Learning Decision Trees via Optimization & Optimization via Learned Decision Trees

- Yingqian Zhang (TU Eindhoven) & Sicco Verwer (TU Delft)
- with assistance of Daniel Vos (TU Delft)

Sicco Verwer – interpretable and safe ML

- Associate professor in algorithms
- Department of Software Technology, TU Delft

- Declarative and flexible machine learning
  - State machines
  - Decision trees

- Automated software analysis
  - Bug discovery
  - Incident response
  - Verification

- Interpretable, safe, and fair AI

- Chess and other board games

Yingqian Zhang – AI for decision making

- Associate professor
  Department of Industrial Engineering, TU/e
  Earlier CS@TU Delft; Econometrics@EUR
- Machine learning <> optimization
  - Decision tree learning
  - ML/DL/RL for combinatorial optimization
- Trustworthy AI
  - Fairness, transparency decision-making

Part 1: Learning by optimization

	f1	f2	f3	f4	t
1	5.1	3.5	1.4	0.2	s
2	4.9	3.0	1.4	0.2	s
3	4.7	3.2	1.3	0.3	v

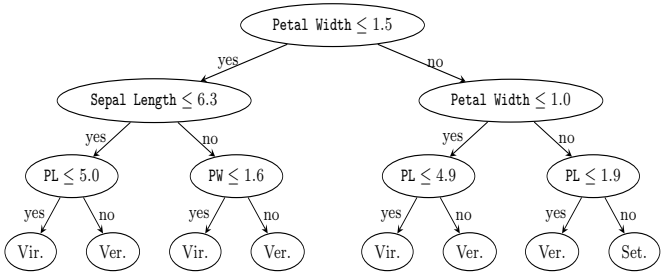
translate

$$l_{5,1} + l_{6,1} + 2 \cdot t_{n,1} \leq 2$$
$$l_{3,2} + l_{4,2} + l_{5,2} - 3 \cdot t_{n,1} \leq 0$$
$$l_{3,1} + l_{4,1} + 2 \cdot t_{n,1} + 2 \cdot t_{n,2} \leq 4$$
$$l_{3,2} + l_{4,2} - 2 \cdot t_{n,2} \leq 0$$
$$l_{6,1} + 1 \cdot t_{n,2} \leq 1$$
$$l_{6,2} - 1 \cdot t_{n,1} - 1 \cdot t_{n,2} \leq 0$$

add objective

$$\min \sum_{l,c} e_{l,c} \quad \text{s.t.}$$
$$\sum_{r:C_r=c} l_{r,l} - M \cdot p_{l,c} \leq e_{l,c}$$

solve



translate

Part 2: Optimization by learning

	f1	f2	f3	f4	t
1	5.1	3.5	1.4	0.2	s
2	4.9	3.0	1.4	0.2	s
3	4.7	3.2	1.3	0.3	v

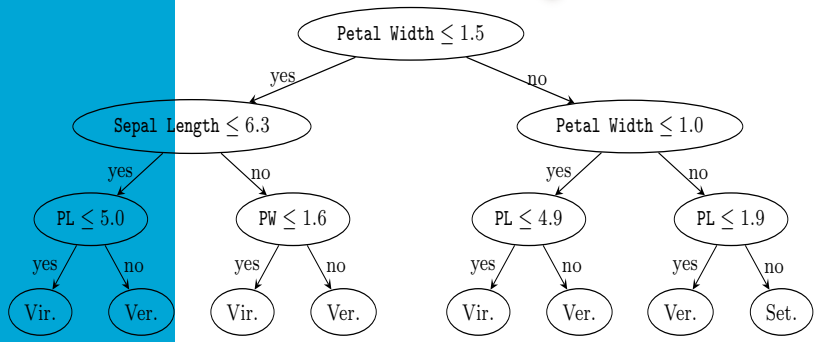
learn

translate

add to problem instance

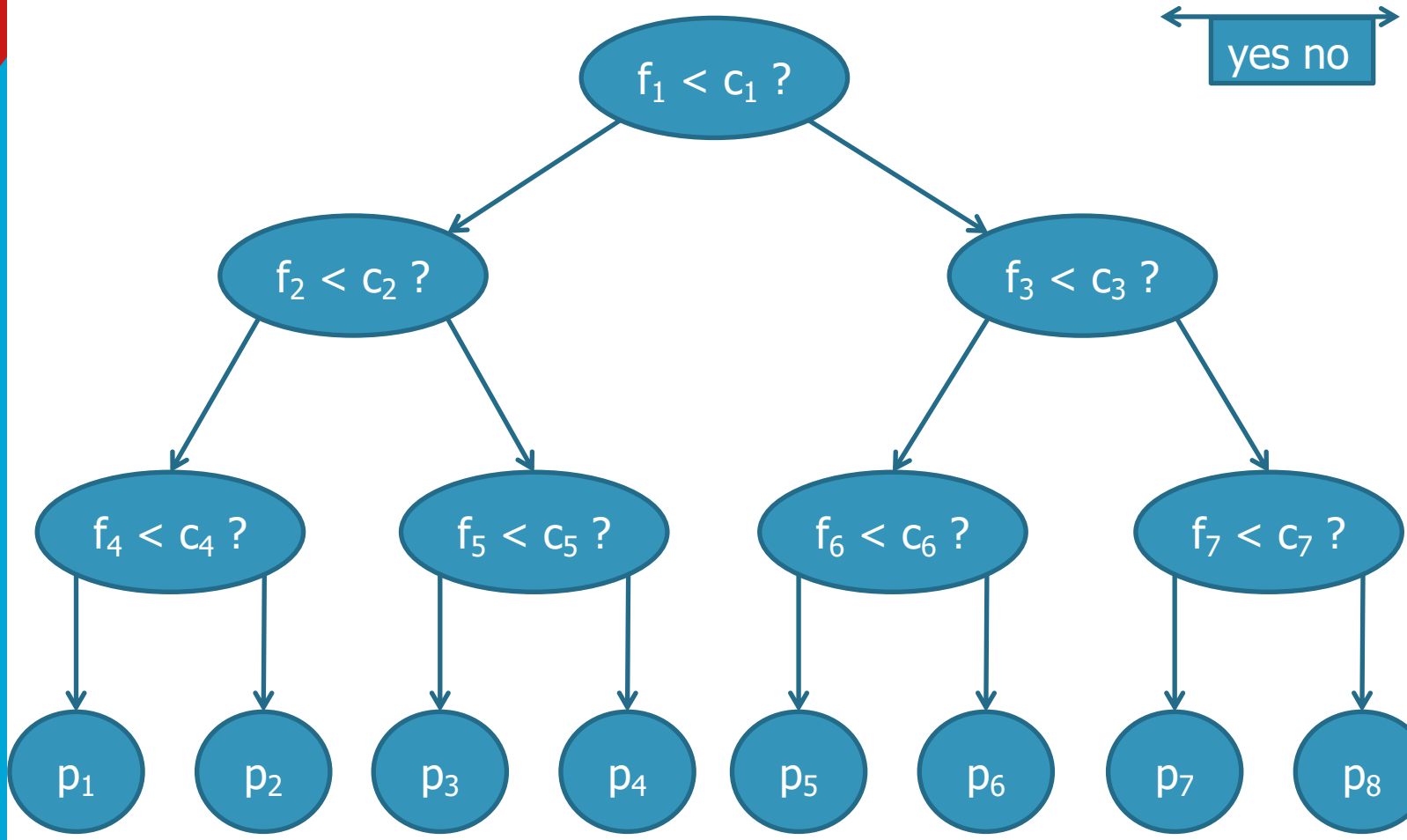
solve

$$l_{5,1} + l_{6,1} + 2 \cdot t_{n,1} \leq 2$$
$$l_{3,2} + l_{4,2} + l_{5,2} - 3 \cdot t_{n,1} \leq 0$$
$$l_{3,1} + l_{4,1} + 2 \cdot t_{n,1} + 2 \cdot t_{n,2} \leq 4$$
$$l_{3,2} + l_{4,2} - 2 \cdot t_{n,2} \leq 0$$
$$l_{6,1} + 1 \cdot t_{n,2} \leq 1$$
$$l_{6,2} - 1 \cdot t_{n,1} - 1 \cdot t_{n,2} \leq 0$$

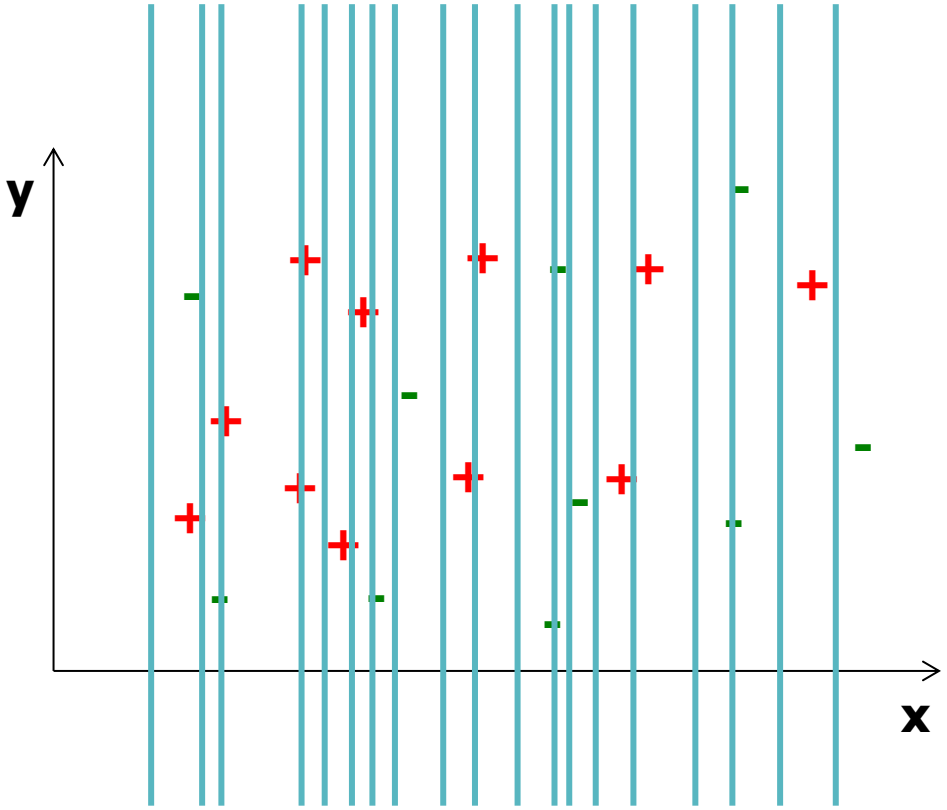


$$\min \textstyle\sum_{l,c} e_{l,c} \quad \text{s.t.}$$
$$\forall_n \quad \textstyle\sum_f f_{n,f} = 1$$
$$\forall_r \quad \textstyle\sum_l l_{r,l} = 1$$
$$\forall_l \quad \textstyle\sum_c p_{l,c} = 1$$
$$\forall_{n,f,b \in bin(f)} \quad M \cdot f_{n,f} + \textstyle\sum_{r \in lr(b)} \sum_{l \in ll(n)} l_{r,l} + \sum_{t \in tl(b)} M \cdot t_{n,t} - \sum_{t \in tl(b)} M \leq M$$
$$\forall_{n,f,b \in bin(f)} \quad M' \cdot f_{n,f} + \textstyle\sum_{r \in rr(b)} \sum_{l \in rl(n)} l_{r,l} - \sum_{t \in tl(b)} M' \cdot t_{n,t} \leq M'$$
$$\forall_{n,f} \quad M'' \cdot f_{n,f} + \textstyle\sum_{\max_{t(f)} < f(r)} \sum_{l \in ll(n)} l_{r,l} + \sum_{f(r) < \min_{t(f)}} \sum_{l \in rl(n)} l_{r,l} \leq M''$$
$$\forall_{l,c} \quad \textstyle\sum_{r : C_r = c} l_{r,l} - M''' \cdot p_l \leq e_{l,c}$$

Decision trees

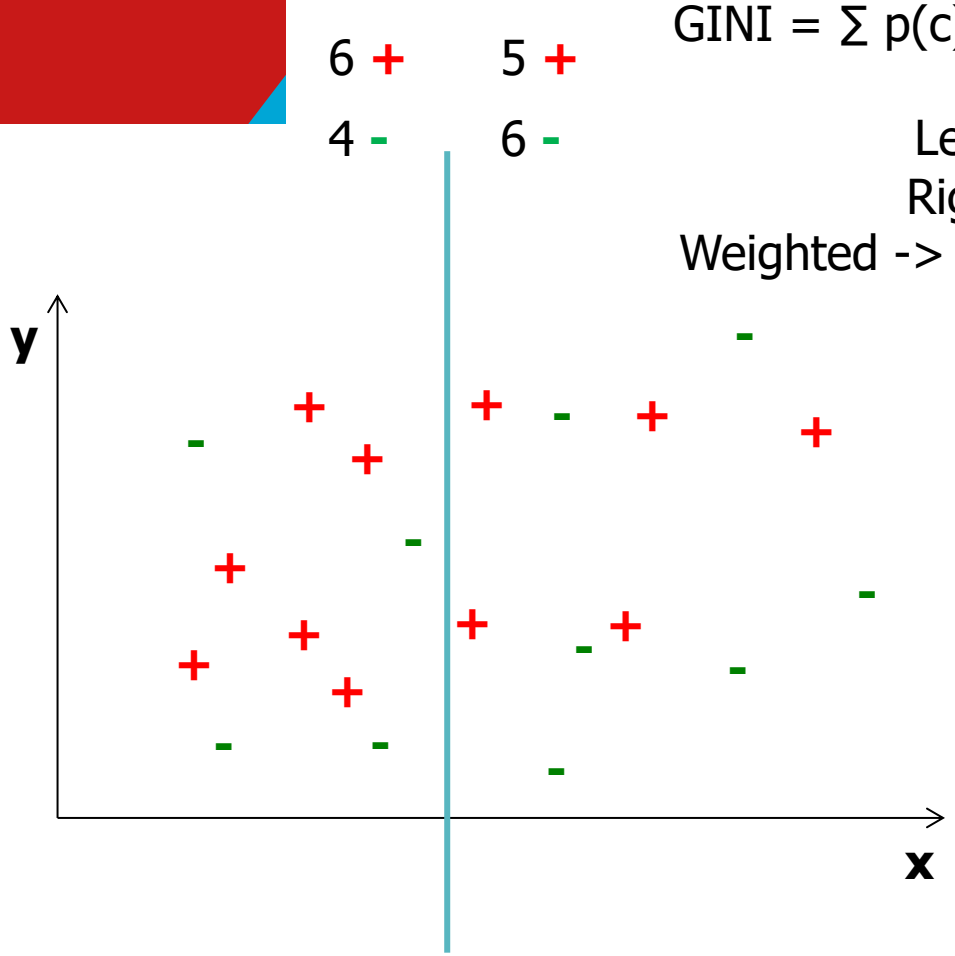


Key Algorithm: CART



1. Try all splits

Key Algorithm: CART



GINI = $\sum p(c)(1-p(c))$

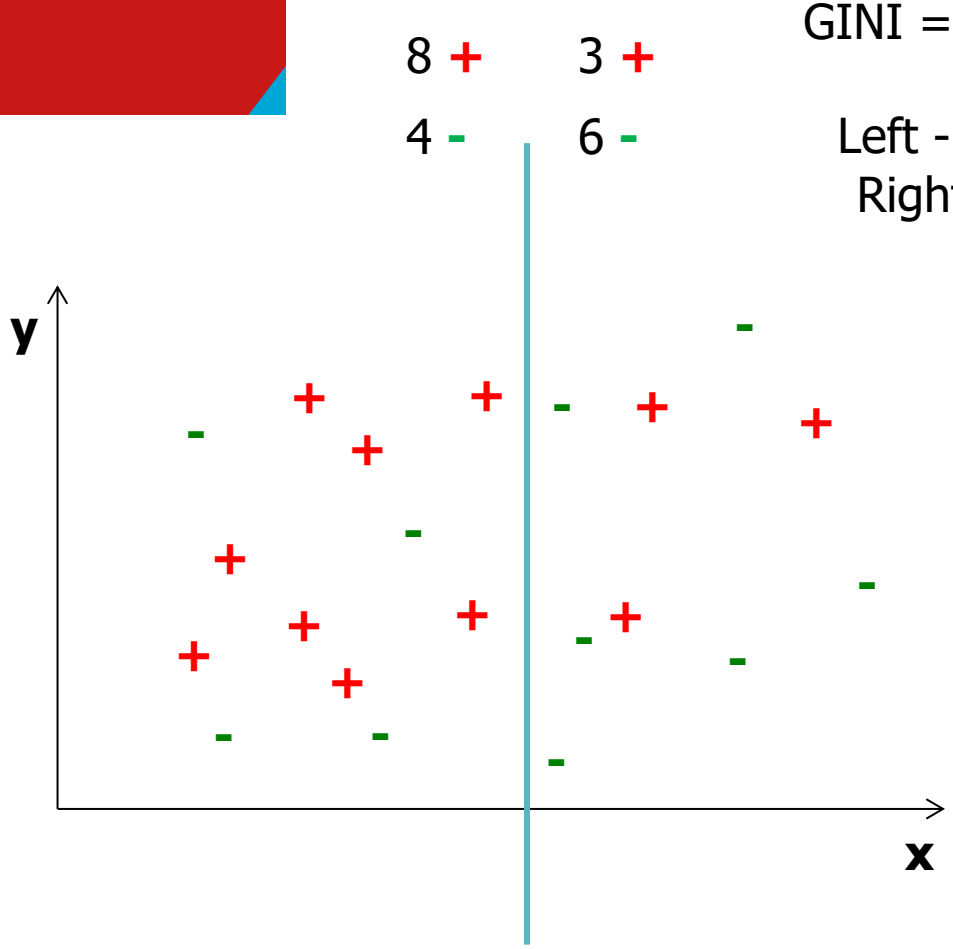
Left -> 0.6*0.4 + 0.4*0.6 = 0.48

Right -> 2*(5/11)*(6/11) = 0.496

Weighted -> (10/21) * 0.48 + (11/21) * 0.496 = 0.488

1. Try all splits
2. Compute their GINI impurity

Key Algorithm: CART

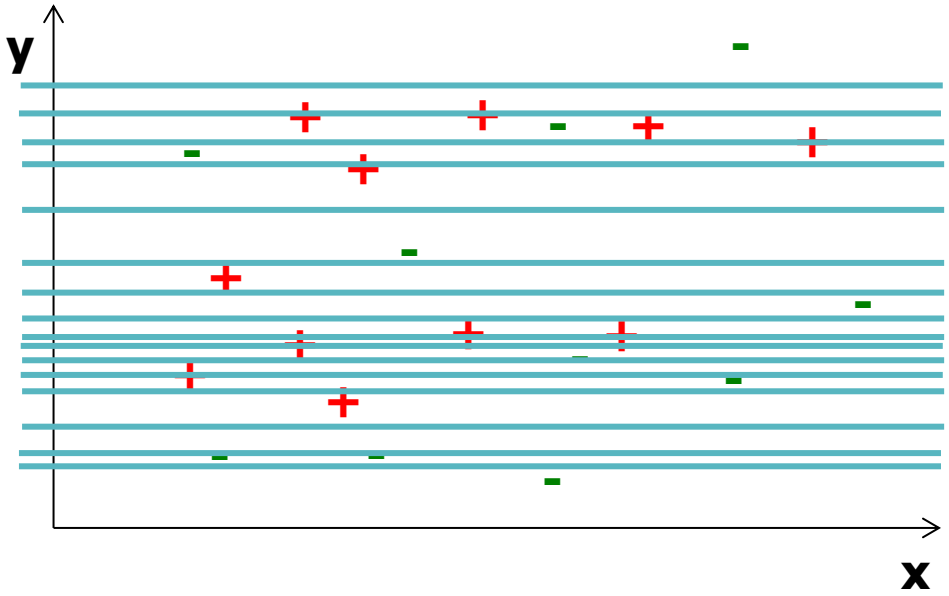


GINI = $\sum p(c)(1-p(c))$

Left -> 2*(8/12) *(4/12) = 0.44
Right -> 2*(3/9)*(6/9) = 0.44
Weighted -> 0.44

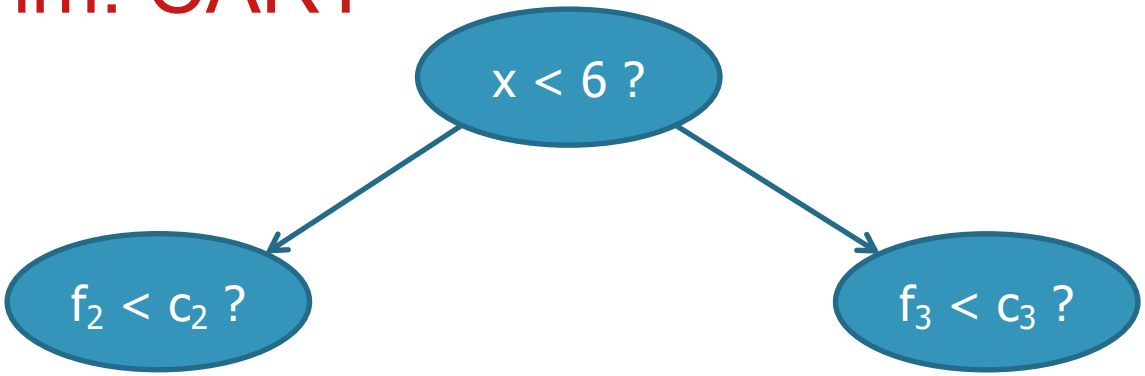
1. Try all splits
2. Compute their GINI impurity
3. Pick the smallest value split

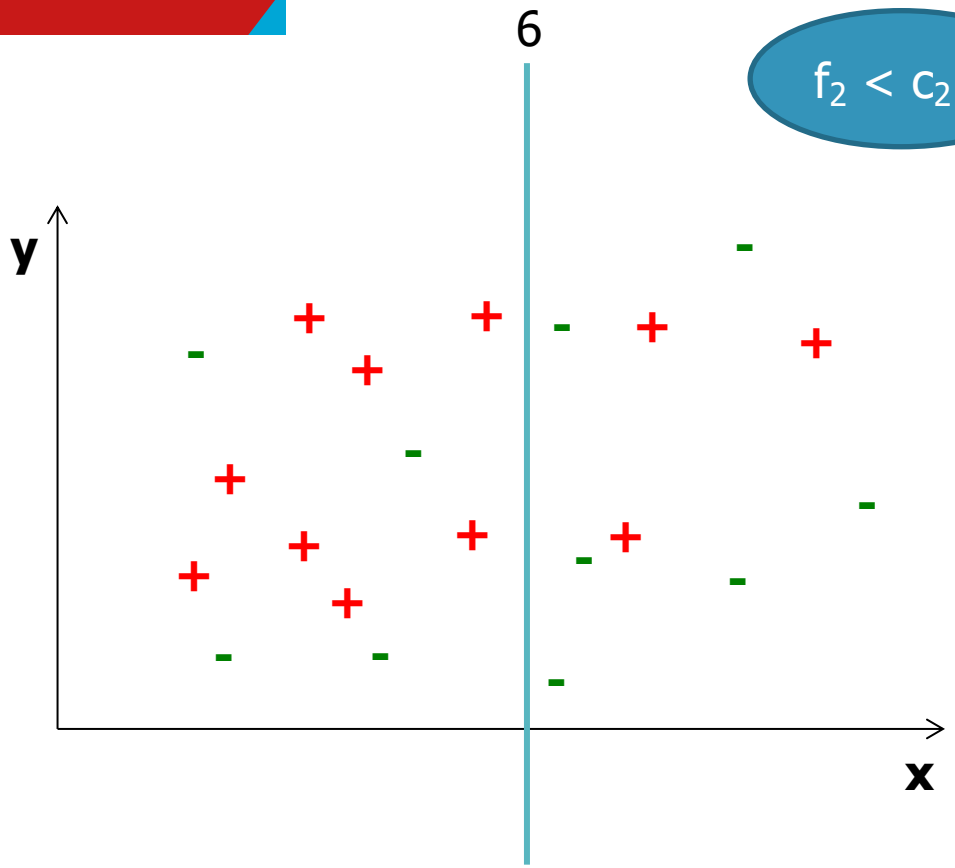
Key Algorithm: CART



1. Try all splits
2. Compute their GINI impurity
3. Pick the smallest value split
4. Do this for all features

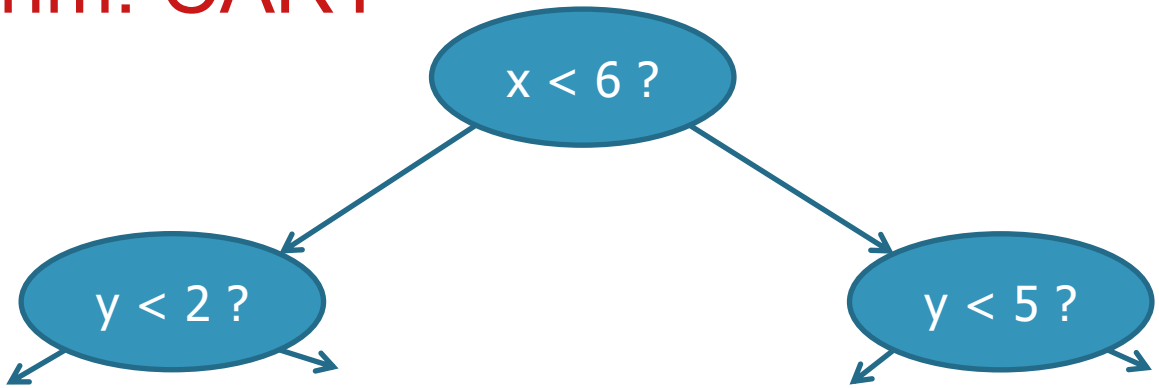
Key Algorithm: CART

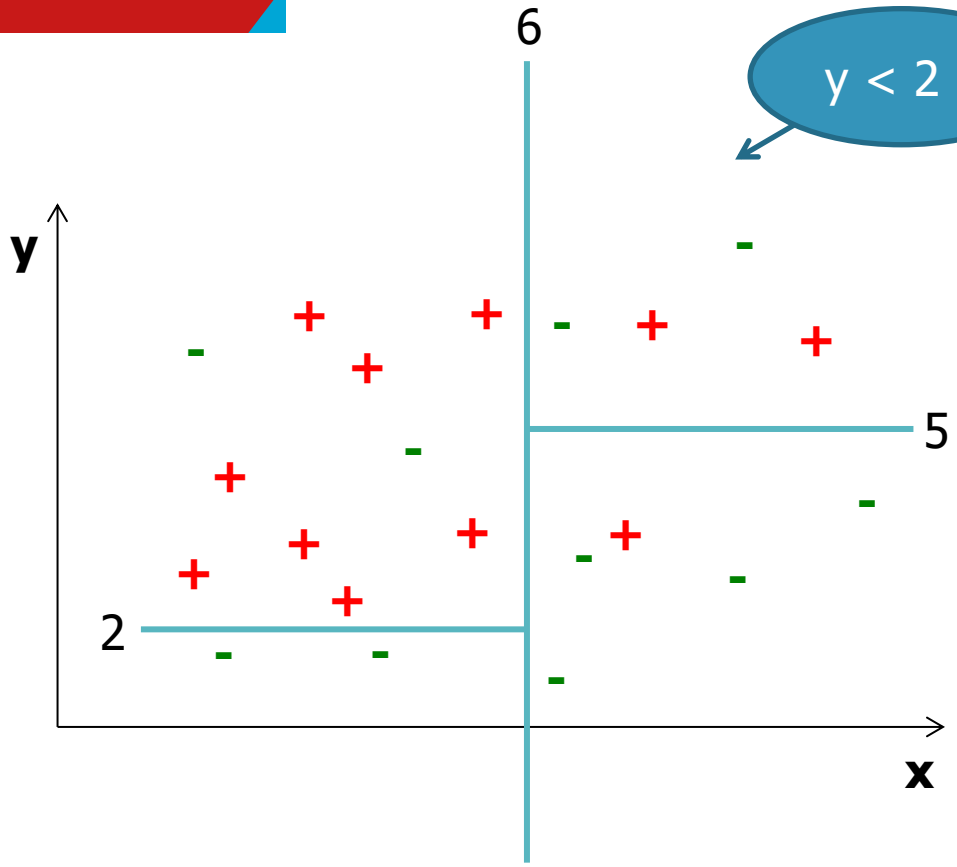




1. Try all splits
2. Compute their GINI impurity
3. Pick the smallest value split
4. Do this for all features
5. Perform the best split
6. Iterate for the new leafs

Key Algorithm: CART

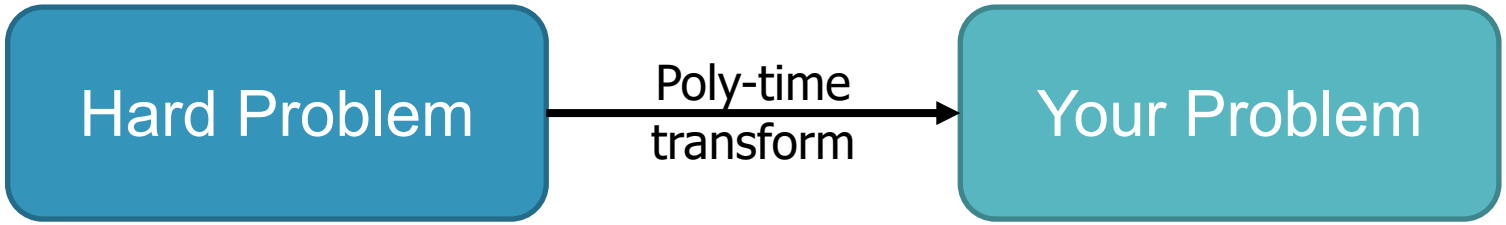




1. Try all splits
2. Compute their GINI impurity
3. Pick the smallest value split
4. Do this for all features
5. Perform the best split
6. Iterate for the new leafs

Why greedy?

- ML loves greedy algorithms
  - CART
  - Expectation Maximization
  - Gradient Descend
  - Agglomerative clustering
- But ML problems are often hard!

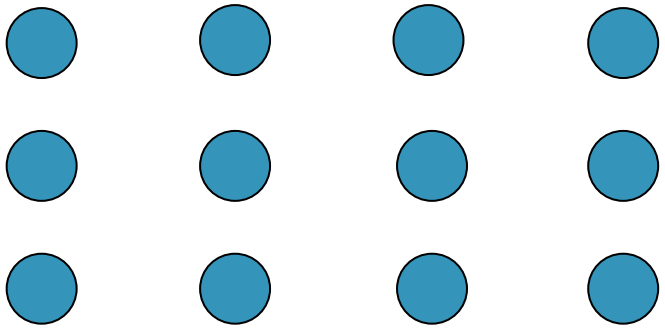


Learning Decision Trees is NP-hard

- Reduction from Exact Cover by 3 sets:
  - Given a set X = {x1, …, xn}, and subsets T = {T1, …, Tt},
    with |Ti| = 3 and Ti $\subset$ X
  - Find a subset T' of T such that $\cup$ T' = X and
    for Ti, Tj $\in$ T' : Ti $\cap$ Tj = $\emptyset$

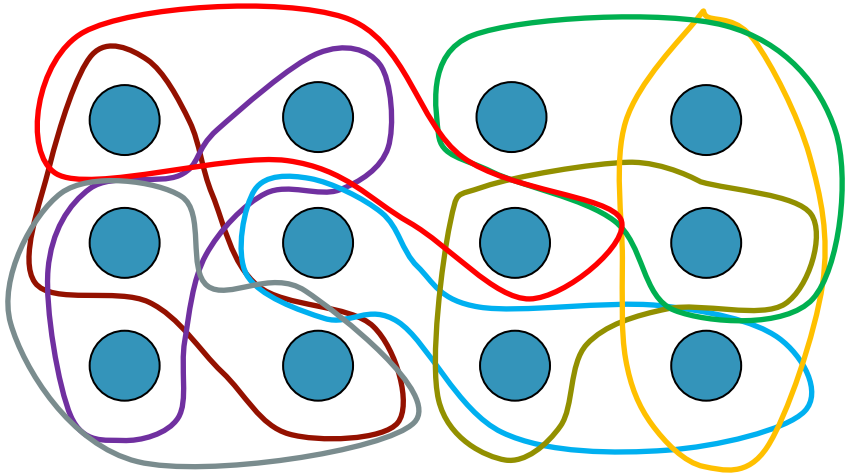
Learning Decision Trees is NP-hard

- Reduction from Exact Cover by 3 sets:
  - Given a set X = {x1, …, xn}, and subsets T = {T1, …, Tt},
    with |Ti| = 3 and Ti ⊂ X
  - Find a subset T' of T such that ∪ T' = X and
    for Ti, Tj ∈ T' : Ti ∩ Tj = ∅



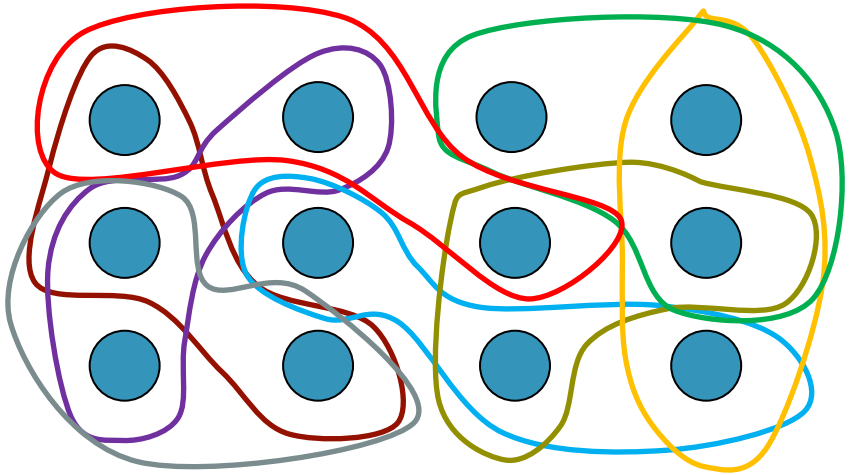
Learning Decision Trees is NP-hard

- Reduction from Exact Cover by 3 sets:
  - Given a set X = {x1, …, xn}, and subsets T = {T1, …, Tt}, with |Ti| = 3 and Ti ⊂ X
  - Find a subset T' of T such that ∪ T' = X and for Ti, Tj ∈ T' : Ti ∩ Tj = ∅



Learning Decision Trees is NP-hard

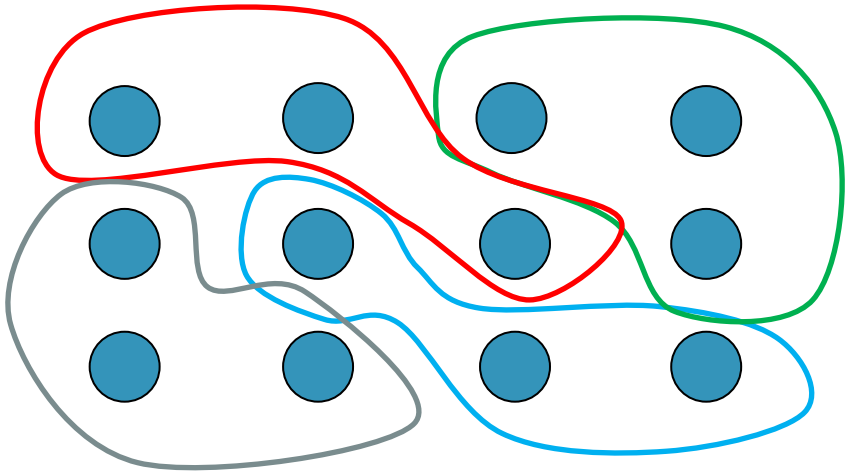
- Reduction from Exact Cover by 3 sets:
  - Given a set X = {x1, …, xn}, and subsets T = {T1, …, Tt}, with |Ti| = 3 and Ti ⊂ X
  - Find a subset T' of T such that ∪ T' = X and for Ti, Tj ∈ T' : Ti ∩ Tj = ∅



Does such a subset exist?

