15.095: Machine Learning under a Modern Optimization Lens

Lecture 5: Optimal Classification Trees

Motivation

- State-of-the-art: Decision trees are formed using greedy heuristics
- Can we use modern optimization techniques to develop approaches that find **globally optimal decision trees**?
- Are these Optimal Trees effective and do they **add value** over current methods?
- Is the Optimal Tree problem **tractable** and does this approach **scale**?

Outline

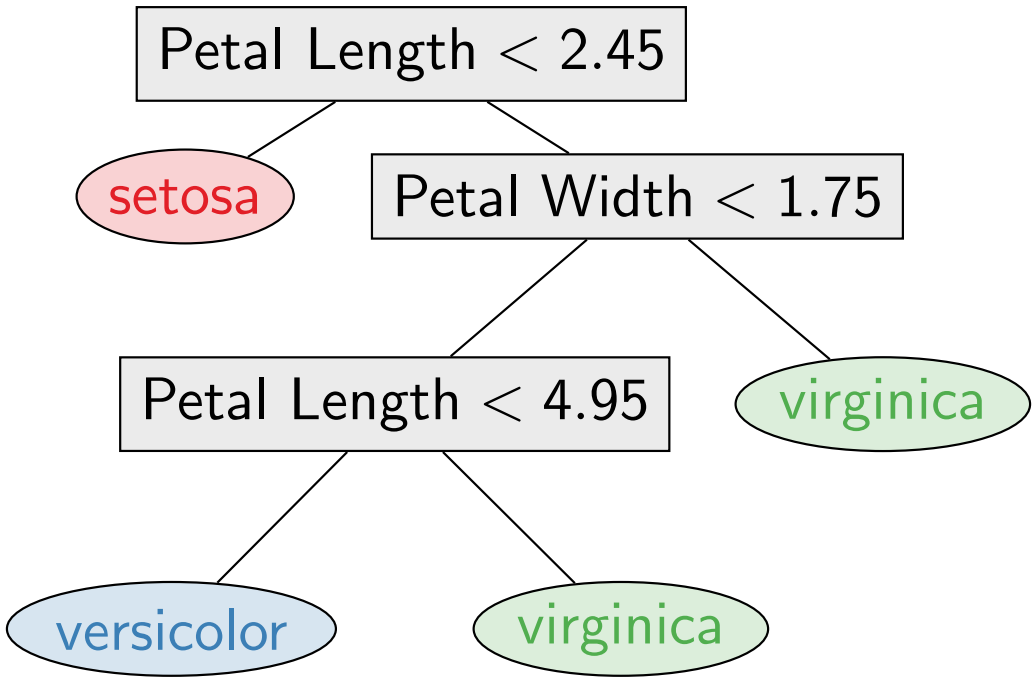
Outline

Classification

- Classification is a key problem in Machine Learning
  - Given training data $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$, we want to learn a function for predicting y based on $\mathbf{x}$
  - $\mathbf{x}_i \in \mathbb{R}^p$ are the features of the data
  - $y_i \in \{1, \ldots, K\}$ are the labels. $K = 2 \implies$ binary classification
- **Example:** Fisher's Iris dataset from UCI Machine Learning Repository
  - One of the most well-known machine learning datasets
  - 150 iris flowers of three different types
  - Four measurements for each flower: petal width/height and sepal width/height
  - Task: Predict the iris type using the measurements

Decision Trees

- Decision trees are one of the most-widely used techniques in machine learning
  - Create a recursive partitioning of the features to classify points
  - Decision trees are **interpretable**



CART — Classification and Regression Trees

- CART (Breiman et al, 1984) is a state-of-the-art decision tree learner
- Widespread use in academia and industry ($\sim$ 38,000 citations!)

CART — Classification and Regression Trees

- CART (Breiman et al, 1984) is a state-of-the-art decision tree learner
- Widespread use in academia and industry ($\sim$ 38,000 citations!)

- One of the most impactful works in our field:

CART — Classification and Regression Trees

- CART (Breiman et al, 1984) is a state-of-the-art decision tree learner
- Widespread use in academia and industry ($\sim$ 38,000 citations!)

- One of the most impactful works in our field:
  - **Dantzig**—Linear Programming and Extensions

CART — Classification and Regression Trees

- CART (Breiman et al, 1984) is a state-of-the-art decision tree learner
- Widespread use in academia and industry ($\sim$ 38,000 citations!)

- One of the most impactful works in our field:
  - **Dantzig**—Linear Programming and Extensions 9,000

CART — Classification and Regression Trees

- CART (Breiman et al, 1984) is a state-of-the-art decision tree learner
- Widespread use in academia and industry ($\sim$ 38,000 citations!)

- One of the most impactful works in our field:
  - **Dantzig**—Linear Programming and Extensions 9,000
  - **Bellman**—Dynamic programming and Lagrange multipliers

CART — Classification and Regression Trees

- CART (Breiman et al, 1984) is a state-of-the-art decision tree learner
- Widespread use in academia and industry ($\sim$ 38,000 citations!)

- One of the most impactful works in our field:
  - **Dantzig**—Linear Programming and Extensions 9,000
  - **Bellman**—Dynamic programming and Lagrange multipliers 18,000

CART — Classification and Regression Trees

- CART (Breiman et al, 1984) is a state-of-the-art decision tree learner
- Widespread use in academia and industry ($\sim$ 38,000 citations!)

- One of the most impactful works in our field:
  - **Dantzig**—Linear Programming and Extensions 9,000
  - **Bellman**—Dynamic programming and Lagrange multipliers 18,000
  - **Karp**—Reducibility among Combinatorial Problems

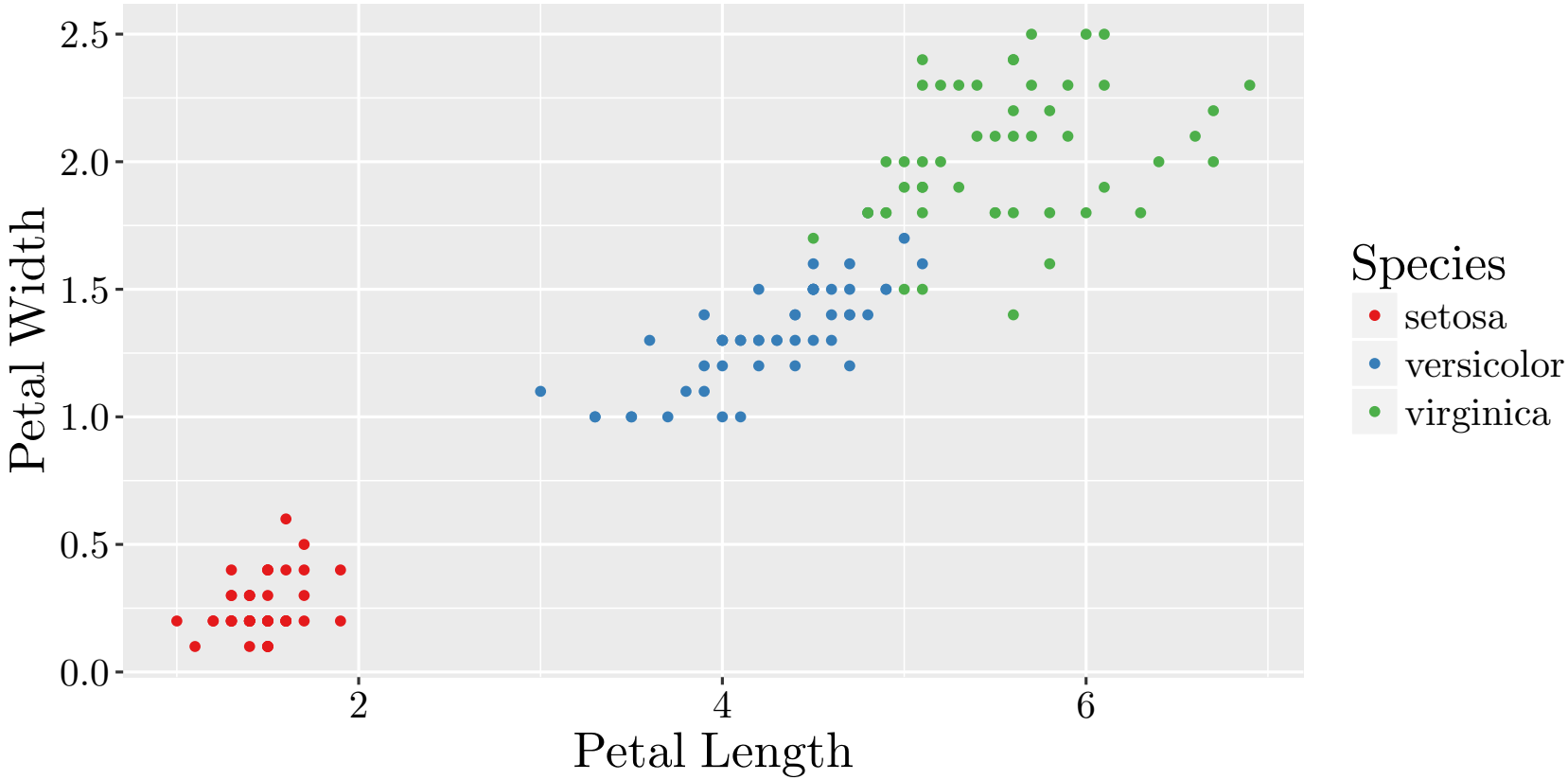
CART — Classification and Regression Trees

- CART (Breiman et al, 1984) is a state-of-the-art decision tree learner
- Widespread use in academia and industry ($\sim$ 38,000 citations!)

- One of the most impactful works in our field:
  - **Dantzig**—Linear Programming and Extensions 9,000
  - **Bellman**—Dynamic programming and Lagrange multipliers 18,000
  - **Karp**—Reducibility among Combinatorial Problems 10,000

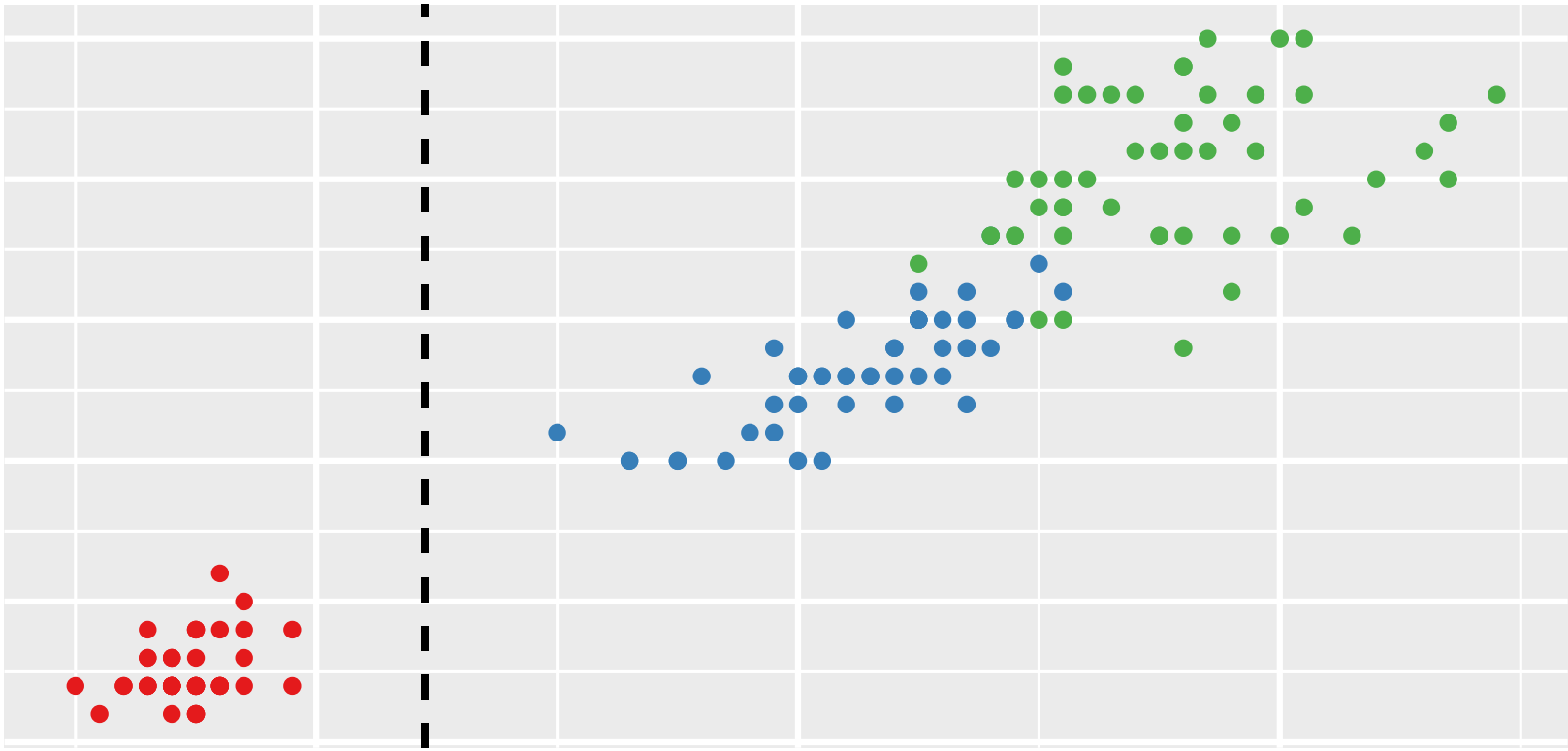
How CART works

- Each split branches on the value of on a single feature
- Greedy approach to partitioning—make a locally optimal split then recurse on both children



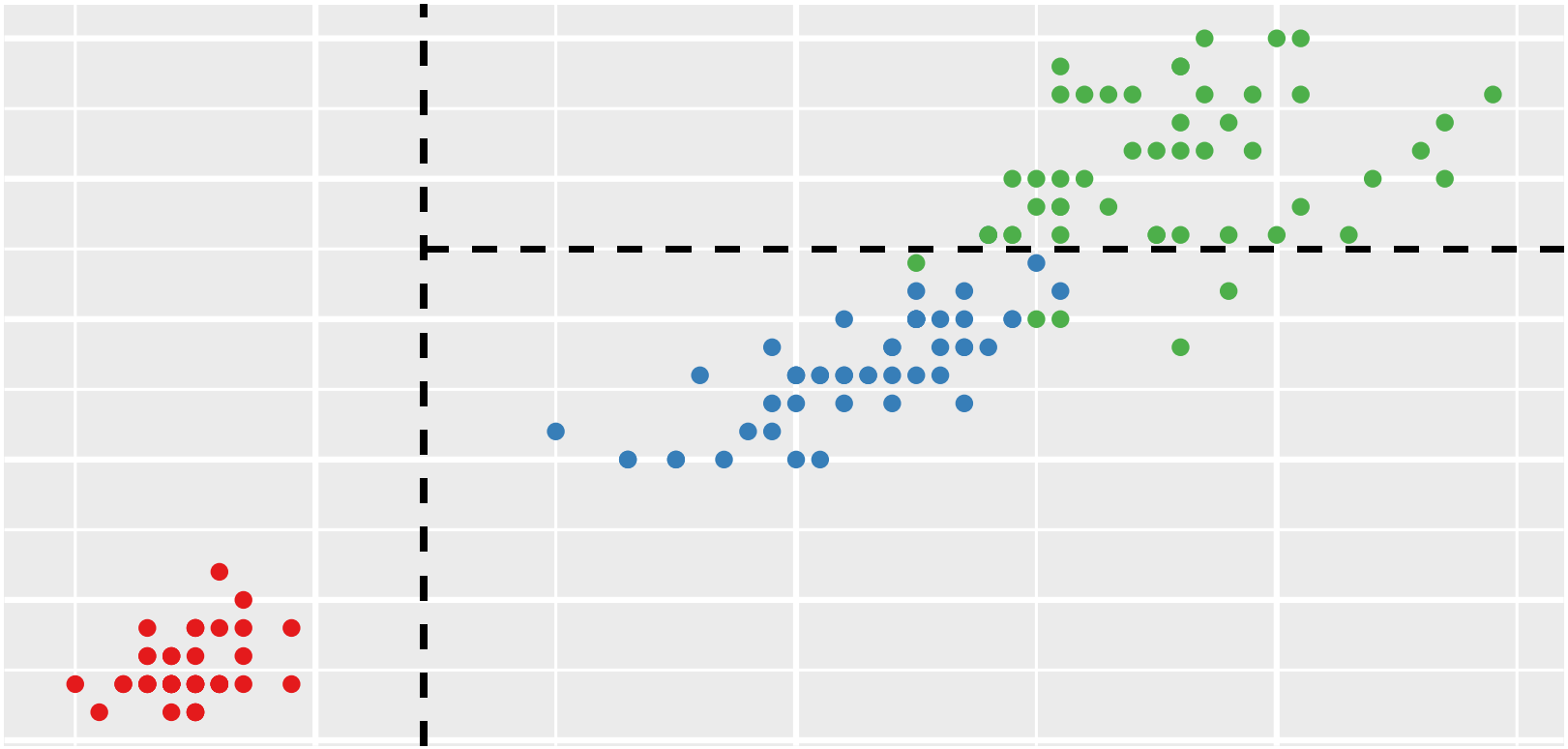
How CART works

- Each split branches on the value of on a single feature
- Greedy approach to partitioning—make a locally optimal split then recurse on both children



