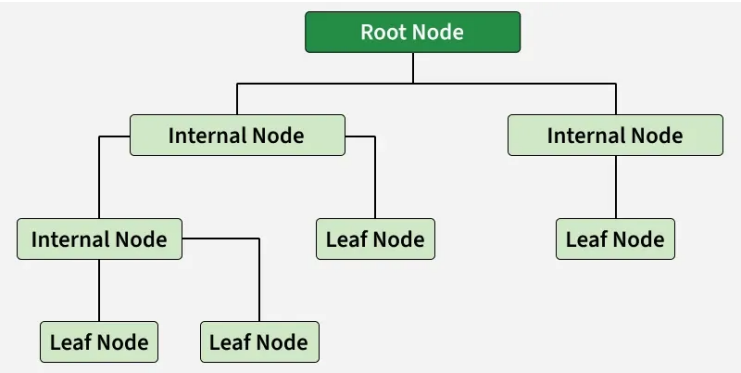
CART Regression Tree from Scratch

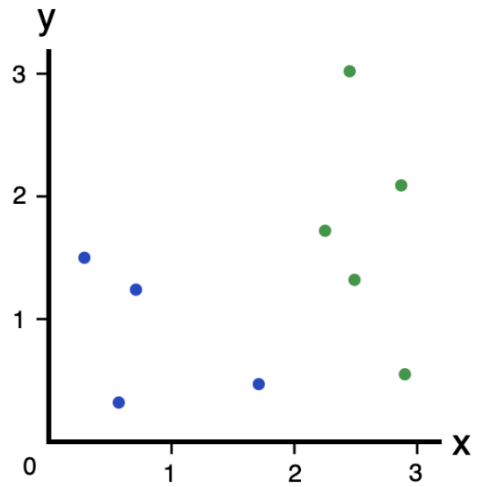
Team: λ = {Qiuli Lai, Dequan Zhang} ⊆ DSI ⊆ Brown University

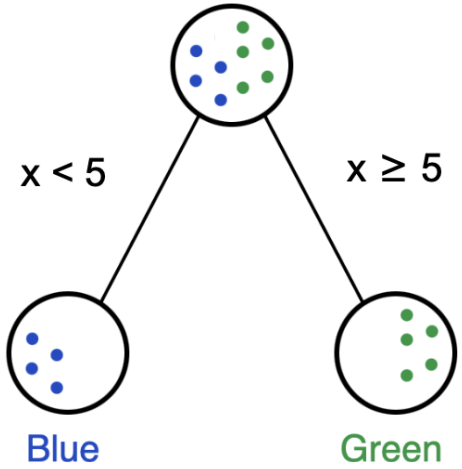
Github repository: https://github.com/DQZ25/Data2060-Final-Project

Preliminary

Def(Decision Tree). A decision tree is a predictor $h: \mathcal{X} \to \mathbf{Y}$ that assigns a label to an instance x by starting at the root node of a tree and following a sequence of feature-based tests until reaching a leaf node.







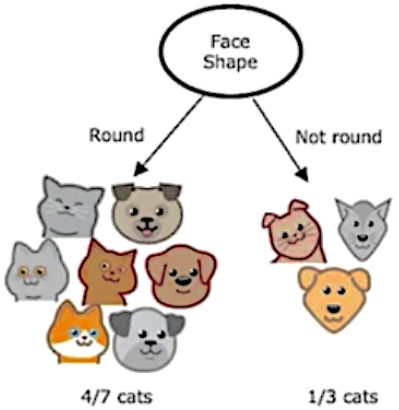
simple decision stump, split on threshold s = 5

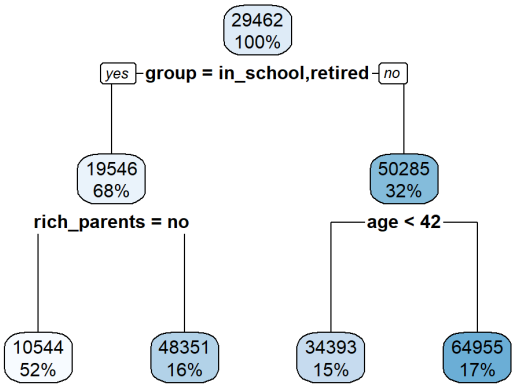
Preliminary

Classification and Regression Trees (CART), introduced by *Leo Breiman*, construct decision trees for classification or regression problems by recursively partitioning the input space through binary splits.

Classification Tree. A classification tree is a predictor $h : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{Y}$ is a finite set of class labels, represented by a rooted tree whose internal nodes specify tests of the form $x_j \leq t$. Each leaf node L_k is assigned a class label $c_k \in \mathcal{Y}$. For any input $x \in \mathcal{X}$, the prediction is $h(x) = c_k$.

Regression Tree. A regression tree is a predictor $h : \mathcal{X} \to R$, represented by a rooted tree whose internal nodes specify tests of the form $x_j \leq t$. Each leaf node L_k is associated with a constant value $c_k \in R$. For any input $x \in \mathcal{X}$, the prediction is $h(x) = c_k$.