Learning Decision Trees is NP-hard

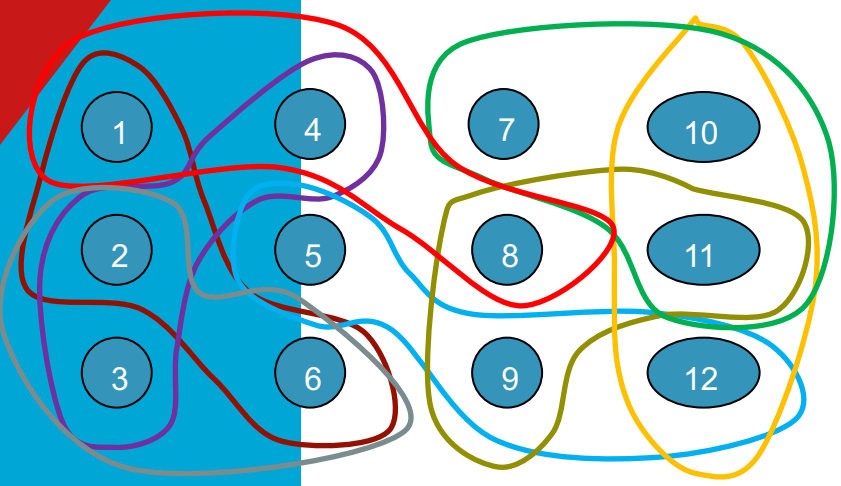
- Reduction from Exact Cover by 3 sets:
  - Given a set X = {x1, …, xn}, and subsets T = {T1, …, Tt},
    with |Ti| = 3 and Ti ⊂ X
  - Find a subset T' of T such that ∪ T' = X and
    for Ti, Tj ∈ T' : Ti ∩ Tj = ∅



Does such a subset exist?

The reduction

- Construct a data set:



	1	2	3	4	5	6	7	8	C
1	1	1	0	0	0	0	0	0	1
2	1	0	1	1	0	0	0	0	1
3	0	0	1	1	0	0	0	0	1
4	0	1	0	1	0	0	0	0	1
5	0	0	0	0	1	0	0	0	1
6	1	0	1	0	0	0	0	0	1
7	0	0	0	0	0	1	0	0	1
8	0	1	0	0	0	0	1	0	1
9	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	1	0	1	1
11	0	0	0	0	0	1	1	1	1
12	0	0	0	0	1	0	0	1	1
13	0	0	0	0	0	0	0	0	0

The reduction

- Construct a data set:

	1	2	3	4	5	6	7	8	C
1	1	1	0	0	0	0	0	0	1
2	1	0	1	1	0	0	0	0	1
3	0	0	1	1	0	0	0	0	1
4	0	1	0	1	0	0	0	0	1
5	0	0	0	0	1	0	0	0	1
6	1	0	1	0	0	0	0	0	1
7	0	0	0	0	0	1	0	0	1
8	0	1	0	0	0	0	1	0	1
9	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	1	0	1	1
11	0	0	0	0	0	1	1	1	1
12	0	0	0	0	1	0	0	1	1
13	0	0	0	0	0	0	0	0	0

There exists a DT with 12/3 = 4 decision nodes iff the X3C instance is true

The reduction

- Construct a data set:

	1	2	3	4	5	6	7	8	C
1	1	1	0	0	0	0	0	0	1
2	1	0	1	1	0	0	0	0	1
3	0	0	1	1	0	0	0	0	1
4	0	1	0	1	0	0	0	0	1
5	0	0	0	0	1	0	0	0	1
6	1	0	1	0	0	0	0	0	1
7	0	0	0	0	0	1	0	0	1
8	0	1	0	0	0	0	1	0	1
9	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	1	0	1	1
11	0	0	0	0	0	1	1	1	1
12	0	0	0	0	1	0	0	1	1
13	0	0	0	0	0	0	0	0	0

There exists a DT with 12/3 = 4 decision nodes iff the X3C instance is true

The reduction

- Does this problem look hard to you?

	1	2	3	4	5	6	7	8	C
1	1	1	0	0	0	0	0	0	1
2	1	0	1	1	0	0	0	0	1
3	0	0	1	1	0	0	0	0	1
4	0	1	0	1	0	0	0	0	1
5	0	0	0	0	1	0	0	0	1
6	1	0	1	0	0	0	0	0	1
7	0	0	0	0	0	1	0	0	1
8	0	1	0	0	0	0	1	0	1
9	0	0	0	0	1	0	1	0	1
10	0	0	0	0	0	1	0	1	1
11	0	0	0	0	0	1	1	1	1
12	0	0	0	0	1	0	0	1	1
13	0	0	0	0	0	0	0	0	0

The reduction

- Does this problem look hard to you?
- I would simple add a “union” feature…
- Is this how greedy algorithms solve NP-hard problems?

	1	2	3	4	5	6	7	8	U	C
1	1	1	0	0	0	0	0	0	1	1
2	1	0	1	1	0	0	0	0	1	1
3	0	0	1	1	0	0	0	0	1	1
4	0	1	0	1	0	0	0	0	1	1
5	0	0	0	0	1	0	0	0	1	1
6	1	0	1	0	0	0	0	0	1	1
7	0	0	0	0	0	1	0	0	1	1
8	0	1	0	0	0	0	1	0	1	1
9	0	0	0	0	1	0	1	0	1	1
10	0	0	0	0	0	1	0	1	1	1
11	0	0	0	0	0	1	1	1	1	1
12	0	0	0	0	1	0	0	1	1	1
13	0	0	0	0	0	0	0	0	0	0

The reduction

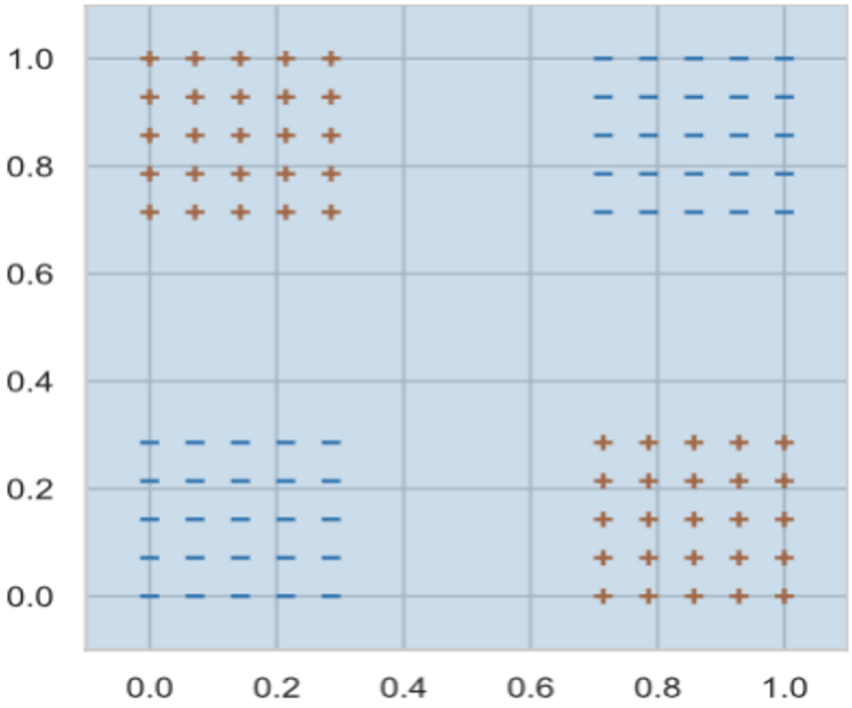
- Does this problem look hard to you?

	1	2	3	4	5	6	7	8	U	C
1	1	1	0	0	0	0	0	0	1	1
2	1	0	1	1	0	0	0	0	1	1
3	0	0	1	1	0	0	0	0	1	1
4	0	1	0	1	0	0	0	0	1	1
5	0	0	0	0	1	0	0	0	1	1
					0	0	0	0	1	1
					0	1	0	0	1	1
					0	0	1	0	1	1
					1	0	1	0	1	1
					0	1	0	1	1	1
					0	1	1	1	1	1
					1	0	0	1	1	1
13	0	0	0	0	0	0	0	0	0	0

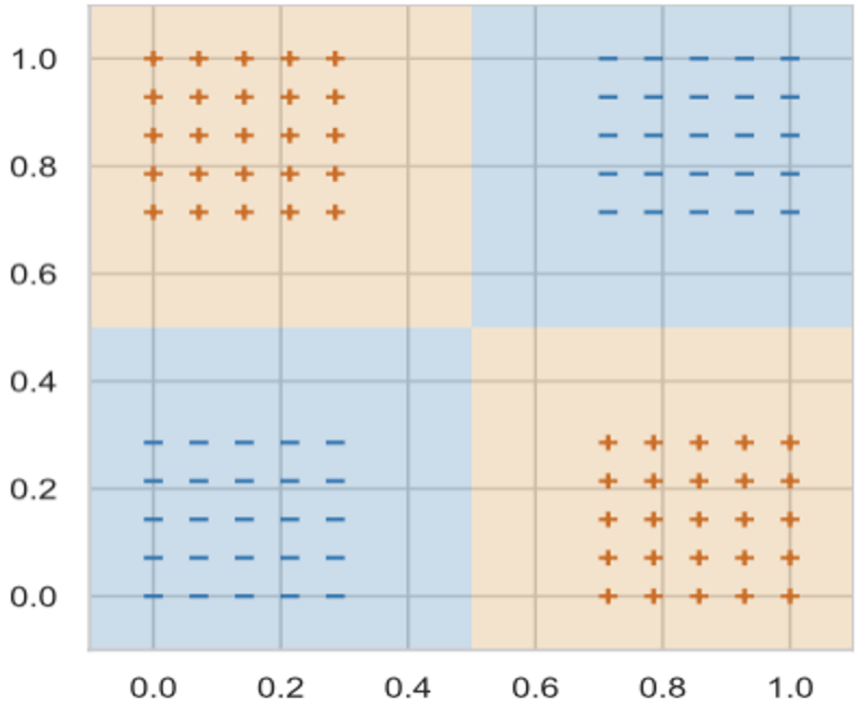
Typical in learning theory:

The problem is hard when forced to use a specific model, but can be easy when using a different one!

Greedy problems: no guarantees!



greedy



optimal

Learning optimal trees

- Via search and backtracking
  - Speeding up via dynamic programming
- Using local search or evolutionary algorithms
- By letting mathematical solvers search for you
  - Mixed Integer Programming (MIP)
  - Satisfiability (SAT)
  - Constraint Programming (CP)

Learning optimal trees

- Via search and backtracking
  - Speeding up via dynamic programming
- Using local search or evolutionary algorithms
- By letting mathematical solvers search for you
  - ***Mixed Integer Linear Programming (MILP)***
  - Satisfiability (SAT)
  - Constraint Programming (CP)

Learning by optimization

	f1	f2	f3	f4	t
1	5.1	3.5	1.4	0.2	s
2	4.9	3.0	1.4	0.2	s
3	4.7	3.2	1.3	0.3	v

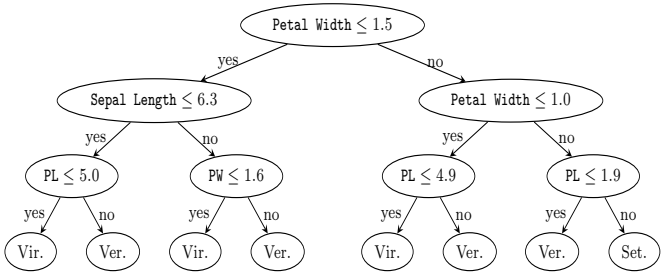
translate

$$l_{5,1} + l_{6,1} + 2 \cdot t_{n,1} \leq 2$$
$$l_{3,2} + l_{4,2} + l_{5,2} - 3 \cdot t_{n,1} \leq 0$$
$$l_{3,1} + l_{4,1} + 2 \cdot t_{n,1} + 2 \cdot t_{n,2} \leq 4$$
$$l_{3,2} + l_{4,2} - 2 \cdot t_{n,2} \leq 0$$
$$l_{6,1} + 1 \cdot t_{n,2} \leq 1$$
$$l_{6,2} - 1 \cdot t_{n,1} - 1 \cdot t_{n,2} \leq 0$$

add objective

$$\min \sum_{l,c} e_{l,c} \quad \text{s.t.}$$
$$\sum_{r:C_r=c} l_{r,l} - M \cdot p_{l,c} \leq e_{l,c}$$

solve



translate

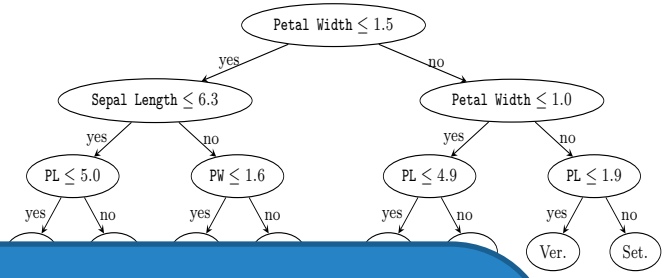
Learning by optimization

	f1	f2	f3	f4	t
1	5.1	3.5	1.4	0.2	s
2	4.9	3.0	1.4	0.2	s
3	4.7	3.2			

translate

$$\min \sum_{l,c} e_{l,c} \quad \text{s.t.}$$

$$\sum \; l_{r,l} - M \cdot p_{l,c} \leq e_{l,c}$$



Finds better solutions than CART

Bonus: **Flexibility**

Create your own objective function!
Add your own specific constraints!

$$l_{6,2} - 1 \cdot t_{n,1} - 1 \cdot t_{n,2} \leq 0$$

Encoding decision trees

- NP-hard optimization problem:
  - Finding an optimal classification/regression tree of depth at most k for a given dataset of n rows (samples) and m features/attributes

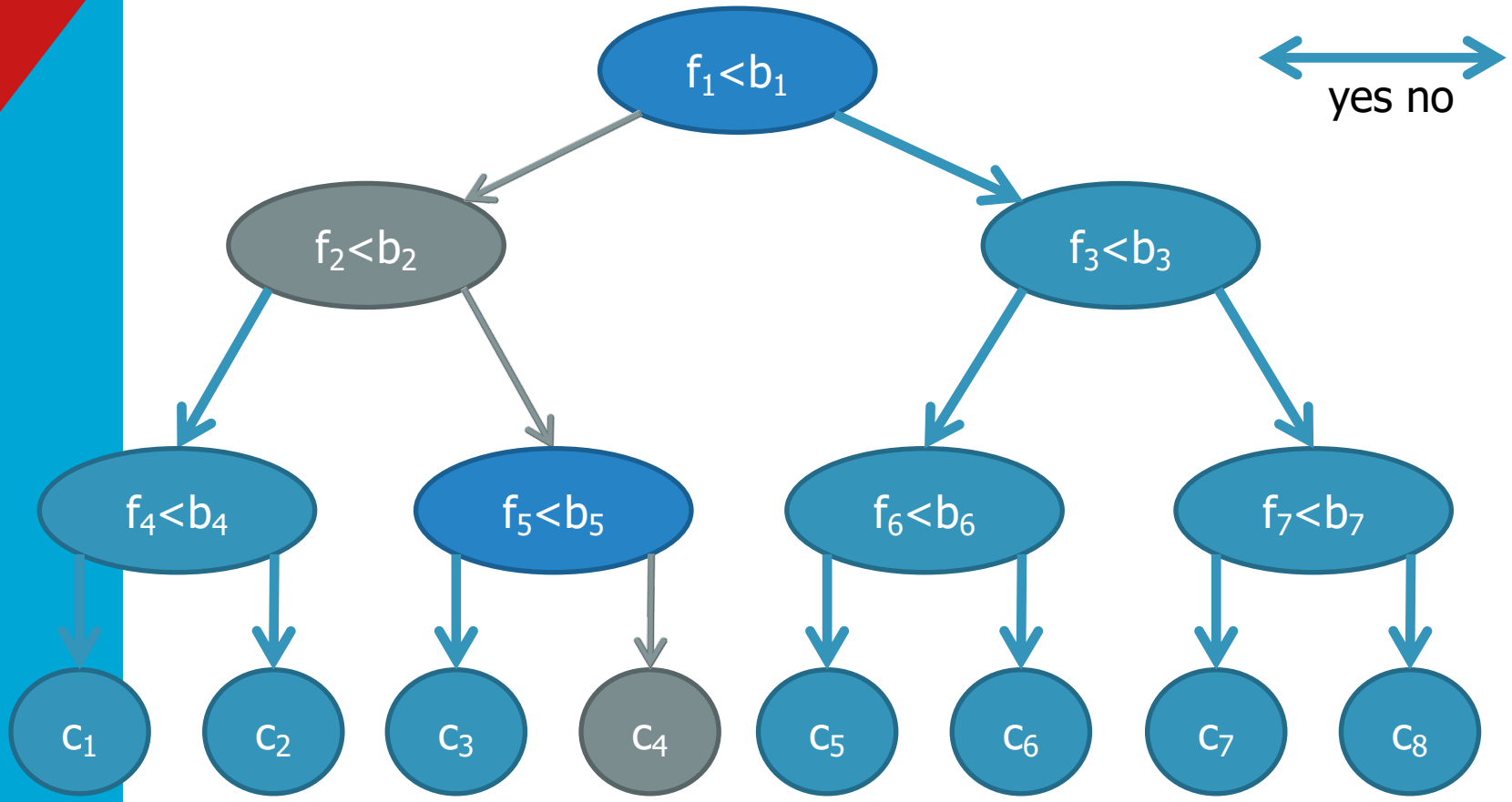
- 3 parts:
  - encoding internal nodes – paths to leaves (this talk)
  - encoding leaves – what to predict for each row (see paper)
  - encoding the objective (see paper)

TU/e

Based on

DTLP (Verwer & Zhang 2017)

OCT (Bertsimas & Dunn 2017)



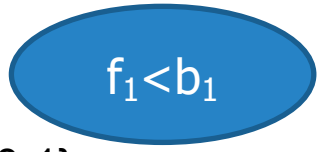
TUDelft

Gurobi

- Open your notebook!
- Three key API functions:
  - **model.addConstr(…)**
  - **gp.quicksum(… for i in …)**
  - **model.addVars(…)**

Encoding the decision tree

- Decision nodes

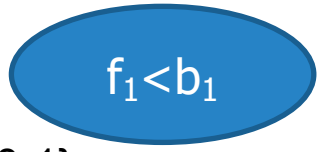


  - Node m splits on feature j: $a_{j,m}$ in {0,1}
  - Using decision threshold b: b_m in [0,1]

  - Each node splits on exactly 1 feature:

Encoding the decision tree

- Decision nodes



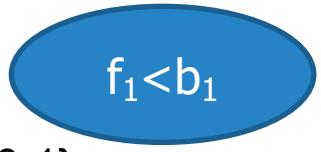
  - Node m splits on feature j: $a_{j,m}$ in {0,1}
  - Using decision threshold b: b_m in [0,1]

  - Each node splits on exactly 1 feature:

    - $\text{sum}_j\ a_{j,m} = 1$

Encoding the decision tree

- Decision nodes



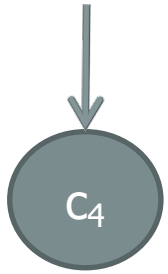
  - Node m splits on feature j: $a_{j,m}$ in {0,1}
  - Using decision threshold b: b_m in [0,1]

  - Each node splits on exactly 1 feature:

    - $\text{sum}_j\ a_{j,m} = 1$

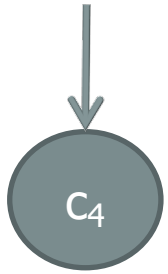
```
for m in nodes:
    model.addConstr(gp.quicksum(a[j, m] for j in range(n_features)) == 1)
```

Encoding the decision tree



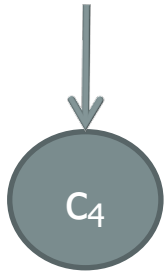
- Leaf nodes
  - Leaf t predicts class y: $c_{t,y}$ in {0,1}
  - Each leaf predicts exactly 1 class:

Encoding the decision tree



- Leaf nodes
  - Leaf t predicts class y: $c_{t,y}$ in {0,1}
  - Each leaf predicts exactly 1 class:
    - $\text{sum}_y\ c_{t,y} = 1$

Encoding the decision tree

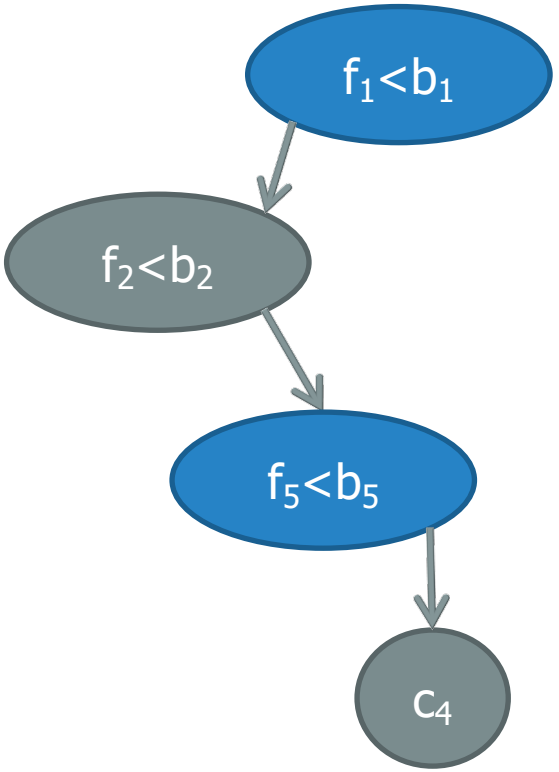


- Leaf nodes
  - Leaf t predicts class y: $c_{t,y}$ in {0,1}
  - Each leaf predicts exactly 1 class:
    - $\text{sum}_y\ c_{t,y} = 1$

```
for t in leaves:
    model.addConstr(gp.quicksum(c[t, k] for k in range(n_classes)) == 1)
```

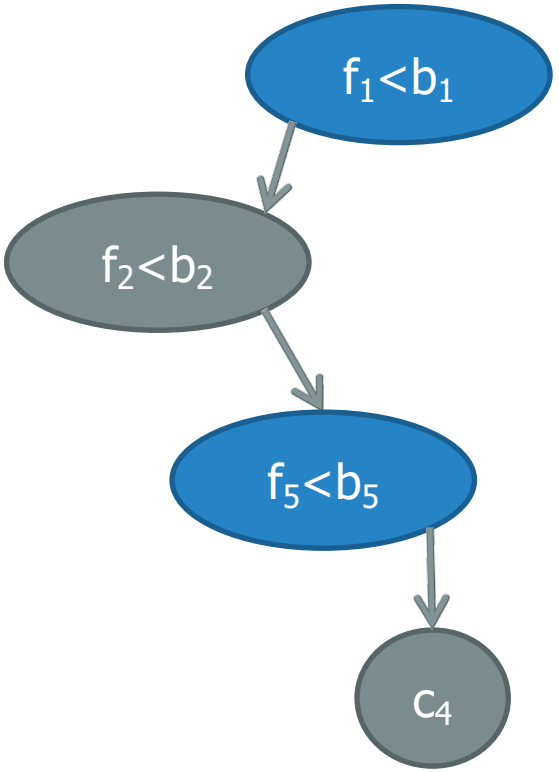
Encoding the decision tree

- The path of data rows
  - Row i ends in leaf t: $z_{i,t}$ in {0,1}
  - Each row ends in exactly 1 leaf:



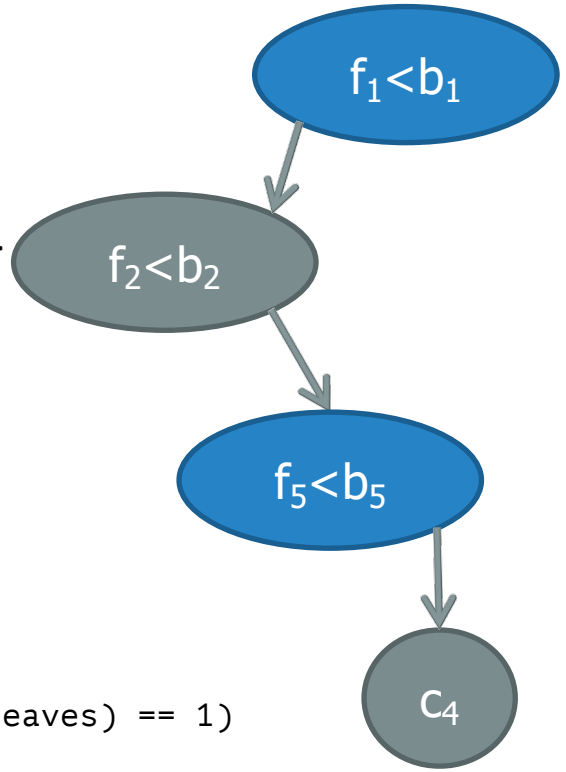
Encoding the decision tree

- The path of data rows
  - Row i ends in leaf t: $z_{i,t}$ in {0,1}
  - Each row ends in exactly 1 leaf:
    - $\text{sum}_t\ z_{i,t} = 1$



Encoding the decision tree

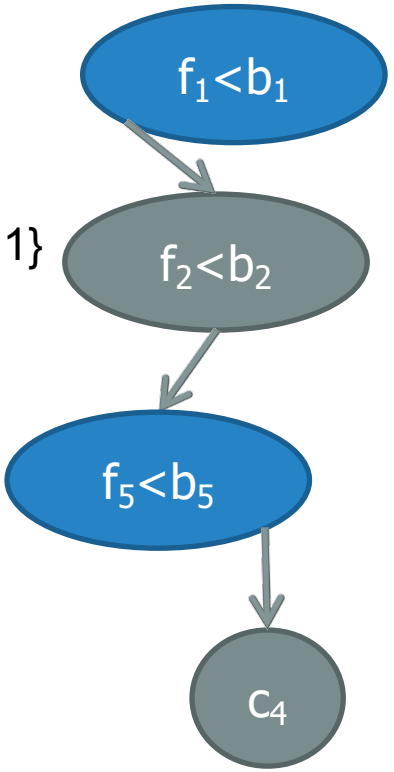
- The path of data rows
  - Row i ends in leaf t: $z_{i,t}$ in {0,1}
  - Each row ends in exactly 1 leaf:
    - $\text{sum}_t\ z_{i,t} = 1$




```
for i in range(n_samples):
    model.addConstr(gp.quicksum(z[i, t] for t in leaves) == 1)
```

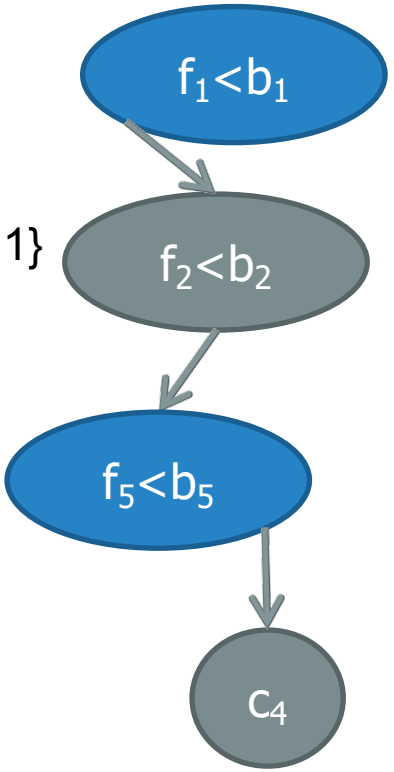
Encoding the decision tree

- The path of data rows
  - Node m splits on feature j: $a_{j,m}$ in {0,1}
  - Row i has **constant** feature values: $X_{i,j}$
  - How to obtain the value of feature f_m for node m?



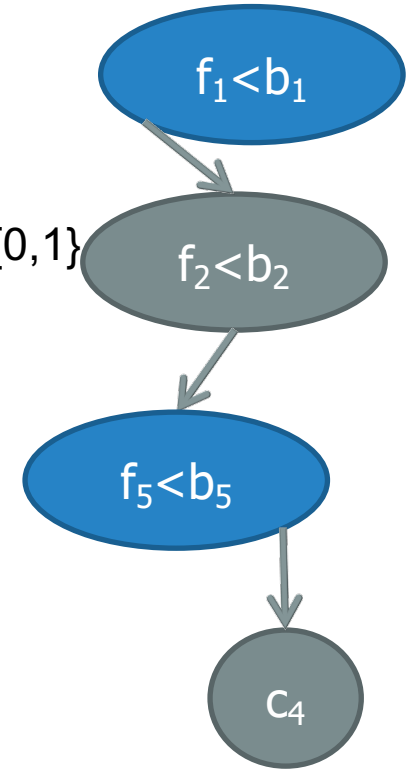
Encoding the decision tree

- The path of data rows
  - Node m splits on feature j: $a_{j,m}$ in {0,1}
  - Row i has **constant** feature values: $X_{i,j}$
  - How to obtain the value of feature f_m for node m?
    - $f_m = \text{sum}_j\ X_{i,j} * a_{j,m}$



Encoding the decision tree

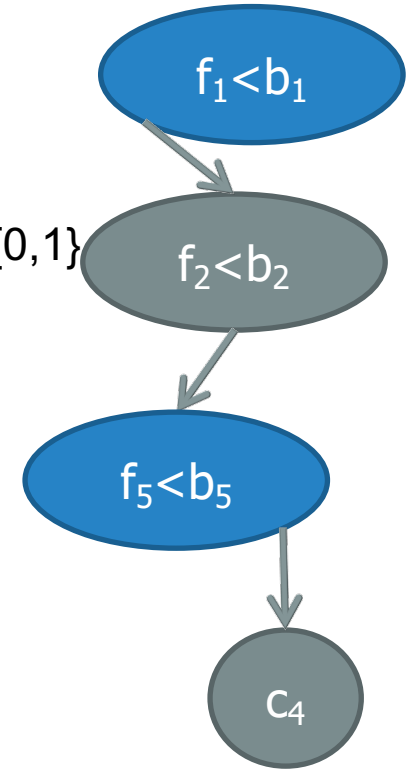
- The path of data rows
  - Node m splits on feature j: aj,m in {0,1}
  - Row i has constant feature values: Xi,j
  - How to obtain the value of feature fm for node m?
    - fm = sumj Xi,j * aj,m




```
for i in range(n_samples):
    for m in nodes:
        model.addConstr(f[i, m] == gp.quicksum(a[j, m] * X[i, j] for j in
          range(n_features)))
```

Encoding the decision tree

- The path of data rows
  - Node m splits on feature j: aj,m in {0,1}
  - Row i has constant feature values: Xi,j
  - How to obtain the value of feature fm for node m?
    - fm = sumj Xi,j * aj,m

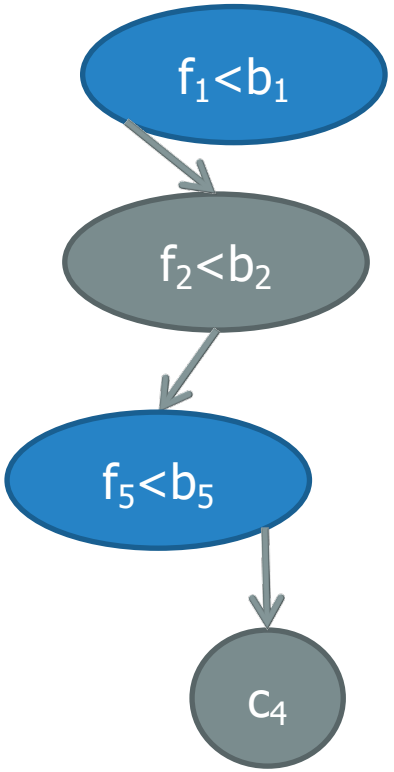



```
for i in range(n_samples):
    for m in nodes:
        model.addConstr(f[i, m] == gp.quicksum(a[j, m] * X[i, j] for j in
          range(n_features)))
```

This is a constant!

