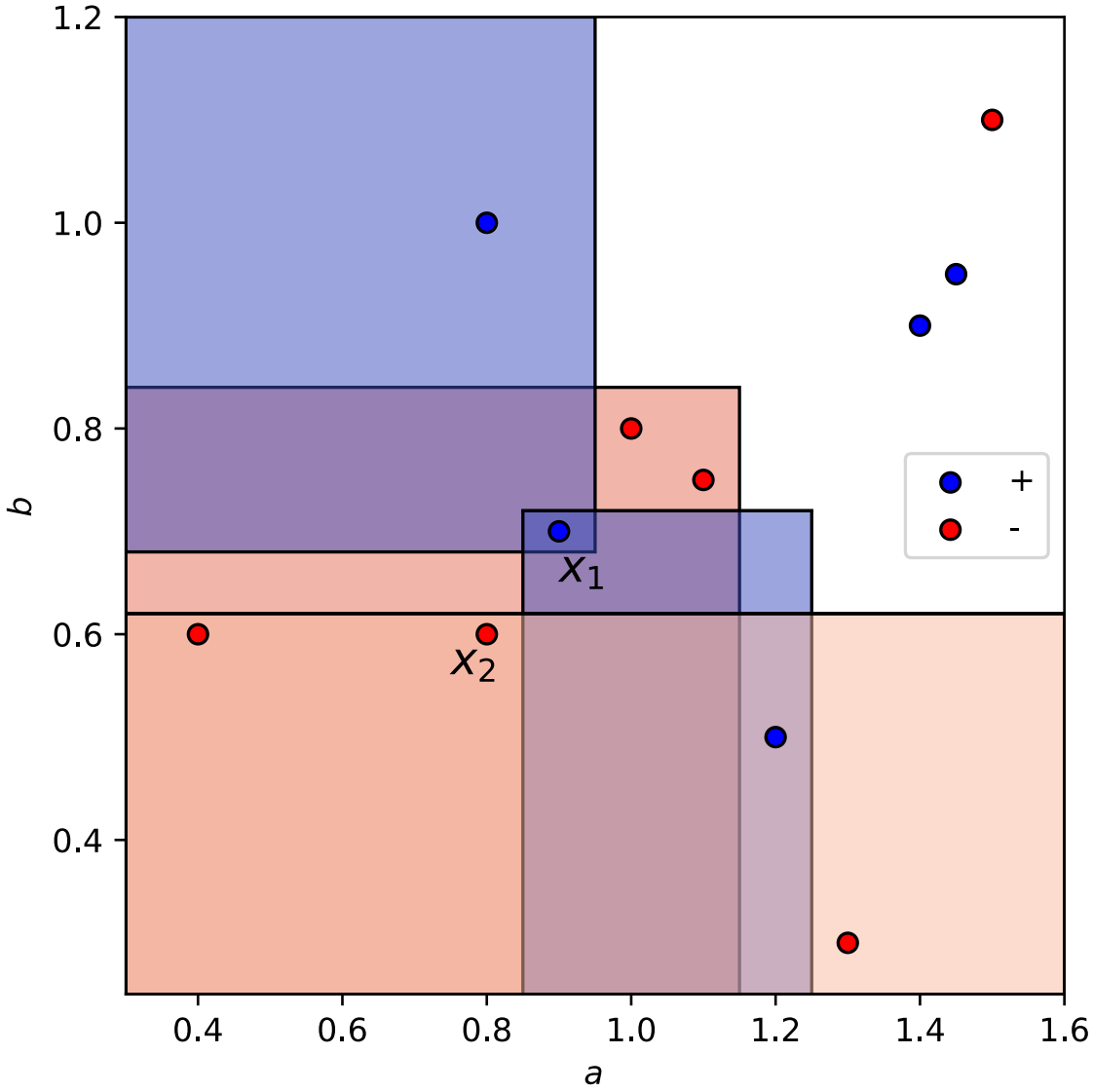
MONASH University

Better Short than Greedy: Interpretable Models through Optimal Rule Boosting

SIAM INTERNATIONAL CONFERENCE ON DATA MINING (SDM21)

Mario Boley, Simon Teshuva, Pierre Le Bodic, Geoff Webb
mario.boley@monash.edu

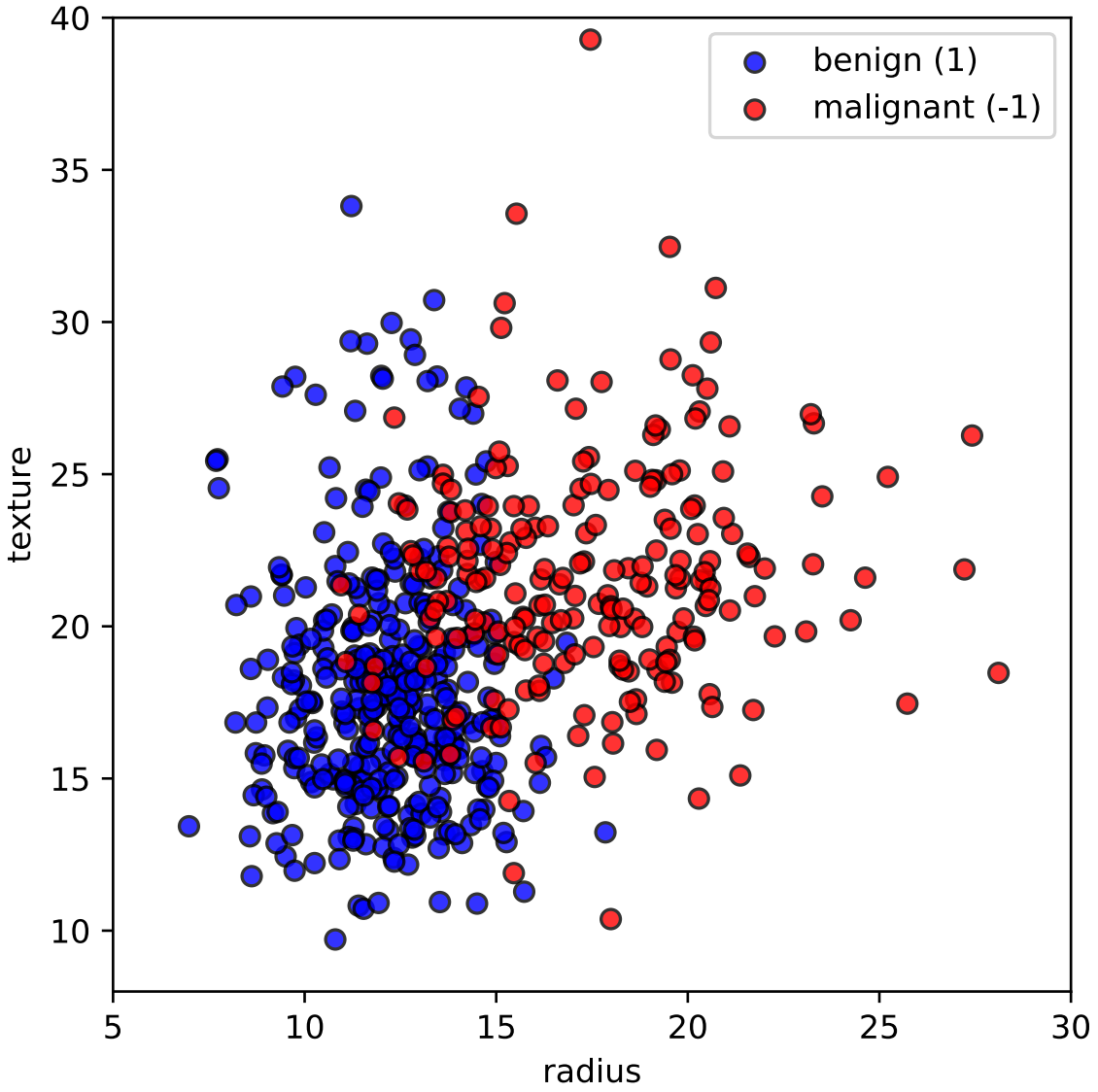
Department for Data Science and AI
Monash University, Australia



Additive rule ensembles – an interpretable model

Example (breast cancer):

```
+1.61 if radius<=14

-1.89 if radius>=15.5 & texture>=16.4

+1.00 if radius<=17.1 & texture<=18.6

-1.26 if radius>=12.3 & texture<=26.6 &
         texture>=20
```



[e.g., Cohen + Singer, 1999; Friedman + Popescu, 2008; Dembczynski et al., 2010; Lakkaraju et al. 2016]

Additive rule ensembles – an interpretable model

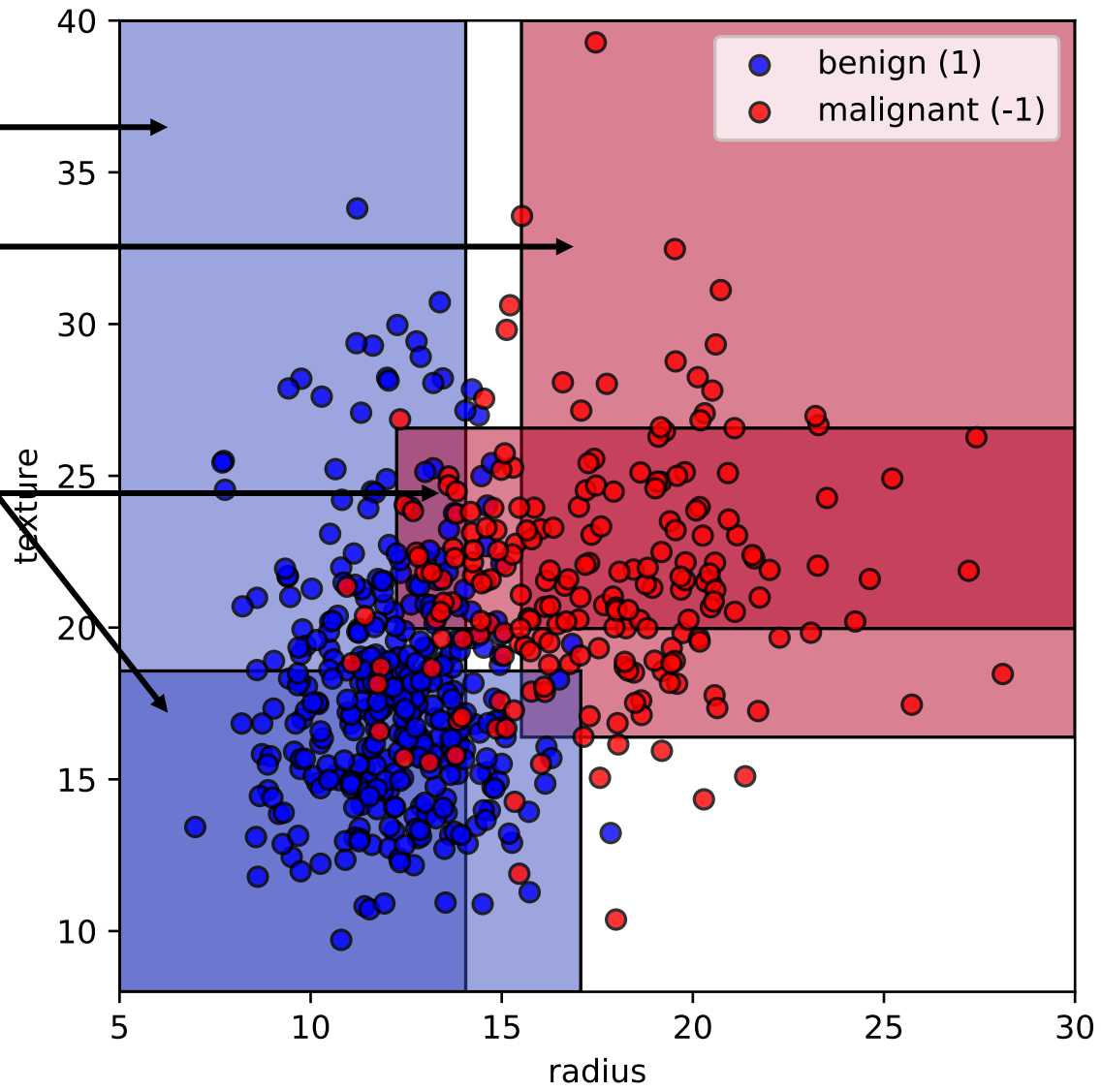
Example (breast cancer):

```
+1.61 if radius<=14

-1.89 if radius>=15.5 & texture>=16.4

+1.00 if radius<=17.1 & texture<=18.6

-1.26 if radius>=12.3 & texture<=26.6 &
         texture>=20
```

Formally: $f(x) = \sum_{i=1}^{k} w_i q_i(x)$

weights (consequents)

queries (antecedents)



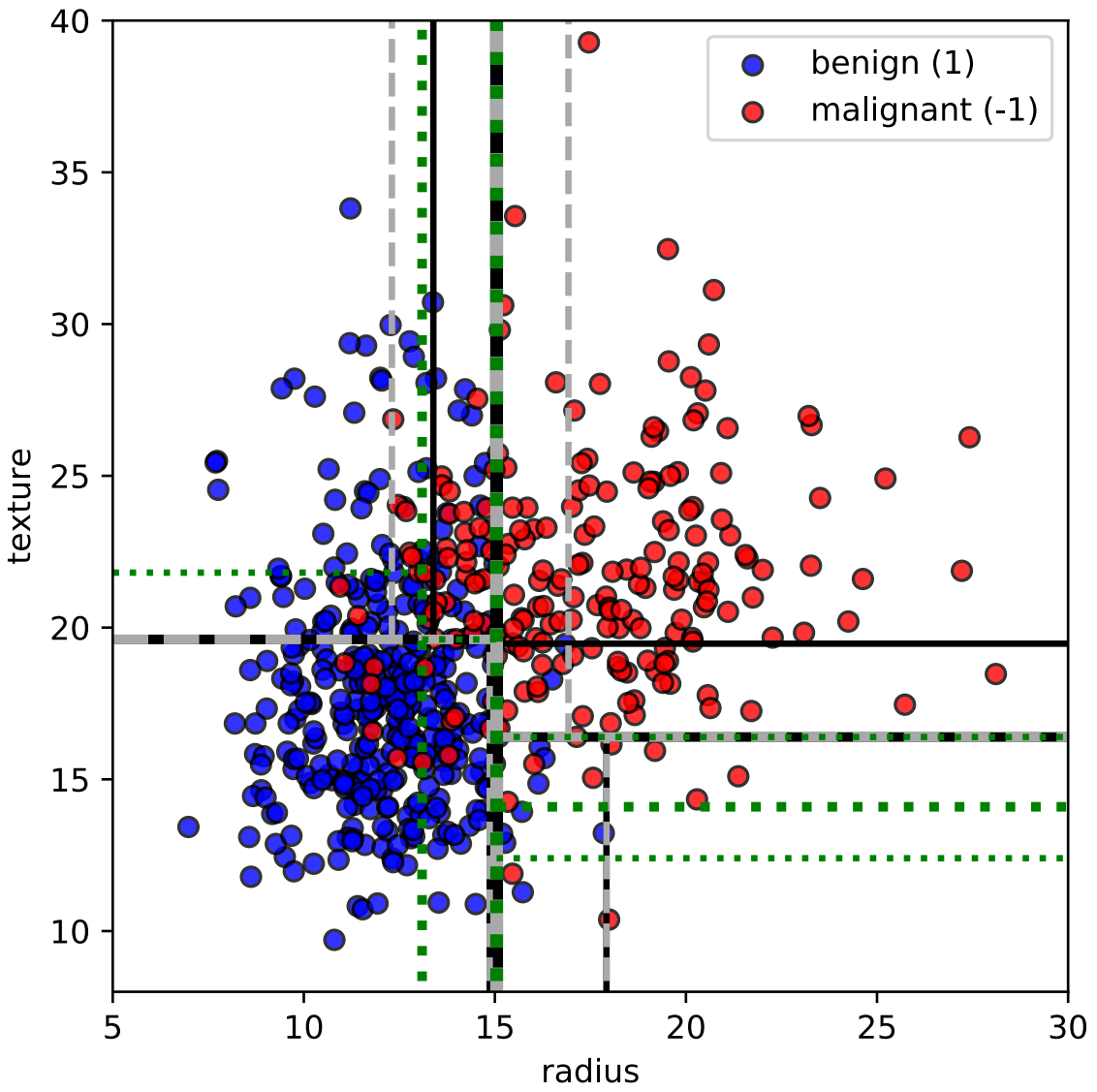
[e.g., Cohen + Singer, 1999; Friedman + Popescu, 2008; Dembczynski et al., 2010; Lakkaraju et al. 2016]

Problems of existing fitting methods

Pool-and-select methods

1. Extract candidate queries
   - tree ensembles (RuleFit)
   - frequent patterns
   - pattern sampling
2. Select subset
   - $\sum_{i=1}^{n} l\left(y_i, w^T q(x_i)\right) + \lambda \|w\|_1$ (RuleFit)
   - sub-modular optimisation
   - randomised search

Problem: pool tends to not contain optimal building blocks (or too large for effective optimisation)



Problems of existing fitting methods

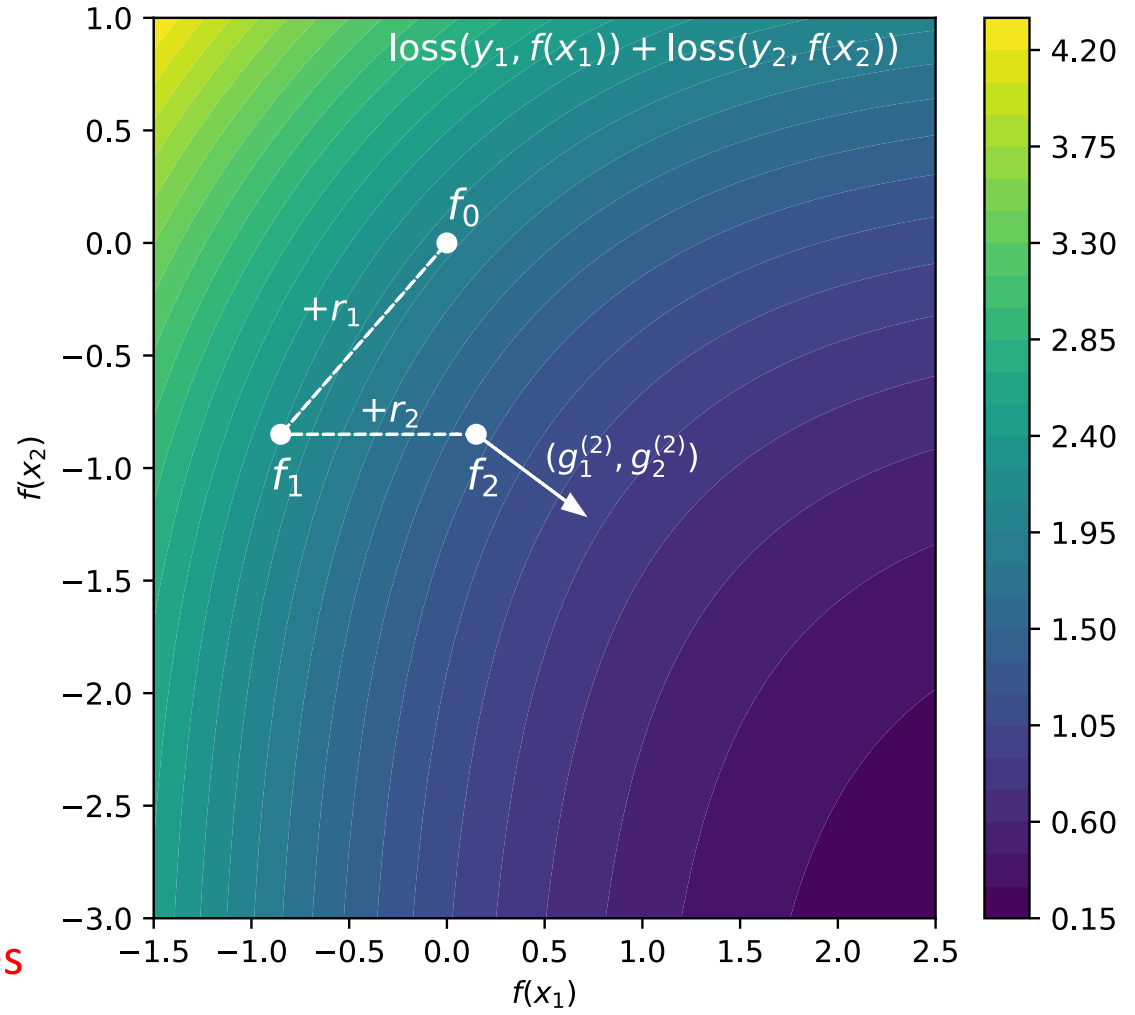
Pool-and-select methods

1. Extract candidate queries
   - tree ensembles (RuleFit)
   - frequent patterns
   - pattern sampling
2. Select subset
   - $\sum_{i=1}^{n} l\left(y_i, w^T q(x_i)\right) + \lambda \|w\|_1$ (RuleFit)
   - sub-modular optimisation
   - randomised search

Rule gradient boosting

1. Start with empty model $f_0(x) = 0$
2. Iteratively find $f_t(x) = f_{t-1}(x) + w_t q_t(x)$ by search in output space

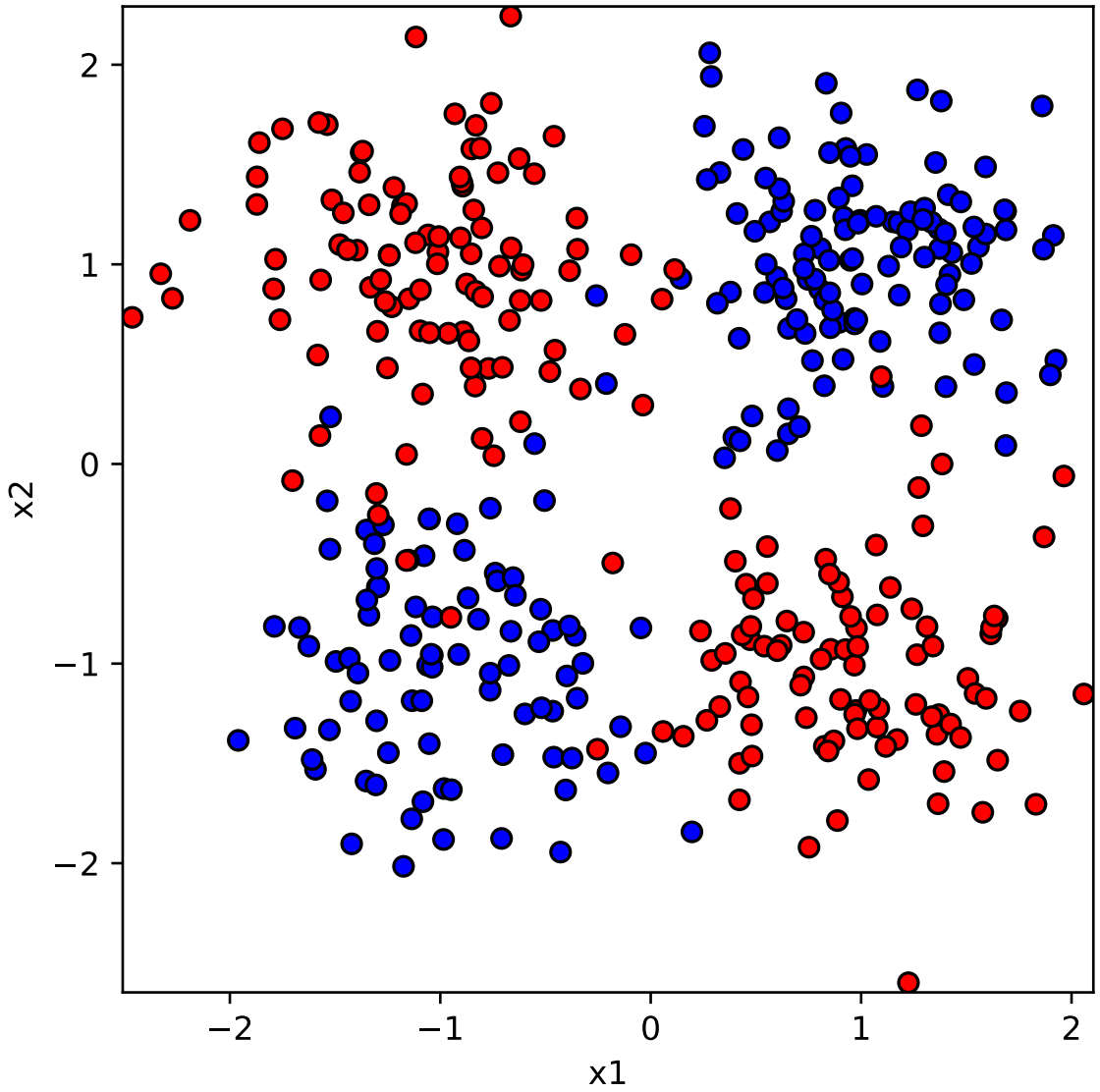
Problem: greedy (or beam) search for individual rules
→ "wastes" space in rule ensemble for sub-optimal rules
→ contradicts goal of comprehensibility



