



Ahmed Ibrahim

ECE 9039/9309 MACHINE LEARNING

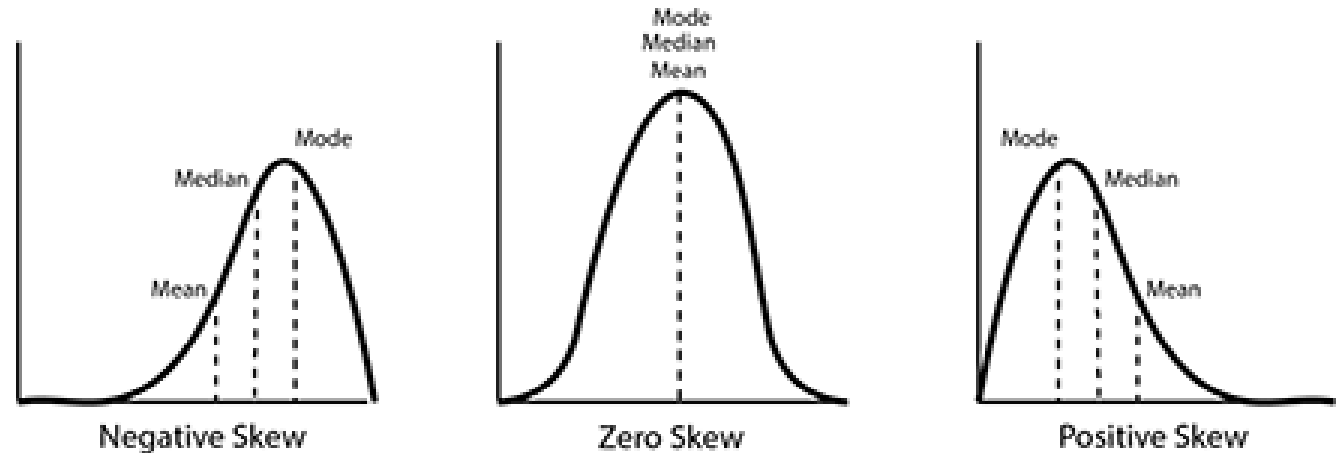
Last Lecture

- Probabilistic Classifiers
 - Naïve Bayes Classifier
- Instance-based classifiers
 - K-Nearest Neighbors (K-NN)
- More Classifiers: SVMs
- Clustering
 - K-means & K-means++
 - Gaussian Mixture Model (GMM)

Outlines

- Admin:
 - Resources on OWL
 - Assignment – Logarithm Transformation
- Cross-validation
- Outliers Detection
- Decision Trees
 - Regression Tree
 - Function Approximation
- Bagging
- Random Forest

Logarithm Transformation



- Many real-world datasets have a skewed distribution. Logarithm transformation can help normalize such distributions.
- By transforming skewed data into a more normal distribution, it is easier to interpret relationships between variables.
- To apply a logarithm transformation to a feature, you replace each value x in the feature with its logarithm, $\log(x)$.

Anomaly Detection

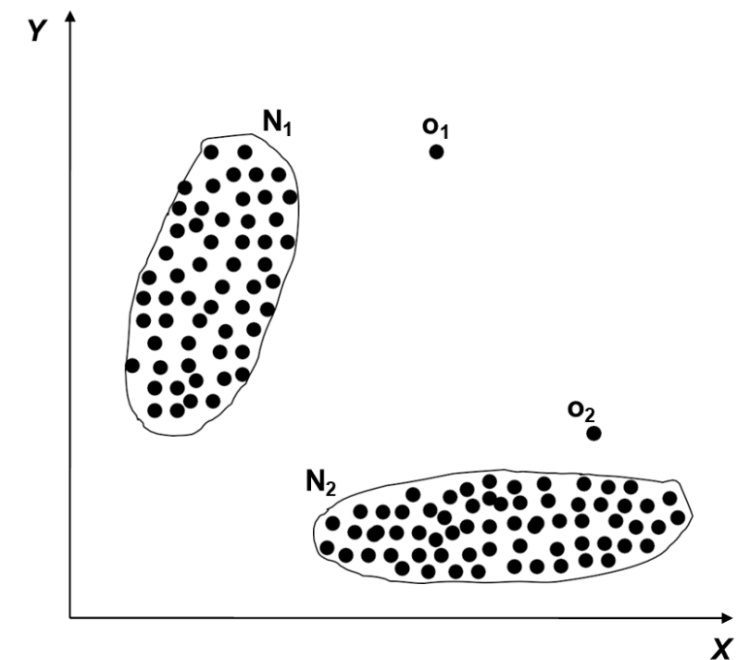
A thick, hand-drawn style orange line that underlines the title "Anomaly Detection". It starts under the first letter 'A' and ends under the last letter 'n'.

What are Anomalies?

- Anomaly is a pattern in the data that does not conform to the expected behavior. Also referred to as outliers, exceptions, peculiarities, etc.
- Anomalies translate to significant (often critical) real-life entities
 - Cyber attack
 - Defective products in manufacturing
- Anomaly Detection – Uncovering instances that deviate strongly from the norm

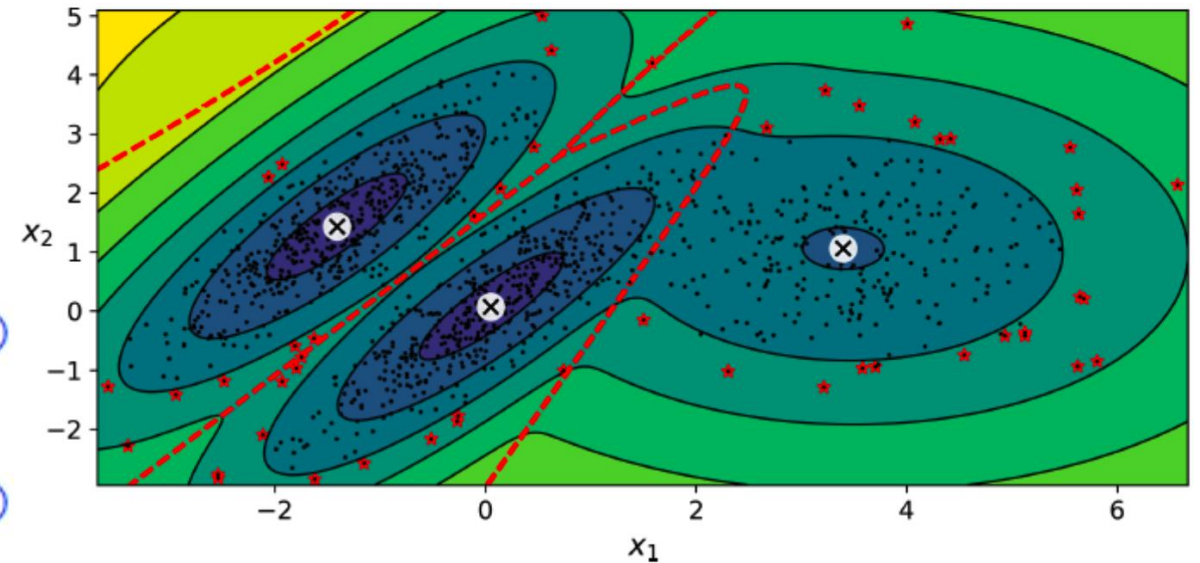
Common Outlier Detection Techniques

- **Clustering-Based Methods** – Utilize clustering algorithms to find outliers. Data points that do not belong to any cluster or are far from cluster centroids are considered outliers (e.g., **K-means**).
- **Classification-Based Methods** – Treat outlier detection as a classification problem (e.g., **k-nearest neighbours - KNN**).
 - KNN: If this distance is significantly larger than the average distance of the majority of points in the dataset, the data point is considered an anomaly.
- **Density-Based Methods** – Detect outliers by evaluating the density of regions in the data space. Unlike their neighbours, points in low-density areas are outliers (e.g., **Gaussian Mixture Models - GMM**).
 - GMM: Any instance in a low-density region can be considered an anomaly. You must define what density threshold you want to use.



Anomaly Detection Algorithm Using GMM

```
import numpy as np
from sklearn.datasets import make_moons
from sklearn.mixture import GaussianMixture
import matplotlib as mpl
import matplotlib.pyplot as plt
X, y = make_moons(n_samples=1000, noise=0.05)
gm = GaussianMixture(n_components=3, n_init=10)
gm.fit(X)
densities = gm.score_samples(X)
density_threshold = np.percentile(densities, 4)
anomalies = X[densities < density_threshold]
```



Source: [HML book]

Choosing an Algorithm

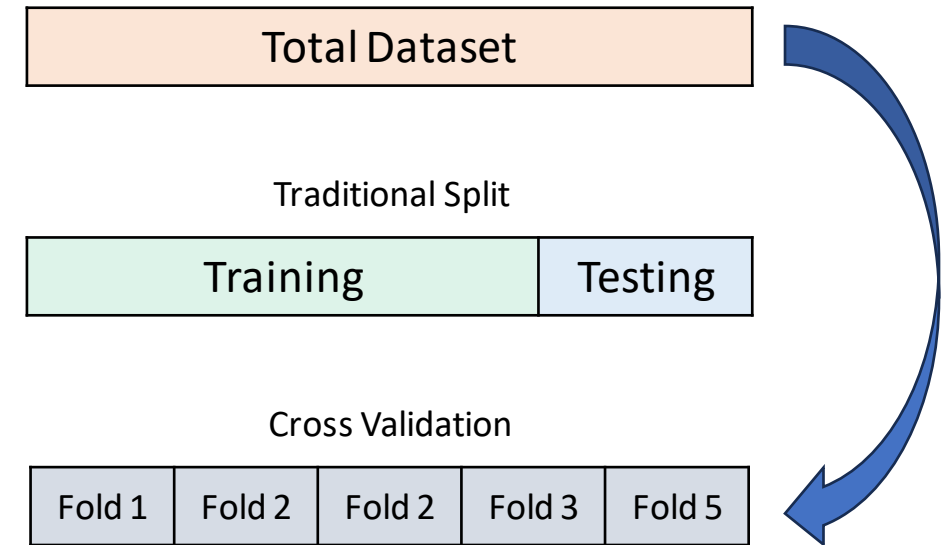
- The choice of algorithm often depends on the dataset's nature and the task's specific requirements.
- Some algorithms work better for high-dimensional data, while others are better suited for time-series data like LSTM (Long Short-Term Memory) Networks.
- It's also common to use a **combination** of these methods to improve the robustness and accuracy of anomaly detection.
- When applying anomaly detection algorithms, it's essential to clearly understand what constitutes normal behavior in the data to accurately identify deviations.

Cross Validation

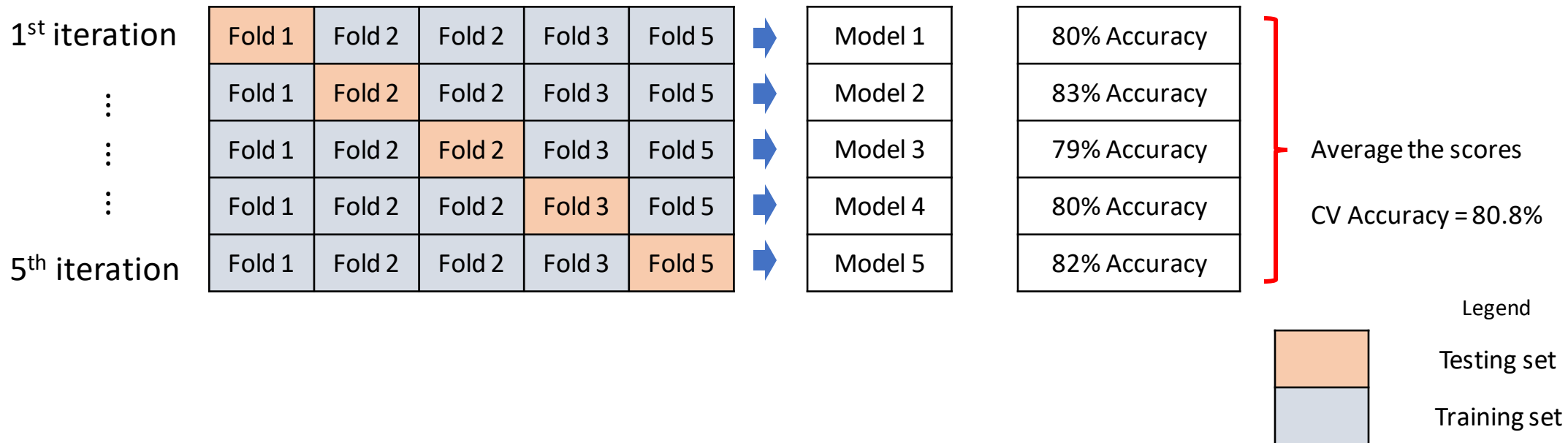
A thick, hand-drawn style orange line that underlines the title "Cross Validation".

Cross Validation (CV)

- Cross-validation is a widely used machine learning technique for assessing how accurate a predictive model will perform in practice.
- Cross-validation is a method where a specific subset (or fold) of the dataset is set aside and not used in the model's training. This subset is later utilized to evaluate the model's performance before completion.
- Goal:
 - Ensuring unbiased performance evaluation.
 - It helps in comparing the performance of different models.
 - Fine-tuning of model parameters.
 - Highlighting inconsistent performance across folds (Data Issues).



k -Fold Cross Validation



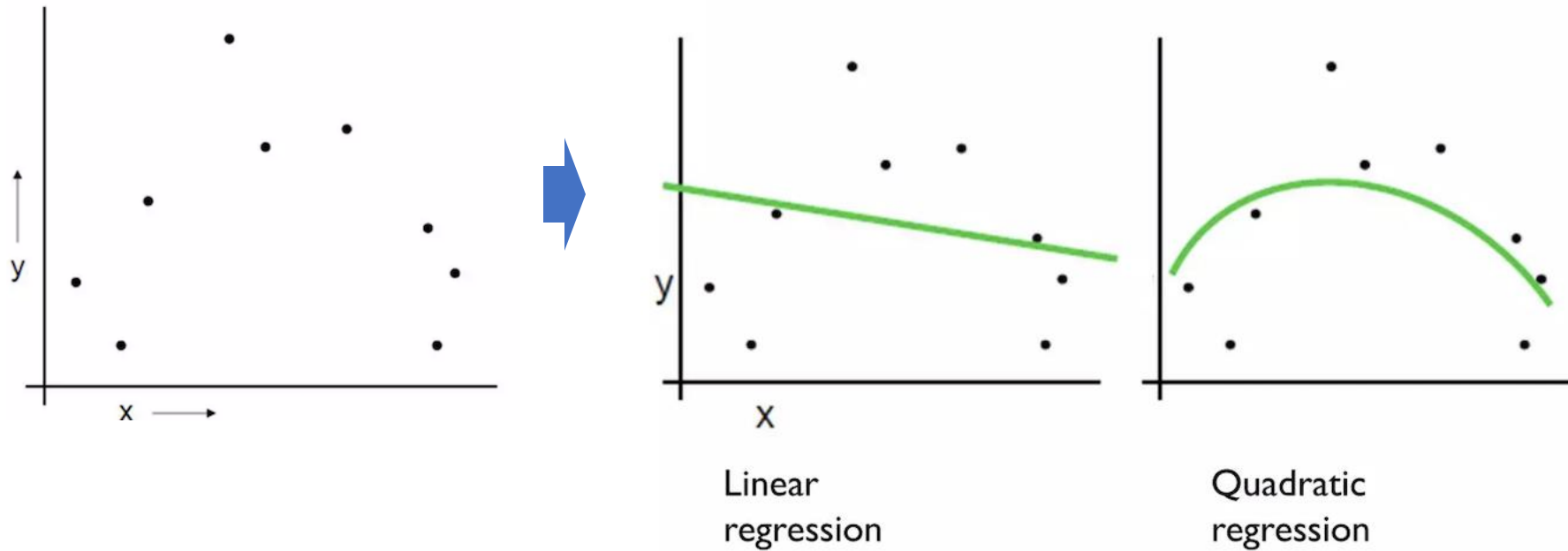
- K -fold cross-validation is considered a benchmark for validating model performance, offering a more comprehensive evaluation than the basic train/test split method.

k -Fold Cross Validation Steps

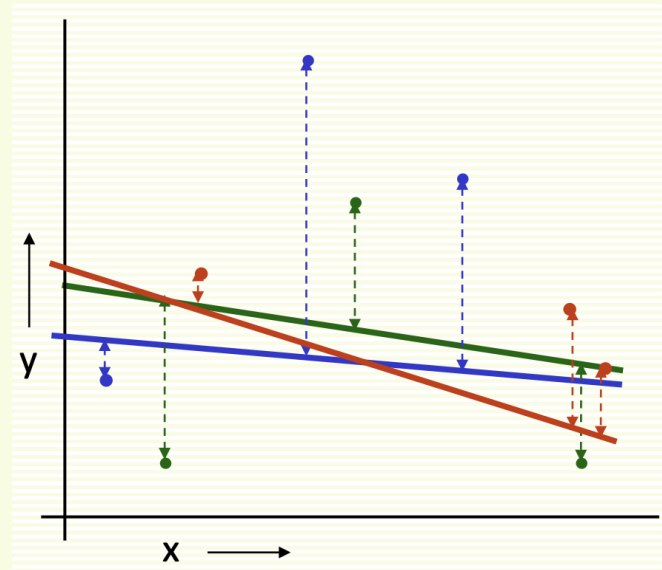
- Step 1 – Randomly split your dataset into ' k ' segments, or "folds," to ensure each is representative of the whole.
- Step 2 – Use ' $k - 1$ ' folds for model training; reserve the remaining fold for testing to gauge model accuracy.
- Step 3 – Record the model's performance (e.g., accuracy for classification, MSE for regression) for each test fold.
- Step 4 – Rotate the testing fold until each fold has been used for validation **exactly once**.
- Step 5 – Calculate the average of the ' k ' recorded performance metrics. This average represents your model's cross-validation error or accuracy, providing a robust **estimate of its performance on unseen data**.