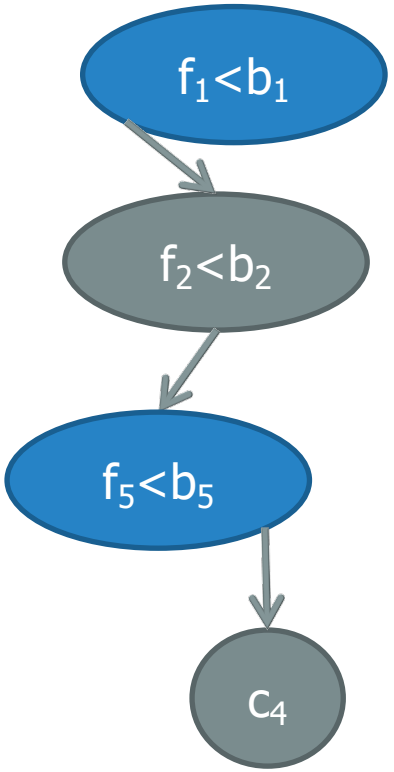
Encoding the decision tree

- The path of data rows
  - If row i ends in leaf t ($z_{i,t} = 1$),
  - Then for each right ancestor node m:
    - $f_m >= b_m$
  - How to force this constraint when $z_{i,t} = 1$?



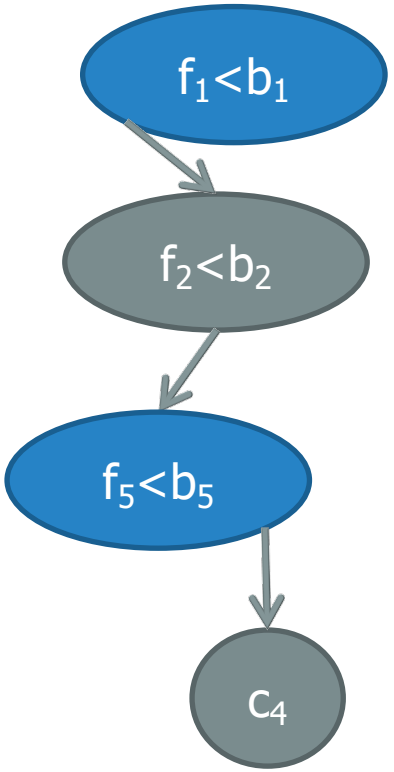
Encoding the decision tree

- The path of data rows
  - If row i ends in leaf t ($z_{i,t} = 1$),
  - Then for each right ancestor node m:
    - $f_m >= b_m$
  - How to force this constraint when $z_{i,t} = 1$?
    - $f_m >= b_m - (1 - z_{i,t})$



Encoding the decision tree

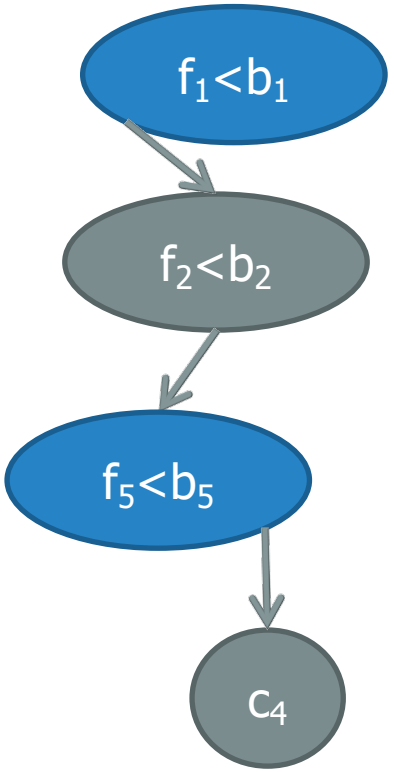
- The path of data rows
  - If row i ends in leaf t ($z_{i,t} = 1$),
  - Then for each right ancestor node m:
    - $f_m >= b_m$
  - How to force this constraint when $z_{i,t} = 1$?
    - $f_m >= b_m - (1 - z_{i,t})$




```
for i in range(n_samples):
    for t in leaves:
        A_l, A_r = ancestors(t)
        for m in A_r:
            model.addConstr(f[i, m] >= b[m] - (1 - z[i, t]))
```

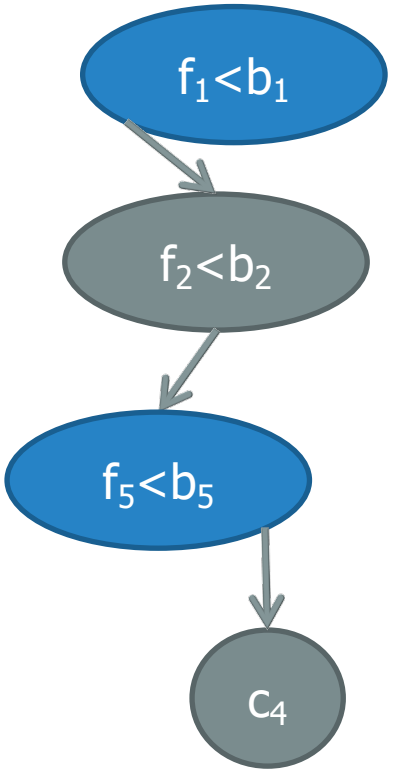
Encoding the decision tree

- The path of data rows
  - If row i ends in leaf t ($z_{i,t} = 1$),
  - Then for each left ancestor node m:
    - $f_m < b_m$
  - How to force this constraint when $z_{i,t} = 1$?



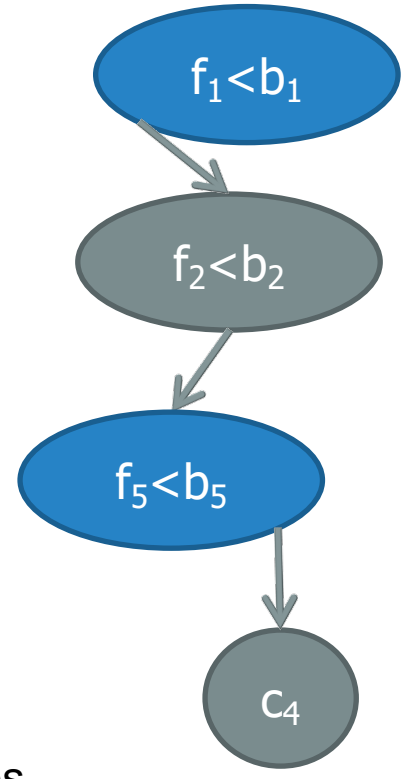
Encoding the decision tree

- The path of data rows
  - If row i ends in leaf t ($z_{i,t} = 1$),
  - Then for each left ancestor node m:
    - $f_m < b_m$
  - How to force this constraint when $z_{i,t} = 1$?
    - $f_m < b_m + (1 - z_{i,t})$



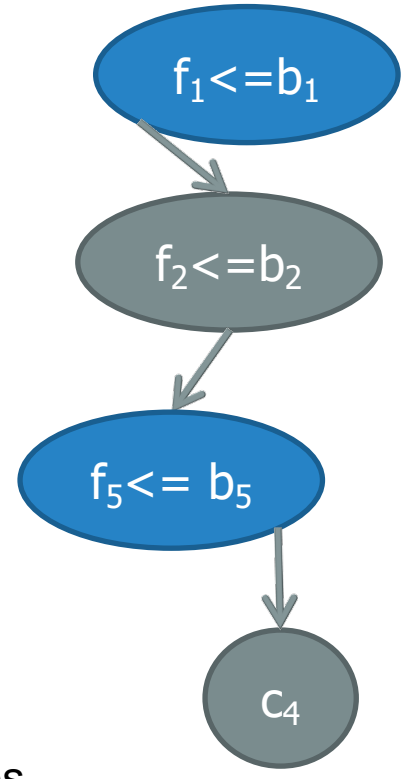
Encoding the decision tree

- The path of data rows
  - If row i ends in leaf t ($z_{i,t} = 1$),
  - Then for each left ancestor node m:
    - $f_m < b_m$
  - How to force this constraint when $z_{i,t} = 1$?
    - $f_m < b_m + (1 - z_{i,t})$
  - But MILP solvers only handle non-strict inequalities…



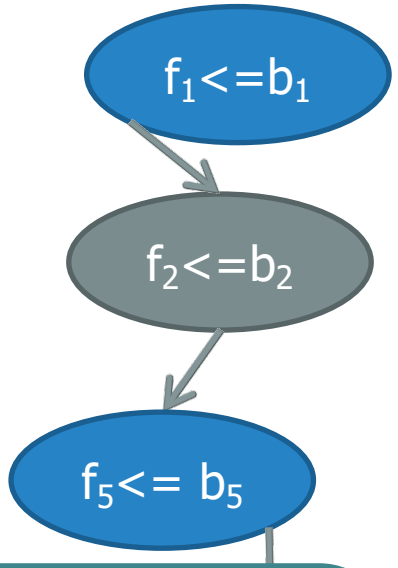
Encoding the decision tree

- The path of data rows
  - If row i ends in leaf t ($z_{i,t} = 1$),
  - Then for each left ancestor node m:
    - $f_m < b_m$
  - How to force this constraint when $z_{i,t} = 1$?
    - $f_m + \varepsilon <= b_m + (1 - z_{i,t})*(1 + \varepsilon)$
  - But MILP solvers only handle non-strict inequalities…



Encoding the decision tree

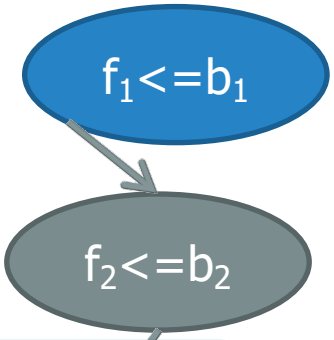
- The path of data rows
  - If row i ends in leaf t ($z_{i,t} = 1$),
  - Then for each left ancestor node m:
    - $f_m < b_m$
  - How to force this constraint when $z_{i,t} = 1$?




```
for i in range(n_samples):
    for t in leaves:
        A_l, A_r = ancestors(t)
        for m in A_l:
            model.addConstr(f[i, m] + epsilon <= b[m] + (1 + epsilon) * (1 - z[i, t]))
```

Encoding the decision tree

- The path of data rows
  - If row i ends in leaf t ($z_{i,t} = 1$),
  - Then for each left ancestor node m:
    - $f_m < b_m$
  - How to force this constraint w
    - $f_m + \varepsilon <= b_m + (1 - z_{i,t})*(1 +$
  - But MILP solvers only handle

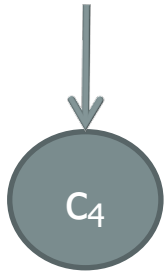


Inequalities of the form

$z * M + v >= v'$

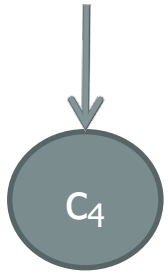
are called big-M constraints, with M an upperbound on v (in our case 1 + ε) an easy way to encode logical constraints in MILP

Making predictions



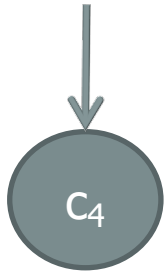
- We have
  - Row i ends in leaf t: $z_{i,t}$
  - Leaf t predicts class y: $c_{t,y}$
  - Row i has class y_i as label (a constant)
- How to use these to define the error e_i for row i?

Making predictions



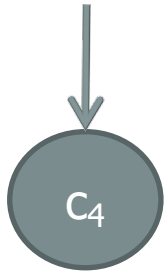
- We have
  - Row i ends in leaf t: $z_{i,t}$
  - Leaf t predicts class y: $c_{t,y}$
  - Row i has class y_i as label (a constant)
- How to use these to define the error e_i for row i?
- If $y_i \neq y$:

Making predictions



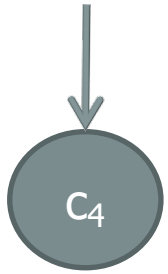
- We have
  - Row i ends in leaf t: $z_{i,t}$
  - Leaf t predicts class y: $c_{t,y}$
  - Row i has class y_i as label (a constant)
- How to use these to define the error e_i for row i?
- If $y_i \neq y$:
  - $e_i >= z_{i,t} + c_{t,y} - 1$

Making predictions



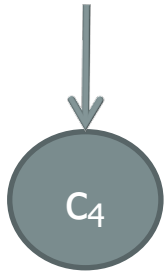
- We have
  - Row i ends in leaf t: $z_{i,t}$
  - Leaf t predicts class y: $c_{t,y}$
  - Row i has class y_i as label (a constant)
- How to use these to define the error e_i for row i?
- If $y_i \neq y$:
  - $e_i >= z_{i,t} + c_{t,y} - 1$
- If $y_i = y$:

Making predictions



- We have
  - Row i ends in leaf t: $z_{i,t}$
  - Leaf t predicts class y: $c_{t,y}$
  - Row i has class y_i as label (a constant)
- How to use these to define the error e_i for row i?
- If $y_i \neq y$:
  - $e_i >= z_{i,t} + c_{t,y} - 1$
- If $y_i = y$:
  - no constraint needed!

Making predictions



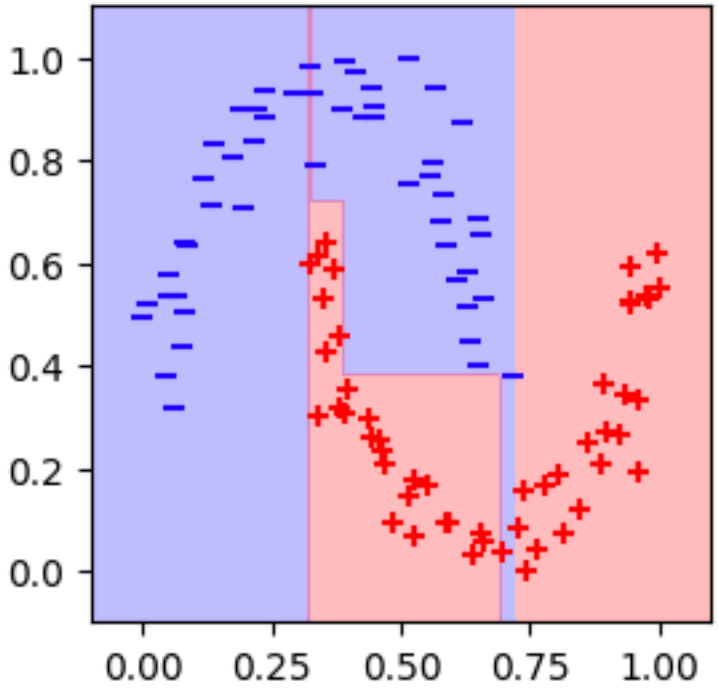
- We have
  - Row i ends in leaf t: $z_{i,t}$
  - Leaf t predicts class y: $c_{t,y}$
  - Row i has class y_i as label (a constant)
- How to use these to define the error e_i for row i?
- If $y_i \neq y$:
  - $e_i >= z_{i,t} + c_{t,y} - 1$
- If $y_i = y$:
  - no constraint needed!


```
for i in range(n_samples):
    for t in leaves:
      model.addConstr(e[i] >= z[i, t] + (1 - c[t, y[i]]) - 1)
```

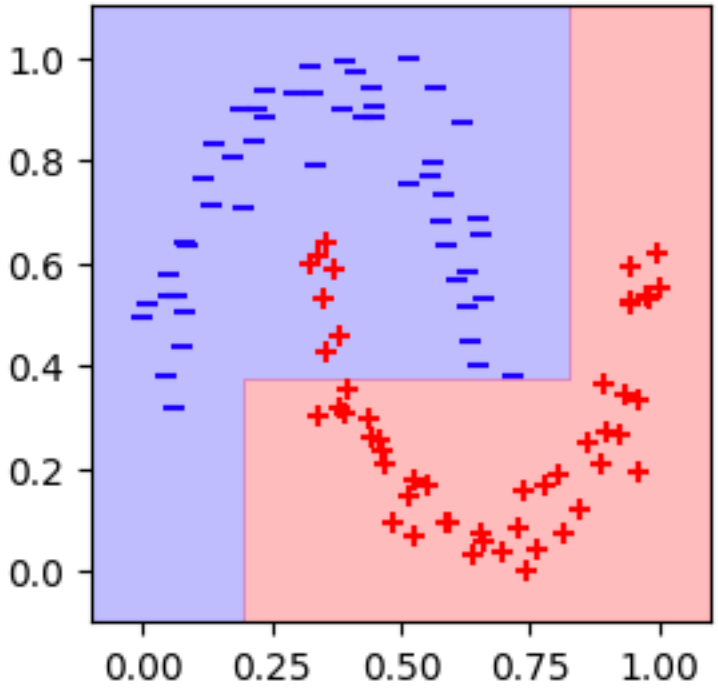
Code for GUROBI

```
for m in nodes:
    model.addConstr(gp.quicksum(a[j, m] for j in range(n_features)) == 1)
for t in leaves:
    model.addConstr(gp.quicksum(c[t, k] for k in range(n_classes)) == 1)
for i in range(n_samples):
    model.addConstr(gp.quicksum(z[i, t] for t in leaves) == 1)
    for m in nodes:
        model.addConstr(f[i, m] == gp.quicksum(a[j, m] * X[i, j] for j in
range(n_features)))
    for t in leaves:
        A_l, A_r = ancestors(t)
        for m in A_r:
            model.addConstr(f[i, m] >= b[m] - (1 - z[i, t]))
        for m in A_l:
            model.addConstr(f[i, m] + epsilon <= b[m] + (1 + epsilon) * (1 - z[i, t]))
        model.addConstr(e[i] >= z[i, t] + (1 - c[t, y[i]]) - 1)
```

You should get something like



Using your encoding



Using CART

This works, now what?

- Better trees than CART, but slower
- SOTA contains many improvements/tricks
  - BinOCT (Verwer & Zhang, 2019)
  - Optimal Sparse Trees (Hu, Rudin & Seltzer, 2019)
  - Encoding in Constraint Programming (Verhaege et al. 2020)
  - Using Itemset Mining (Aglin, Nijssen, Schaus, 2020)
  - Using Satisfiability (Avellaneda, 2020 - Schidler & Szeider, 2021)
  - With Max-Flow (Aghaei, Gomez, Vayanos, 2020)
  - Via Column Generation (Firat et al. 2020)
  - Via Dynamic Programming (Demirovic et al. 2022)
  - …

TU/e

Exercises:

- Mess around:
  - When is the MILP slow? What parameter(s) influences it most?
  - Find a case where your encoding is much better than CART!
  - ...
- Break symmetries to speedup:
  - For a pair of leaves t, t' from one parent, ensure their predictions are different
  - …
- Extend:
  - Find a tree with Oblique (Multivariate) splits
  - Force a leaf to have at least N samples
  - Make the classifier fair, ensure that the number of errors is equal (+- 1) over all classes
  - Aim for 0 false positives (no negatives classified as positive)
  - Minimize the tree size, e.g., by forcing some decision nodes to return true

TUDelft

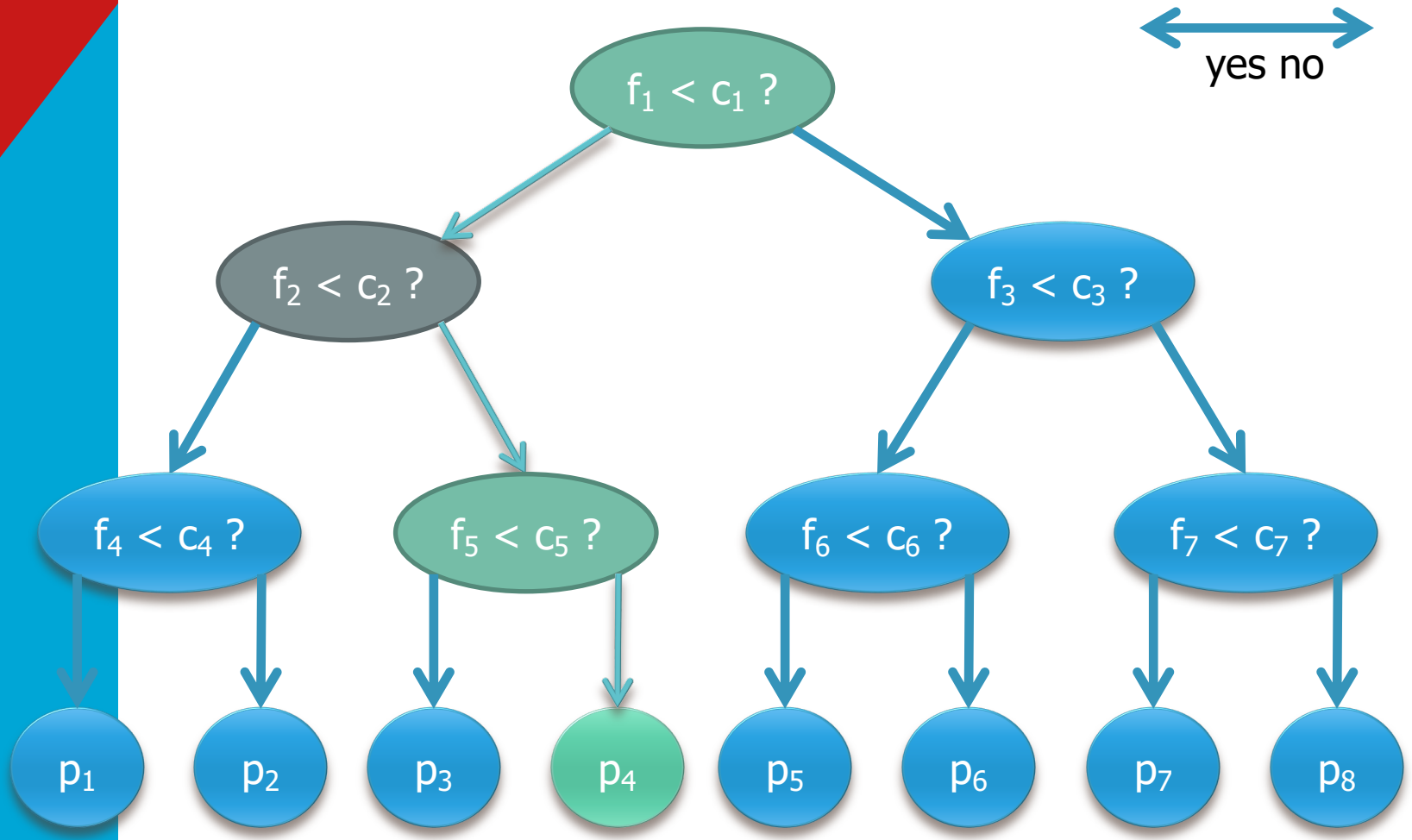
TU/e

BinOCT

- Sicco Verwer, Yingqian Zhang
- AAAI, 2019

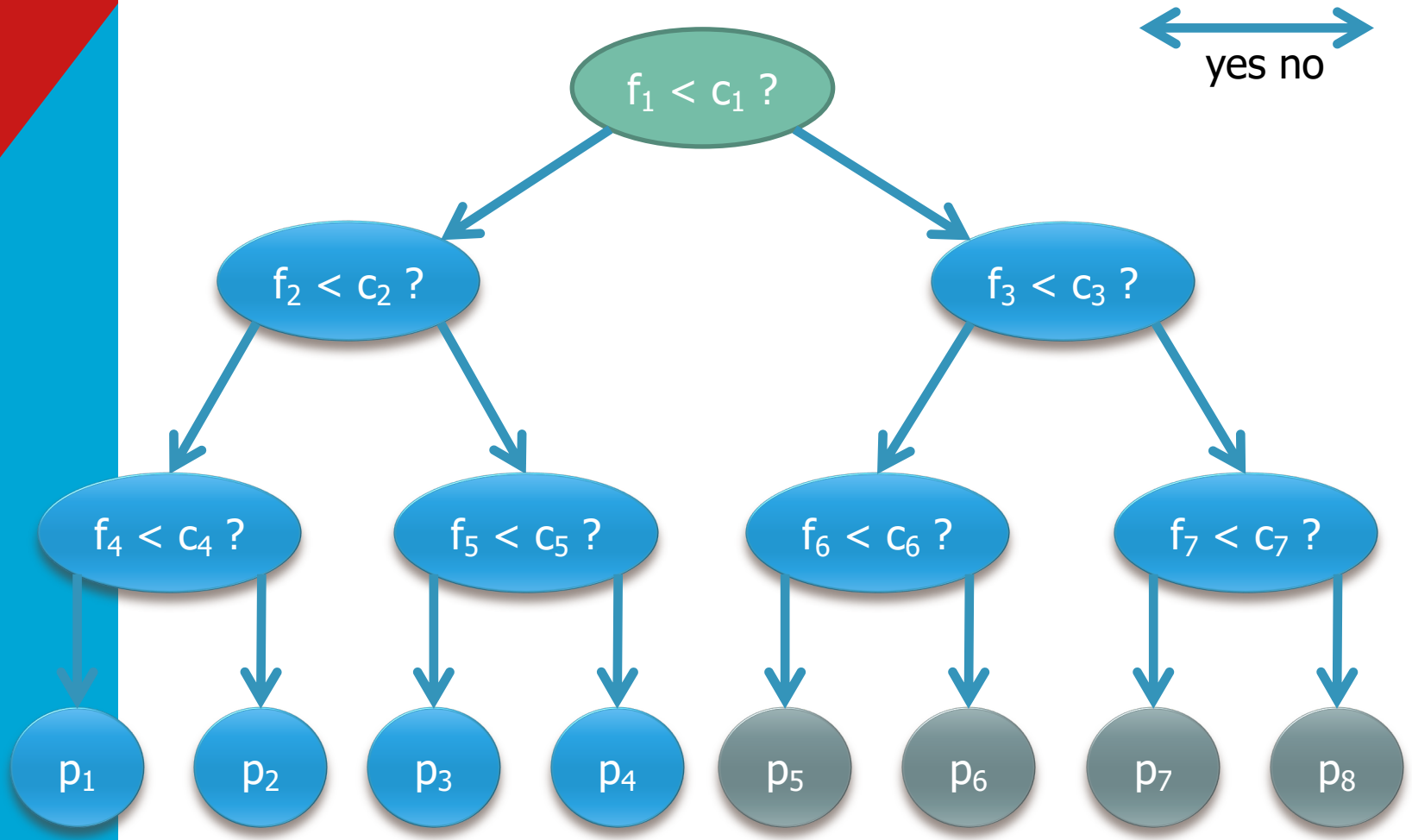
TUDelft

OCT (Bertsimas & Dunn 2017):



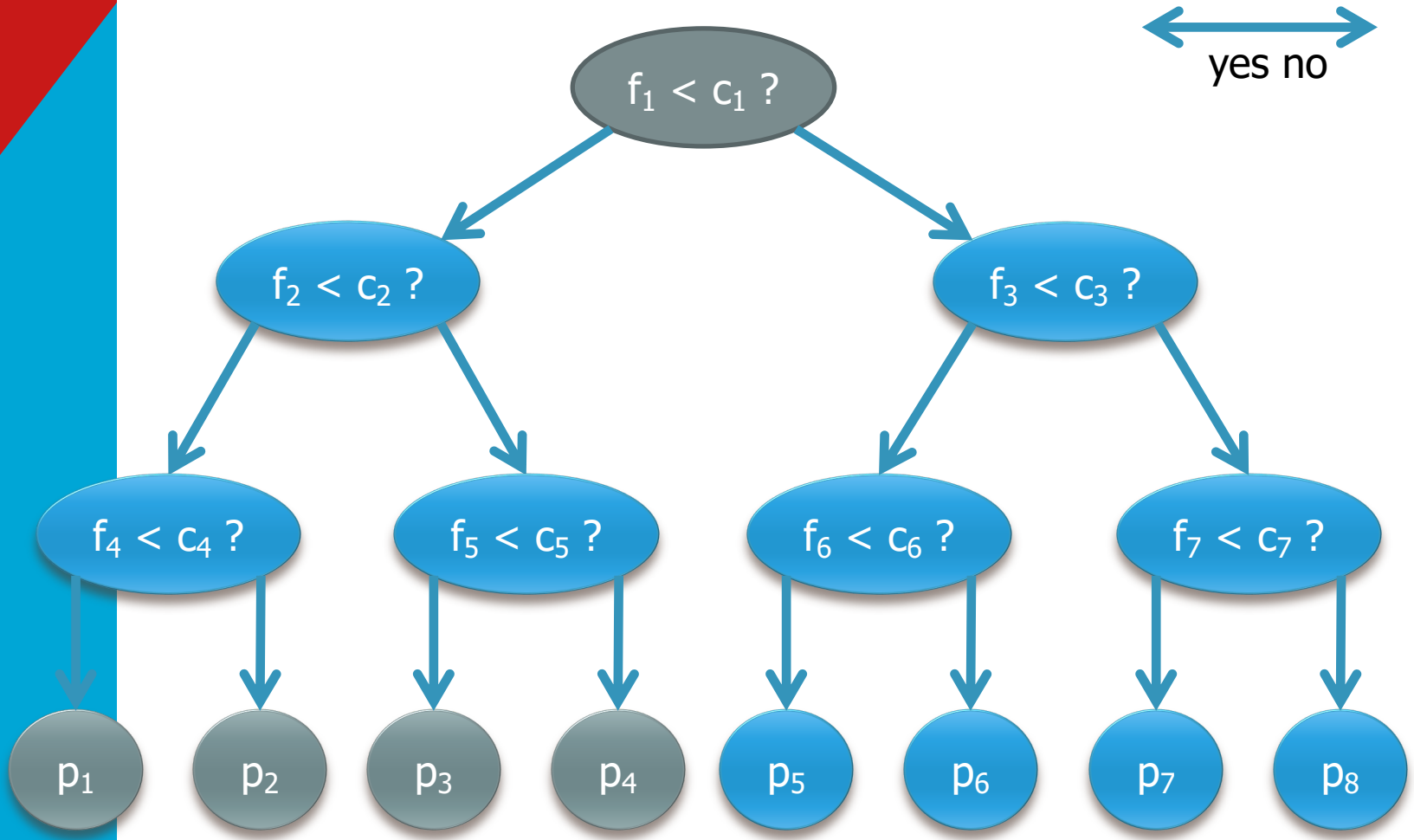
requires *num_rows * num_leafs * depth* constraints

BinOCT approach:



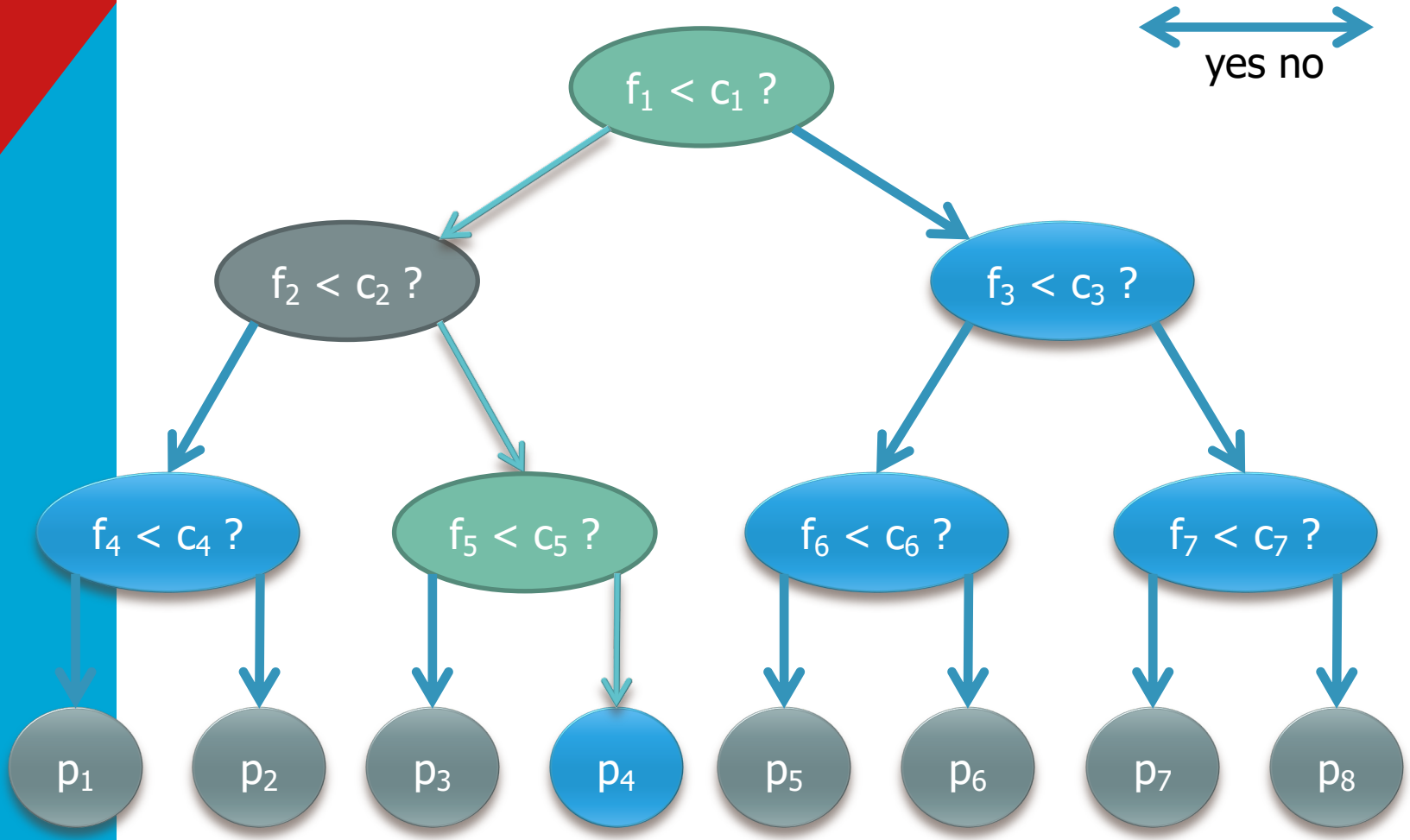
if a node is true, then all of its right children are false