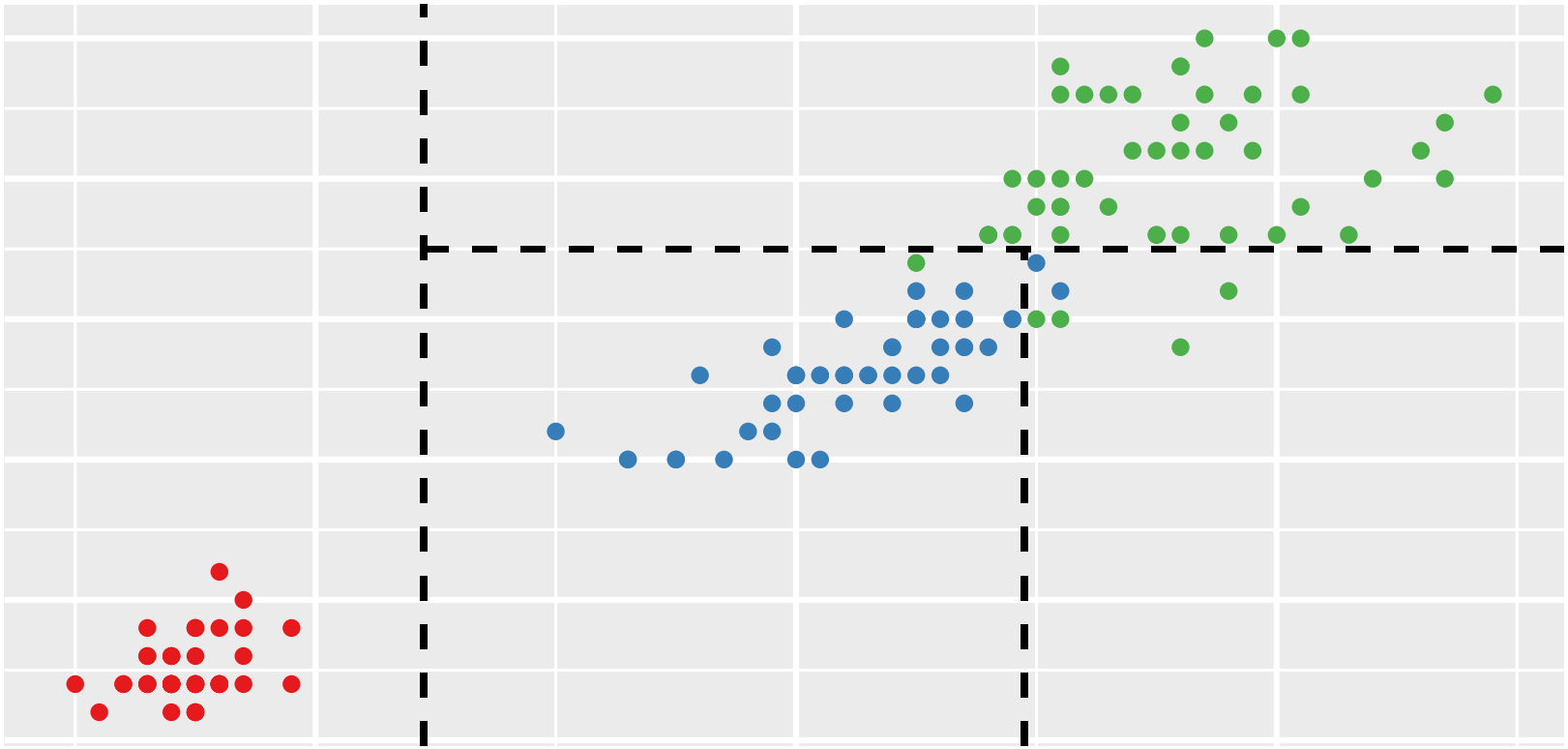
How CART works

- Each split branches on the value of on a single feature
- Greedy approach to partitioning—make a locally optimal split then recurse on both children



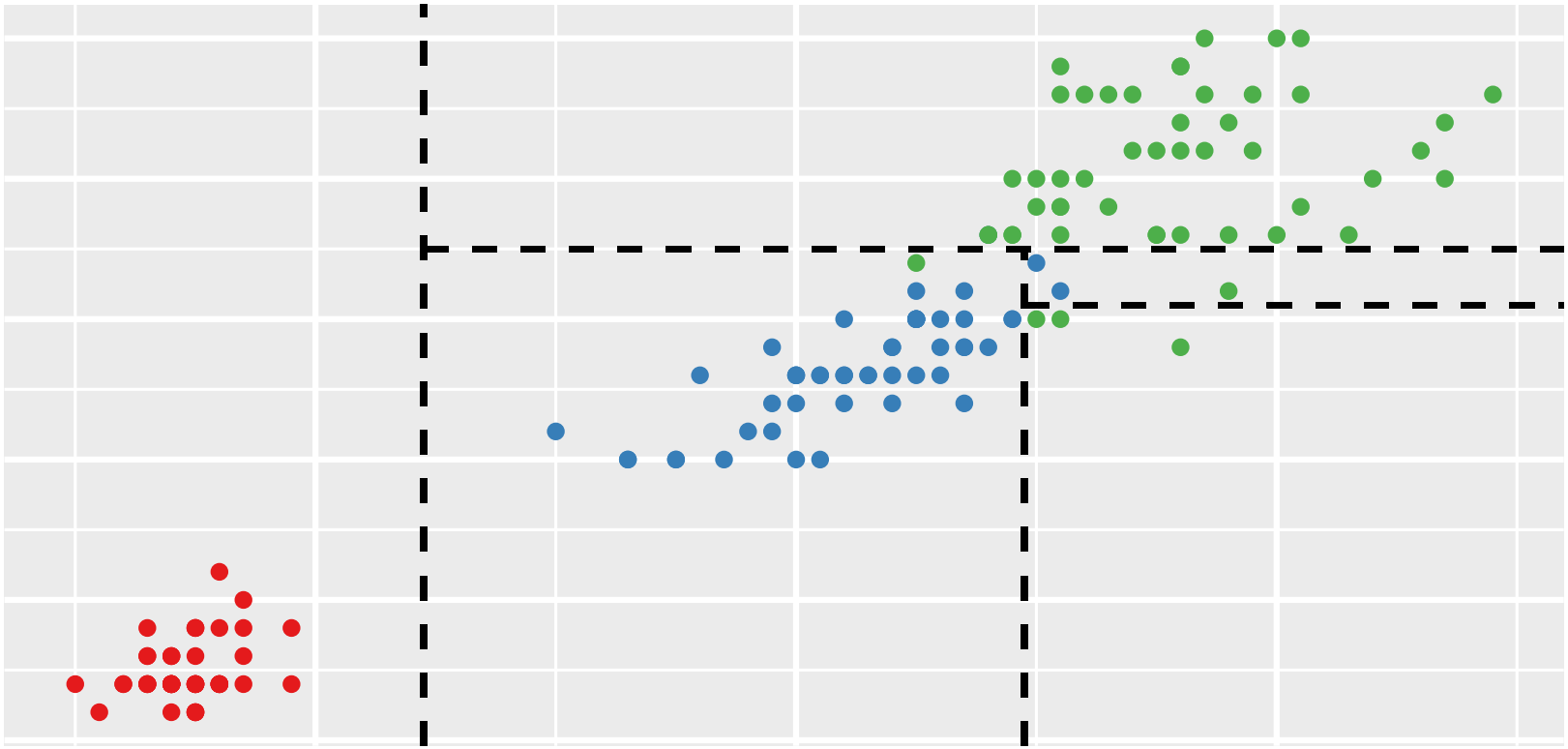
How CART works

- Each split branches on the value of on a single feature
- Greedy approach to partitioning—make a locally optimal split then recurse on both children



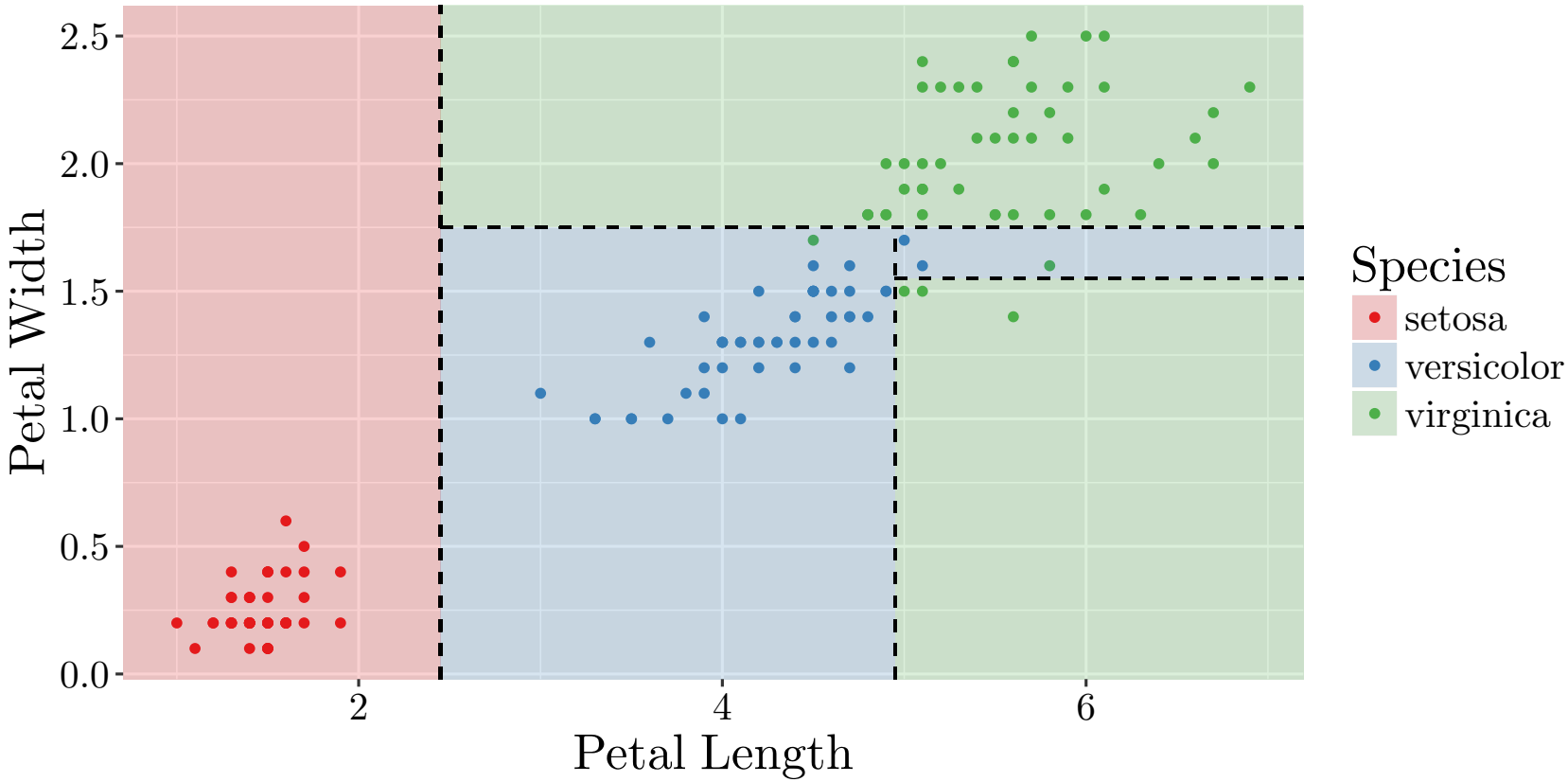
How CART works

- Each split branches on the value of on a single feature
- Greedy approach to partitioning—make a locally optimal split then recurse on both children



How CART works

- Each split branches on the value of on a single feature
- Greedy approach to partitioning—make a locally optimal split then recurse on both children



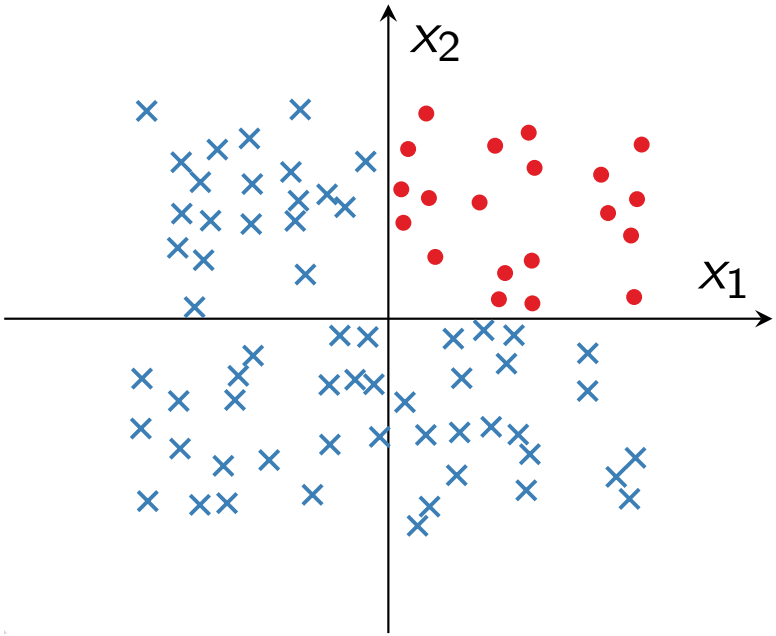
When to stop growing the tree?

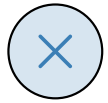
- Too many splits likely leads to overfitting on training data
- Control the complexity of the tree with a regularizer:

$$\min \ \text{error} + \alpha * \text{complexity}$$

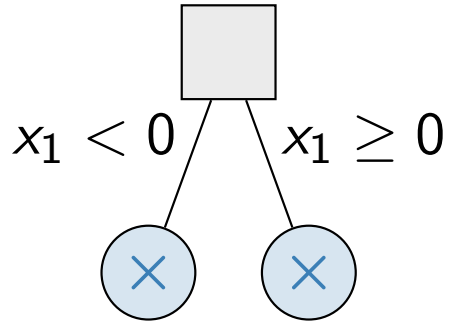
- **Problem:** Greedy methods will only consider a split if it reduces the error by more than α
  - Strong splits can be *hidden* behind weaker splits
  - Might miss good solutions if α too large (early stopping)
  - Might overfit if α is too small

Example of early stopping for greedy methods

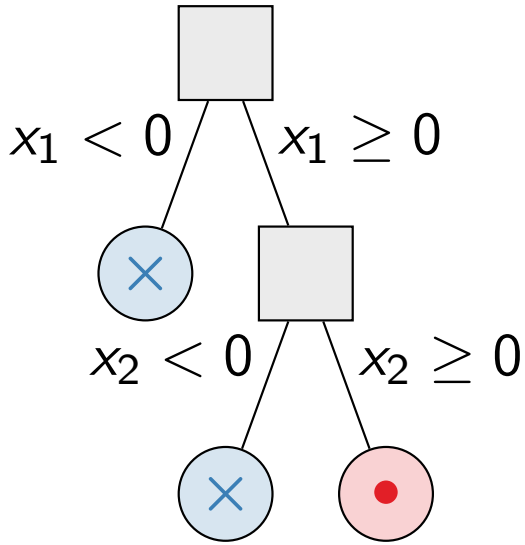




$\text{obj} = 0.25 + 0\alpha$



$\text{obj} = 0.25 + 1\alpha$



$\text{obj} = 0 + 2\alpha$

Fix the problem by growing then pruning

- **Solution:** Two-step process — Grow then prune
- **Step 1: Growing**
  - Choose split that locally maximizes an impurity measure
  - Repeat recursively until no more splits possible
  - Grows a deep tree that probably overfits training data
- **Step 2: Pruning**
  - Work back up the tree and remove splits that do not reduce misclassification error by at least α
- Prevents a weak split hiding a strong split

Why use an impurity measure?

