

Classical supervised learning

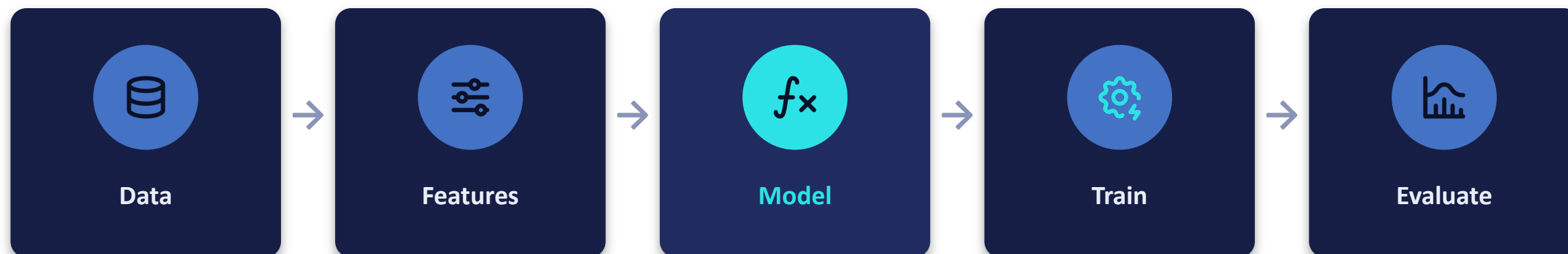
Linear models to boosted decision trees



The honest workhorse of physics analysis — and a baseline deep nets must earn the right to beat

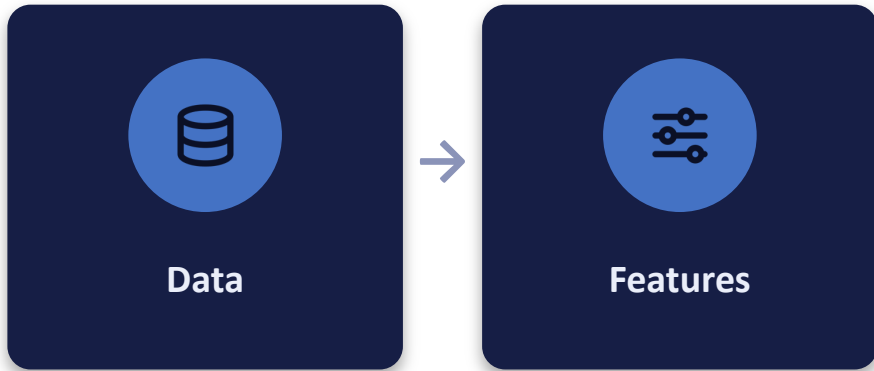
We are filling in the model box

$$\theta^* = \operatorname{argmin}_{\theta} \quad (1/n) \sum_i \ell(f_{\theta}(\mathbf{x}_i), y_i) \quad \text{empirical risk minimization}$$



This hour = the model box. Which family of functions f_{θ} do we search, and how? We start with a line and end with a forest.

Representing the data



So what you use in $f_{\theta}(x_i)$, y_i is:

x_i is a d -dim vector (one row)
 y_i is a scalar (the label/value)
 i indicates the instance (event)

- HEP data are typically provided in form of ntuples
- ML input data are $n \times d$ matrices X with n “instances” (e.g. collision events) and d “features” (variables)
- Example: $Z \rightarrow l^+ l^-$ event, measure leptons pT and pseudorapidity η predict invariant mass of the 2 lepton system: m_{inv}

η_1	η_2	pT1	pT2	m_{inv}
0.7	-0.4	45	38	81
0	1	76	51	62
1	1	47	63	55

n

d

....