BinOCT approach:



if a node is false, then all of its left children are false

BinOCT approach:



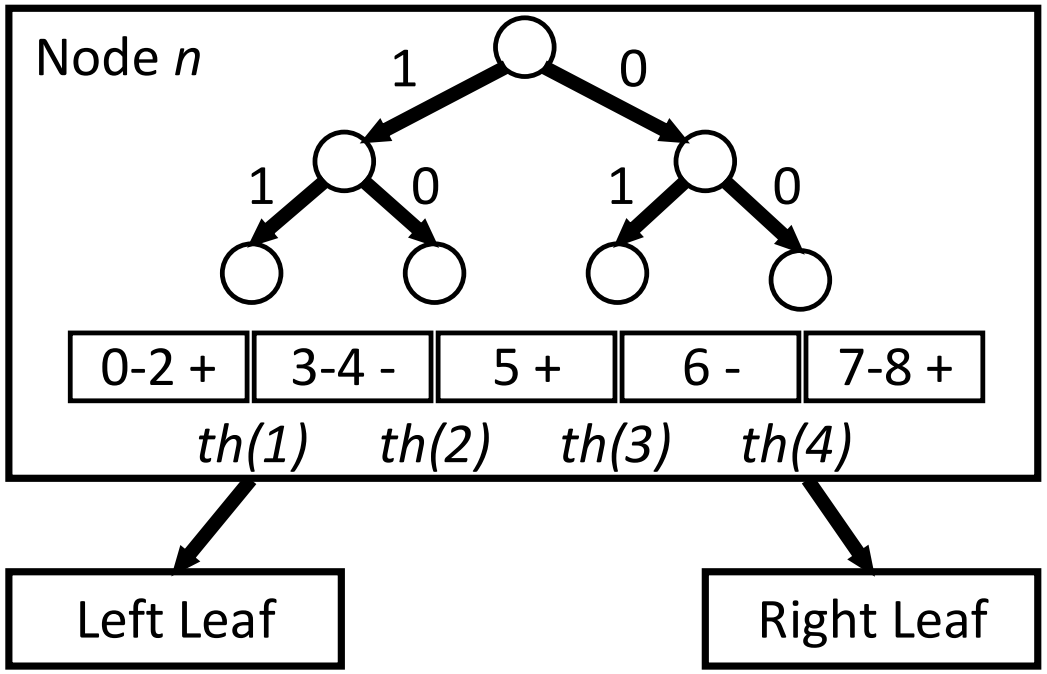
exactly one leaf can be true, encoding the path

BinOCT - decision nodes

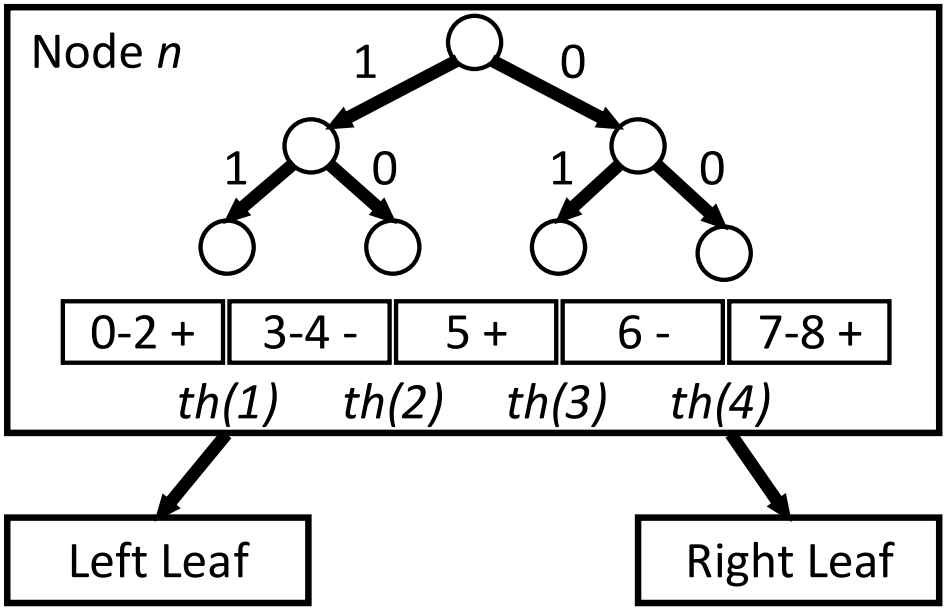
4 possible thresholds

value	class
0	+
1	+
2	+
3	-
4	-
5	+
6	-
7	+
8	+

In every node, a threshold is chosen using binary values



Decision nodes



$t_{n,1}$	$t_{n,2}$	$l_{0,1}$	$l_{1,1}$	$l_{2,1}$	$l_{3,1}$	$l_{4,1}$	$l_{5,1}$	$l_{6,1}$	$l_{7,1}$	$l_{8,1}$
1	**1**	*1*	*1*	*1*	*0*	*0*	*0*	*0*	*0*	*0*
1	**0**	*1*	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*
0	**1**	*1*	*1*	*1*	*1*	*1*	*1*	*0*	*0*	*0*
0	**0**	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*0*	*0*

$t_{n,i}$ = threshold value for node n

$l_{j,1}$ = row j reaches left leaf

Encoding size

Method	# decision variables	# constraints
BinOCT	$O(2^K(F + C + log(T_{max})))$	$O(R + 2^K(F \cdot T_{all} + C))$
DTIP	$O(R \cdot K)$	$O(R \cdot 2^{K-1})$
OCT	$O(R \cdot 2^K)$	$O(2^{K-1}(R \cdot K + C))$

- K = tree depth
- F = number of features
- C = number of classes
- T_{max} = maximum number of decision thresholds
- T_{all} = total number of possible decision thresholds

Column generation for optimal classification trees

- See:
- Firat, M., Crognier, G, Gabor, A., Hurkens C, and Zhang, Y. Computers & Operations Research, 2020

TU/e

ROCT

- Daniël Vos, Sicco Verwer
- AAAI, 2022

TU Delft

Find the difference

Find the difference

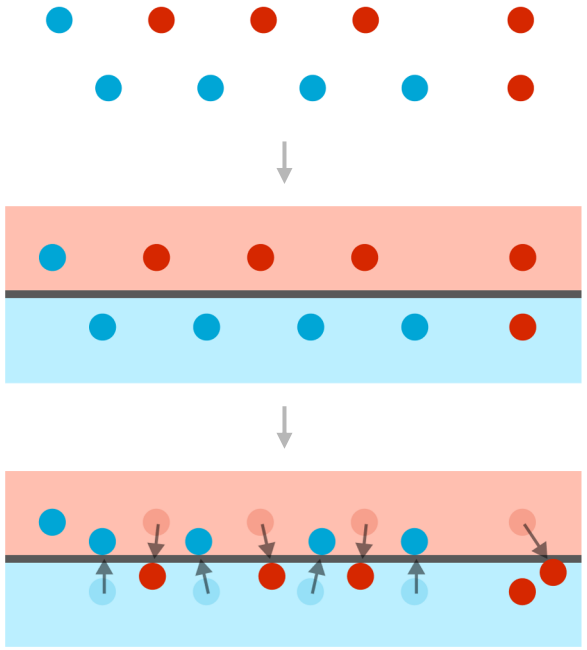
classified as

Stop Sign

classified as

Max Speed 100

Decision trees suffer from adversarial examples

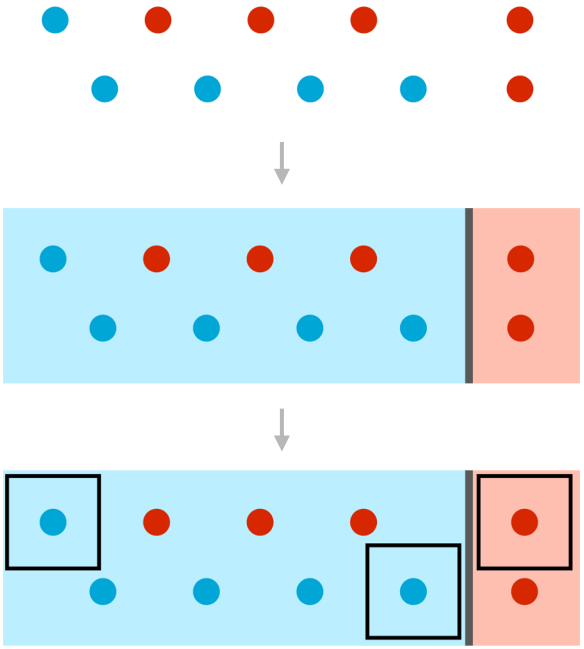


80% accurate

0% accurate against attacks

Inspired by example of Chen et al. (2019) 'Robust Decision Trees Against Adversarial Examples'

We aim to fit trees using robust splits



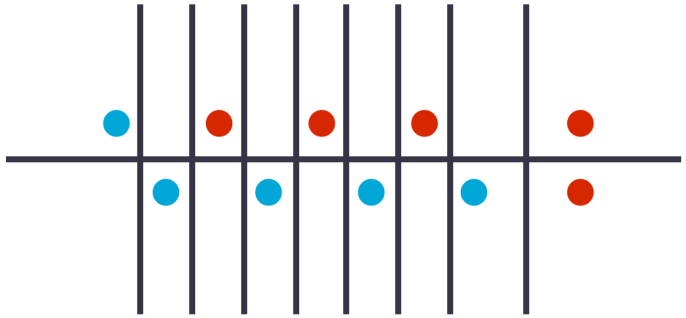
70% accurate

70% accurate against attacks

Inspired by example of Chen et al. (2019) 'Robust Decision Trees Against Adversarial Examples'

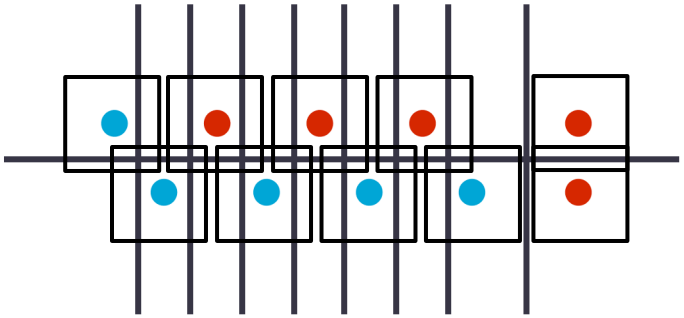
TU/e

Regular trees: Training

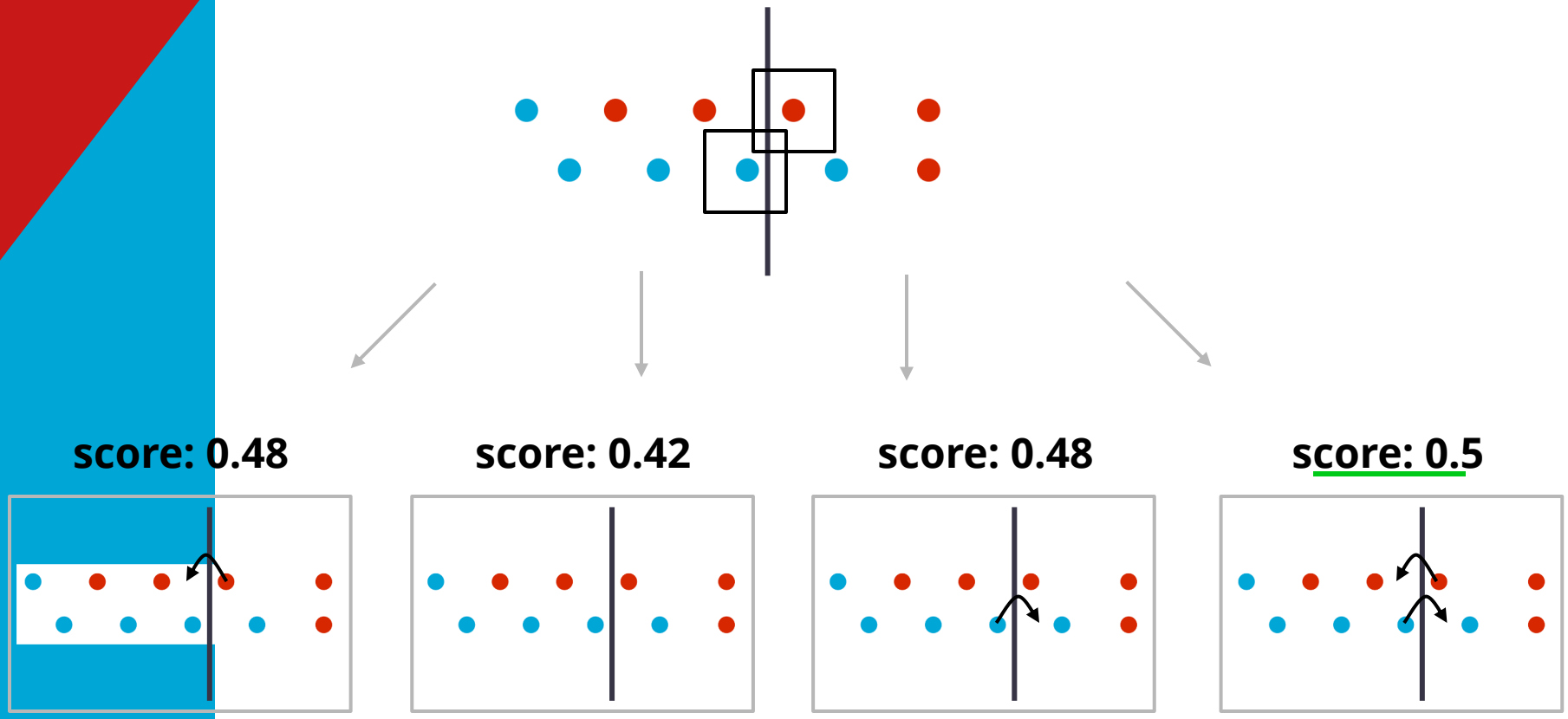


TU Delft

Robust trees: Training

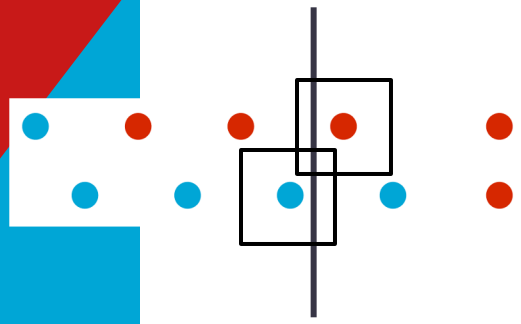


Robust trees: Scoring splits (maximum Gini impurity)



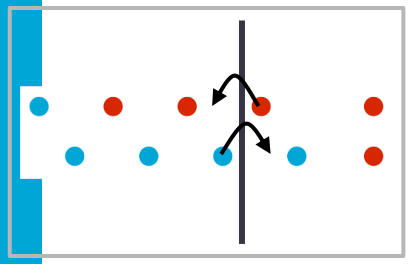
Chen et al. (2019) 'Robust Decision Trees Against Adversarial Examples'

Robust trees: Fast split scoring with GROOT

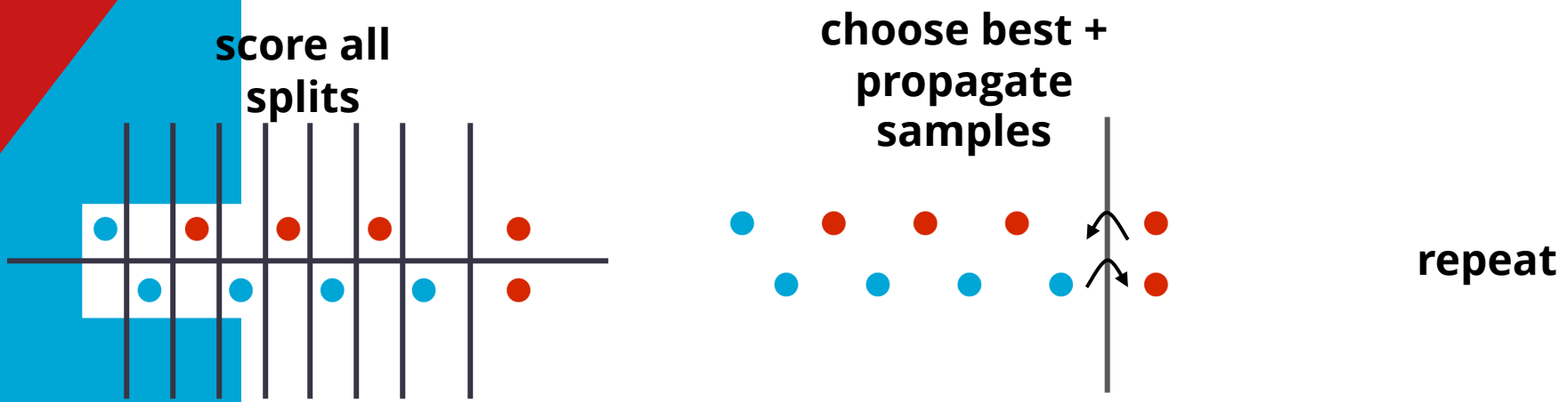


$$m'_0 = \frac{l_1(r_0 + i_0) - l_0(r_1 + i_1)}{l_1 + r_1 + i_1} + \frac{(l_0 + r_0 + i_0)m'_1}{l_1 + r_1 + i_1}$$

score: 0.5

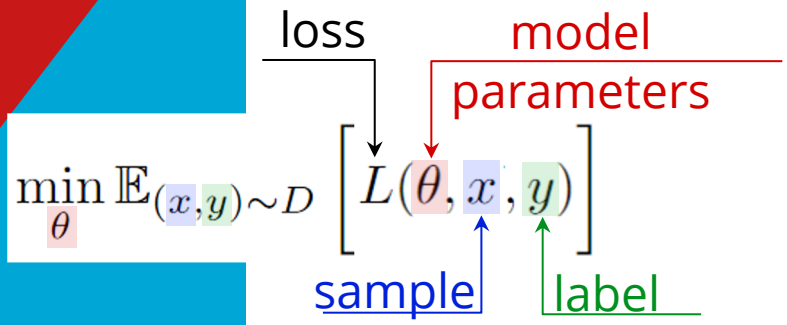


Robust trees: GROOT and Chen et al. are greedy

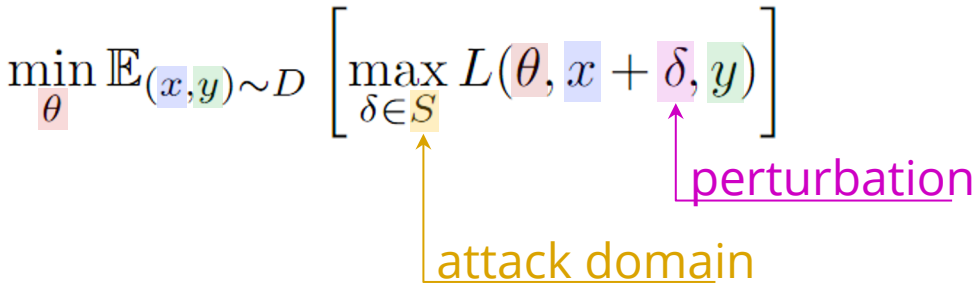


‘Regular’ vs robust learning

Regular learning

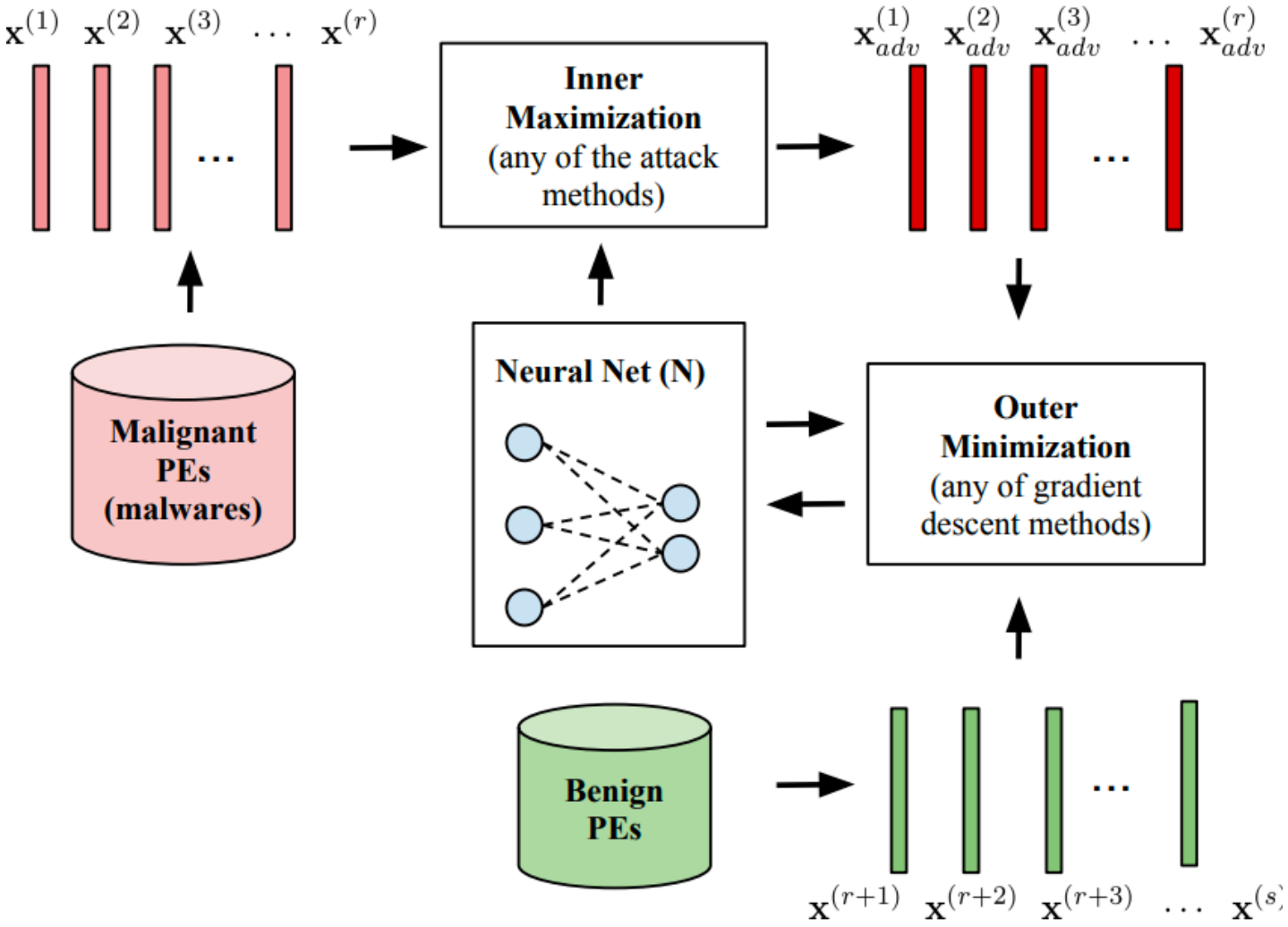


Robust learning



From Madry et al. (2017) ‘Towards deep learning models resistant to adversarial attacks’

Robust learning via hardening



In the case of decision trees

Robust learning

$$\min_{\theta} \mathbb{E}_{(x,y)\sim D} \left[\max_{\delta \in S} L(\theta, x+\delta, y) \right]$$

perturbation

attack domain

Robust tree learning

prediction of leaf

$$\min_{\theta} \sum_{(x,y)\sim D} \left[\bigvee_{t \in \mathcal{T}_L^{S(x)}} c_t \neq y \right]$$

t

leaves reachable by attack

In the case of decision trees

Robust learning

$$\min_{\theta} \mathbb{E}_{(x,y)\sim D} \left[\max_{\delta \in S} L(\theta, x+\delta, y) \right]$$

perturbation

attack domain

Robust tree learning

prediction of leaf

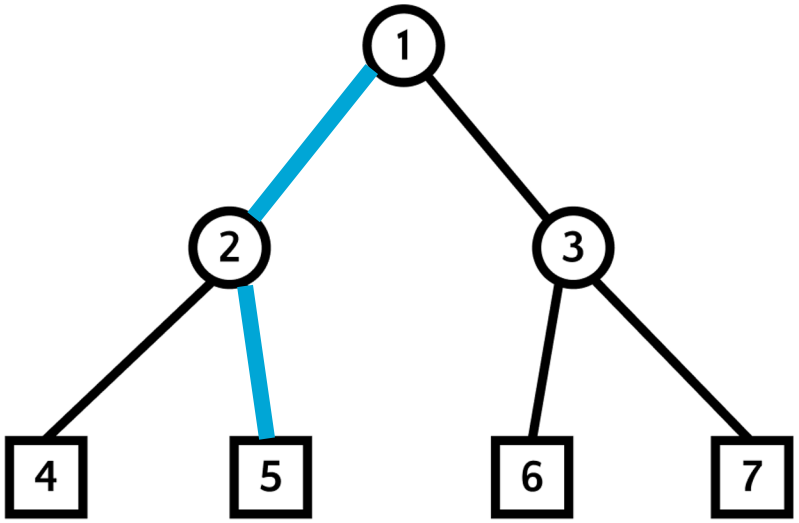
$$\min_{\theta} \sum_{(x,y)\sim D} \left[\bigvee_{t \in \mathcal{T}_L^{S(x)}} c_t \neq y \right]$$

t

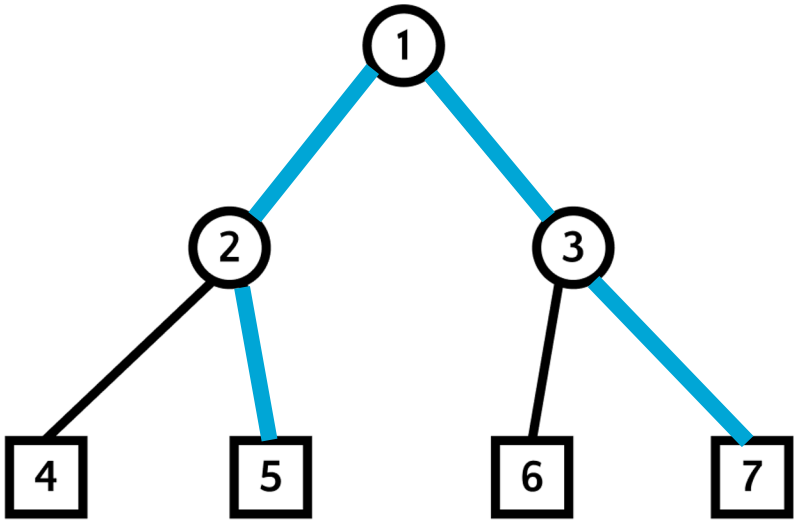
leaves reachable by attack

A single solver call for box-shaped attacks!

What makes this different from other optimal trees?



'normal' classification trees



robust classification trees

Resulting method: ROCT

$$\min. \sum_{i=1}^{n} e_i$$

subject to:

$$\sum_{j=1}^{p} a_{jm} = 1, \qquad \forall m \in \mathcal{T}_B$$

$$b_{vm} \Rightarrow b_{(v+1)m}, \qquad \forall m \in \mathcal{T}_B, v{=}1..|V_j| - 1$$

$$\bigwedge_{m \in A_l(t)} s_{im0} \bigwedge_{m \in A_r(t)} s_{im1} \wedge [c_t \neq y_i] \Rightarrow e_i, \qquad \forall t \in \mathcal{T}_L, i{=}1..n$$

continuous threshold variables:

$$\mathbf{X_i} \cdot \mathbf{a_m} \leq b'_m \Rightarrow s_{im0} \qquad \forall m \in \mathcal{T}_B, i{=}1..n$$

$$\mathbf{X_i} \cdot \mathbf{a_m} > b'_m \Rightarrow s_{im1} \qquad \forall m \in \mathcal{T}_B, i{=}1..n$$

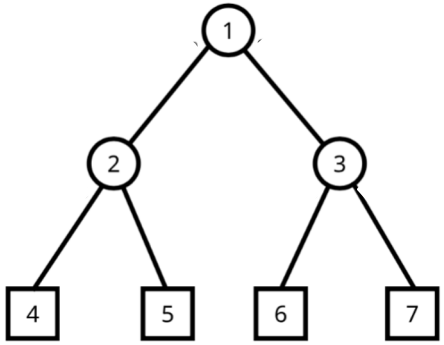
binary threshold variables:

$$a_{jm} \wedge \neg b_{v^l m} \Rightarrow s_{im0}, \qquad \forall m \in \mathcal{T}_B, i{=}1..n, j{=}1..p$$

$$a_{jm} \wedge b_{v^r m} \Rightarrow s_{im1}, \qquad \forall m \in \mathcal{T}_B, i{=}1..n, j{=}1..p$$

GUROBI OPTIMIZATION

SAT



After 30 minutes, ROCT can improve robustness

Algorithm	Mean adv. accuracy	Mean rank	Wins
TREANT	.692 ± .013	5.167 ± .604	7
Binary-MILP	.714 ± .013	3.958 ± .576	10
MILP	.720 ± .015	2.917 ± .454	12
RC2-MaxSAT	.724 ± .014	2.667 ± .393	10
GROOT	.726 ± .015	2.375 ± .450	16
Binary-MILP-warm	.726 ± .015	2.083 ± .399	16
LSU-MaxSAT	.729 ± .014	2.125 ± .303	13
MILP-warm	**.735** ± .015	**1.583** ± .225	**17**

ROCT: summary

- Turn robust learning into optimization
- ROCT beats GROOT but it is close
  - Can prove optimality
- Limitations:
  - Only ran on small datasets
  - Box-shaped attackers
  - Binary classification, but works with one-vs-all

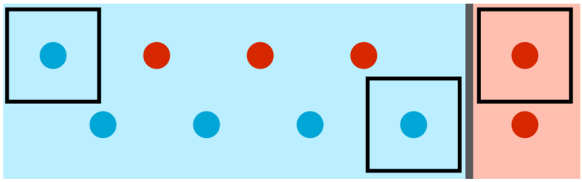
Gurobi

- Open your notebook!
- Three key API functions:
  - **model.addConstr(…)**
  - **gp.quicksum(… for i in …)**
  - **model.addVars(…)**

Attacking:

- We adapt our code to attack an existing model using MIP
  - (although not NP-hard, attacking random forests is…)

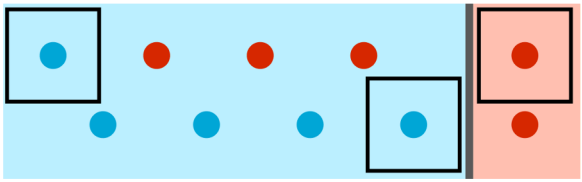
- We can modify data points within a predefined box (radius):



Attacking:

- We adapt our code to attack an existing model using MIP
  - (although not NP-hard, attacking random forests is…)

- We can modify data points within a predefined box (radius):

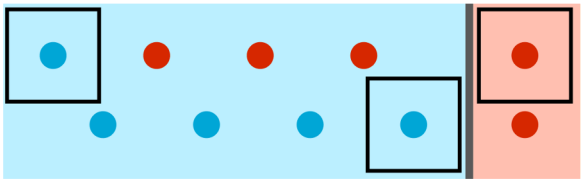


```
for j in range(n_features):
  model.addConstr(x[i, j] <= X[i, j] + perturbation_radius)
  model.addConstr(x[i, j] >= X[i, j] - perturbation_radius)
```

Attacking:

- We adapt our code to attack an existing model using MIP
  - (although not NP-hard, attacking random forests is…)

- We can modify data points within a predefined box (radius):



```
for j in range(n_features):
  model.addConstr(x[i, j] <= X[i, j] + perturbation_radius)
  model.addConstr(x[i, j] >= X[i, j] - perturbation_radius)
```

This is a free variable!

Attacking: leafs and nodes

- We require every row to end in one leaf z

```
for i in range(n_samples):
  model.addConstr(gp.quicksum(z[i, t] for t in leaves) == 1)
```

Attacking: leafs and nodes

- We require every row to end in one leaf z

```
for i in range(n_samples):
  model.addConstr(gp.quicksum(z[i, t] for t in leaves) == 1)
```

- For each node, we compute the tested value f

```
for i in range(n_samples):
  for m in nodes:
    model.addConstr(f[i, m] == gp.quicksum(a[j, m] * x[i, j] for j in
          range(n_features)))
```

Attacking: leafs and nodes

- We require every row to end in one leaf z

```
for i in range(n_samples):
  model.addConstr(gp.quicksum(z[i, t] for t in leaves) == 1)
```

- For each node, we compute the tested value f

```
for i in range(n_samples):
  for m in nodes:
    model.addConstr(f[i, m] == gp.quicksum(a[j, m] * x[i, j] for j in
          range(n_features)))
```

This is a free variable!

