the updated weights.
- Combine learners using their learned contributions.

Later learners focus more on previously misclassified examples.

AdaBoost Weight Update

Binary labels: $y_i, h_m(x_i) \in \{-1, +1\}$

$$\varepsilon_m = \sum_i w_i \mathcal{I}[y_i \neq h_m(x_i)]$$

$$\alpha_m = \frac{1}{2} \log \left(\frac{1 - \varepsilon_m}{\varepsilon_m} \right)$$

$$w_i \leftarrow w_i \exp(-\alpha_m y_i h_m(x_i))$$

then normalize

With $\varepsilon_m < 0.5$, misclassified examples gain relative weight.
Correctly classified examples lose relative weight.

Final prediction: $\text{sign} \left(\sum_m \alpha_m h_m(x) \right)$

Gradient Boosting

Fit a new learner to residuals or negative gradients

$$r_{im} = -\frac{\partial \ell(y_i, f(x_i))}{\partial f(x_i)}$$

Evaluate the derivative at the current model f_{m-1} .

Fit h_m to the targets r_{im} .

$$f_m(x) = f_{m-1}(x) + \eta h_m(x)$$

Each learner moves the ensemble toward lower loss.

Gradient Boosting as Optimization

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta} \ell$$

Gradient descent	Gradient boosting
Updates parameters	Updates a prediction function
Parameter space	Function space
Uses a gradient direction	Fits a learner to negative gradients
Optimizes one model	Adds learners stage by stage

Gradient boosting performs stage-wise optimization in function space.

A Unified View

Bagging, Random Forest, and Boosting

Method	Diversity / correction	Training	Main effect
Bagging	Bootstrap samples	Independent	Reduce variance
Random forest	Bootstrap + random features	Independent	Reduce variance
AdaBoost	Adaptive sample weights	Sequential	Focus on difficult samples
Gradient boosting	Residuals / negative gradients	Sequential	Reduce loss iteratively

The methods differ in how they create and combine learners.

Bagging vs. Boosting: A Unified View

Bagging

Independent learners

Diverse training samples

Average or vote

Main goal: reduce variance

Boosting

Sequential learners

Correct current errors

Accumulate contributions

Main goal: iterative improvement

Both combine multiple learners into one predictor.

Modern Tree Ensemble Models

Independent aggregation

Decision trees

Bagging

Random forest

Sequential improvement

Decision trees

Boosting

AdaBoost / Gradient boosting

XGBoost / LightGBM / CatBoost

Partitioning, diversity, aggregation, and iterative optimization.

Summary

Topic	Core idea
Decision trees	Recursively partition the feature space.
Tree complexity	Deeper trees fit more detail but can overfit.
Bagging	Train independently on bootstrap samples and aggregate.
Random forest	Add random feature subsets at each split.
Boosting	Add learners sequentially to improve the ensemble.

Combining decision structures can improve generalization.

Thanks