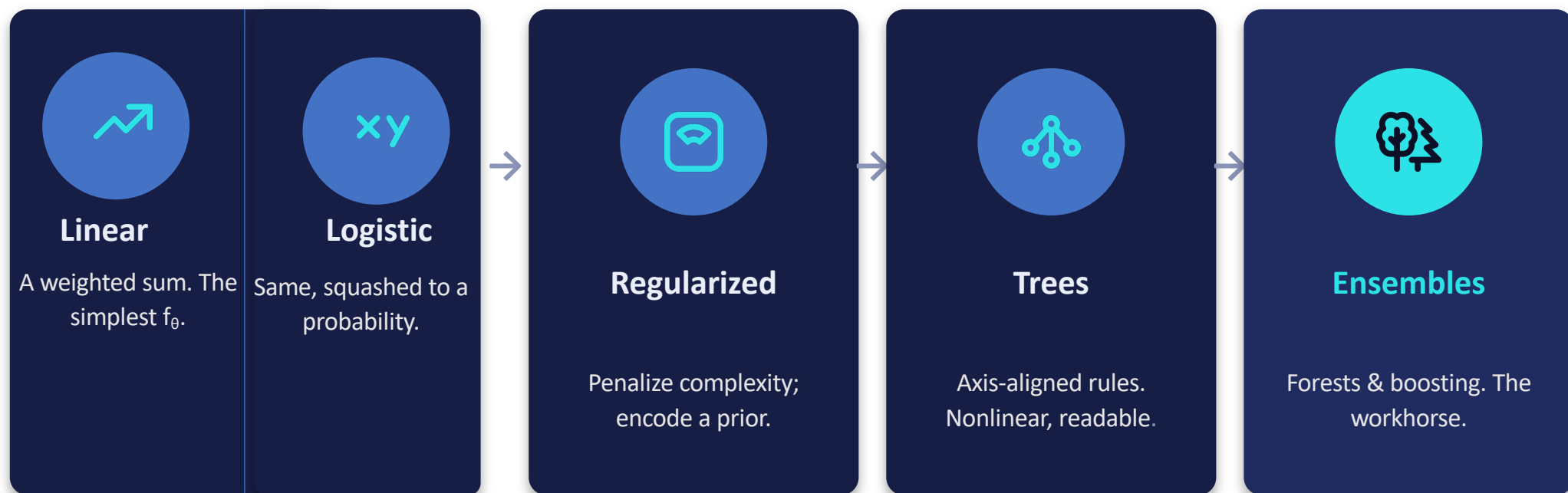
data matrix X

True value/label: y

one event $i (x_i, y_i)$

From a line to a forest

concepts, each adding one idea



Increasing model flexibility, left to right — and, unmanaged, increasing variance. Complexity is earned, not assumed.

Linear regression



A weighted sum of features

$$f_{\theta}(x_i) = w \cdot x_i + b$$

Loss: mean squared error

$$\frac{1}{2} \sum (y_i - f_{\theta}(x_i))^2$$

— which, recall, is the Gaussian negative log-likelihood.

w is d-dim.vector; one weight per input feature

b is a scalar bias

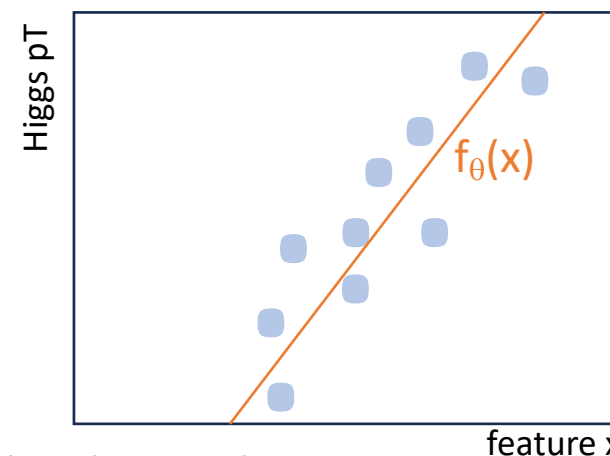
the free parameters are $\theta = (w, b)$

features				target
#electrons	#muons	Jet pT	b-jet pT	Higgs pT
1	0	45	35	40
0	1	76	51	62
1	1	47	63	55

x_i

y_i

$$f_{\theta}(x_i) = w_1 x_{i1} + w_2 x_{i2} + w_3 x_{i3} + w_4 x_{i4} + b$$



*Shown for one feature;
with all d features the line
becomes a hyperplane.*

Linear regression



A weighted sum of features

$$\hat{f}_{\theta}(x_i) = w \cdot x_i + b$$

Loss: mean squared error

$$\frac{1}{2} \sum (y_i - \hat{f}_{\theta}(x_i))^2$$

— which, recall, is the Gaussian negative log-likelihood.