This is a constant!

Attacking: paths

- Before:

```
for i in range(n_samples):
  for t in leaves:
    A_l, A_r = ancestors(t)
    for m in A_r:
      model.addConstr(f[i, m] >= b[m] - (1 - z[i, t]))
    for m in A_l:
      model.addConstr(f[i, m] + epsilon <= b[m] + (1 + epsilon) * (1 - z[i, t]))
```

- Now:

```
for i in range(n_samples):
  for t in leaves:
    for m in A_r:
      model.addConstr(f[i, m] - epsilon >= b[m] - (1 + epsilon) * (1 - z[i, t]))
    for m in A_l:
      model.addConstr(f[i, m] + epsilon <= b[m] + (1 + epsilon) * (1 - z[i, t]))
```

Attacking: paths

- Before:

```
for i in range(n_samples):
  for t in leaves:
    A_l, A_r = ancestors(t)
    for m in A_r:
      model.addConstr(f[i, m] >= b[m] - (1 - z[i, t]))
    for m in A_l:
      model.addConstr(f[i, m] + epsilon <= b[m] + (1 + epsilon) * (1 - z[i, t]))
```

- Now:

Why?

```
for i in range(n_samples):
  for t in leaves:
    for m in A_r:
      model.addConstr(f[i, m] - epsilon >= b[m] - (1 + epsilon) * (1 - z[i, t]))
    for m in A_l:
      model.addConstr(f[i, m] + epsilon <= b[m] + (1 + epsilon) * (1 - z[i, t]))
```

Attacking: paths

- Before:

```
for i in range(n_samples):
  for t in leaves:
    A_l, A_r = ancestors(t)
    for m in A_r:
      model.addConstr(f[i, m] >= b[m] - (1 - z[i, t]))
    for m in A_l:
      mo                                              - z[i, t]))
```

- Now

```
for i in
  for t
    for m in A_r:
      model.addConstr(f[i, m] - epsilon >= b[m] - (1 + epsilon) * (1 - z[i, t]))
    for m in A_l:
      model.addConstr(f[i, m] + epsilon <= b[m] + (1 + epsilon) * (1 - z[i, t]))
```

Otherwise, optimizer finds examples exactly on the threshold, which can be incorrect due to inaccuracies

Attacking: objective

- Maximize errors!
- We add variable $w_{i,t}$: true if row i makes an error in leaf t

```
model.addConstr(e[i] <= gp.quicksum(w[i, t] for t in leaves))
```

- Constraints for w:

Attacking: objective

- Maximize errors!
- We add variable $w_{i,t}$: true if row i makes an error in leaf t

```
model.addConstr(e[i] <= gp.quicksum(w[i, t] for t in leaves))
```

- Constraints for w:

```
for i in range(n_samples):
  for t in leaves:
    model.addConstr(w[i, t] <= z[i, t])
    model.addConstr(w[i, t] <= 1 - c[t, y[i]])
```

Code for GUROBI

```
for i in range(n_samples):
  for j in range(n_features):
    model.addConstr(x[i, j] <= X[i, j] + perturbation_radius)
    model.addConstr(x[i, j] >= X[i, j] - perturbation_radius)

for i in range(n_samples):
  model.addConstr(gp.quicksum(z[i, t] for t in leaves) == 1)
  for m in nodes:
    model.addConstr(f[i, m] == gp.quicksum(a[j, m] * x[i, j] for j in range(n_features)))
    for t in leaves:
      A_l, A_r = ancestors(t)
      for m in A_r:
        model.addConstr(f[i, m] - epsilon >= b[m] - (1 + epsilon) * (1 - z[i, t]))
      for m in A_l:
        model.addConstr(f[i, m] + epsilon <= b[m] + (1 + epsilon) * (1 - z[i, t]))

for i in range(n_samples):
  for t in leaves:
    model.addConstr(w[i, t] <= z[i, t])
    model.addConstr(w[i, t] <= 1 - c[t, y[i]])
  model.addConstr(e[i] <= gp.quicksum(w[i, t] for t in leaves))
```

Lab

- Test also groot/ROCT:
  - pip install groot-tree
  - pip install roct
- Run the cells below to learn a more robust model.
- Play with the radius, is the result what you expected?
- The attack also works against oblique splits!
- You can add any of the earlier constaints!
- But ROCT would need some modification…

Main take-aways

- ML problems are hard
  - *Can (and should?) be solved by solvers for such problems?*
  - *Is it really hard in practice?*
- Modeling by encoding:
  - The model
  - How the model processes data
  - The objective function
  - *Any additional constraints you desire*
- Many extentions/speedups in the past few years, can find optimal decision trees in minutes, even **robust ones**
  - ***Please do not simply use CART!***

Side-note: Discrimination

Side-note: Discrimination

Side-effect of inventory management

Black Barbie sells less often than White Barbie

TU/e

Solutions?

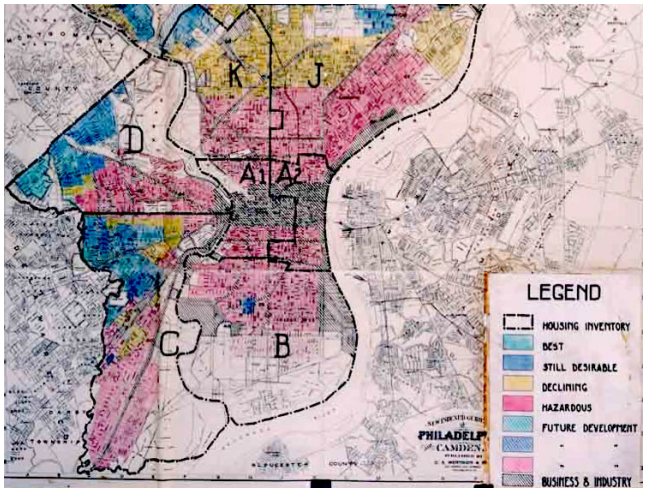
- …

TU Delft

So, Let's Omit Sensitive Attributes …

- *Just removing the sensitive attributes does not help!*

- Other attributes may be highly correlated with the sensitive attribute:
  - gender ←→ profession
  - race ←→ postal code
  - …



The HOLC maps are part of the records of the FHLBB (RG195) at the National Archives II

- **We call this the *redlining effect***

Solution

- Basic idea:
  - Instead of only maximizing accuracy/likelihood
  - Learn a model that maximizes accuracy, while minimizing discrimination
- First: what is discrimination?

Measuring discrimination

- Berkeley discrimination case:

	Applicants	Admitted
Men	8442	**44%**
Women	4321	35%

- The probability of acceptance:
  - Pr(admitted | men) = 0.44
  - Pr(admitted | women) = 0.35

- Standard discrimination measure = difference in acceptance probability
  - Pr(admitted | men) - Pr(admitted | women) = 0.09

For general models: preferential sampling

- Randomly draw examples from every sensitive group
- From a discriminated group:
  - Draw more examples with positive labels
  - Draw less examples with negative labels
- From a favored group:
  - Draw less examples with positive labels
  - Draw more examples with negative labels
- Until a data set is created with 0 discrimination
  - ***Maintain the same fraction of positive/negative labels!***
- Learn a model from this data

For general models: different thresholds

- Learn a probabilistic model
- Such a model gives a probability P(i) that example i belongs to the positive class
  - Naïve Bayes is such a classifier
- Use different decision thresholds t for the different sensitive groups, such that
  - If P(i) > t, then i is labeled as positive
  - **The fraction of examples labeled as positive is roughly equal for every sensitive group**

For general models: different models

- Learn a probabilistic model **for every sensitive group s**
- Such a model gives a probability P(i) that example i belongs to the positive class
  - Naïve Bayes is such a classifier
- Use different decision thresholds t for the different groups, such that
  - If $P_s(i) > t$, then i is labeled as positive
  - The fraction of examples labeled as positive is roughly equal for each group

Even better: *optimal/declarative learning*

	f1	f2	f3	f4	t
1	5.1	3.5	1.4	0.2	s
2	4.9	3.0	1.4	0.2	s
3	4.7	3.2	1.3	0.3	v

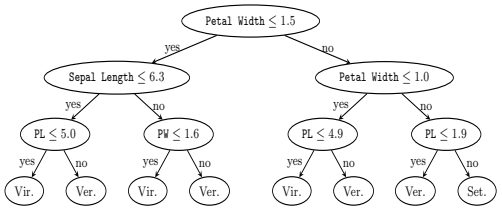
translate

$$l_{5,1} + l_{6,1} + 2 \cdot t_{n,1} \leq 2$$
$$l_{3,2} + l_{4,2} + l_{5,2} - 3 \cdot t_{n,1} \leq 0$$
$$l_{3,1} + l_{4,1} + 2 \cdot t_{n,1} + 2 \cdot t_{n,2} \leq 4$$
$$l_{3,2} + l_{4,2} - 2 \cdot t_{n,2} \leq 0$$
$$l_{6,1} + 1 \cdot t_{n,2} \leq 1$$
$$l_{6,2} - 1 \cdot t_{n,1} - 1 \cdot t_{n,2} \leq 0$$

add objective

$$\min \sum_{l,c} e_{l,c} \quad \text{s.t.}$$
$$\sum_{r:C_r=c} l_{r,l} - M \cdot p_{l,c} \leq e_{l,c}$$

solve



translate

add bias constraints

Verwer and Zhang. "Learning decision trees with flexible constraints and objectives using integer optimization" *CPAIOR, 2017*

Part 2: Optimization by learning

	f1	f2	f3	f4	t
1	5.1	3.5	1.4	0.2	s
2	4.9	3.0	1.4	0.2	s
3	4.7	3.2	1.3	0.3	v

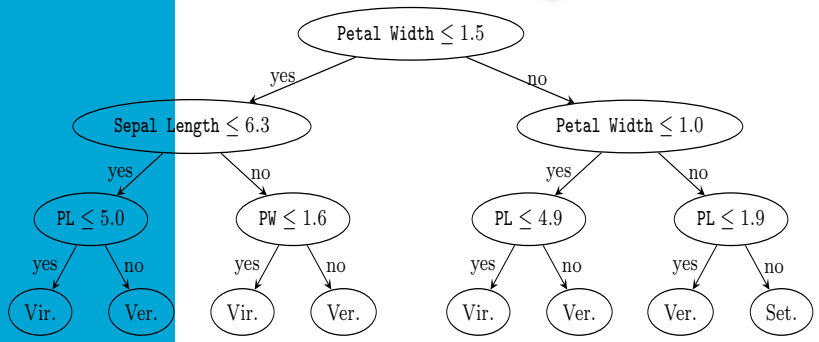
learn

translate

add to problem instance

solve

$$l_{5,1} + l_{6,1} + 2 \cdot t_{n,1} \leq 2$$
$$l_{3,2} + l_{4,2} + l_{5,2} - 3 \cdot t_{n,1} \leq 0$$
$$l_{3,1} + l_{4,1} + 2 \cdot t_{n,1} + 2 \cdot t_{n,2} \leq 4$$
$$l_{3,2} + l_{4,2} - 2 \cdot t_{n,2} \leq 0$$
$$l_{6,1} + 1 \cdot t_{n,2} \leq 1$$
$$l_{6,2} - 1 \cdot t_{n,1} - 1 \cdot t_{n,2} \leq 0$$



$$\min \textstyle\sum_{l,c} e_{l,c} \quad \text{s.t.}$$
$$\forall_n \quad \textstyle\sum_f f_{n,f} = 1$$
$$\forall_r \quad \textstyle\sum_l l_{r,l} = 1$$
$$\forall_l \quad \textstyle\sum_c p_{l,c} = 1$$
$$\forall_{n,f,b \in bin(f)} \quad M \cdot f_{n,f} + \textstyle\sum_{r \in lr(b)} \sum_{l \in ll(n)} l_{r,l} + \sum_{t \in tl(b)} M \cdot t_{n,t} - \sum_{t \in tl(b)} M \leq M$$
$$\forall_{n,f,b \in bin(f)} \quad M' \cdot f_{n,f} + \textstyle\sum_{r \in rr(b)} \sum_{l \in rl(n)} l_{r,l} - \sum_{t \in tl(b)} M' \cdot t_{n,t} \leq M'$$
$$\forall_{n,f} \quad M'' \cdot f_{n,f} + \textstyle\sum_{\max_{t(f)} < f(r)} \sum_{l \in ll(n)} l_{r,l} + \sum_{f(r) < \min_{t(f)}} \sum_{l \in rl(n)} l_{r,l} \leq M''$$
$$\forall_{l,c} \quad \textstyle\sum_{r:C_r=c} l_{r,l} - M''' \cdot p_l \leq e_{l,c}$$

TU/e

White-box optimization

- Sicco Verwer, Yingqian Zhang, Qing Chuan Ye
- Artificial Intelligence journal, 2017

TUDelft

Mathematical models

- Example: Vehicle Routing problem with Time windows (VRPTW)

Domain knowledge

$$\text{(VRPTW)} \quad \min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ijk}$$

subject to

$$\sum_{k \in K} \sum_{j \in \Delta^+(i)} x_{ijk} = 1, \qquad \forall i \in N$$

$$\sum_{j \in \Delta^+(0)} x_{0jk} = 1, \qquad \forall k \in K$$

$$\sum_{i \in \Delta^-(j)} x_{ijk} - \sum_{i \in \Delta^+(j)} x_{jik} = 0, \qquad \forall k \in K, \forall j \in N$$

$$\sum_{i \in \Delta^-(n+1)} x_{i,n+1,k} = 1, \qquad \forall k \in K$$

$$x_{ijk}(w_{ik} + s_i + t_{ij} - w_{jk}) \leq 0, \qquad \forall k \in K, \forall (i,j) \in A$$

$$a_i(\sum_{j \in \Delta^+(i)} x_{ijk}) \leq w_{ik} \leq b_i(\sum_{j \in \Delta^+(i)} x_{ijk}), \qquad \forall k \in K, \forall i \in N$$

$$E \leq w_{ik} \leq L, \qquad \forall k \in K, \forall i \in \{0, n+1\}$$

$$\sum_{i \in N} d_i \sum_{j \in \Delta^+(i)} x_{ijk} \leq C, \qquad \forall k \in K$$

$$x_{ijk} \geq 0, \qquad \forall k \in K, \forall (i,j) \in A$$

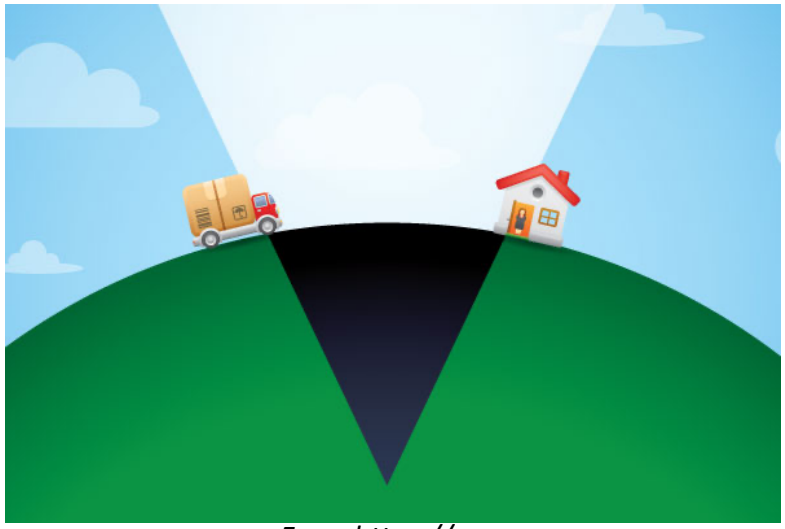
$$x_{ijk} \text{ binary}, \qquad \forall k \in K, \forall (i,j) \in A.$$

from Cordeau et al.

optimization solver

Best route: hub->A->C->B ->… -> hub

Real-world VRPTW

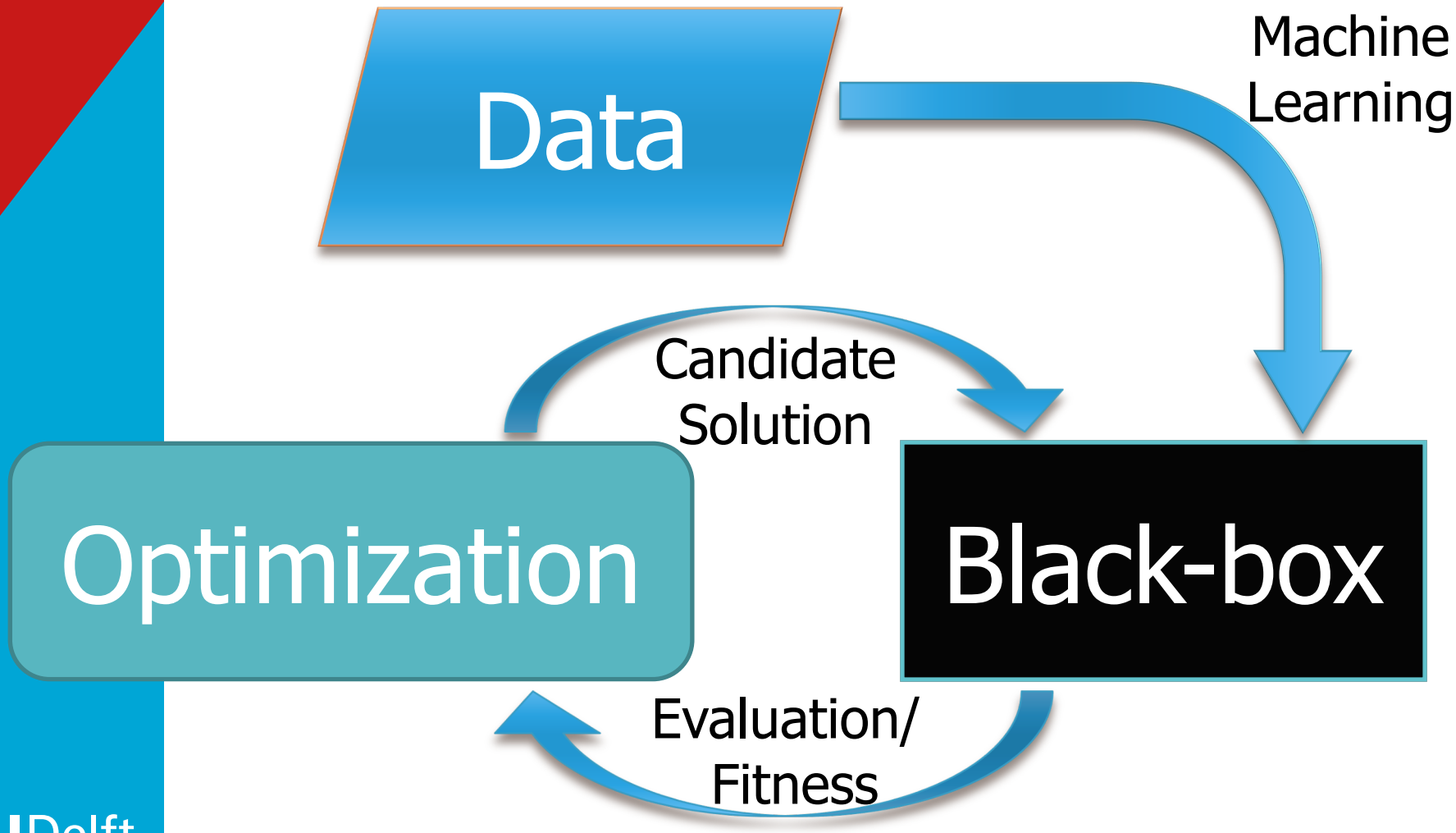


From https://ecommercegermany.com

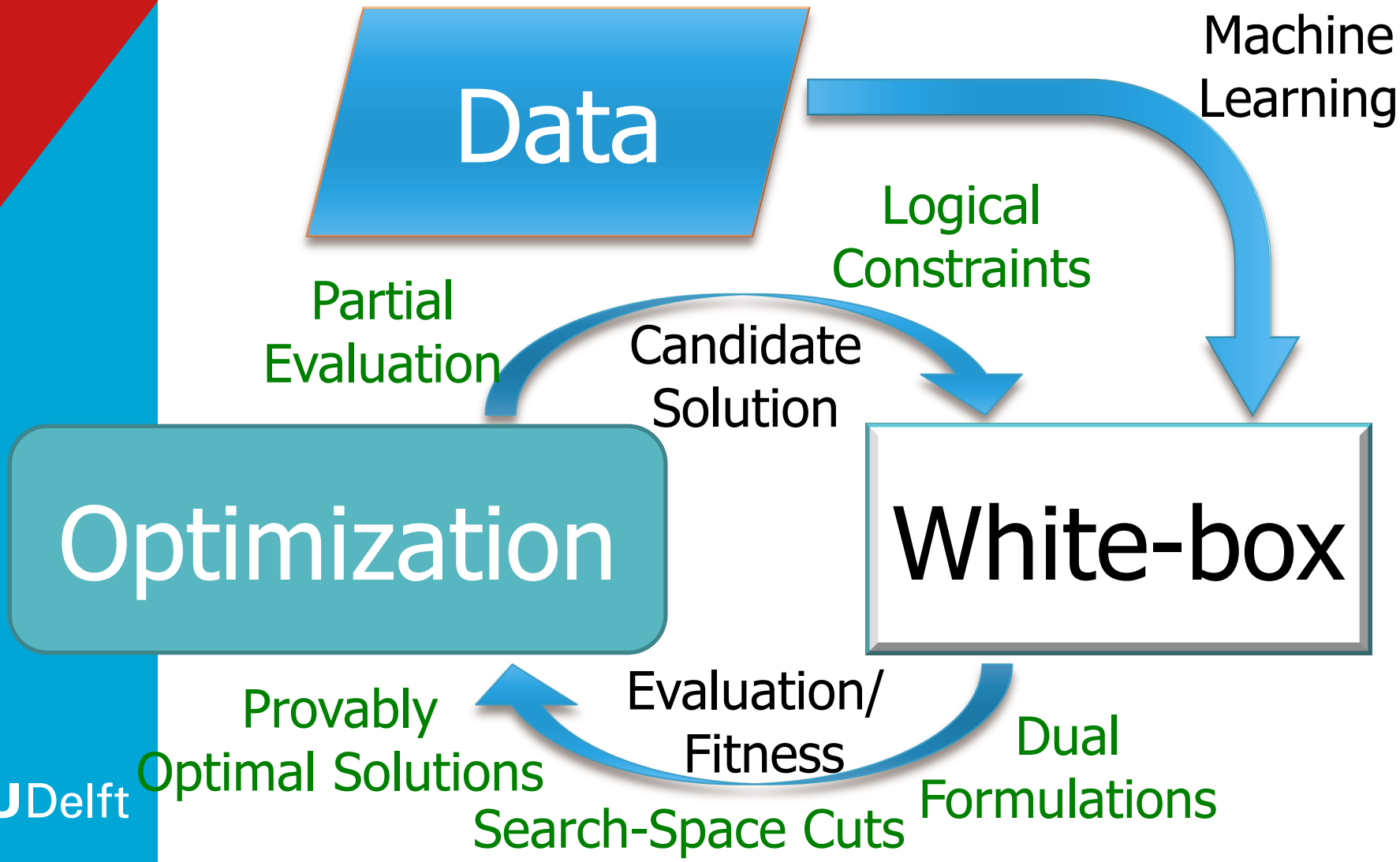
- Hard to handle uncertainties and dynamics
- Hard to model
  - different situations (weekend, weekday…)
  - unknown relations (weather vs travel time…)

What could possibly go wrong?!

Traditional optimization from data



Insight: learned models are simple!



Mathematical modelling

- Traditionally a manual process
  - Interview experts to determine problem constraints
  - Develop model based on gained knowledge
  - Model uncertainties using stochastic variables
- Automatic model construction?
  - Typical task in machine learning
  - How to use learned models in optimization?

Example: Dutch flower auctions

- Sequential auctions
- Approximately 125 000 transactions per day
- Ordering matters!
  different orderings of items to sell influence the selling prices (revenue),
  due to bidder's preferences and their budget limits

- Optimization problem:
  - How to order a set of items, such that the total revenue is optimized

Example

- Two bidders: A, B
- Two types of items: r1, r2, with reserve price 4
  - A: v(r1) = 5, v(r2)=6, budget = 5
  - B: v(r1) = 0, v(r2)=5, budget = 5
  - Reservation prices are v(ri) - 1
- Two possible orderings:
  - <r1, r2>
    - r1 -> A for 4, r2 -> B for 4; with revenue 8
  - <r2, r1>
    - r2 -> A for 5, r1 unsold; with revenue 5

Complex - example - agents

	A1	A2	A3	A4	A5	A6	A7	A8
budget	78	37	80	60	119	103	46	63
v(r1)	34	24	20	41		38	24	
v(r2)	59	30		21		58	25	42
v(r3)			61	30	53			
v(r4)	74		22					

Complex - example - agents

	A1	A2	A3	A4	A5	A6	A7	A8
budget	78	37	80	60	119	103	46	63
v(r1)	34	24	20	41		38	24	
v(r2)	59	30		21		58	25	42
v(r3)			61	30	53			
v(r4)	74		22					

r1 and r2 are popular, r4 is very valuable for agent A1

strategy: try selling r4 first to A1(?)

Complex - example - auctions

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	r2 59	r4 22	r1 41	r2 58	r2 45	r1 24	r2 42	r4 22	r4 22	r2 25	r2 21	r4 20	r2 21	r2 19	r3 53
2	r2 59	r2 58	r2 45	r4 22	r4 22	r4 22	r2 42	r4 20	r2 30	r4 20	r4 20	r2 25	r2 21	r1 39	r4 20
3	r1 41	r1 38	r4 74	r4 22	r2 58	r4 22	r2 42	r2 30	r4 22	r2 25	r2 20	r4 20	r1 21	r2 21	r2 19
4	r3 61	r4 74	r1 41	r3 53	r4 20	r4 20	r4 20	r4 20	r3 53	r4 20	r2 58	r1 38	r2 42	r1 24	r3 19
5	r2 59	r3 61	r2 58	r4 20	r3 53	r3 52	r1 41	r3 19	r2 45	r3 19	r2 42	r4 20	r4 20	r3 17	r2 30

Complex - example - auctions

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	r2 59	r4 22	r1 41	r2 58	r2 45	r1 24	r2 42	r4 22	r4 22	r2 25	r2 21	r4 20	r2 21	r2 19	r3 53
2	r2 59	r2 58	r2 45	r4 22	r4 22	r4 22	r2 42	r4 20	r2 30	r4 20	r4 20	r2 25	r2 21	r1 39	r4 20
3	r1 41	r1 38	r4 74	r4 22	r2 58	r4 22	r2 42	r2 30	r4 22	r2 25	r2 20	r4 20	r1 21	r2 21	r2 19
4	r3 61	r4 74	r1 41	r3 53	r4 20	r4 20	r4 20	r4 20	r3 53	r4 20	r2 58	r1 38	r2 42	r1 24	r3 19
5	r2 59	r3 61	r2 58	r4 20	r3 53	r3 52	r1 41	r3 19	r2 45	r3 19	r2 42	r4 20	r4 20	r3 17	r2 30

r1 is rare

wait a minute… what is that 39 doing there?

Complex - example - auctions

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	r2 59	r2 58	r2 45	r4 22	r4 22	r4 22	r2 42	r4 20	r2 30	r4 20	r4 20	r2 25	r2 21	r1 39	r4 20

	A1	A2	A3	A4	A5	A6	A7	A8
budget	78	37	80	60	119	103	46	63
v(r1)	34	24	20	41		38	24	
v(r2)	59	30		21		58	25	42
v(r3)			61	30	53			
v(r4)	74		22					

Complex - example - auctions

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	r2 59 -> A1	r2 58 -> A6	r2 45 -> A6	r4 22 -> A3	r4 22 -> A3	r4 22 -> A3	r2 42 -> A8	r4 20 -> -	r2 30 -> A2	r4 20 -> -	r4 20 -> -	r2 25 -> A7	r2 21 -> A4	r1 39 -> **A4**	r4 20 -> -

A4 is out of budget

	A1	A2	A3	A4	A5	A6	A7	A8
budget	78	37	80	60	119	103	46	63
v(r1)	34	24	20	41		38	24	
v(r2)	59	30		21		58	25	42
v(r3)			61	30	53			
v(r4)	74		22					

Complex - example - auctions

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	r2 59	r4 22	r1 41	r2 58	r2 45	r1 24	r2 42	r4 22	r4 22	r2 25	r2 21	r4 20	r2 21	r2 19	r3 53
2	r2 59	r2 58	r2 45	r4 22	r4 22	r4 22	r2 42	r4 20	r2 30	r4 20	r4 20	r2 25	r2 21	r1 39	r4 20
3	r1 41	r1 38	r4 74	r4 22	r2 58	r4 22	r2 42	r2 30	r4 22	r2 25	r2 20	r4 20	r1 21	r2 21	r2 19
4	r3 61	r4 74	r1 41	r3 53	r4 20	r4 20	r4 20	r4 20	r3 53	r4 20	r2 58	r1 38	r2 42	r1 24	r3 19
5	r2 59	r3 61	r2 58	r4 20	r3 53	r3 52	r1 41	r3 19	r2 45	r3 19	r2 42	r4 20	r4 20	r3 17	r2 30

how to order r1?

Complex - example - auctions

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	r2 59	r4 22	r1 41	r2 58	r2 45	r1 24	r2 42	r4 22	r4 22	r2 25	r2 21	r4 20	r2 21	r2 19	r3 53
2	r2 59	r2 58	r2 45	r4 22	r4 22	r4 22	r2 42	r4 20	r2 30	r4 20	r4 20	r2 25	r2 21	r1 39	r4 20
3	r1 41	r1 38	r4 74	r4 22	r2 58	r4 22	r2 42	r2 30	r4 22	r2 25	r2 20	r4 20	r1 21	r2 21	r2 19
4	r3 61	r4 74	r1 41	r3 53	r4 20	r4 20	r4 20	r4 20	r3 53	r4 20	r2 58	r1 38	r2 42	r1 24	r3 19
5	r2 59	r3 61	r2 58	r4 20	r3 53	r3 52	r1 41	r3 19	r2 45	r3 19	r2 42	r4 20	r4 20	r3 17	r2 30

how to order r1?
index does not matter, but aim for at most one r2 between the first and second r1

Complex - example - auctions

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	r2 59	r4 22	r1 41	r2 58	r2 45	r1 24	r2 42	r4 22	r4 22	r2 25	r2 21	r4 20	r2 21	r2 19	r3 53
2	r2 59	r2 58	r2 45	r4 22	r4 22	r4 22	r2 42	r4 20	r2 30	r4 20	r4 20	r2 25	r2 21	r1 39	r4 20
3	r1 41	r1 38	r4 74	r4 22	r2 58	r4 22	r2 42	r2 30	r4 22	r2 25	r2 20	r4 20	r1 21	r2 21	r2 19
4	r3 61	r4 74	r1 41	r3 53	r4 20	r4 20	r4 20	r4 20	r3 53	r4 20	r2 58	r1 38	r2 42	r1 24	r3 19
5	r2 59	r3 61	r2 58	r4 20	r3 53	r3 52	r1 41	r3 19	r2 45	r3 19	r2 42	r4 20	r4 20	r3 17	r2 30

r4 is common

how to order it?

Complex - example - auctions

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	r2 59	r4 22	r1 41	r2 58	r2 45	r1 24	r2 42	r4 22	r4 22	r2 25	r2 21	r4 20	r2 21	r2 19	r3 53
2	r2 59	r2 58	r2 45	r4 22	r4 22	r4 22	r2 42	r4 20	r2 30	r4 20	r4 20	r2 25	r2 21	r1 39	r4 20
3	r1 41	r1 38	r4 74	r4 22	r2 58	r4 22	r2 42	r2 30	r4 22	r2 25	r2 20	r4 20	r1 21	r2 21	r2 19
4	r3 61	r4 74	r1 41	r3 53	r4 20	r4 20	r4 20	r4 20	r3 53	r4 20	r2 58	r1 38	r2 42	r1 24	r3 19
5	r2 59	r3 61	r2 58	r4 20	r3 53	r3 52	r1 41	r3 19	r2 45	r3 19	r2 42	r4 20	r4 20	r3 17	r2 30

order r4 before any r2

TU/e

The problem

Theorem 1. *Given a set of items R, preferences $v_i : R \to \mathbb{R}^+$, and budgets b_i for every bidder i. The problem of deciding whether there exists an ordering that obtains a revenue of at least $K \in \mathbb{R}^+$ is NP-hard.*

TUDelft

The problem

Theorem 1. *and budgets b_i for every bidder dering that obtains a revenue of*

This assumes every agent's preferences and budget limits are known!