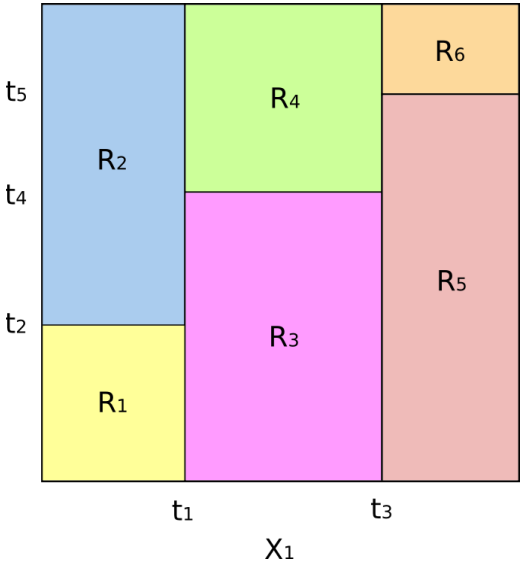
Representations defines a space of all possible programs

Regression Trees. The CART representation for regression is a function learned as a binary tree consisting of internal nodes, branches and terminal leaf nodes. Let the training dataset contain n samples and d features, written as $\{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$ with $x^{(i)} \in R^d$. Each internal node applies a univariate split of the form $x_j \leq t$, which recursively partitions the feature space into disjoint leaf regions $R_1, \ldots, R_M$. Each region R_m, where $m \in \{1, \ldots, M\}$, corresponds to a leaf node and stores the mean of the training responses assigned to that region.

For an input sample $x = (x_1, x_2, \ldots, x_d)$, the prediction is obtained by routing x from the root to its corresponding leaf region R_m and returning the leaf average

$$\hat{y}(x) = \frac{1}{|R_m|} \sum_{(x^{(i)}, y^{(i)}) \in R_m} y^{(i)}.$$

This expression shows that $\hat{y}(x)$ is the mean of the target values for all training samples that fall in the same leaf region R_m as the input x.



Loss Function/Impurity how to score the program

A common impurity criterion for CART regression is the Mean Squared Error (MSE). Given a leaf region R_m with prediction $\bar{y}_{R_m}$, the impurity of the leaf is defined as the average squared deviation of the target values from their mean:

$$I(R_m) = \frac{1}{|R_m|} \sum_{(x^{(i)}, y^{(i)}) \in R_m} \left(y^{(i)} - \bar{y}_{R_m} \right)^2 .$$

Squared

Error

Mean

target value

predicted value

Optimizer

CART Regression uses a greedy, top down procedure called recursive binary splitting. At each internal node the algorithm considers all features and all possible thresholds.

For a feature $x_j, j \in \{1, 2, ...d\}$, let its sorted unique values be

$$v_1 < v_2 < \cdots < v_K.$$

The candidate thresholds are the midpoints

$$s_k = \frac{v_k + v_{k+1}}{2}, \qquad k = 1, \ldots, K-1.$$

For each pair (j, s), the data are partitioned into two regions:

$$R_1 = \{x^{(i)} : x_j^{(i)} \leq s\}, \qquad R_2 = \{x^{(i)} : x_j^{(i)} > s\}.$$

The best constant prediction in each region is the sample mean:

$$\hat{c}_1 = \frac{1}{|R_1|} \sum_{(x^{(i)}, y^{(i)}) \in R_1} y^{(i)}, \qquad \hat{c}_2 = \frac{1}{|R_2|} \sum_{(x^{(i)}, y^{(i)}) \in R_2} y^{(i)}.$$

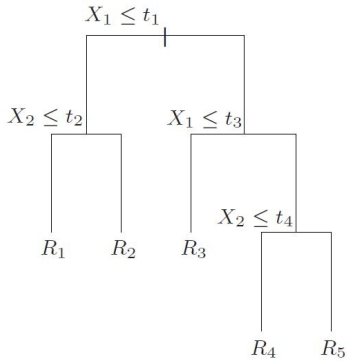
The total squared error for split (j, s) is the Residual Sum of Squares(RSS):

$$\text{Loss}(j, s) = \sum_{(x^{(i)}, y^{(i)}) \in R_1} \left(y^{(i)} - \hat{c}_1\right)^2 + \sum_{(x^{(i)}, y^{(i)}) \in R_2} \left(y^{(i)} - \hat{c}_2\right)^2.$$

CART selects the split that minimizes the objective:

$$\min_{j,s} \left[\sum_{(x^{(i)}, y^{(i)}) \in R_1} \left(y^{(i)} - \hat{c}_1\right)^2 + \sum_{(x^{(i)}, y^{(i)}) \in R_2} \left(y^{(i)} - \hat{c}_2\right)^2 \right].$$

The threshold that achieves the lowest loss across all candidate pairs (j, s) is chosen as the best split.