Heuristic search detrimental to comprehensibility

Example: “noisy parity”

$$C \sim \mathrm{Unif}\left(\{-1,1\}^d\right)$$
$$X|C \sim \mathrm{Norm}(C, \sigma^2 I)$$
$$Y|C = \prod_{i=1}^{d} C_i$$



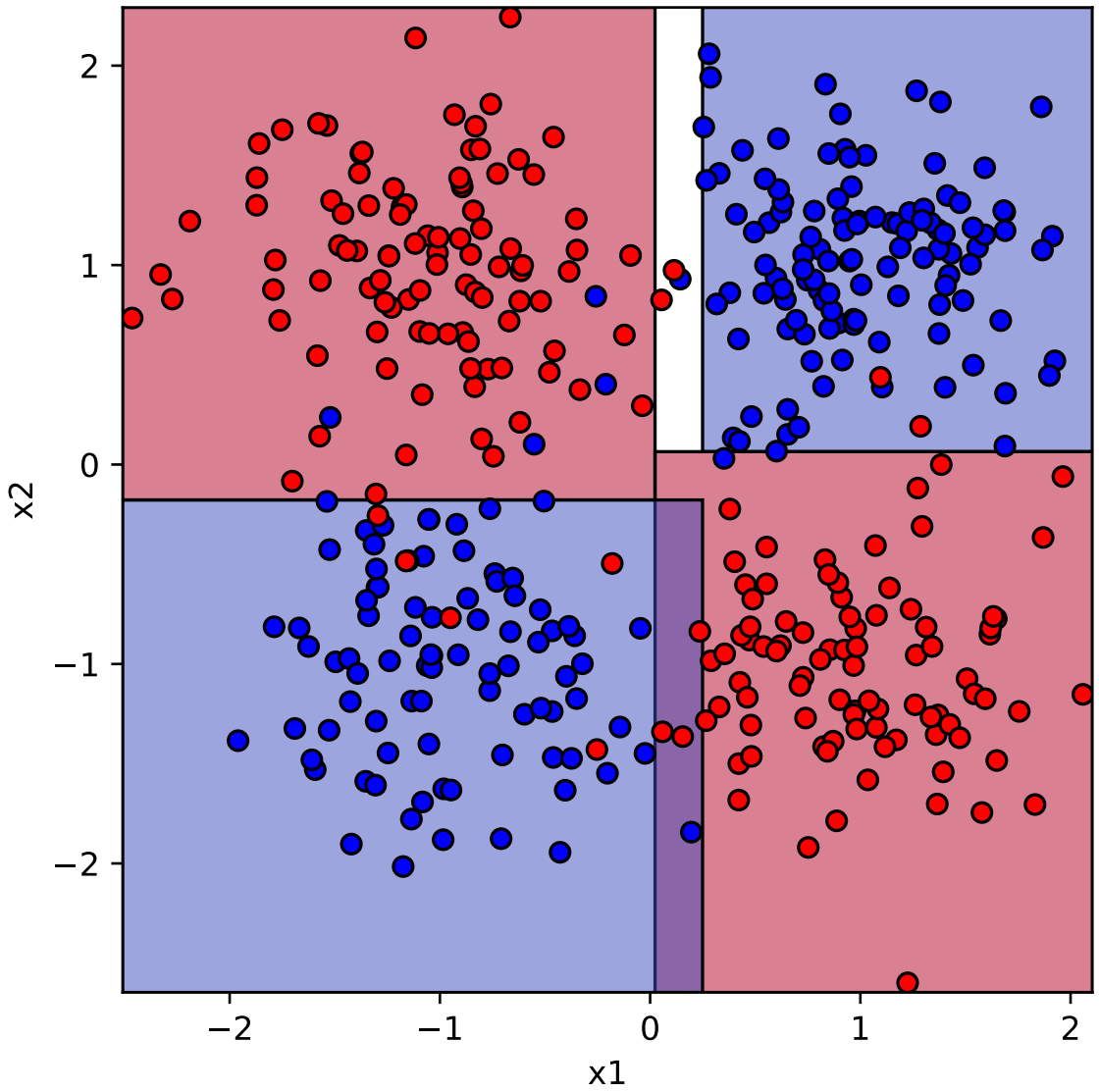
Heuristic search detrimental to comprehensibility

Example: "noisy parity"

$$C \sim \mathrm{Unif}(\{-1,1\}^d)$$
$$X|C \sim \mathrm{Norm}(C, \sigma^2 I)$$
$$Y|C = \prod_{i=1}^{d} C_i$$

Optimal ensemble (…of $\mathrm{k} = 2^d$ rules)

$$\sum_{N \subseteq [d]} (-1)^{|N|} \prod_{i \in N} \delta(X_i \leq 0) \prod_{i \in [d] \setminus \mathrm{N}} \delta(X_i > 0)$$



Mario Boley, Simon Teshuva, Pierre Le Bodic, Geoff Webb, SDM 2021

Heuristic search detrimental to comprehensibility

Example: "noisy parity"

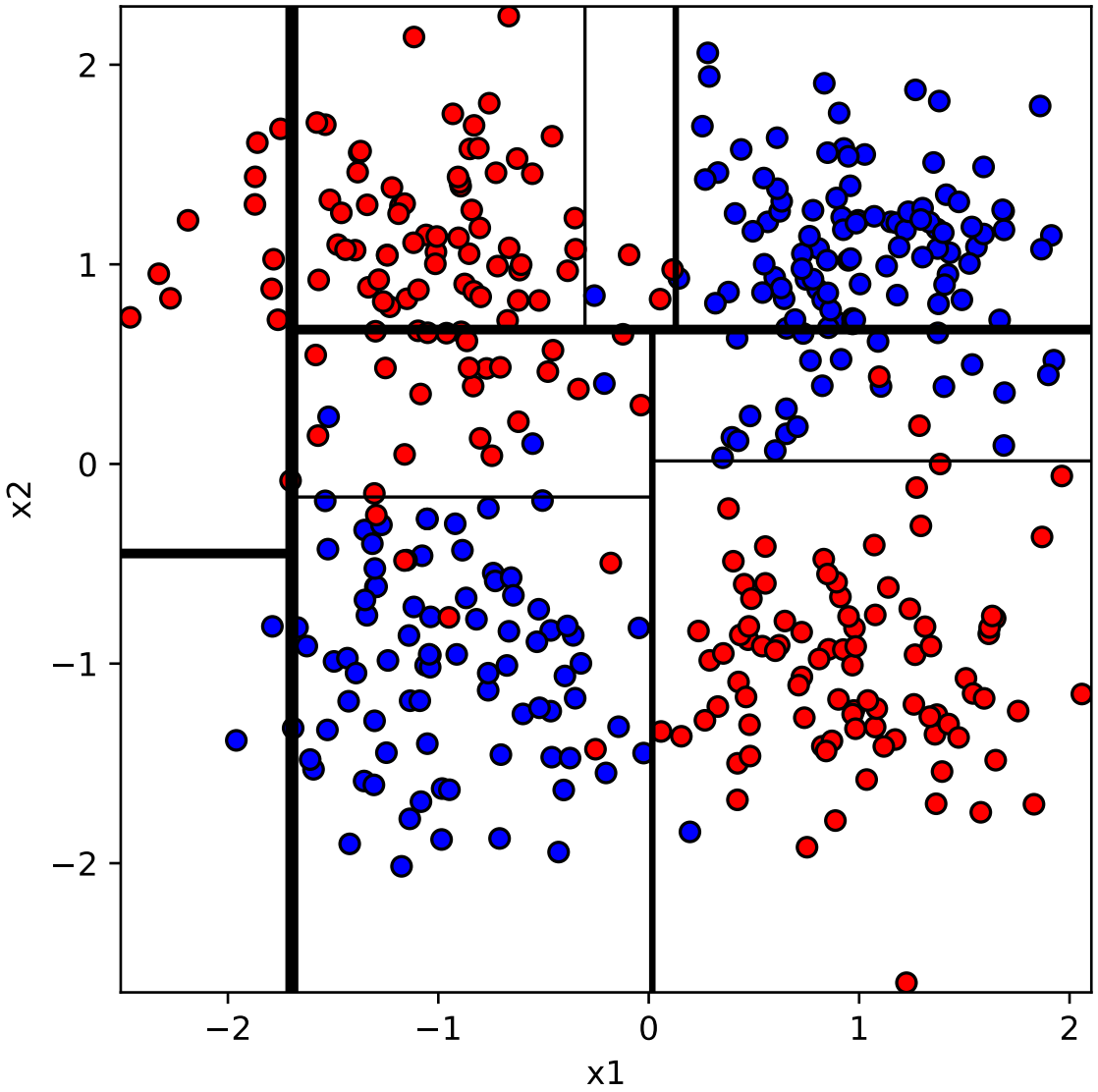
$$C \sim \mathrm{Unif}(\{-1,1\}^d)$$
$$X|C \sim \mathrm{Norm}(C, \sigma^2 I)$$
$$Y|C = \prod_{i=1}^{d} C_i$$

Optimal ensemble (…of $\mathrm{k} = 2^d$ rules)

$$\sum_{N \subseteq [d]} (-1)^{|N|} \prod_{i \in N} \delta(X_i \leq 0) \prod_{i \in [d] \setminus \mathrm{N}} \delta(X_i > 0)$$

RuleFit

Greedy decision trees provides bad building blocks



Heuristic search detrimental to comprehensibility

Example: "noisy parity"

$$C \sim \mathrm{Unif}(\{-1,1\}^d)$$
$$X|C \sim \mathrm{Norm}(C, \sigma^2 I)$$
$$Y|C = \prod_{i=1}^{d} C_i$$

Optimal ensemble (…of $\mathrm{k} = 2^d$ rules)

$$\sum_{N \subseteq [d]} (-1)^{|N|} \prod_{i \in N} \delta(X_i \leq 0) \prod_{i \in [d] \setminus \mathrm{N}} \delta(X_i > 0)$$

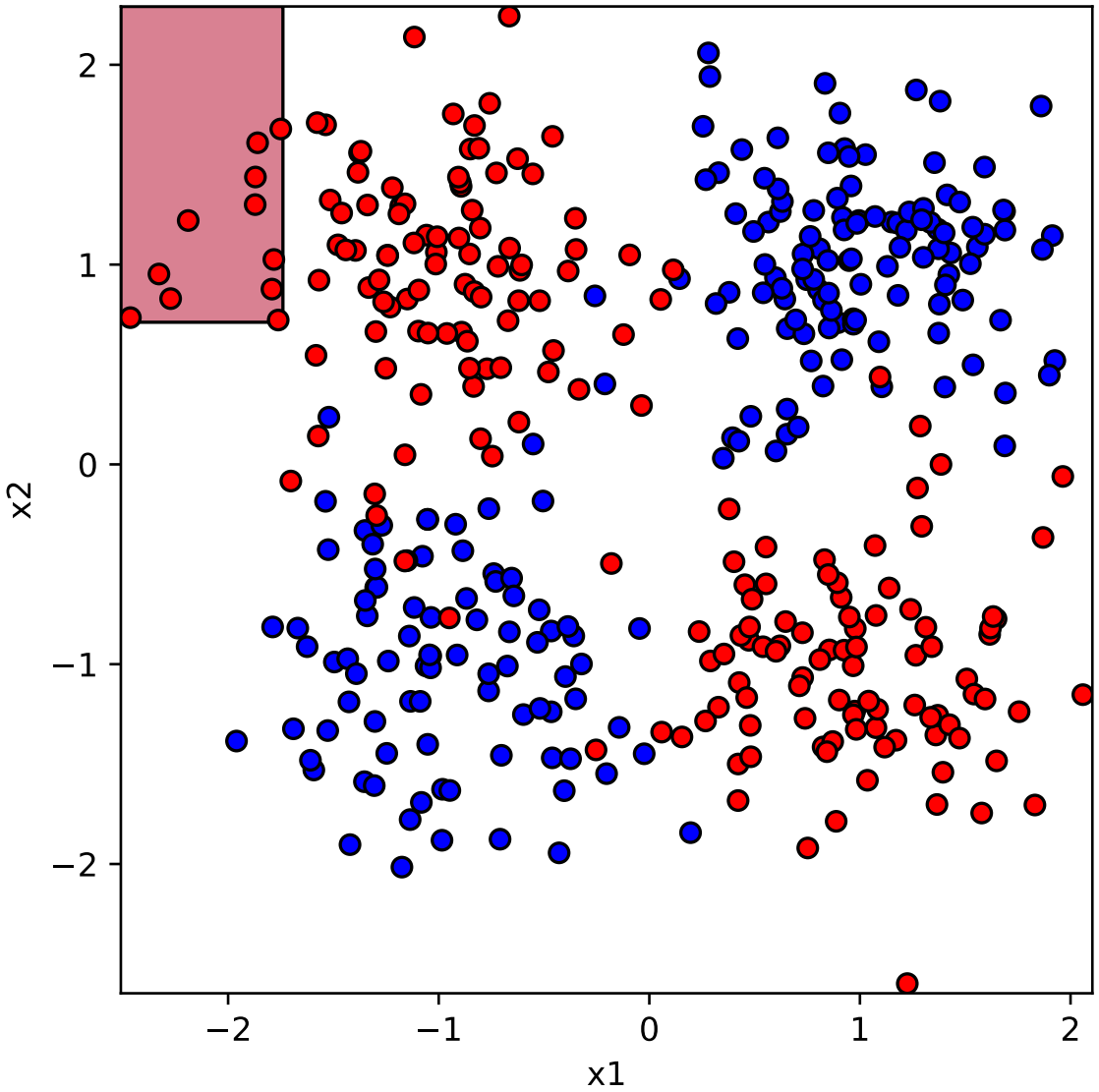
RuleFit

Greedy decision trees provides bad building blocks

Greedy rule boosting

Catches up earlier…

…but also needs extra rules to reach optimal accuracy



Heuristic search detrimental to comprehensibility

Example: "noisy parity"

$$C \sim \text{Unif}(\{-1,1\}^d)$$
$$X|C \sim \text{Norm}(C, \sigma^2 I)$$
$$Y|C = \prod_{i=1}^{d} C_i$$

Optimal ensemble (…of $\text{k} = 2^d$ rules)

$$\sum_{N \subseteq [d]} (-1)^{|N|} \prod_{i \in N} \delta(X_i \leq 0) \prod_{i \in [d] \setminus \text{N}} \delta(X_i > 0)$$

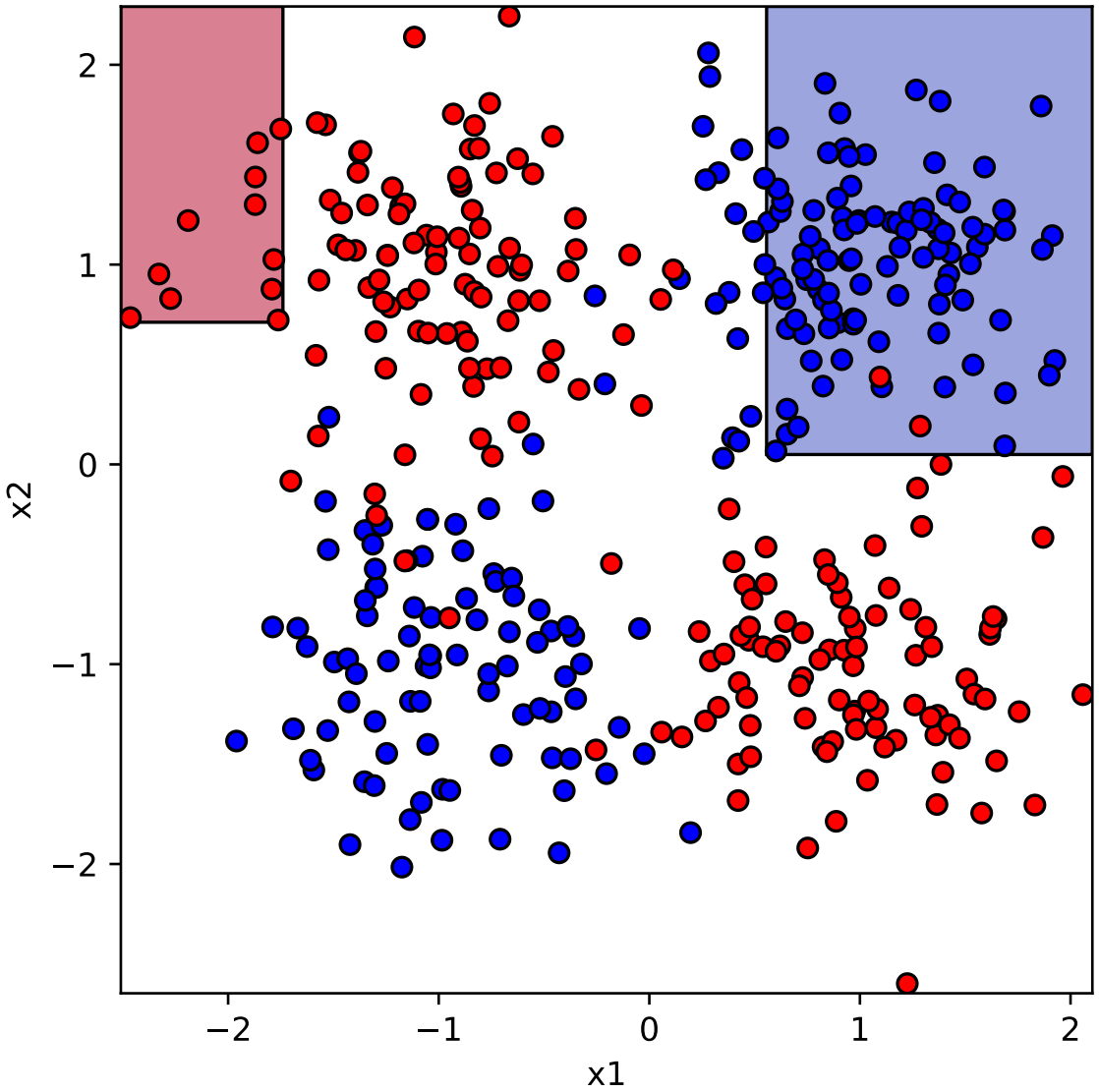
RuleFit

Greedy decision trees provides bad building blocks

Greedy rule boosting

Catches up earlier…

…but also needs extra rules to reach optimal accuracy



Heuristic search detrimental to comprehensibility

Example: "noisy parity"

$$C \sim \text{Unif}(\{-1,1\}^d)$$
$$X|C \sim \text{Norm}(C, \sigma^2 I)$$
$$Y|C = \prod_{i=1}^{d} C_i$$

Optimal ensemble (…of $\text{k} = 2^d$ rules)

$$\sum_{N \subseteq [d]} (-1)^{|N|} \prod_{i \in N} \delta(X_i \leq 0) \prod_{i \in [d] \setminus \text{N}} \delta(X_i > 0)$$

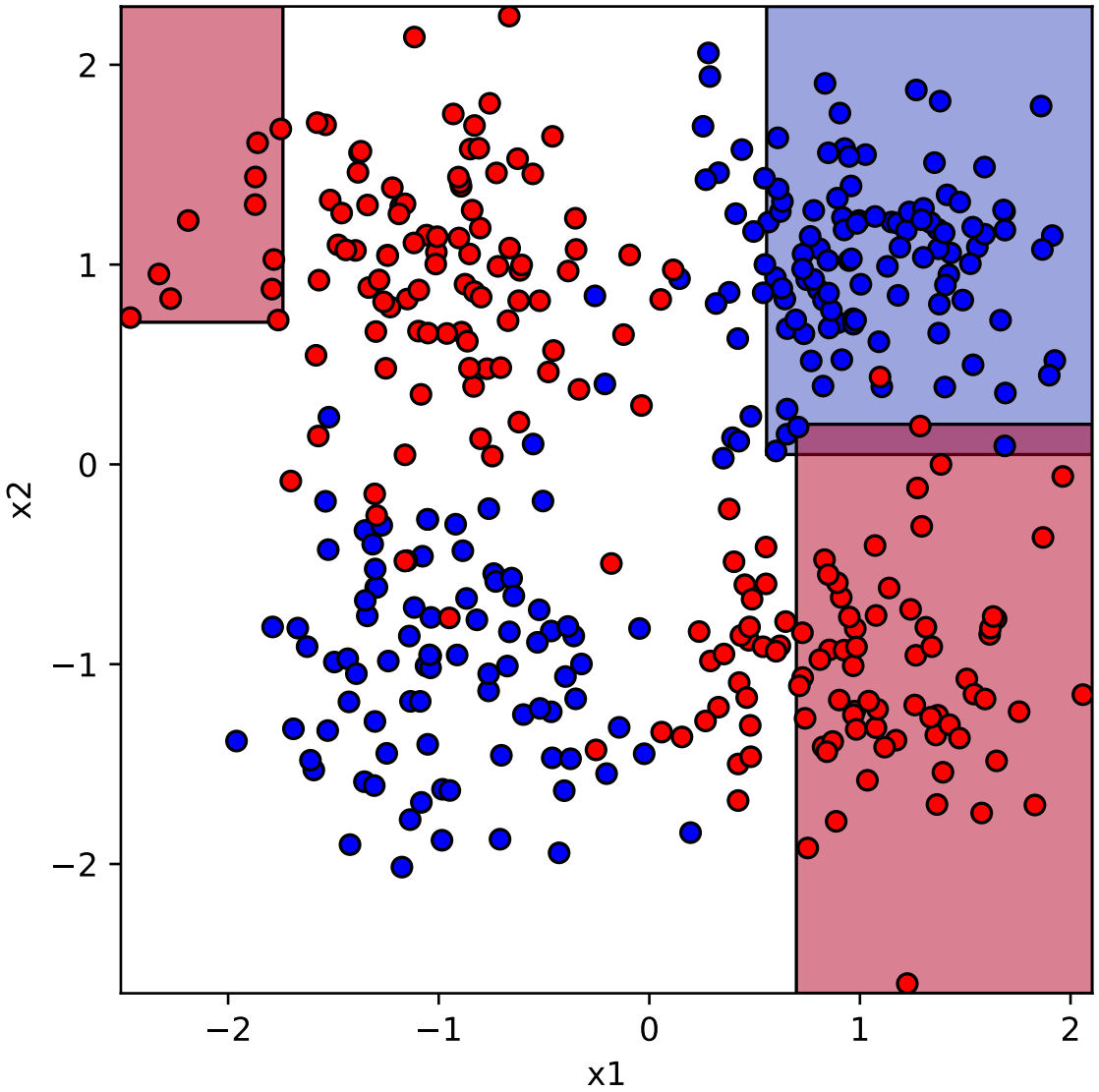
RuleFit

Greedy decision trees provides bad building blocks

Greedy rule boosting

Catches up earlier…

…but also needs extra rules to reach optimal accuracy



Mario Boley, Simon Teshuva, Pierre Le Bodic, Geoff Webb, SDM 2021

Heuristic search detrimental to comprehensibility

Example: "noisy parity"

$$C \sim \text{Unif}(\{-1,1\}^d)$$
$$X|C \sim \text{Norm}(C, \sigma^2 I)$$
$$Y|C = \prod_{i=1}^{d} C_i$$

Optimal ensemble (…of $\text{k} = 2^d$ rules)

$$\sum_{N \subseteq [d]} (-1)^{|N|} \prod_{i \in N} \delta(X_i \leq 0) \prod_{i \in [d] \setminus \text{N}} \delta(X_i > 0)$$