- CART uses a nonlinear impurity measure (like *gini* or *twoing* criteria) to grow the tree greedily
- Trees grown to optimize impurity measure won't necessarily optimize misclassification error
- Why use an impurity measure rather than misclassification error?

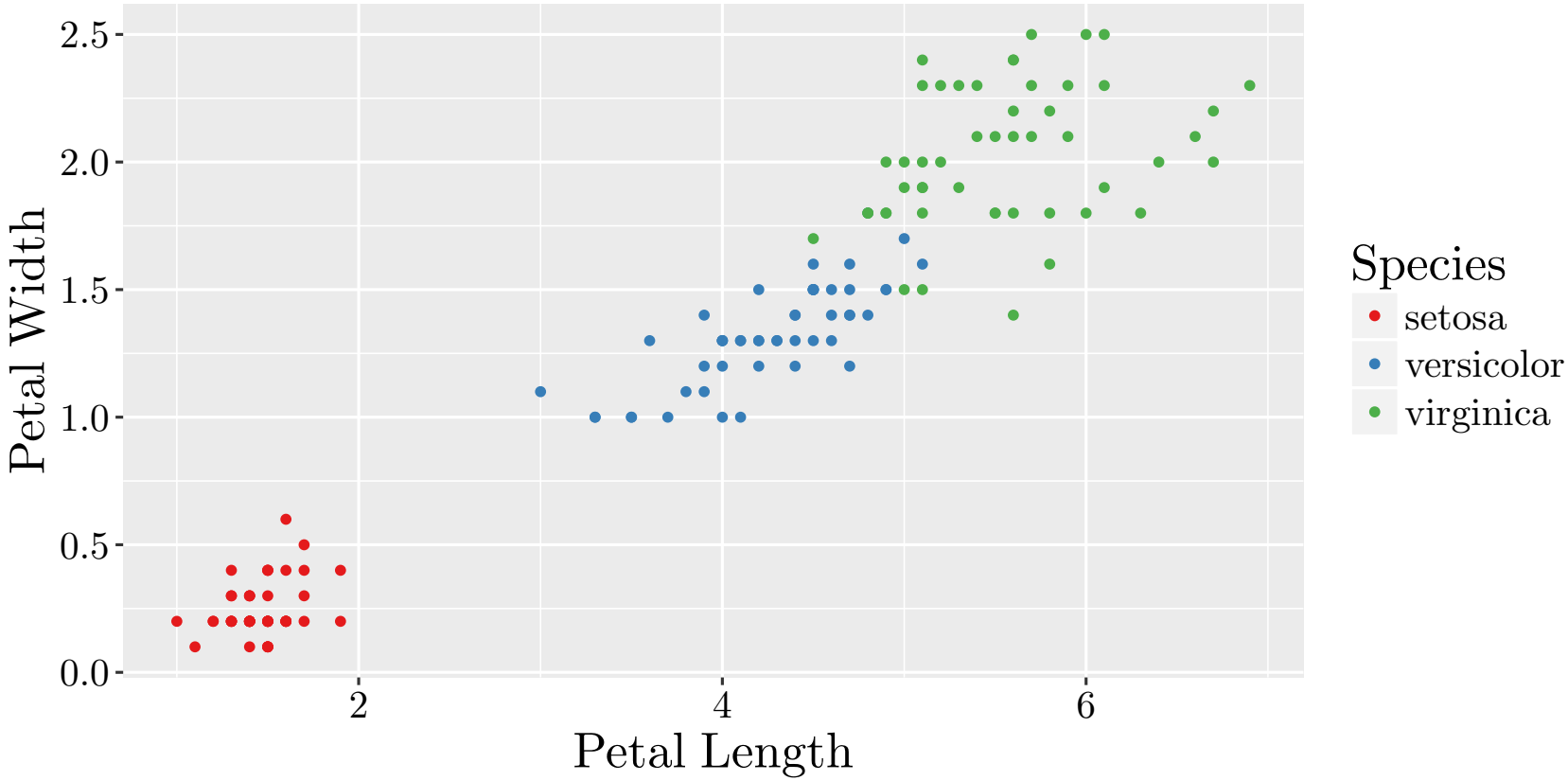
Why use an impurity measure?

- CART uses a nonlinear impurity measure (like *gini* or *twoing* criteria) to grow the tree greedily
- Trees grown to optimize impurity measure won't necessarily optimize misclassification error
- Why use an impurity measure rather than misclassification error?

> *. . . the [misclassification] criterion does not seem to appropriately reward splits that are more desirable in the context of the continued growth of the tree. . . . This problem is largely caused by the fact that our tree growing structure is based on a one-step optimization procedure.*
>
> — Breiman et al. (1984)

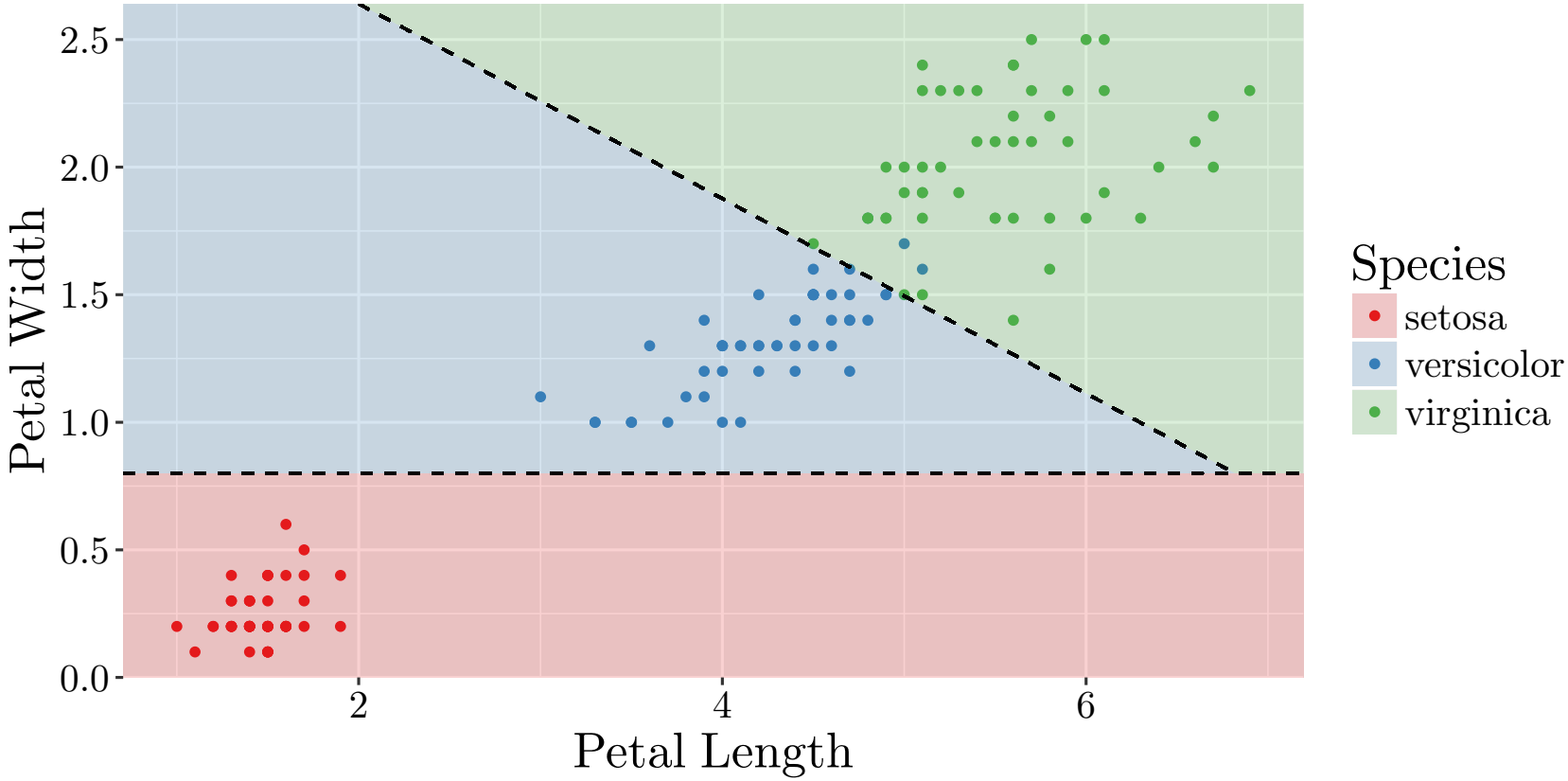
Extension—Hyperplane Splits

- CART and other methods require splits to be parallel to axes
- Using hyperplane splits can be more natural
  - Referred to as **oblique decision trees**
  - No good way to construct these in the literature: exponentially many hyperplanes to consider



Extension—Hyperplane Splits

- CART and other methods require splits to be parallel to axes
- Using hyperplane splits can be more natural
  - Referred to as **oblique decision trees**
  - No good way to construct these in the literature: exponentially many hyperplanes to consider



Addressing the limitations of CART

- CART's one-step growing procedure leads to complications:
  - Splits are only locally-optimal, overall tree could be far from optimal
  - Have to grow and prune tree in two-stage process
  - Have to use impurity measure while choosing splits

- What if we could solve the entire decision problem at once to find **globally optimal** trees instead?

- There have been many attempts to find methods for globally optimal decision trees in the literature, using a variety of techniques

- To date, there has not been a globally optimal decision tree method that is tractable and is able to scale to the typical problem sizes seen in classification

Breiman's take on globally optimal trees

> *Finally, another problem frequently mentioned (by others, not by us) is that the tree procedure is only one-step optimal and not overall optimal. . . . If one could search all possible partitions . . . the two results might be quite different.*
>
> *We do not address this problem. At this stage of computer technology, an overall optimal tree growing procedure does not appear feasible for any reasonably sized data set.*
>
> — Breiman et al. (1984)

- CART's use of locally-optimal splits rather than globally-optimal was guided by practical limitations at the time.

Progress of MIO in the past 25 years

- Speed up between CPLEX 1.2 (1991) and CPLEX 11 (2007): 29,000 times
- Gurobi 1.0 (2009) comparable to CPLEX 11
- Speed up between Gurobi 1.0 and Gurobi 7.5 (2017): 73 times
- Total software speedup: 2,117,000 times
- Hardware speedup: $10^{6.2}$= 1,560,000 times
- Total Speedup: **3.3 trillion times!**

Our approach

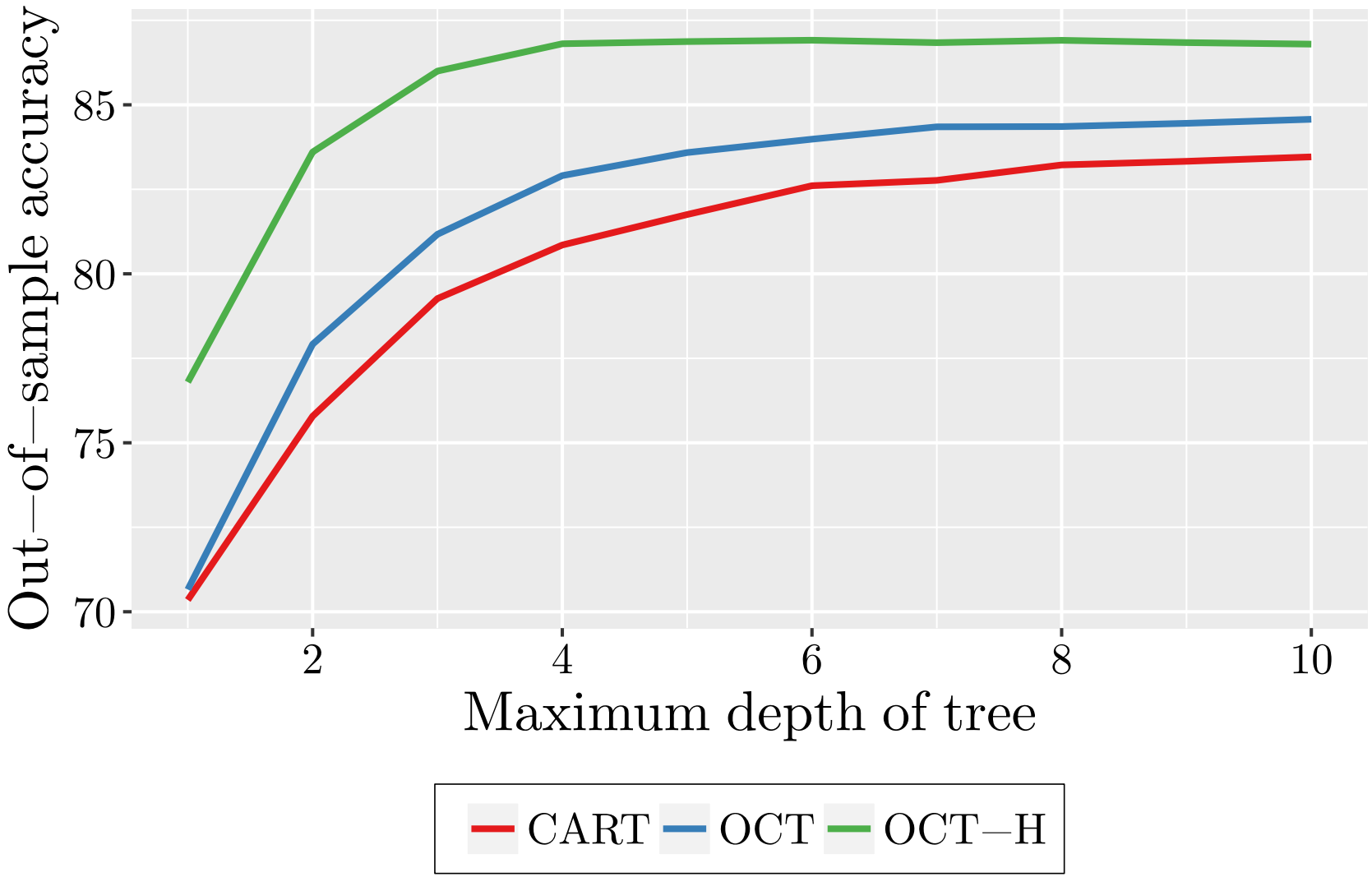
- Given the progress and speedups in Mixed-Integer Optimization (MIO), can we now construct a method for finding globally optimal decision trees?

- **Aspirations:** We seek new decision trees that:
  - Are globally optimal (or near-optimal)
  - Can be found in times appropriate for the application

- **Big question:** Will these trees lead to better results in practice?