Usually, they are not…

The real problem

- We aim to maximize this objective function

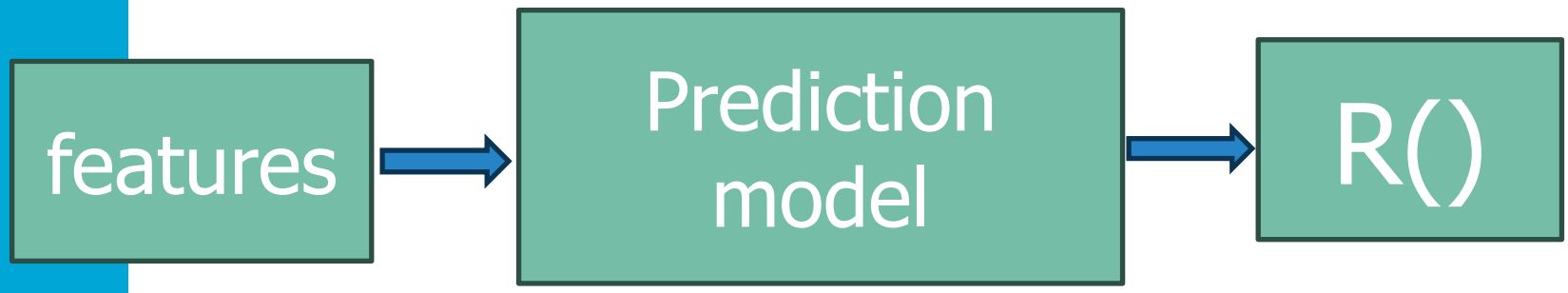
$$V(r_1 \dots r_n) = \sum_{1 \le k \le n} R(r_k, \{r_j \mid j < k\}, \{r_l \mid k < l\})$$

- where R(r,{a,b},{c,d}) is the expected reward for item r given that a,b are auctioned before, and c,d are auctioned after

- We do not know how to compute R()!

Machine learning

- Every day auction generates a lot of data:
  - The ordering of items to be sold
  - The reserve price of each item
  - The price of each sold item
- No bidders' information; but their preferences/behaviours are implicit in the data (influence the selling price with different ordering)

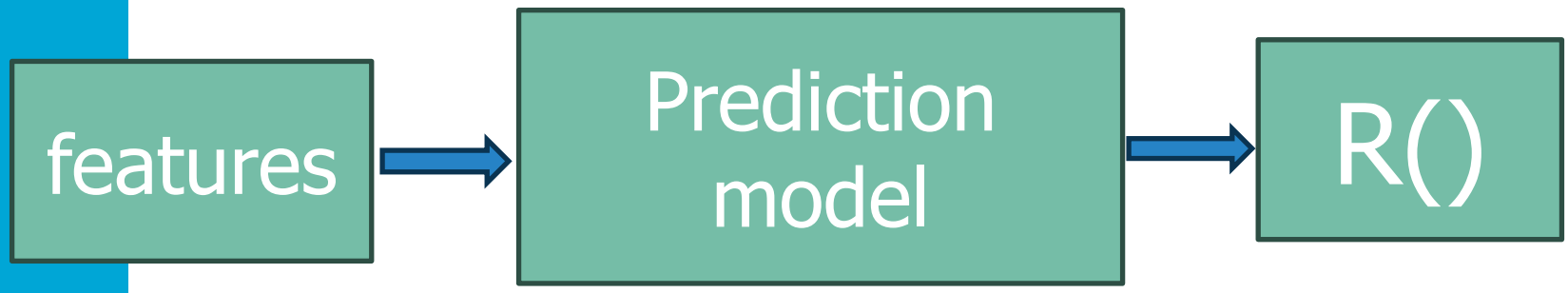


The real problem

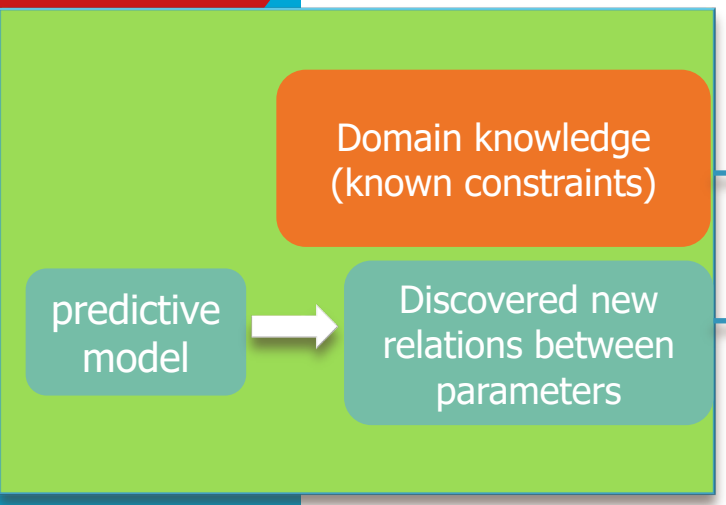
- We aim to maximize

$$V(r_1 \dots r_n) = \sum_{1 \leq k \leq n} R(r_k, \{r_j \mid j < k\}, \{r_l \mid k < l\})$$

- where
  - ***R(r,{a,b},{c,d}) is given by a model learned from data***



White box optimization framework



$$\max \sum_{1 \le i \le n} \sum_{r \in R} p_{i,r}$$

$$\begin{aligned} \sum_{r \in R} x_{i,r} &= 1 && \text{for all } 1 \le i \le n \\ \sum_{1 \le i \le n} x_{i,r} &= n_r && \text{for all } r \in R \end{aligned}$$

$$p_{i,r} = \sum_{l \in L_r} c_{l,r} \cdot z_{i,l,r}, \quad \text{for all } 1 \le i \le n, r \in R$$

$$\begin{aligned} \mathtt{fv}_{f,i} + (M_{f,i} - c) \cdot \sum_{l \in L} z_{i,l,r} &\le M_{f,i} && \text{for all } 1 \le i \le n, r \in R, (f \le c) \in D_r \\ \mathtt{fv}_{f.i} + (m_{f.i} - c) \cdot \sum_{l \in L} z_{i.l.r} &\ge m_{f.i} && \text{for all } 1 \le i \le n, r \in R, (f \le c) \in D_r \\ \sum_l z_{i,l,r} &= x_{i,r} && \text{for all } 1 \le i \le n, r \in R \end{aligned}$$

$$\begin{aligned} \mathtt{sold}_{i,r} &= \sum_{j<i} x_{j,r} && \text{for all } 1 \le i \le n, r \in R \\ \mathtt{diff}_{i,r,r'} &= \mathtt{sold}_{i,r} - \mathtt{sold}_{i,r'} && \text{for all } 1 \le i \le n, r, r' \in R, r \ne r' \\ \mathtt{remain}_{i,r} &= \sum_{j>i} x_{j,r} && \text{for all } 1 \le i \le n, r \in R \\ \mathtt{index}_i &= i && \text{for all } 1 \le i \le n \end{aligned}$$

Building prediction models

- Calculate features from data
  - Feature 1: The items already auctioned
  - Feature 2: The items still to be auctioned
  - Feature 3: Paired differences of feature 1
  - Feature 4: Revenues obtained per item type and total
  - Feature 5: The position in the auction
- Suppose we have two orderings:
  - (r1, r2) and (r2, r1), with revenues (4,4) and (5,0)

Data set

type	revenue	sold r_1	sold r_2	$diff_{1,2}$	sum	sum r_1	sum r_2	index
r_1	4	0	0	0	0	0	0	1
r_2	4	1	0	1	4	4	0	2
r_2	5	0	0	0	0	0	0	1
r_1	0	0	1	-1	5	0	5	2

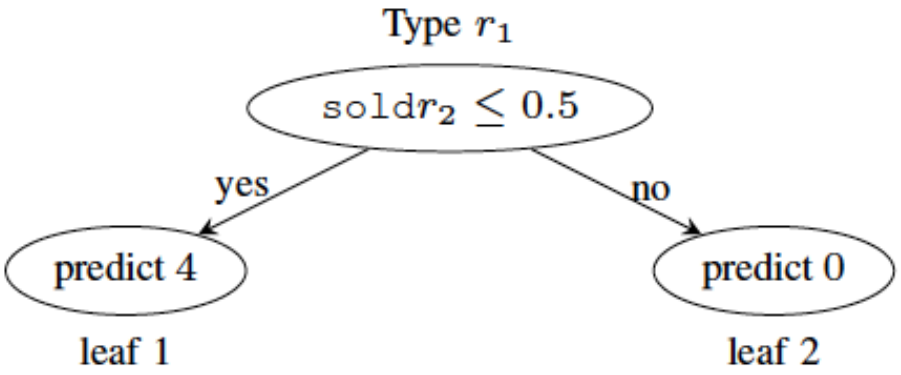
Data set

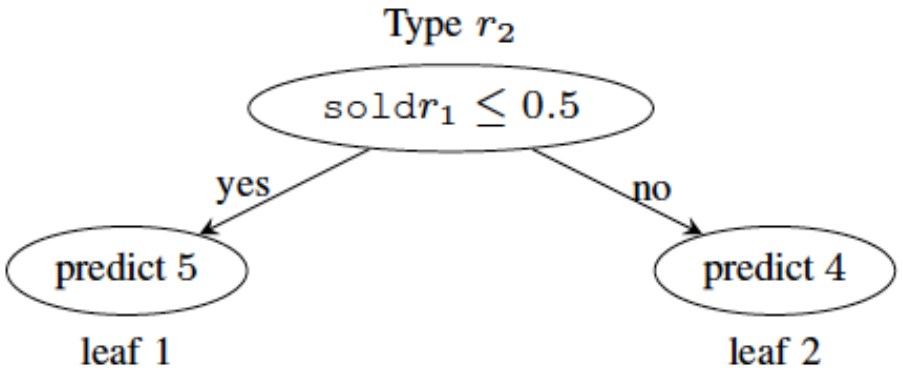
type	revenue	sold r_1	sold r_2	$diff_{1,2}$	sum	sum r_1	sum r_2	index
r_1	4	0	0	0	0	0	0	1
r_2	4	1	0	1	4	4	0	2
r_2	5	0	0	0	0	0	0	1
r_1	0	0	1	-1	5	0	5	2

Predict revenue from features

Data set and learned trees

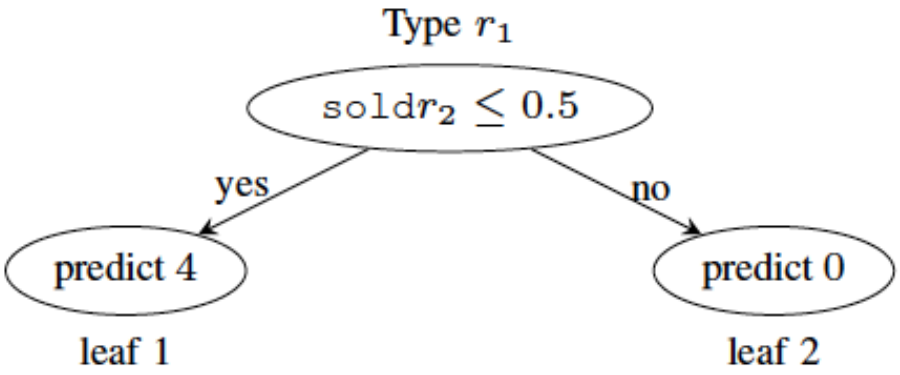
type	revenue	sold r_1	sold r_2	$diff_{1,2}$	sum	sum r_1	sum r_2	index
r_1	4	0	0	0	0	0	0	1
r_2	4	1	0	1	4	4	0	2
r_2	5	0	0	0	0	0	0	1
r_1	0	0	1	-1	5	0	5	2

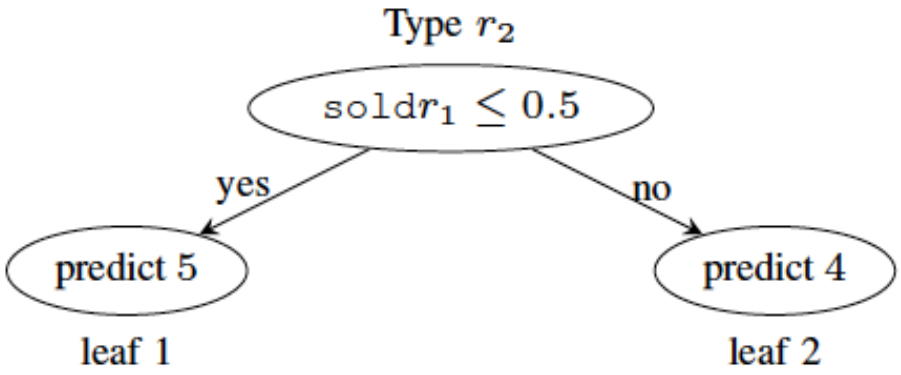




Finding an optimal order

- Given a new set of items {r1,r2,r2,r2}, compute:
  - r1,r2,r2,r2 → 4 + 4 + 4 + 4 = 16
  - r2,r1,r2,r2 → 5 + 0 + 4 + 4 = 13
  - r2,r2,r1,r2 → 5 + 5 + 0 + 4 = 14
  - r2,r2,r2,r1 → 5 + 5 + 5 + 0 = 15





Finding an optimal order

- Given a new set of items {r1,r2,r2,r2}, compute:
  - r1,r2,r2,r2 → 4 + 4 + 4 + 4 = 16
  - r2,r1,r2,r2 → 5 + 0 + 4 + 4 = 13
  - r2,r2,r1,r2 → 5 + 5 + 0 + 4 = 14
  - r2,r2,r2,r1 → 5 + 5 + 5 + 0 = 15

In this way, we compute:
"What is the value of X if we do Y?"
instead of the more common
"What is the value of X?"

The real problem

- We aim to maximize

$$V(r_1 \dots r_n) = \sum_{1 \leq k \leq n} R(r_k, \{r_j \mid j < k\}, \{r_l \mid k < l\})$$

- where
  - ***R(r,{a,b},{c,d}) is given by a model learned from data***

- Depending on the used features, this problem remains NP-hard for regression trees, linear regression, … …see paper for proofs

White-box optimization

1. **Translate** a learned model *M* to ILP constraints
2. Add constraints for **feature computation** for *M*

- *Evaluate R(i, L, L') directly in the solver!*
- The solver performs the search and cuts

White-box optimization

- Define the item order:

$$\begin{array}{rcll} \sum_{r \in R} x_{i,r} & = & 1 & \text{for all } 1 \leq i \leq n \\ \sum_{1 \leq i \leq n} x_{i,r} & = & n_r & \text{for all } r \in R \end{array}$$

- where $x_{i,r}$ Boolean
- $x_{i,r} = 1$ means a type r item is auctioned at index/position i

White-box optimization

- Define the item order:

$$\begin{aligned} \textstyle\sum_{r \in R} x_{i,r} &= 1 && \text{for all } 1 \leq i \leq n \\ \textstyle\sum_{1 \leq i \leq n} x_{i,r} &= n_r && \text{for all } r \in R \end{aligned}$$

- From these we compute features:

$$\begin{aligned} \texttt{sold}_{i,r} &= \textstyle\sum_{j<i} x_{j,r} && \text{for all } 1 \leq i \leq n, r \in R \\ \texttt{diff}_{i,r,r'} &= \texttt{sold}_{i,r} - \texttt{sold}_{i,r'} && \text{for all } 1 \leq i \leq n, r, r' \in R, r \neq r' \\ \texttt{remain}_{i,r} &= \textstyle\sum_{j>i} x_{j,r} && \text{for all } 1 \leq i \leq n, r \in R \\ \texttt{index}_i &= i && \text{for all } 1 \leq i \leq n \end{aligned}$$

White-box optimization

- Aim to maximize R():

$$\max \sum_{1 \leq i \leq n} \sum_{r \in R} p_{i,r}$$

- where $p_{i,r}$ is the expected revenue for a type r item is auctioned at position i
- $p_{i,r}$ is also included in another feature:

$$\text{sum}_{i,r} = \sum_{1 \leq j \leq i} p_{j,r} \quad \text{for all } 1 \leq i \leq n, r \in R$$

White-box optimization

- Aim to maximize R():

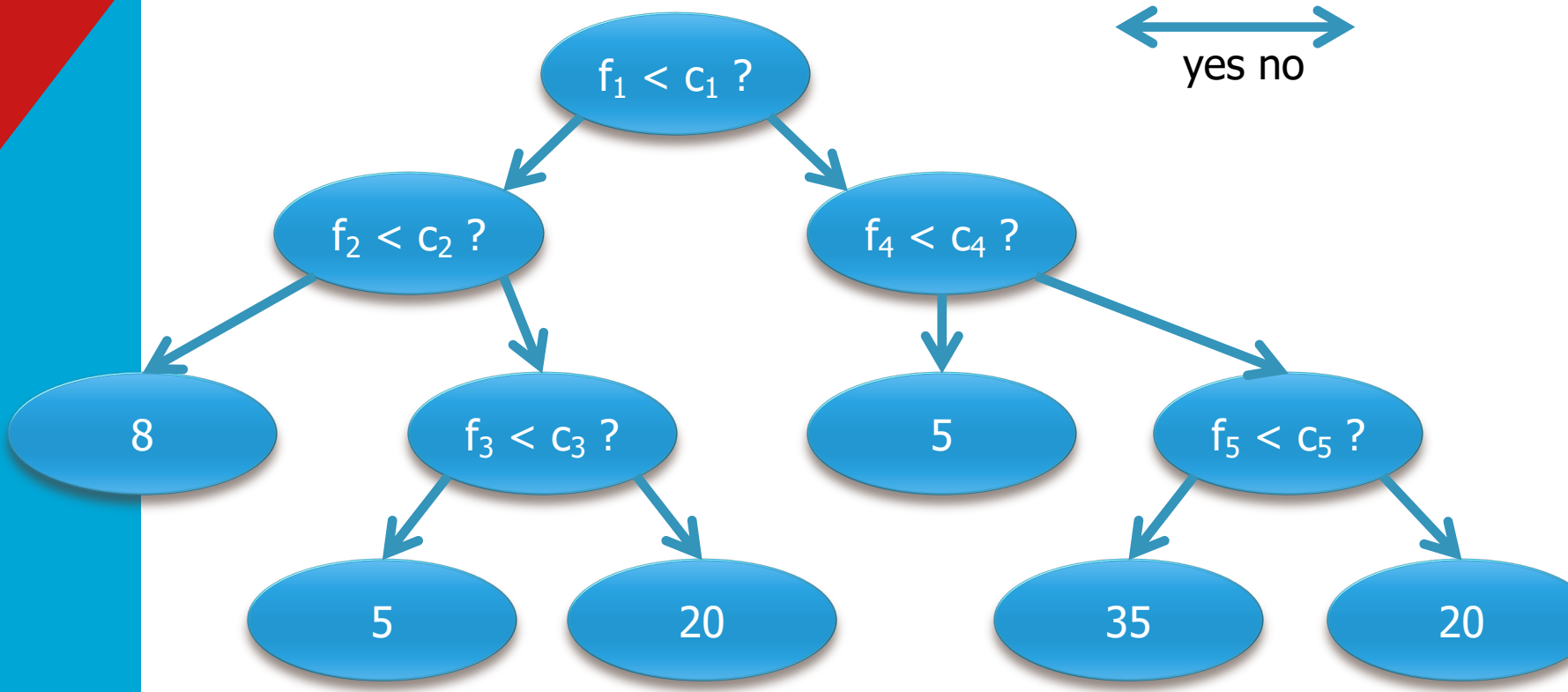
$$\max \sum_{1 \leq i \leq n} \sum_{r \in R} p_{i,r}$$

- where $p_{i,r}$ is the expected payment for a type r item is auctioned at position i
- $p_{i,r}$ is also included in another feature:

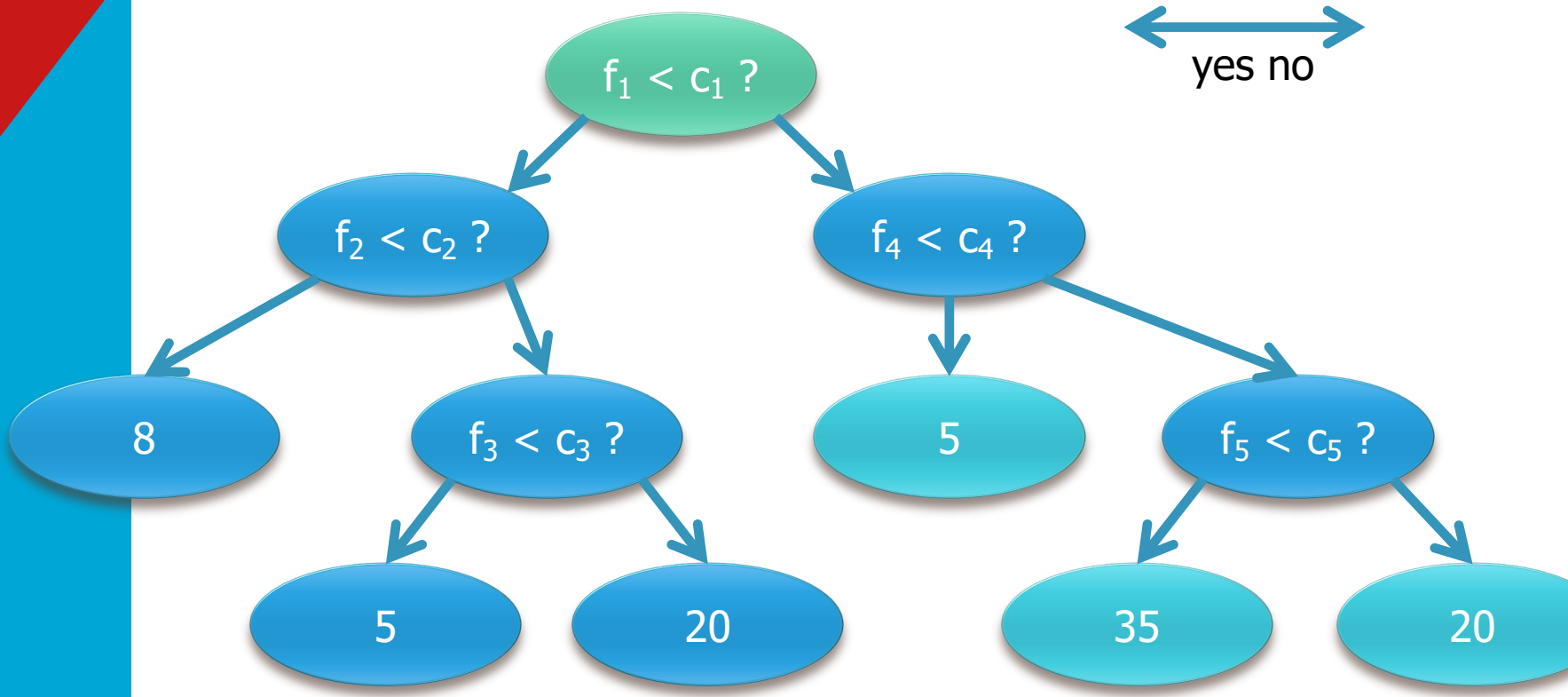
$$\text{sum}_{i,r} = \sum_{1 \leq j \leq i} p_{j,r} \quad \text{for all } 1 \leq i \leq n, r \in R$$

Predictions can be looped back into the model!

Encoding regression trees

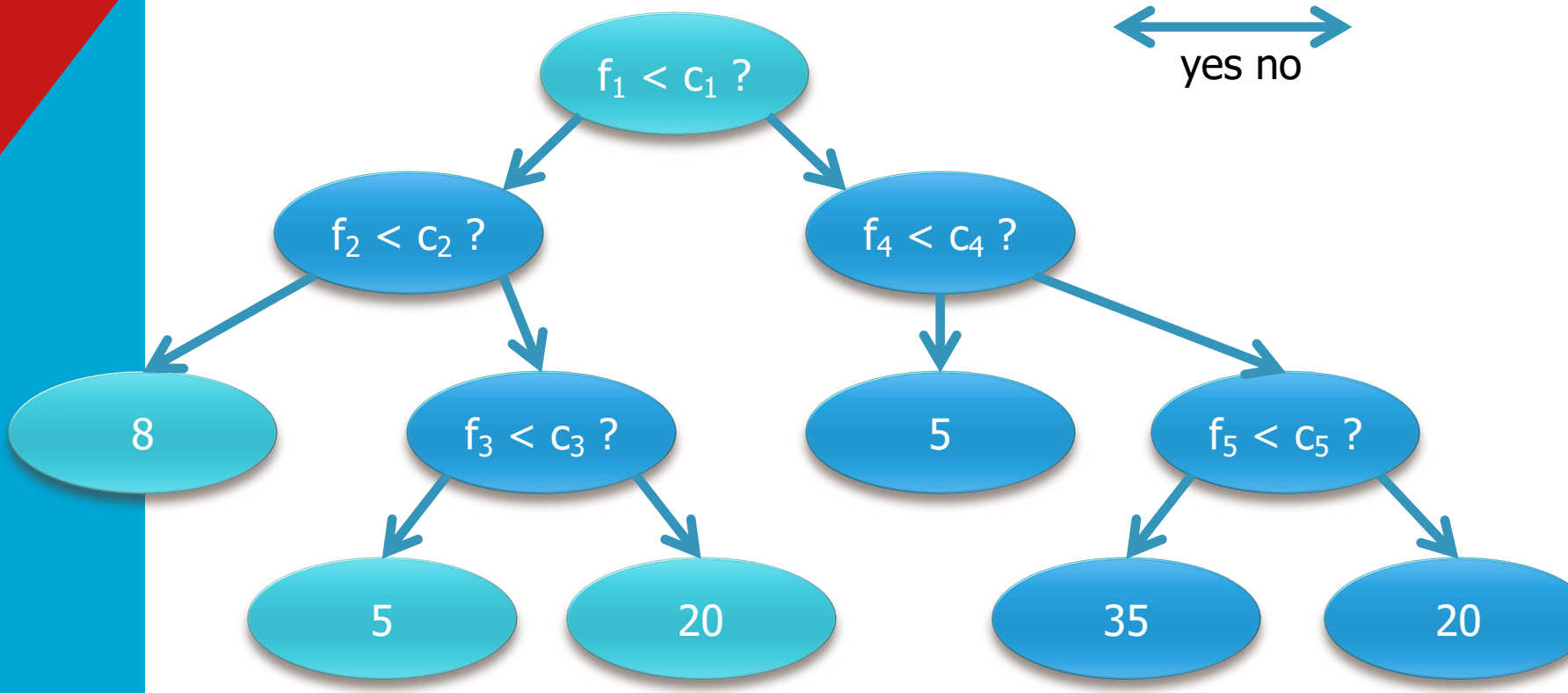


Encoding regression trees



if a node is true, then all of its right children are false

Encoding regression trees



if a node is false, then all of its left children are false

Encoding regression trees

$$\mathtt{fv}_{f,i} + (M_{f,i} - c) \cdot \sum_{l \in L} z_{i,l,r} \leq M_{f,i} \quad \text{for all } 1 \leq i \leq n, r \in R, (f \leq c) \in D_r$$
$$\mathtt{fv}_{f,i} + (m_{f,i} - c) \cdot \sum_{l \in L'} z_{i,l,r} \geq m_{f,i} \quad \text{for all } 1 \leq i \leq n, r \in R, (f \leq c) \in D_r$$

- $z_{i,l,r}$ are Boolean values for leaf nodes
- L and L' are the left and right children for decision node with constraint (f < c) in the tree for type r
- $M_{f,i}$ and $m_{f,i}$ are the maximum and minimum values for feature f at index i
- For every feature, $fv_{f,i}$ is replaced by the right-hand side of the corresponding feature calculation

The above constraints ensure that when zi,l,r obtains a value of 1, all of the binary test in the parent nodes on the path to l in the tree for type r return true at index i.

Encoding regression trees

- Exactly one leaf is true for every index-type

$$\sum_l z_{i,l,r} \quad = \quad x_{i,r} \quad \text{for all } 1 \leq i \leq n, r \in R$$

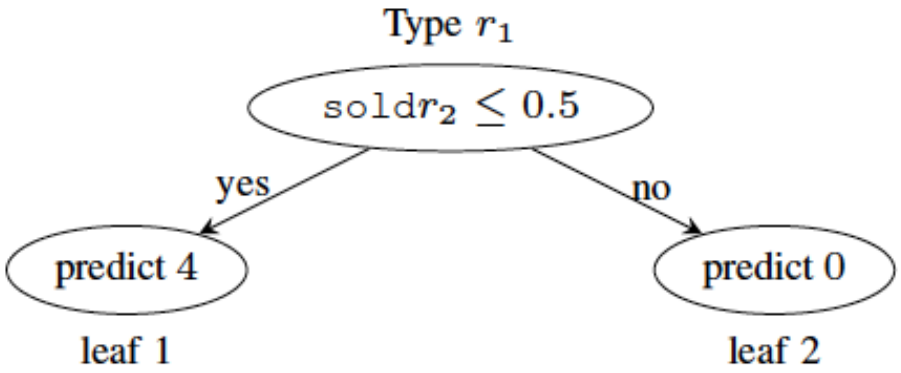
- The sum of predictions give the expected payment

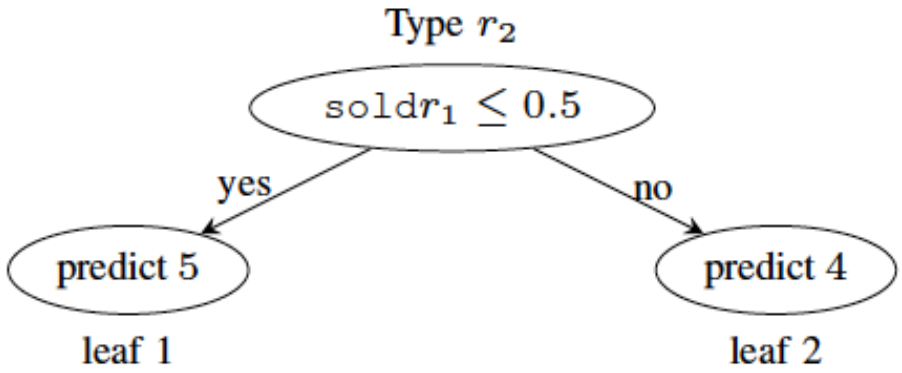
$$p_{i,r} = \sum_{l \in L_r} c_{l,r} \cdot z_{i,l,r}, \quad \text{for all } 1 \leq i \leq n, r \in R$$

where $c_{l,r}$ is the prediction for leaf l in tree r

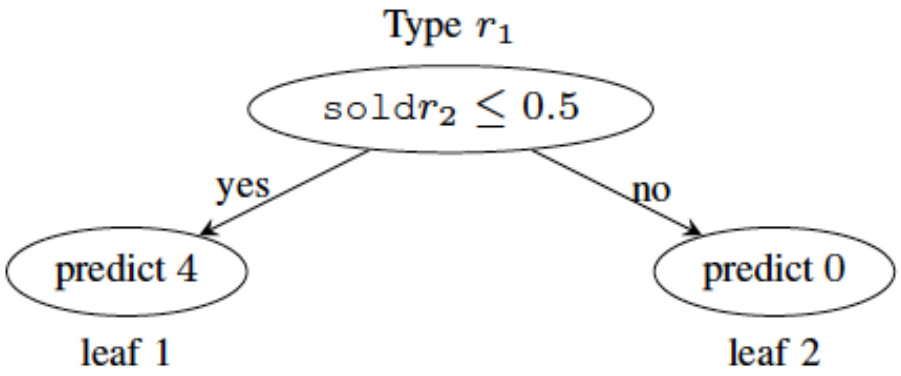
Example

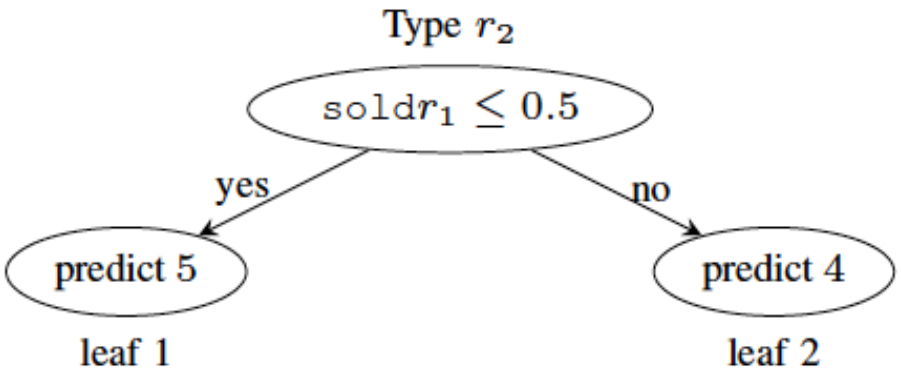
- Suppose we need to order $\{r_1, r_1, r_2\}$ for