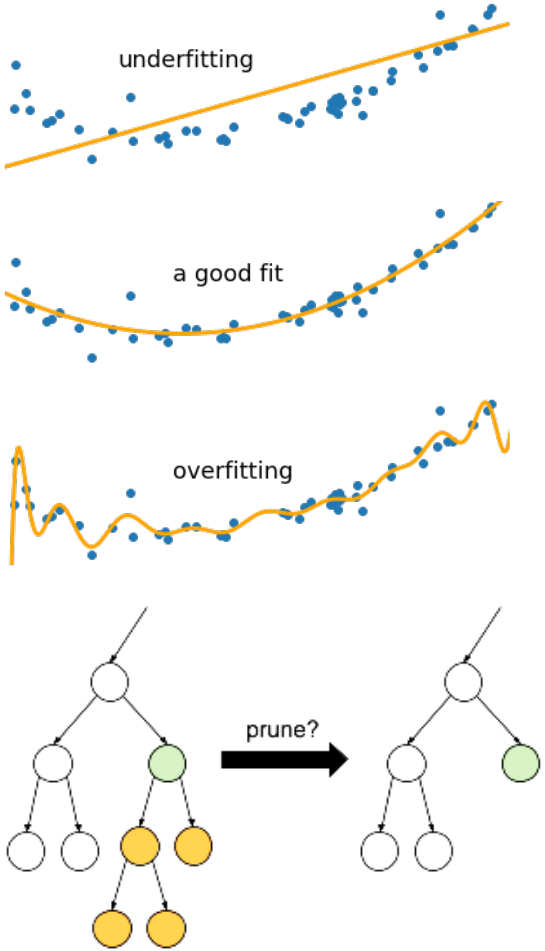


Regularization

Decision trees tend to grow very deep and overfit the training data.

Overfitting occurs when a model fits the training data too closely and performs poorly on new data.

Pruning controls this behavior by removing branches that do not add meaningful predictive value, which reduces complexity and improves the model's ability to generalize.



Regularization — Pruning

There are two kinds of pruning:
1. **Pre-pruning** (limit the growth of the tree during construction by any stop criteria)
2. **Post-pruning** (cutting-off unnecessary branches after complete tree construction)

(1) **Top-down pruning.** This strategy begins evaluation at the root and proceeds downward. At each node, the algorithm checks whether removing the corresponding subtree improves the chosen evaluation metric.Common variants include:

- **Pessimistic Error Pruning (PEP)**. Subtrees with sufficiently large estimated error are removed, based on a pre-specified threshold.
- **Critical Value Pruning (CVP).** Subtrees are removed when their informativeness is below a chosen critical value.

(2) **Bottom-up pruning.** This strategy begins evaluation at the leaves and proceeds upward. Each internal node is examined after both of its children have been evaluated, which allows a more accurate assessment of subtree contributions. This often yields better pruned trees, although the computational cost is higher. Typical variants include:

- **Minimum Error Pruning (MEP).** Among all possible pruned versions of the tree, the subtree with the lowest estimated error on a validation set is selected.
- **Reduced Error Pruning (REP)**. Internal nodes are replaced by leaves as long as the accuracy on a validation set does not decrease.
- **Cost-complexity pruning (CCP).** A sequence of subtrees is constructed by removing the weakest internal nodes.

Cost-complexity pruning (CCP)

Intuitively, we prune whenever removing the subtree reduces the cost more than it increases the risk.

1.2 Cost Complexity Definitions

- $T_{\max}$: the fully grown tree
- $R(t)$: risk (impurity or loss) of node t

$$R(T) = \sum_{t \in leaf nodes} R(t)$$

- T_t: subtree rooted at node t
- $|\tilde{T}|$: number of leaf nodes in the tree
- $|\tilde{T}_t|$: number of leaf nodes in subtree T_t

2 Cost Complexity Pruning

2.1 Objective

We choose a subtree T by minimizing

$$\min_{T \le T_{\max}} R(T) + \alpha|\tilde{T}|.$$

Cost Complexity Penality

Special cases:

$$\alpha = 0 \Rightarrow T_0 = T_{\max}, \qquad \alpha = \infty \Rightarrow T_\infty$$

is a stump.

2.2 Nested Subtrees

- The sequence of optimal subtrees is nested (CART monograph).
- As α increases, we only need to evaluate the nested sequence

$$T_\infty < \cdots < T_1 < T_0 = T_{\max}.$$

3 Pruning Rule

3.1 Pruning Criterion

Prune all child nodes of t if

$$(\tilde{T}_t - 1)\alpha > R(t) - R(T_t) \quad \Longleftrightarrow \quad \alpha > \frac{R(t) - R(T_t)}{\tilde{T}_t - 1}.$$

- Penalty: $(|\tilde{T}_t| - 1)\alpha$
- Reward: $R(t) - R(T_t)$
- If the tree is unconstrained, $R(T_t)$ tends to zero.