Two ways to fit it

● Normal equation

$w = (X^T X)^{-1} X^T y$. Exact, closed-form — but the matrix inverse costs $O(d^3)$ and hates collinearity.

● Gradient descent

Step downhill on the loss. Scales to huge d and n , and is the same engine every later model uses.

Gradient descent

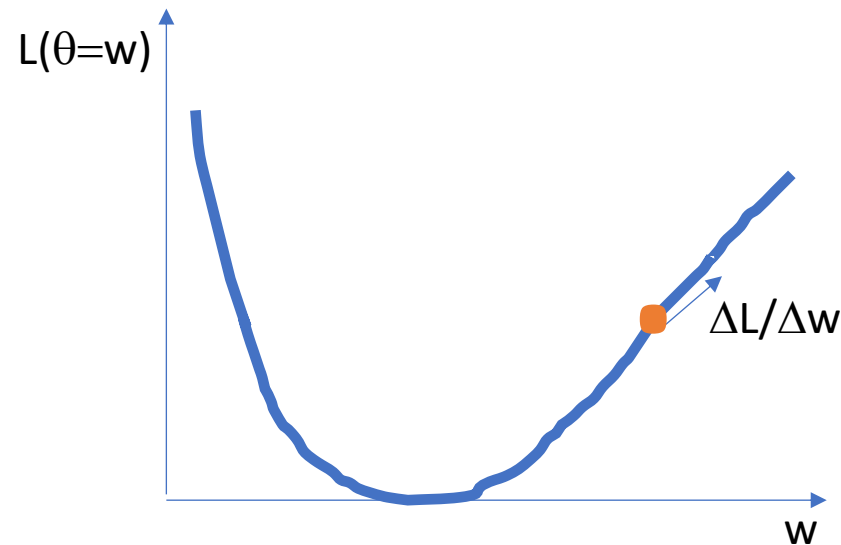
$$\theta \leftarrow \theta - \eta \nabla_{\theta} L(\theta) \quad \text{step against the gradient; } \eta \text{ is the learning rate}$$

$\theta = (w, b)$: the same rule updates every parameter simultaneously

with one parameter, the gradient is just the slope :

Choose initial w

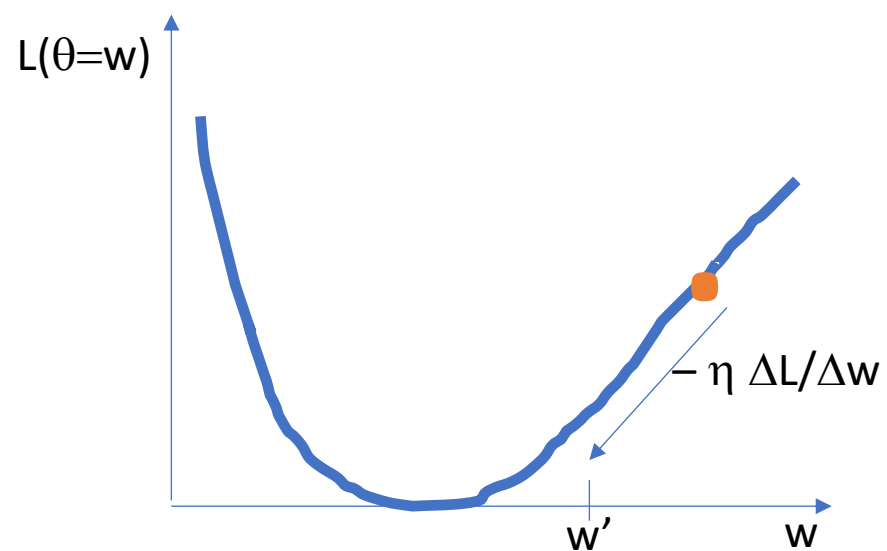
Calculate update: $w' = w - \eta \Delta L / \Delta w$



This way to minimum! => “-”