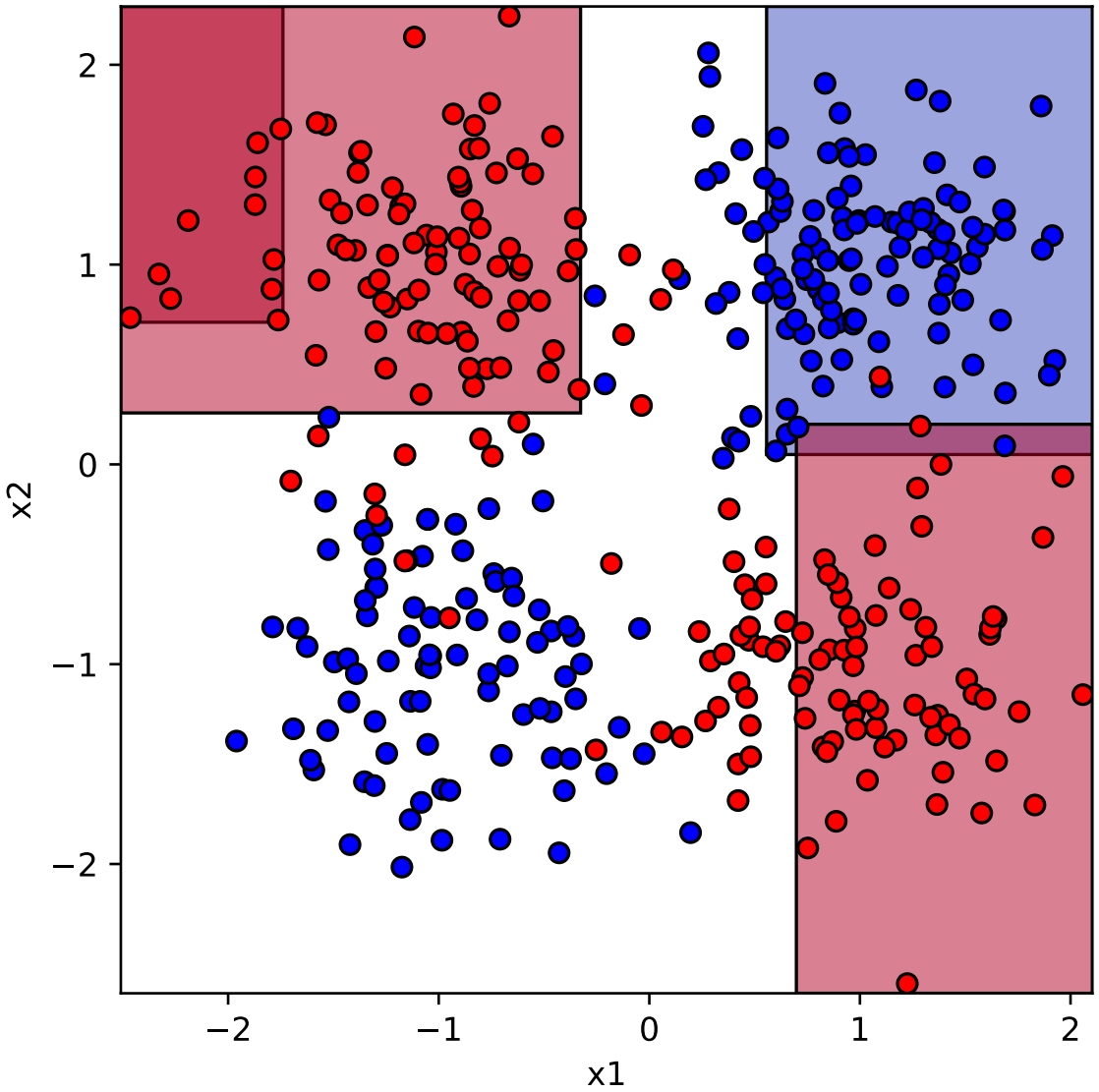
RuleFit

Greedy decision trees provides bad building blocks

Greedy rule boosting

Catches up earlier…

…but also needs extra rules to reach optimal accuracy



Heuristic search detrimental to comprehensibility

Example: “noisy parity”

$$C \sim \text{Unif}(\{-1,1\}^d)$$
$$X|C \sim \text{Norm}(C, \sigma^2 I)$$
$$Y|C = \prod_{i=1}^{d} C_i$$

Optimal ensemble (…of $\text{k} = 2^d$ rules)

$$\sum_{N \subseteq [d]} (-1)^{|N|} \prod_{i \in N} \delta(X_i \leq 0) \prod_{i \in [d] \setminus \text{N}} \delta(X_i > 0)$$

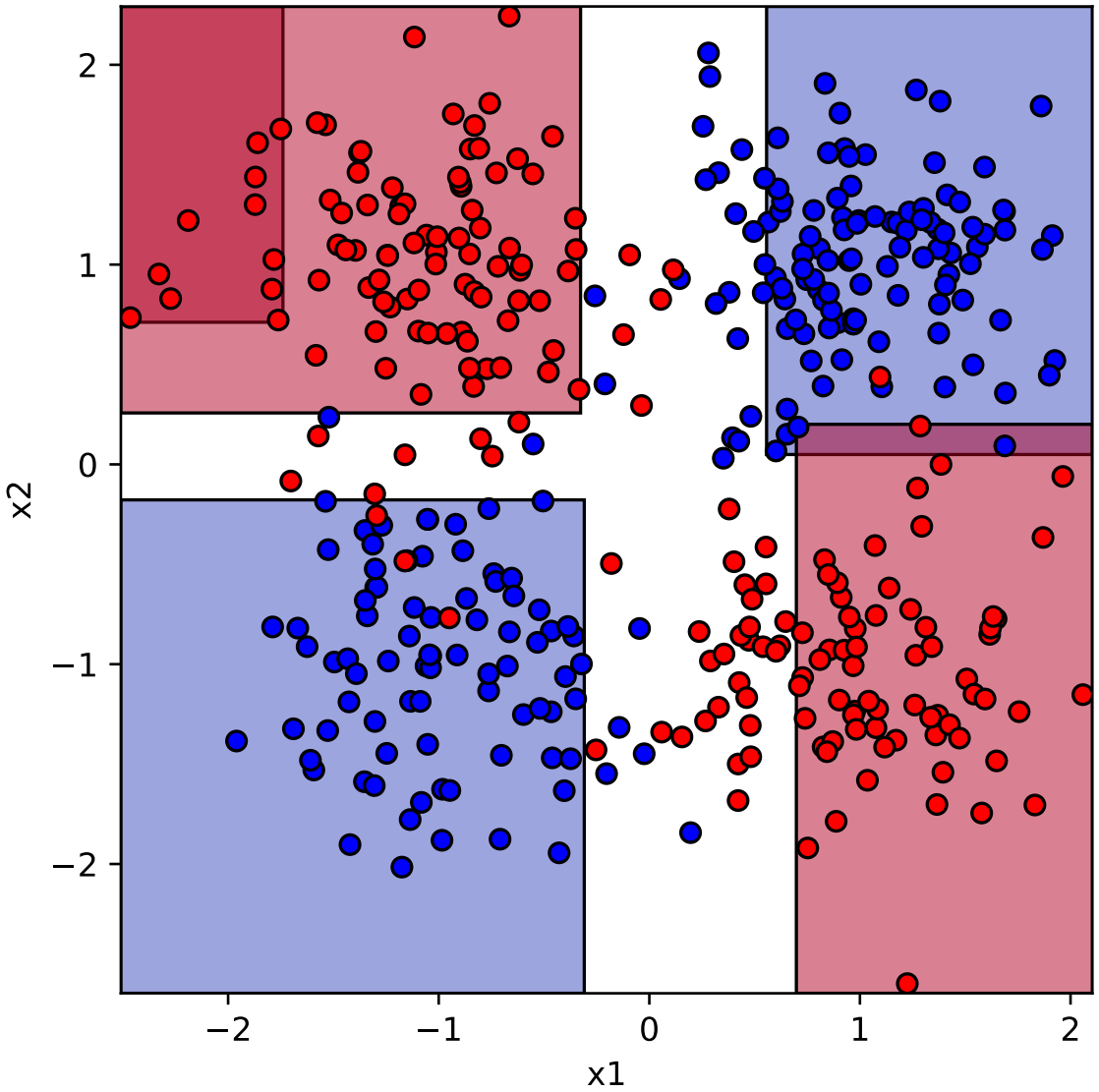
RuleFit

Greedy decision trees provides bad building blocks

Greedy rule boosting

Catches up earlier…

…but also needs extra rules to reach optimal accuracy



Heuristic search detrimental to comprehensibility

Example: “noisy parity”

$$C \sim \text{Unif}(\{-1,1\}^d)$$
$$X|C \sim \text{Norm}(C, \sigma^2 I)$$
$$Y|C = \prod_{i=1}^{d} C_i$$

Optimal ensemble (…of $\text{k} = 2^d$ rules)

$$\sum_{N \subseteq [d]} (-1)^{|N|} \prod_{i \in N} \delta(X_i \leq 0) \prod_{i \in [d] \setminus \text{N}} \delta(X_i > 0)$$

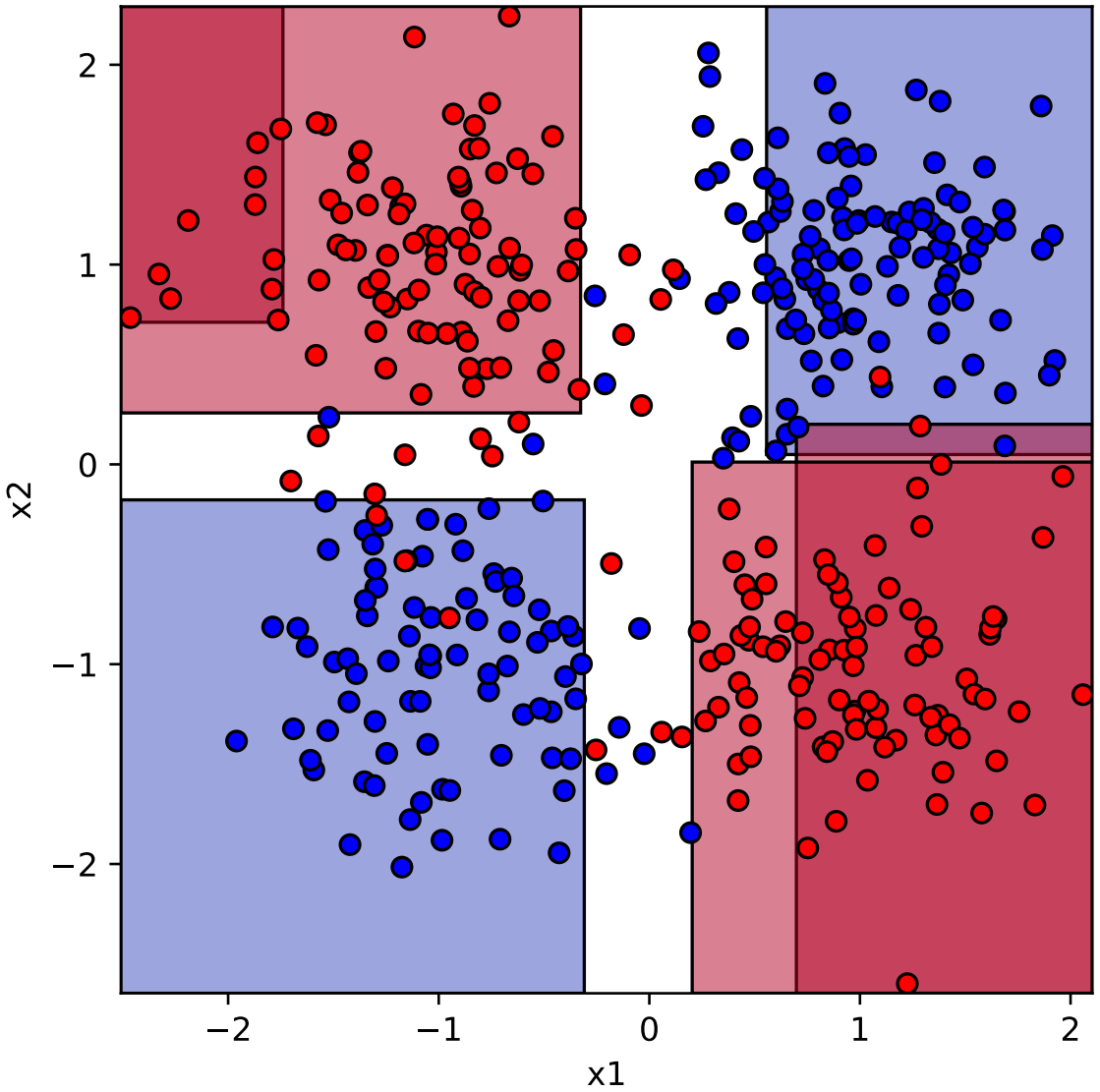
RuleFit

Greedy decision trees provides bad building blocks

Greedy rule boosting

Catches up earlier…

…but also needs extra rules to reach optimal accuracy



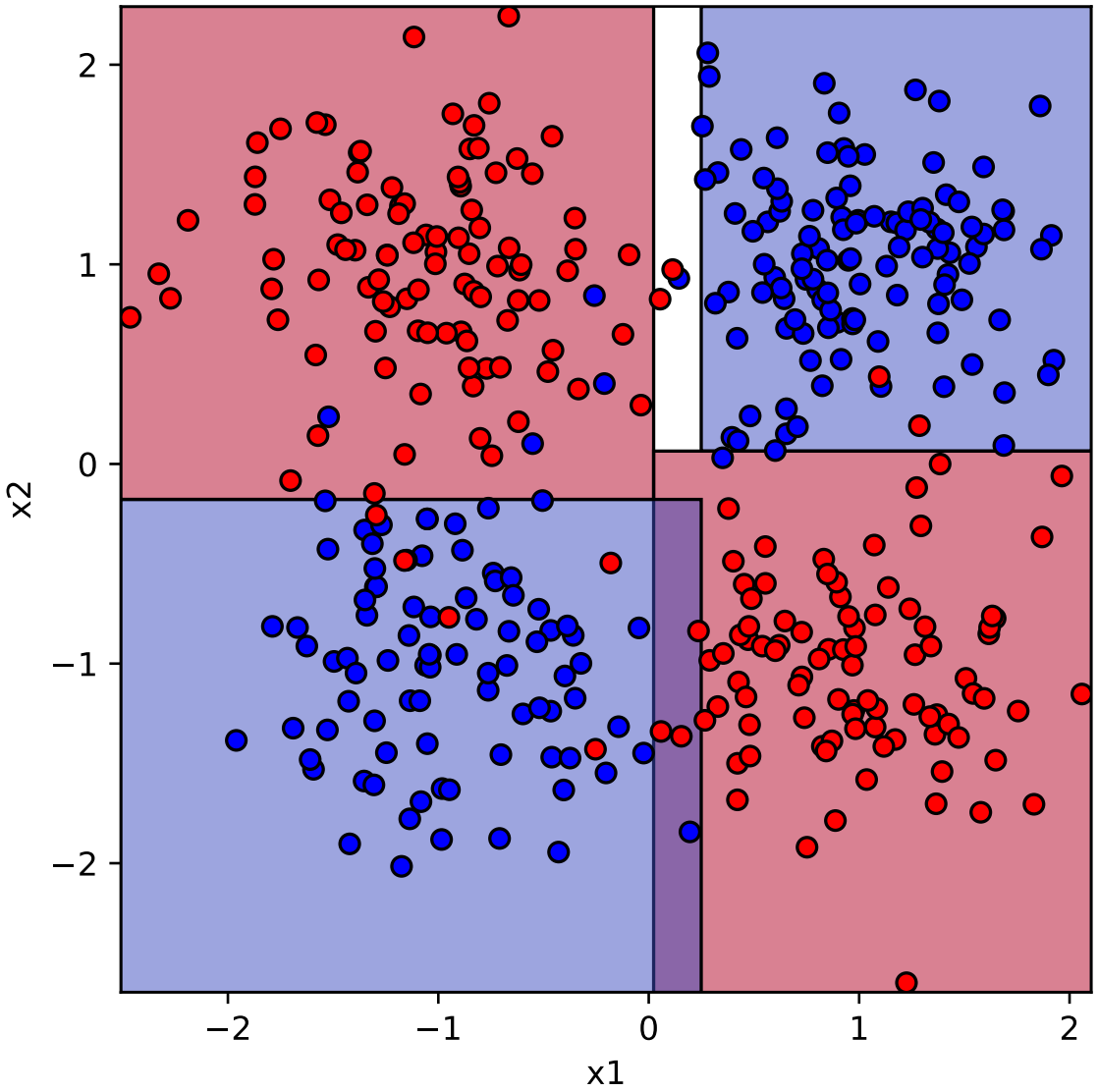
This work: *optimal* rule boosting

Contribution

Efficient optimal base learner for “second order” gradient rule boosting

In particular

1. Theorem: effective linear time upper bound of boosting objective function
2. New formulation of query equivalence-class search for OPUS-style branch-and-bound (paper)
3. Demonstration: optimal rule boosting consistently outperforms RuleFit and greedy rule boosting at bearable computational cost



This work: *optimal* rule boosting

Contribution

Efficient optimal base learner for “second order” gradient rule boosting

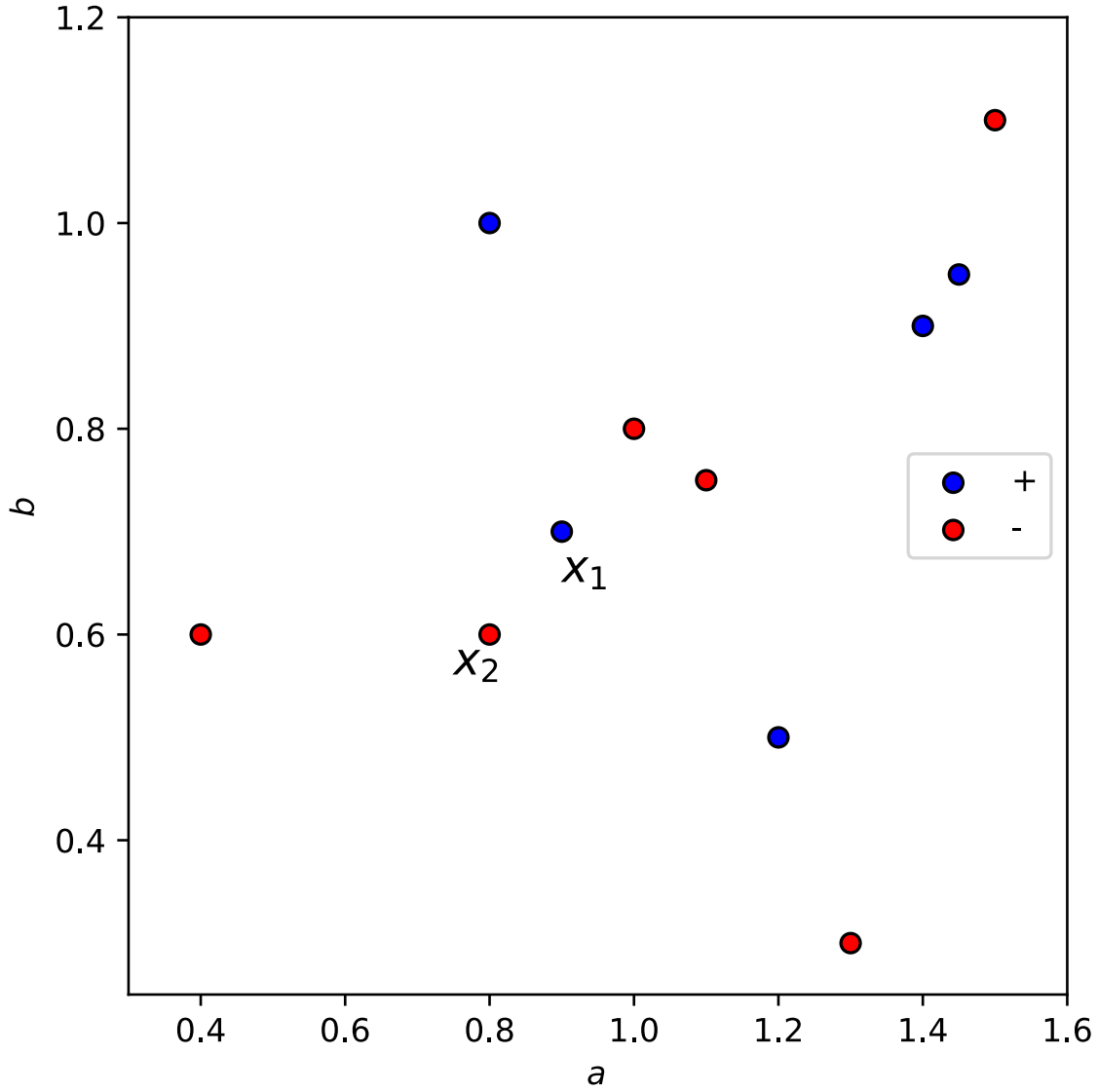
In particular

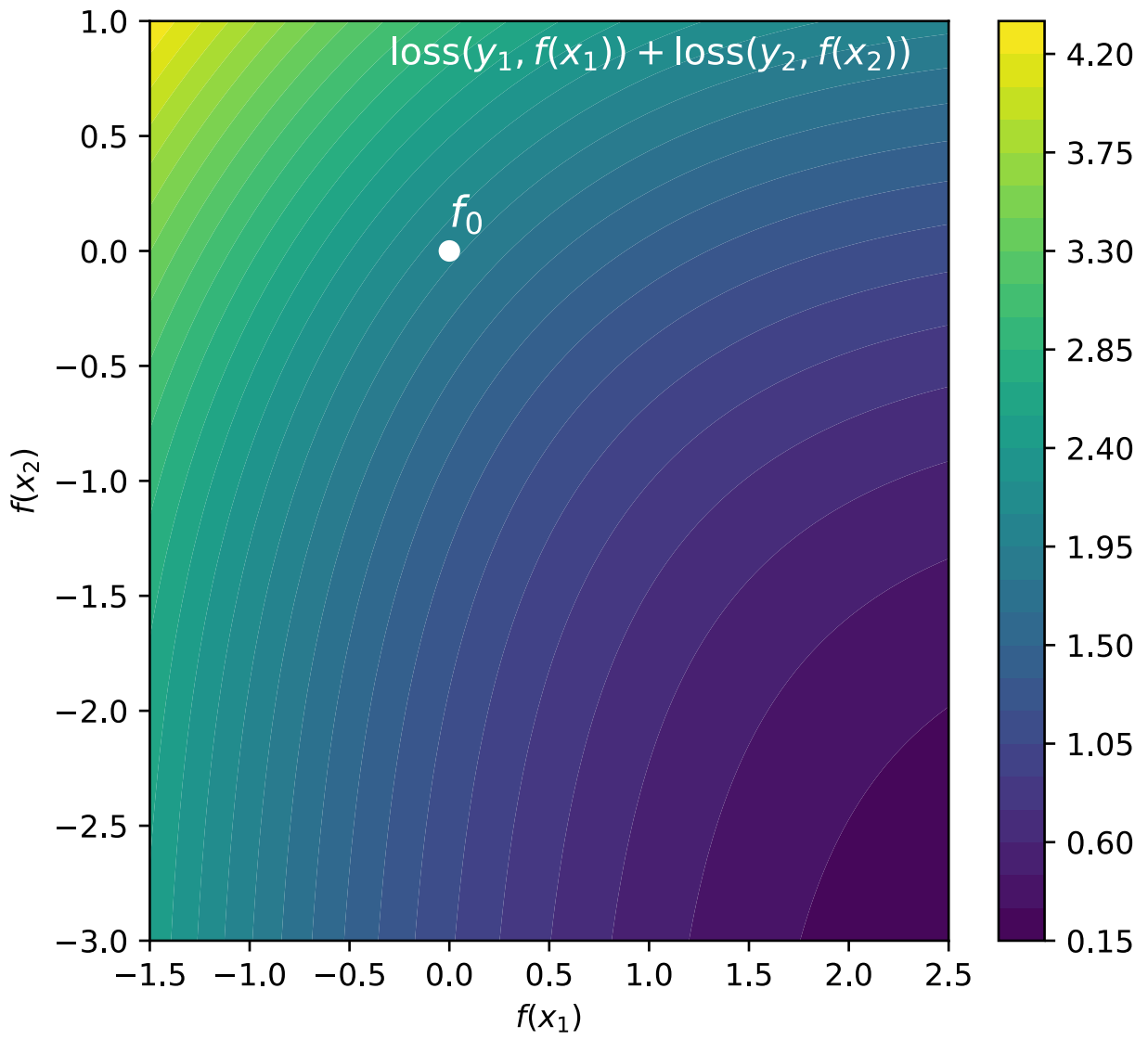
1. Theorem: effective linear time upper bound of boosting objective function
2. New formulation of query equivalence-class search for OPUS-style branch-and-bound (paper)
3. Demonstration: optimal rule boosting consistently outperforms RuleFit and greedy rule boosting at bearable computational cost
4. Open source Python implementation

   https://github.com/marioboley/realkd.py

```
$ pip3 install realkd
$ python3

Python 3.7.7 (default, Mar 10 2020, 15:43:03)
>>> from realkd.datasets import noisy_parity
>>> x, y = noisy_parity(n=400, d=2)
>>>
>>> from realkd.rules import XGBRuleEstimator
>>> bl = XGBRuleEstimator(loss='logistic', reg=1)
>>>
>>> from realkd.rules import RuleBoostingEstimator
>>> rules = RuleBoostingEstimator(4, bl))
>>>
>>> rules.fit(x, y)
>>> rules.rules_
   +1.8655 if x1>=0.25 & x2>=0.06
   -1.7714 if x1<=0.02 & x2>=-0.18
   -1.8316 if x1>=0.02 & x2<=0.06
   +1.6870 if x1<=0.25 & x2<=-0.18
>>>
```