Example



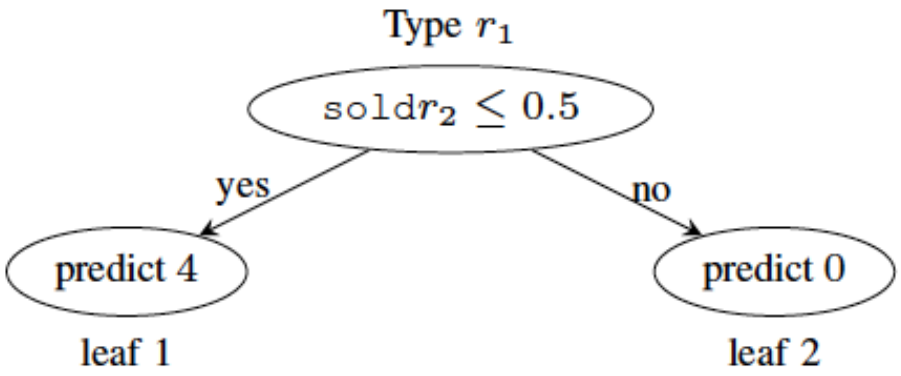


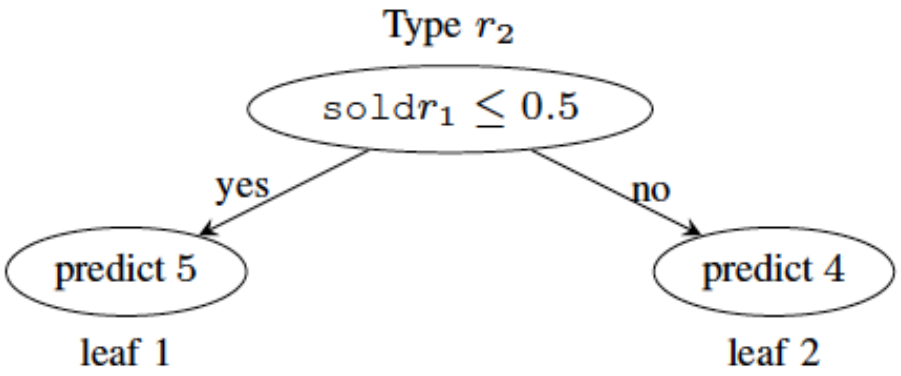
$$\max \sum_{1 \leq i \leq 3} p_{i,r_1} + p_{i,r_2}$$

$$\textit{where } p_{i,r_1} = 4z_{i,1,r_1}$$

$$\textit{and } p_{i,r_2} = 5z_{i,1,r_2} + 4z_{i,2,r_2}$$

Example



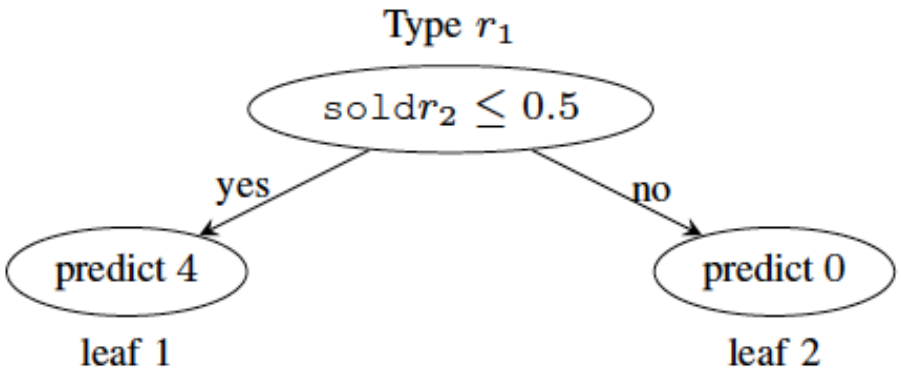


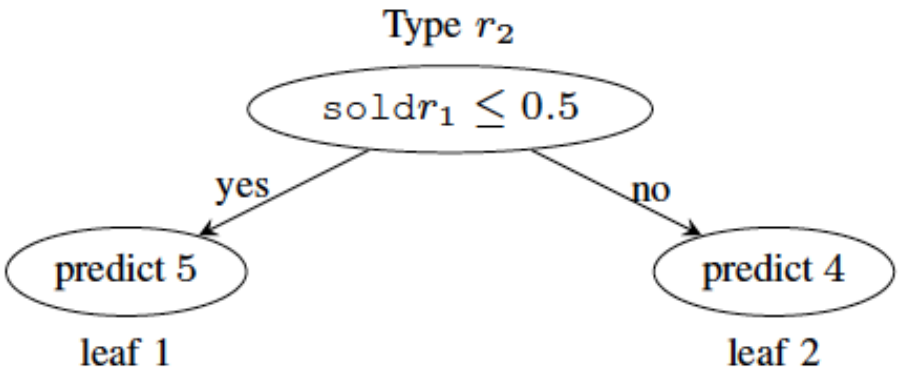
$$\max \sum_{1 \leq i \leq 3} p_{i,r_1} + p_{i,r_2}$$

$$\begin{aligned} \textit{where } p_{i,r_1} &= 4z_{i,1,r_1} \\ \textit{and } p_{i,r_2} &= 5z_{i,1,r_2} + 4z_{i,2,r_2} \end{aligned}$$

$$\begin{aligned} x_{1,r_1} + x_{2,r_1} + x_{3,r_1} &= 1 \\ x_{1,r_2} + x_{2,r_2} + x_{3,r_2} &= 2 \\ x_{i,r_1} + x_{i,r_2} &= 1 \end{aligned}$$

Example





$$\max \sum_{1 \leq i \leq 3} p_{i,r_1} + p_{i,r_2}$$

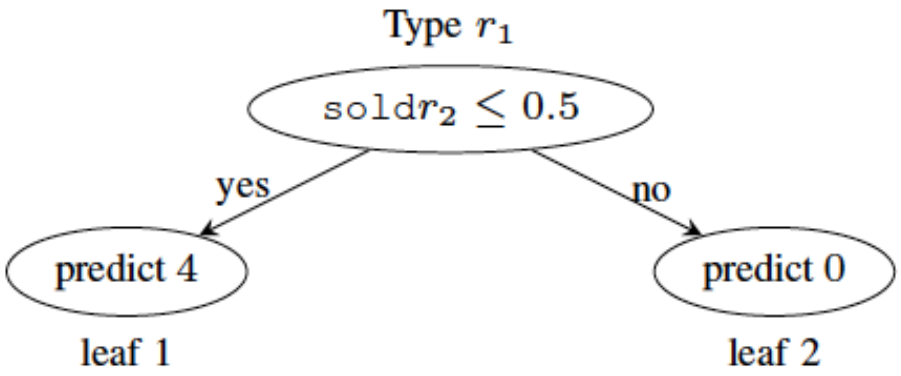
$$\begin{aligned}
\textit{where } p_{i,r_1} &= 4z_{i,1,r_1} \\
\textit{and } p_{i,r_2} &= 5z_{i,1,r_2} + 4z_{i,2,r_2}
\end{aligned}$$

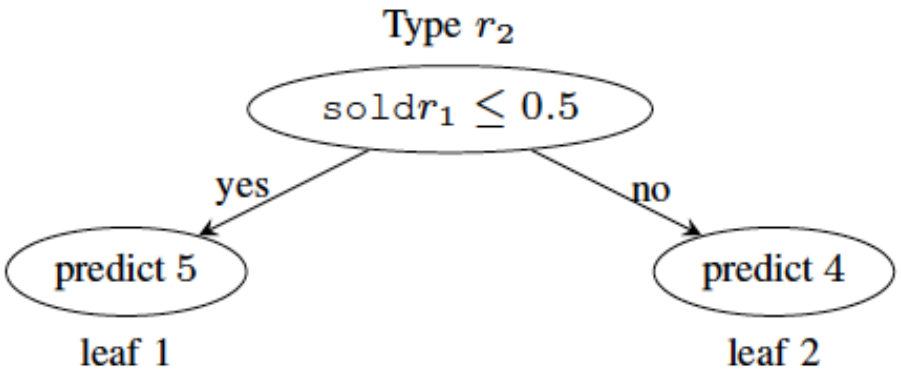
$$\begin{aligned}
\texttt{sold}_{i,r_1} &= x_{1,r_1} + \ldots + x_{i-1,r_1} \\
\texttt{sold}_{i,r_2} &= x_{1,r_2} + \ldots + x_{i-1,r_2}
\end{aligned}$$

$$\begin{aligned}
\texttt{sold}_{i,r_2} + (100 - 0.5)z_{i,1,r_1} &\leq 100 \\
\texttt{sold}_{i,r_2} + (-0.5)z_{i,2,r_1} &\geq 0 \\
\texttt{sold}_{i,r_1} + (100 - 0.5)z_{i,1,r_2} &\leq 100 \\
\texttt{sold}_{i,r_1} + (-0.5)z_{i,2,r_2} &\geq 0
\end{aligned}$$

$$\begin{aligned}
x_{1,r_1} + x_{2,r_1} + x_{3,r_1} &= 1 \\
x_{1,r_2} + x_{2,r_2} + x_{3,r_2} &= 2 \\
x_{i,r_1} + x_{i,r_2} &= 1
\end{aligned}$$

Example





$$\max \sum_{1 \leq i \leq 3} p_{i,r_1} + p_{i,r_2}$$

$$\begin{aligned} \textit{where } p_{i,r_1} &= 4z_{i,1,r_1} \\ \textit{and } p_{i,r_2} &= 5z_{i,1,r_2} + 4z_{i,2,r_2} \end{aligned}$$

$$\begin{aligned} \texttt{sold}_{i,r_1} &= x_{1,r_1} + \ldots + x_{i-1,r_1} \\ \texttt{sold}_{i,r_2} &= x_{1,r_2} + \ldots + x_{i-1,r_2} \end{aligned}$$

$$\begin{aligned} \texttt{sold}_{i,r_2} + (100 - 0.5) z_{i,1,r_1} &\leq 100 \\ \texttt{sold}_{i,r_2} + (-0.5) z_{i,2,r_1} &\geq 0 \\ \texttt{sold}_{i,r_1} + (100 - 0.5) z_{i,1,r_2} &\leq 100 \\ \texttt{sold}_{i,r_1} + (-0.5) z_{i,2,r_2} &\geq 0 \end{aligned}$$

$$\begin{aligned} x_{1,r_1} + x_{2,r_1} + x_{3,r_1} &= 1 \\ x_{1,r_2} + x_{2,r_2} + x_{3,r_2} &= 2 \\ x_{i,r_1} + x_{i,r_2} &= 1 \end{aligned}$$

$$\begin{aligned} z_{i,1,r_1} + z_{i,2,r_1} &= x_{i,r_1}, \\ z_{i,1,r_2} + z_{i,2,r_2} &= x_{i,r_2}. \end{aligned}$$

Gurobi

- Open your notebook!
- Three key API functions:
  - **model.addConstr(…)**
  - **gp.quicksum(… for i in …)**
  - **model.addVars(…)**

Encoding the auction ordering

- Ordering items
  - item type t is auctioned at index i: $x_{i,t}$ in {0,1}
  - one item is auctioned at every index:

Encoding the auction ordering

- Ordering items
  - item type t is auctioned at index i: $x_{i,t}$ in {0,1}
  - one item is auctioned at every index:

```
for i in index:
  model.addConstr(gp.quicksum(x[i,t] for t in itemtype) == 1)
```

Encoding the auction ordering

- Ordering items
  - item type t is auctioned at index i: $x_{i,t}$ in {0,1}
  - one item is auctioned at every index:

```
for i in index:
  model.addConstr(gp.quicksum(x[i,t] for t in itemtype) == 1)
```

  - the number of type t available = the number of type t auctioned

```
for t in itemtype:
  model.addConstr(gp.quicksum(x[i,t] for i in index) == nrItems[t])
```

Encoding the leafs (no copy)

- leaf l is reached at index i: $z_{i,l}$ in {0,1}
- the value tree t predicts index i: $v_{i,t}$ in [0,M]

- we create a separate $z_{i,l}$ for each tree: $z[t]_{i,l}$ in {0,1}

Encoding the leafs (no copy)

- leaf l is reached at index i: $z_{i,l}$ in {0,1}
- the value tree t predicts index i: $v_{i,t}$ in [0,M]

- we create a separate $z_{i,l}$ for each tree: $z[t]_{i,l}$ in {0,1}

```
for i in index:
  for t in itemtype:
    model.addConstr(gp.quicksum(z[t][i,l] for l in leafs[t]) == x[i,t])
```

Encoding the leafs (no copy)

- leaf l is reached at index i: $z_{i,l}$ in {0,1}
- the value tree t predicts index i: $v_{i,t}$ in [0,M]

- we create a separate $z_{i,l}$ for each tree: $z[t]_{i,l}$ in {0,1}

```
for i in index:
  for t in itemtype:
    model.addConstr(gp.quicksum(z[t][i,l] for l in leafs[t]) == x[i,t])
```

- a leaf in tree t is reached iff an item of type t is auctioned

```
    model.addConstr(gp.quicksum(value(l,t) * z[t][i,l] for l in leafs[t]) ==
          v[i,t])
```

Encoding the leafs (no copy)

- leaf l is reached at index i: $z_{i,l}$ in {0,1}
- the value tree t predicts index i: $v_{i,t}$ in [0,M]

- we create a separate $z_{i,l}$ for each tree: $z[t]_{i,l}$ in {0,1}

```
for i in index:
  for t in itemtype:
    model.addConstr(gp.quicksum(z[t][i,l] for l in leafs[t]) == x[i,t])
```

- a leaf in tree t is reached iff an item of type t is auctioned

```
    model.addConstr(gp.quicksum(value(l,t) * z[t][i,l] for l in leafs[t]) ==
          v[i,t])
```

- and it predicts the leaf value

Encoding the leafs (no copy)

- leaf l is reached at index i: $z_{i,l}$ in {0,1}
- the value tree t predicts index i: $v_{i,t}$ in [0,M]

- we create a separate $z_{i,l}$ for each tree: $z[t]_{i,l}$ in {0,1}

```
for i in index:
  for t in itemtype:
    model.addConstr(gp.quicksum(z[t][i,l] for l in leafs[t]) == x[i,t])
```

- a leaf in tree t is reached iff an item of type t is auctioned

```
    model.addConstr(gp.quicksum(value(l,t) * z[t][i,l] for l in leafs[t]) ==
          v[i,t])
```

- and it predicts the leaf value

This is a constant!

Encoding the nodes (no copy)

- for tree t, node n evaluates to true at index i: $y[t]_{i,n}$ in {0,1}

```
for t in itemtype:
  for i in index:
    for n in nodes[t]:
      f = feature(n,t)
      model.addConstr(y[t][i,n]*threshold(n,t) <= feat[features[f]][i])
      model.addConstr(threshold(n,t) + y[t][i,n]*maxvalues[f] >=
          feat[features[f]][i])
```

Encoding the nodes (no copy)

- for tree t, node n evaluates to true at index i: $y[t]_{i,n}$ in {0,1}

```
for t in itemtype:
  for i in index:
    for n in nodes[t]:
      f = feature(n,t)
      model.addConstr(y[t][i,n]*threshold(n,t) <= feat[features[f]][i])
      model.addConstr(threshold(n,t) + y[t][i,n]*maxvalues[f] >=
          feat[features[f]][i])
```

This is a constant!

This is a free variable!

Encoding the structure (no copy)

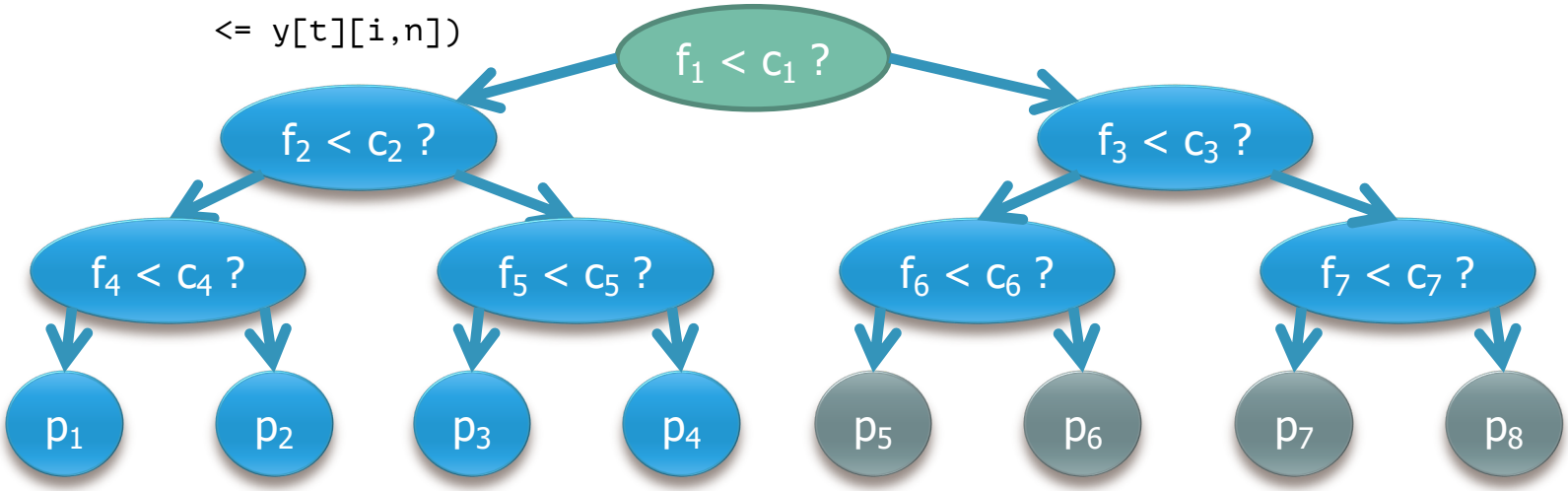
- for tree t, node n evaluates to true at index i: $y[t]_{i,n}$ in {0,1}

```
for t in itemtype:
  for i in index:
    for n in nodes[t]:
      model.addConstr(gp.quicksum(z[t][i,l] for l in left_children(t,n)) <=
          1 - y[t][i,n])
      model.addConstr(gp.quicksum(z[t][i,l] for l in right_children(t,n))
          <= y[t][i,n])
```

Encoding the structure (no copy)

- for tree t, node n evaluates to true at index i: $y[t]_{i,n}$ in {0,1}

```
for t in itemtype:
  for i in index:
    for n in nodes[t]:
      model.addConstr(gp.quicksum(z[t][i,l] for l in left_children(t,n)) <=
          1 - y[t][i,n])
      model.addConstr(gp.quicksum(z[t][i,l] for l in right_children(t,n))
          <= y[t][i,n])
```



if a node is true, then all of its right children are false

Encoding the features

```
for i in index:
  model.addConstr(feat["sumv"][i] ==

  model.addConstr(feat["index"][i] ==


for i in index:
  for t in itemtype:
    model.addConstr(feat["sold_" + t][i] ==

    model.addConstr(feat["remain_" + t][i] ==


for i in index:
  for t in itemtype:
    model.addConstr(feat["sum_" + t][i] ==
```

Encoding the features

```
for i in index:
  model.addConstr(feat["sumv"][i] ==
      gp.quicksum(v[j,t] for j in range(i) for t in itemtype))
  model.addConstr(feat["index"][i] ==


for i in index:
  for t in itemtype:
    model.addConstr(feat["sold_" + t][i] ==

    model.addConstr(feat["remain_" + t][i] ==


for i in index:
  for t in itemtype:
    model.addConstr(feat["sum_" + t][i] ==
```

Encoding the features

```
for i in index:
  model.addConstr(feat["sumv"][i] ==
      gp.quicksum(v[j,t] for j in range(i) for t in itemtype))
  model.addConstr(feat["index"][i] ==
      i)

for i in index:
  for t in itemtype:
    model.addConstr(feat["sold_" + t][i] ==

    model.addConstr(feat["remain_" + t][i] ==


for i in index:
  for t in itemtype:
    model.addConstr(feat["sum_" + t][i] ==
```

Encoding the features

```
for i in index:
  model.addConstr(feat["sumv"][i] ==
      gp.quicksum(v[j,t] for j in range(i) for t in itemtype))
  model.addConstr(feat["index"][i] ==
      i)

for i in index:
  for t in itemtype:
    model.addConstr(feat["sold_" + t][i] ==
        gp.quicksum(x[j,t] for j in range(i)))
    model.addConstr(feat["remain_" + t][i] ==


for i in index:
  for t in itemtype:
    model.addConstr(feat["sum_" + t][i] ==
```

Encoding the features

```
for i in index:
  model.addConstr(feat["sumv"][i] ==
      gp.quicksum(v[j,t] for j in range(i) for t in itemtype))
  model.addConstr(feat["index"][i] ==
      i)

for i in index:
  for t in itemtype:
    model.addConstr(feat["sold_" + t][i] ==
        gp.quicksum(x[j,t] for j in range(i)))
    model.addConstr(feat["remain_" + t][i] ==
        gp.quicksum(x[j,t] for j in range(i+1, len(index))))

for i in index:
  for t in itemtype:
    model.addConstr(feat["sum_" + t][i] ==
```

Encoding the features

```
for i in index:
  model.addConstr(feat["sumv"][i] ==
      gp.quicksum(v[j,t] for j in range(i) for t in itemtype))
  model.addConstr(feat["index"][i] ==
      i)

for i in index:
  for t in itemtype:
    model.addConstr(feat["sold_" + t][i] ==
        gp.quicksum(x[j,t] for j in range(i)))
    model.addConstr(feat["remain_" + t][i] ==
        gp.quicksum(x[j,t] for j in range(i+1, len(index))))

for i in index:
  for t in itemtype:
    model.addConstr(feat["sum_" + t][i] ==
        gp.quicksum(v[j,t] for j in range(i)))
```

Encoding the features

```
for i in index:
  model.addConstr(feat["sumv"][i] ==
      gp.quicksum(v[j,t] for j in range(i) for t in itemtype))
  model.addConstr(feat["index"][i] ==
      i)

for i in index:
  for t in itemtype:
    model.addConstr(feat["sold_" + t][i] ==
        gp.quicksum(x[j,t] for j in range(i)))
    model.addConstr(feat["remain_" + t][i] ==
```

Predictions can be looped back into the model!

```
    model.addConstr(feat["sum_" + t][i] ==
        gp.quicksum(v[j,t] for j in range(i)))
```

Encoding the features

```
for i in index:
  model.addConstr(feat["sumv"][i] ==
      gp.quicksum(v[j,t] for j in range(i) for t in itemtype))
  model.addConstr(feat["index"][i] ==
      i)

for i in index:
  for t in itemtype:
    model.addConstr(feat["sold_" + t][
        gp.quicksum(x[j,t] for j in ra
    model.addConstr(feat["remain_" + t][
```

Any feature! As long as you can efficiently represent it in a MIP…

Predictions can be looped back into the model!

```
    model.addConstr(feat["sum_" + t][i] ==
        gp.quicksum(v[j,t] for j in range(i)))
```

Code for Gurobi

```
for t in itemtype:
  model.addConstr(gp.quicksum(x[i,t] for i in index) == nrItems[t])
for i in index:
  model.addConstr(gp.quicksum(x[i,t] for t in itemtype) == 1)


for i in index:
  model.addConstr(feat["sumv"][i] == gp.quicksum(v[j,t] for j in range(i) for t in itemtype))
  model.addConstr(feat["index"][i] == i)


for i in index:
  for t in itemtype:
    model.addConstr(feat["sold_" + t][i] == gp.quicksum(x[j,t] for j in range(i)))
    model.addConstr(feat["remain_" + t][i] == gp.quicksum(x[j,t] for j in range(i+1, len(index))))
    model.addConstr(feat["sum_" + t][i] == gp.quicksum(v[j,t] for j in range(i)))
```

Lab

- Get it working!
- Run the found order on the auction simulator:
  - `!python3 auction/experiment.py test`
- Create new bidders and data:
  - `!python3 auction/experiment.py run`
- Create new features:
  - More A items have been auctioned than B items
  - The difference between numbers sold of different items
  - The difference between sold values of same/different items
  - …

Two optimization strategies

Black-box

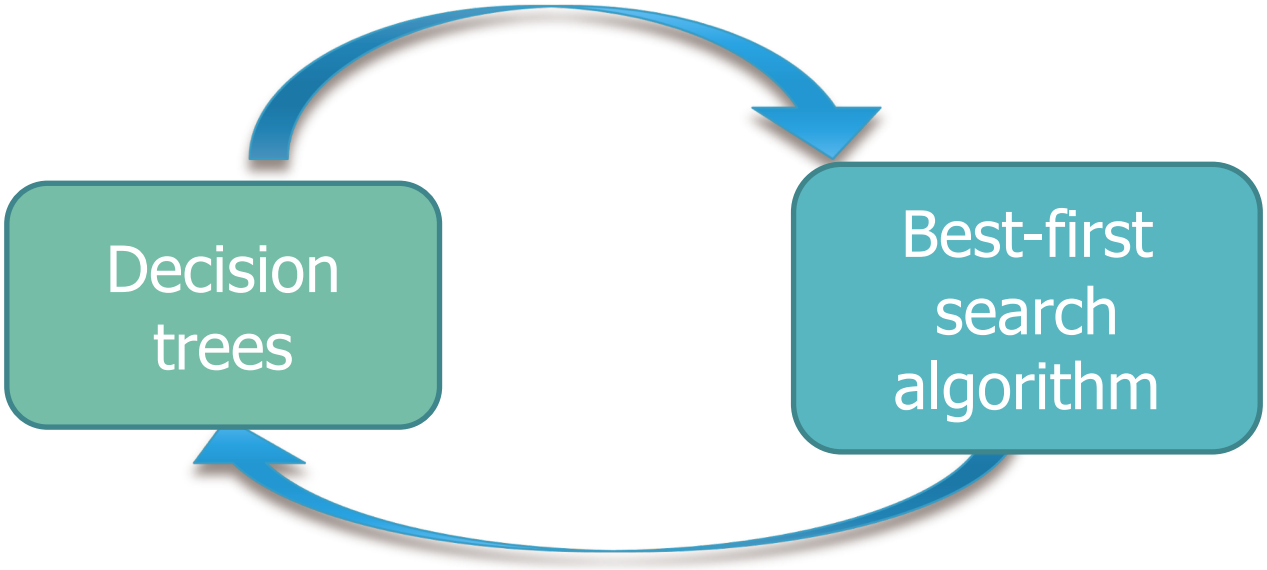
- Use regression trees for predictions
- Search for optimal by
  - Evolutionary computation, best-first search, …

White-box

- Map regression trees to ILP/SAT
- Solve using solver (CPLEX)

Black-box optimization

P(r1) + P(r2) + P(r3)



Candidate solution:
ordering (r1,r2,r3)

White-box optimization

- Translate a learned model M to ILP/SAT constraints
- Add constraints for feature computation for M

- Evaluate R(i, L, L') directly in the solver!
- The solver performs the search and cuts

White-box optimization

- Translate a learned model M to ILP/SAT constraints
- Add constraints for feature computation for M

- Evaluate R(i, L, L') directly in the solver!
- The solver performs the search and cuts

1. Use as a data module directly in existing OR models
2. Model unknown behaviors/properties
3. Automatic updates of constraints and parameters
4. ...

Advantages/Disadvantages

Black-box

- Fast
- Use custom heuristics and dynamic cuts

- Optimality requires searching everything
- Logical cuts impossible

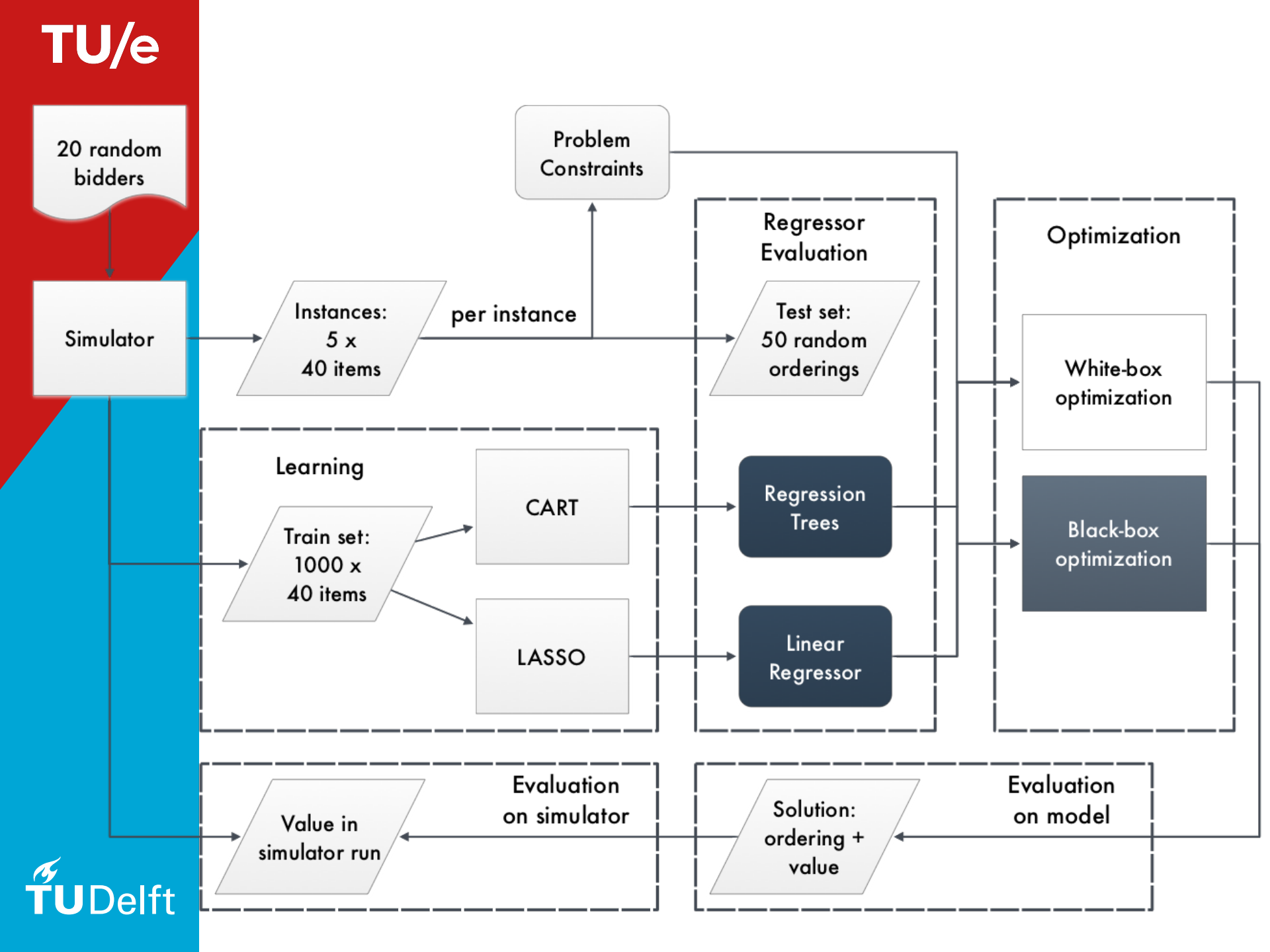
White-box

- Computes inside ILP
- Cuts and heuristics come with the solver

- Can provably find the optimum
- Can apply logical cuts

Evaluation with an auction simulator

- Generate 40 random agents for 8 item types
  - For every item type generate a popularity p and sparsity s
  - Every item type i has a base value bi of 25 + 5i
- Every agent can desire 1 to 5 of these types
  - Agents are likely to desire similar items, using p
  - Every desired item gets assigned a value from [0.5,2.0] * bi
- Every agent has a random budget from [25,150]
  - Increased by a random addition if less than desired values
- 40 items in every auction
  - Popular types are more likely generated, using s

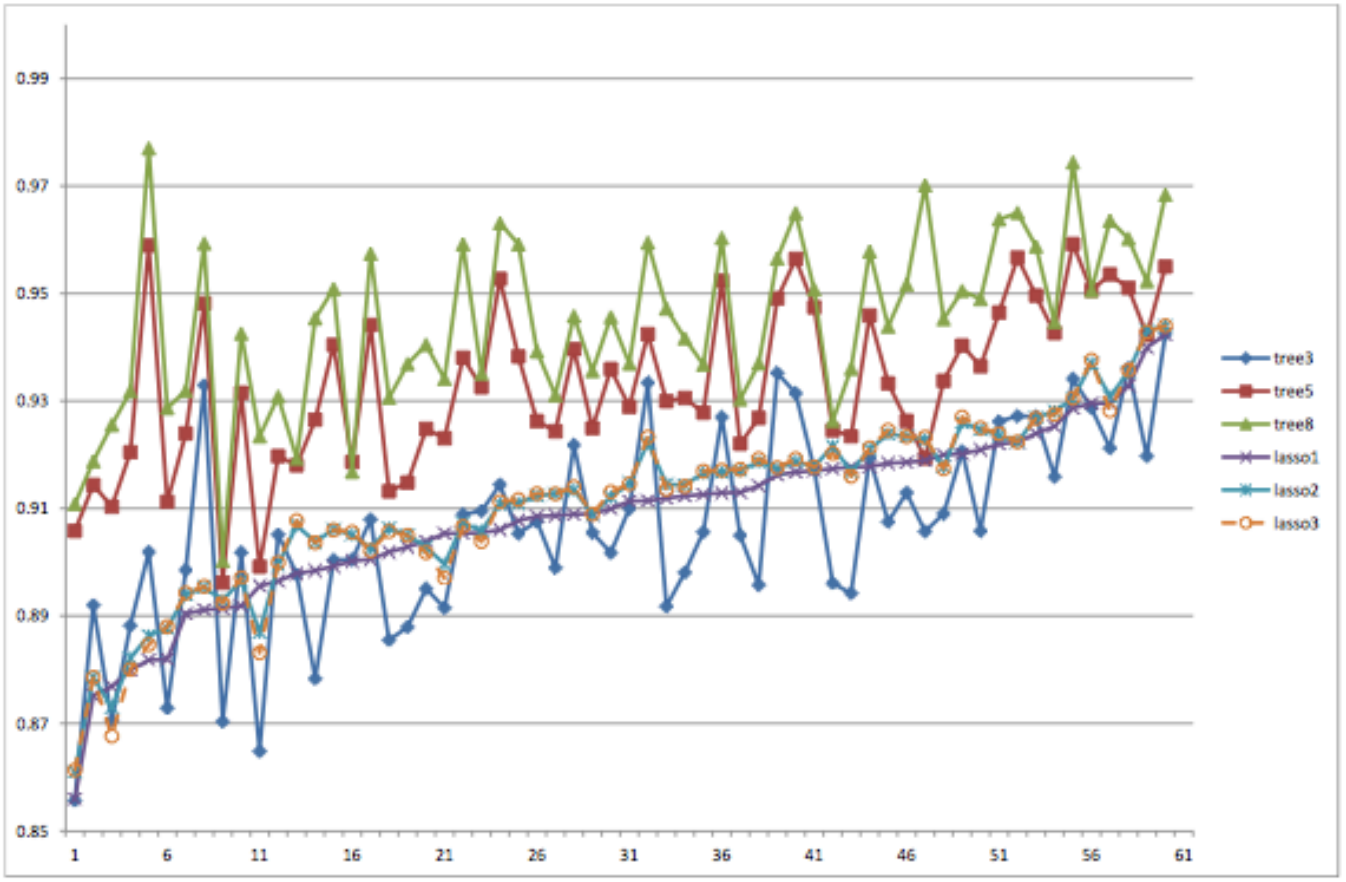

TU/e
20 random bidders
Simulator
Problem Constraints
Instances: 5 x 40 items
per instance
Regressor Evaluation
Test set: 50 random orderings
Optimization
White-box optimization
Learning
Train set: 1000 x 40 items
CART
LASSO
Regression Trees
Linear Regressor
Black-box optimization
Value in simulator run
Evaluation on simulator
Solution: ordering + value
Evaluation on model
TUDelft