3.2 Algorithm

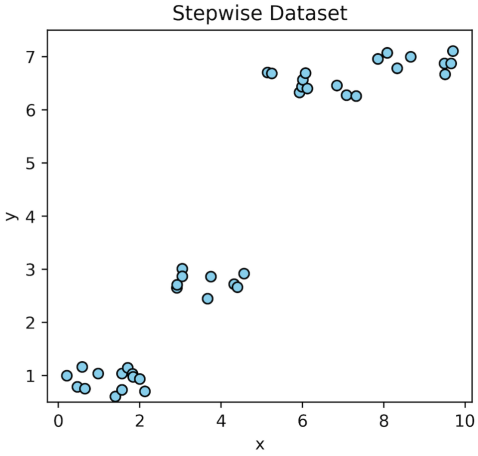
1. Compute the effective cost

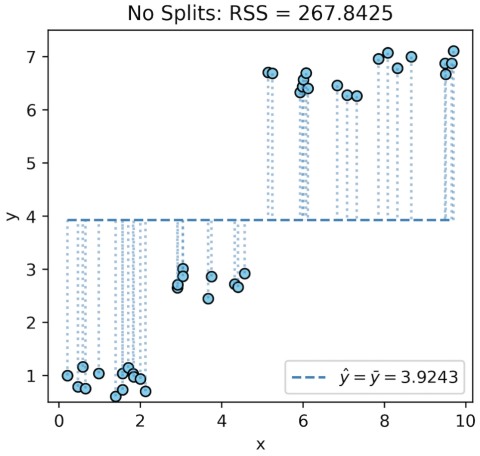
$$\alpha_t = \frac{R(t) - R(T_t)}{|\tilde{T}_t| - 1}$$

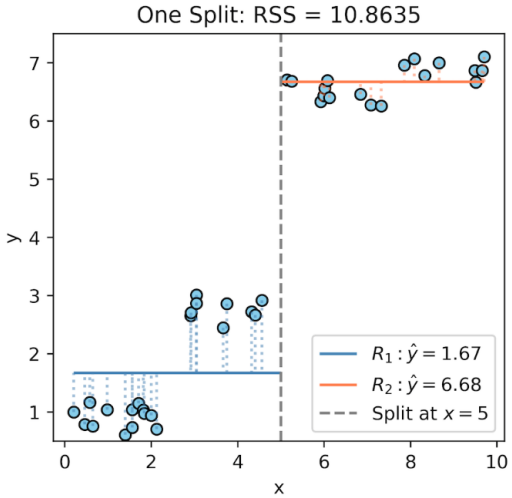
for every internal node t.

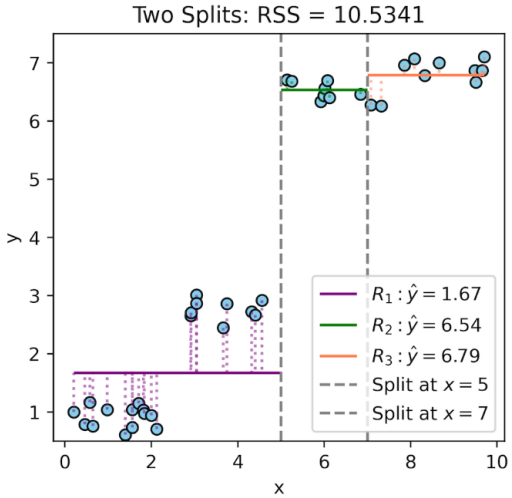
2. For a given α, prune node t (turn it into a leaf) when $\alpha_t < \alpha$.
3. Increase α and repeat until only one leaf remains.

