Boosting

Hicham Janati · *INSEA* · hjanati@insea.ac.ma

FSML · based on chapter 7 of the lecture notes

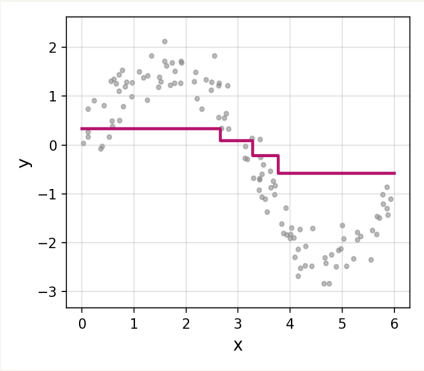
Gradient boosting

Correct the errors, again and again

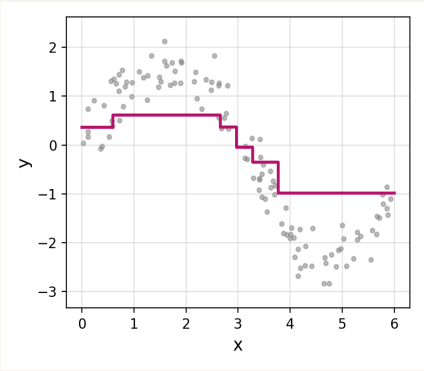
Random forests: **strong** models (deep trees), built **in parallel** and averaged, to reduce the **variance**



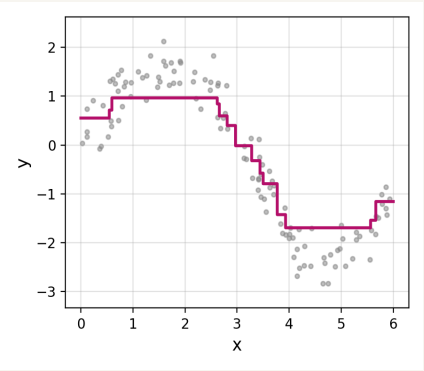
Can we do the opposite ? **Weak** models (shallow trees), built **in sequence**, to reduce the **bias** ?



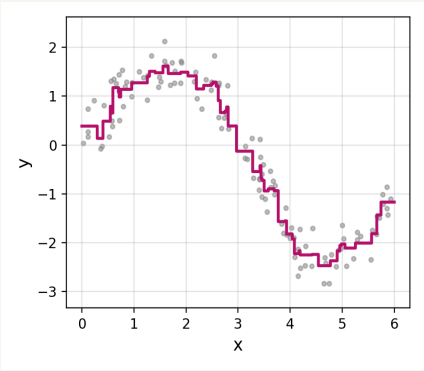
F_1: one small tree



$F_2 = \psi(F_1)$



$F_5 = \psi(F_4)$



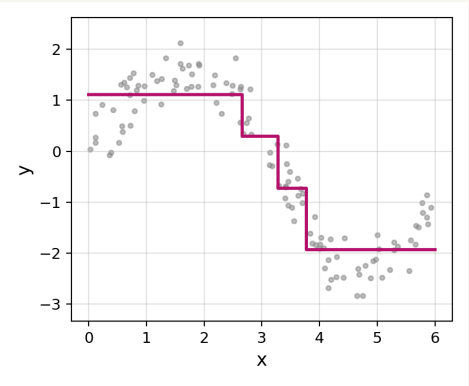
$F_{50} = \psi(F_{49})$

This idea dates back to the 1990s: "Boost" a weak model over and over like a human learner

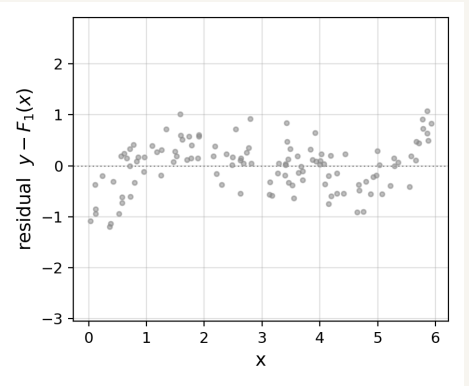
Boosting, step by step

Fit **one small regression tree** F_1 (depth 2) to the data:

F_1 is too simple. Where does it fail ? Look at its **errors**:



F_1: train MSE **0.228**

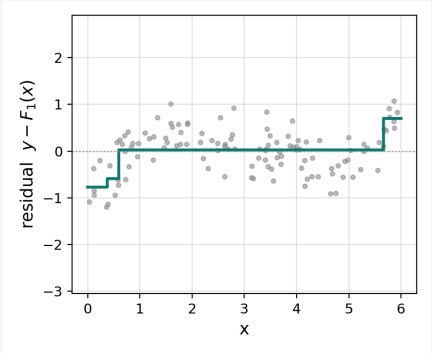


the errors $r_i = y_i - F_1(\mathbf{x}_i)$

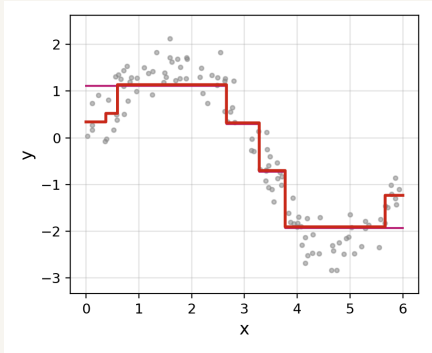
If we could **predict residuals** with a new tree $\hat{T}$, we could correct F_1 :

$$F_2(\mathbf{x}_i) = F_1(\mathbf{x}_i) + \hat{T}(\mathbf{x}_i) \approx F_1(\mathbf{x}_i) + r_i = F_1(\mathbf{x}_i) + y_i - F_1(\mathbf{x}_i) = y_i$$

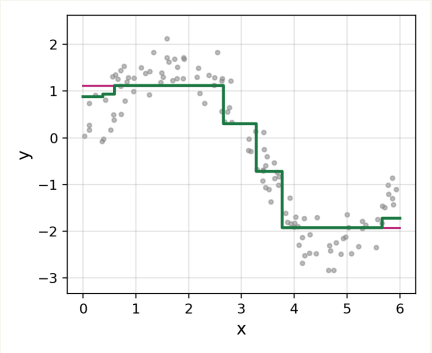
Predict the errors



fit tree$_2$ to the **residuals**



$F_2 = F_1 + \text{tree}_2$



$F_2 = F_1 + \eta\,\text{tree}_2$

Hold on: do we actually trust tree$_2$ to be **perfect** ?