Gradient boosting fits stage-wise in *output* Space

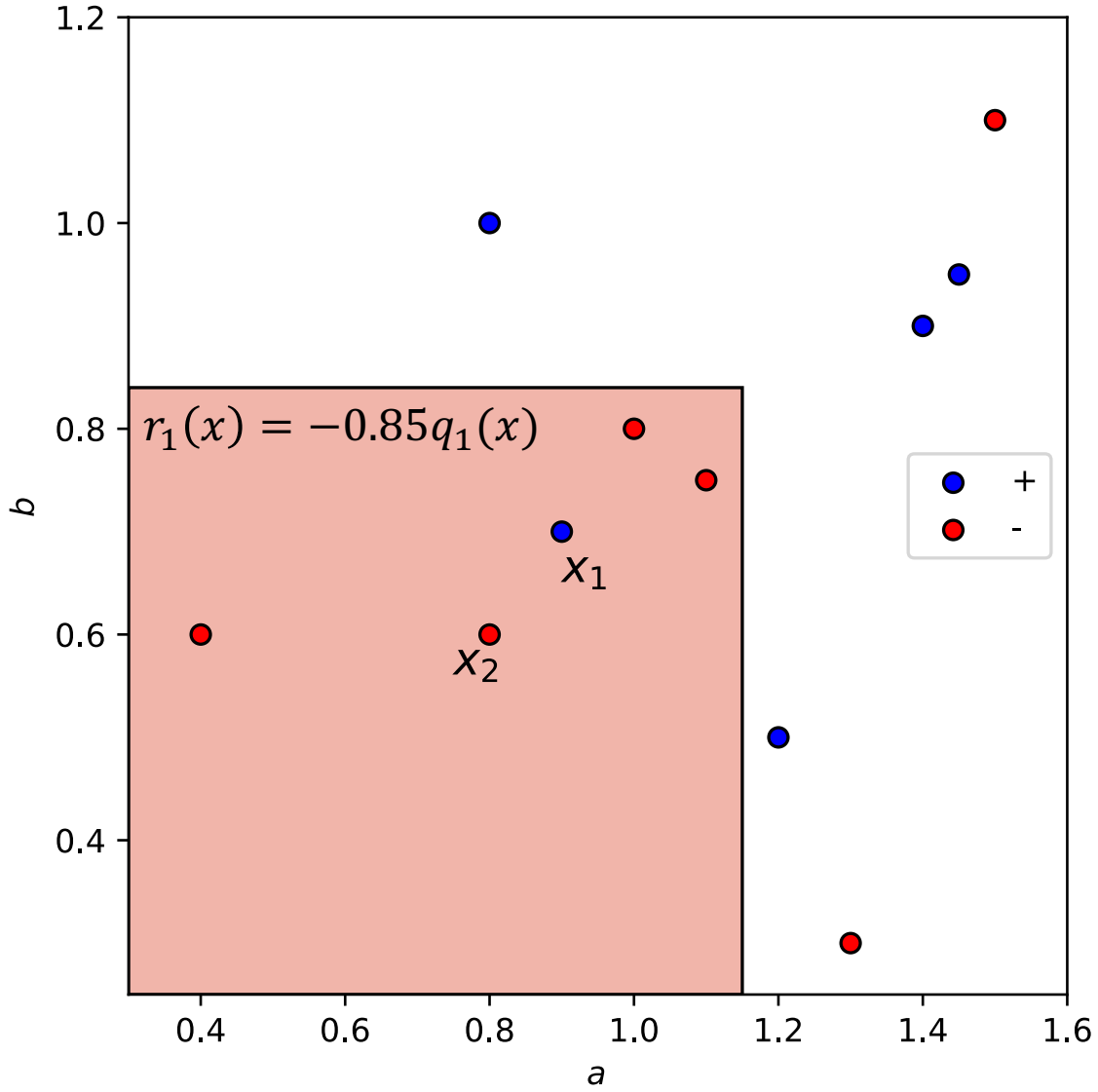


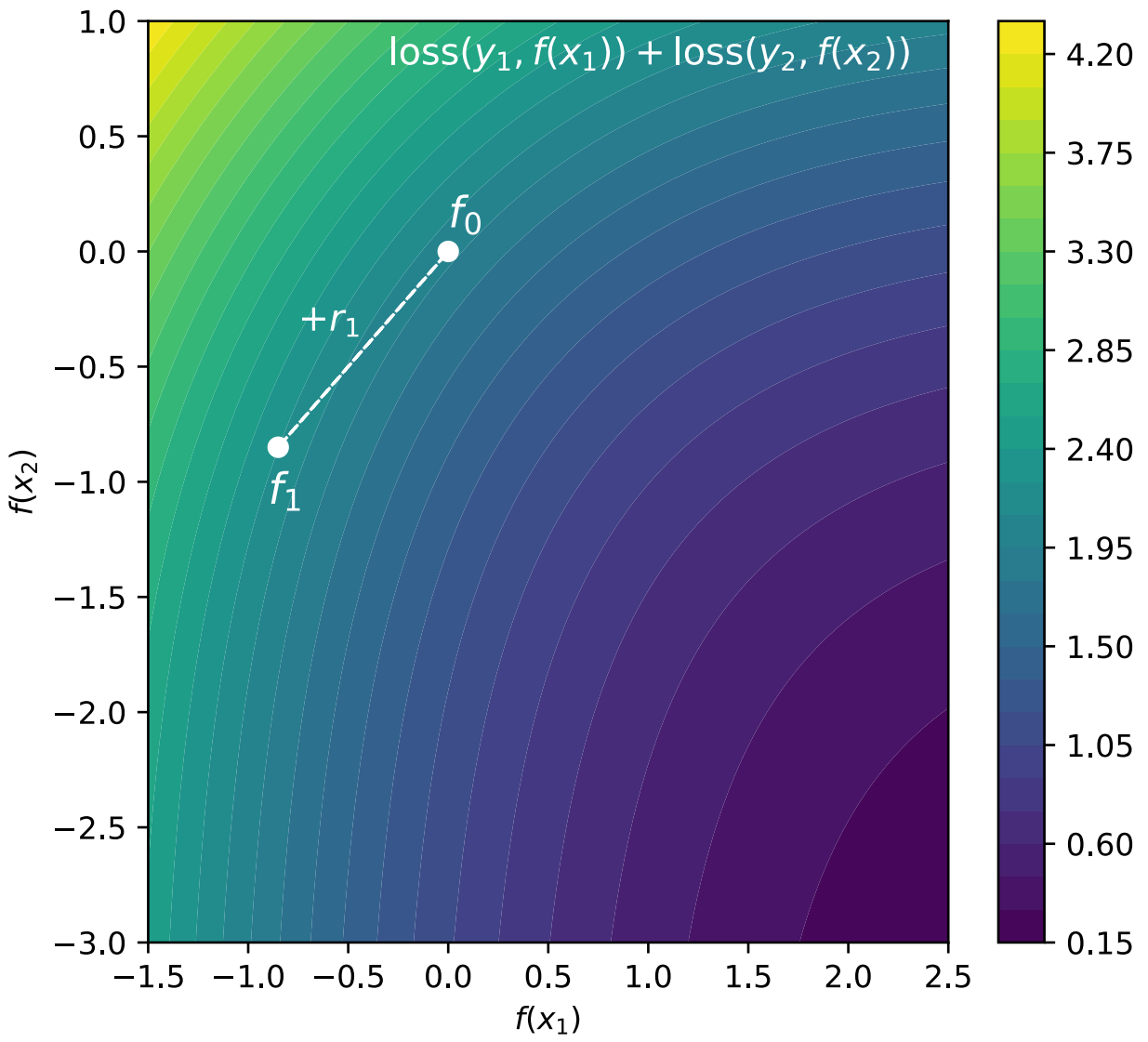


[Friedman 2001 gradient boosting machine]

Mario Boley, Simon Teshuva, Pierre Le Bodic, Geoff Webb, SDM 2021

Gradient boosting fits stage-wise in *output* Space

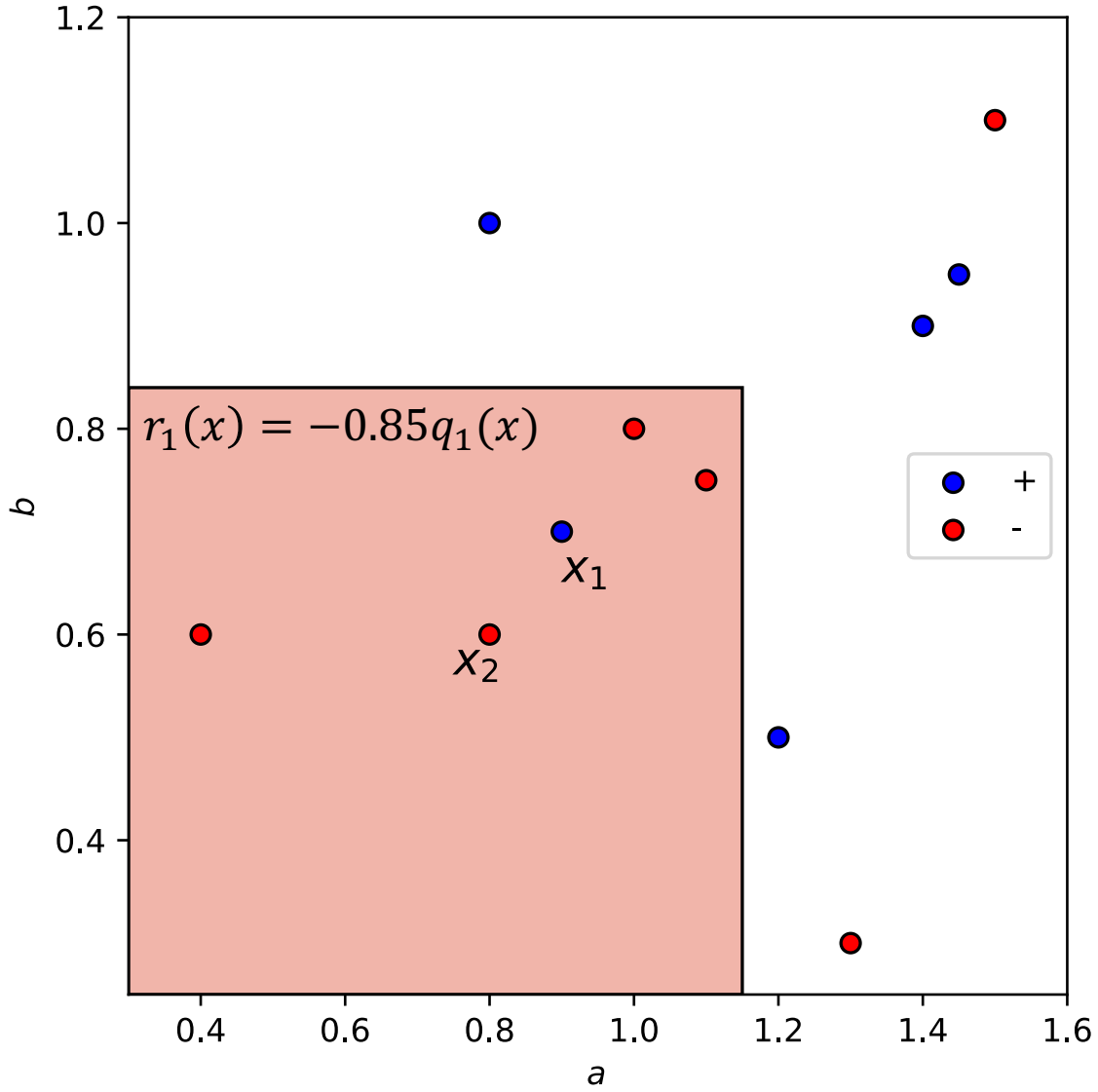


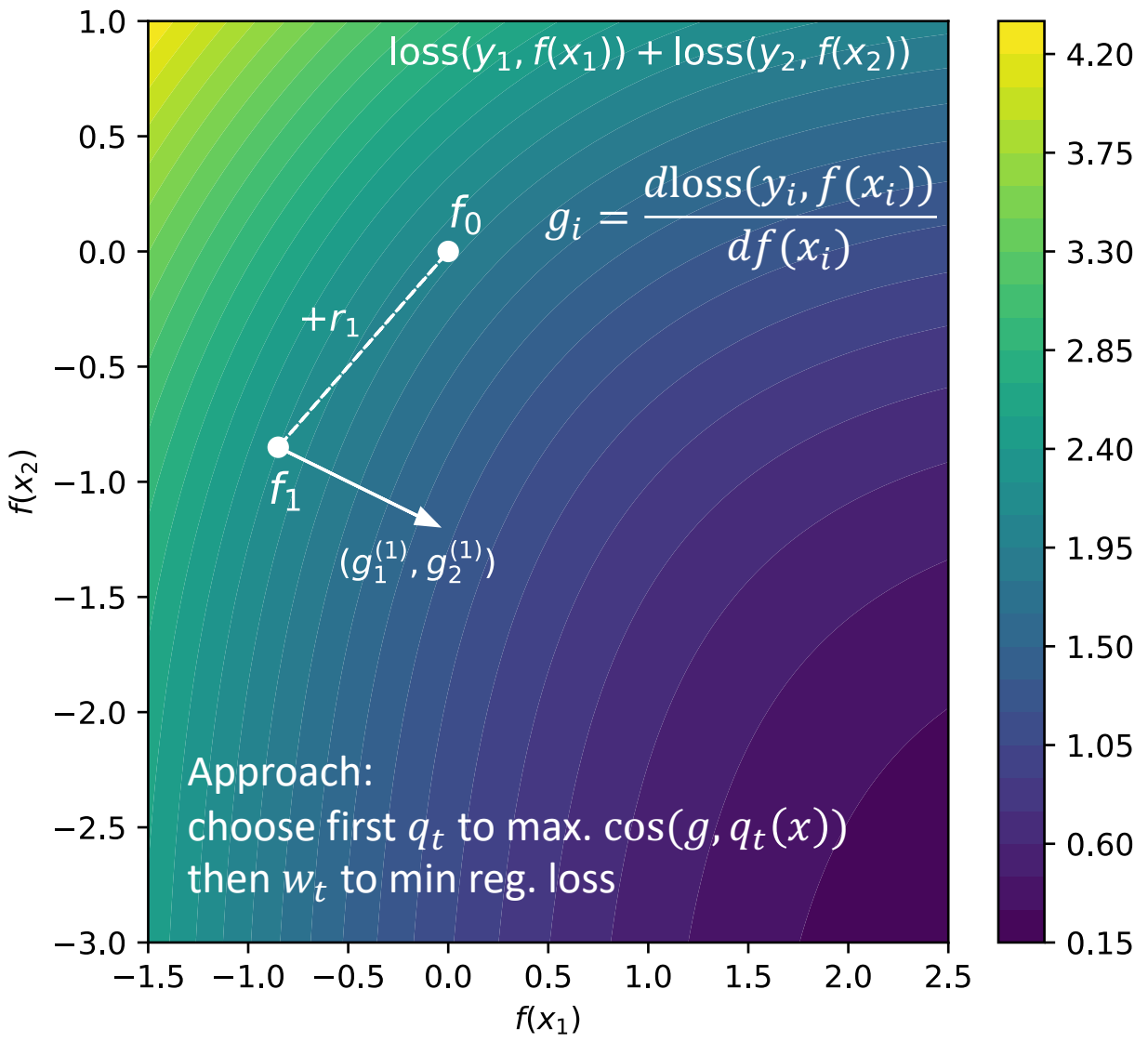


[Friedman 2001 gradient boosting machine]

Mario Boley, Simon Teshuva, Pierre Le Bodic, Geoff Webb, SDM 2021

Idea: guide search by loss gradient of model output

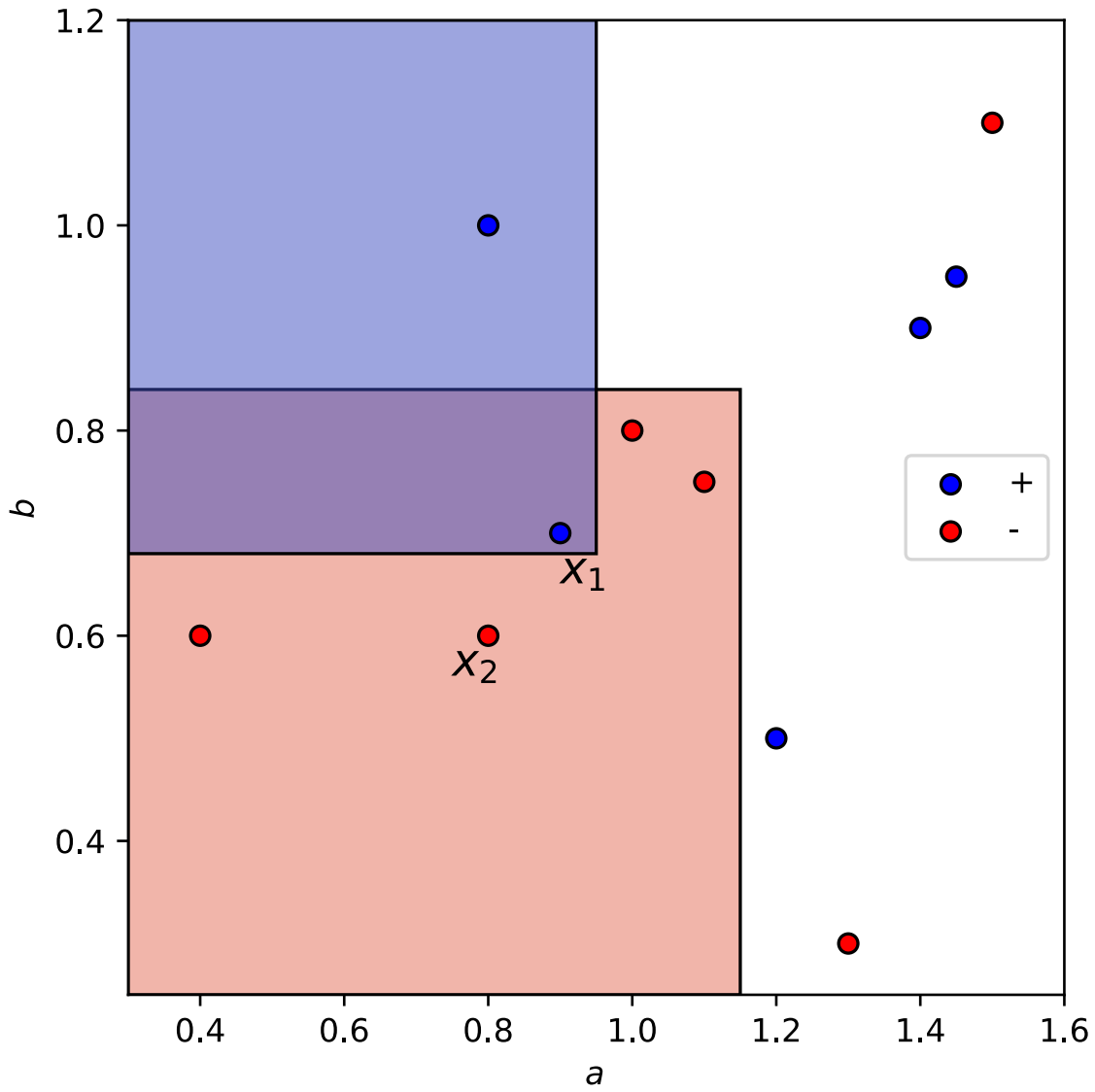


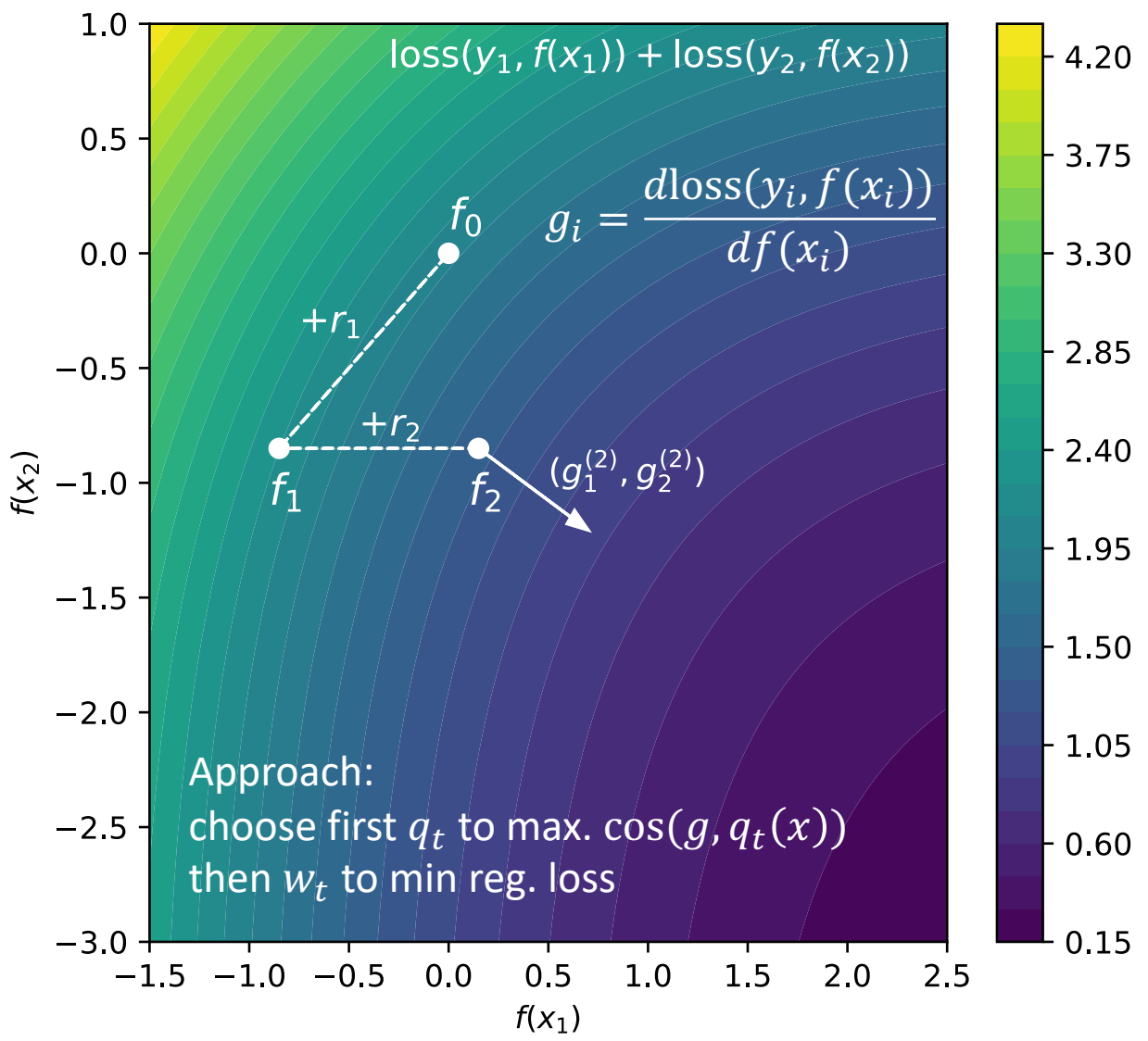


[Friedman 2001 gradient boosting machine]