Example – 3Fold CV

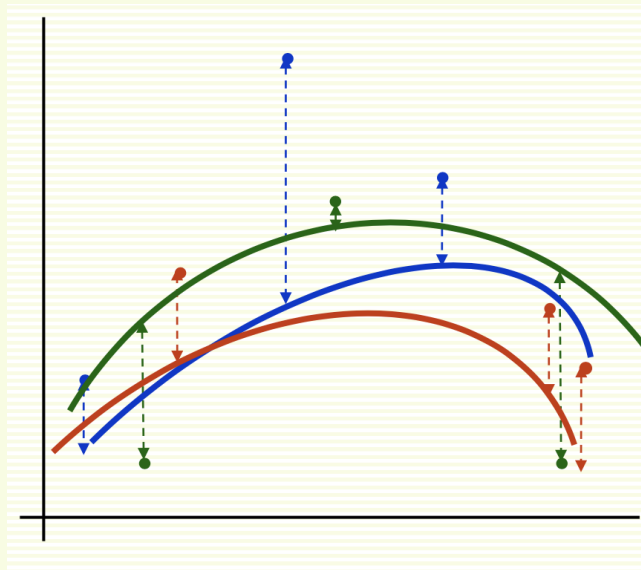
Which is the best?



Example – 3Fold CV (cont.)



Linear Regression
 $\text{MSE}_{3\text{-Fold}} \Rightarrow 2.05$



Quadratic Regression
 $\text{MSE}_{3\text{-Fold}} \Rightarrow 1.11$

Leave-One-Out Cross-Validation (LOOCV)

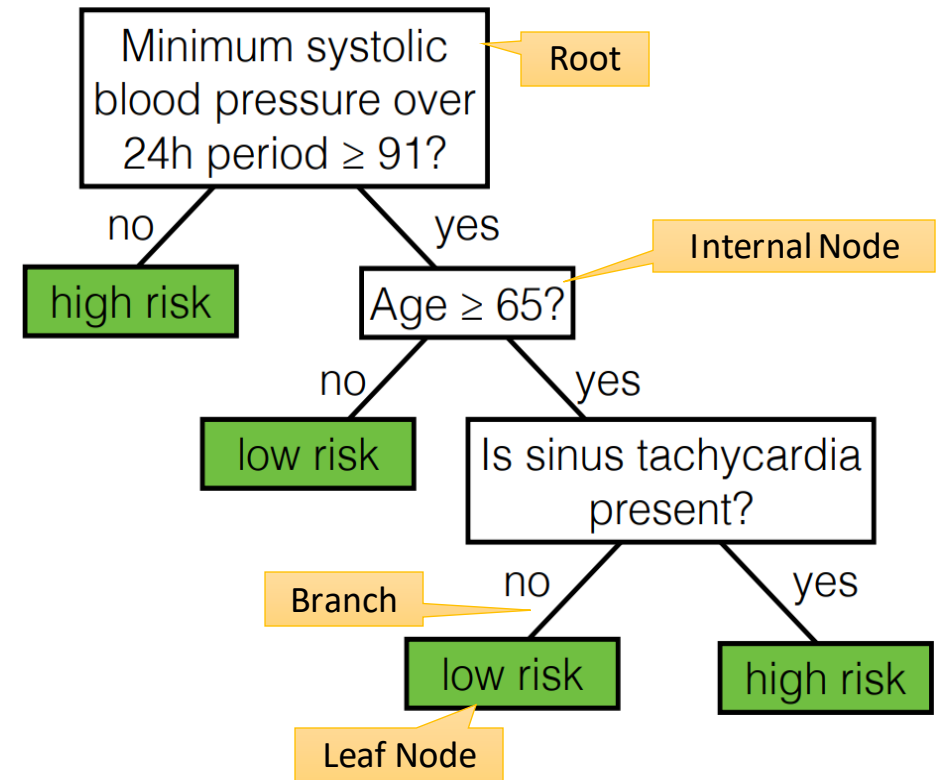
- Leave-One-Out Cross-Validation is a **specialized version** of k -fold cross-validation used for model evaluation.
- In LOOCV, the dataset is divided such that each fold contains **exactly one data point** as the test set and the remainder of the dataset as the training set.
- This process is **repeated** for each observation in the dataset, so each one is used once as the test set.
- PROS:
 - LOOCV allows an **in-depth analysis** of the model's performances across the entire dataset. This can help identify specific data points that are **difficult to predict**.
 - Also, it can help assess the stability and robustness of the model.
- CONS – Generally, LOOCV is considered a highly computationally demanding method.

Decision Trees

A thick, hand-drawn style orange line underlining the title "Decision Trees".

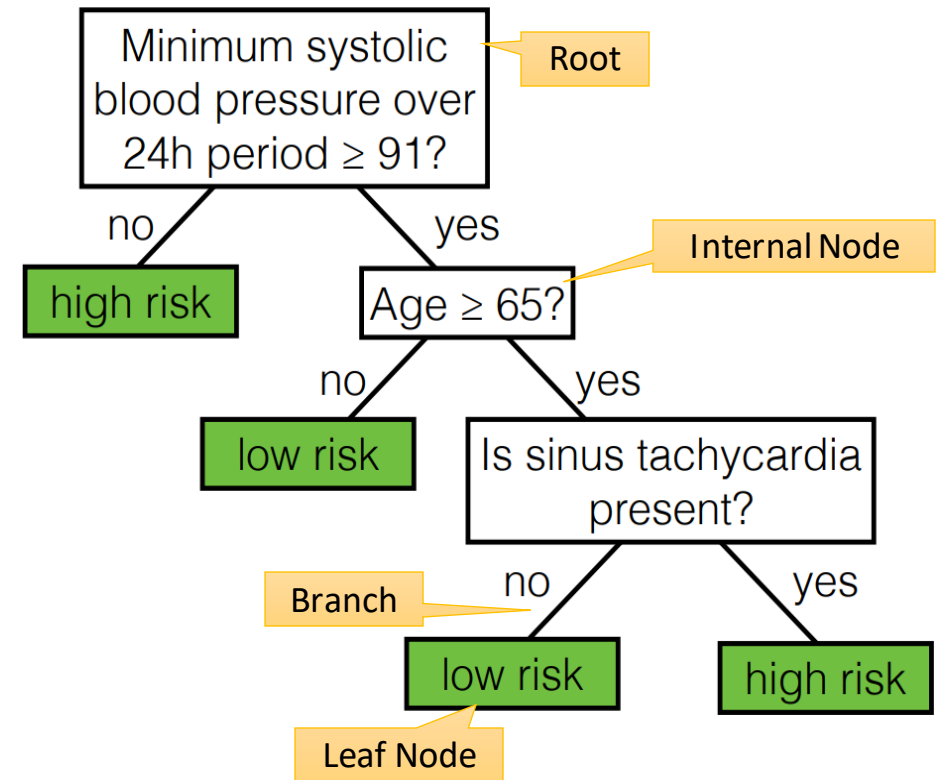
Decision Tree

- Decision trees (DTs) are a popular ML technique used for both classification (when classifying things into **categories**) and regression (when predicting **numerical values**) problems.
- DT consists of terminal nodes/leaves (arrows pointing to them) and Internal nodes/branches (arrows pointing to and away from them).
- DT classifies through a sequence of questions; the next question asked depends on the answer to the current question.



Decision Tree (cont.)

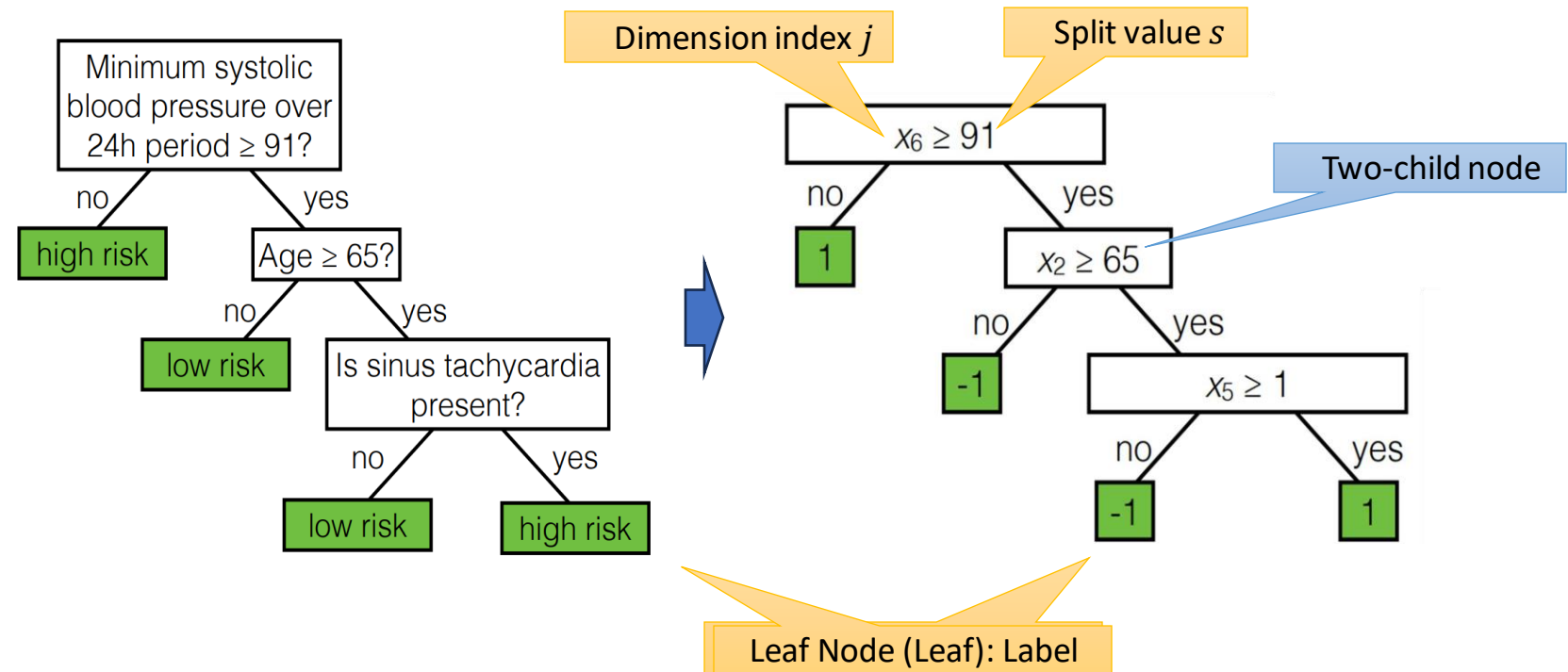
- This approach is particularly useful for non-metric data; questions can be asked in a “yes-no” or “true-false” style.
- The benefit of decision trees:
 - **Interpretability**: a tree can be expressed as a logical expression
 - **Rapid classification**: a sequence of simple queries
 - Good for tabular datasets



Decision Tree (cont.)

- Features:

- x_1 : date
- x_2 : age
- x_3 : height
- x_4 : weight
- x_5 : sinus tachycardia?
- x_6 : min systolic bp, 24h
- x_7 : latest diastolic bp



Dimension index j

Split value s

$x_6 \geq 91$

no

yes

1

no

yes

-1

$x_2 \geq 65$

no

yes

-1

yes

1

$x_5 \geq 1$

no

yes

-1

1

Two-child node

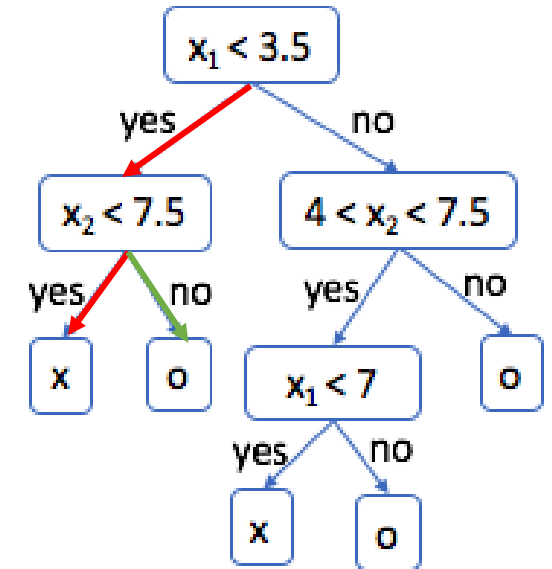
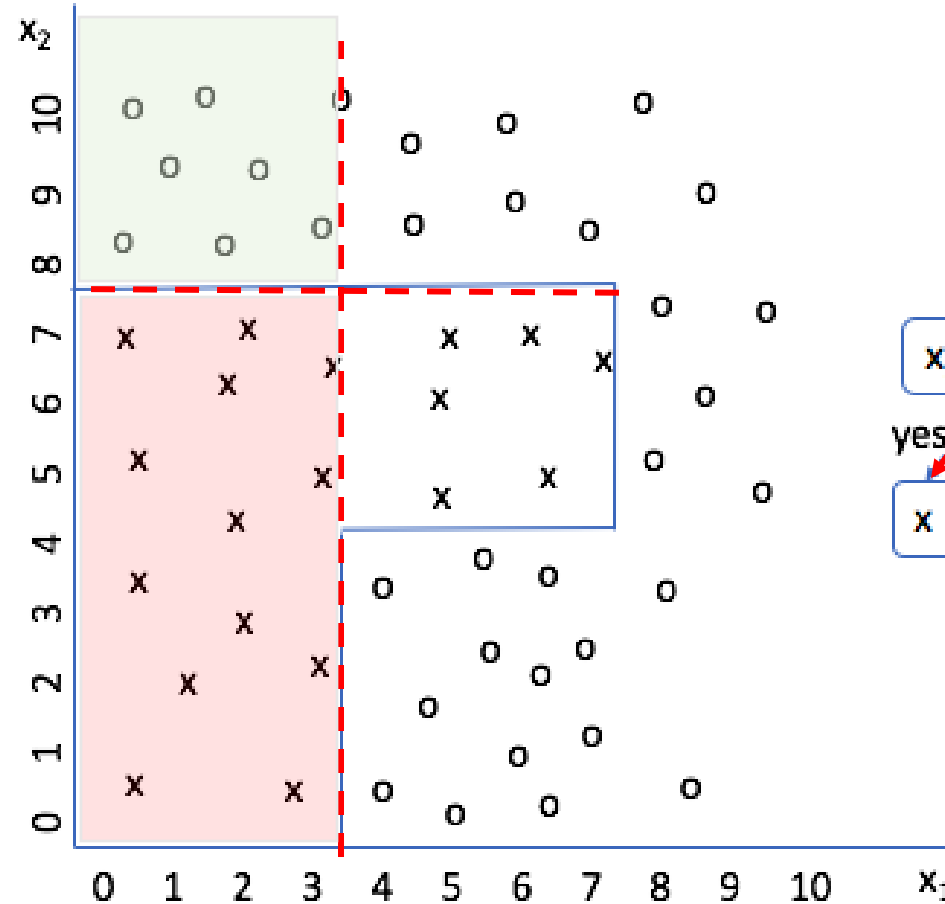
Leaf Node (Leaf): Label

How to Grow A Tree?

- Given a set D of labeled training samples with a set of features.
- A decision tree progressively splits the training set into smaller and smaller subsets
 - **Pure node** – all the samples at that node have the same class label; no need to further split a pure node.
 - **Recursive tree-growing process** – Given data at a node, decide the node as a leaf node or find another feature to split the node.

Partitioning Feature Space

- The illustrated decision tree partitions the feature space into regions.



How to Build a DT?

- Goal: Predict whether an applicant is accepted or rejected based on the following features:
 - ❓ Feature# 1: Test Score (numerical, range from 0 to 100)
 - ❓ Feature# 2: Experience (years, numerical, from 0 to 20)
 - ❓ Classification **Labels**: Accepted (1) or Rejected (0)
- When building decision trees, splits are assessed using a *Gini Impurity* (GI) criterion.
- The GI can be defined for a dataset with
- J classes as:
$$\text{Gini Impurity (GI)} = 1 - \sum_{j=1}^J p_j^2$$
- Where p_j is the proportion of items labeled with class j in the set.

Applicant	Test Score	Experience (in years)	Accepted
A	85	15	1
B	90	7	1
C	78	10	0
D	75	5	0
E	60	2	0
F	95	18	1

G|

Characteristics

- **Range** – Gini impurity ranges from $[0:0.5]$ in a binary classification
 - 0 being pure, meaning all elements belong to a single class, and
 - 0.5, indicating the data is evenly split between two classes.
- **Purpose** – It measures the variability of a set.