No: tree$_2$ is a noisy guess of the errors. Add it with a **small weight** η: the **learning rate** (step size).

Repeat: $F_m = F_{m-1} + \eta\,\text{tree}_m$, each tree fitted to the residuals of F_{m-1}.

Gradient Boosting Machines

Gradient boosting machines

Consider a regression model F with the **squared loss** $\ell(y, F(\mathbf{x})) = \frac{1}{2}\big(y - F(\mathbf{x})\big)^2$.

F is evaluated on the training data $\mathbf{x}_1, \dots, \mathbf{x}_n$:

$$F(\mathbf{x}) = \begin{pmatrix} F(\mathbf{x}_1) \\ F(\mathbf{x}_2) \\ \vdots \\ F(\mathbf{x}_n) \end{pmatrix}$$

Can we define ξ such that $\ell(y_i, F(\mathbf{x_i}) + \xi(\mathbf{x_i})) \leq \ell(y_i, F(\mathbf{x_i}))$?

Yes ! **Do gradient descent** on $F(\mathbf{x}_i)$: $\xi(\mathbf{x_i}) = -\eta \dfrac{\partial \ell}{\partial F(\mathbf{x}_i)} = \eta \big(y_i - F(\mathbf{x}_i)\big) = \eta\, r_i$

We cannot evaluate residuals on **new** points: so we **train a tree** $\hat{T}$ to **predict them**.

gradient descent on $F(\mathbf{x}_i)$ becomes: $F(\mathbf{x}_i) + \eta\, \hat{T}(\mathbf{x}_i)$

We rediscovered boosting by doing gradient descent on *function outputs* !

Gradient boosting machines, generally

We want a **strong learner** $F^\star$ for a supervised problem with **any** loss ℓ:

$$F^\star = \underset{F}{\operatorname{argmin}} \sum_{i=1}^{n} \ell\big(y_i, F(\mathbf{x}_i)\big)$$

Start from a **weak learner** F_0 and do **gradient descent in function space**:

$$F_m(\mathbf{x}_i) = F_{m-1}(\mathbf{x}_i) - \eta\, g_i, \qquad g_i \overset{\text{def}}{=} \frac{\partial\, \ell\big(y_i, F_{m-1}(\mathbf{x}_i)\big)}{\partial F(\mathbf{x}_i)}$$

we call $-g_i$ the **pseudo-residuals**

They are known on the **training** data only: fit a tree $\hat{T}_m$ to **predict** them.

$F_m = F_{m-1} + \eta\, \hat{T}_m$: a **Gradient Boosting Machine** (Friedman, 2001).

Boosting a classification

How can we use *gradient* boosting on a classification tree with discrete outputs $y_i \in \{0, 1\}$?

Make it a **logistic** regression !

let $F(\mathbf{x}_i) \in \mathbb{R}$, squash it with the **sigmoid**, train it with the **log-loss**

$$p_i = \sigma\big(F(\mathbf{x}_i)\big) = \frac{1}{1 + e^{-F(\mathbf{x}_i)}}, \qquad \ell(y_i, F(\mathbf{x}_i)) = -\,y_i \log p_i - (1 - y_i) \log(1 - p_i)$$

$$\ell(y_i, F(\mathbf{x}_i)) = \log\big(1 + e^{F(\mathbf{x}_i)}\big) - y_i\, F(\mathbf{x}_i) \qquad g_i = \frac{\partial \ell}{\partial F(\mathbf{x}_i)} = p_i - y_i$$

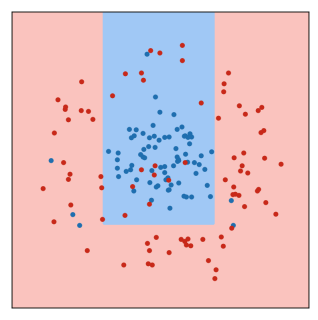
$g_i = p_i - y_i$ is: zero when the model is right, largest when it is **confidently wrong**

GBM recipe: fit a regression tree $\hat{T}$ on $-g_i = y_i - p_i$ and do a descent step: $F_2 = F + \eta \hat{T}$.

Boosting a non-linear boundary

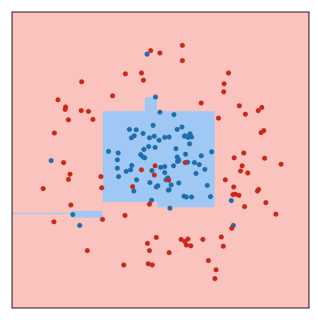
The same recipe on a **2d** classification problem: depth-3 trees, $\eta = 0.3$. How does the boundary evolve with M (the number of trees) ?

train / **test** error



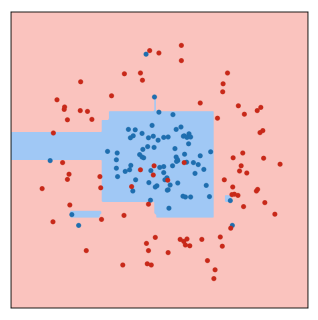
M = 1

0.144 / 0.198



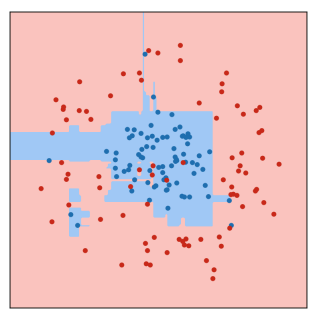
M = 30

0.075 / 0.127



M = 100

0.050 / 0.155



M = 600

0.000 / 0.174

Newton boosting: second-order GBM

GBM does gradient descent on *function outputs*. Why not use **second-order** methods ?