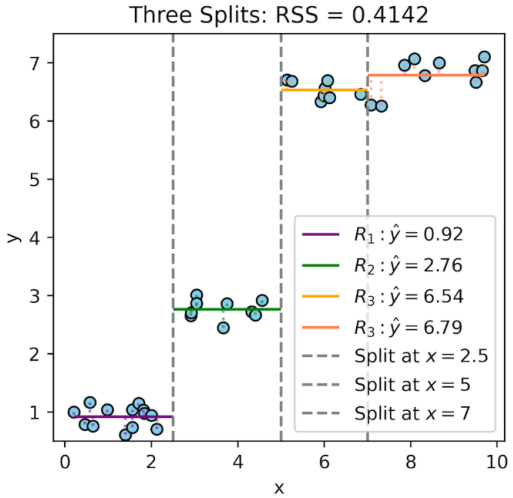
1-D Demo


Stepwise Dataset
x
y

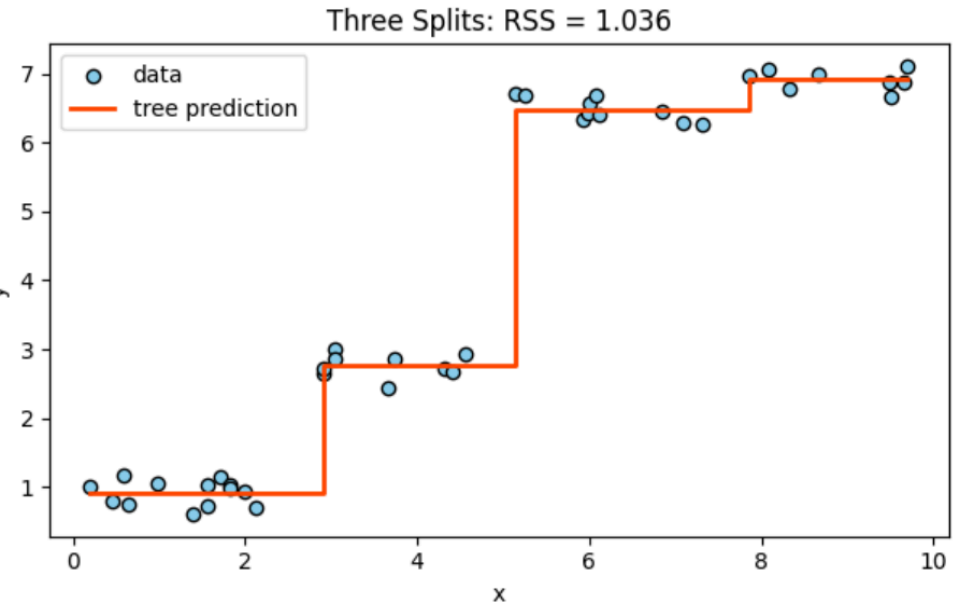

No Splits: RSS = 267.8425
$\hat{y} = \bar{y} = 3.9243$
x
y

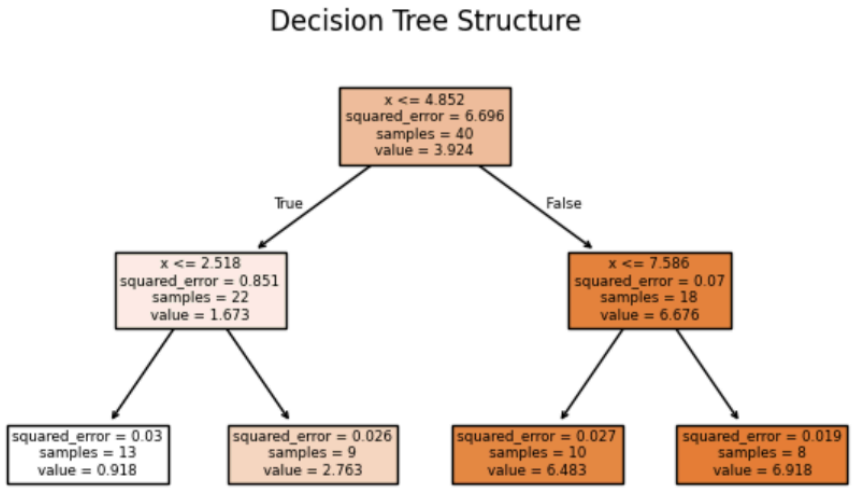

One Split: RSS = 10.8635
$R_1 : \hat{y} = 1.67$
$R_2 : \hat{y} = 6.68$
Split at $x = 5$
x
y


Two Splits: RSS = 10.5341
$R_1 : \hat{y} = 1.67$
$R_2 : \hat{y} = 6.54$
$R_3 : \hat{y} = 6.79$
Split at $x = 5$
Split at $x = 7$
x
y


Three Splits: RSS = 0.4142
$R_1 : \hat{y} = 0.92$
$R_2 : \hat{y} = 2.76$
$R_3 : \hat{y} = 6.54$
$R_3 : \hat{y} = 6.79$
Split at $x = 2.5$
Split at $x = 5$
Split at $x = 7$
x
y

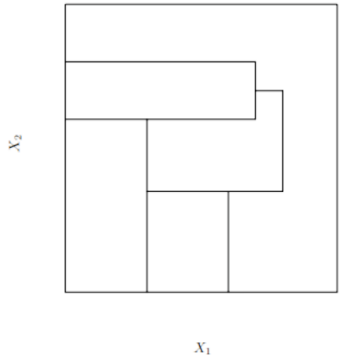
```
from sklearn.tree import DecisionTreeRegressor, plot_tree
```

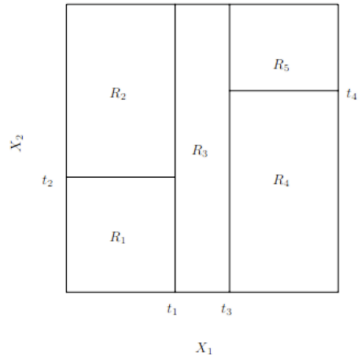




2-D case

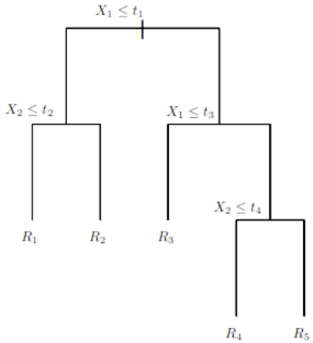
irregular, non-axis-aligned cut which violate the structure imposed by DT