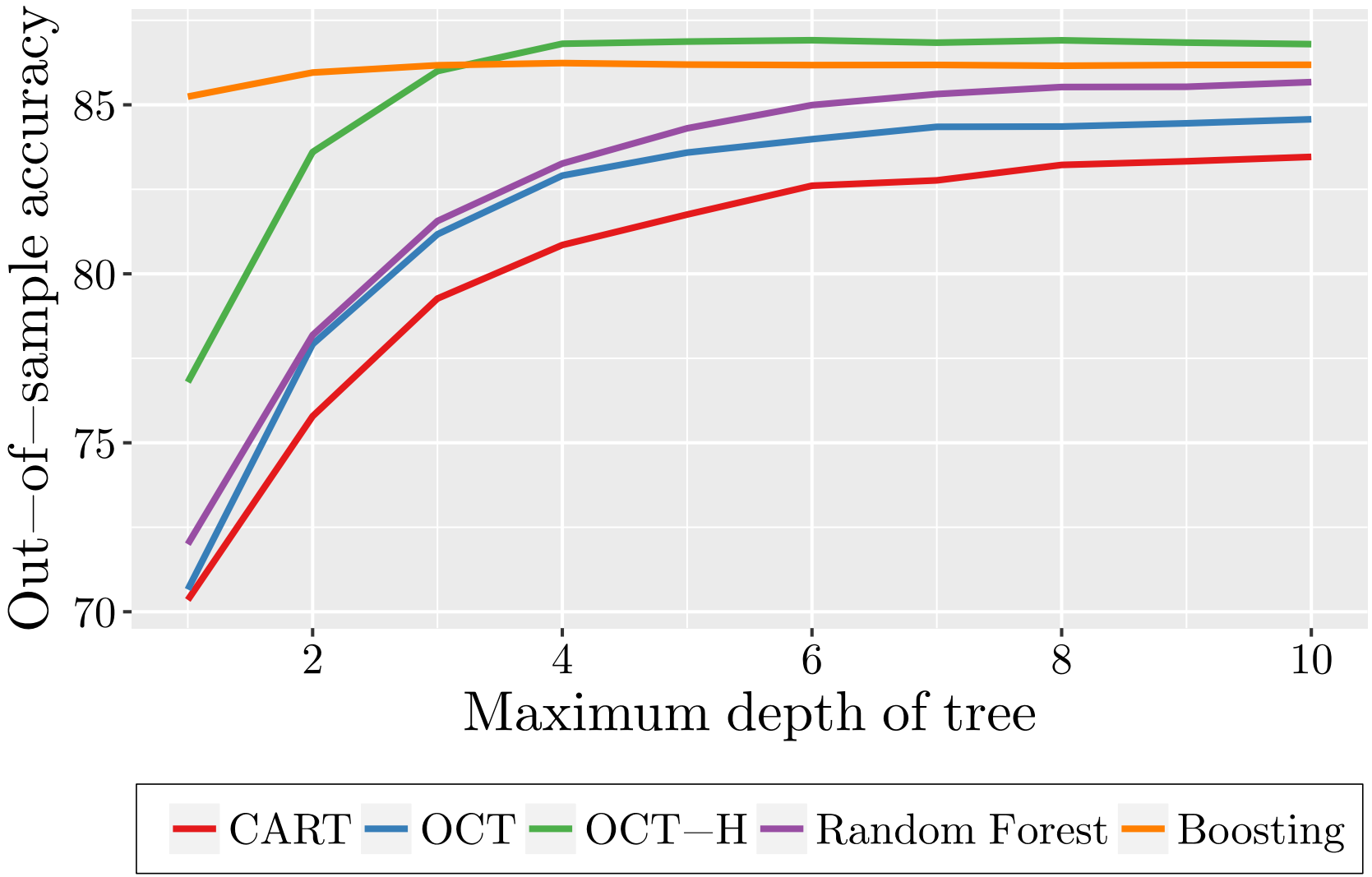
Performance of Optimal Classification Trees

- Average out-of-sample accuracy across 60 real-world datasets:



Performance of Optimal Classification Trees

- Average out-of-sample accuracy across 60 real-world datasets:



Outline

Finding Optimal Classification Trees

- Given training data $(\mathbf{x}_i, y_i), i = 1, \ldots, n$, want to find tree $\mathbb{T}$ that solves the problem

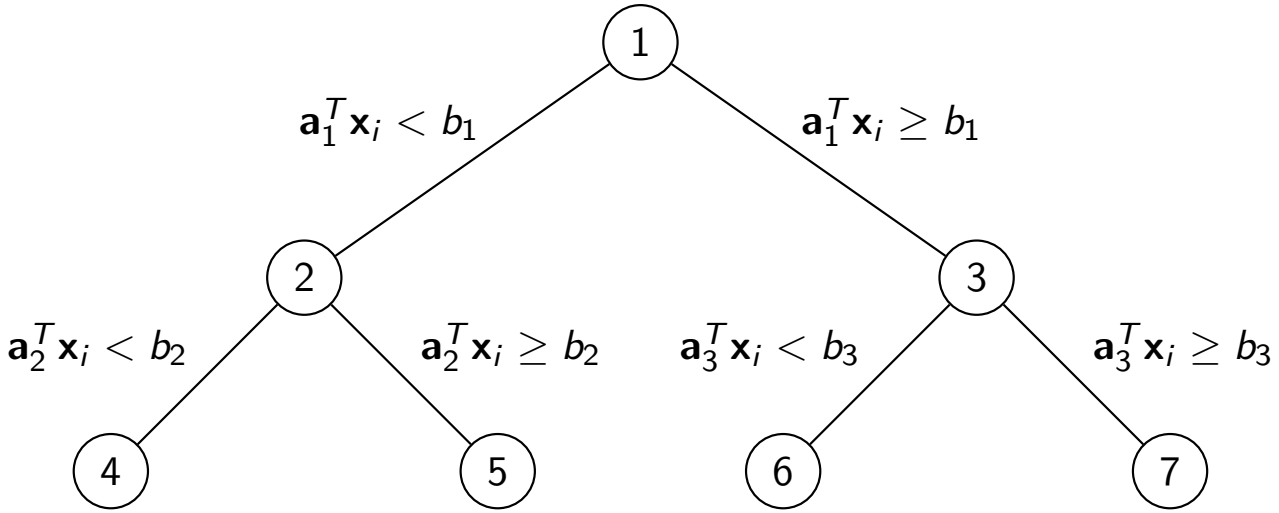
$$\min \quad \texttt{error}(\mathbb{T}, \mathbf{X}, \mathbf{y}) + \alpha \cdot \texttt{complexity}(\mathbb{T})$$

- Same problem that CART attempts to solve with growing and pruning heuristics

- Solving directly avoids need for impurity measure and pruning

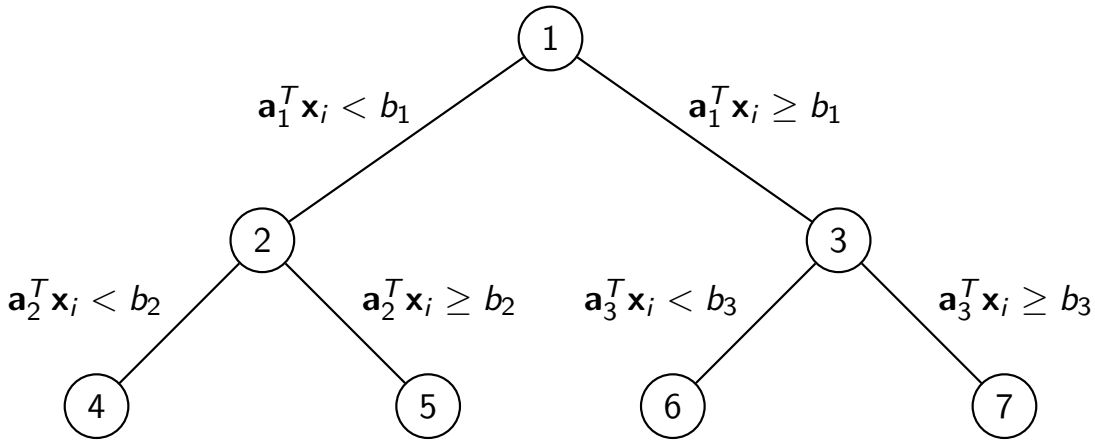
Forming the tree structure

- Given training data $(\mathbf{x}_i, y_i), i = 1, \ldots, n$
- Specify a maximum depth and form a complete tree of that depth



- Define sets for branches, $\mathcal{B} = \{1, 2, 3\}$, and leaves, $\mathcal{L} = \{4, 5, 6, 7\}$.
- Variables $\mathbf{a}_t$, b_t define the split at each branch node $t \in \mathcal{B}$
- We want splits that are parallel to the axes (one feature at a time)
  - Elements of $\mathbf{a}_t$ binary, and $\sum_{j=1}^{p} a_{jt} = 1$, so exactly one element is 1

Allocating points to leaves



- Each point has to be assigned to a leaf $t \in \mathcal{L}$ according to the splits:
  - Binary variables $z_{it} = 1$ if point i assigned to leaf t, 0 otherwise
  - $\sum_{t \in \mathcal{L}} z_{it} = 1$ to ensure each point is assigned to a leaf
- Enforce splitting rules

$$\mathbf{a}_m^T\mathbf{x}_i + \epsilon \leq b_m + M\left(1 - z_{it}\right), \forall \text{ left-branch ancestors } m \text{ of } t$$

$$\mathbf{a}_m^T\mathbf{x}_i \geq b_m - M\left(1 - z_{it}\right), \forall \text{ right-branch ancestors } m \text{ of } t$$

Include option to stop splitting

- At each node t in the tree, we choose either to branch or stop
  - New binary $d_t = 1$ if we choose to apply a split at node t

$$\sum_{j=1}^{p} a_{jt} = d_t$$

$$0 \leq b_t \leq d_t$$

- If $d_t = 0$, prevents a split from being applied
  - Split inequalities becomes $0 < 0$, so all points follow right branch — effectively no split
  - No effect if $d_t = 1$

Enforcing minbucket constraint

- Enforce a minimum number of points assigned to leaf, $N_{\min}$:
  - $l_t = 1$ if leaf t has any points:

$$l_t \geq z_{it}, \quad i = 1, \ldots, n$$

  - Must be at least $N_{\min}$ points in leaf:

$$\sum_{t \in \mathcal{L}} z_{it} \geq N_{\min} \cdot l_t$$

Calculating misclassification

- For each leaf node, the best class to assign is the most common label among points assigned to that node
- Use N_{kt} to count the points of each label k in leaf t:

$$N_{kt} = \sum_{i:y_i=k} z_{it}$$

- Set $N_t = \sum_k N_{kt}$ to be the number of points in each leaf
- The misclassification error, L_t, is the number of points in the leaf that do not belong to the most common class

$$L_t = N_t - \max_k \{N_{kt}\} = \min_k \{N_t - N_{kt}\}$$

Linearizing misclassification terms

$$L_t = \min_k \{N_t - N_{kt}\}$$

- We can linearize this by introducing binary variables c_{kt} to give:

$$\begin{aligned} L_t &\geq N_t - N_{kt} - M(1 - c_{kt}), \quad && k = 1, \ldots, K \\ L_t &\leq N_t - N_{kt} + M c_{kt}, && k = 1, \ldots, K \\ L_t &\geq 0 \end{aligned}$$

- $c_{kt} = 1$ if class k is assigned to leaf t and 0 otherwise
- We can use $M = n$ for valid bounds

Optimal Trees with MIO: Overall objective

- Objective function:

$$\min \quad \texttt{error}(\mathbb{T}, \mathbf{X}, \mathbf{y}) + \alpha \cdot \texttt{complexity}(\mathbb{T})$$

- The misclassification error is

$$\sum_{t \in \mathcal{L}} L_t$$

- The complexity is

$$\sum_{t \in \mathcal{B}} d_t$$

- Overall objective:

$$\min \quad \sum_{t \in \mathcal{L}} L_t + \alpha \cdot \sum_{t \in \mathcal{B}} d_t$$

Complete MIO Formulation

$$\min \quad \frac{1}{\hat{L}} \sum_{t \in \mathcal{L}} L_t + \alpha \cdot \sum_{t \in \mathcal{B}} d_t$$

$$\text{s.t.} \quad L_t \geq N_t - N_{kt} - n(1 - c_{kt}) \qquad L_t \leq N_t - N_{kt} + n c_{kt}, \qquad k = 1, \ldots, K, \ \forall t \in \mathcal{L}$$

$$N_{kt} = \sum_{i=1}^{n} \frac{1}{2}(1 + Y_{ik}) z_{it} \qquad k = 1, \ldots, K, \ \forall t \in \mathcal{L}$$

$$N_t = \sum_{i=1}^{n} z_{it} \qquad L_t \geq 0 \qquad \forall t \in \mathcal{L}$$

$$\mathbf{a}_m^T(\mathbf{x}_i + \boldsymbol{\epsilon}) \leq b_m + (1 + \epsilon_{\max})(1 - z_{it}), \qquad i = 1, \ldots, n, \ \forall t \in \mathcal{L}, \ \forall m \in \mathcal{P}_t^L,$$

$$\mathbf{a}_m^T \mathbf{x}_i \geq b_m - (1 - z_{it}), \qquad i = 1, \ldots, n, \ \forall t \in \mathcal{L}, \ \forall m \in \mathcal{P}_t^R,$$

$$\sum_{t \in \mathcal{L}} z_{it} = 1, \qquad i = 1, \ldots, n,$$

$$\sum_{j=1}^{p} a_{jt} = d_t, \qquad 0 \leq b_t \leq d_t, \qquad \forall t \in \mathcal{B}$$

$$d_t \leq d_{p(t)}, \qquad \forall t \in \mathcal{B}$$

$$\sum_{i=1}^{n} z_{it} \geq N \cdot l_t, \qquad \forall t \in \mathcal{L}$$

$$l_t \geq z_{it}, \qquad \forall t \in \mathcal{L}, \ i = 1, \ldots, n.$$

Modifying formulation to incorporate hyperplane splits

- Our previous constraints on the splits were:

$$\sum_{j=1}^{p} a_{jt} = 1, \quad \forall t \in \mathcal{B}$$

$$a_{jt} \in \{0,1\}, \quad j = 1, \ldots, p, \ \forall t \in \mathcal{B}$$

- We can simply relax integrality on the a_{jt} and replace sum with norm:

$$\|\mathbf{a}_t\|_1 \leq 1, \quad \forall t \in \mathcal{B}$$

- Choose 1-norm because we can linearize it easily

Modifying objective for hyperplane splits

- Previously we penalized the number of splits in the tree:

$$\min \sum_{t \in \mathcal{L}} L_t + \alpha \cdot \sum_{t \in \mathcal{B}} d_t$$

- It seems sensible to penalize added complexity of hyperplane splits
- Add a penalty per variable used in splits
- Introduce binary variable s_{jt} to track if jth feature is used in split t:

$$-s_{jt} \leq a_{jt} \leq s_{jt}$$

- New objective:

$$\min \sum_{t \in \mathcal{L}} L_t + \alpha \cdot \sum_{t \in \mathcal{B}} \sum_{j=1}^{p} s_{jt}$$

How hard is the hyperplane problem?