$$\ell\big(y_i,\, F_{m-1}(\mathbf{x}_i) + \xi\big) \;=\; \ell\big(y_i, F_{m-1}(\mathbf{x}_i)\big) + \frac{\partial \ell}{\partial F(\mathbf{x}_i)}\,\xi + \frac{1}{2}\,\frac{\partial^2 \ell}{\partial F(\mathbf{x}_i)^2}\,\xi^2 + o(\xi^2)$$

with the **gradient** $g_i \overset{\text{def}}{=} \partial \ell / \partial F(\mathbf{x}_i)$ and the **Hessian** $h_i \overset{\text{def}}{=} \partial^2 \ell / \partial F(\mathbf{x}_i)^2$

Newton boosting: second-order GBM

GBM does gradient descent on *function outputs*. Why not use **second-order** methods ?

$$\ell\big(y_i,\, F_{m-1}(\mathbf{x}_i) + \xi\big) \;=\; \ell\big(y_i, F_{m-1}(\mathbf{x}_i)\big) + g_i\,\xi + \frac{1}{2}\,h_i\,\xi^2 + o(\xi^2)$$

with the **gradient** $g_i \overset{\text{def}}{=} \partial\ell/\partial F(\mathbf{x}_i)$ and the **Hessian** $h_i \overset{\text{def}}{=} \partial^2\ell/\partial F(\mathbf{x}_i)^2$

We look for the (small) increment $\xi(\mathbf{x})$ that minimizes

$$\sum_i \ell\big(y_i, F_{m-1}(\mathbf{x}_i) + \xi(\mathbf{x}_i)\big) \;\approx\; \text{const} + \sum_i \Big[g_i\,\xi(\mathbf{x}_i) + \tfrac{1}{2}\,h_i\,\xi(\mathbf{x}_i)^2\Big]$$

equivalent to $\displaystyle\min_{\xi} \sum_i \tfrac{1}{2}\,h_i \left(\xi(\mathbf{x}_i) + \frac{g_i}{h_i}\right)^2$

It's a **weighted regression** on the targets $-g_i/h_i$, with weights h_i

We fit a new tree ξ with **this** objective.

First order vs second order

Training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ and a **twice differentiable** loss ℓ, with

$$g_i \overset{\text{def}}{=} \frac{\partial \ell}{\partial F(\mathbf{x}_i)}, \qquad h_i \overset{\text{def}}{=} \frac{\partial^2 \ell}{\partial F(\mathbf{x}_i)^2}$$

Fit **weak learner** F_0 (a constant function) and compute $g_i = \frac{\partial \ell}{\partial F_0(\mathbf{x}_i)}$

↓

1st order GBM

fit a regression tree $\hat{T}_m$ to the **pseudo-residuals** $-g_i$

2nd second order GBM

fit a regression tree $\hat{T}_m$ to the **targets** $-g_i/h_i$, each point **weighted** by h_i

↓

both add the new tree the same way: $F_m = F_{m-1} + \eta \hat{T}_m$, with η a small **learning rate**.

Example: squared error

Regression with the squared loss $\ell(y_i, F_i) = \frac{1}{2}(y_i - F_i)^2$

$$g_i = F_i - y_i, \qquad h_i = 1$$



1st order GBM

fit a tree to $-g_i = y_i - F_i$: the **residual**

2nd order GBM

fit a tree to $-\frac{g_i}{h_i} = y_i - F_i$, weighted by $h_i = 1$

The Hessian is **constant**: for the squared loss the two methods **coincide**.

Example: log-loss

Classification with the log-loss, $p_i = \sigma(F_i)$: $\ell(y_i, F_i) = -y_i \log p_i - (1 - y_i) \log(1 - p_i)$

What are g_i and h_i for the log-loss ?

$$g_i = p_i - y_i, \qquad h_i = p_i(1 - p_i)$$

What does each method fit its tree to ?

1st order GBM

fit a tree to $-g_i = y_i - p_i$: the **probability gap**

2nd order GBM

fit a tree to $-\dfrac{g_i}{h_i} = \dfrac{y_i - p_i}{p_i(1 - p_i)}$, weighted by $h_i = p_i(1 - p_i)$

Here the Hessian **varies**: confident points weigh little, **uncertain** ones ($p_i \approx \frac{1}{2}$) weigh most.

Leaf values in closed form

A tree is **constant on each leaf**: $\xi(\mathbf{x}_i) = w_l$ for every $\mathbf{x}_i \in R_l$. What is the best w_l ?

$$\min_{\xi} \sum_i \left[g_i \, \xi(\mathbf{x}_i) + \tfrac{1}{2} h_i \, \xi(\mathbf{x}_i)^2 \right]$$

Leaf values in closed form

A tree is **constant on each leaf**: $\xi(\mathbf{x}_i) = w_l$ for every $\mathbf{x}_i \in R_l$. What is the best w_l ?

$$\min_{\xi} \sum_i \left[g_i \, \xi(\mathbf{x}_i) + \tfrac{1}{2} h_i \, \xi(\mathbf{x}_i)^2 \right] = \min_{\xi} \sum_{l=1}^{L} \sum_{i \in R_l} \left[g_i \, \xi(\mathbf{x}_i) + \tfrac{1}{2} h_i \, \xi(\mathbf{x}_i)^2 \right]$$

Leaf values in closed form

A tree is **constant on each leaf**: $\xi(\mathbf{x}_i) = w_l$ for every $\mathbf{x}_i \in R_l$. What is the best w_l ?

$$\min_{\xi} \sum_i \left[g_i\, \xi(\mathbf{x}_i) + \tfrac{1}{2} h_i\, \xi(\mathbf{x}_i)^2 \right] = \min_{\xi} \sum_{l=1}^{L} \sum_{i \in R_l} \left[g_i\, \xi(\mathbf{x}_i) + \tfrac{1}{2} h_i\, \xi(\mathbf{x}_i)^2 \right]$$

$$= \min_{w_1, \dots, w_L} \sum_{l=1}^{L} \left[\underbrace{\left(\sum_{i \in R_l} g_i \right)}\, w_l + \tfrac{1}{2} \underbrace{\left(\sum_{i \in R_l} h_i \right)}\, {w_l}^2 \right]$$