Mario Boley, Simon Teshuva, Pierre Le Bodic, Geoff Webb, SDM 2021

Idea: guide search by loss gradient of model output

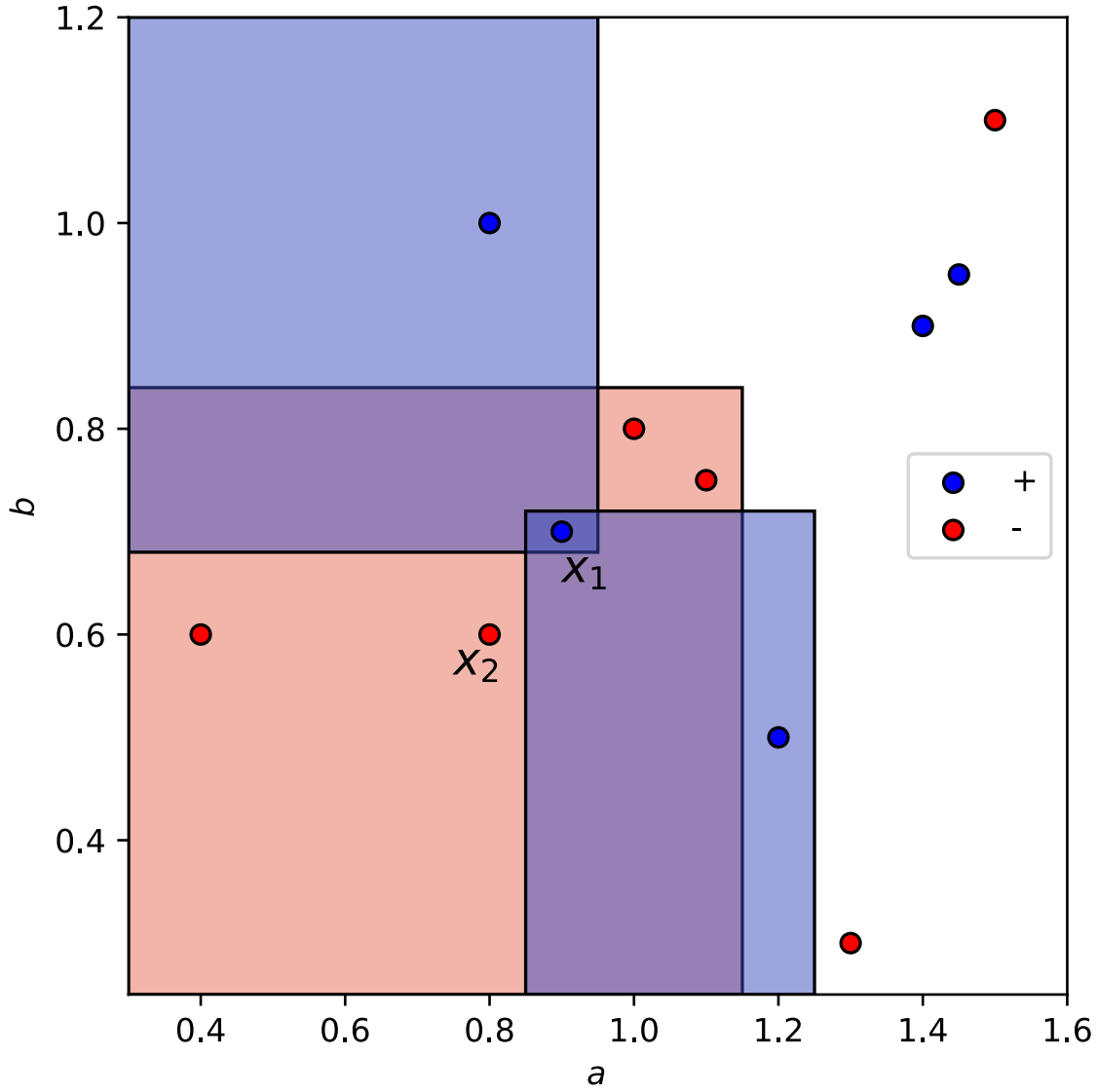


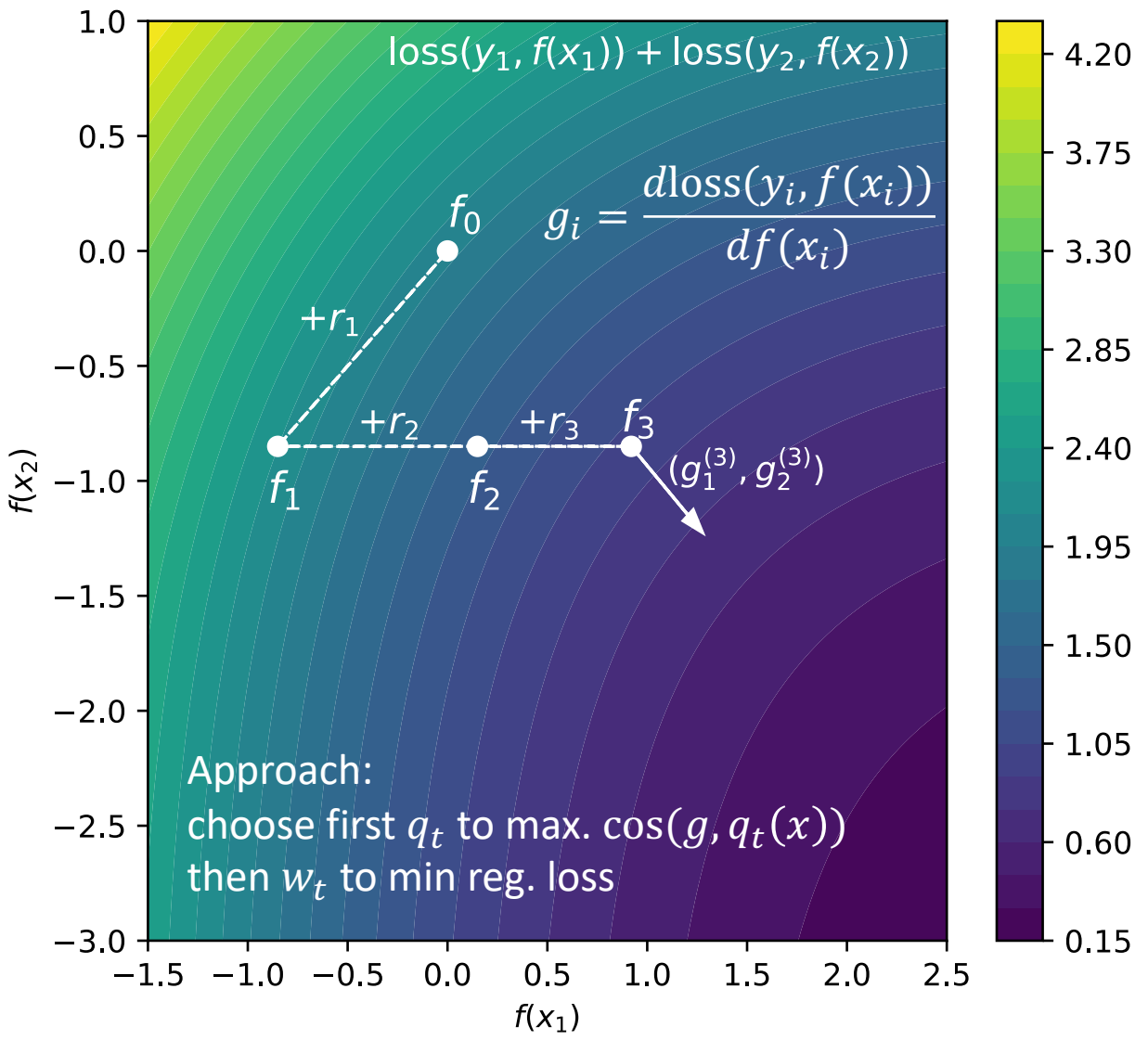


[Friedman 2001 gradient boosting machine]

Mario Boley, Simon Teshuva, Pierre Le Bodic, Geoff Webb, SDM 2021

Idea: guide search by loss gradient of model output

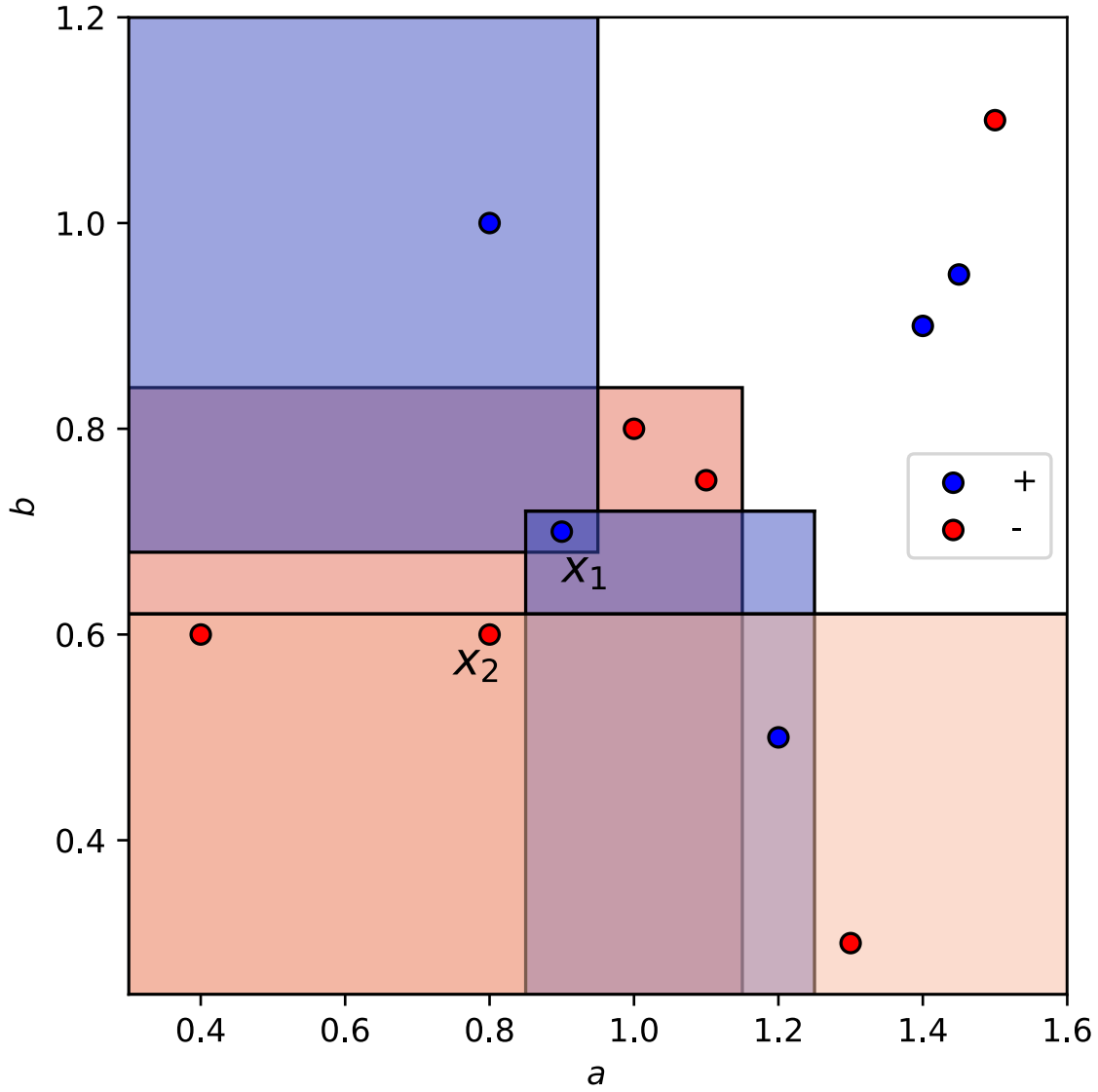


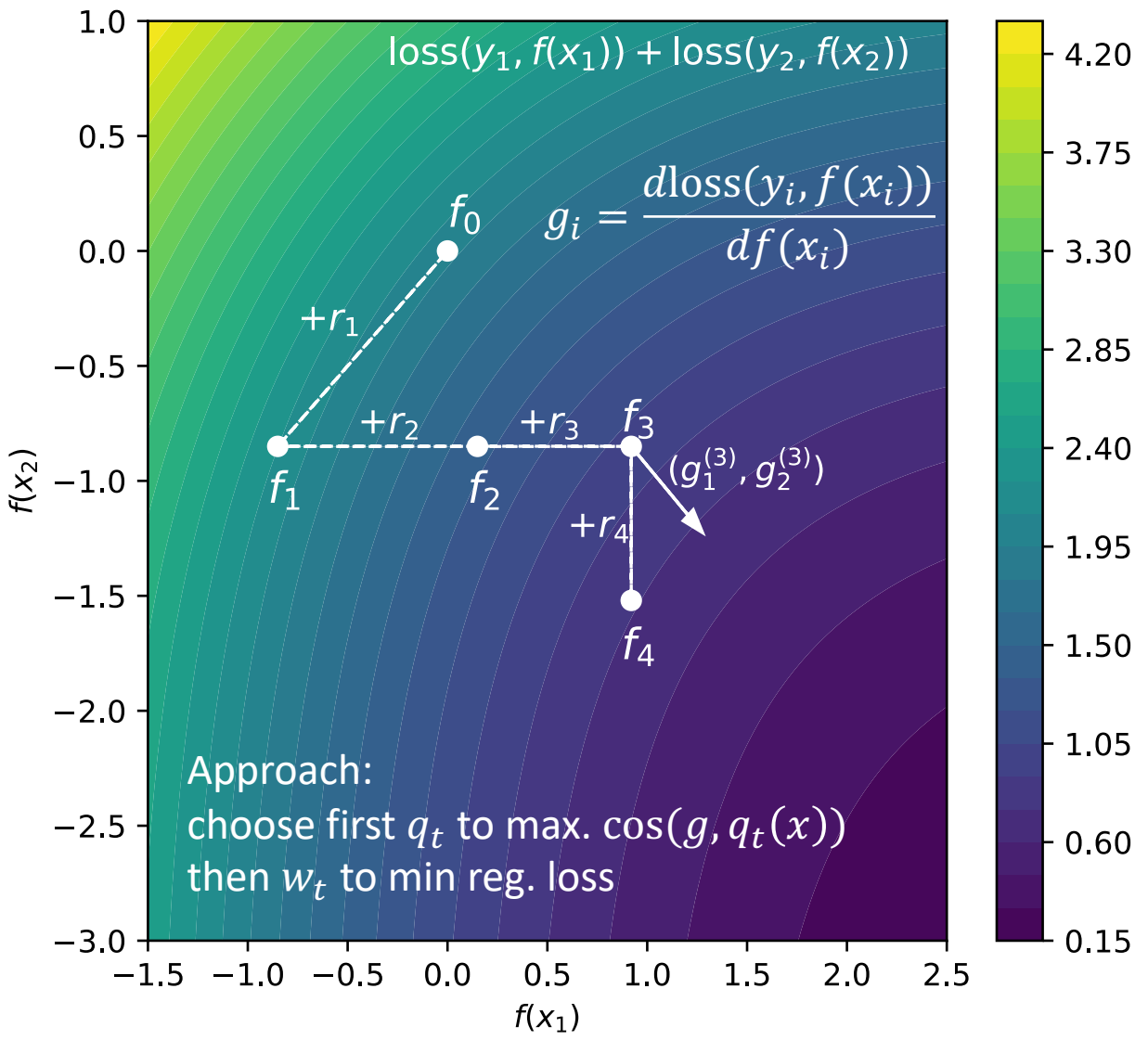


[Friedman 2001 gradient boosting machine]

Mario Boley, Simon Teshuva, Pierre Le Bodic, Geoff Webb, SDM 2021

Idea: guide search by loss gradient of model output

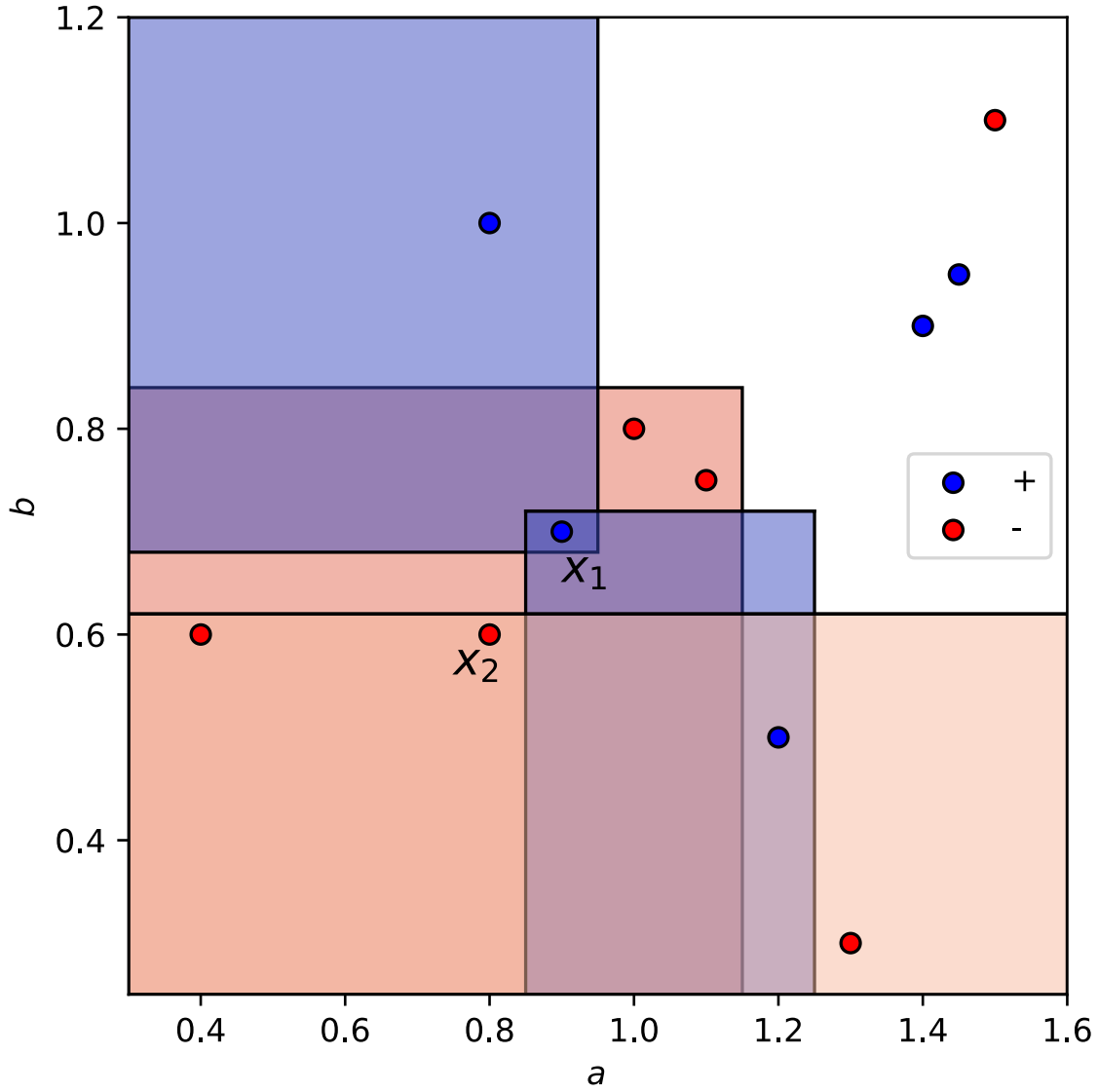


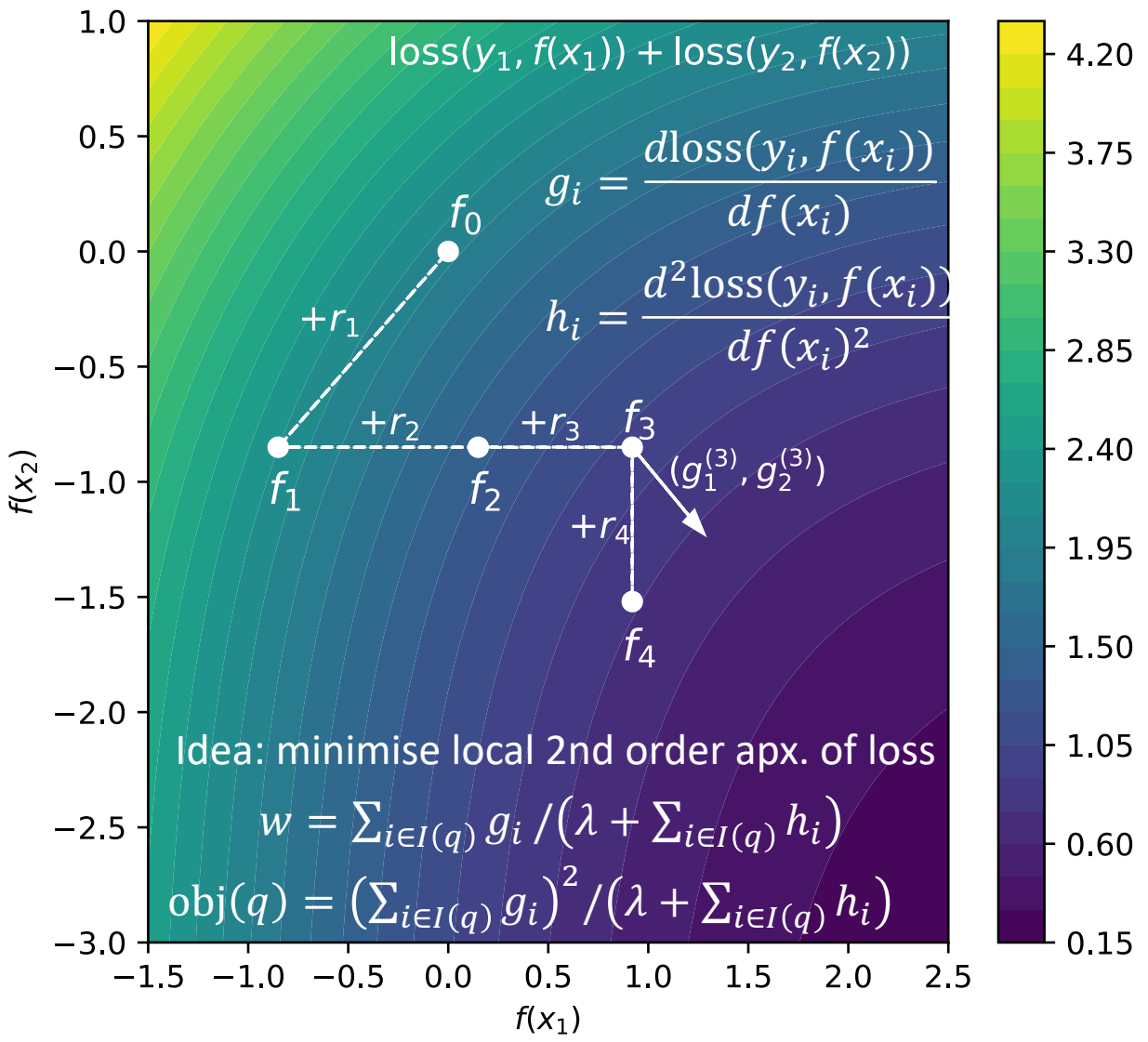


[Friedman 2001 gradient boosting machine]

Mario Boley, Simon Teshuva, Pierre Le Bodic, Geoff Webb, SDM 2021

Variant: “extreme” gradient boosting





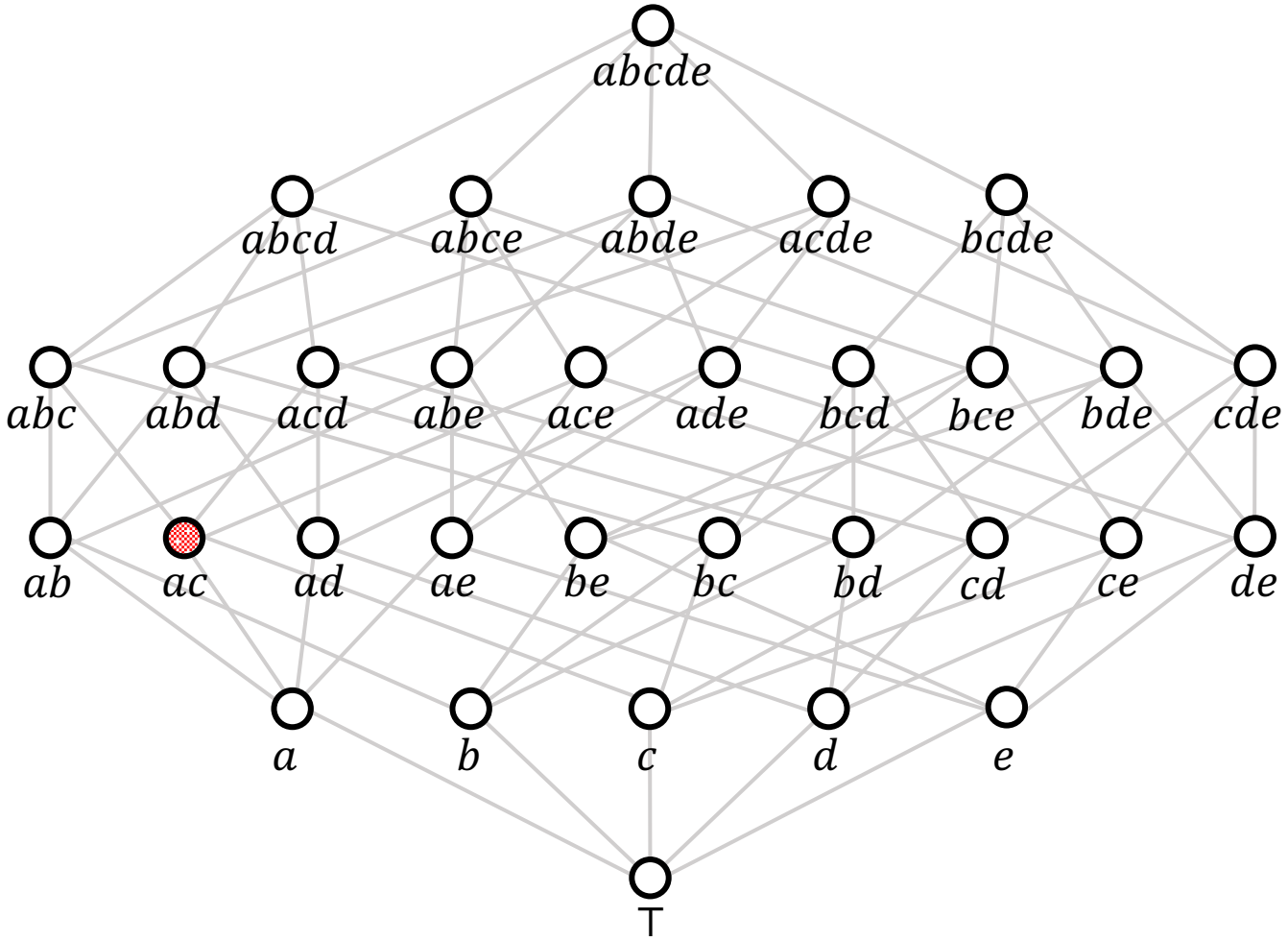
[Chen and Guestrin 2016 Xgboost]