Evaluating models

- Learned prediction models
  - learned trees of different depths of 3, 5, 8 (tree3, tree5, tree8)
  - LASSO regressor with 3 different values for regularization parameter: 1.0, 0.1, 0.000001 (lasso1, lasso2, lasso3)
- Ordering models
  - The ILP models built from the 6 learned regressors
  - best-first search, evaluated using the 6 regressors
  - auctioning the most valuable item first (i.e., mvf)
  - random ordering strategy (i.e., mean5000)
- Lower bound on the optimal ordering
  - the best ordering among 5000 orderings, evaluated in the simulator

TU/e

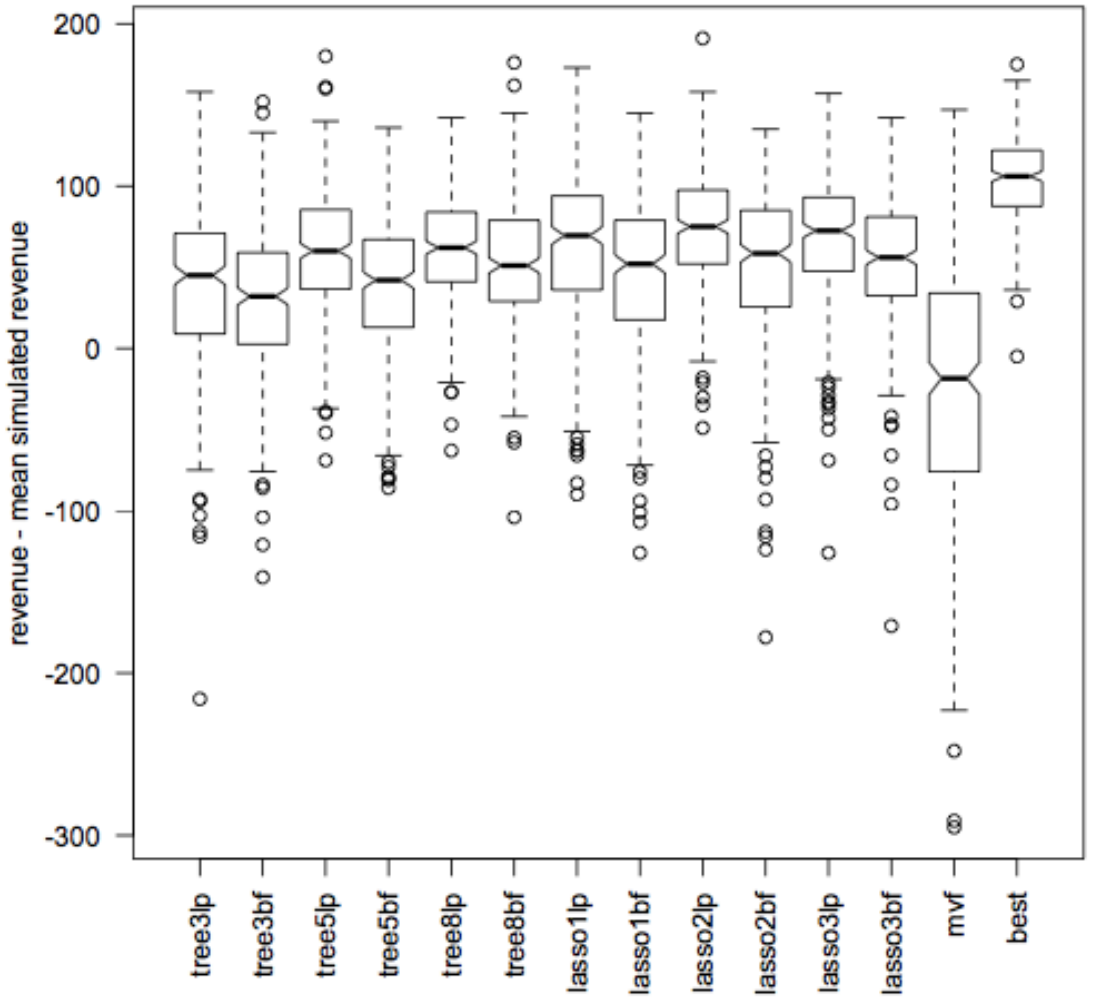
R2 scores for different learning models for 60 different sets of agents. The observed values are those returned by the simulator. Each score is computed from 10000 values.



TUDelft

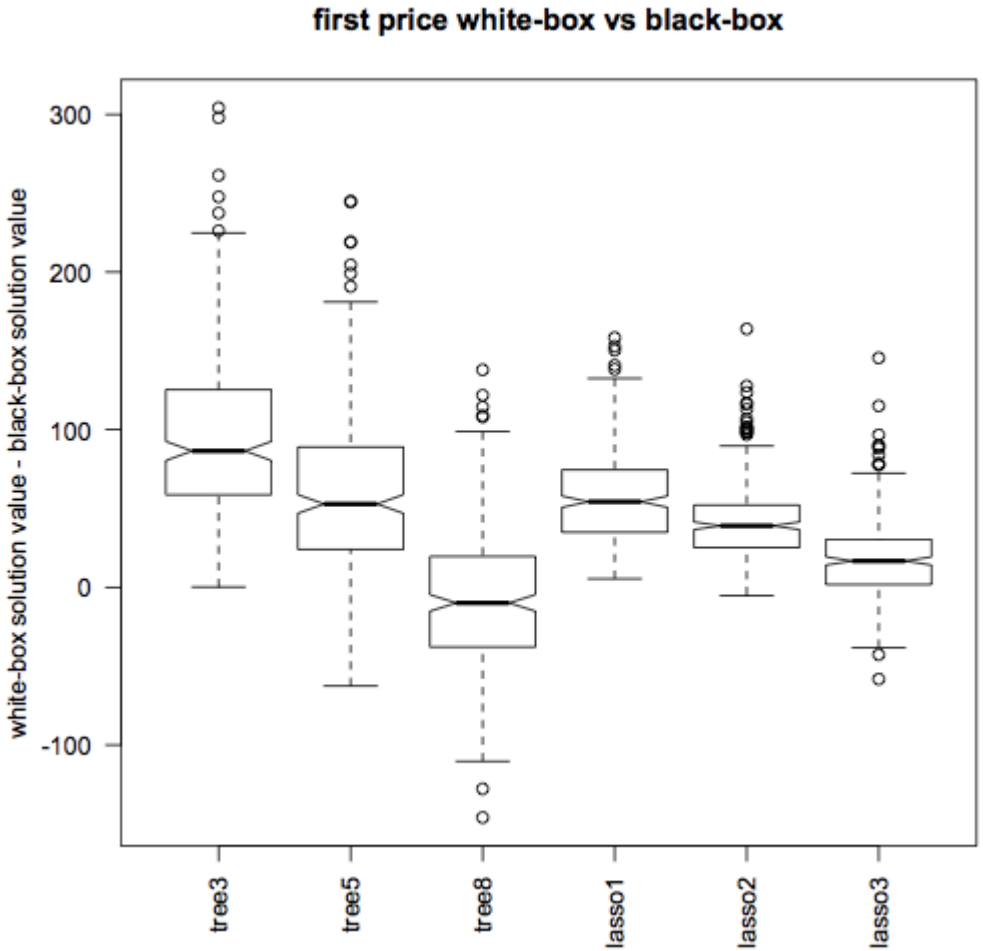
TU/e

The performance of the different ordering methods evaluated by the simulator, compared to mean5000. The simulated auctions are first-price auctions. Each box contains 300 values.

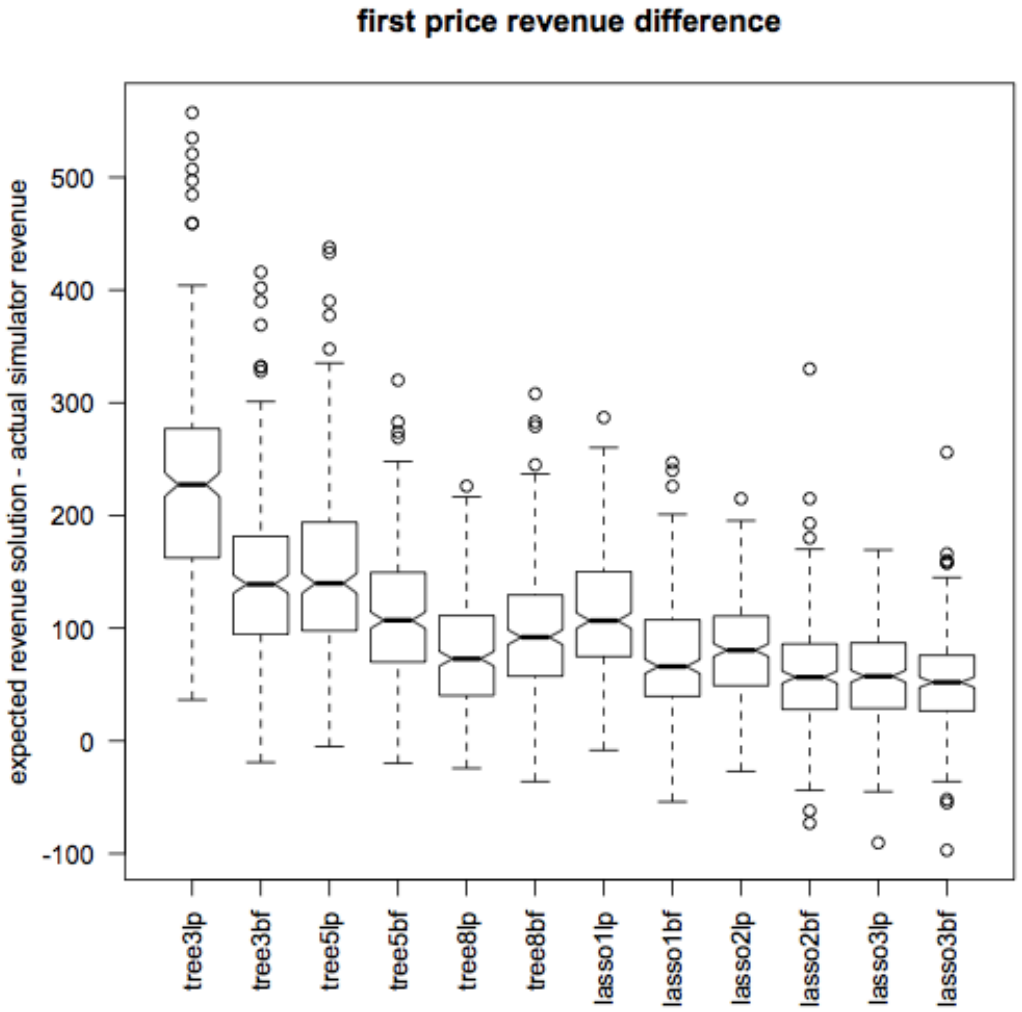


TUDelft

Performance difference between the LP model and the best-first search, evaluated by the predictive model. Each box contains 300 values.



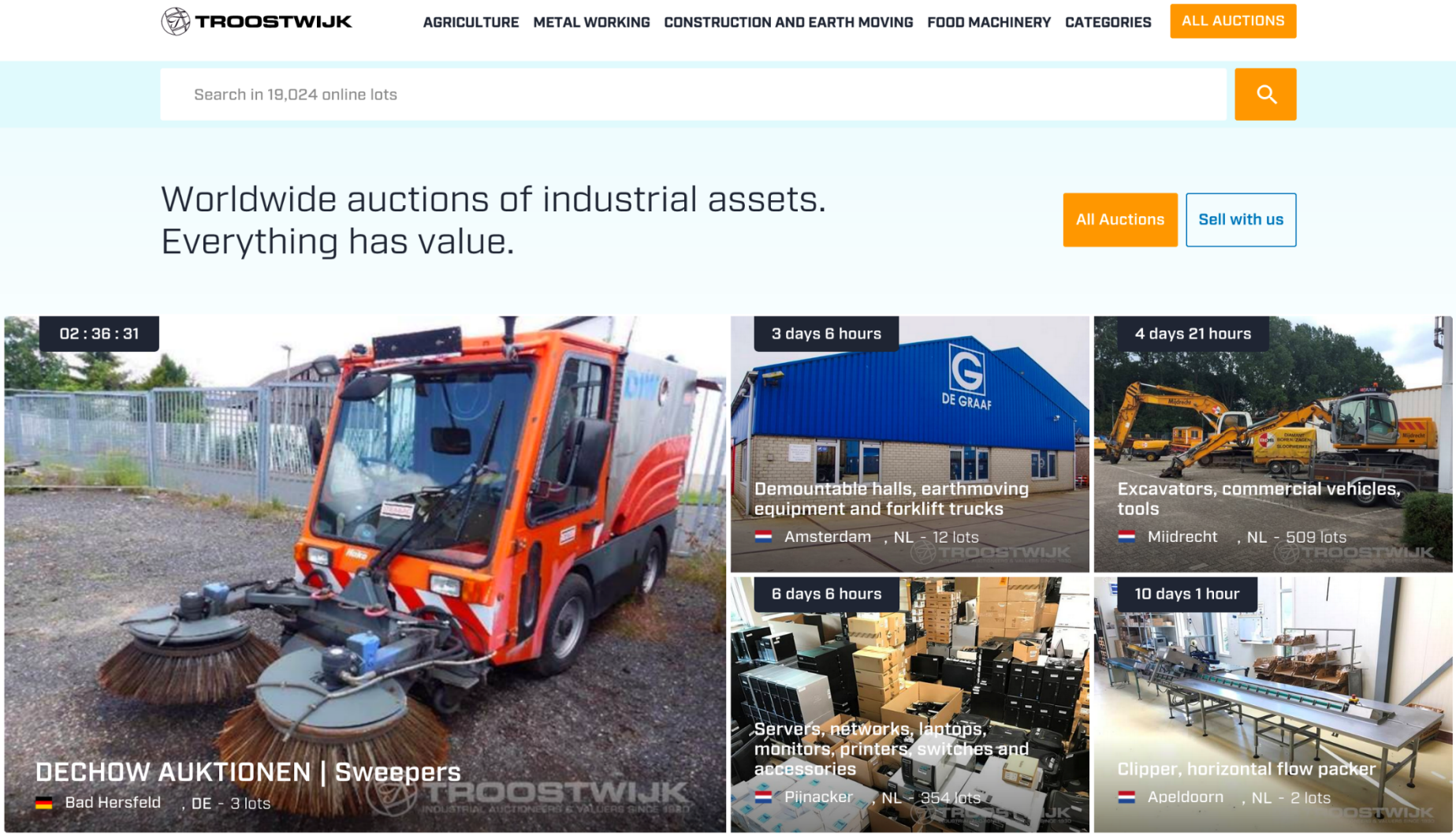
Performance difference between the values evaluated by the model and the values evaluated by the simulator



Put white-box optimization into practice: *Data driven design for online industrial auctions*

Qing Chuan Ye, Jason Rhuggenaath, Yingqian Zhang,
Sicco Verwer, Michiel Jurgen Hilgeman. AMAI.

Online auction sites



Online auctions

- Lots of historic data
  - >1000 auctions (or sales) per year
  - Each sale lasts several weeks; has multiple lots
  - (online) bidders from many countries (> 125K bidders)
- English auction – highest bidder wins
- Wide variety of goods

Auction design

- The objective of auctioneer:
  - maximize the revenue, i.e., maximizing selling price of each lot of each sale
- How to measure whether the auction is successful or not?
  - Winning bid (end price, or selling price)
  - Multiplier = End price/Estimated value
  - (3 classes: low; mid; high)
- Auction design optimization problem:
  - Maximizing the expected revenue of a given auction
  - Main design variables: lot Nr, starting price of each lot

Online auction data

SaleNumber	LotNr	EstValue	SP	SPEV	Main	Sub	Mult	yClass
19604	1	11000	5500	0,5	equipment	telescopic handlers	2,363636	2
19604	2	5000	2000	0,4	vehicles	vans	0,88	2
19604	3	1500	750	0,5	vehicles	cars	0,8	2
19604	4	2000	1000	0,5	equipment	sweeping machines	0,5525	1
19604	5	2500	1250	0,5	vehicles	trailers	0,64	1
19604	40	800	400	0,5	equipment	supplies	0,5	1
19604	44	200	100	0,5	equipment	supplies	4,3	2
19604	46	400	360	0,9	equipment	work shop equipment	2,5	2
19604	47	600	300	0,5	equipment	intern transport	0,733333	1
19604	57	300	250	0,833333	equipment	work shop equipment	2,533333	2

Online auction data --- prediction

F()

SaleNumber	LotNr	EstValue	SP	SPEV	Main	Sub	Mult	yClass
19604	1	11000	5500	0,5	equipment	telescopic handlers	2,3636	2
19604	2	5000	2000	0,4	vehicles	vans	0,88	2
19604	3	1500	750	0,5	vehicles	cars	0,8	2
19604	4	2000	1000	0,5	equipment	sweeping machines	0,5525	1
19604	5	2500	1250	0,5	vehicles	trailers	0,64	1
19604	40	800	400	0,5	equipment	supplies	0,5	1
19604	44	200	100	0,5	equipment	supplies	4,3	2
19604	46	400	360	0,9	equipment	work shop equipment	2,5	2
19604	47	600	300	0,5	equipment	intern transport	0,733333	1
19604	57	300	250	0,833333	equipment	work shop equipment	2,533333	2

Correlation

Feature	Multiplier (all)	Multiplier (sold)
LotNr	**-0.08**	**-0.12**
Allocate	-0.34	-0.03
EstValue	-0.07	0.03
StartPrice	**0.06**	**0.16**
Seller	-0.11	-0.08
CloseTime	-0.05	-0.08
Weekday	-0.02	-0.05
SPEV (StartPrice/EstPrice)	**0.37**	**0.33**
LotsSale	*0.03*	*-0.03*
LotsMain	*-0.06*	*-0.07*
LotsSub	*-0.08*	*-0.12*

Classification Tree prediction

We first predict the performance (multiplier class) of specific lots

- Results: three classes (low, mid, high) multiplier

	Accuracy
Cart depth 3	0.64
Cart depth 5	0.67
Cart depth 7	0.69
Cart depth 10	0.72
Random Forest	0.75

- *In spite of large good variety, we get OK accuracy*

Online auction data --- optimization

F()

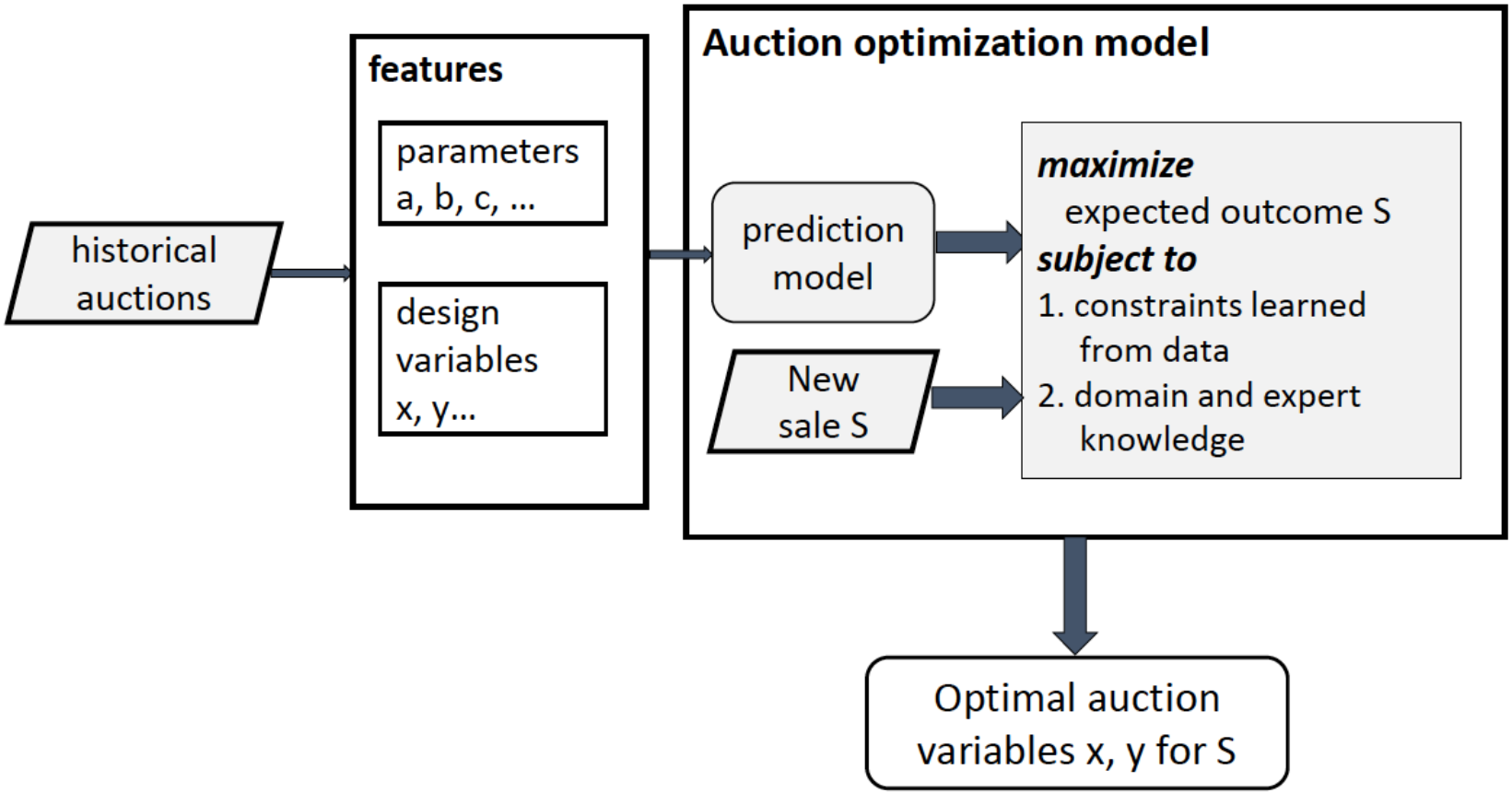
SaleNumber	LotNr	EstValue	SP	SPEV	Main	Sub	Mult	yClass
19604	1	11000	5500	0,5	equipment	telescopic handlers	2,363636	2
19604	X	5000	A	A/5000	vehicles	vans	F(X,..)	F(X,..)
19604	Y	1500	B	B/1500	vehicles	cars	F(Y,..)	F(Y,..)
19604	4	2000	1000	0,5	equipment	sweeping machines	0,5525	1
19604	5	2500	1250	0,5	vehicles	trailers	0,64	1
19604	40	800	400	0,5	equipment	supplies	0,5	1
19604	44	200	100	0,5	equipment	supplies	4,3	2
19604	46	400	360	0,9	equipment	work shop equipment	2,5	2
19604	47	600	300	0,5	equipment	intern transport	0,733333	1
19604	57	300	250	0,833333	equipment	work shop equipment	2,533333	2

Auction optimization: white-box approach

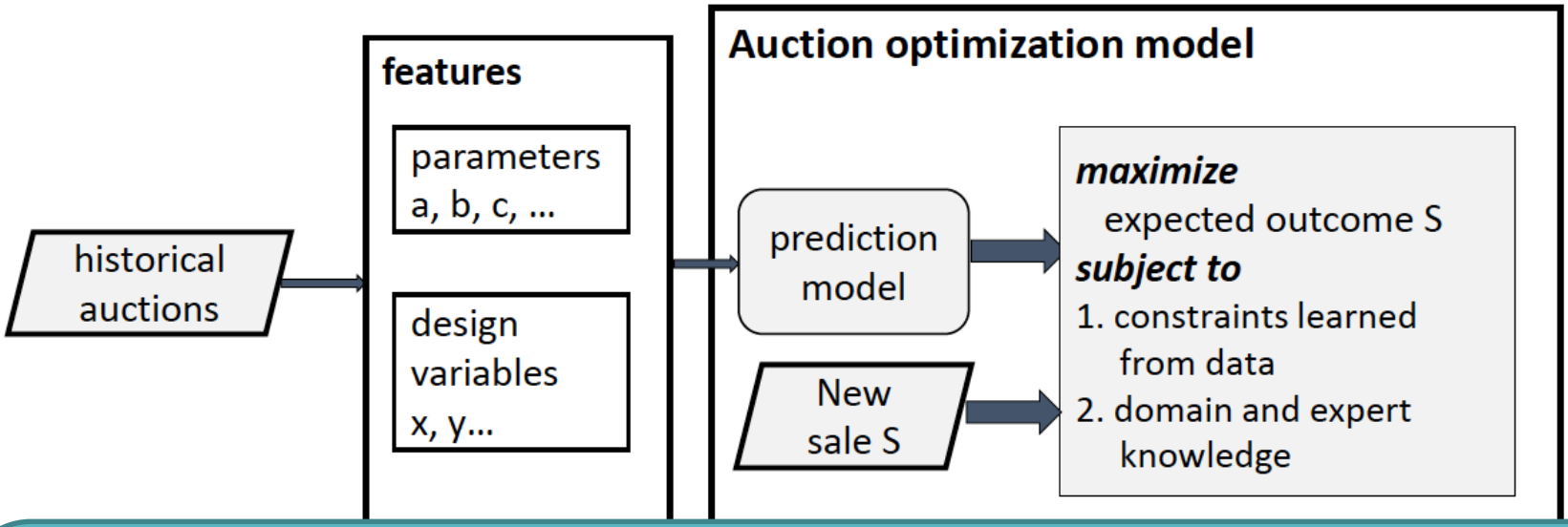
- Translate a learned model M to ILP/SAT constraints
- Add constraints for feature computation for M

- Evaluate F() directly in the solver!
- The solver performs the search and cuts
- Other expert/domain knowlege can be easily added

Auction optimization



Auction optimization



1. Use as a data module directly in existing OR models
2. Model unknown behaviors/properties
3. Integrate data and expert knowledge
4. Automatic updates of constraints and parameters
5. ...

Decision variables & objective

- Design for a set I of N lots:
- Decision variables:
  - Starting price of lot r: s_r
  - Index of lots (lotNr): $x_{i,r} \in \{0, 1\}$
- Objective function: $\max \sum_{1 \leq r \leq N} \sum_{c \in C} c \cdot p_{r,c}$

$p_{r,0} = 1$ means lot r is predicted to be in class 0

Features & constraints

- Features

$$\begin{aligned} \text{StartPrice}_r &= s_r && \text{for } r \in I \\ \text{LotNrRel}_r &= \frac{\{i|x_{i,r}=1\}}{N} && \text{for } r \in I \\ \text{SPEV}_r &= \frac{s_r}{EV_r} && \text{for } r \in I \end{aligned}$$

- One item is sold at every index

$$\begin{aligned} \textstyle\sum_{1\leq r\leq N} x_{i,r} &= 1 && \text{for all } 1 \leq i \leq N \\ \textstyle\sum_{1\leq i\leq N} x_{i,r} &= 1 && \text{for all } 1 \leq r \leq N \end{aligned}$$

- Start prices are bounded using expert knowledge

$$0.4 \times EV_r \leq s_r \leq 1.0 \times EV_r, \text{ for all } r \in I$$

- Every lot r can only end up in one class

$$\textstyle\sum_{c\in C} p_{r,c} = 1 \text{ for all } r \in I$$

- We then construct the relations between (decision) variables and performance variable $p_{r,c}$

- We translate the classification tree model into ILP using linear constraints based on our previous work (Verwer, Zhang, Ye, 2017, AIJ)

Results --- example

SaleNr	LotNr	LotNr New	EstValue	SP	SP New	Main	Sub	yPred	yPred New	yClass
19604	1	31	11000	5500	4526	equipment	telescopic handlers	1	**2**	2
19604	2	27	5000	2000	2000	vehicles	vans	2	**2**	2
19604	3	19	1500	750	600	vehicles	cars	1	**2**	2
19604	4	17	2000	1000	800	equipment	sweeping machines	2	**2**	1
19604	5	38	2500	1250	1000	vehicles	trailers	1	**2**	1
19604	40	29	800	400	625	equipment	supplies	1	**2**	1
19604	44	7	200	100	156	equipment	supplies	1	**2**	2
19604	46	41	400	360	312	equipment	work shop equipment	1	**2**	2
19604	47	20	600	300	267	equipment	intern transport	1	**2**	1
19604	57	42	300	250	234	equipment	work shop equipment	1	**2**	2

Results --- general

Lots	Acc	% Class 2 (new design)	% Higher SPEV
47	43	96	62
36	53	81	72
6	50	100	33
71	62	100	85
56	41	100	68

1. low accuracy - due to predicting a new sale
2. optimization nearly perfect, *always optimal*
3. assigning higher starting prices than experts
4. ordering very different from expert assignment

Conclusion & Future work

- Model more advanced ML models (random forest, FNN) instead of single decision tree
- Make optimization robust, currently every prediction is a point estimate
- Apply methodology with more traditional combinatorial problems with uncertainties and unknown relations

Links to research topics: DT learning

Math. Opt. for DT

- Many papers, see survey, e.g. Carrizosa et al, 2021, Costa et al. 2023

Links to fairness

- Many surveys, e.g., Mehrabi et al., 2021
- Calders, T., and Verwer, S. (2010). Three naive bayes approaches for discrimination-free classification. *Data mining and knowledge discovery*.

- How DT contributes to Explainable and Responsible AI, see
  - XAI, see Rudin et al, 2022
  - Responsible AI, see Blockeel et al. 2023

Links to research lines: white-box optimization

- Empirical model learning
  - Survey paper (Lombardi & Milano, 2018)
- Constraint learning for optimization
  - Survey paper (Fajemisin et al. 2023)
- Surrogate modelling & blackbox optimization
  - Survey paper, e.g., Bliek 2022
- Decision-focused learning
  - Survey paper (Wilder et al. 2019)

References

- Verwer, S., Zhang, Y., and Ye, Q. C. (2017). Auction optimization using regression trees and linear models as integer programs. Artificial Intelligence.
- Verwer, S. and Zhang, Y. (2017). Learning decision trees with flexible constraints and objectives using integer optimization. CPAIOR
- Verwer, S. and Zhang, Y., (2019). Learning optimal classification trees using a binary linear program formulation. AAAI
- Ye, Q. C., Rhuggenaath, J., Zhang, Y., Verwer, S. E., & Hilgeman, M. J. (2021). Data driven design for online industrial auctions. Annals of Mathematics and Artificial Intelligence
- Firat, M., Crognier, G, Gabor, A., Hurkens C, and Zhang, Y. Computers & Operations Research, 2020
- Vos, D. and Verwer, S. (2022). Robust optimal classification trees against adversarial examples. AAAI

References

- Fajemisin, A., Maragno, D., & Hertog, D. D. (2023). Optimization with constraint learning: A framework and survey. EJOR
- M. Lombardi and M. Milano (2018). Boosting Combinatorial Problem Modeling with Machine Learning. IJCAI
- Wilder, B., Dilkina, B., & Tambe, M. (2019). Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. AAAI
- Carrizosa, E., Molero-Río, C. and Romero Morales, D., 2021. Mathematical optimization in classification and regression trees. Top
- Rudin, C., Chen, C., Chen, Z., Huang, H., Semenova, L. and Zhong, C., 2022. Interpretable machine learning: Fundamental principles and 10 grand challenges. Statistic Surveys
- Blockeel, H., Devos, L., Frénay, B., Nanfack, G. and Nijssen, S., 2023. Decision trees: from efficient prediction to responsible AI. Frontiers in Artificial Intelligence
- Costa, V.G. and Pedreira, C.E., 2023. Recent advances in decision trees: An updated survey. Artificial Intelligence Review, 56(5), pp.4765-4800.
- Bliek, L., 2022. A survey on sustainable surrogate-based optimisation. *Sustainability*

News! Job Shop Scheduling Benchmark

A open sourced repo for benchmarking scheduling solutions

https://github.com/ai-for-decision-making-tue/Job_Shop_Scheduling_Benchmark_Environments_and_Instances

- Benchmark instances of variants of JSP problems + various solution methods (DRL, GA, hybrid)
- Environment of developing other (learning based) solution methods

Solution methods	Job Shop	Flow Show	Flexible Job Shop	SDST	Assembly	Online Arrivals
Load Balancing Heuristics	✓	✓	✓	✓	✓	
Dispatching Rules	✓	✓	✓		✓	✓*
Genetic Algorithm	✓	✓	✓	✓	✓	
FJSP-DRL	✓	✓	✓			
Hybrid (GA+DRL) – coming soon						

Reijnen, R., van Straaten, K., Bukhsh, Z. and Zhang, Y., 2023. Job Shop Scheduling Benchmark: Environments and Instances for Learning and Non-learning Methods. arXiv