Gradient descent

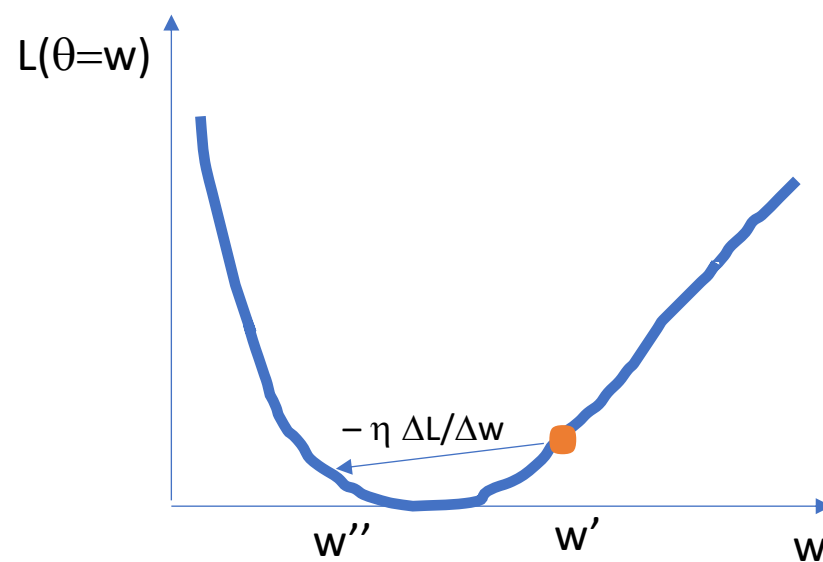
$$\theta \leftarrow \theta - \eta \nabla_{\theta} \mathcal{L}(\theta) \quad \text{step against the gradient; } \eta \text{ is the learning rate}$$



$$w' = w - \eta \Delta L / \Delta w$$

Gradient descent

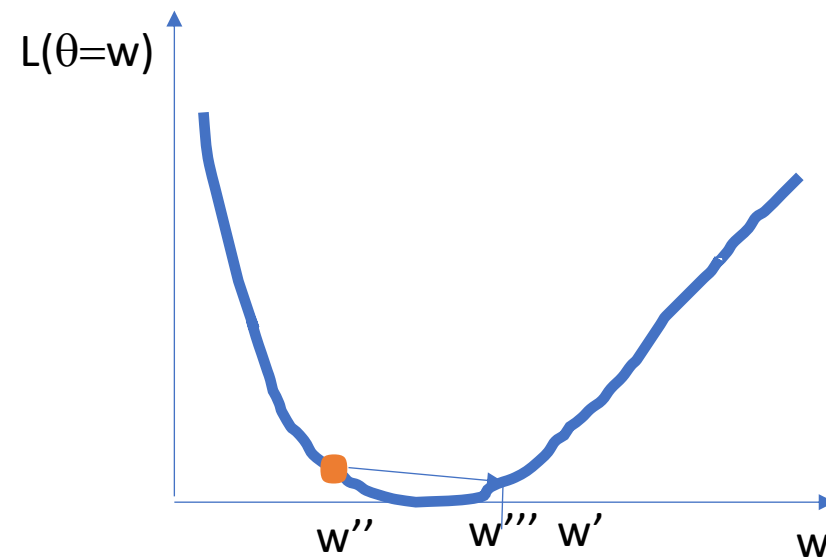
$$\theta \leftarrow \theta - \eta \nabla_{\theta} L(\theta) \quad \text{step against the gradient; } \eta \text{ is the learning rate}$$