Mario Boley, Simon Teshuva, Pierre Le Bodic, Geoff Webb, SDM 2021

How to optimise gradient boosting objective?

	p_a	p_b	p_c	p_d	p_e	g	h
x_1	■	■	■	■		5.3	1.0
x_2	■	■				-0.5	3.0
x_3	■		■			4.7	0.5
x_4		■	■	■	■	-1.0	1.0
x_5			■	■	■	-1.2	1.5
x_6	■	■			■	-0.7	2.0

$$\text{obj}(q) = \frac{\left(\sum_{i \in I(q)} g_i\right)^2}{\lambda + \sum_{i \in I(q)} h_i}$$

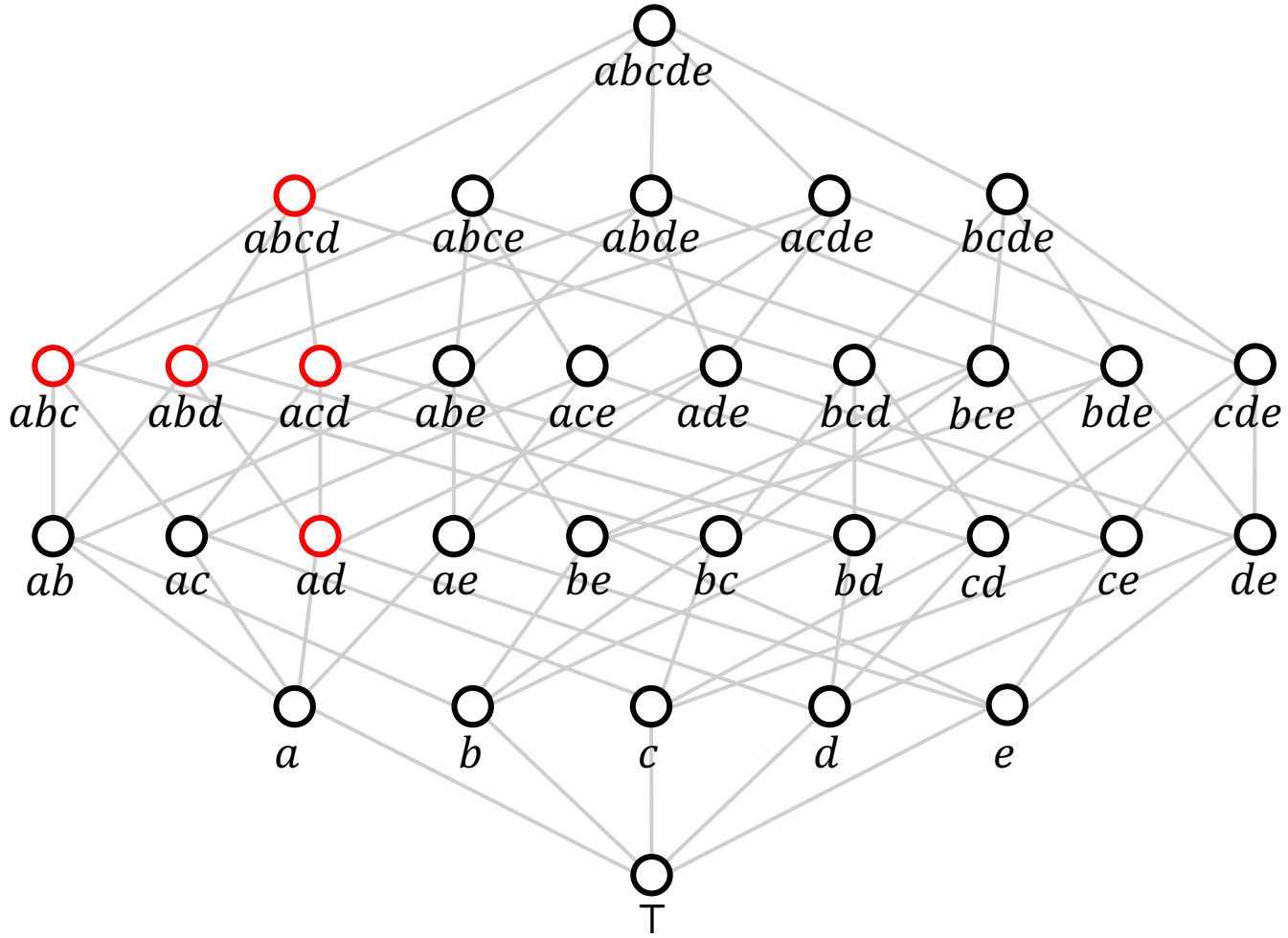


$\{A: A \subseteq \mathcal{P}\} = \mathcal{Q}$

$\max\{\text{obj}(q): q \in \mathcal{Q}\}$

Observation: many conjunctions describe same data

	p_a	p_b	p_c	p_d	p_e	g	h
x_1	■	■	■	■		5.3	1.0
x_2	■	■				-0.5	3.0
x_3	■		■			4.7	0.5
x_4		■	■	■	■	-1.0	1.0
x_5			■	■	■	-1.2	1.5
x_6	■	■			■	-0.7	2.0



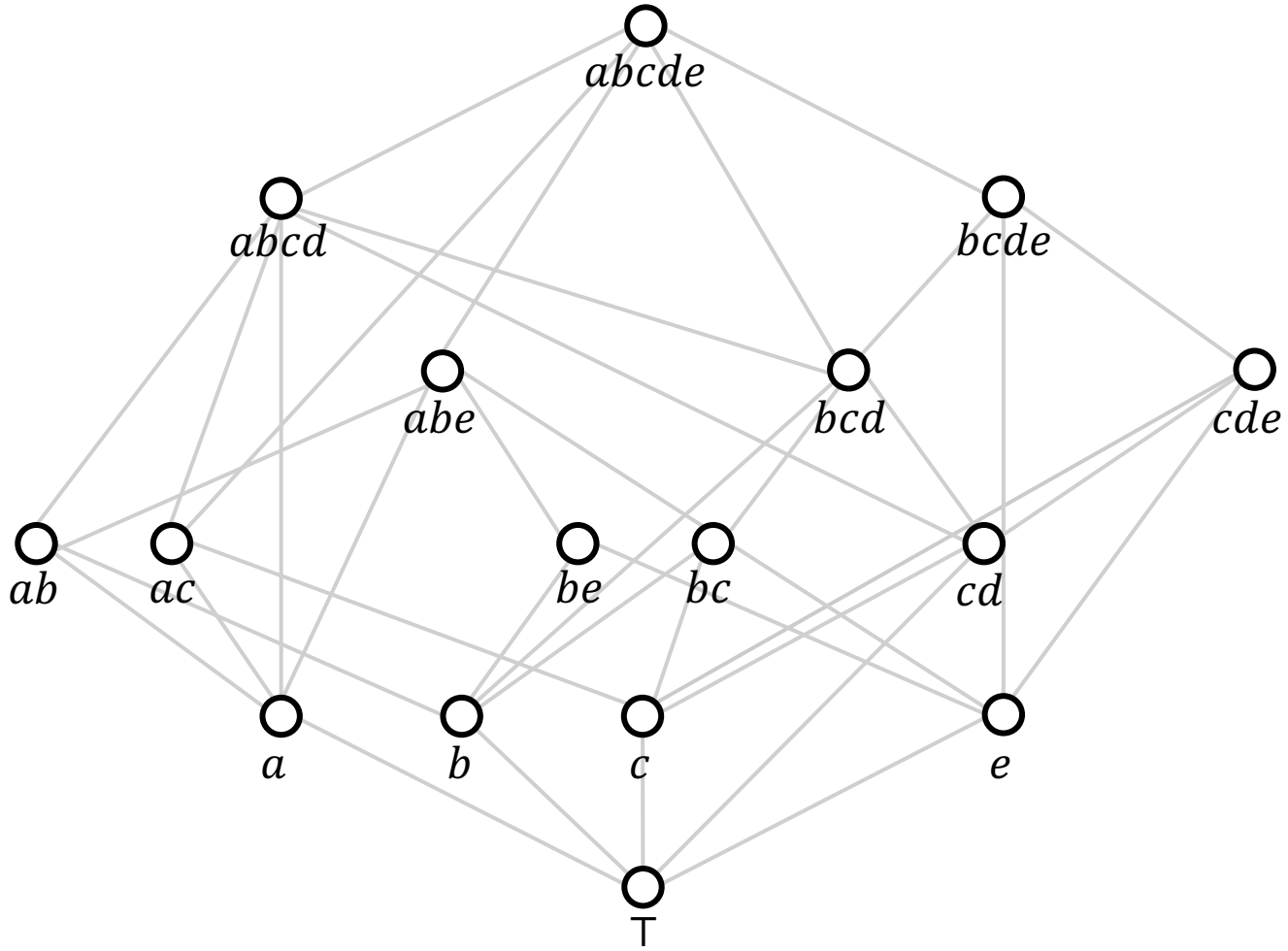
$$\text{obj}(q) = \frac{\left(\sum_{i \in I(q)} g_i\right)^2}{\lambda + \sum_{i \in I(q)} h_i}$$

○ $\{A : A \subseteq \mathcal{P}\} = \mathcal{Q}$

[Bastide et. al, 2000; Boley and Grosskreutz 2009]

Sufficient to search one representative per extent

	p_a	p_b	p_c	p_d	p_e	g	h
x_1	■	■	■	■		2.3	1.0
x_2	■	■				-0.5	3.0
x_3	■		■			2.7	0.5
x_4		■	■	■	■	-2.0	1.0
x_5			■	■	■	-1.2	1.5
x_6	■	■			■	-2.0	2.0



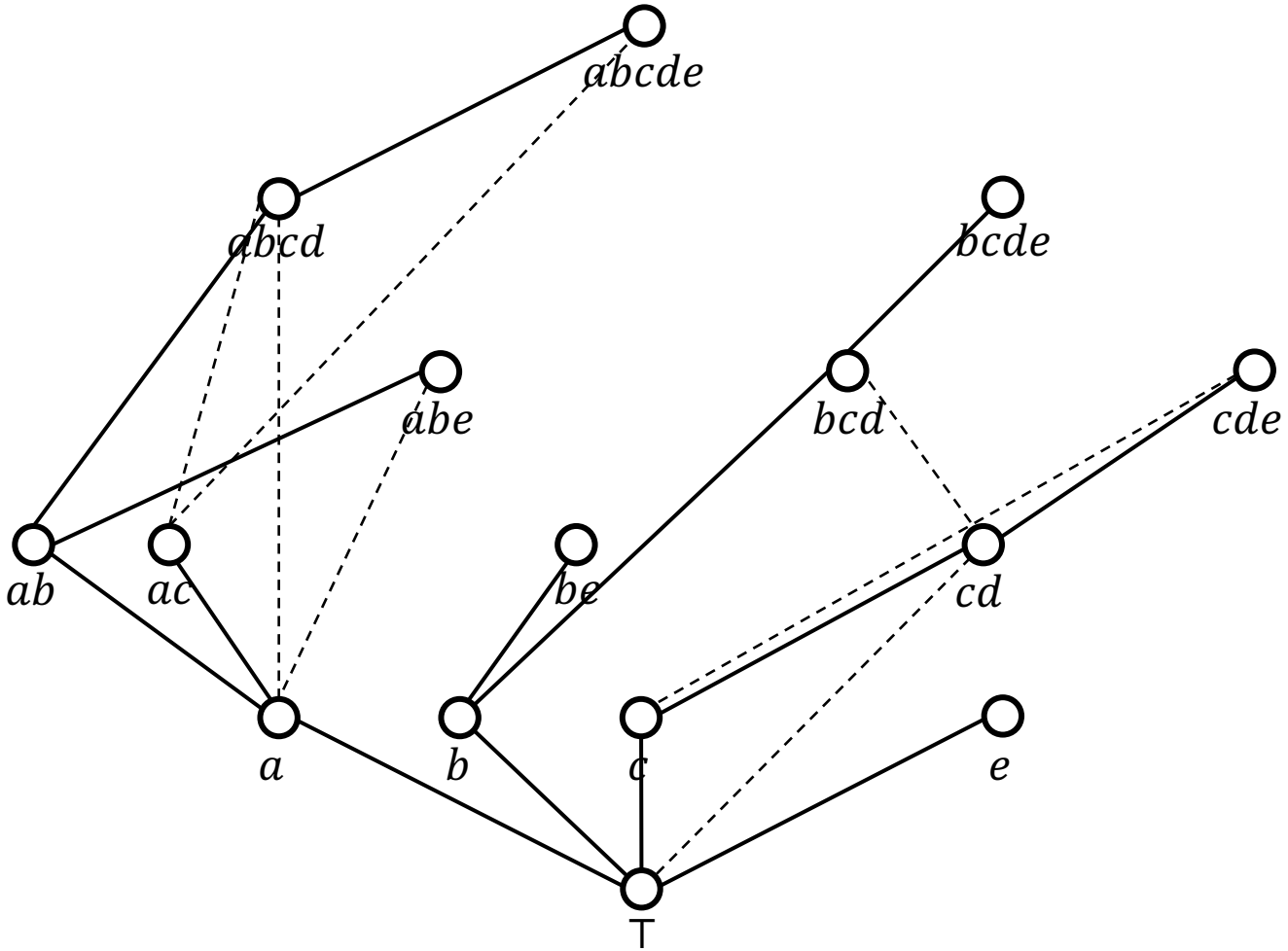
$$\text{obj}(q) = \frac{\left(\sum_{i \in I(q)} g_i\right)^2}{\lambda + \sum_{i \in I(q)} h_i}$$

[Bastide et. al, 2000; Boley and Grosskreutz 2009]

1st component: non-redundant branching

	p_a	p_b	p_c	p_d	p_e	g	h
x_1	■	■	■	■		2.3	1.0
x_2	■	■				-0.5	3.0
x_3	■		■			2.7	0.5
x_4		■	■	■	■	-2.0	1.0
x_5			■	■	■	-1.2	1.5
x_6	■	■			■	-2.0	2.0

$$\text{obj}(q) = \frac{\left(\sum_{i \in I(q)} g_i\right)^2}{\lambda + \sum_{i \in I(q)} h_i}$$

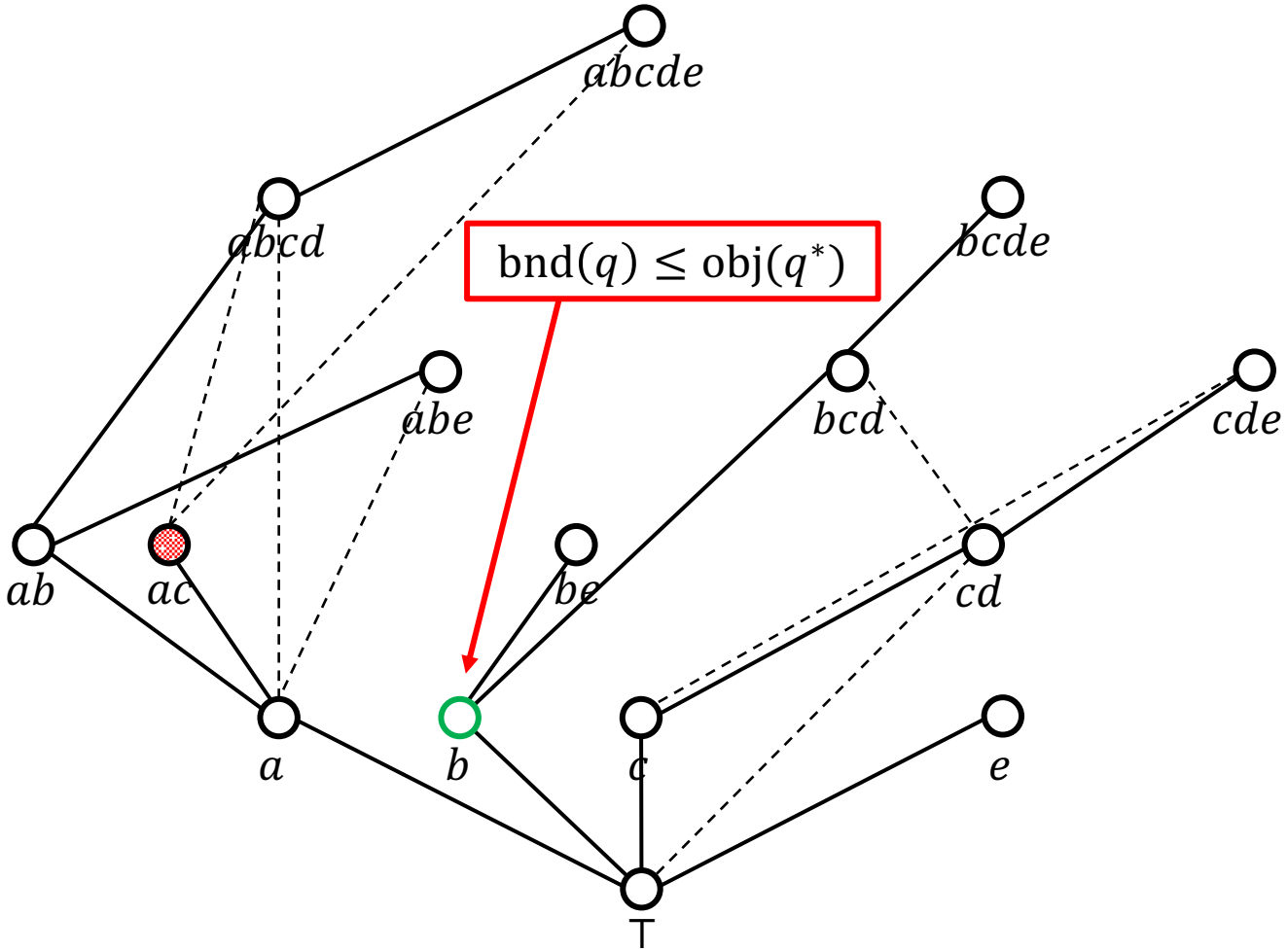


2nd component: bound value to prune branches

	p_a	p_b	p_c	p_d	p_e	g	h
x_1	■	■	■	■		2.3	1.0
x_2	■	■				-0.5	3.0
x_3	■		■			2.7	0.5
x_4		■	■	■	■	-2.0	1.0
x_5			■	■	■	-1.2	1.5
x_6	■	■			■	-2.0	2.0

$$\text{obj}(q) = \frac{\left(\sum_{i \in I(q)} g_i\right)^2}{\lambda + \sum_{i \in I(q)} h_i}$$

$$\text{bnd}(q) \geq \max\{\text{obj}(q') : q' \succcurlyeq q\}$$

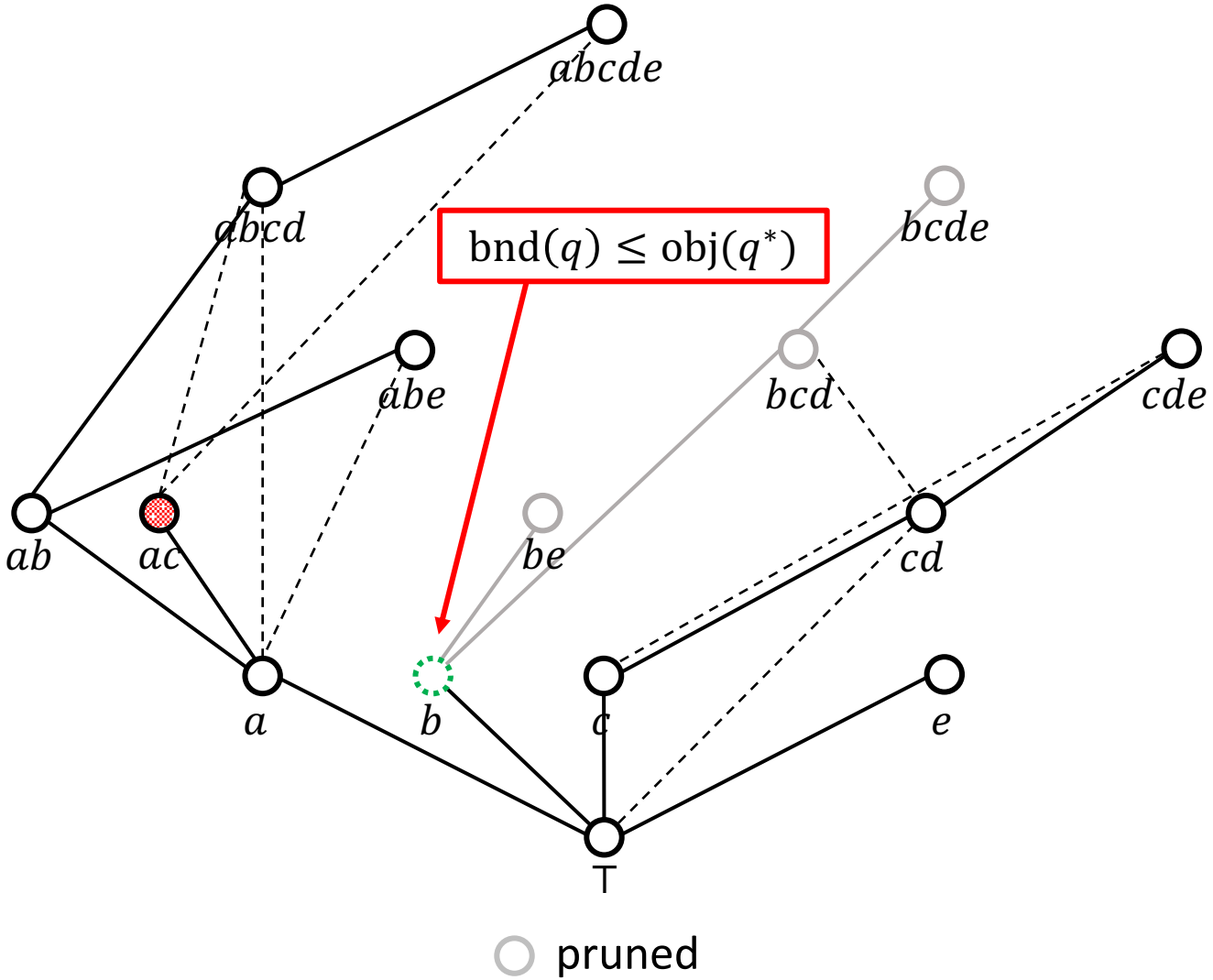


2nd component: bound value to prune branches

	p_a	p_b	p_c	p_d	p_e	g	h
x_1	■	■	■	■		2.3	1.0
x_2	■	■				-0.5	3.0
x_3	■		■			2.7	0.5
x_4		■	■	■	■	-2.0	1.0
x_5			■	■	■	-1.2	1.5
x_6	■	■			■	-2.0	2.0

$$\text{obj}(q) = \frac{\left(\sum_{i \in I(q)} g_i\right)^2}{\lambda + \sum_{i \in I(q)} h_i}$$