Step 1: Calculate Gini Impurity for Test Score Split

- Split on Test Score (x_1):
 - Right Split (Scores > 80): **A, B, F** (All accepted)
 - $GI = 1 - \left(\left(\frac{3}{3} \right)^2 + \left(\frac{0}{3} \right)^2 \right) = 0$
 - Left Split (Scores ≤ 80): **C, D, E** (3 rejected)
 - $GI = 1 - \left(\left(\frac{0}{3} \right)^2 + \left(\frac{3}{3} \right)^2 \right) = 0$
- Weighted Average Gini for the split of Test Score (x_1):

$$\left(\frac{3}{6} \right) \times 0 + \left(\frac{3}{6} \right) \times 0 = 0$$

Total items


proportion of items
labeled with class



Applicant	Test Score x_1	Experience (in years) x_2	Accepted
A	85	15	1
B	90	7	1
C	78	10	0
D	75	5	0
E	60	2	0
F	95	18	1

Step 2: Calculate Gini Impurity for Experience Split

- Split on Experience (x_2):
 - Right Split (Experience > 10 years): A, F (All accepted)
 - $GI = 1 - \left(\left(\frac{2}{2} \right)^2 + \left(\frac{0}{2} \right)^2 \right) = 0$
 - Left Split (Experience ≤ 10 years): B, C, D, E (1 accepted, 3 rejected)
 - $GI = 1 - \left(\left(\frac{3}{4} \right)^2 + \left(\frac{1}{4} \right)^2 \right) = 0.375$
- Weighted Average Gini for Experience (x_2):
 $\left(\frac{4}{6} \right) \times 0.375 + \left(\frac{2}{6} \right) \times 0 = 0.25$

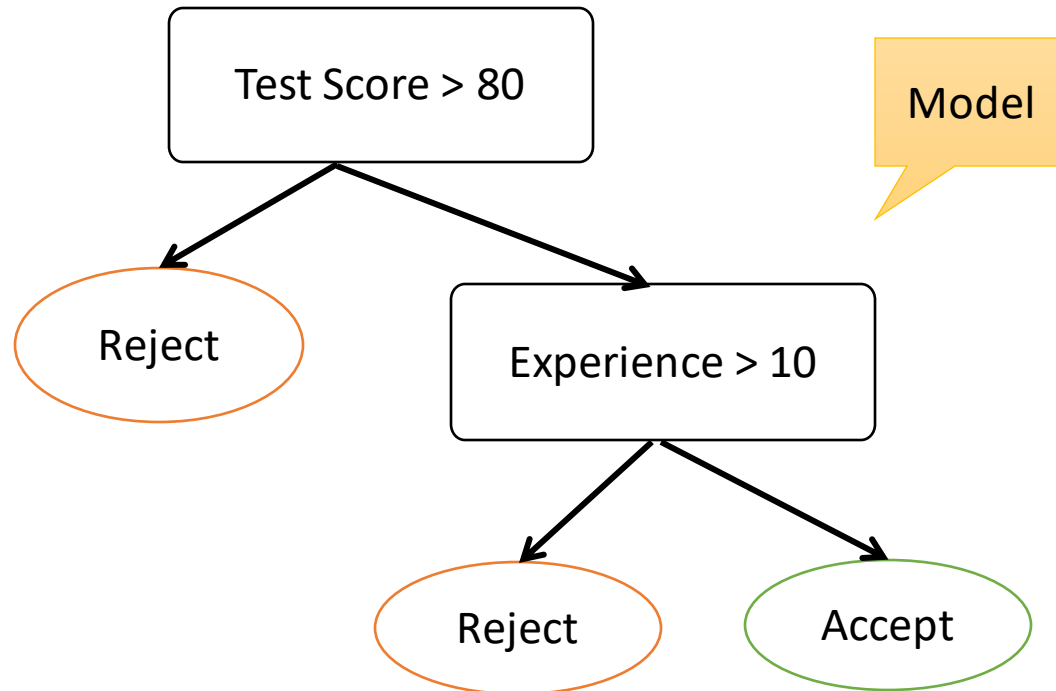


Applicant	Test Score x_1	Experience (in years) x_2	Accepted
A	85	15	1
B	90	7	1
C	78	10	0
D	75	5	0
E	60	2	0
F	95	18	1

Step 3: Compare the Weighted Average GIs

x_1 : Weighted Average GI is Zero

x_2 : Weighted Average GI is 0.25

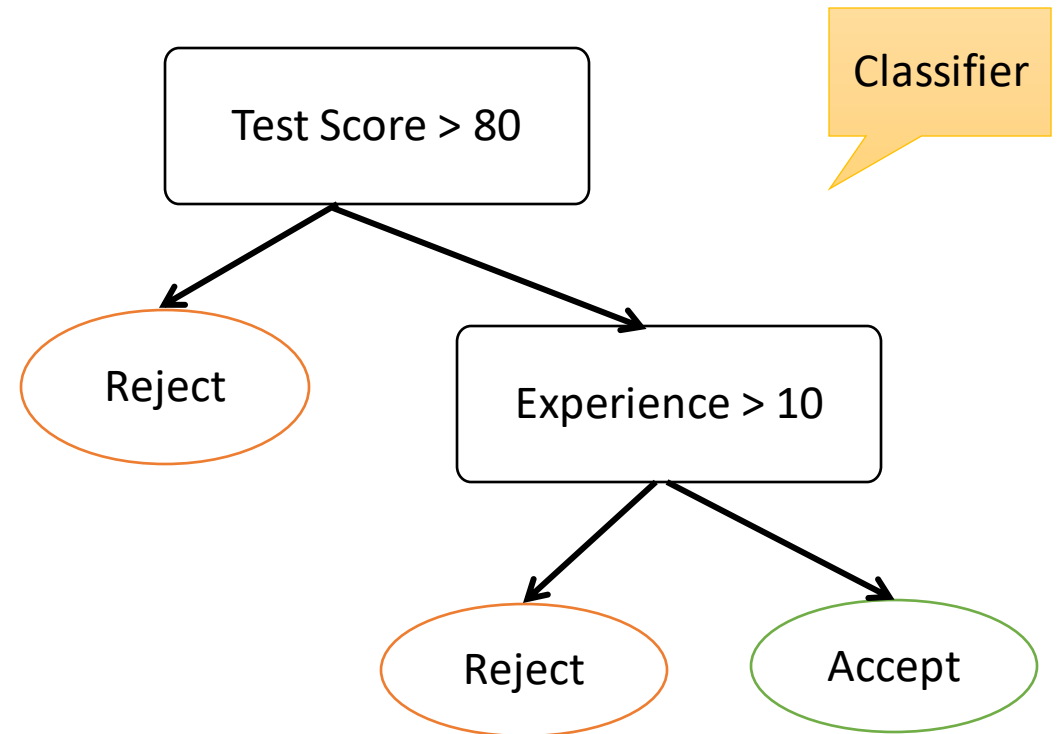


Applicant	Test Score	Experience (in years)	Accepted
A	85	15	1
B	90	7	1
C	78	10	0
D	75	5	0
E	60	2	0
F	95	18	1

New Instance's Classification

- Once a model is developed, it can categorize new data points.

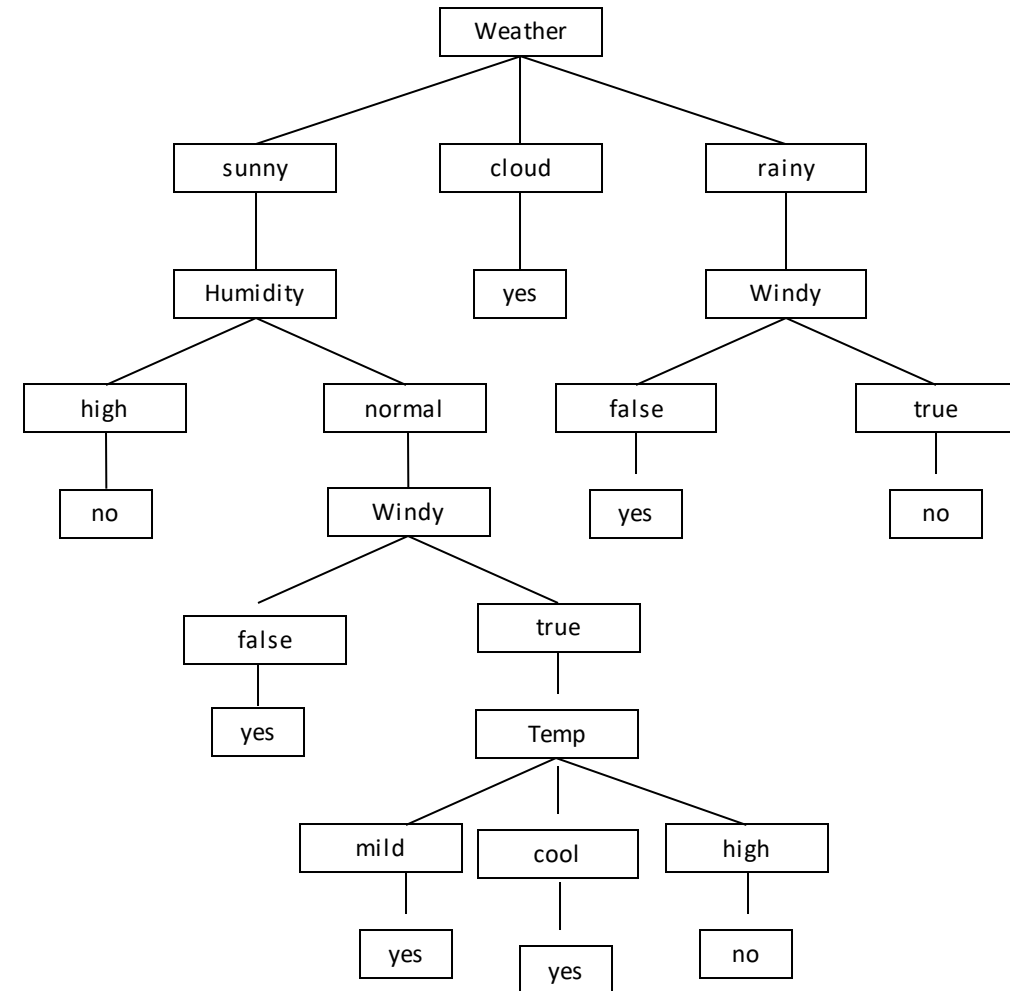
Applicant	Test Score	Experience (in years)	Accepted
Z	89	11	??



When to Stop Splitting?

- If we continue to grow the tree fully until each leaf node corresponds to the lowest impurity, then the data have typically been **overfitting**.
- If splitting is stopped too early, the model may not achieve a sufficiently low error rate on the training data, leading to poor performance.
- Solution!
 - Stop splitting when the best candidate split at a node reduces the impurity by less than the preset amount (threshold: depth of the tree)
 - How to set the threshold? Stop when a node has a small no. of points or some fixed percentage of the total training set (say 5%)

Overfitting Example

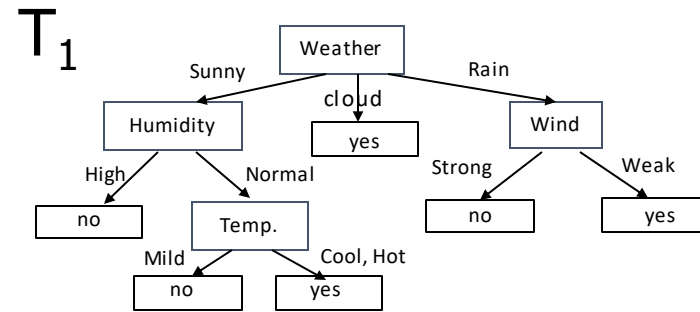


Pruning

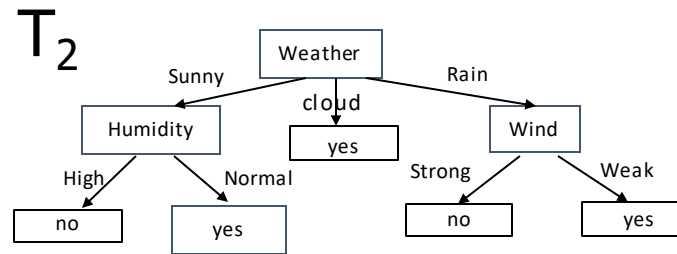
- Sometimes, stopping the growth of a tree too soon can lead to not checking future possibilities enough. This means the tree might stop growing before it reaches the **best accuracy**.
- Here, **Pruning** works in the opposite way to splitting:
 - **first**, let the tree grow completely until each leaf has the least mixed information.
 - **next**, look at every pair of leaves from the same parent. If removing a pair doesn't raise the mixed information too much, then those leaves are removed, and their parent is considered a leaf now.

Reduce-Error Pruning

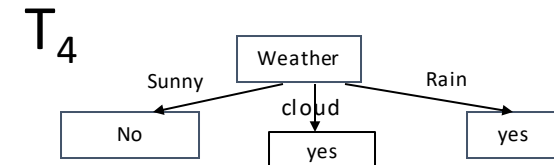
- Illustrative Example:



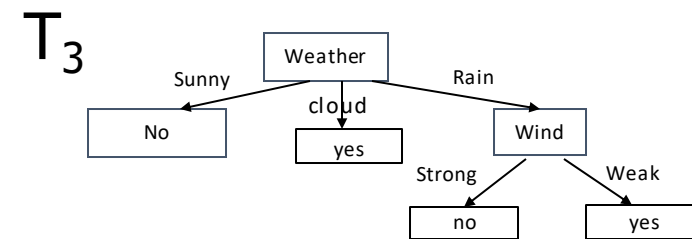
Classification Error = $x + \Delta x$



Classification Error = x

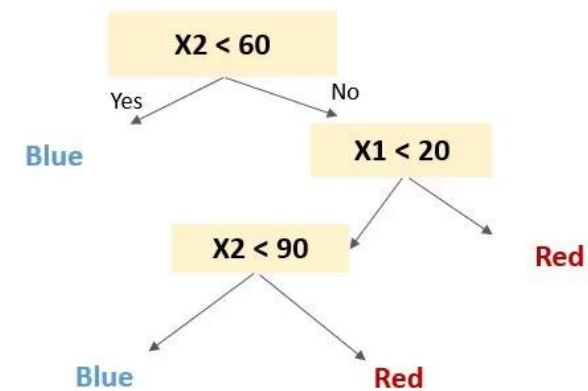
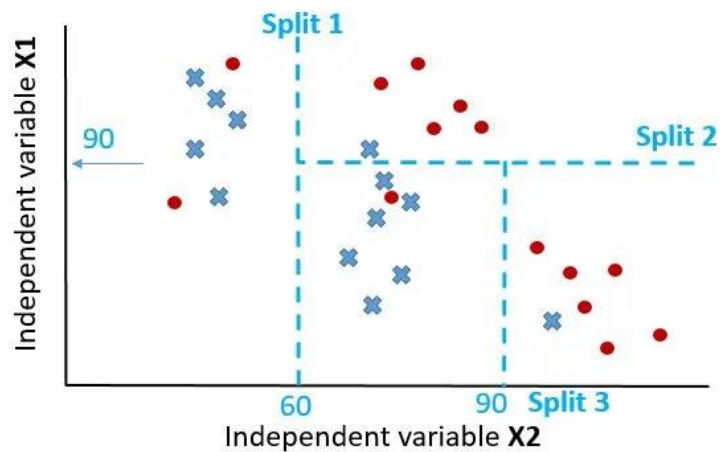
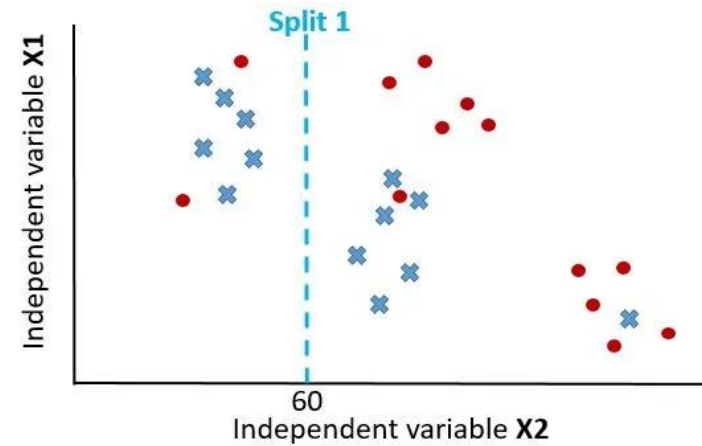
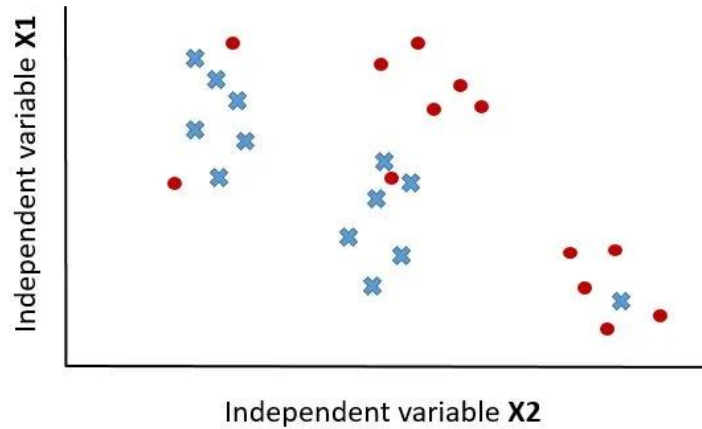


Classification Error = $x + 5\Delta x$



Classification Error = $x + 2\Delta x$

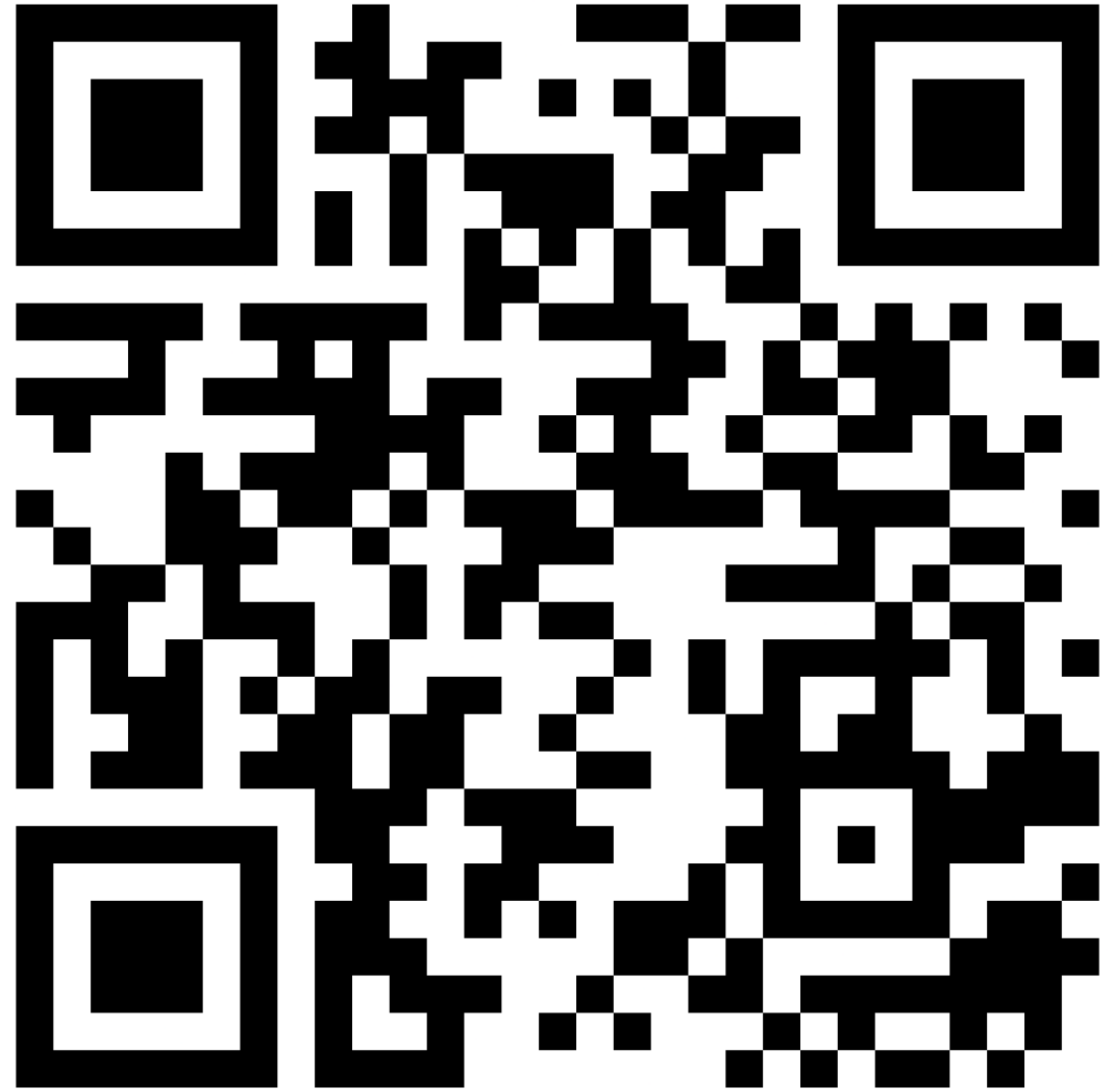
Imperfect Partitioning



Attendance



You can use the provided link if you don't have a mobile phone or if your phone lacks a QR-Code reader – <https://rebrand.ly/ECEFeb13>