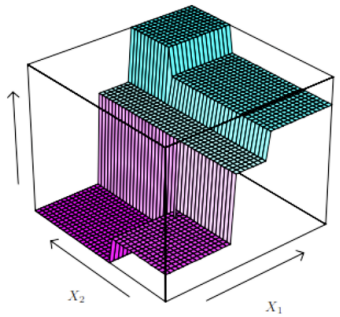




In contrast, the top-right panel shows a valid CART partition. Each region R1 → R5 is formed by recursive binary splits, where each cut is along a single feature dimension at a time

DT structure





Prediction surface visualized

FIGURE 8.3. Top Left: *A partition of two-dimensional feature space that could not result from recursive binary splitting.* Top Right: *The output of recursive binary splitting on a two-dimensional example.* Bottom Left: *A tree corresponding to the partition in the top right panel.* Bottom Right: *A perspective plot of the prediction surface corresponding to that tree.*

Pseudo Code

input:

--training set $S = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$, n sampels with d features.

--if all target values in S are identical return $\frac{1}{|S|}\sum_{i=1}^{m} y^{(i)}$

--if $|S| = 0$ return a leaf with the parent mean target

for each $j \in \{1, \dots, d\}$

--V be the sorted unique values of feature x_j in S

--**for** (v_r, v_{r+1}) in V

----Compute threshold $t = \frac{v_r + v_{r+1}}{2}$

----Split the data $R_1 = \{(x, y) \in S : x_j \leq t\}$

----Split the data $R_2 = \{(x, y) \in S : x_j > t\}$

----If either subset is empty, skip this threshold.

----Compute cost $\mathrm{C}(j, t) = RSS(R_1) + RSS(R_2)$

--Select the best split (j^*, t^*) by $\arg\min_{j,t} \mathrm{Cost}(j, t)$

--Partition the data $S_1 = \{(x, y) \in S : x_{j^*} \leq t^*\}$

--Partition the data $S_2 = \{(x, y) \in S : x_{j^*} > t^*\}$

--Recursively build subtrees $T_1 = \mathrm{CART\_REGRESSION}(S_1)$

--Recursively build subtrees $T_2 = \mathrm{CART\_REGRESSION}(S_2)$

--**for each** internal node t

----Compute complexity parameter $\alpha_t = \frac{R(t) - R(T_t)}{|T_t| - 1}$

----Prune with smallest α_t , $R_\alpha(T) = R(T) + \alpha|T|$ and repeat

----choose the best tree with minimum cost-complexity

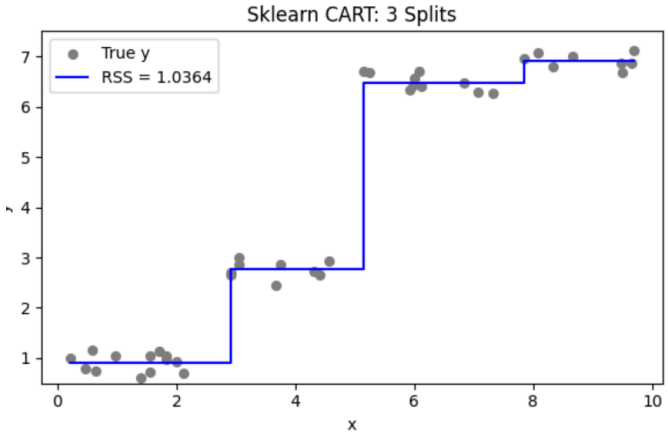
output:

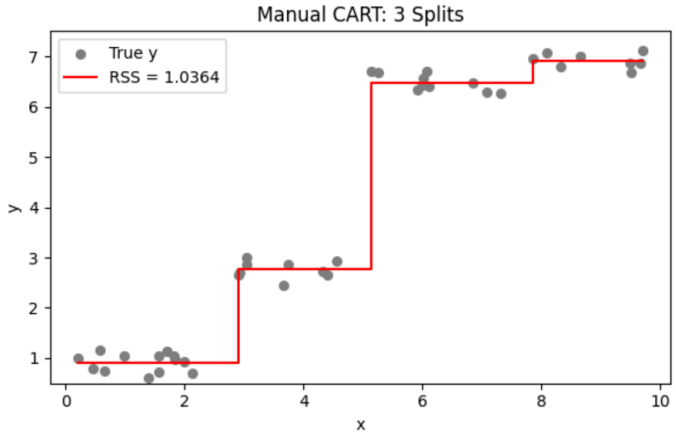
--test $x_{j^*} \leq t^*$ and pruned tree.