$$\text{bnd}(q) \geq \max\{\text{obj}(q') : q' \succcurlyeq q\}$$

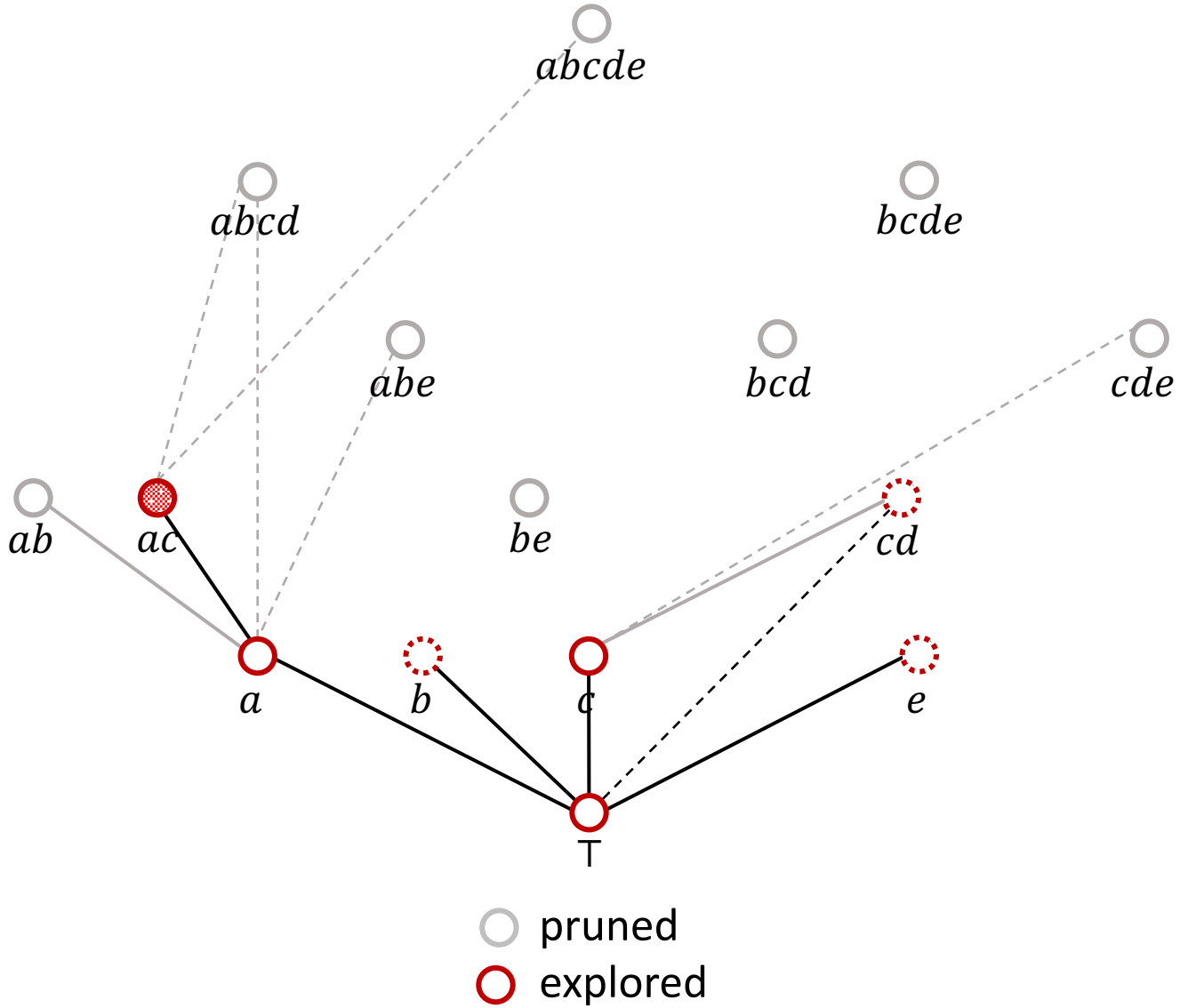


Together can reduce computation substantially

	p_a	p_b	p_c	p_d	p_e	g	h
x_1	■	■	■	■		2.3	1.0
x_2	■	■				-0.5	3.0
x_3	■		■			2.7	0.5
x_4		■	■	■	■	-2.0	1.0
x_5			■	■	■	-1.2	1.5
x_6	■	■			■	-2.0	2.0

$$\text{obj}(q) = \frac{\left(\sum_{i \in I(q)} g_i\right)^2}{\lambda + \sum_{i \in I(q)} h_i}$$

$$\text{bnd}(q) \geq \max\{\text{obj}(q') : q' \succcurlyeq q\}$$

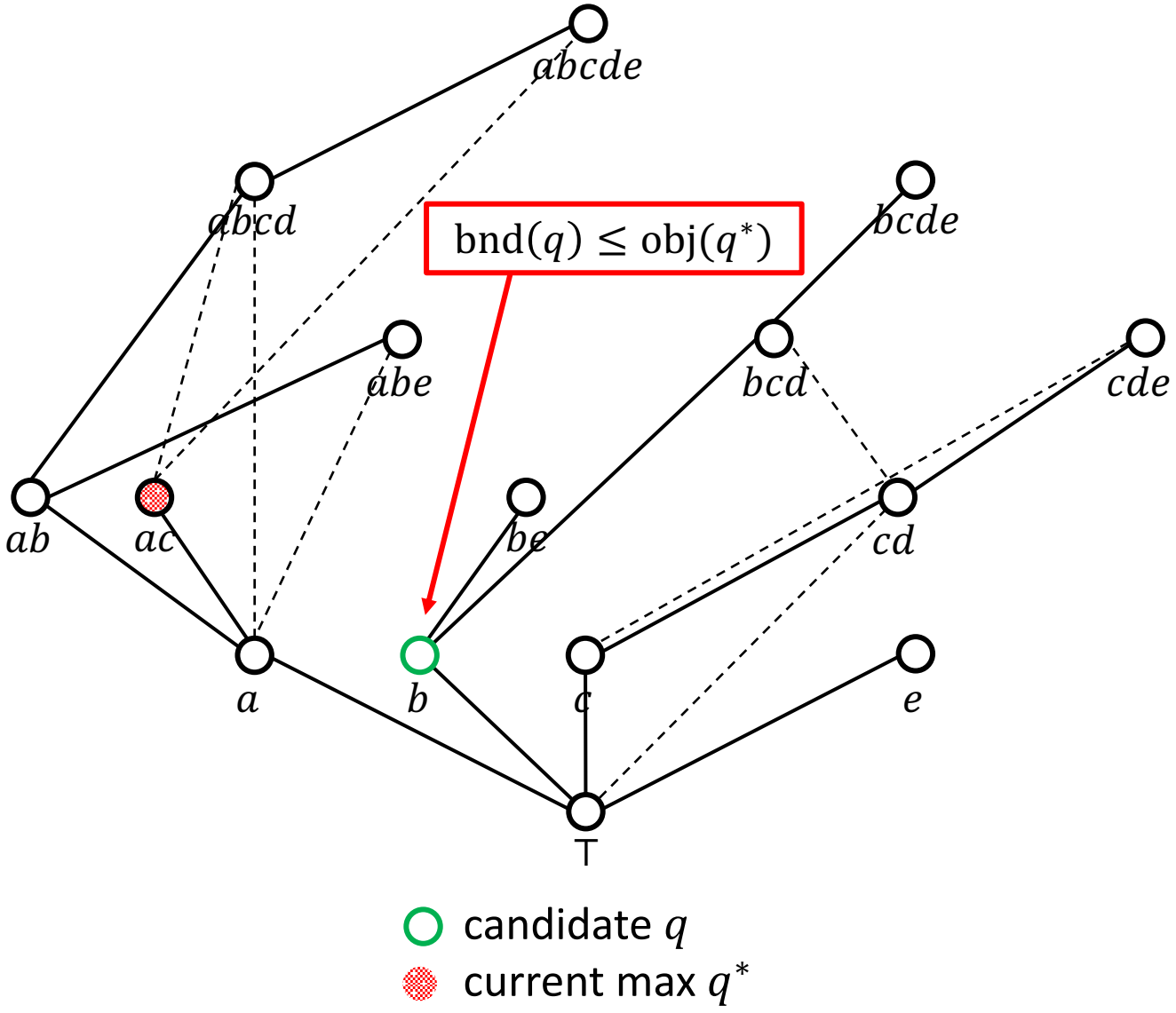


How to define/compute bounding function?

	p_a	p_b	p_c	p_d	p_e	g	h
x_1	■	■	■	■		2.3	1.0
x_2	■	■				-0.5	3.0
x_3	■		■			2.7	0.5
x_4		■	■	■	■	-2.0	1.0
x_5			■	■	■	-1.2	1.5
x_6	■	■			■	-2.0	2.0

$$\text{obj}(q) = \frac{\left(\sum_{i \in I(q)} g_i\right)^2}{\lambda + \sum_{i \in I(q)} h_i}$$

$$\text{bnd}(q) \geq \max\{\text{obj}(q') : q' \succcurlyeq q\}$$

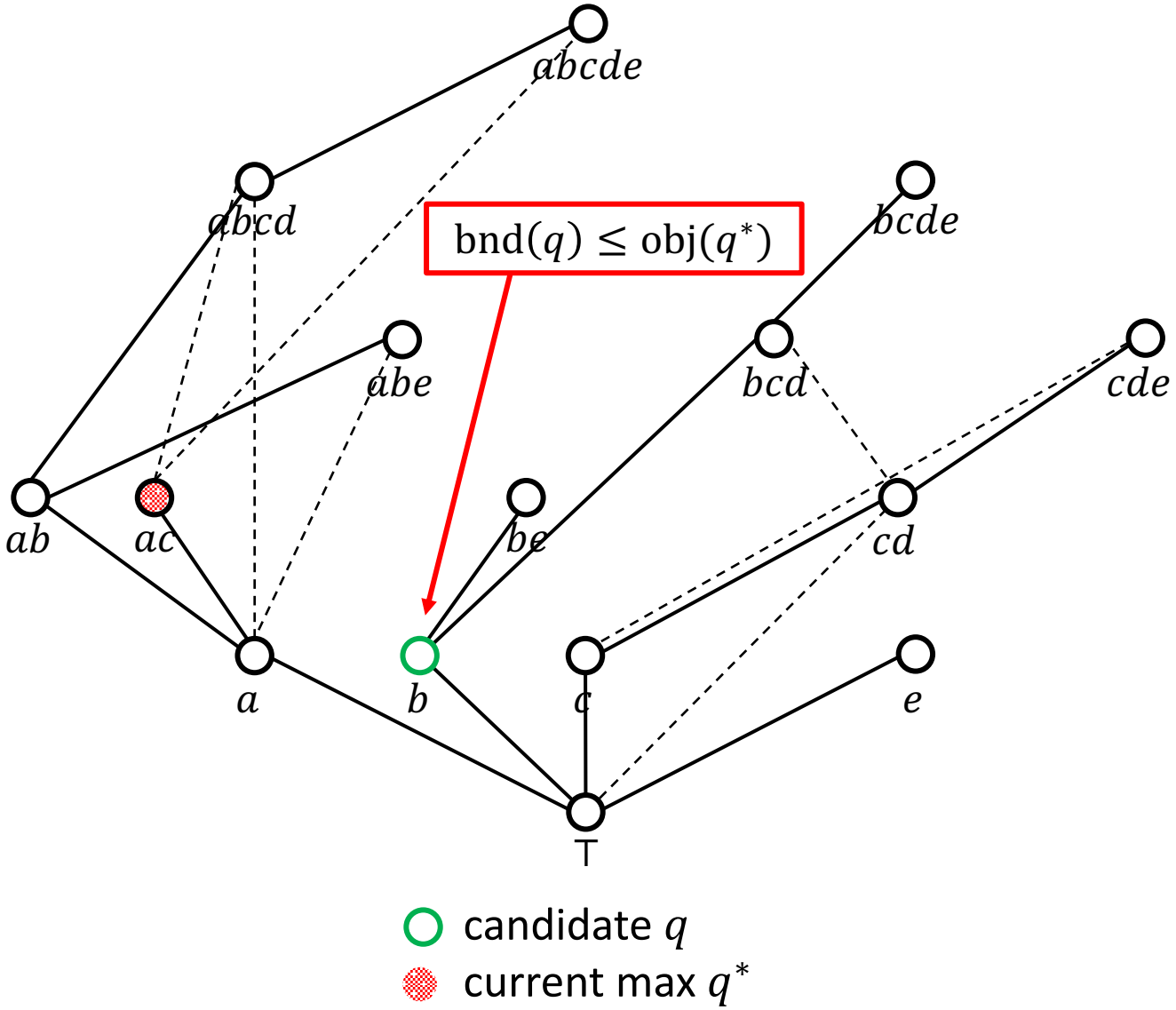


Solve relaxed problem: find optimal data subset

	p_a	p_b	p_c	p_d	p_e	g	h
x_1	■	■	■	■		2.3	1.0
x_2	■	■				-0.5	3.0
x_3	■		■			2.7	0.5
x_4		■	■	■	■	-2.0	1.0
x_5			■	■	■	-1.2	1.5
x_6	■	■			■	-2.0	2.0

$$\text{obj}(q) = \frac{\left(\sum_{i \in I(q)} g_i\right)^2}{\lambda + \sum_{i \in I(q)} h_i}$$

$$\begin{aligned} \text{bnd}(q) &= \max\{\text{obj}(J) : J \subseteq I(q)\} \\ &\geq \max\{\text{obj}(q') : q' \succcurlyeq q\} \end{aligned}$$



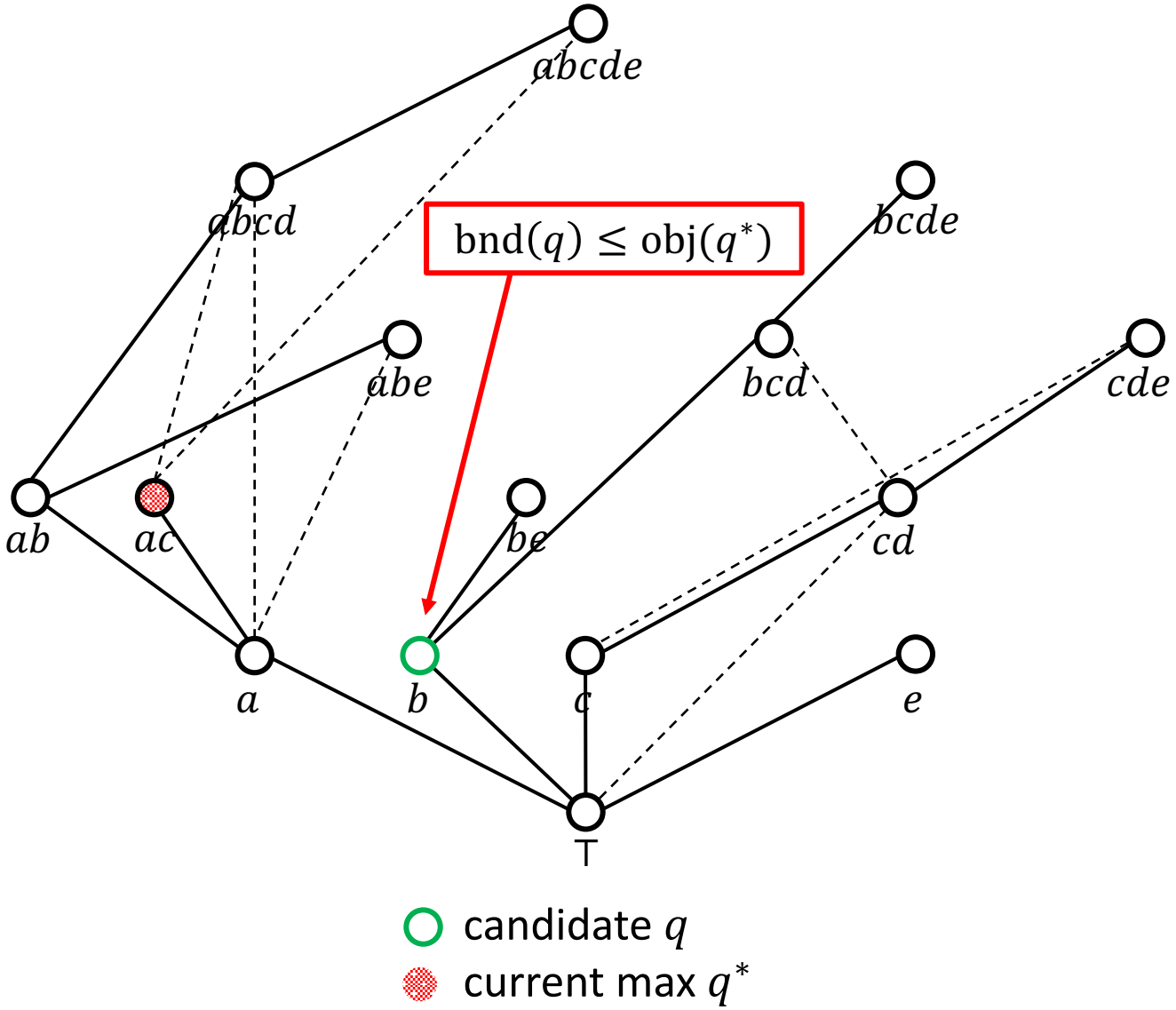
[Moroshita and Sese 2000; Grosskreutz et al. 2008, Boley et al. 2017]

Solve relaxed problem: find optimal data subset

	p_a	p_b	p_c	p_d	p_e	g	h
x_1	■	■	■	■		2.3	1.0
x_2	■	■				-0.5	3.0
x_3	■		■			2.7	0.5
x_4		■	■	■	■	-2.0	1.0
x_5			■	■	■	-1.2	1.5
x_6	■	■			■	-2.0	2.0

$$\text{obj}(q) = \frac{\left(\sum_{i \in I(q)} g_i\right)^2}{\lambda + \sum_{i \in I(q)} h_i}$$

$$\begin{aligned} \text{bnd}(q) &= \max\{\text{obj}(J) : J \subseteq I(q)\} \\ &\geq \max\{\text{obj}(q') : q' \succcurlyeq q\} \end{aligned}$$



[Moroshita and Sese 2000; Grosskreutz et al. 2008, Boley et al. 2017]

Main result: linear time bound with pre-sorting

	p_a	p_b	p_c	p_d	p_e	g	h	g/h
x_1	■	■	■	■		2.3	1.0	2.3
x_2	■	■				-0.5	3.0	-0.16
x_3	■		■			2.7	0.5	5.4
x_4		■	■	■	■	-2.0	1.0	-2.0
x_5			■	■	■	-1.2	1.5	-0.8
x_6	■	■			■	-2.0	2.0	-1.0

pre-sort wrt g/h

	p_a	p_b	p_c	p_d	p_e	g	h	g/h
x_4		■	■	■	■	-2.0	1.0	-2.0
x_6	■	■			■	-2.0	2.0	-1.0
x_5			■	■	■	-1.2	1.5	-0.8
x_2	■	■				-0.5	3.0	-0.16
x_1	■	■	■	■		2.3	1.0	2.3
x_3	■		■			2.7	0.5	5.4

$$\text{obj}(q) = \frac{\left(\sum_{i \in I(q)} g_i\right)^2}{\lambda + \sum_{i \in I(q)} h_i}$$

Can check all prefixes and suffixes in time $O(|I(q)|)$

$$\begin{aligned}\text{bnd}(q) &= \max\{\text{obj}(J) : J \subseteq I(q)\} \\ &= \max\{\text{obj}(I_k(q)), \text{obj}(I_{-k}(q)) : 0 \leq k \leq |I(q)|\} \\ &\geq \max\{\text{obj}(q') : q' \succcurlyeq q\}\end{aligned}$$

optimum attained by prefix or suffix of sorted sequence

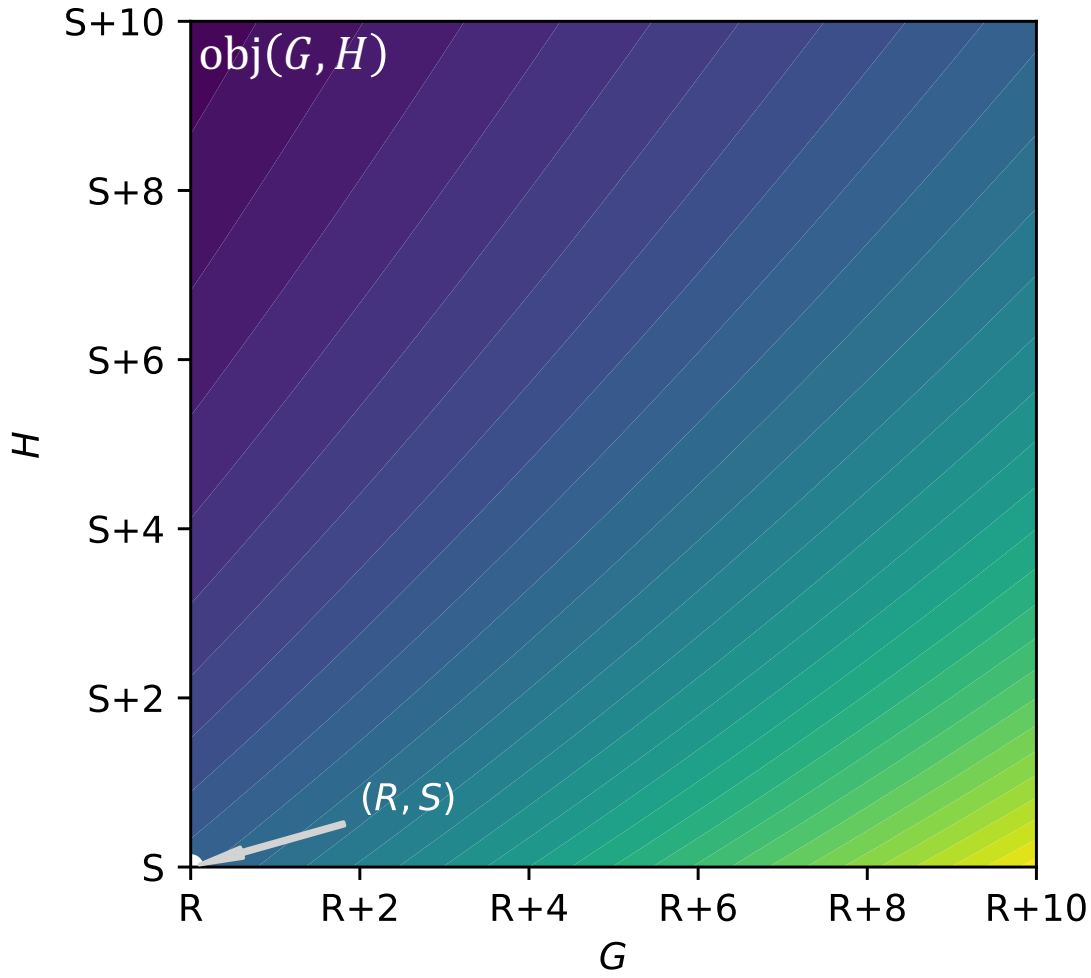
Proof: sequences with gaps are dominated

	p_a	p_b	p_c	p_d	p_e	g	h	g/h
x_4		■	■	■	■	-2.0	1.0	-2.0
x_6	■	■			■	-2.0	2.0	-1.0
x_5			■	■	■	-1.2	1.5	-0.8
x_2	■	■				-0.5	3.0	-0.16
x_1	■	■	■	■		2.3	1.0	2.3
x_3	■		■			2.7	0.5	5.4