$$w'' = w' - \eta \Delta L / \Delta w$$

Gradient descent

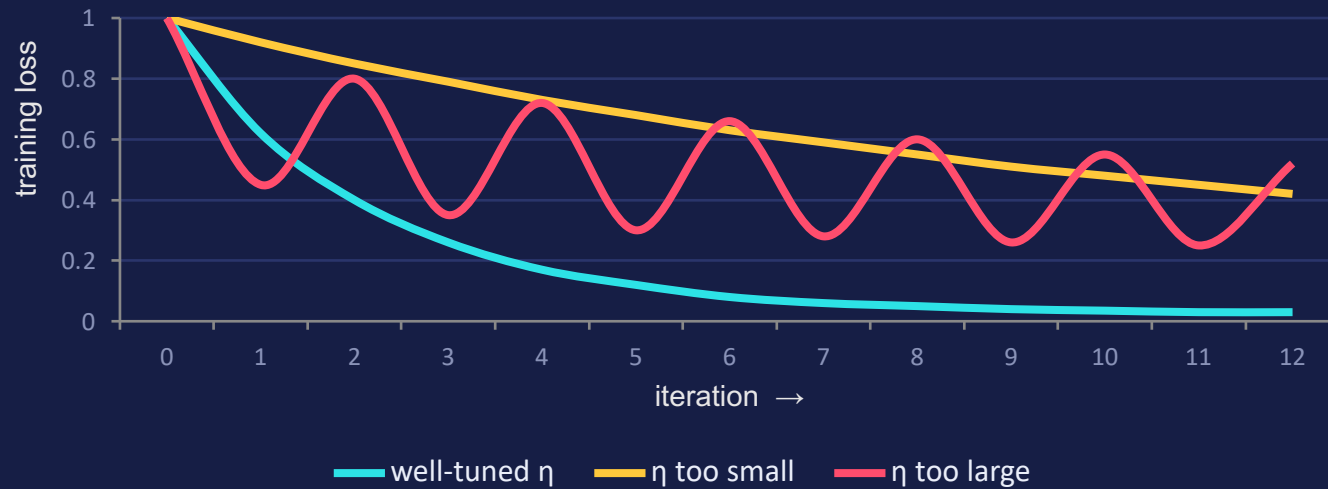
$$\theta \leftarrow \theta - \eta \nabla_{\theta} L(\theta) \quad \text{step against the gradient; } \eta \text{ is the learning rate}$$



$$w''' = w'' - \eta \Delta L / \Delta w$$

Gradient descent

$$\theta \leftarrow \theta - \eta \nabla \theta \mathcal{L}(\theta) \quad \text{step against the gradient; } \eta \text{ is the learning rate}$$



The learning rate decides everything

Too small

Converges, but wastes your compute budget.

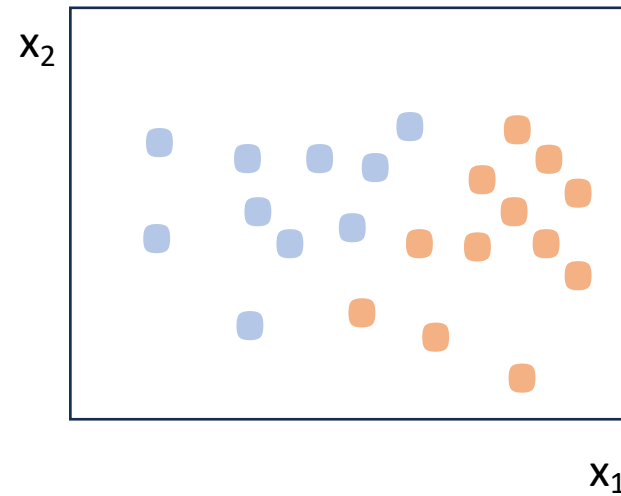
Too large

Overshoots and oscillates — or diverges.

Variants

Momentum & Adam adapt the step; more in Course 3.

Logistic regression



Logistic regression

xy

Squash the linear score to (0, 1)

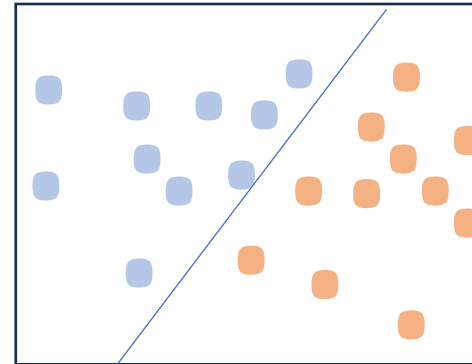
$$p_i = \sigma(w \cdot x_i + b) = 1 / (1 + e^{-(w \cdot x_i + b)})$$

- the sigmoid function σ

Loss: binary cross-entropy

$$L = -\sum_i [y_i \log p_i + (1-y_i) \log(1-p_i)]$$

— the Bernoulli negative log-likelihood.