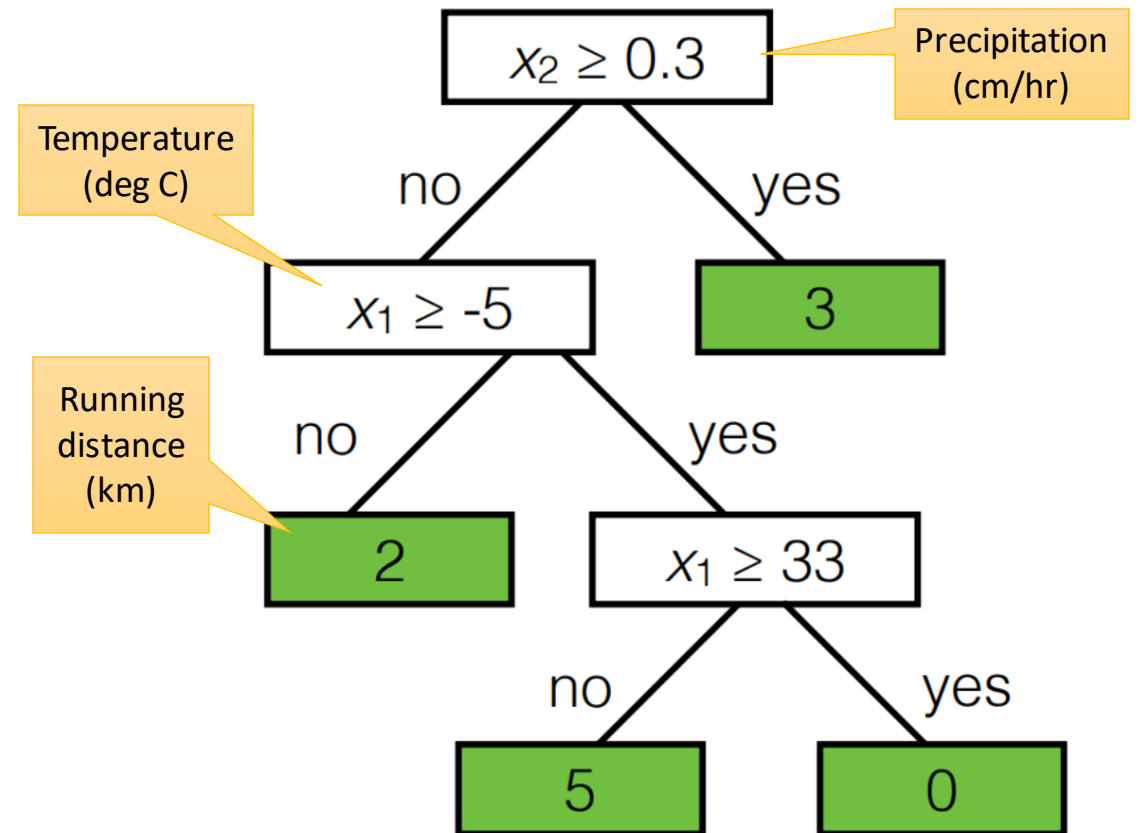


Regression Tree

A thick, hand-drawn style orange line underlining the title "Regression Tree".

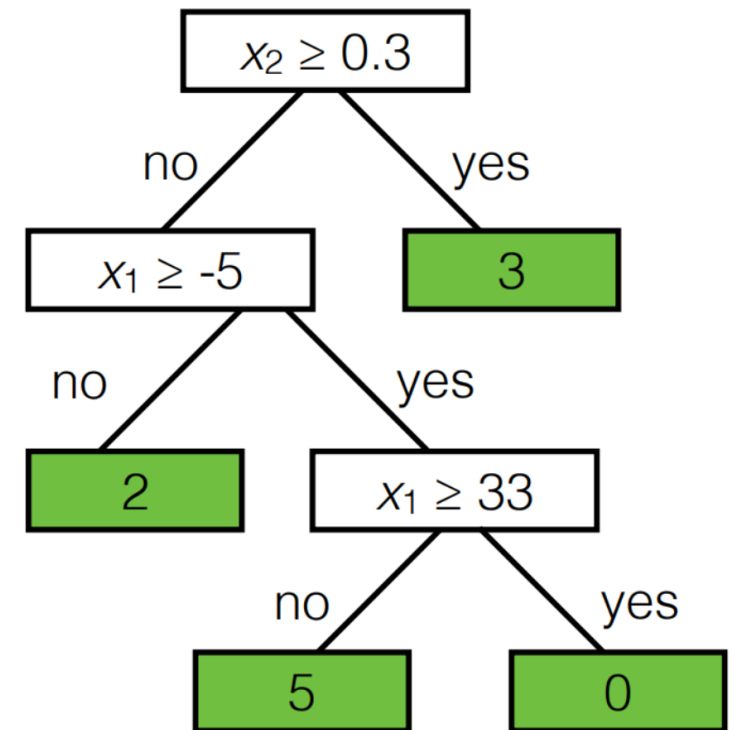
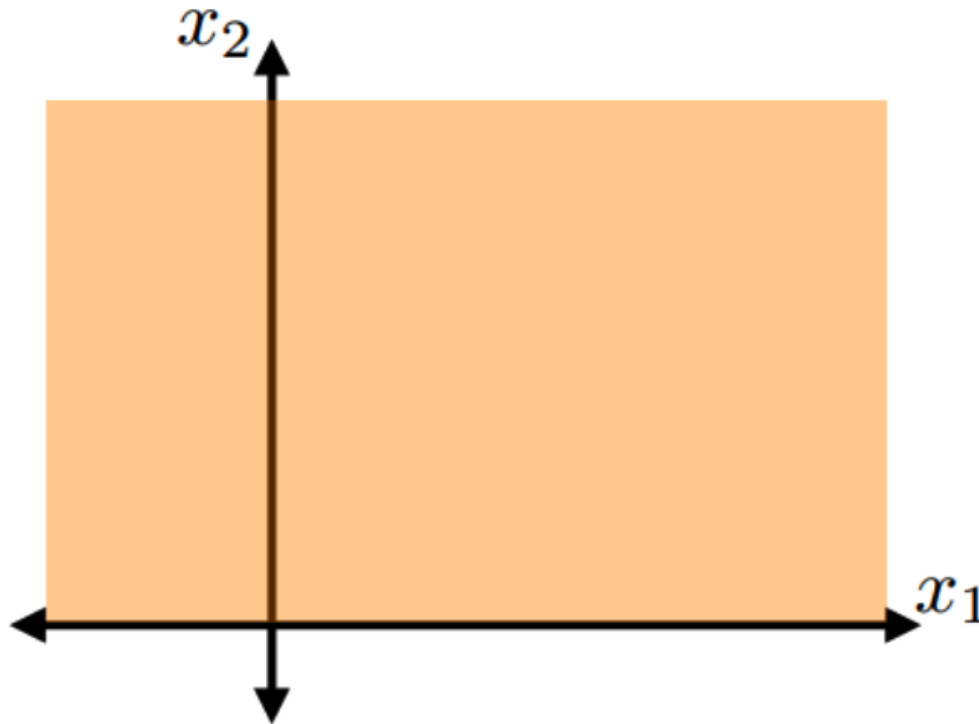
Regression Tree

- A regression tree is a decision tree designed for continuous outcome prediction. Unlike classification trees that predict a discrete label for each input, regression trees predict a numerical quantity.
- Features:
 - x_1 : temperature (deg C)
 - x_2 : precipitation (cm/hr)
- Label $\rightarrow y$: running distance (km)



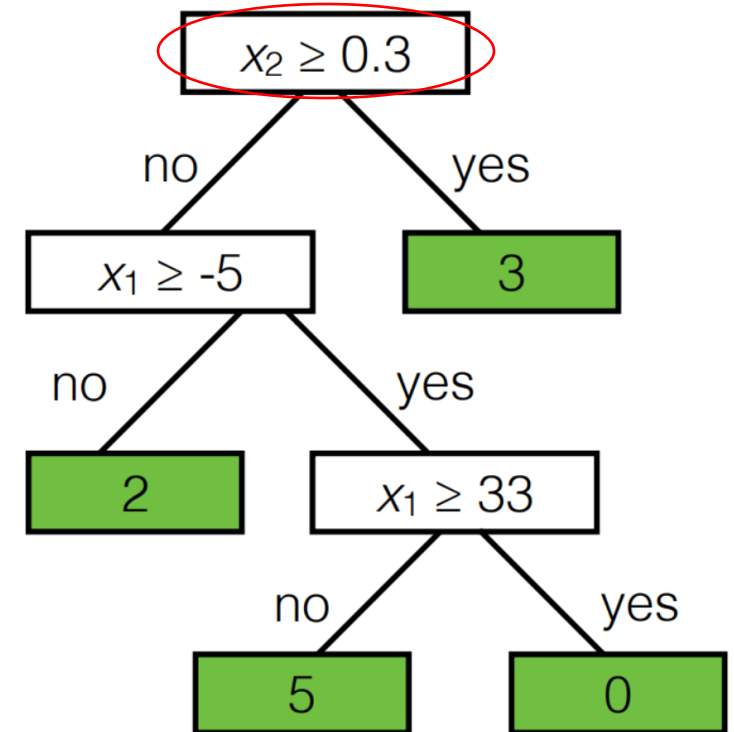
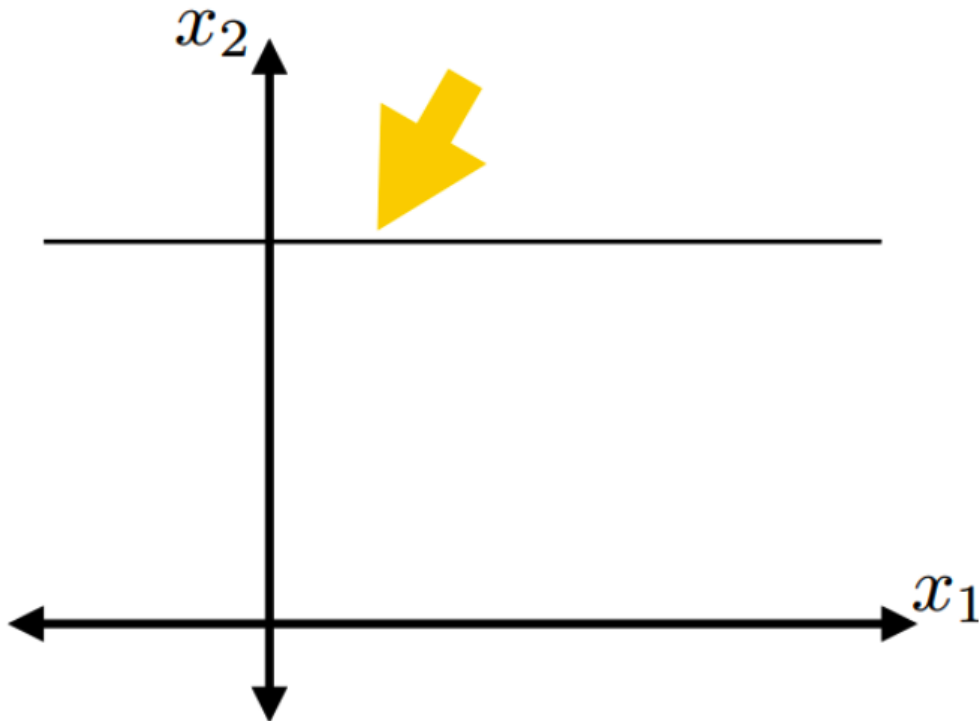
Regression Tree

- Tree defines an axis-aligned “partition” of the feature space:



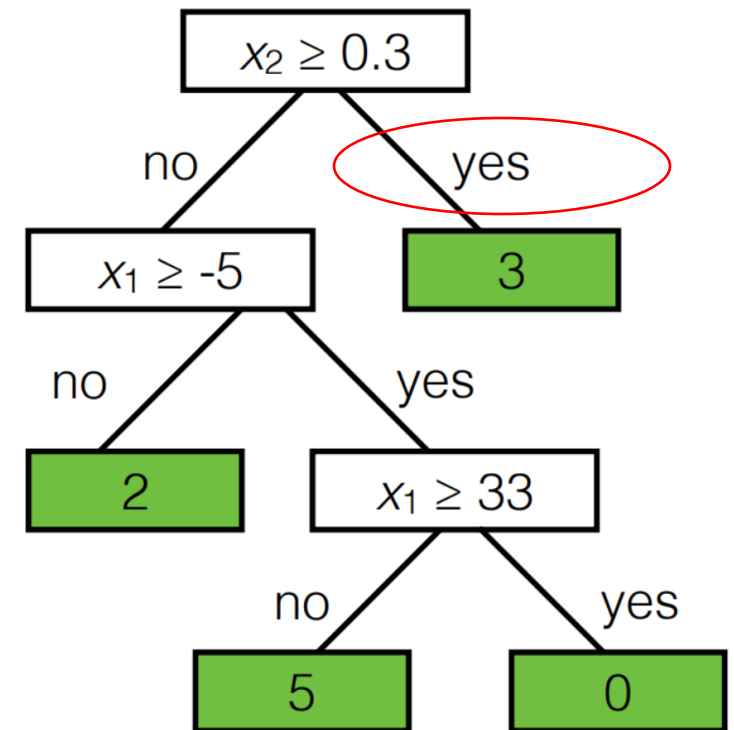
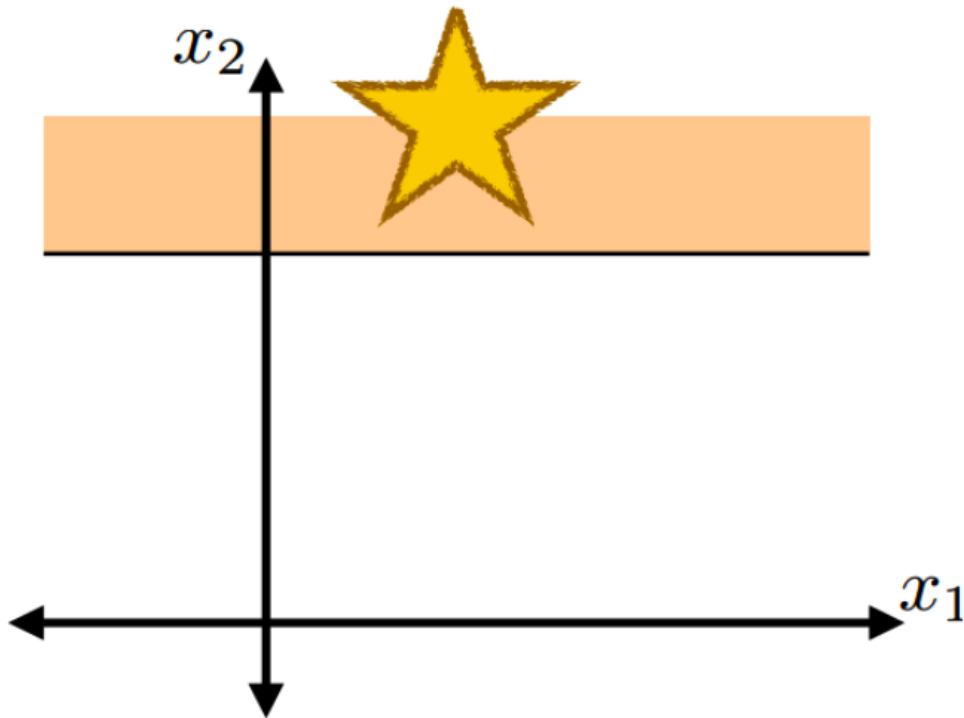
Regression Tree

- Tree defines an axis-aligned “partition” of the feature space:



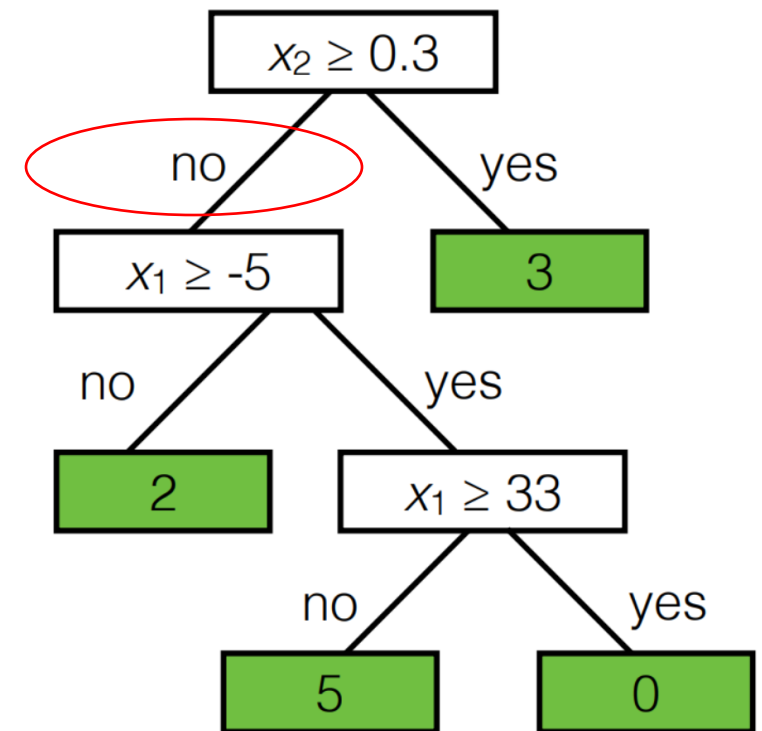
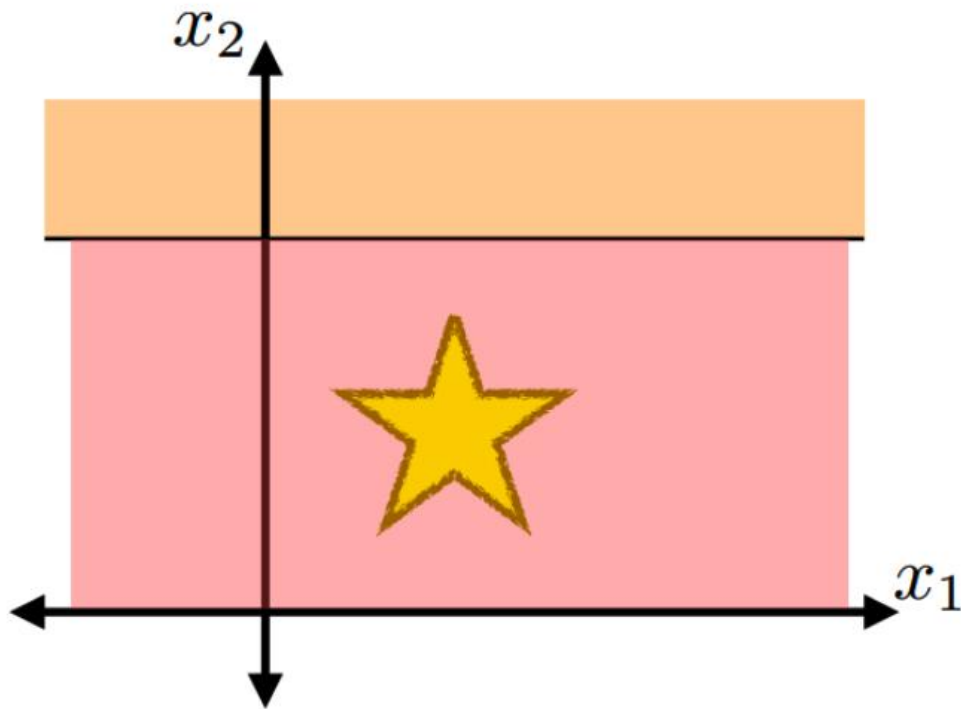
Regression Tree

- Tree defines an axis-aligned “partition” of the feature space:



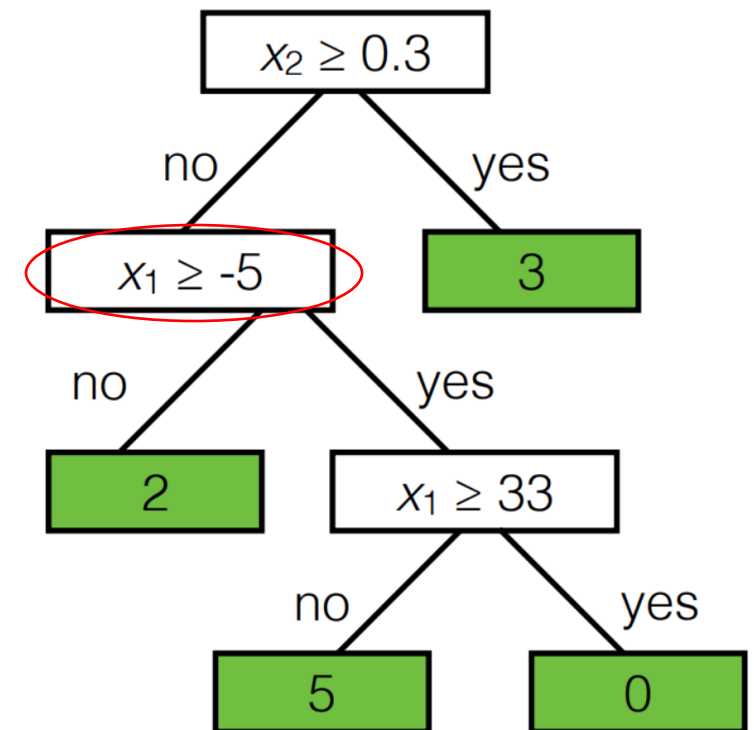
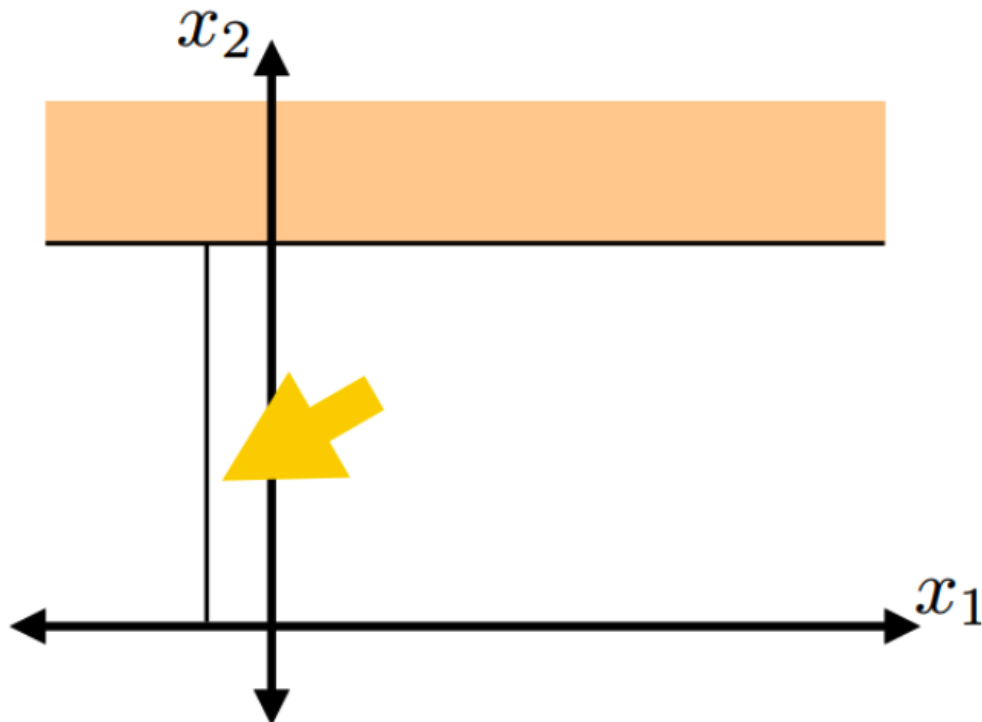
Regression Tree

- Tree defines an axis-aligned “partition” of the feature space:



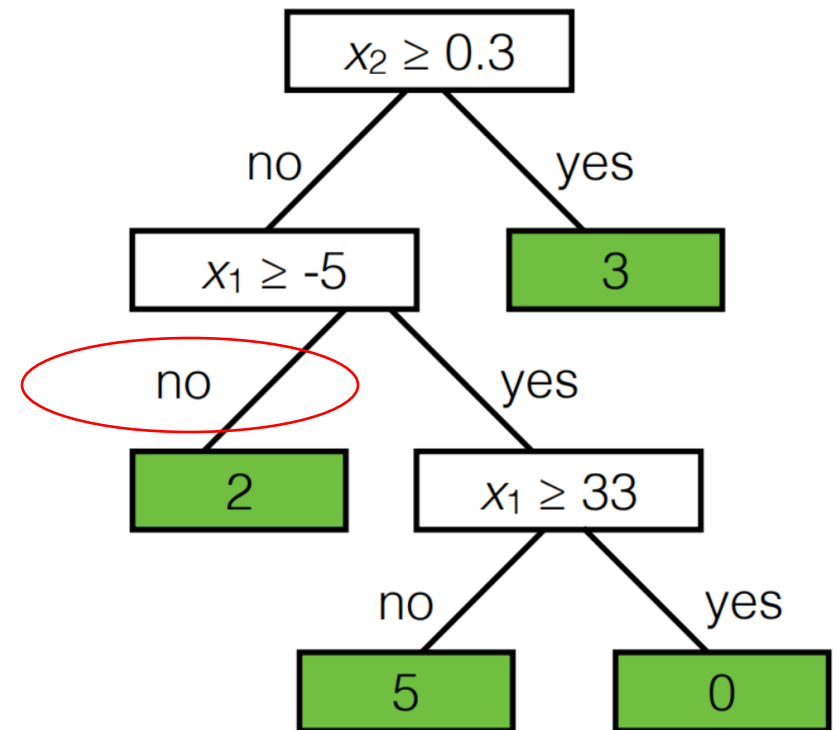
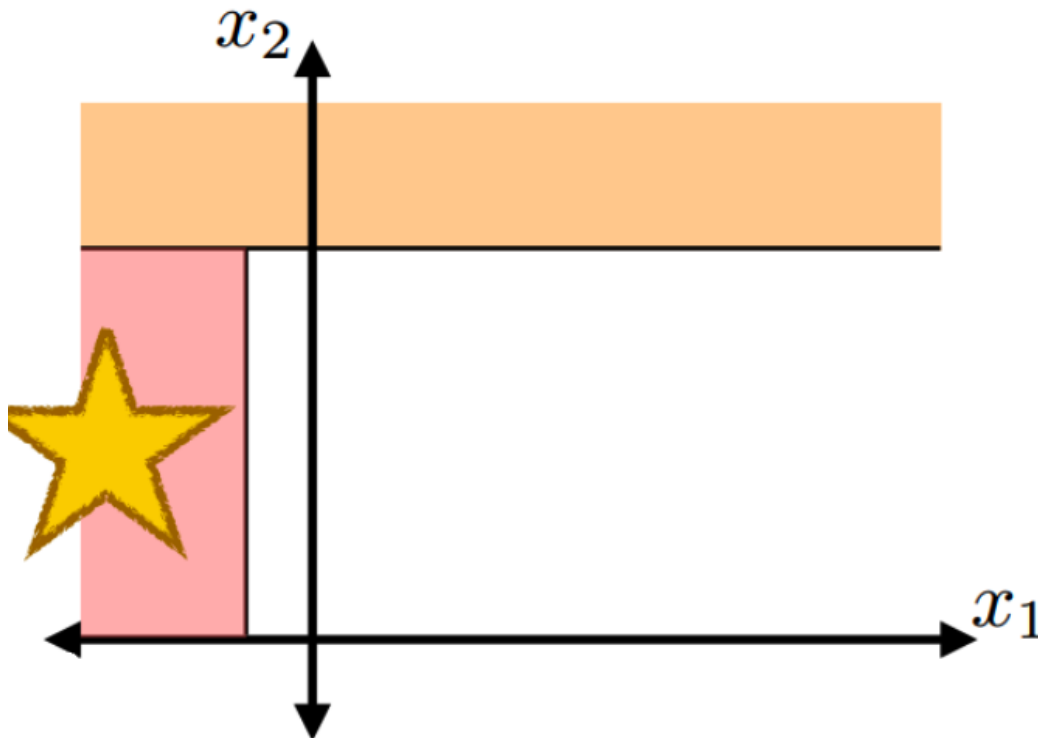
Regression Tree

- Tree defines an axis-aligned “partition” of the feature space:



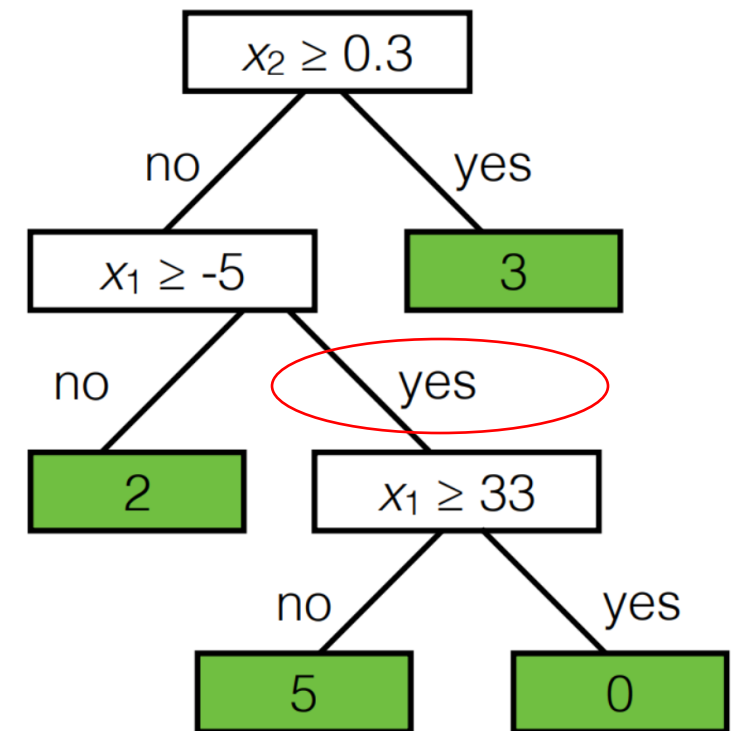
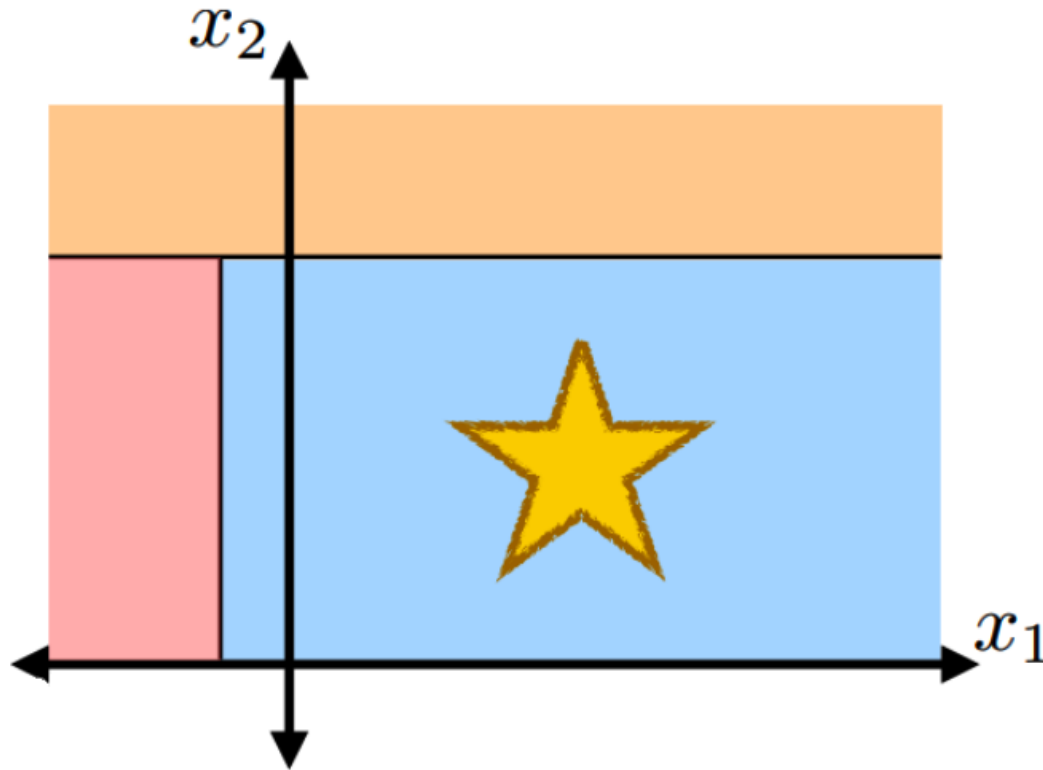
Regression Tree

- Tree defines an axis-aligned “partition” of the feature space:



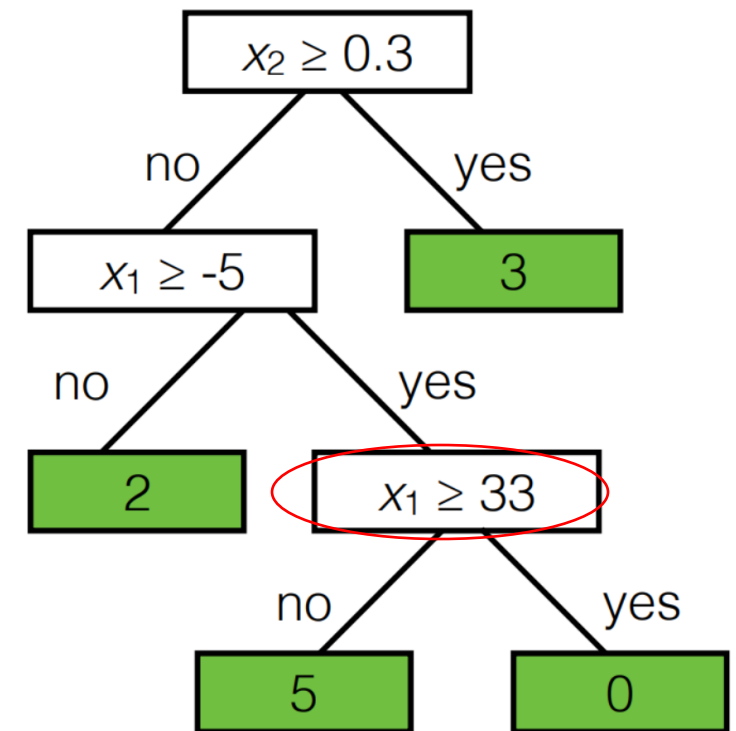
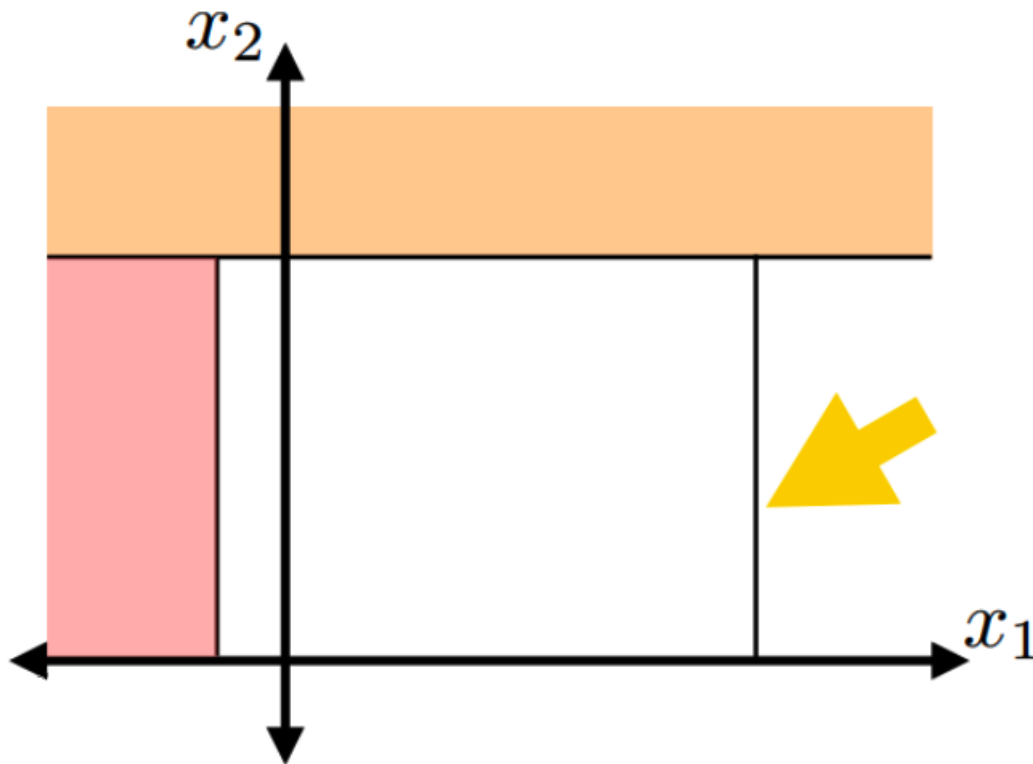
Regression Tree