$\frac{R}{S}$

$$\text{obj}(q) = \frac{\left(\sum_{i \in I(q)} g_i\right)^2}{\lambda + \sum_{i \in I(q)} h_i}$$

$$\text{obj}(G, H) = \frac{G^2}{\lambda + H}$$



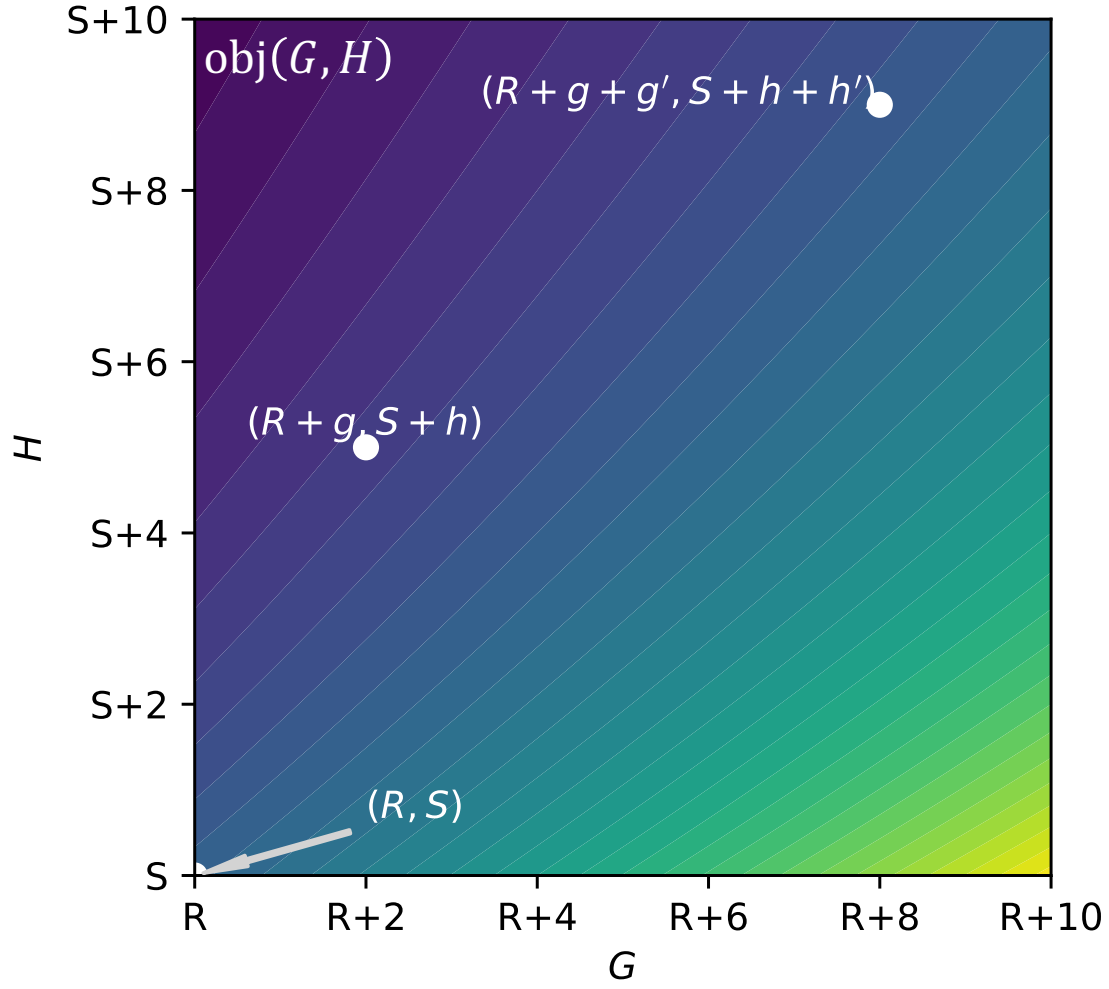
Proof: sequences with gaps are dominated

	p_a	p_b	p_c	p_d	p_e	g	h	g/h
x_4		■	■	■	■	-2.0	1.0	-2.0
x_6	■	■			■	-2.0	2.0	-1.0
x_5			■	■	■	-1.2	1.5	-0.8
x_2	■	■				-0.5	3.0	-0.16
x_1	■	■	■	■		2.3	1.0	2.3
x_3	■		■			2.7	0.5	5.4

$\frac{R}{S}$ $\frac{g'}{h'} \leq \frac{g}{h}$

$$\text{obj}(q) = \frac{\left(\sum_{i \in I(q)} g_i\right)^2}{\lambda + \sum_{i \in I(q)} h_i}$$

$$\text{obj}(G, H) = \frac{G^2}{\lambda + H}$$



Proof: sequences with gaps are dominated

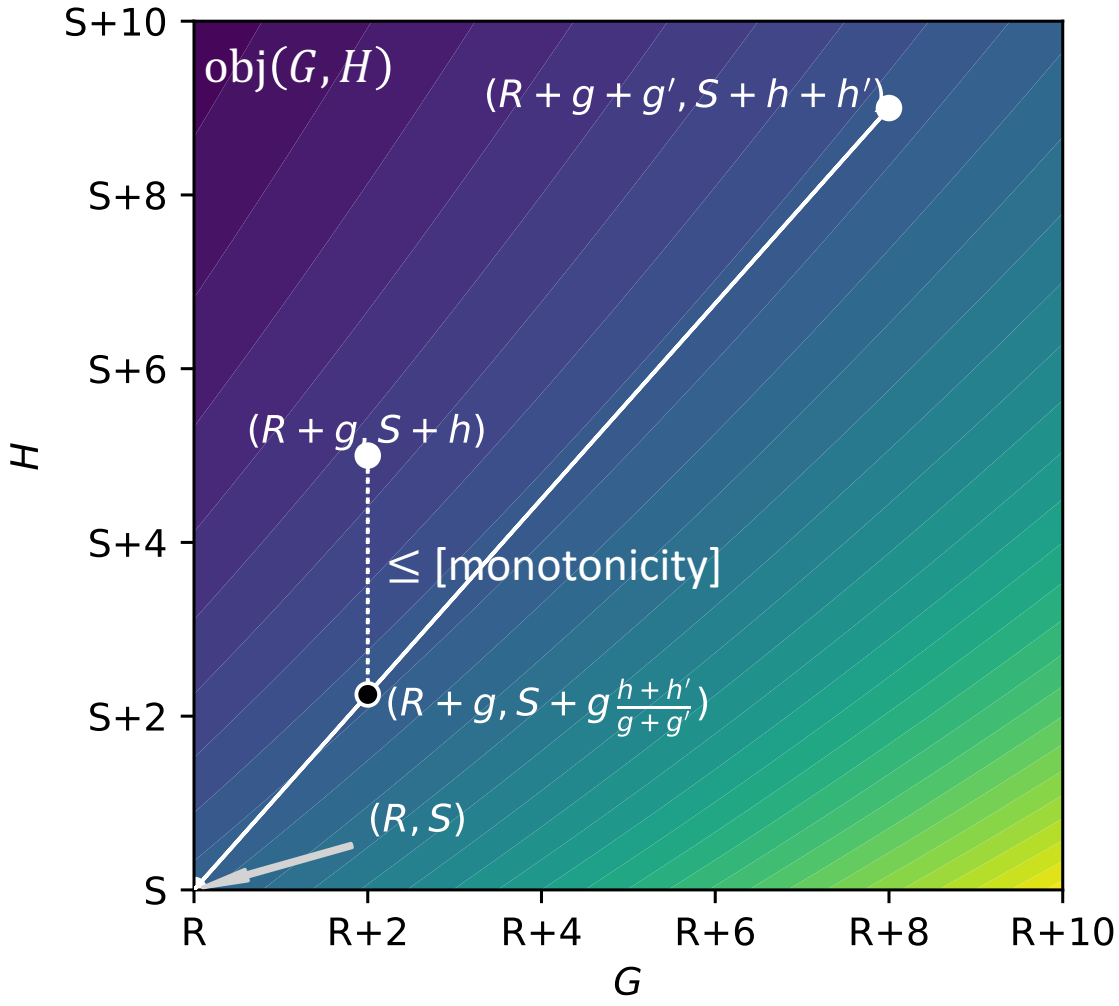
	p_a	p_b	p_c	p_d	p_e	g	h	g/h
x_4		■	■	■	■	-2.0	1.0	-2.0
x_6	■	■			■	-2.0	2.0	-1.0
x_5			■	■	■	-1.2	1.5	-0.8
x_2	■	■				-0.5	3.0	-0.16
x_1	■	■	■	■		2.3	1.0	2.3
x_3	■		■			2.7	0.5	5.4

$\frac{R}{S}$

$\frac{g'}{h'} \leq \frac{g}{h}$

$$\text{obj}(q) = \frac{\left(\sum_{i \in I(q)} g_i\right)^2}{\lambda + \sum_{i \in I(q)} h_i}$$

$$\text{obj}(G, H) = \frac{G^2}{\lambda + H}$$



Proof: sequences with gaps are dominated

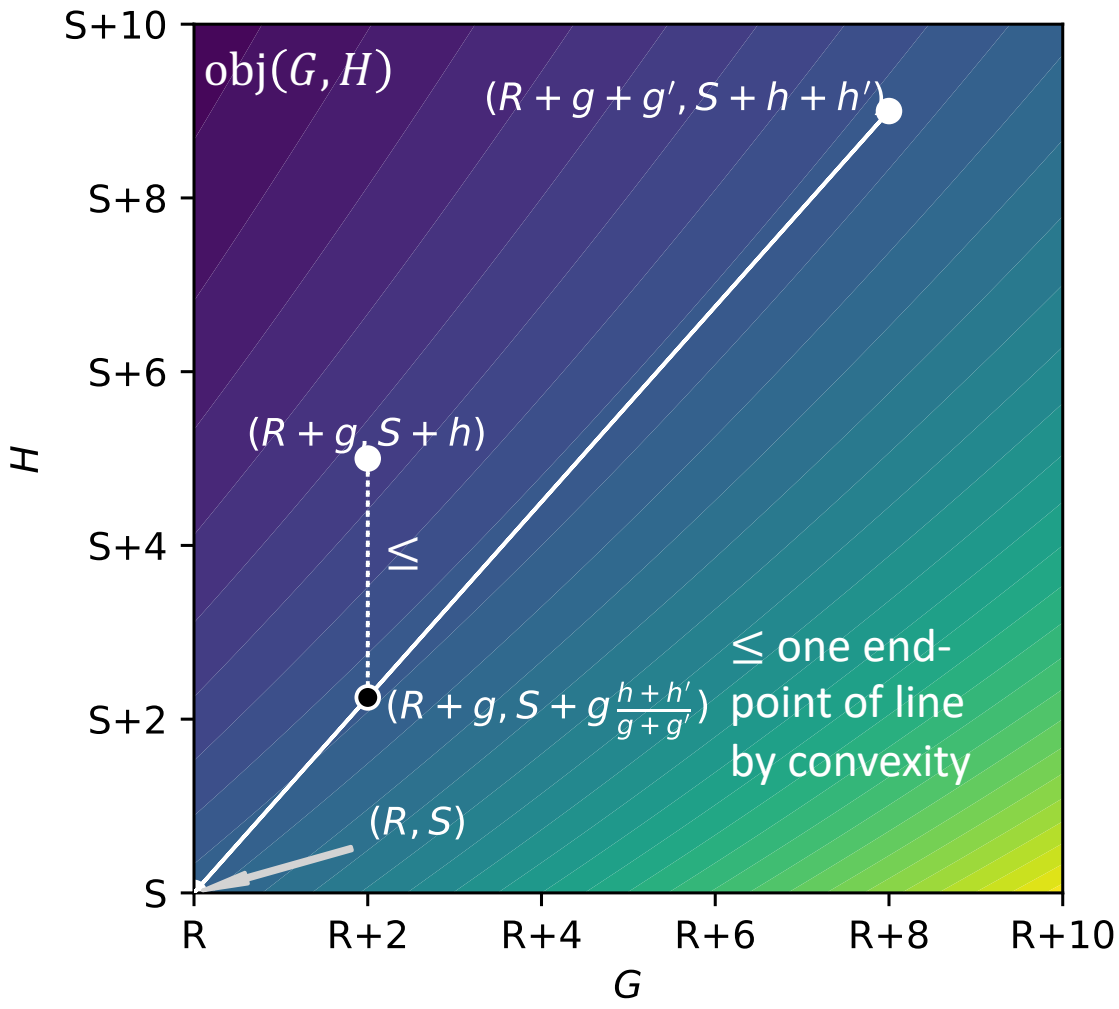
	p_a	p_b	p_c	p_d	p_e	g	h	g/h
x_4		■	■	■	■	-2.0	1.0	-2.0
x_6	■	■			■	-2.0	2.0	-1.0
x_5			■	■	■	-1.2	1.5	-0.8
x_2	■	■				-0.5	3.0	-0.16
x_1	■	■	■	■		2.3	1.0	2.3
x_3	■		■			2.7	0.5	5.4

$\frac{R}{S}$

$\frac{g'}{h'} \leq \frac{g}{h}$

$$\text{obj}(q) = \frac{\left(\sum_{i \in I(q)} g_i\right)^2}{\lambda + \sum_{i \in I(q)} h_i}$$

$$\text{obj}(G, H) = \frac{G^2}{\lambda + H}$$



Evaluate effect on accuracy / comprehensibility

Example (used cars)

optimal base learner

```
+16192 if PS>=100 & year>=2003
+10596 if count<=86 & PS>=180 & year>=2009
- 8360 if PS in [100,180] & year in [2003,2009]
+ 5837 if PS>=180 & year <=2003
+ 2497 if km<=70000
```

greedy base learner

```
+16202 if PS>=100 & year>=2003
+10612 if count<=86 & PS>=180 & year>=2009
+ 9791 if year>=2015
+ 5790 if PS>=180 & year<=2003
- 8414 if PS in [100,180] & year in [2003,2009]
```

greedy finds important rule later than optimal rule boosting

Evaluate effect on accuracy / comprehensibility

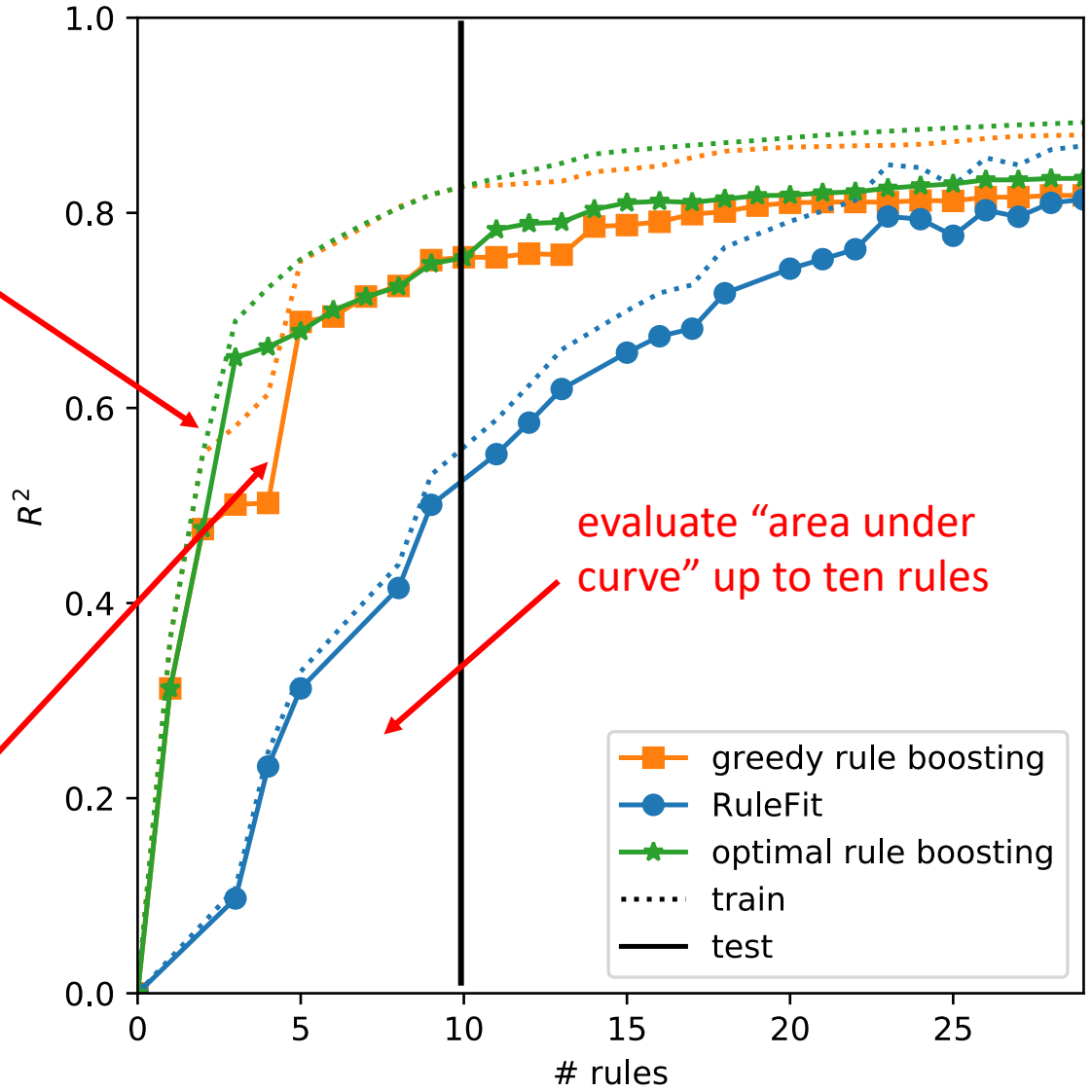
Example (used cars)

optimal base learner

```
+16192 if PS>=100 & year>=2003
+10596 if count<=86 & PS>=180 & year>=2009
- 8360 if PS in [100,180] & year in [2003,2009]
+ 5837 if PS>=180 & year <=2003
+ 2497 if km<=70000
```

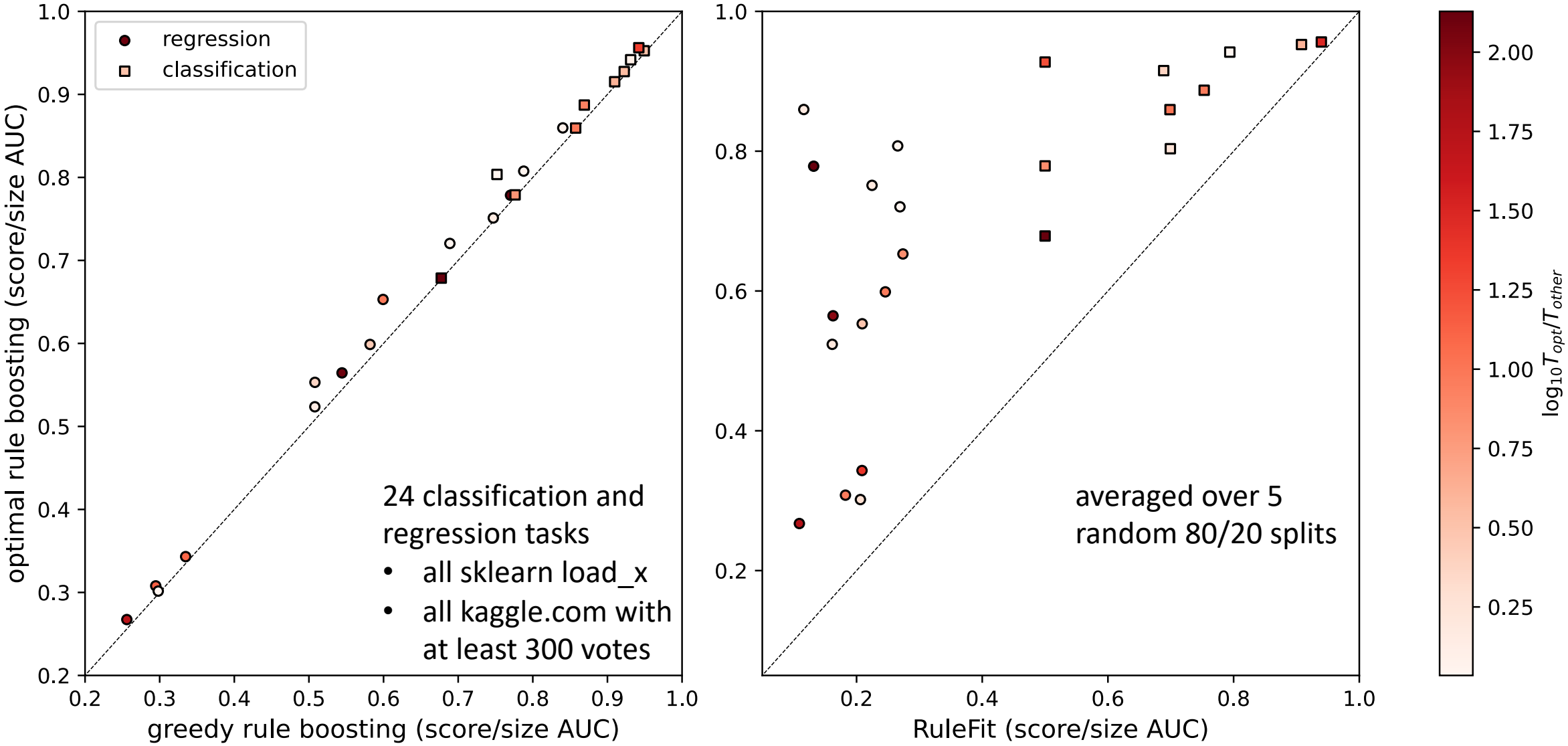
greedy base learner

```
+16202 if PS>=100 & year>=2003
+10612 if count<=86 & PS>=180 & year>=2009
+ 9791 if year>=2015
+ 5790 if PS>=180 & year<=2003
- 8414 if PS in [100,180] & year in [2003,2009]
```



greedy finds important rule later than optimal rule boosting
→ dent in R^2 / # rules curve

Optimal rules consistently outperform myopic rules



Mario Boley, Simon Teshuva, Pierre Le Bodic, Geoff Webb, SDM 2021

To take away…

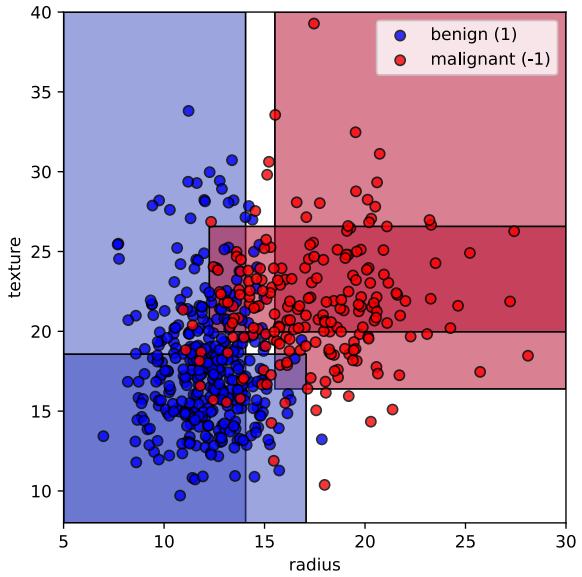
Additive rule ensembles are interpretable models…

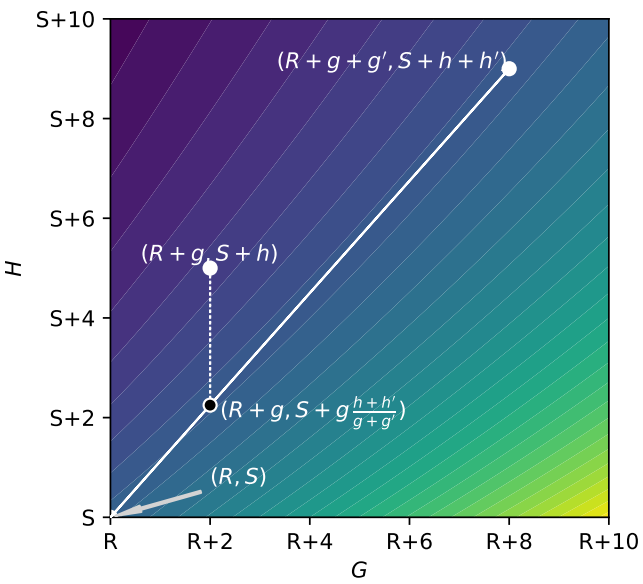
…but myopic search methods compromise interpretability

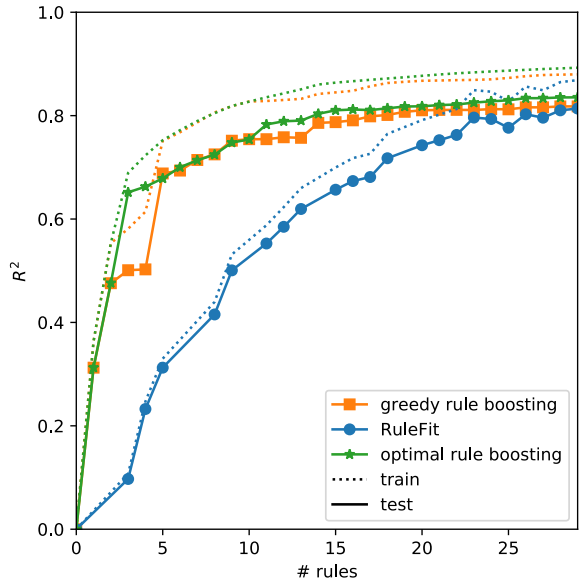
Efficient optimal rule fitting is often possible…

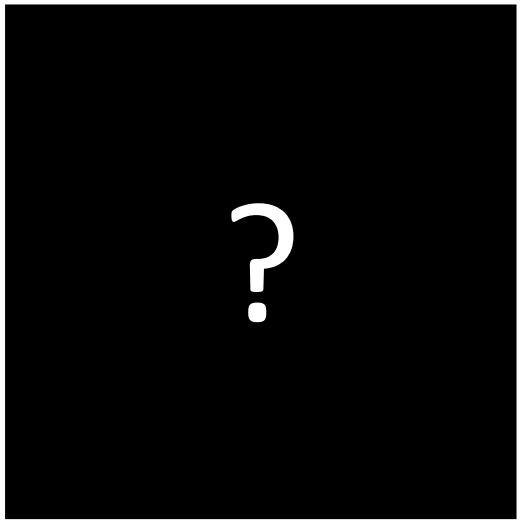
…and results in ensembles that consistently outperform greedy boosting ensembles

Future work: push optimality to ensemble level…









Mario Boley, Simon Teshuva, Pierre Le Bodic, Geoff Webb, SDM 2021