Leaf values in closed form

A tree is **constant on each leaf**: $\xi(\mathbf{x}_i) = w_l$ for every $\mathbf{x}_i \in R_l$. What is the best w_l ?

$$\min_{\xi} \sum_i \left[g_i\, \xi(\mathbf{x}_i) + \tfrac{1}{2} h_i\, \xi(\mathbf{x}_i)^2 \right] = \min_{\xi} \sum_{l=1}^{L} \sum_{i \in R_l} \left[g_i\, \xi(\mathbf{x}_i) + \tfrac{1}{2} h_i\, \xi(\mathbf{x}_i)^2 \right]$$

$$= \min_{w_1, \dots, w_L} \sum_{l=1}^{L} \left[\underbrace{\left(\sum_{i \in R_l} g_i \right)}_{G_l} w_l + \tfrac{1}{2} \underbrace{\left(\sum_{i \in R_l} h_i \right)}_{H_l} {w_l}^2 \right]$$

Leaf values in closed form

A tree is **constant on each leaf**: $\xi(\mathbf{x}_i) = w_l$ for every $\mathbf{x}_i \in R_l$. What is the best w_l ?

$$\min_{\xi} \sum_i \left[g_i\, \xi(\mathbf{x}_i) + \tfrac{1}{2} h_i\, \xi(\mathbf{x}_i)^2 \right] = \min_{\xi} \sum_{l=1}^{L} \sum_{i \in R_l} \left[g_i\, \xi(\mathbf{x}_i) + \tfrac{1}{2} h_i\, \xi(\mathbf{x}_i)^2 \right]$$

$$= \min_{w_1, \dots, w_L} \sum_{l=1}^{L} \left[\underbrace{\left(\sum_{i \in R_l} g_i \right)}_{G_l} w_l + \tfrac{1}{2} \underbrace{\left(\sum_{i \in R_l} h_i \right)}_{H_l} {w_l}^2 \right]$$

$$= \min_{w_1, \dots, w_L} \sum_{l=1}^{L} \left[G_l\, w_l + \tfrac{1}{2} H_l\, {w_l}^2 \right]$$

Leaf values in closed form

A tree is **constant on each leaf**: $\xi(\mathbf{x}_i) = w_l$ for every $\mathbf{x}_i \in R_l$. What is the best w_l ?

$$\min_{\xi} \sum_i \left[g_i\, \xi(\mathbf{x}_i) + \tfrac{1}{2} h_i\, \xi(\mathbf{x}_i)^2 \right] = \min_{\xi} \sum_{l=1}^{L} \sum_{i \in R_l} \left[g_i\, \xi(\mathbf{x}_i) + \tfrac{1}{2} h_i\, \xi(\mathbf{x}_i)^2 \right]$$

$$= \min_{w_1,\dots,w_L} \sum_{l=1}^{L} \left[\underbrace{\left(\sum_{i \in R_l} g_i \right)}_{G_l} w_l + \tfrac{1}{2} \underbrace{\left(\sum_{i \in R_l} h_i \right)}_{H_l} {w_l}^2 \right]$$

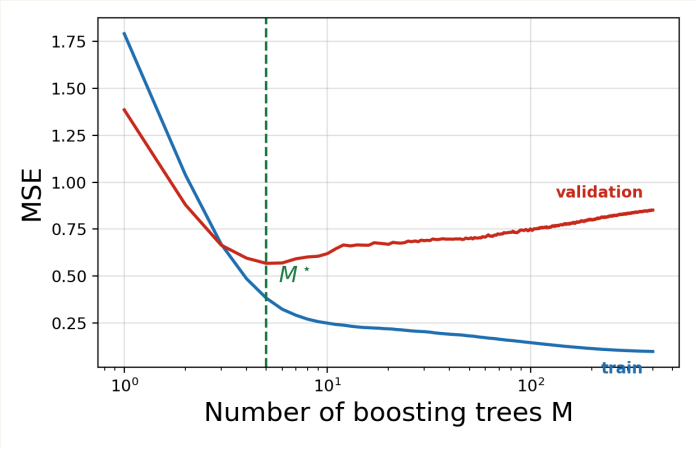
$$= \min_{w_1,\dots,w_L} \sum_{l=1}^{L} \left[G_l\, w_l + \tfrac{1}{2} H_l\, {w_l}^2 \right] = \sum_{l=1}^{L} \min_{w_1,\dots,w_L} \left[G_l\, w_l + \tfrac{1}{2} H_l\, {w_l}^2 \right]$$

Leaf values in closed form: $\quad (\forall\, l = 1..L) \quad w_l^\star = -\dfrac{G_l}{H_l}$

Controlling overfitting

Boosting and overfitting

Random forests never overfits with large number of trees. What about boosting ?



Yes !: with a large number of trees, boosting overfits.

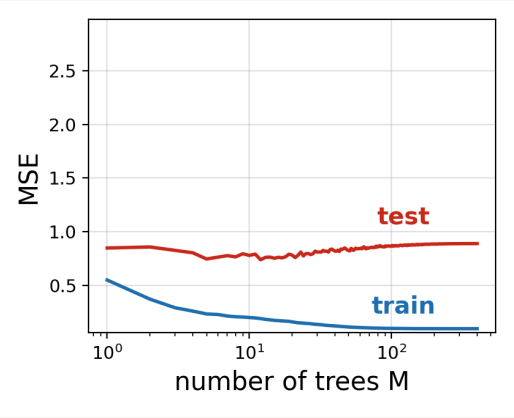
So how do we choose M ?