Why it matters here

● Calibrated output

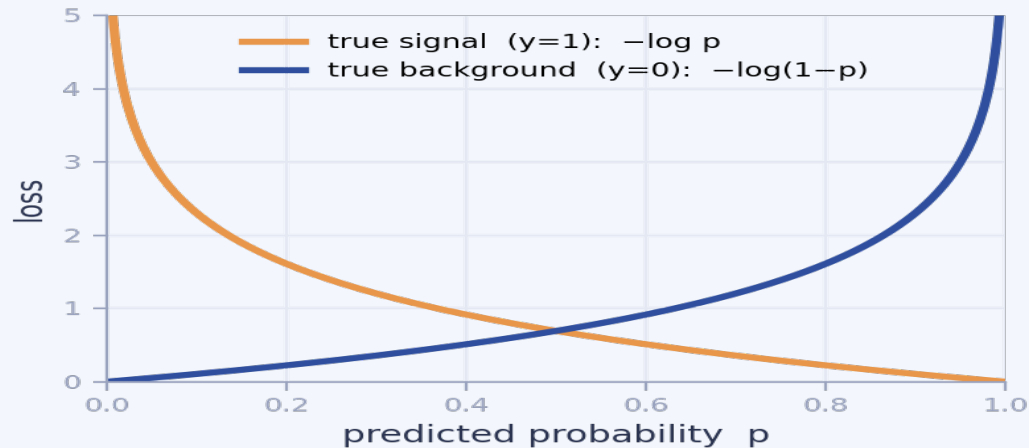
The score is a probability, not just a label — essential when you cut on it to define a signal region.

● Linear boundary

The decision surface is a hyperplane. Simple, and often not enough — motivating everything that follows.

How wrong is a predicted probability? Log loss

$$L_i = - [y_i \cdot \log p_i + (1-y_i) \cdot \log(1-p_i)]$$

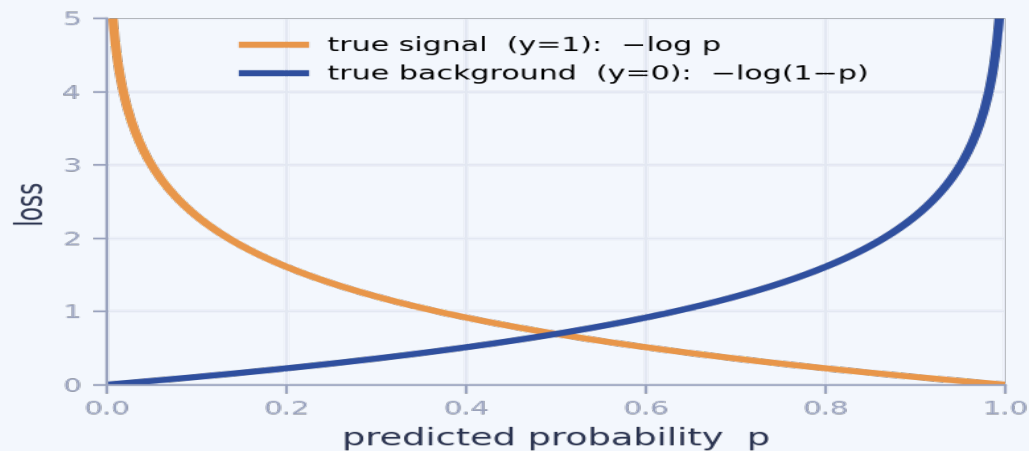


Confident and wrong is punished without bound — $p \rightarrow 0$ for a true signal sends the loss to infinity.

score $s_i \rightarrow p_i = \sigma(s_i) \rightarrow \text{loss } L_i$

How wrong is a predicted probability? Log loss

$$L_i = - [y_i \cdot \log p_i + (1-y_i) \cdot \log(1-p_i)]$$



Confident and wrong is punished without bound — $p \rightarrow 0$ for a true signal sends the loss to infinity.

Its gradient is the residual

score $s_i \rightarrow p_i = \sigma(s_i) \rightarrow \text{loss } L_i$

$$\frac{\partial L}{\partial w} = (p - y) x$$

$$-\partial L_i / \partial s = y_i - p_i$$

- true signal $y=1$, $p = 0.40$ **residual +0.60**
big push to raise w
- true signal $y=1$, $p = 0.95$ **residual +0.05**
tiny nudge. Already right, so barely move.
- true background, $p = 0.30$ **residual -0.30**
moderate push the other way, lowering w



Every event carries a residual $y_i - p_i$ — one continuous number. We will use those residuals later in the boosting tree.