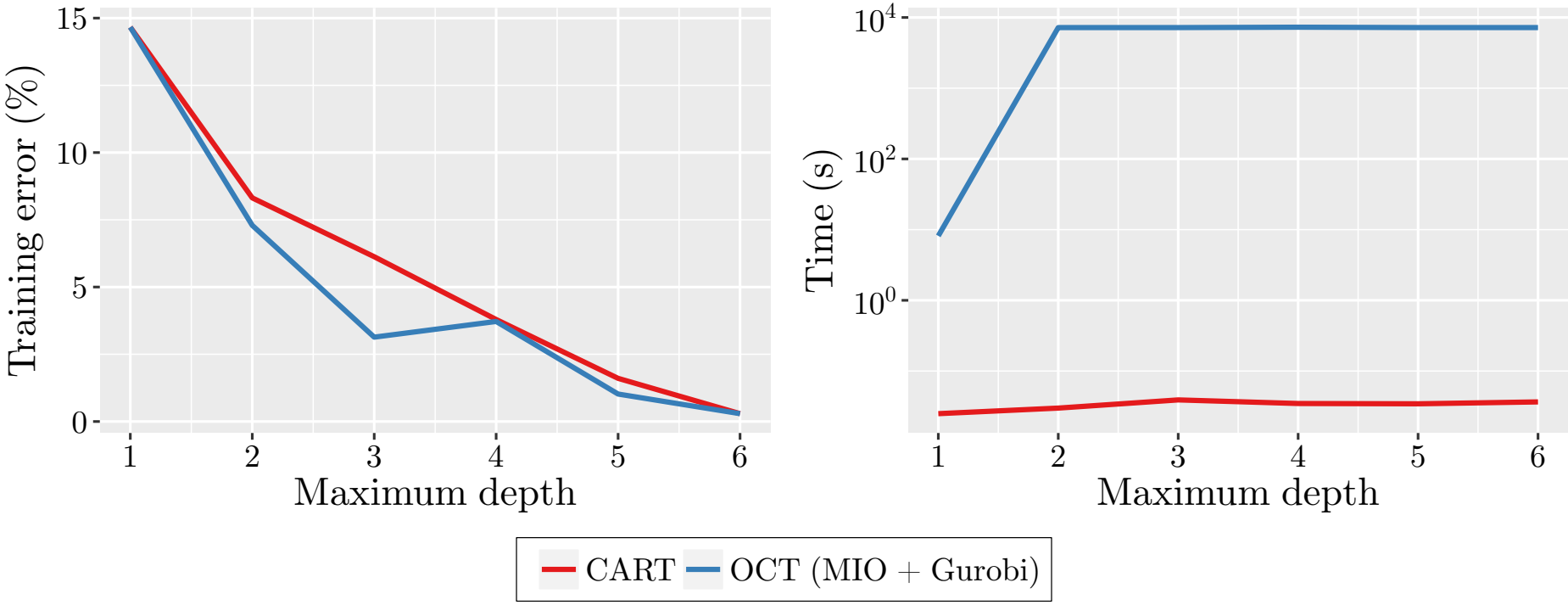
- Allowing for hyperplane splits does not meaningfully change the problem size or difficulty

- Unlike other approaches in the literature, our trees with hyperplane splits are about as easy to find as axes-parallel trees when using MIO

Outline

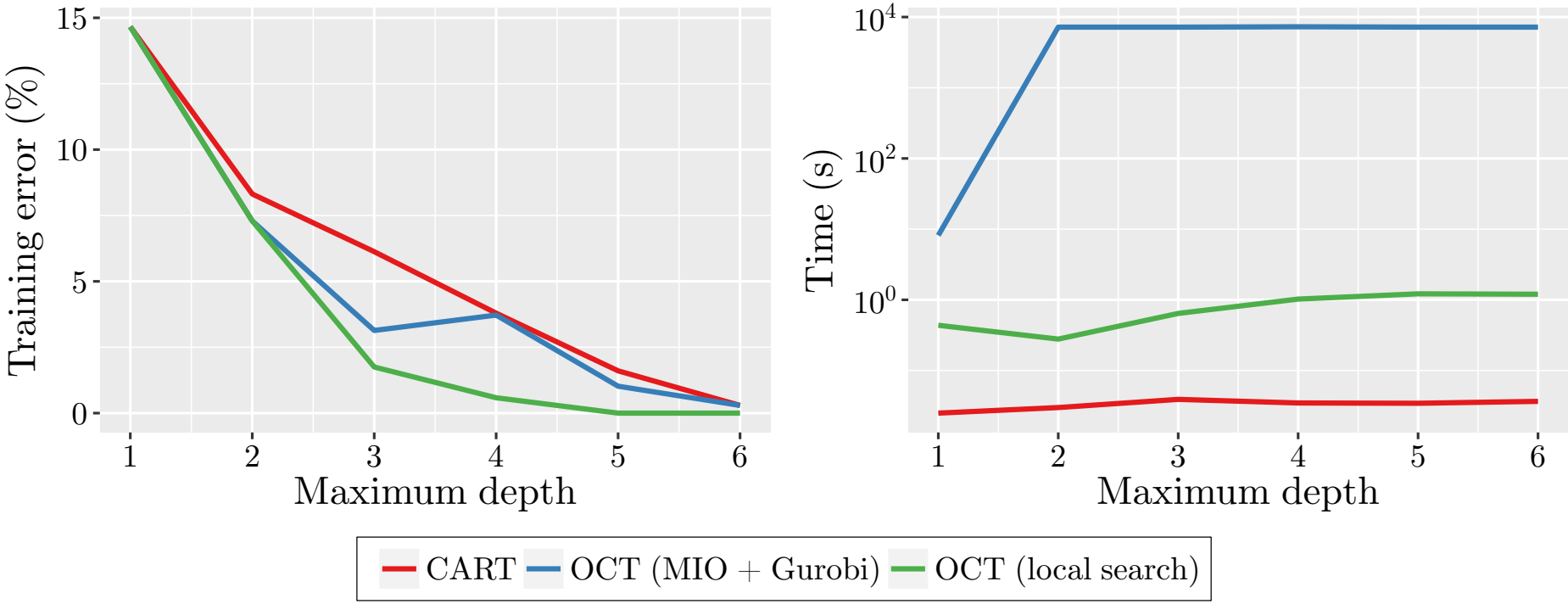
How good is Gurobi?

- Banknote Authentication dataset: $n = 1372$, $p = 4$, $K = 2$
- Small improvements over CART
- **Much slower:** Hit 2 hour timelimit for each tree (vs. 0.12 sec)



How good is the local search?

- Banknote Authentication dataset: $n = 1372$, $p = 4$, $K = 2$
- Significant improvements over both CART and Gurobi-based method
- **Fast:** 100 random restarts takes $\sim$ 1 sec, vs. $\sim$ 0.1 sec for CART



Speeding up solve times

- Large problem size
  - There are $n \cdot 2^D$ binary z variables to track point allocation to leaves
  - Leads to terrible scaling with both D and n
- Which variables do we need to optimize to find a tree?
  - Split variables $\mathbf{a}$ and b
- Given the splits, everything else is closed-form solvable
  - Divide the points up according to splits
  - Use majority rule in leaves to calculate error
- The real problem is

$$\min_{\mathbb{T}} \quad \texttt{error}(\mathbb{T}, \mathbf{X}, \mathbf{y}) + \alpha \cdot \texttt{complexity}(\mathbb{T})$$

- Decision space is independent of n and scales linearly with p and number of splits
- Need a way of searching this reduced decision space

Local search heuristic for Optimal Trees

- We have developed a heuristic local search procedure with random restarts to efficiently optimize over this reduced search space

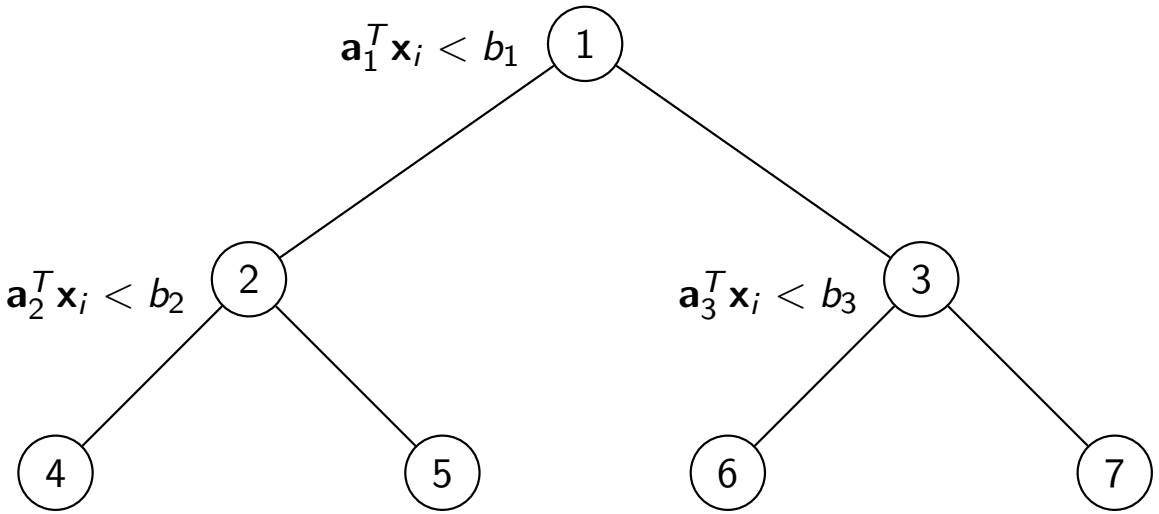
- Overview of one local search iteration:
  - Choose node in the tree at random
  - Re-optimize split at this node so it is locally optimal
    - ★ If improved, exit and start local search iteration on new tree
  - If no split can be improved, terminate

- Repeat local search iterations until no improvements found

- Use different starting trees as random restarts

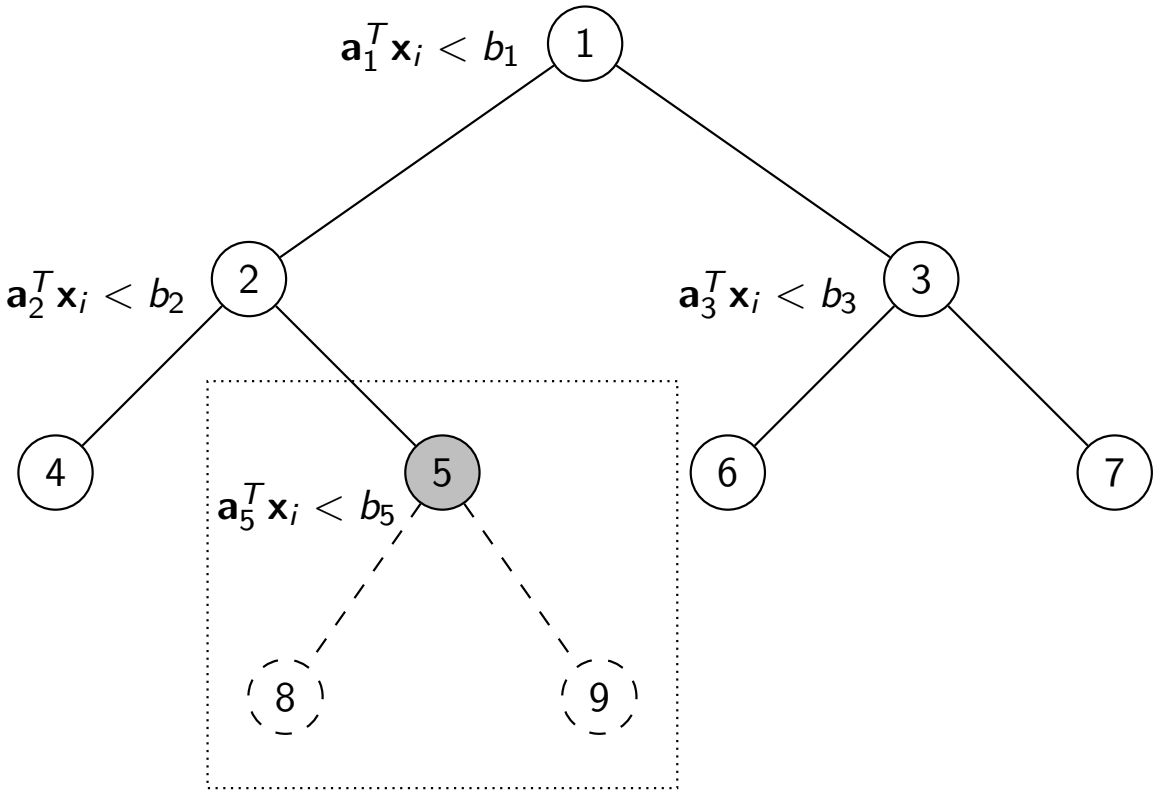
How to re-optimize splits

- How we re-optimize depends on which node we have chosen at random



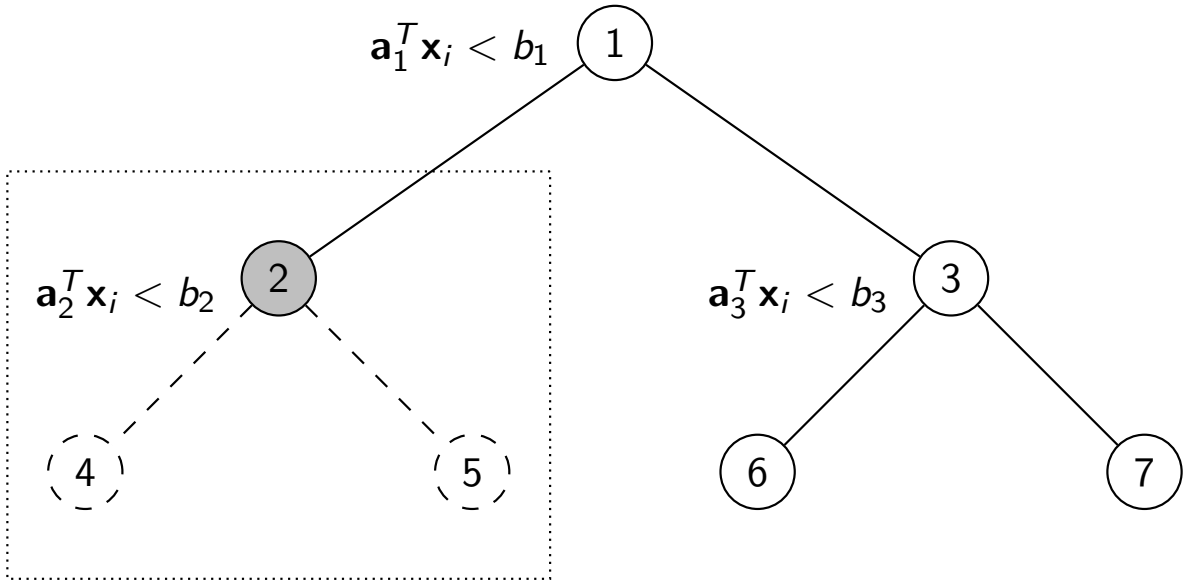
How to re-optimize splits—leaf nodes

- We apply CART algorithm to split leaf node into two new leaves:



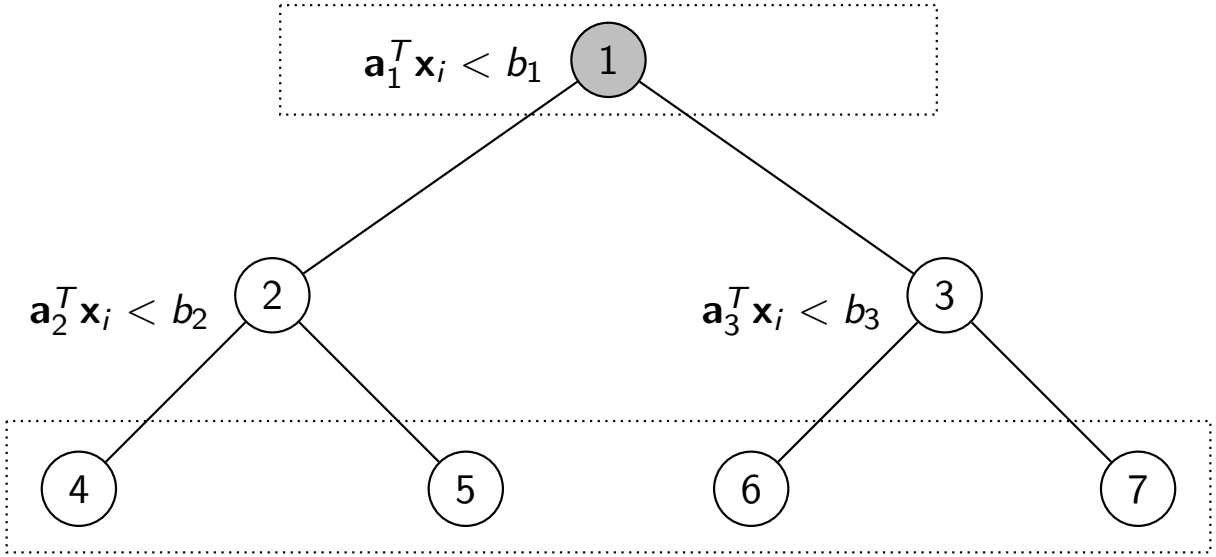
How to re-optimize splits—branch nodes with leaves

- If the branch node has leaves on both branches, we can also just apply CART to find two new leaves:



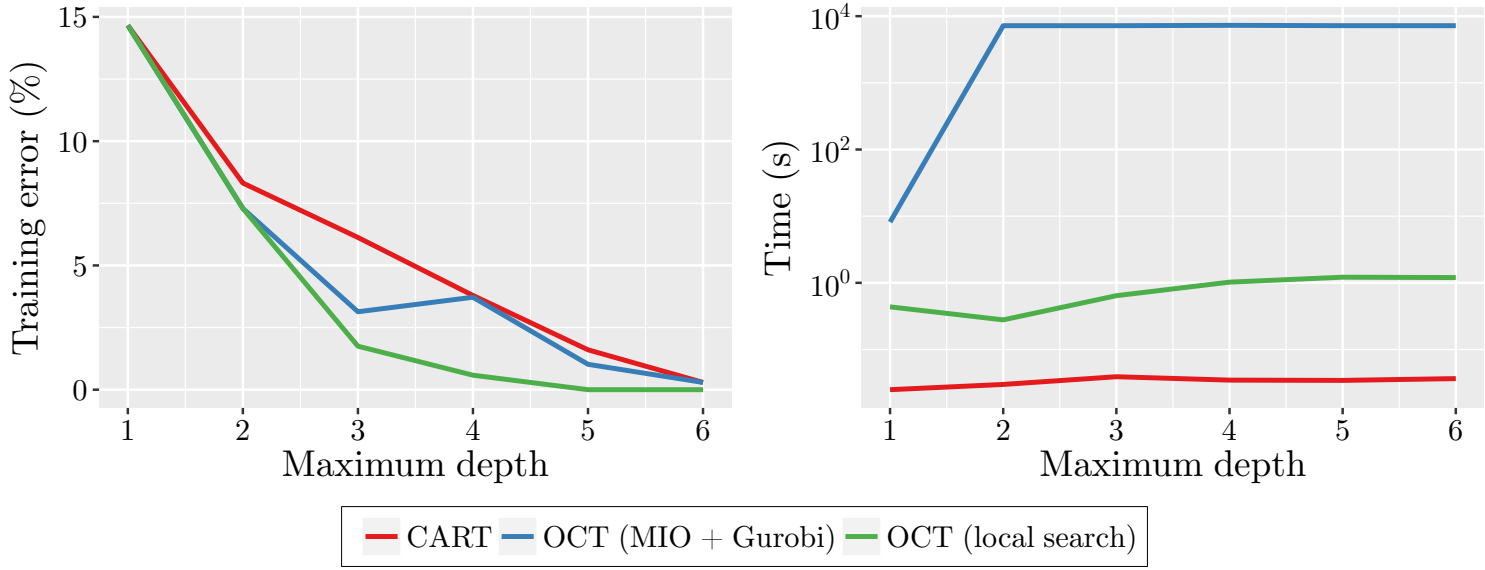
How to re-optimize splits—branch nodes with subtrees

- If the branch node has a subtree on either side, we use a modified CART algorithm:
  - Consider each possible split location
  - Divide the points into the left and right subtrees according to split
  - Find best labels for leaves and calculate error
  - Choose the split with the best error
- Same as CART except for more complicated error calculation



How good is the local search?