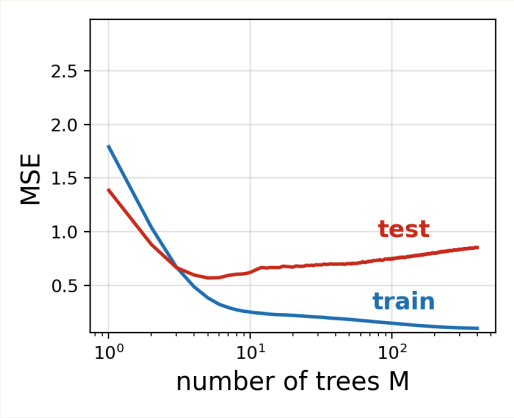
Keep adding trees and evaluating: if test error jumps: stop ! This is called *Early stopping*

The learning rate

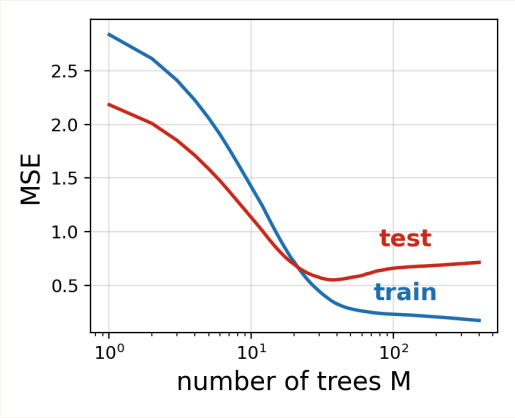
The learning rate controls the descent:



$\eta = 1$



$\eta = 0.3$



$\eta = 0.05$

ℓ_2 regularization

Second change: the objective itself **penalizes the tree**, **L2** on the leaf values $w_1, \dots, w_L$.

$$\text{obj} = \sum_{l=1}^{L} \left[G_l \, w_l + \tfrac{1}{2} H_l \, {w_l}^2 \right] + \tfrac{1}{2} \lambda \sum_{l=1}^{L} {w_l}^2 = \sum_{l=1}^{L} \left[G_l \, w_l + \tfrac{1}{2} \left(H_l + \lambda \right) {w_l}^2 \right]$$

$$w_l^\star = - \frac{G_l}{H_l + \lambda}$$

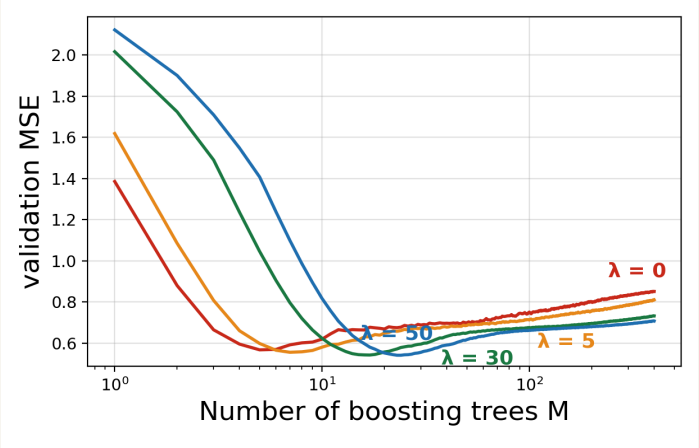
When can the unpenalized $-G_l / H_l$ **blow up** ?

A leaf of only **confidently-wrong** points has $H_l \approx 0$ and $G_l \neq 0$: λ keeps the denominator off zero.

An **L1** variant exists too: $\alpha \sum_l |w_l|$, lasso-style.

ℓ_2 regularization

XGBoost's λ shrinks each leaf value, $w_l^\star = -G_l/(H_l + \lambda)$. What is its effect ?



ℓ_2 penalty **delays and softens** the overfit.

Modern GBDTs (Gradient boosted decision trees)

Modern implementations



Chen & Guestrin

2016



Ke et al., Microsoft

2017



HistGradientBoosting

2019

Eventually converged to similar capabilities, with (some) different engineering decisions. Main ideas:

1. Second-order optimization

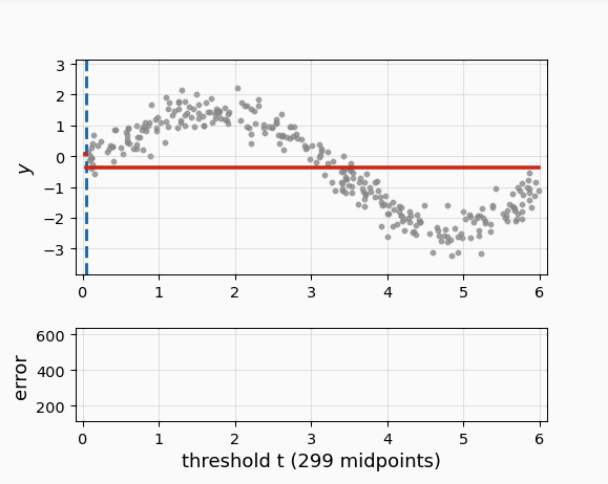
2. Handling missing values (default direction)

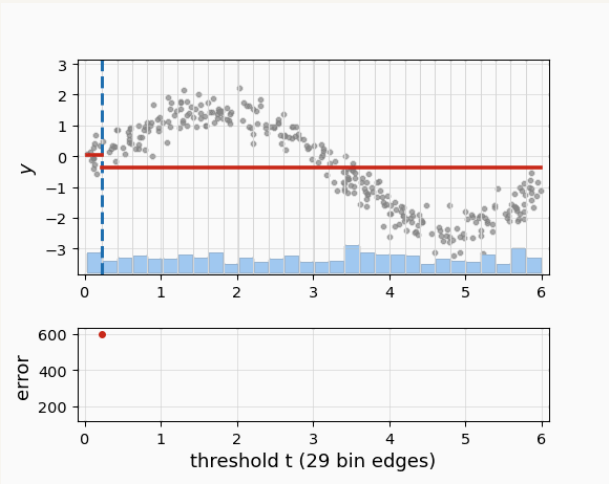
3. Handling big n up to millions of rows with histograms

4. Handling raw categorical variables

Handling large n: histogram binning / feature quantization

One split scan: **sort** the feature, then try **every midpoint**.







Per node: an $O(d\,n\log n)$ sort, then $O(d\,n)$ to scan. With $n \sim$ **millions**, this scan **is** the training time.



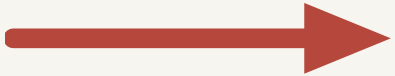
Idea: bin each feature **once** into a histogram, then only the bin edges are candidates.

Categorical features ?