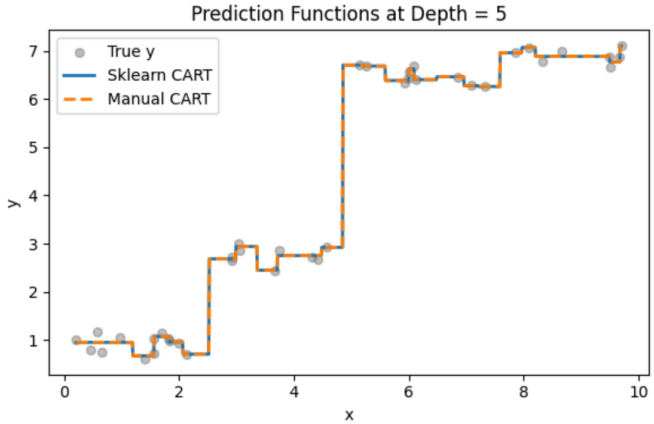
Simple check passed

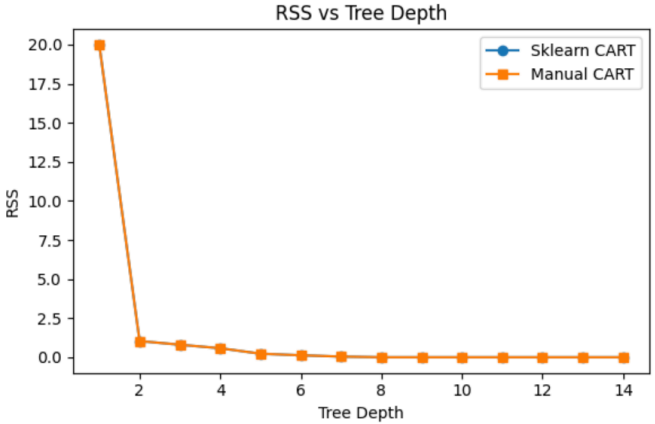
```
--- START OF TREE ---
[X0 <= 900.000] impurity = 1700.000, n = 4
├── Left:
  [X0 <= 700.000] impurity = 0.000, n = 2
  ├── Left:
    Leaf: predict = 120.000, impurity = 0.000, n = 1
  └── Right:
    Leaf: predict = 150.000, impurity = 0.000, n = 1
└── Right:
  [X0 <= 1100.000] impurity = 0.000, n = 2
  ├── Left:
    Leaf: predict = 200.000, impurity = 0.000, n = 1
  └── Right:
    Leaf: predict = 250.000, impurity = 0.000, n = 1
---- END OF TREE ----
```

Result 1


Sklearn CART: 3 Splits
True y
RSS = 1.0364
x


Manual CART: 3 Splits
True y
RSS = 1.0364
x
y

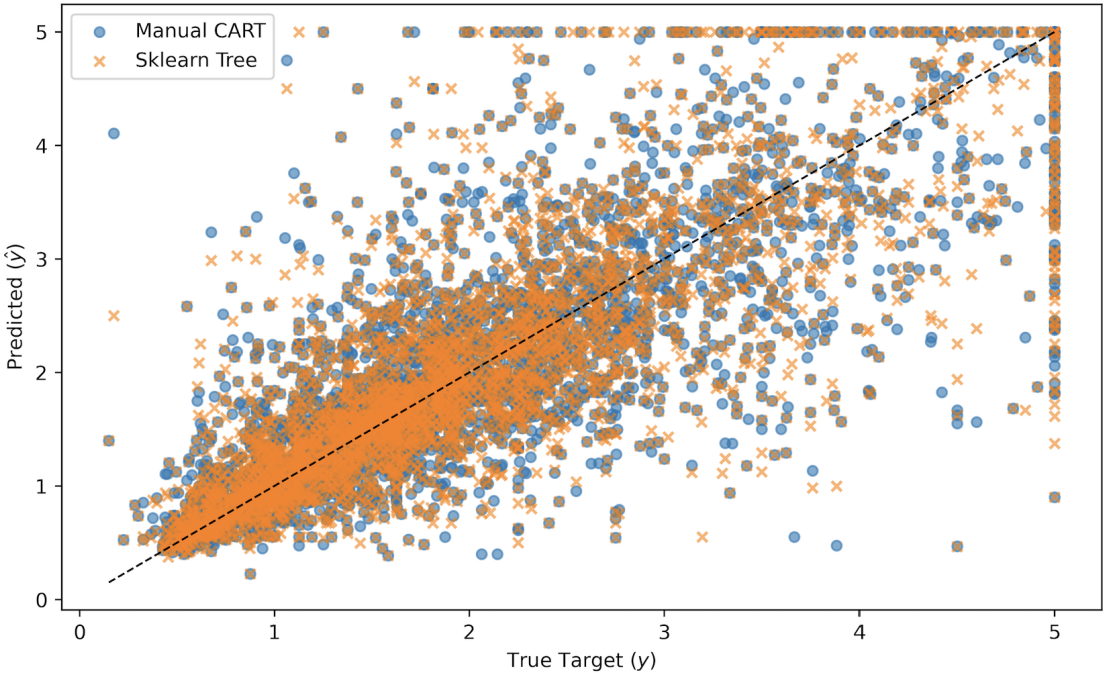

Prediction Functions at Depth = 5
True y
Sklearn CART
Manual CART
x
y


RSS vs Tree Depth
Sklearn CART
Manual CART
RSS
Tree Depth

Result 2

Apply both our manual implementation of a CART regression tree and Scikit-learn's `DecisionTreeRegressor` to the California Housing dataset.