- Tree defines an axis-aligned “partition” of the feature space:



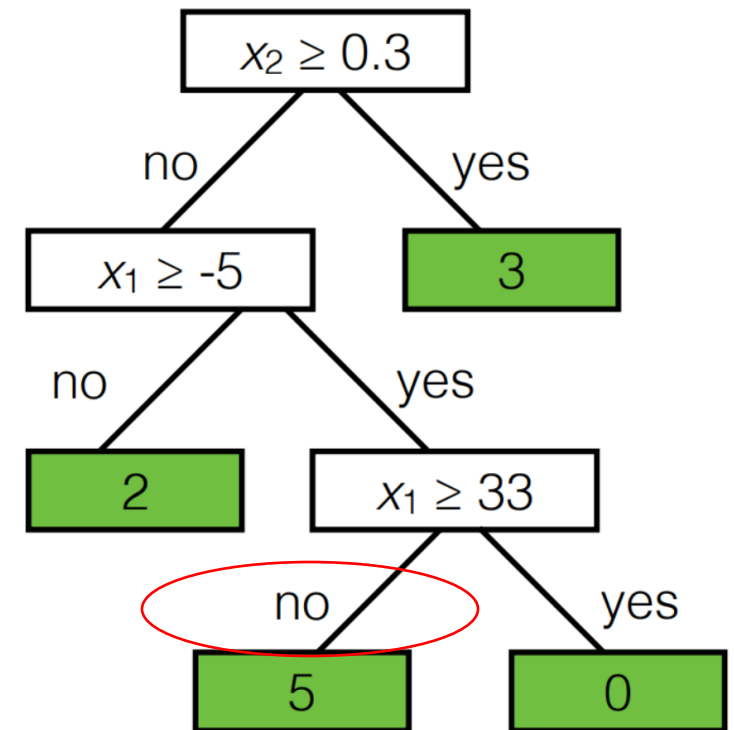
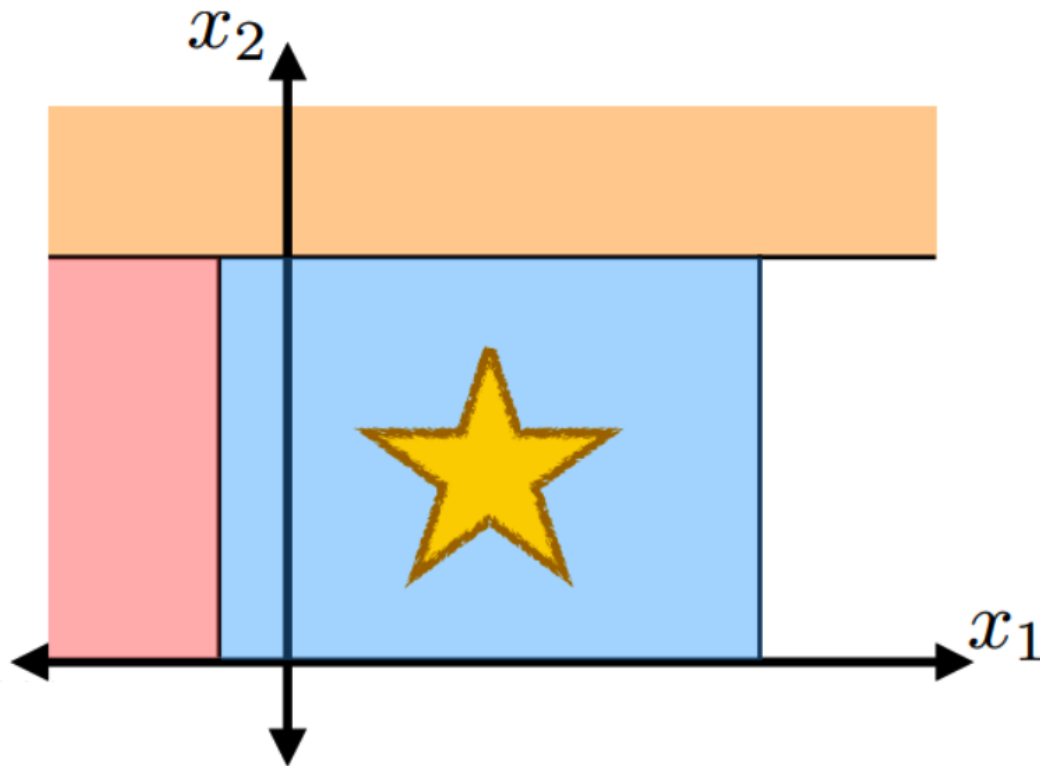
Regression Tree

- Tree defines an axis-aligned “partition” of the feature space:



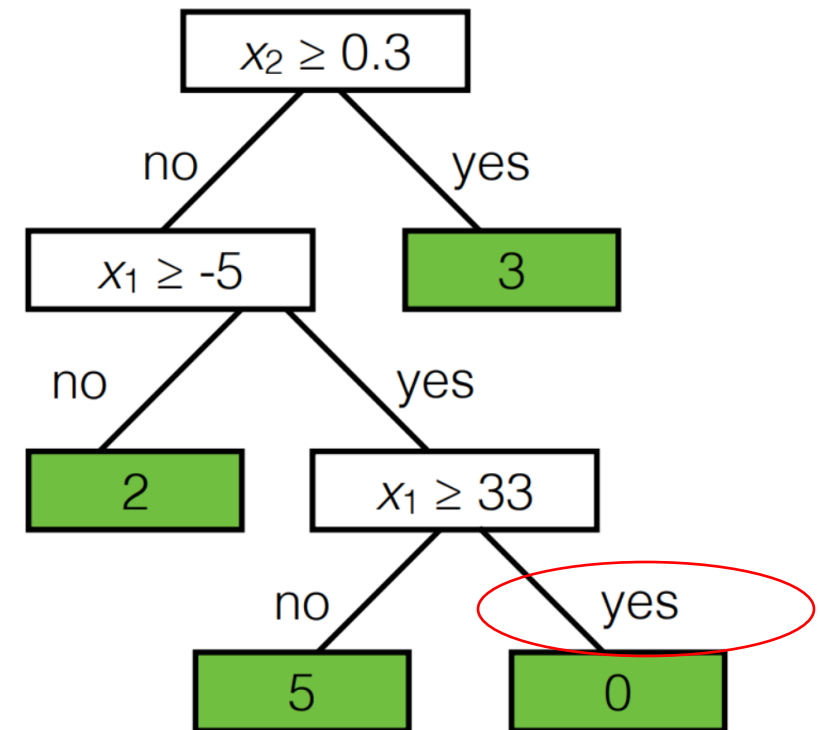
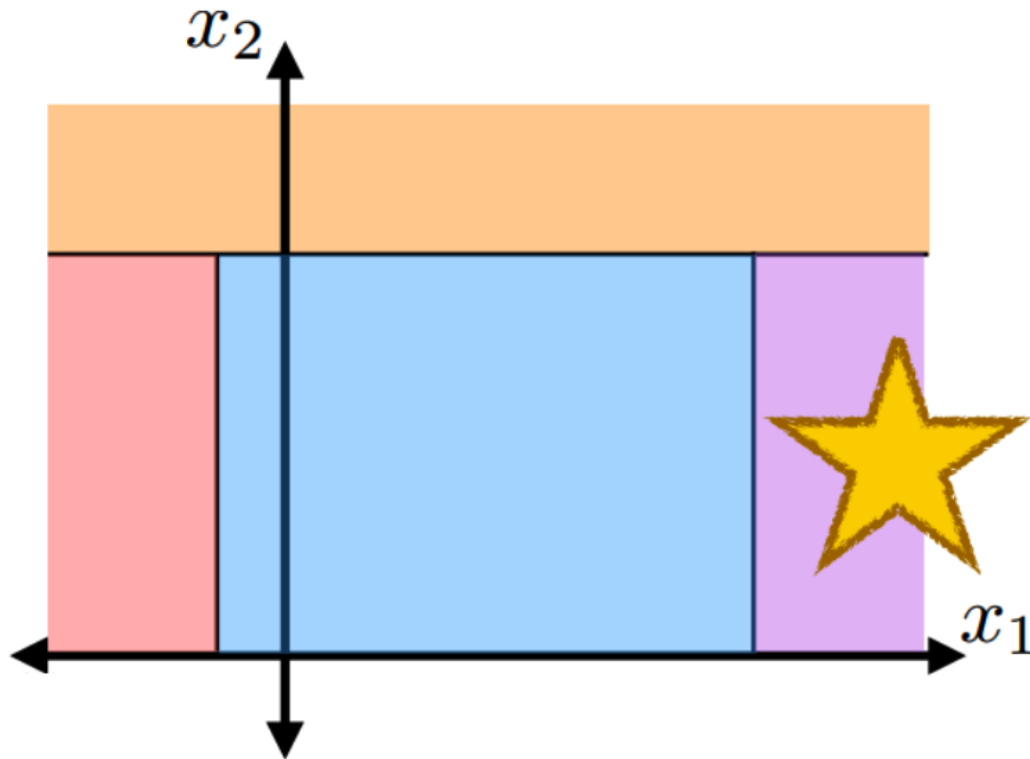
Regression Tree

- Tree defines an axis-aligned “partition” of the feature space:



Regression Tree

- Tree defines an axis-aligned “partition” of the feature space:

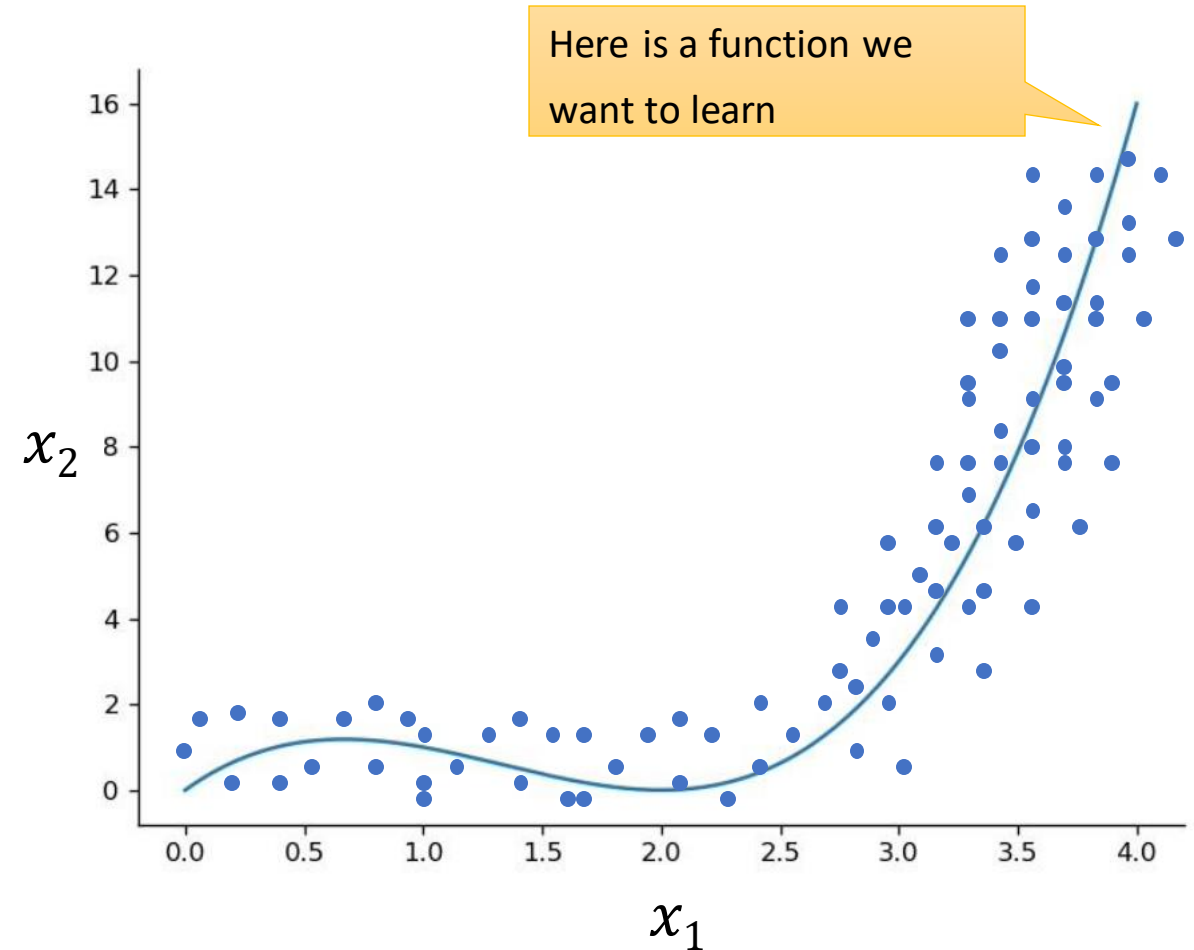


Function Approximation

A thick, hand-drawn style orange line that underlines the title "Function Approximation". It starts under the 'F' and ends under the 'n'.

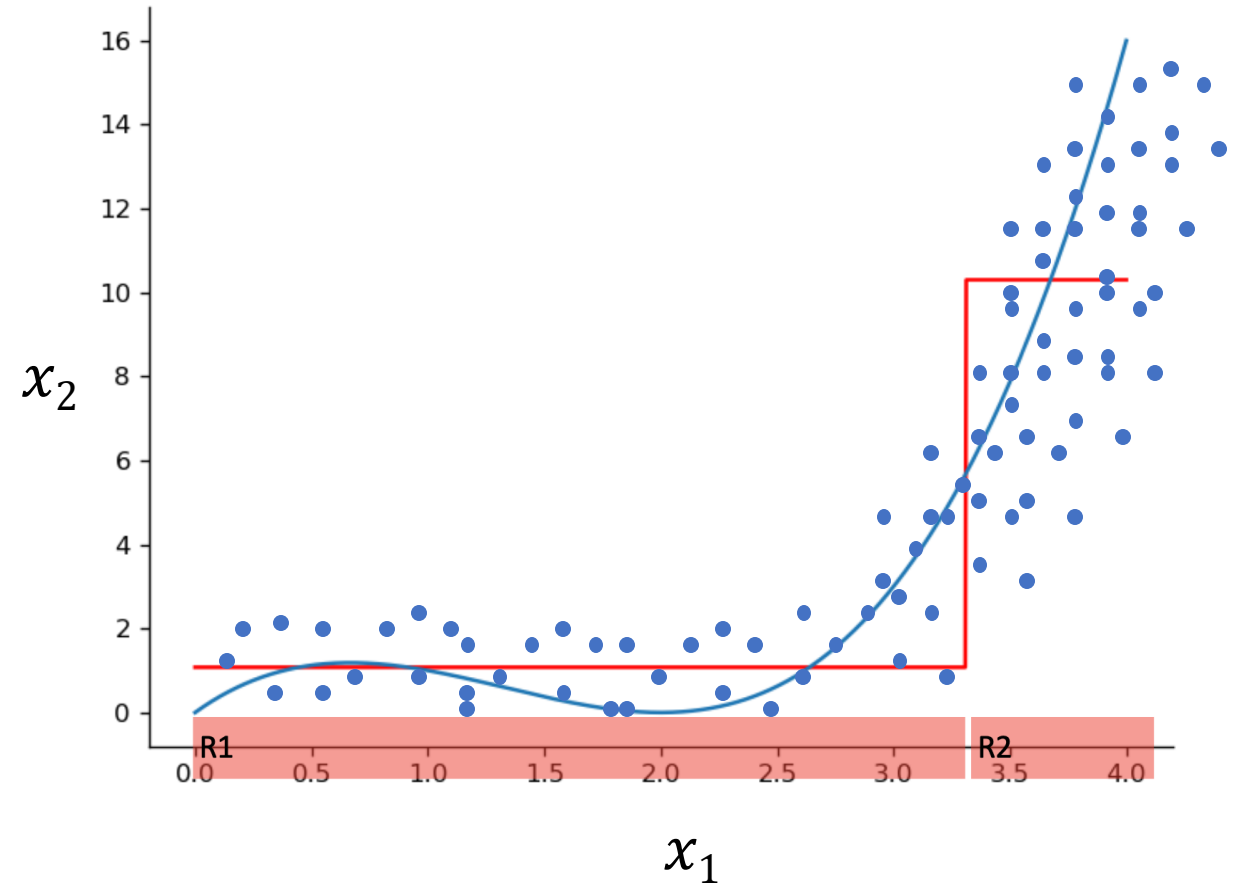
Function Approximation

- Function approximation is finding a function that closely matches a set of points or a dataset.
- In the context of decision trees, **function approximation** involves creating a **model/tree** that can predict the output variable as accurately as possible for given input variables.
- A regression tree will recursively split the feature space into (hyper) rectangles and fit a constant function to each (hyper) rectangle.