How can a tree handle a categorical variable like this one ?

profession
student
engineer
doctor
entrepreneur
⋮

One hot encoding



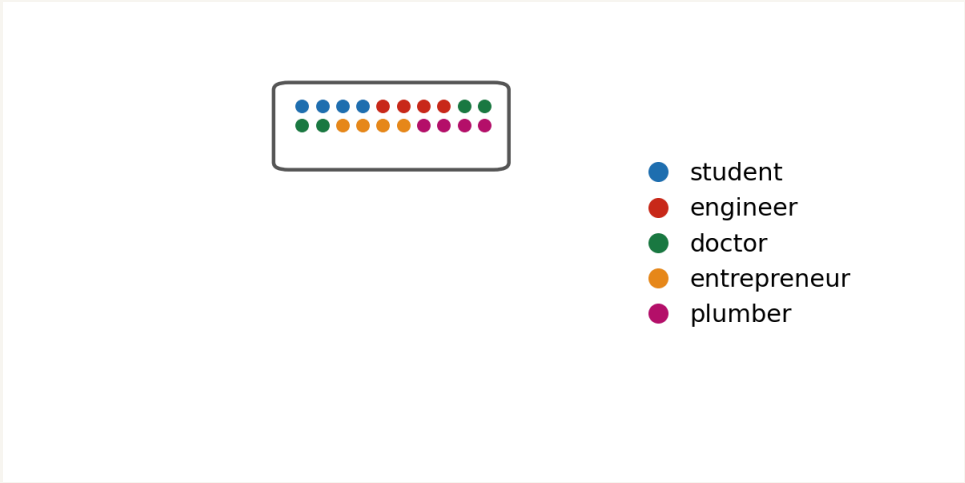
stud.	eng.	doc.	entr.	plum.
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
⋮	⋮	⋮	⋮	⋮

What is the **problem** with this ?

Can we do better ?

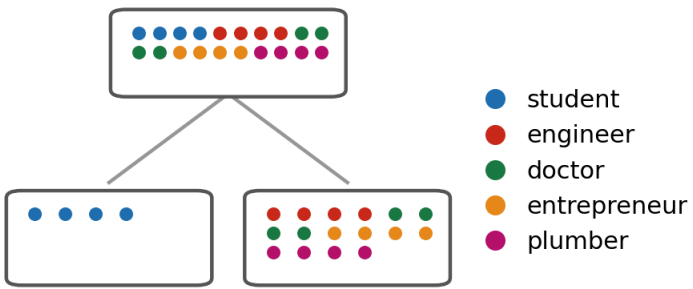
Categorical features: naive splits ?

Keep the column as it is and splits create subgroups of categories:



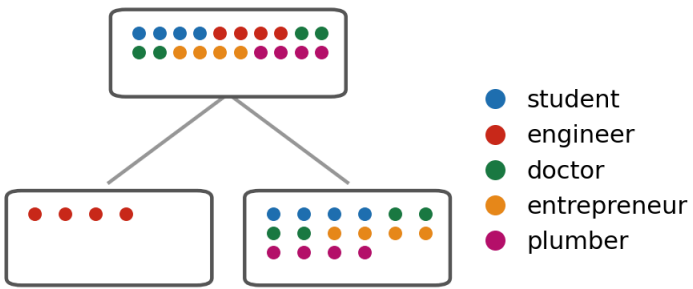
Categorical features: naive splits ?

Keep the column as it is and splits create subgroups of categories:



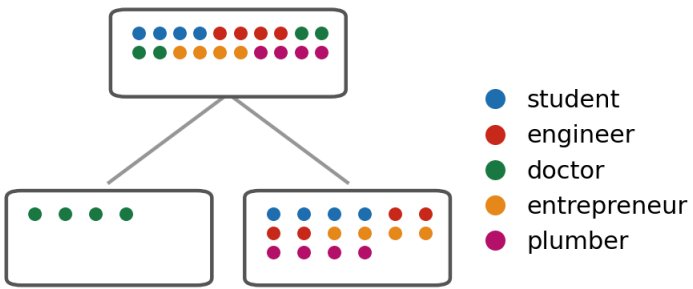
Categorical features: naive splits ?

Keep the column as it is and splits create subgroups of categories:



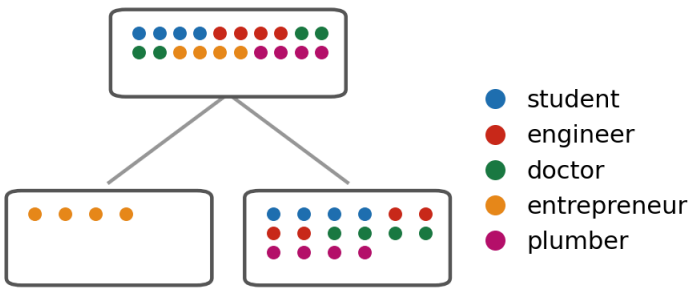
Categorical features: naive splits ?

Keep the column as it is and splits create subgroups of categories:



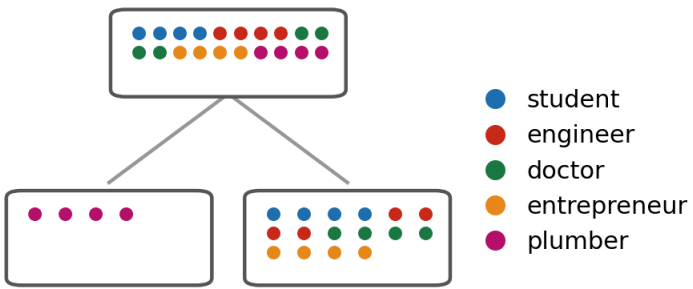
Categorical features: naive splits ?

Keep the column as it is and splits create subgroups of categories:



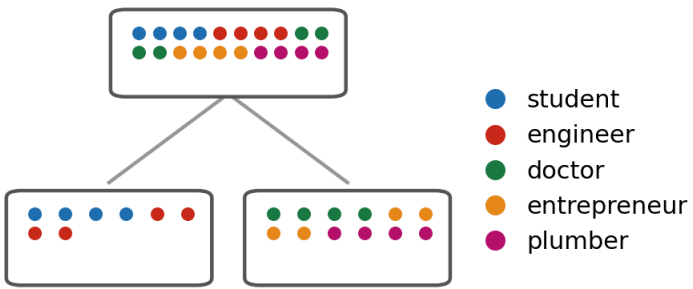
Categorical features: naive splits ?