Regularization: L2 and L1

Underdetermination:

features are collinear or more features than events are provided

-> the least-squares problem is ill-posed — infinitely many weight vectors fit equally well, the solution is non-unique and wildly unstable, and coefficients blow up.

-> select a handful of relevant features with the help of regularization of the weights

$$w = (w_1, w_2, \dots, w_n)$$

L2:

$$\|w\|_2 = \sqrt{w_1^2 + w_2^2 + \dots + w_n^2}$$

L1:

$$\|w\|_1 = |w_1| + |w_2| + \dots + |w_n|$$

$$R(w) = L1 \text{ or } L2$$

$$\text{minimize } \mathcal{L}(w) + \lambda \cdot \mathcal{R}(w)$$

$\mathcal{L}(w)$ = the loss you already descend $\lambda \cdot \mathcal{R}(w)$ = the penalty (this slide)

Regularization: L2 and L1

Underdetermination:

features are collinear or more features than events are provided
 -> the least-squares problem is ill-posed — infinitely many weight vectors fit equally well, the solution is non-unique and wildly unstable, and coefficients blow up.
 -> select a handful of relevant features with the help of regularisation

Highly correlated features

feature	$L(w)$	$L(w) + \lambda \ w\ ^2$	$L(w) + \lambda \ w\ $
lead_jet_pT	+2.32	+0.58	0.00
HT	-1.40	+0.57	+0.23
sumjetE	+0.83	+0.58	+1.32
MET	+1.18	+1.12	+0.97
dphi_jj	+0.56	+0.53	+0.31
n_trk	-0.04	-0.04	0.00
evt_phi	0.00	0.00	0.00
pileup	+0.15	+0.14	0.00
Σ (energy trio)	1.75	1.73	1.55

minimize $L(w)$ + $\lambda \cdot R(w)$

$L(w)$ = the loss you already descend $\lambda \cdot R(w)$ = the penalty (this slide)

Regularization: L2 and L1

minimize $\mathcal{L}(\mathbf{w}) + \lambda \cdot \mathcal{R}(\mathbf{w})$

$\mathcal{L}(\mathbf{w})$ = the loss you already descend $\lambda \cdot \mathcal{R}(\mathbf{w})$ = the penalty (this slide)

L2 · Ridge

$$\|w\|_2 = \sqrt{w_1^2 + w_2^2 + \dots + w_n^2}$$

$+ \lambda \|\mathbf{w}\|^2 \Leftrightarrow$ Gaussian prior on $\mathbf{w}$



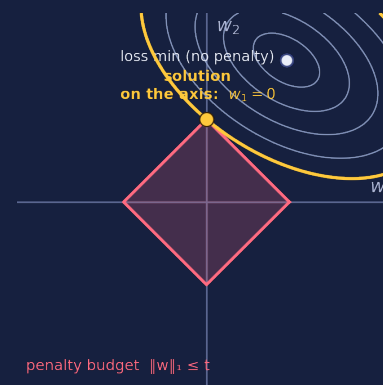
Shrinks weights smoothly.

Nothing forced to zero — every feature stays, just damped. Best when many features each contribute a little (and for correlated inputs).

L1 · Lasso

$$\|w\|_1 = |w_1| + |w_2| + \dots + |w_n|$$

$+ \lambda \|\mathbf{w}\|_1 \Leftrightarrow$ Laplace prior on $\mathbf{w}$



Corners on the axes drive weights to exactly zero.

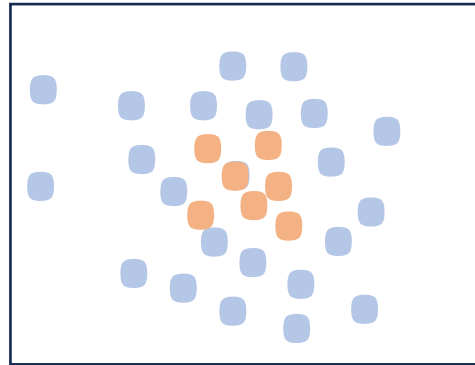
So it both shrinks and prunes $\rightarrow$ feature selection. Best when most features are irrelevant.