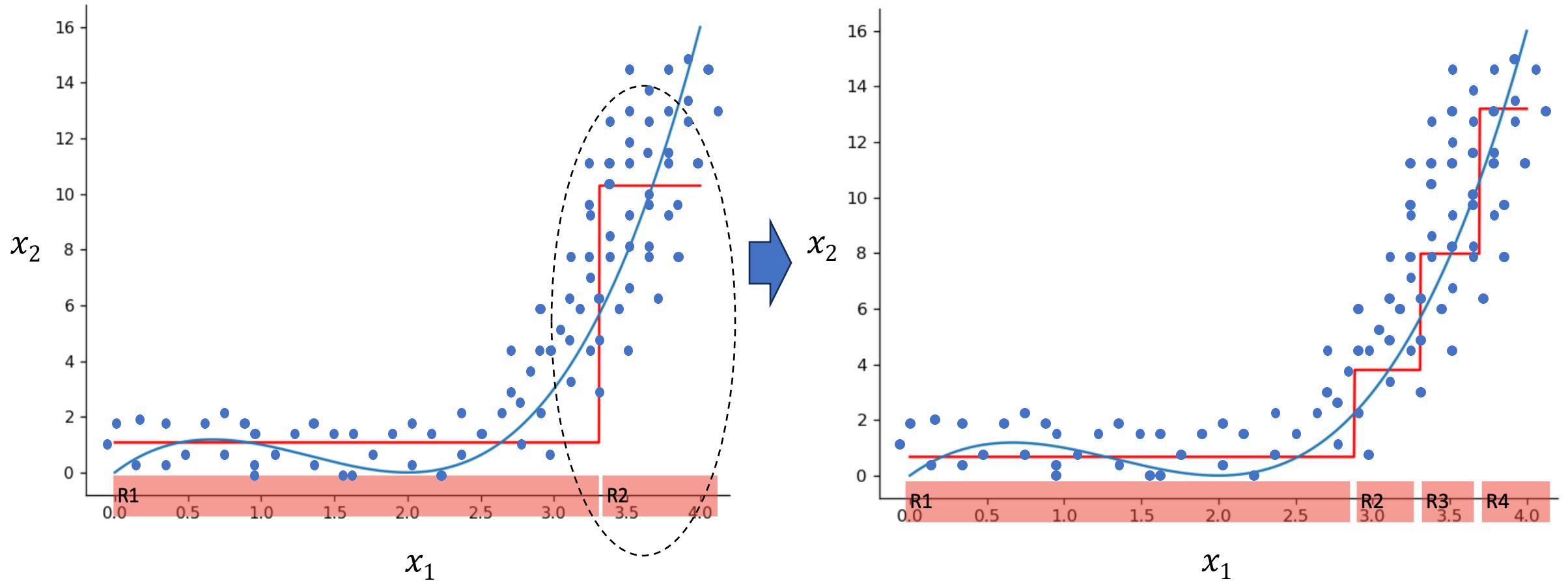


Heuristic For FA (cont.)

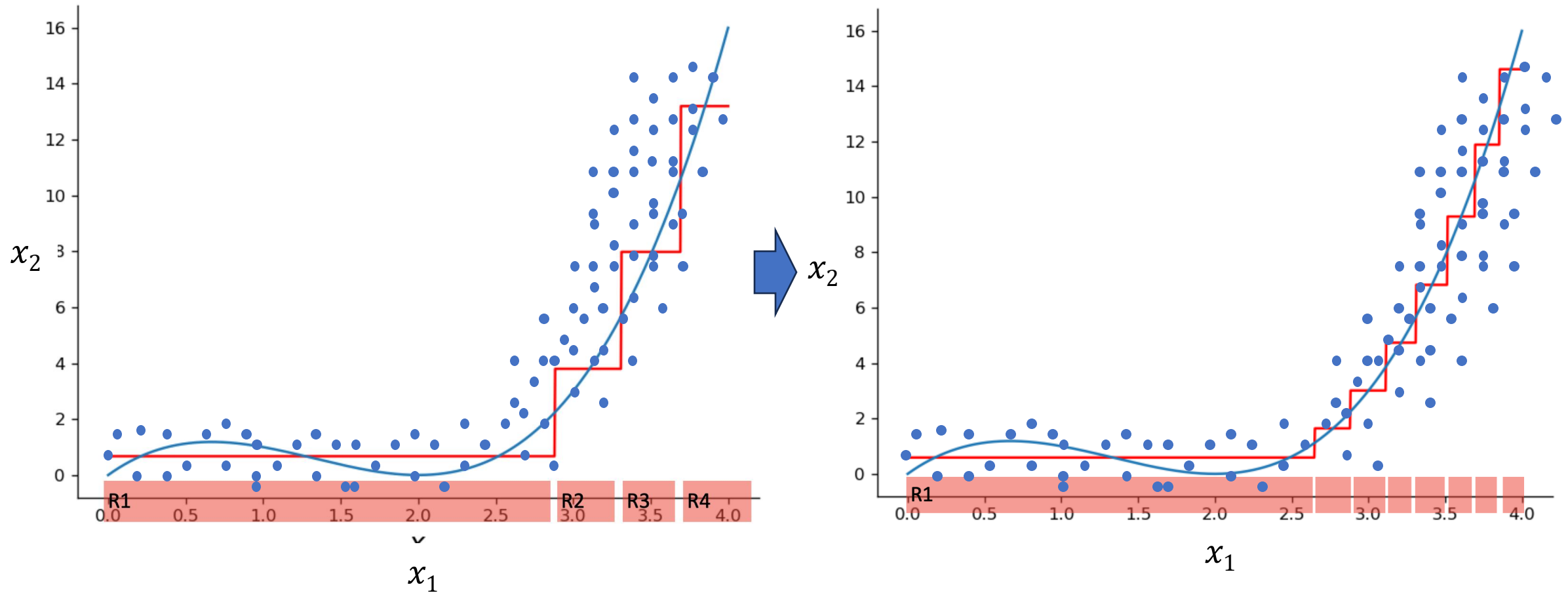
- Greedily decide where to split by determining which split leads to the largest gain in accuracy/reduction in loss
- Terminate when
 - Did enough splits, or
 - Rectangles have a minimum number of observations



Heuristic For FA (cont.)

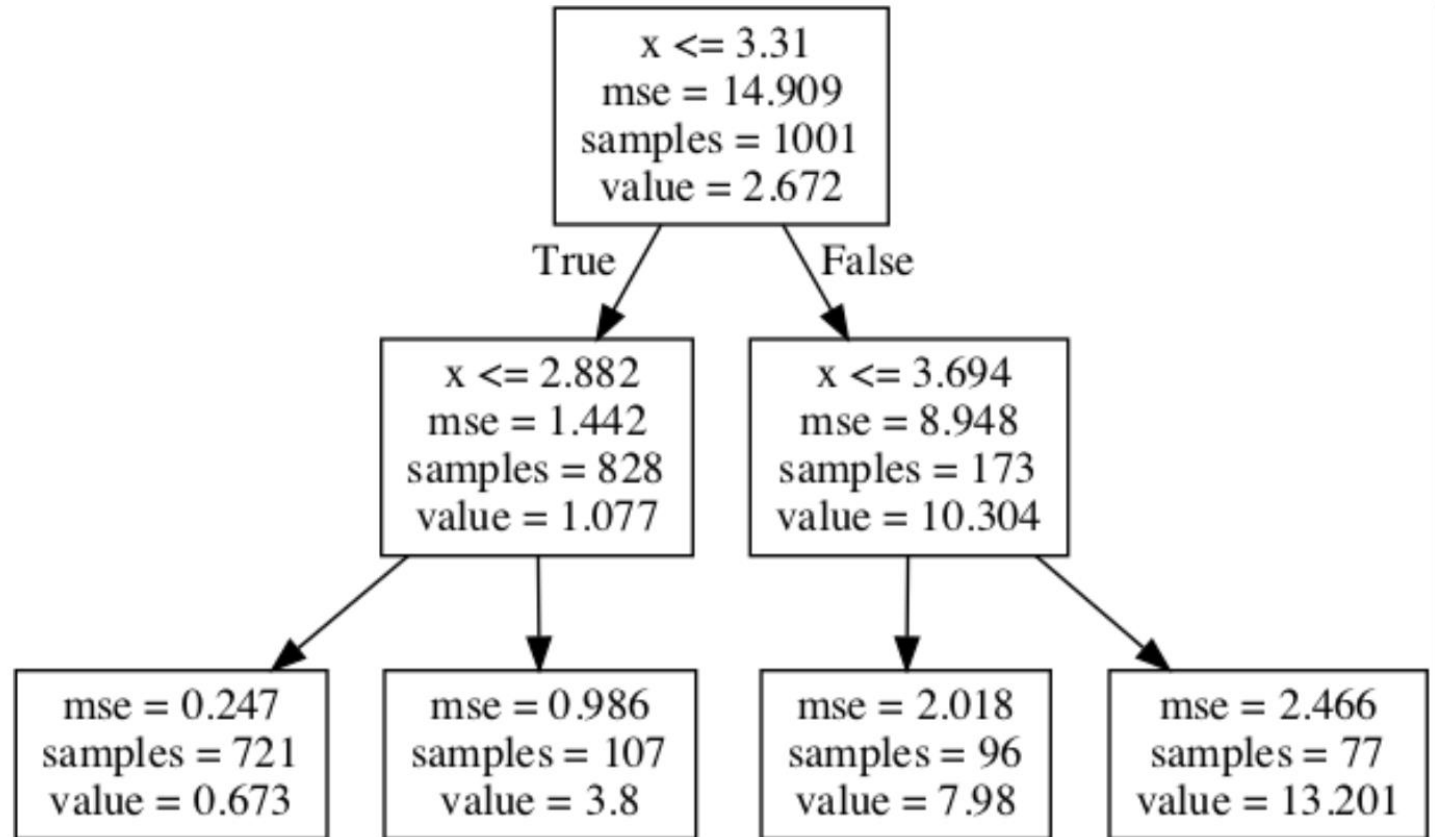


Heuristic For FA (cont.)



Visualizing The Process

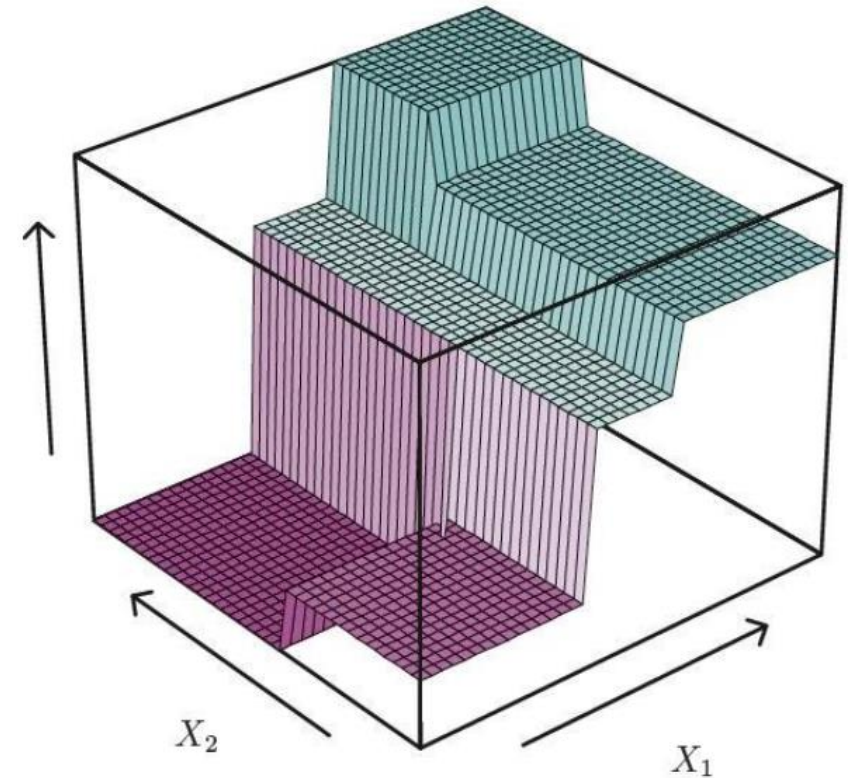
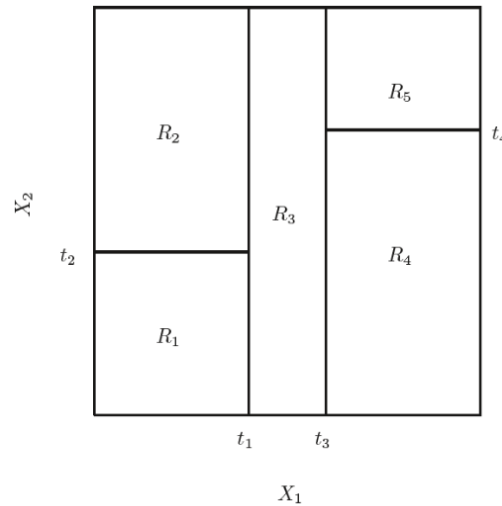
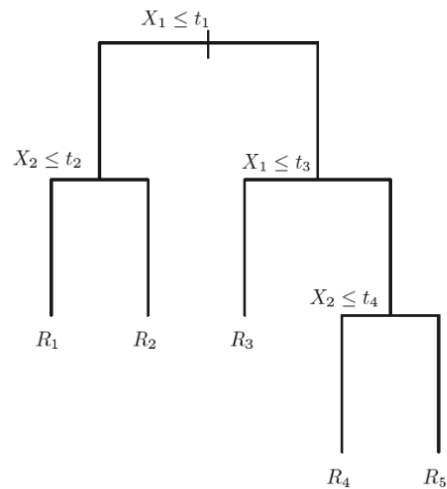
- **Splitting Criterion** – The data at each node is split based on a feature and a threshold that minimizes the target variable's variance (or mean squared error, MSE) in the resulting child nodes.
- The process is repeated recursively for each child node until a stopping criterion is met (such as a **minimum number of samples** in a node or a **minimum reduction in variance**).



Part of the full regression tree

What About More Dimensions?

- It's the same process, but now we can split on other variables!
- Node still is a decision on one feature.



Decision Tree Flaws

- The DT algorithm is **greedy**, which means we select the best feature based on the split at that node, not the resulting splits in future nodes.
- Although trees are invariant to scale, they can overfit easily when they get very deep. We can limit this by pruning the tree, which might reduce predictive power.
- Trees can fit the training data closely, **capturing noise or minor fluctuations**. This is a key aspect of the instability of decision trees.
- DTs **lack smoothness** for regression.

How can we overcome DT flaws?

A few ways to do this, including but not limited to:

- Bagging
- Random Forest
- Boosting

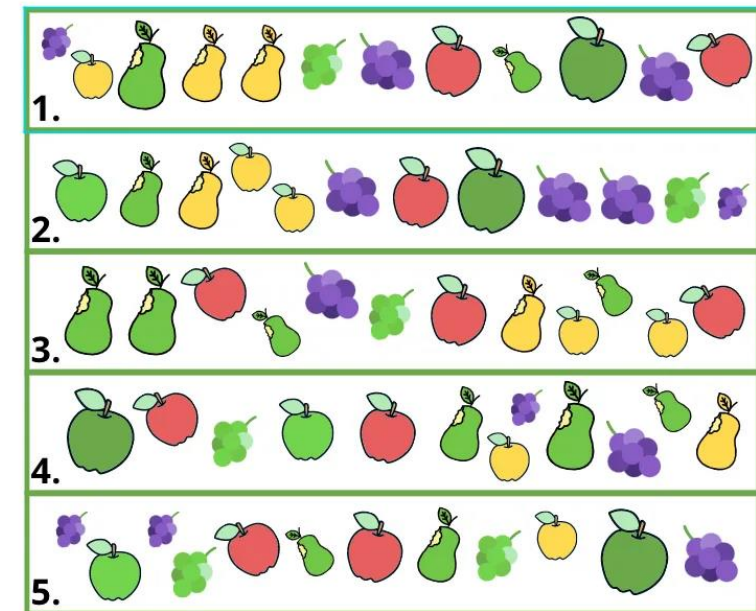
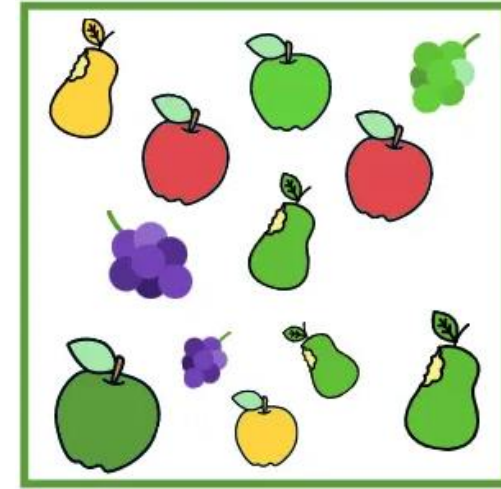
Bagging

Bagging

- Bootstrap Aggregation, commonly known as **bagging**, is an ML ensemble meta-algorithm designed to improve the stability and accuracy of algorithms used in classification and regression.
- **bagging** also reduces variance and helps to avoid overfitting.
- Although it is usually applied to decision trees, it can be used with any other method.