- Banknote Authentication dataset: $n = 1372$, $p = 4$, $K = 2$
- Significant improvements over both CART and Gurobi-based method
- **Fast:** 100 random restarts takes 0.98 sec, vs. 0.12 sec for CART



Local search for hyperplane trees

- Need a method for optimizing a single hyperplane split in order to apply the local search idea to hyperplanes
  - Difficulty: no good way to search all possible hyperplanes
  - Easy to do as a small MIO problem, but doesn't scale
- Coordinate descent:
  - Optimize coefficients in the hyperplane one-by-one
  - Can find the optimal coefficient with CART-like algorithm
  - Repeat until no improvement
  - Avoid splits getting stuck in local minima by restarting with random hyperplane
- Perturbation heuristic gives same performance as Gurobi-based local search on small problems

Outline

Computational experiments

- **Aspirations**
  - Develop methods that scale to real-world applications
  - Demonstrate added value over state-of-the-art methods
  - Learn how dataset characteristics affect performance
- Seek to comprehensively benchmark performance of Optimal Trees with two sets of experiments:
  - **Synthetic:** Generate datasets from a "true" tree and compare to trees created by methods
  - **Real-world:** Comparison of methods on large set of real-world data sets to measure performance in practical settings

Synthetic experiments

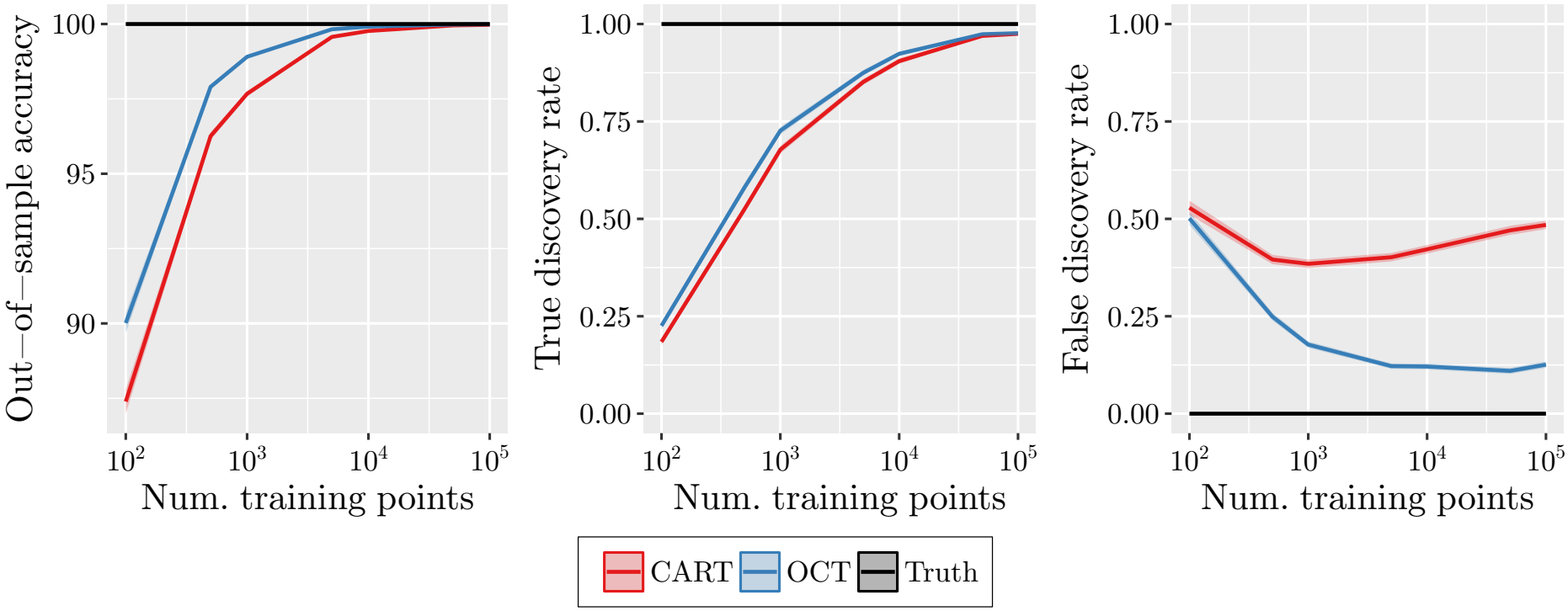
- Generate training and test datasets according to "ground truth" tree
- Induce decision trees on the training set with CART and OCT and compare these trees to the ground truth using the test set

- Gives us full control over the dataset size and dimension

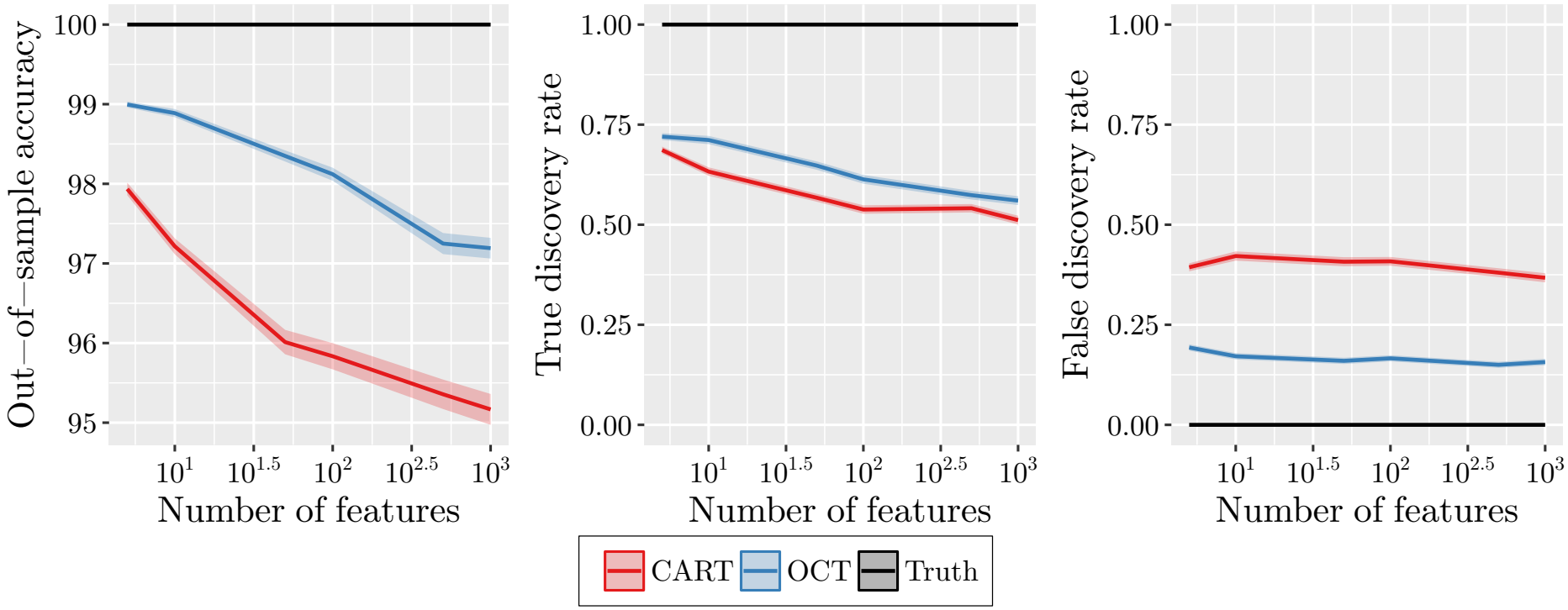
- Measuring quality of trees:
  - **Out-of-sample accuracy**, the accuracy on the test set
  - **True discovery rate**, proportion of splits in ground truth that appear in generated tree
  - **False discovery rate**, proportion of splits in generated tree that are not in ground truth