Keep the column as it is and splits create subgroups of categories:



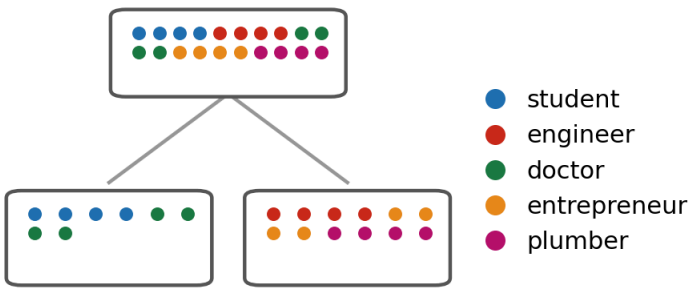
Categorical features: naive splits ?

Keep the column as it is and splits create subgroups of categories:



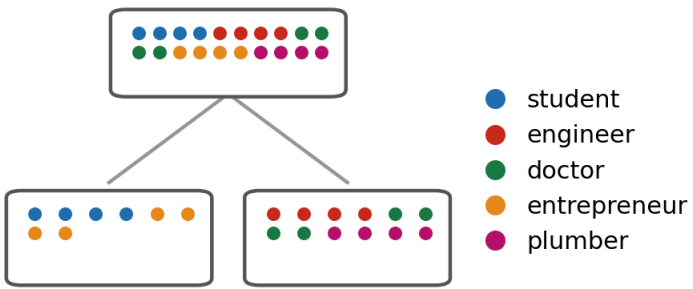
Categorical features: naive splits ?

Keep the column as it is and splits create subgroups of categories:



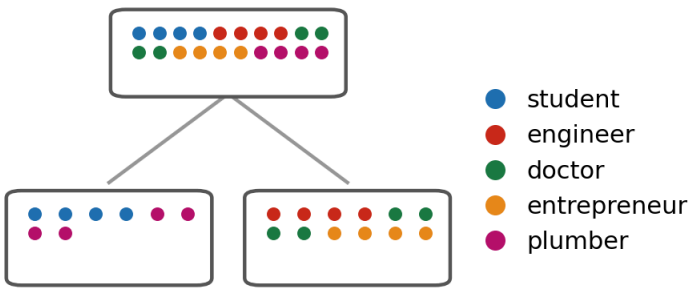
Categorical features: naive splits ?

Keep the column as it is and splits create subgroups of categories:



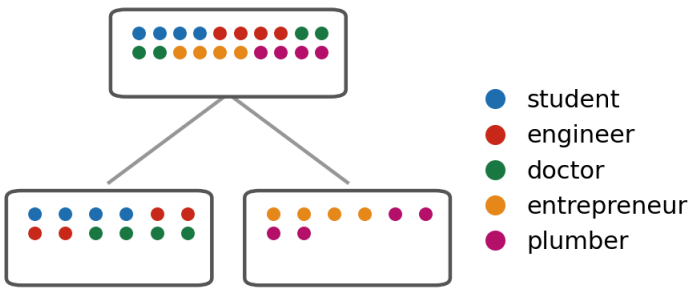
Categorical features: naive splits ?

Keep the column as it is and splits create subgroups of categories:



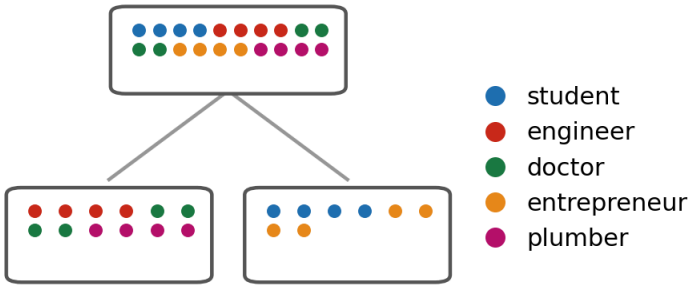
Categorical features: naive splits ?

Keep the column as it is and splits create subgroups of categories:



Categorical features: naive splits ?

Keep the column as it is and splits create subgroups of categories:



How many such splits are there ?



$$2^{K-1} - 1 \text{ !}$$

- $K = 5 \rightarrow 15$
- $K = 20 \rightarrow 524\,287$

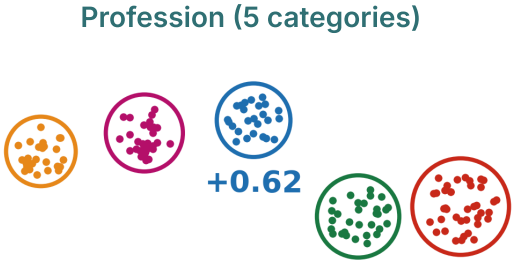
Categorical features: sorted by a score



Compute each category's **"ideal"** leaf value

$$-\frac{G_c}{H_c+\lambda} = -\frac{\sum_{x_i=c} g_i}{\sum_{x_i=c} h_i + \lambda}$$

Categorical features: sorted by a score



Compute each category's **"ideal"** leaf value

$$-\frac{G_c}{H_c+\lambda} = -\frac{\sum_{x_i=c} g_i}{\sum_{x_i=c} h_i+\lambda}$$

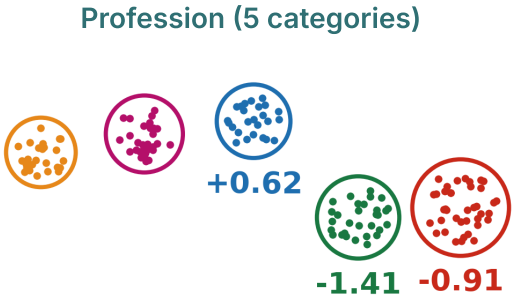
Categorical features: sorted by a score



Compute each category's **"ideal"** leaf value

$$-\frac{G_c}{H_c+\lambda} = -\frac{\sum_{x_i=c} g_i}{\sum_{x_i=c} h_i + \lambda}$$