Our implementation uses greedy recursive splitting based on weighted MSE and includes cost-complexity pruning logic. We set ccp_alpha=0.0 and max_depth=40 to match sklearn's settings for a fair comparison.



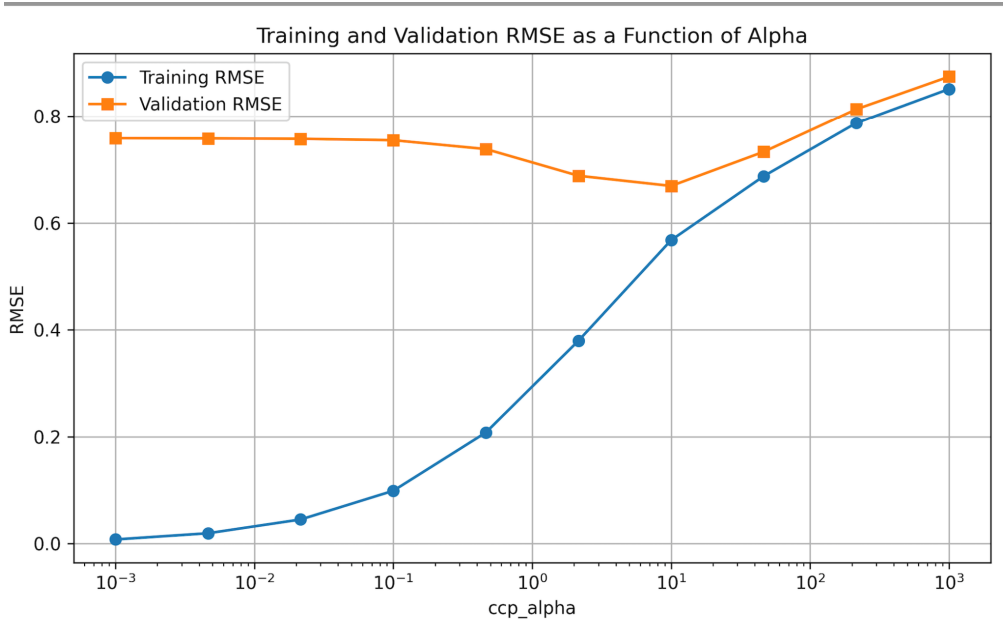
Manual CART: RSS = 2048.0247, RMSE = 0.7044, Leaves = 15806
Sklearn CART: RSS = 2062.7911, RMSE = 0.7069, Leaves = 15861

Alpha Parameter Tuning

Alpha too small: The tree is complex, **overfits**, and validation RMSE is very high.

Alpha too large: The tree becomes **underfit**, the validation RMSE **raise** again.

Therefore, the best pruning strength lies in the middle of the tested range, where the validation RMSE is lowest. So, we can choose from the range **1 to 10** as our alpha for this situation.



Summary

Based on all these experiments, which are conceptual demonstrations, model building, model testing, pruning validation, and real-world dataset evaluation, we proved the custom **CART regression** model is both theoretically correct and practically reliable.

Limitations

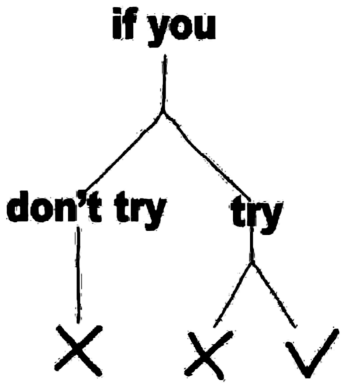
- **Overfitting**. Deep trees can capture noise instead of structure, which harms generalization.
- **Instability**. Small changes in the training data can lead to large changes in the learned tree.-
- **Piecewise constant outputs**. Predictions are stepwise and cannot extrapolate beyond observed ranges.
- **Imbalanced classification bias**. CART may favor majority classes when class distributions are skewed.
- **Greedy splitting**. Local impurity reduction does not ensure a globally optimal tree.

Advantages

- **Scalability**. Efficient on large datasets.
- **Interpretability**. The tree structure is easy to understand and visualize, and remains interpretable even when model assumptions do not hold.
- **Support for missing values**. Some implementations can split on features with missing entries.
- **Versatility**. Works with both numerical and categorical variables.
- **Multiclass capability**. Naturally handles multiclass outcomes.
- **Feature importance**. Provides impurity-based measures of feature relevance.

Interesting Things

- CART builds a piecewise-constant model and use simple structure to approximate the highly nonlinear relationship
- Cost-Complexity Pruning lets us use different parameter alpha to choose optimal model and makes us to better communicate the trees.



Difficulties

- Implementing RSS-based split require careful handling with the edge situations, like empty set and pure nodes.
- The bottom-up CCP pruning require many computation.

Thank you for listening.