- Repeated 400 times for each problem

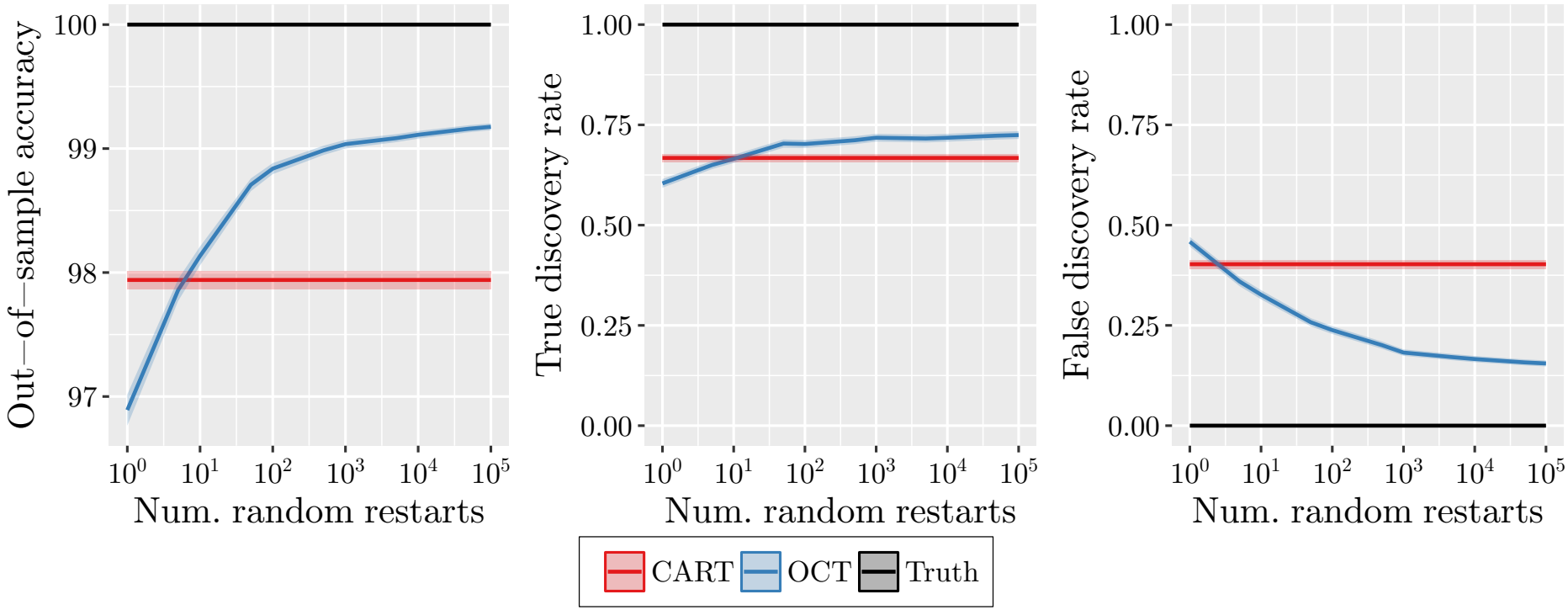
Number of training points



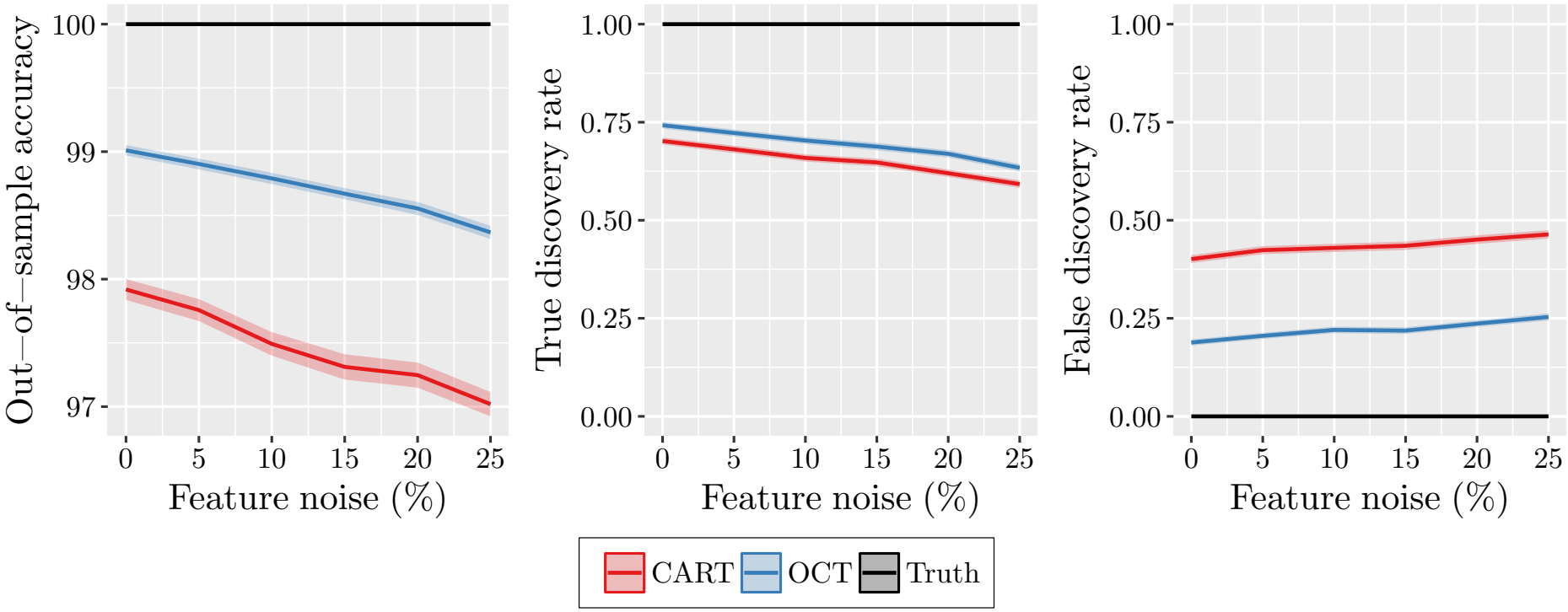
Dimension of points



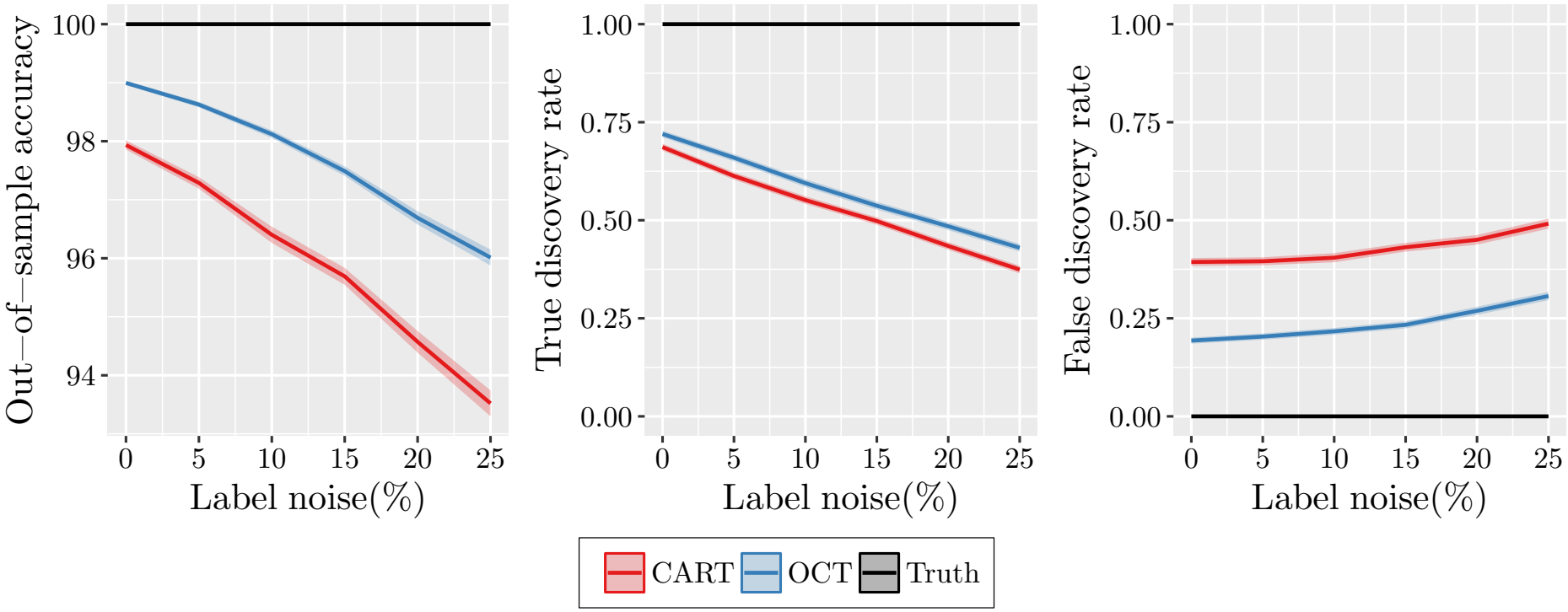
Number of random restarts



Feature noise



Label noise


Out−of−sample accuracy
100
98
96
94
0
5
10
15
20
25
Label noise(%)
True discovery rate
1.00
0.75
0.50
0.25
0.00
0
5
10
15
20
25
Label noise(%)
False discovery rate
1.00
0.75
0.50
0.25
0.00
0
5
10
15
20
25
Label noise(%)
CART
OCT
Truth

Summary of synthetic results

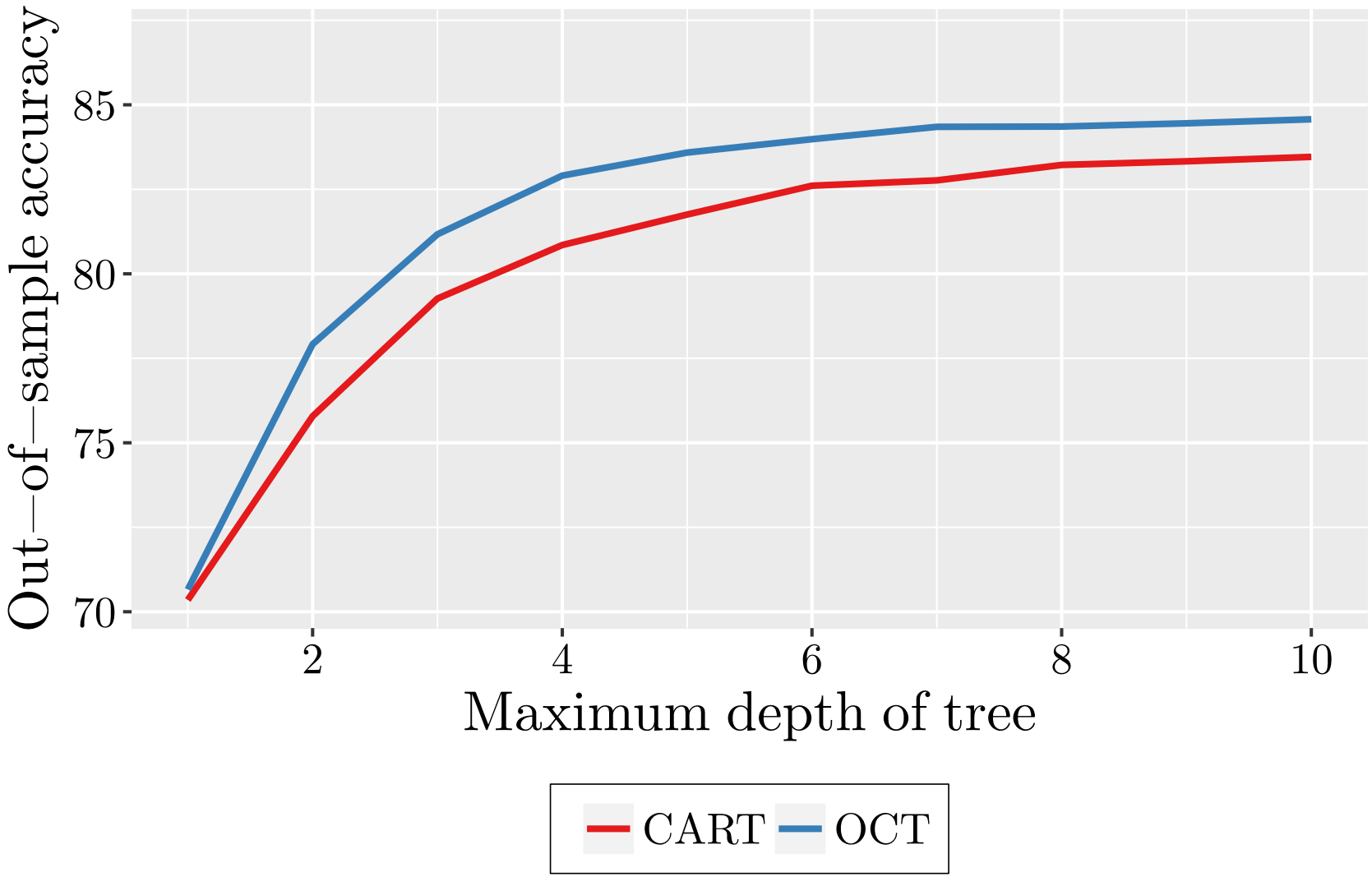
- In each case, OCT generates trees that are a much closer match structurally for the ground truth
- As problem difficulty increases ($n \downarrow$, $p \uparrow$), performance gap between OCT and CART increases
- OCT performs stronger than CART even under very large levels of noise
- OCT gives small improvements in true discovery rate, and significant reductions in false discovery rate $\implies$ significantly better for interpretation

Real-world datasets

- 60 datasets from UCI Machine Learning Repository
  - Wide range of values for n (50–245,000), p (2–500) and K (2–10)
- For each dataset:
  - Split data into training and testing sets (75/25)
  - Train model on training and report error on test set
- Repeated five times for each dataset and results averaged
  - Minimizes the effect of any particular training/validation/test split
- Carried out for CART, OCT

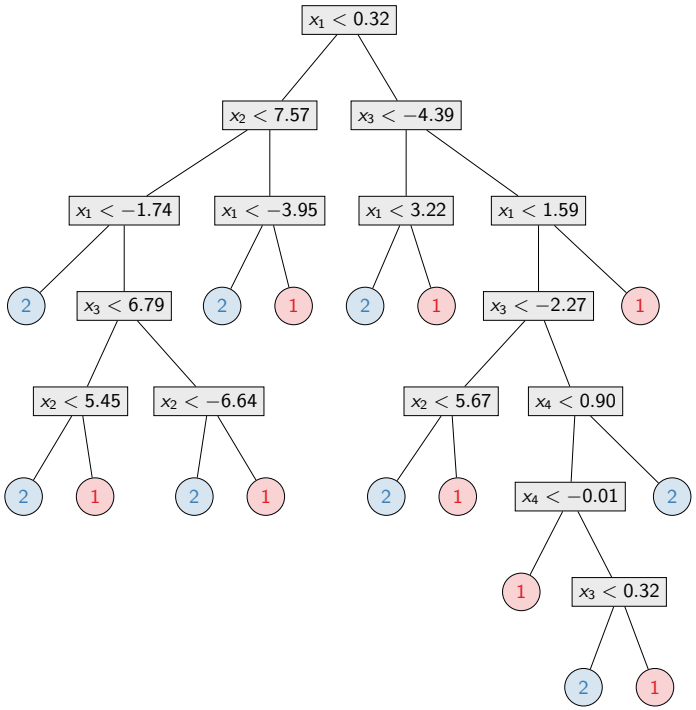
Performance on real-world datasets

- Average out-of-sample accuracy across all 60 datasets:

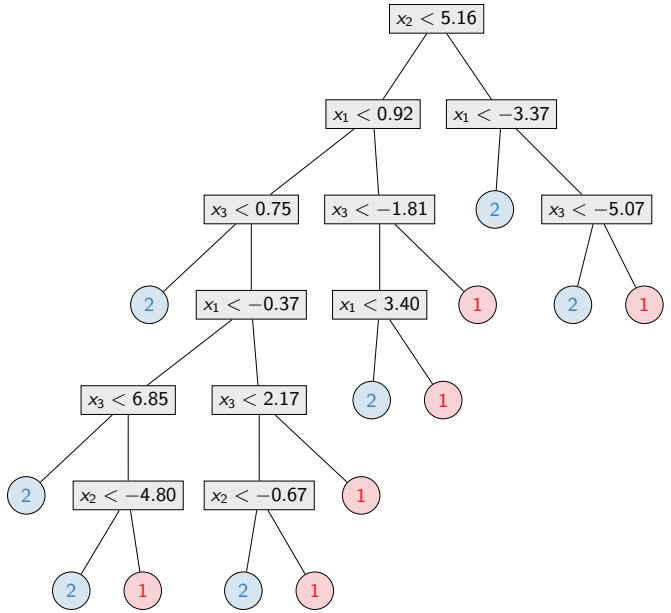


Interpretability

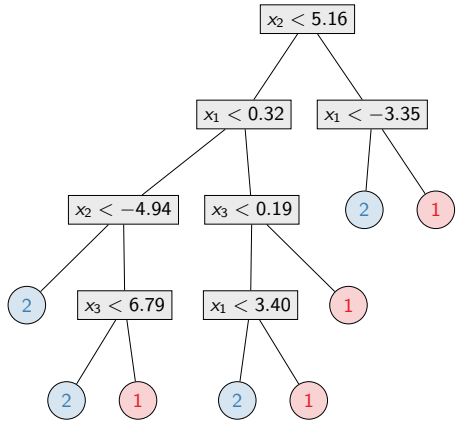
CART (test 98.3%; depth 7 with 15 splits)



OCT (test 99.1%; depth 6 with 12 splits)



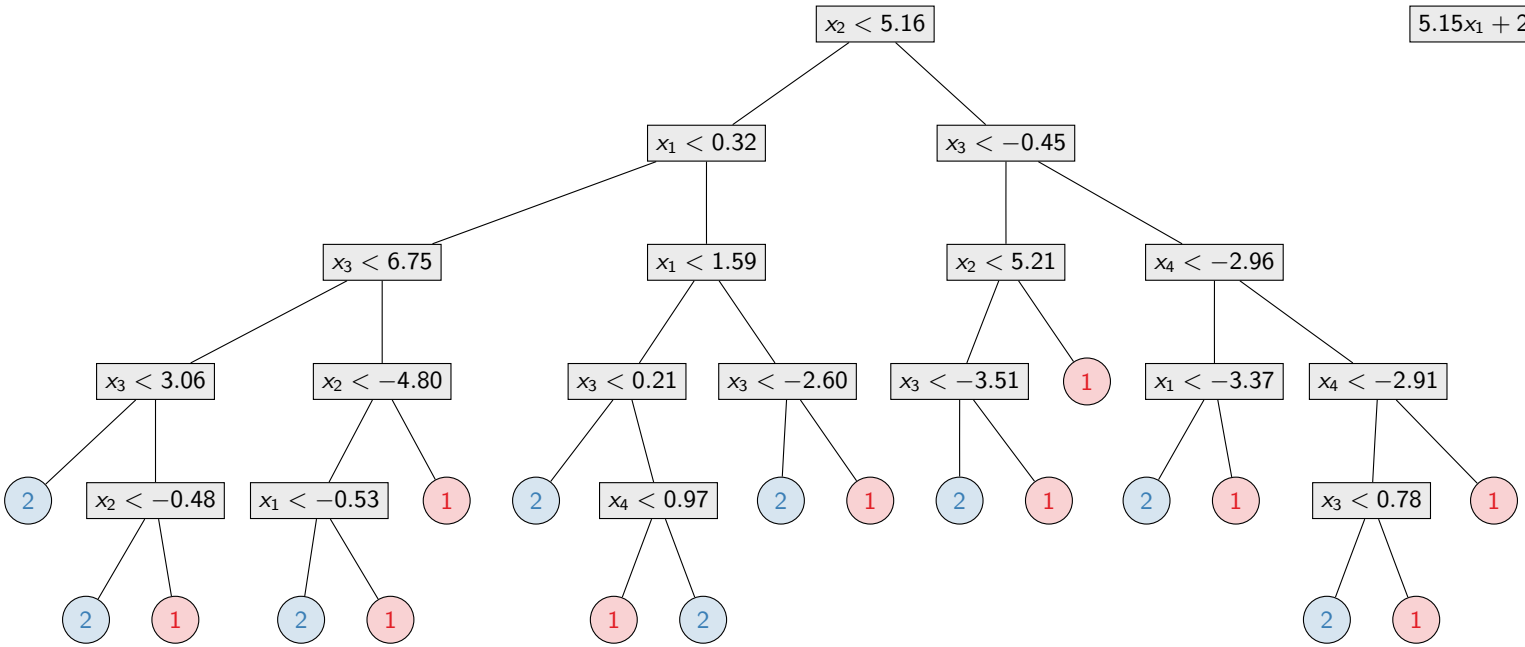
OCT (test 98.3%; depth 4 with 7 splits)



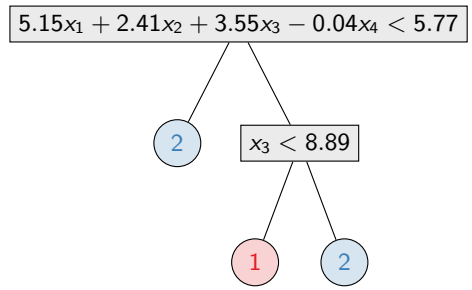
Interpretability of Hyperplane Trees

- Hyperplane splits are less interpretable than parallel
- Are hyperplane trees less interpretable than parallel?