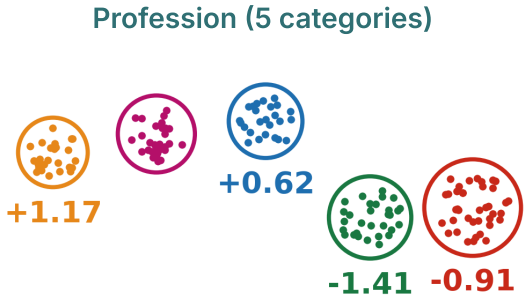
Categorical features: sorted by a score



Compute each category's **"ideal"** leaf value

$$-\frac{G_c}{H_c+\lambda} = -\frac{\sum_{x_i=c} g_i}{\sum_{x_i=c} h_i + \lambda}$$

Categorical features: sorted by a score



Compute each category's **"ideal"** leaf value

$$-\frac{G_c}{H_c+\lambda} = -\frac{\sum_{x_i=c} g_i}{\sum_{x_i=c} h_i+\lambda}$$

Categorical features: sorted by a score



Compute each category's **"ideal"** leaf value

$$-\frac{G_c}{H_c+\lambda} = -\frac{\sum_{x_i=c} g_i}{\sum_{x_i=c} h_i+\lambda}$$

Categorical features: sorted by a score



Compute each category's **"ideal"** leaf value

$$-\frac{G_c}{H_c+\lambda} = -\frac{\sum_{x_i=c} g_i}{\sum_{x_i=c} h_i + \lambda}$$

Categorical features: sorted by a score



Compute each category's **"ideal"** leaf value

$$-\frac{G_c}{H_c+\lambda} = -\frac{\sum_{x_i=c} g_i}{\sum_{x_i=c} h_i+\lambda}$$

Categorical features: sorted by a score



Compute each category's **"ideal"** leaf value

$$-\frac{G_c}{H_c+\lambda} = -\frac{\sum_{x_i=c} g_i}{\sum_{x_i=c} h_i+\lambda}$$

Categorical features: sorted by a score



Compute each category's **"ideal"** leaf value

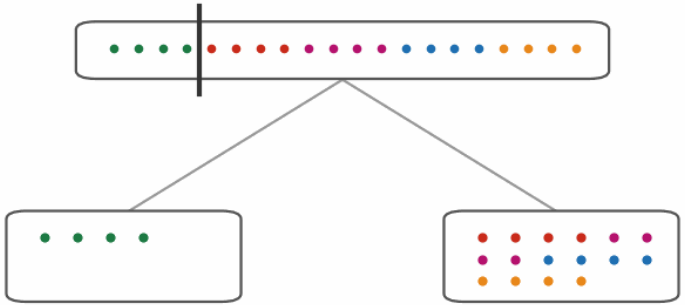
$$-\frac{G_c}{H_c+\lambda} = -\frac{\sum_{x_i=c} g_i}{\sum_{x_i=c} h_i+\lambda}$$

Categorical features: sorted by a score



Compute each category's **"ideal"** leaf value

$$-\frac{G_c}{H_c+\lambda} = -\frac{\sum_{x_i=c} g_i}{\sum_{x_i=c} h_i + \lambda}$$



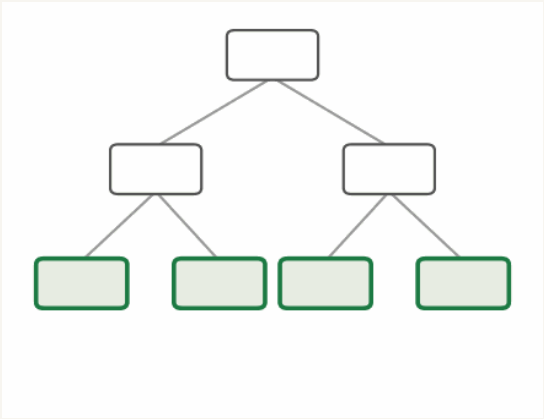


$K-1$ splits instead of $2^{K-1}-1$!

Notable differences: how trees are grown

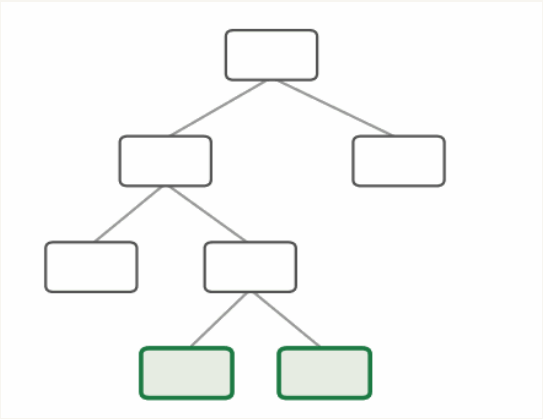
How the trees are constructed:

XGBoost: level-wise



Split the leaves level by level

LightGBM: leaf-wise



Split the **highest-gain leaf**

Leaf-wise reaches lower error with fewer leaves, but **quicker to overfit** on small data.

What to remember

1. **Boosting** (1990s): a **sequence** of weak learners: $F_m = F_{m-1} + \eta\xi_m$
2. **GBM** (2001): formalized as **gradient descent in function space**: fit ξ_m to $-\partial\ell/\partial F(\mathbf{x}_i)$
3. **Modern GBDTs** (2016); trees as the weak learner; XGBoost, LightGBM add **Newton** steps and efficient implementations with **native** handling of **missing values** and **categorical** variables

---

4. They **overfit in the number of trees** M ; chosen by **early stopping** on a validation curve
5. Require tuning the learning rate η with **cross-validation**
6. They can be **regularized**: λ (L2) or α (L1) on the leaf values, chosen by **cross-validation**

GBDTs are **very competitive** on tabular datasets, even against deep neural nets.

Contents

1. Gradient boosting
2. Gradient Boosting Machines
3. Controlling overfitting
4. Modern GBDTs (Gradient boosted decision trees)