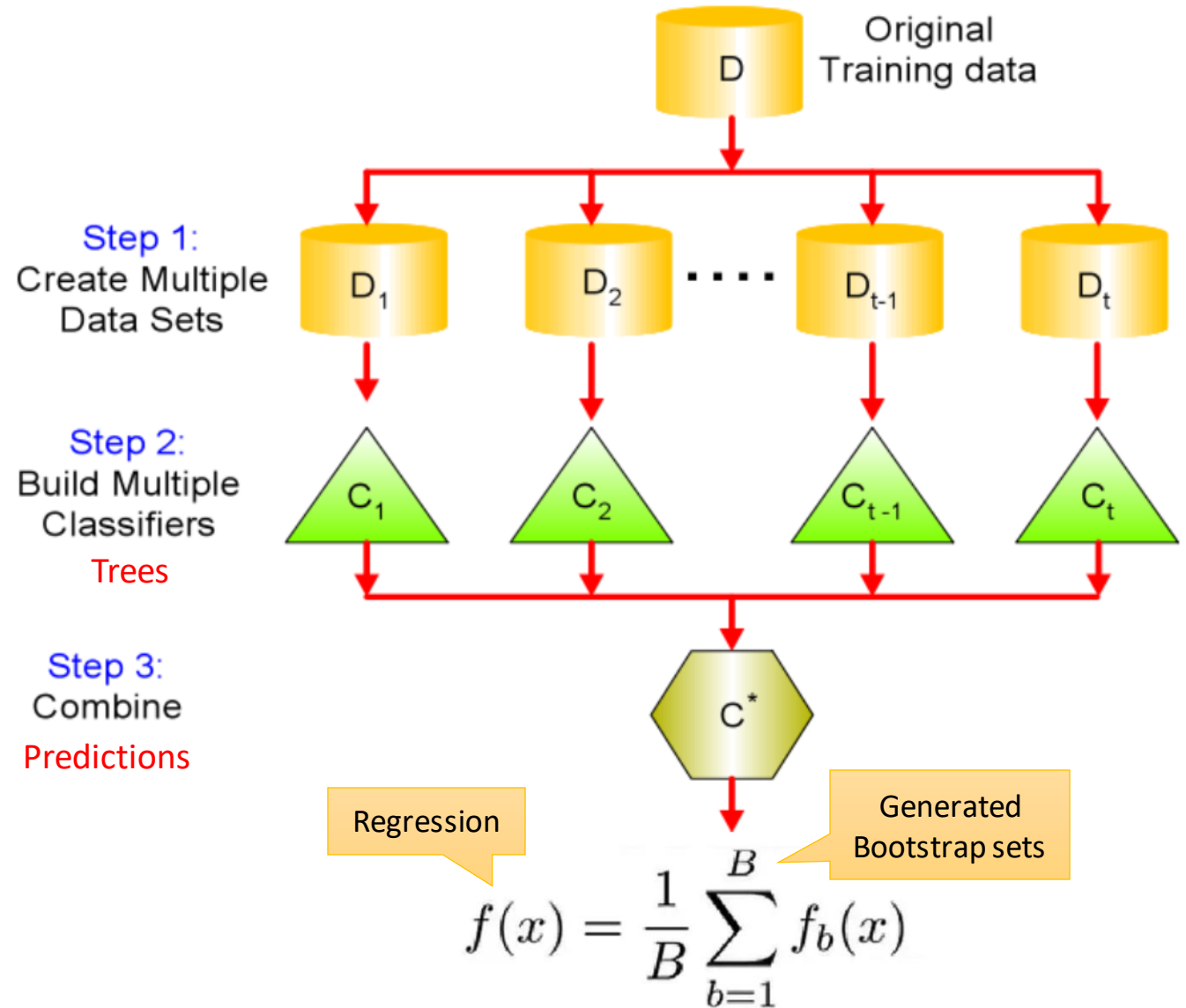
Sampling with replacement

- **Bagging** involves creating multiple datasets from the original using bootstrapping (sampling with replacement).
- Each new dataset is created by randomly selecting observations with replacements from the original dataset, typically the same size as the original.



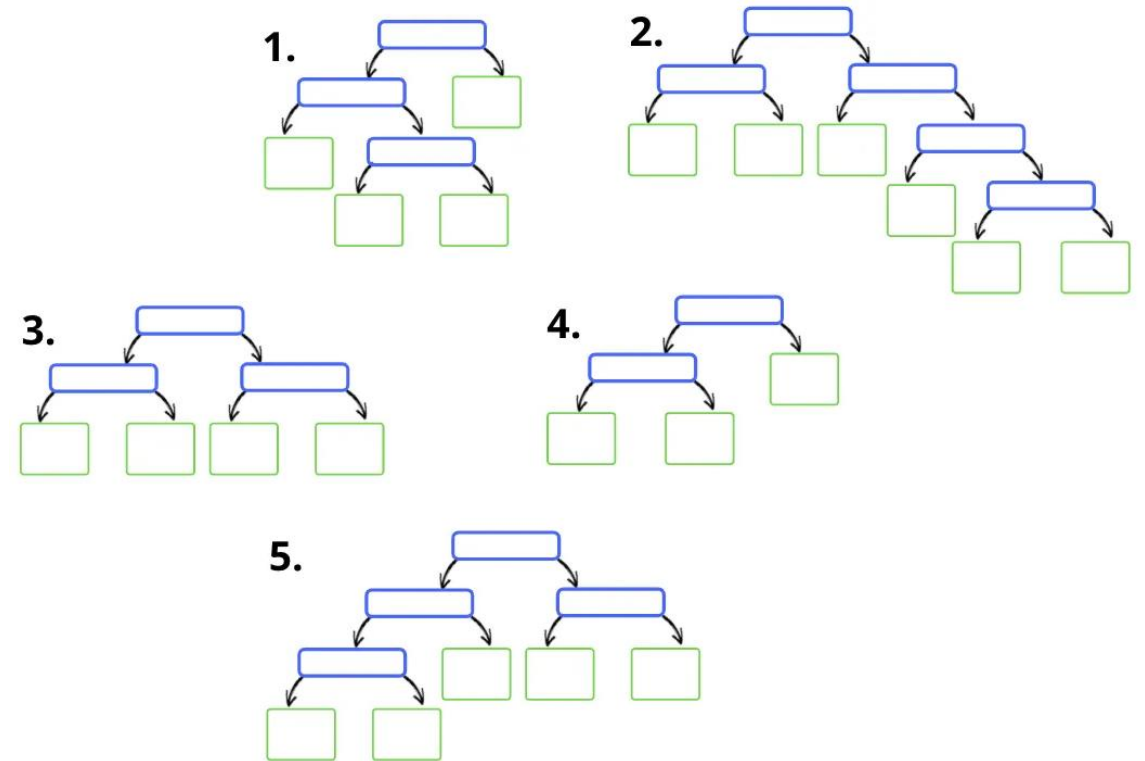
Steps to Perform Bagging

- Combine Predictions – The multiple models are then combined by **averaging the predictions** in the case of regression, or by **majority voting** in the case of classification.
- Regression can also involve taking the median to provide the final prediction, which can improve robustness to outliers.



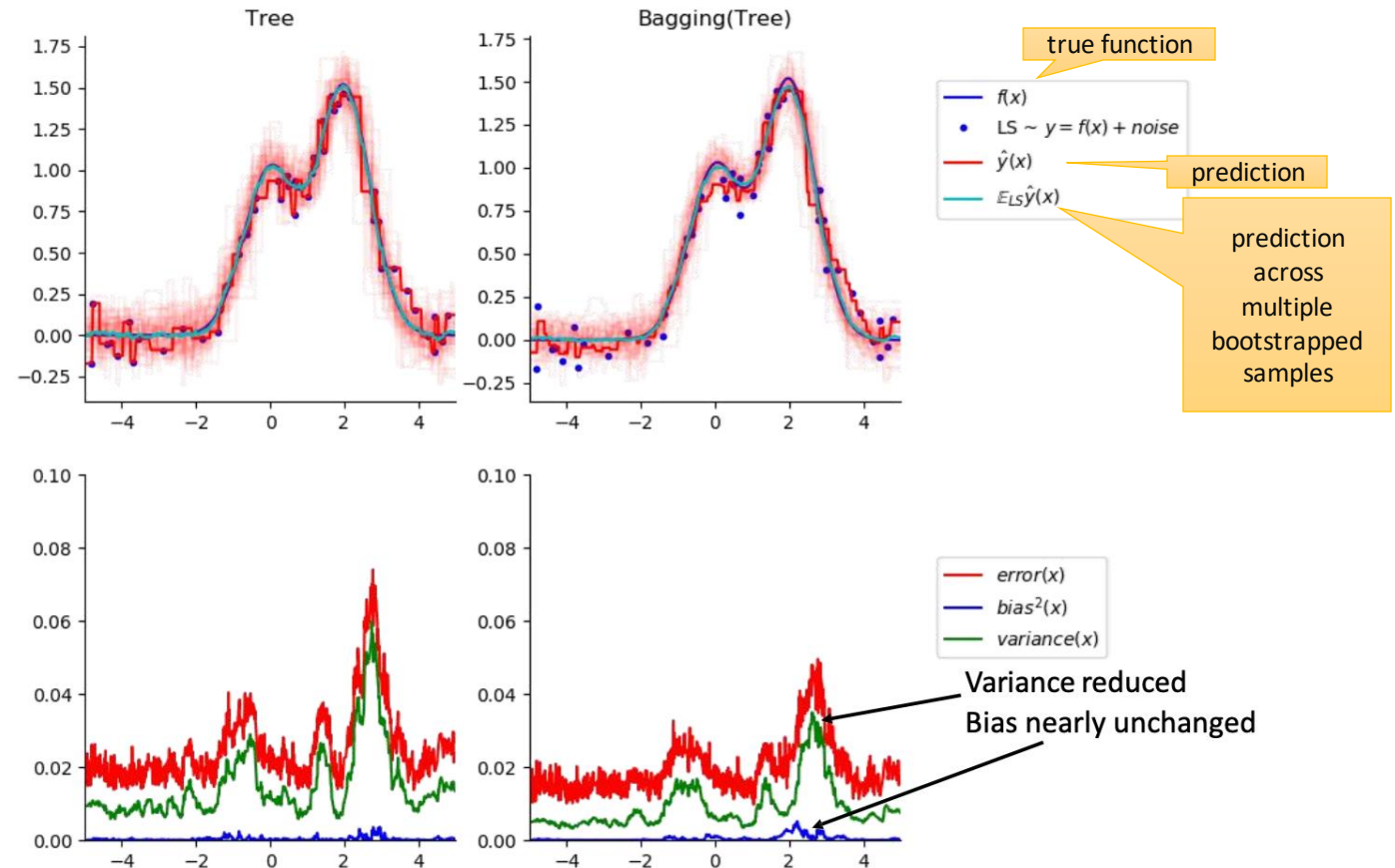
Build Multiple Trees/Models

- Bagging in the scikit-learn default setting is 100 samples.
- For more robust modeling, it's common to use several hundred trees.
- While adding more trees can enhance the model's performance, there is a point of diminishing returns where additional trees no longer yield significant improvements but require more computational resources.



Variance of Bagging

- The comparison between the single tree and the bagging approach shows that bagging typically reduces variance without significantly increasing bias, which results in a more reliable and accurate predictive model.



Bagging For Classification

- Bagging enhances an effective classifier without overfitting by reducing variance and thus improving its consistency across different training datasets.
- Training effective classifiers on diverse data samples and merging their predictions can average out anomalies, yielding a more general and stable model.
- Bagging may worsen a classifier with high bias, as it cannot correct fundamental flaws and may amplify existing errors through aggregation.

Random Forest

A thick, hand-drawn style orange line that underlines the title "Random Forest".

Random Forest

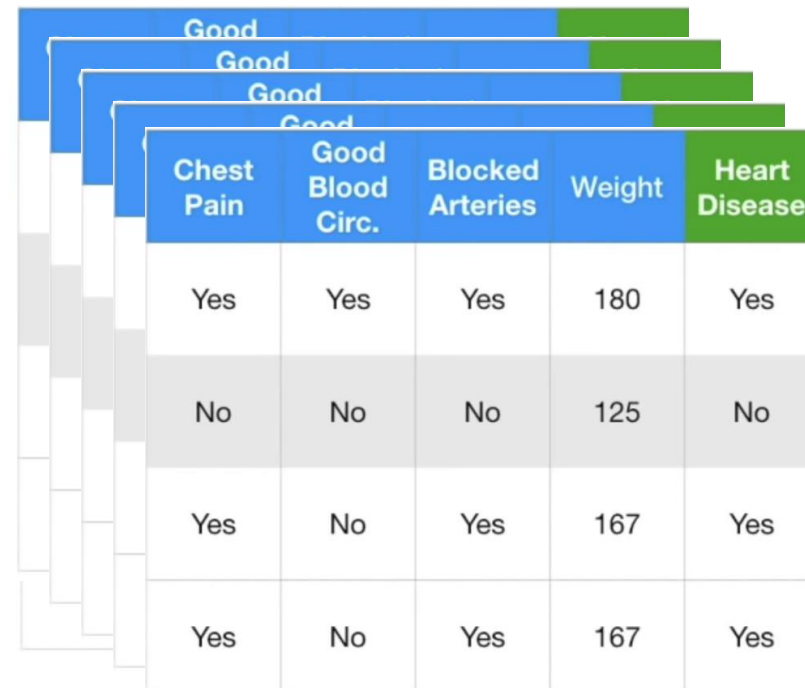
- Random Forest is considered an improvement over bagging for **decision trees**.
- Bagging may result in correlated trees if they are repeatedly split on the same features. Random Forest mitigates this by randomly choosing different feature subsets for each split, ensuring greater tree **diversity** and **less focus** on dominant features.
- This random feature selection creates a more diverse set of trees, often leading to a stronger overall model and substantial performance gains.

Training a Model

- Suppose we have the following sample dataset that has been Bootstrapped a number of times.
- Step 1 – Take repeated bootstrap samples from the training set

Original Dataset

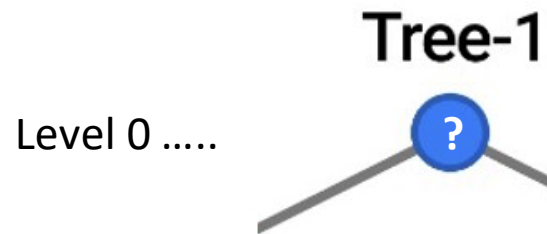
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes



Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

Training a Model (Cont.)

- Step 2 – Create a tree:



...we randomly
select 2.

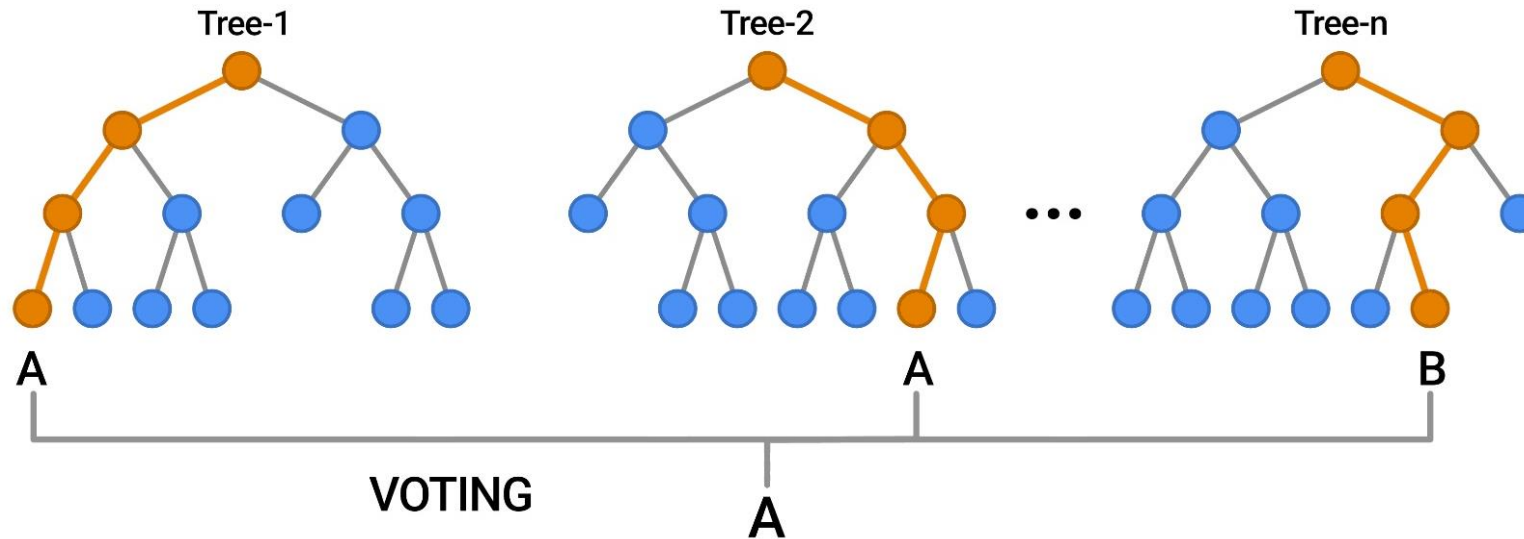
Next, only consider a random subset of
the variables of each split.

Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

Training a Model (Cont.)

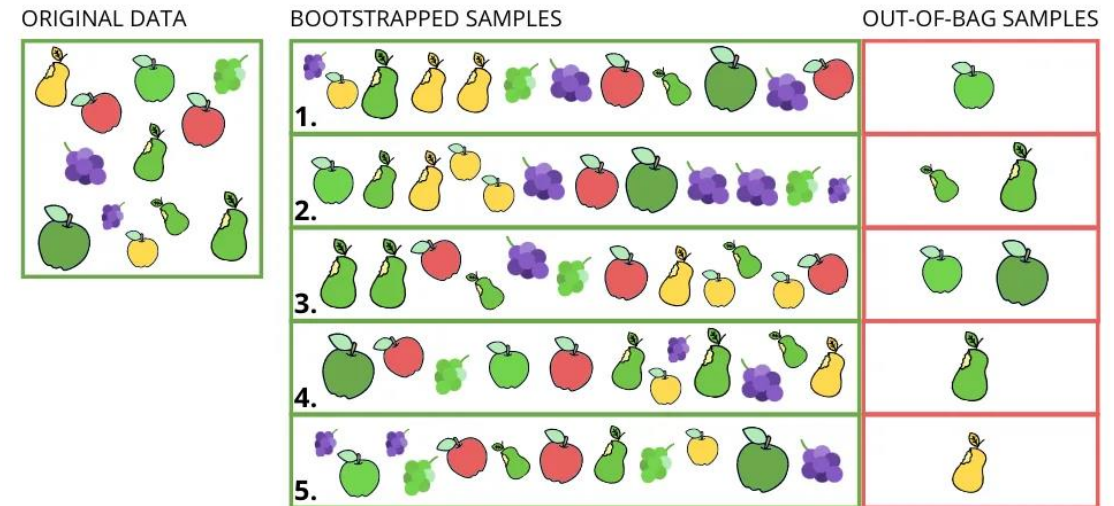
- Step 3 – Create a forest and take the majority vote:



- Using a bootstrapped sample and considering only a subset of the variables at each step results in many trees.
- The variety is what makes random forests more effective than individual decision trees.

Out-Of-Bag (OOB) Scoring

- In classification tasks, we evaluate the Random Forest model using standard measures such as accuracy, precision, and recall. Additionally, bootstrapping introduces a special metric known as **out-of-bag (OOB) scoring**.
- About one-third of the original data points typically aren't selected for each bootstrap sample during the bootstrapping process, acting as "unseen" data for each tree.



- The figure above shows the out-of-bag samples for each bootstrap sample. Note that the proportion isn't exactly one-third for each tree, due to how the samples were created.

RF Wrap up

Random forests are popular among data scientists because:

- They are easily parallelized
- Low bias, and lower variance than something like a tree or bagging
- We can examine how important a feature was in predicting the data

Next

- Boosting





Thank
you