OCT (test 99.5%; depth 5)



OCT-H (99.5%; depth 2)

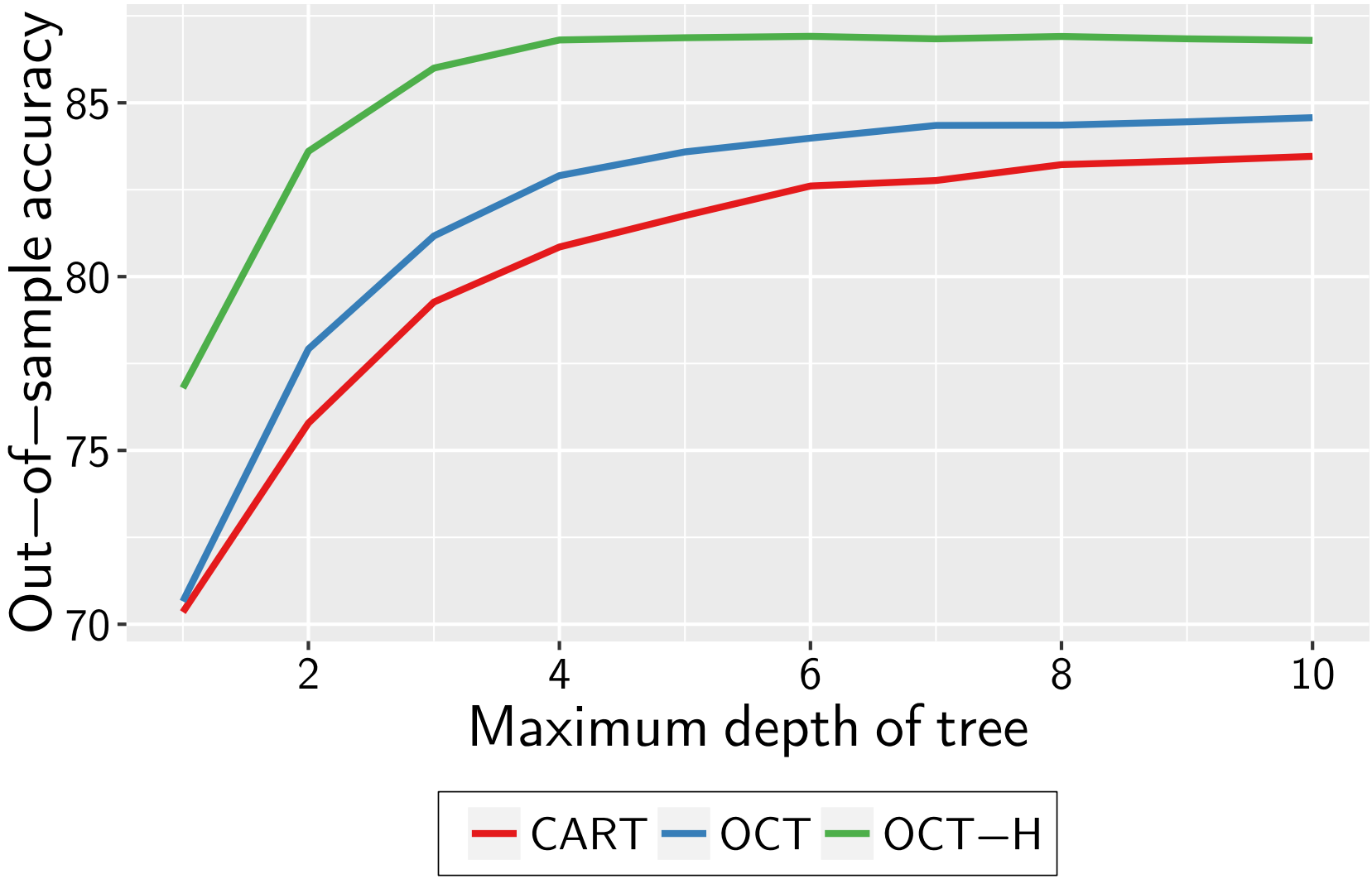


Tradeoff between interpretability and accuracy

- CART is widely used throughout industry and academia, and is a state-of-the-art decision tree learner
- However, it does **not** achieve state-of-the-art accuracy
- Other tree-based ensemble methods do give state-of-the-art accuracies (Random Forests and Boosting)
- Ensemble methods sacrifice interpretability for accuracy by averaging multiple trees
- **Currently:** Have to choose between interpretability or accuracy depending on application
- **Aspiration:** Achieve state-of-the-art accuracy without sacrificing the interpretability of a single decision tree

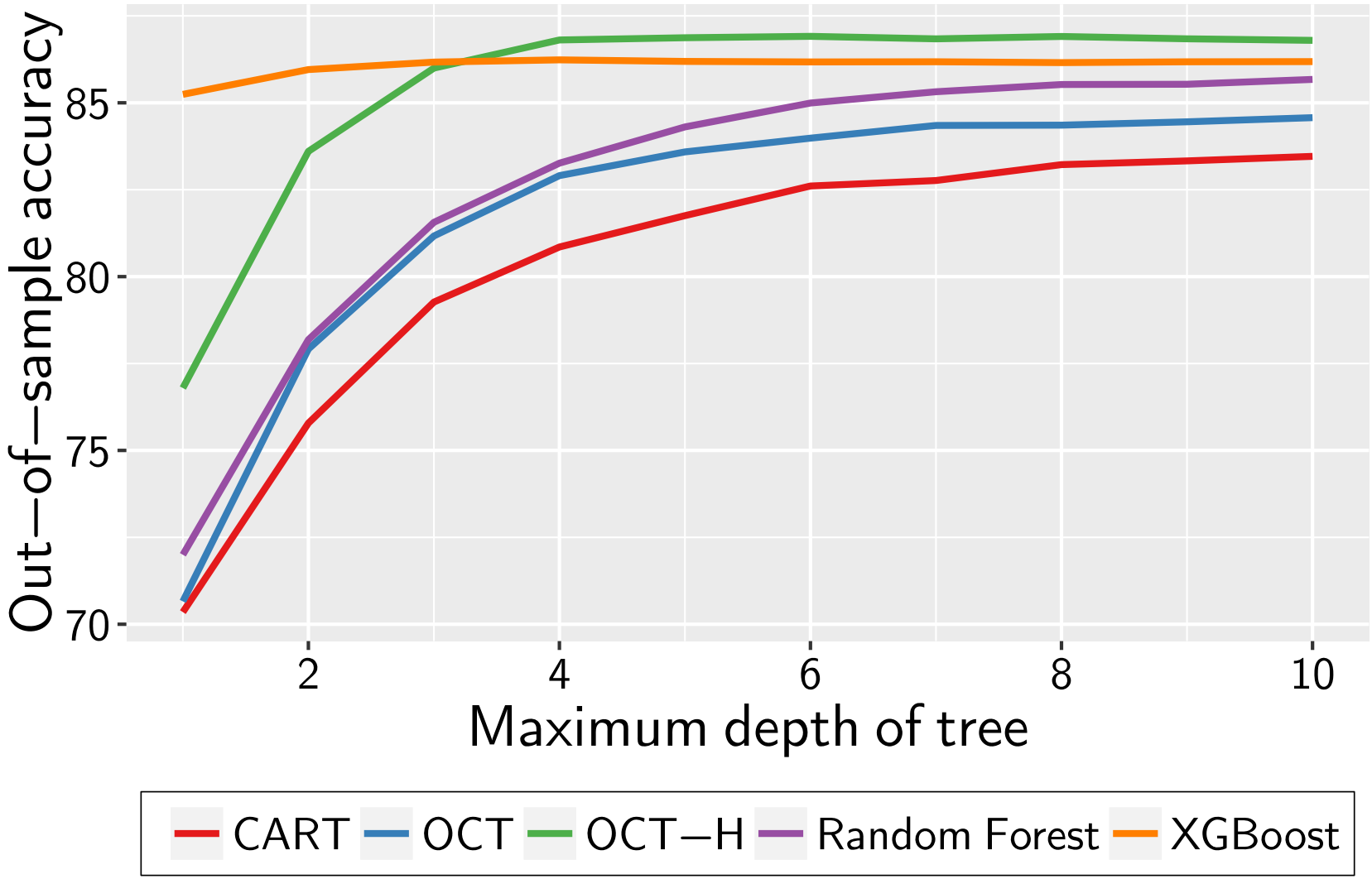
Performance on real-world datasets

- Average out-of-sample accuracy across all 60 datasets:



Performance on real-world datasets

- Average out-of-sample accuracy across all 60 datasets:



Outline

Optimal Trees in Action

- Our experiments demonstrate that Optimal Trees improve significantly upon CART
  - Average improvement in accuracy of 3% for classification
- That wasn't the point of developing these methods!
- **Key Question:** Will these improved trees lead to better results in real-world settings?
- What does a 3% increase in accuracy actually mean in a practical setting?

Case Study: Identification of Brain Injury in Children

- Children present at the ER with apparent head injury
- Doctor needs to decide whether child has *clinically-important traumatic brain injury* (ciTBI), which requires acute intervention
- Traumatic brain injury is a leading cause of death and disability in children worldwide
  - Cannot afford to miss any patient with ciTBI
- Determination of ciTBI can be made by conducting a CT scan, but not feasible to simply scan all patients:
  - Radiation from CT has an estimated rate of lethal malignancies in children of 1 in 1000 to 1 in 5000, increased risk as age decreases
  - Cost of conducting scan, limited resources at hospital
- Need high detection rate while minimizing number of superfluous scans conducted

The current state-of-the-art

- Kuppermann et al. (2009). "Identification of children at very low risk of clinically-important brain injuries after head trauma: a prospective cohort study". *Lancet*.

- Data from the Pediatric Emergency Care Applied Research Network (PECARN)
  - 42,000 children presenting at ER
  - 376 were determined to have ciTBI ($< 1\%$)

- Kuppermann et al. used CART to build decision trees to identify which children should have CT scans
  - Trees can easily be interpreted and understood by doctors
  - Following these CART models is now current medical practice

Predictors in the data

History	Physical	Other
age	GCS	intubated
gender	altered mental status	sedated
ethnicity	agitated	pharmacologically paralyzed
race	sleepy	injury mechanism
amnesia	slow reacting	injury severity
loss of conscience	repetitive questions	
duration	other	
seizures	palpable skull fracture	
initiation	scalp hematoma	
duration	size	
acting normal	location	
headache	supraclavicular trauma	
initiation	neurological deficit	
severity	other substantial injury	
vomiting	intoxication	
episodes	bulging anterior fontanelle	
first vomit		
last vomit		
dizziness		

Applying OCT to prescribe CT scans

- Good application to test real-world impact of OCT
  - CART already used in practice by doctors
  - Data already collected and available
- Use exactly the same approach as Kuppermann et al., just with OCT in place of CART

CART vs. OCT: Age less than 2 years

CART

Decision	Training		Test	
	ciTBI	No ciTBI	ciTBI	No ciTBI
CT	80	4040	17	995
No CT	1	4453	0	1132

OCT

Decision	Training		Test	
	ciTBI	No ciTBI	ciTBI	No ciTBI
CT	80	2300	17	459
No CT	1	6193	0	1668

CART vs. OCT: Aged 2 years or older

CART

Decision	Training		Test	
	ciTBI	No ciTBI	ciTBI	No ciTBI
CT	237	11629	33	2029
No CT	7	13482	1	4276

OCT

Decision	Training		Test	
	ciTBI	No ciTBI	ciTBI	No ciTBI
CT	241	9020	33	1804
No CT	3	16091	1	4501

Summary of results

- **Children younger than 2 years:**
  - CART: 5132 CT scans (47.9% of patients), 1 ciTBI missed
  - OCT: 2856 CT scans (26.6% of patients), 1 ciTBI missed
  - Significantly increased specificity 20–25% without affecting the sensitivity.
- **Children aged 2 years and older:**
  - CART: 13928 CT scans (43.9% of patients), 8 ciTBI missed
  - OCT: 11098 CT scans (35.0% of patients), 4 ciTBI missed
  - Increased specificity 5–10% and cut misses in half.
- OCT gives significantly better performance than CART, particularly for the younger, pre-verbal children
  - This group faces highest risk from radiation
- Easy to implement and can have a significant impact on current medical practice

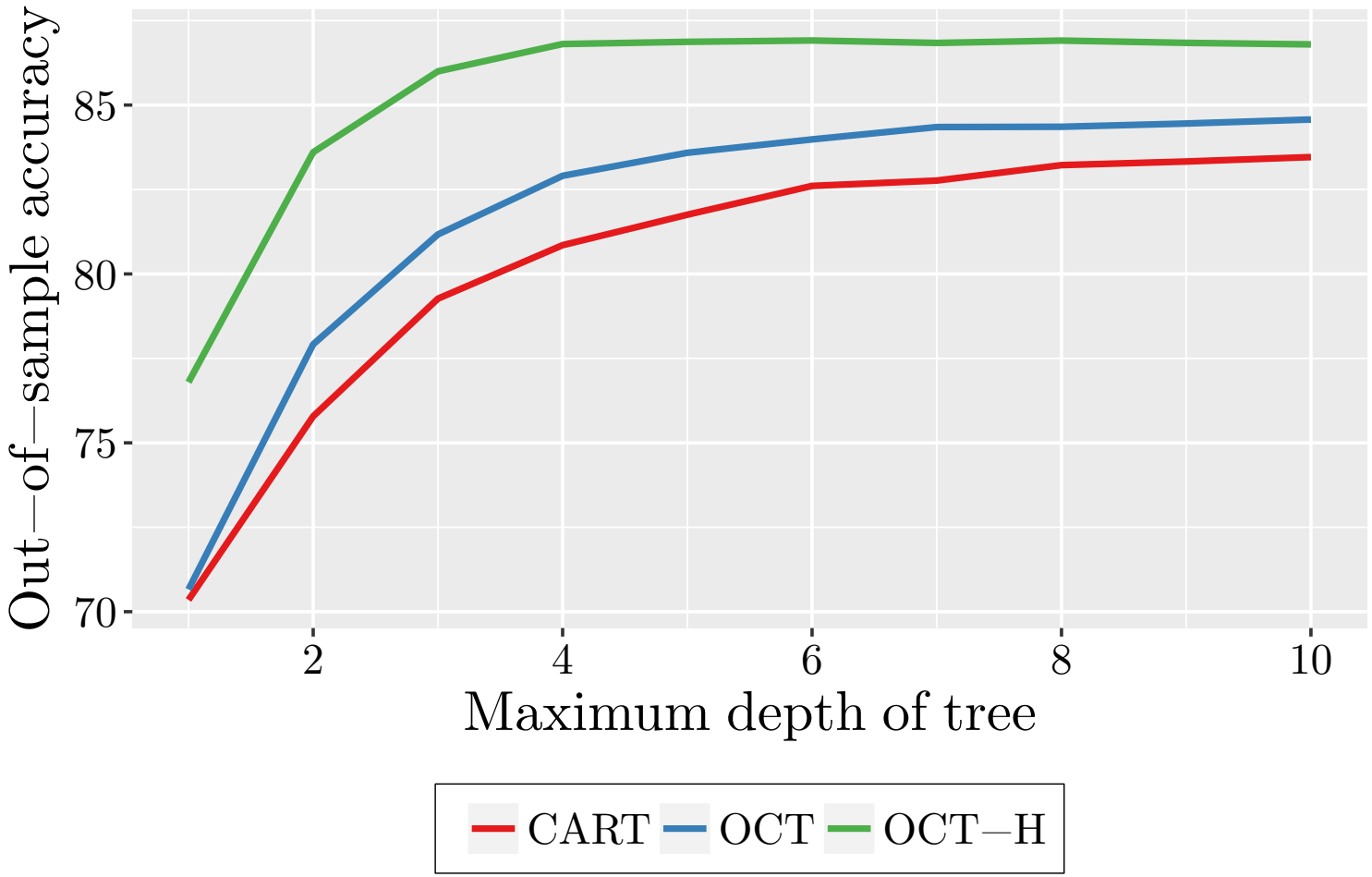
Outline

Conclusions

- Motivated by the incredible speedup in MIO, we formulated the decision tree problem from a global perspective for classification, regression and prescription
- We develop a high-performance and scalable heuristic for solving the optimal tree problem that delivers improved solutions significantly faster than MIO
- Comprehensive experiments show that our global approach improves significantly upon the greedy approach of CART and has competitive performance with the state-of-the-art, while maintaining the interpretability of a single decision tree
- Real-world case studies demonstrate that our improved methods lead to significant impact

Conclusions

- Optimization Edge: 1.3%
- Optimization + Hyperplanes: 3.3%



Conclusions

- Optimization Edge: 1.3%
- Optimization + Hyperplanes: 3.3%
- State-of-the-art accuracy whilst maintaining interpretability