LASSO = Least Absolute Shrinkage and Selection Operator

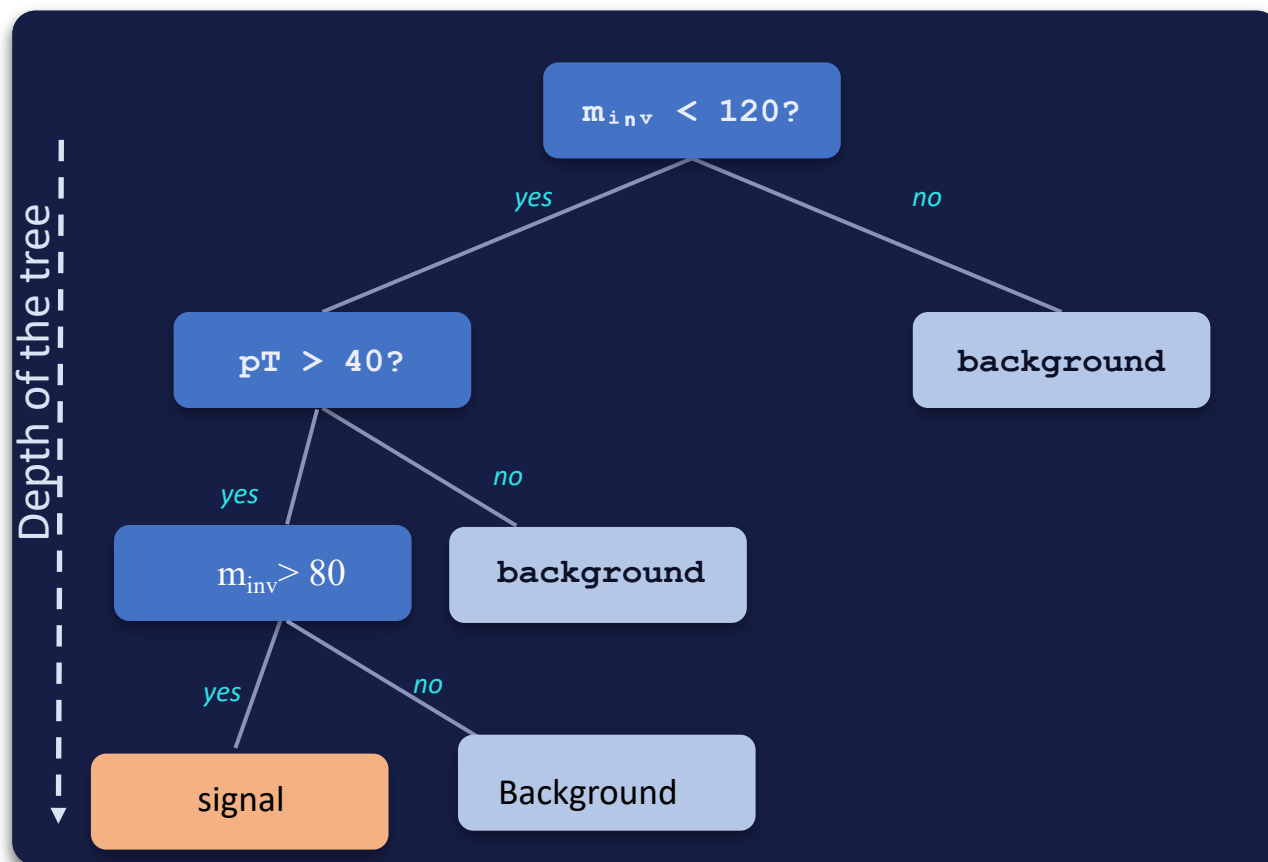


The MAP view (maximum a posteriori): a penalty is a prior. Minimizing $\mathcal{L}(\mathbf{w}) + \lambda \cdot \mathcal{R}(\mathbf{w})$ = maximizing the posterior (likelihood $\times$ prior). λ sets how strongly you believe the prior.

Where learning a linear function does not help



Decision trees



How a tree learns

- **Greedy splits**

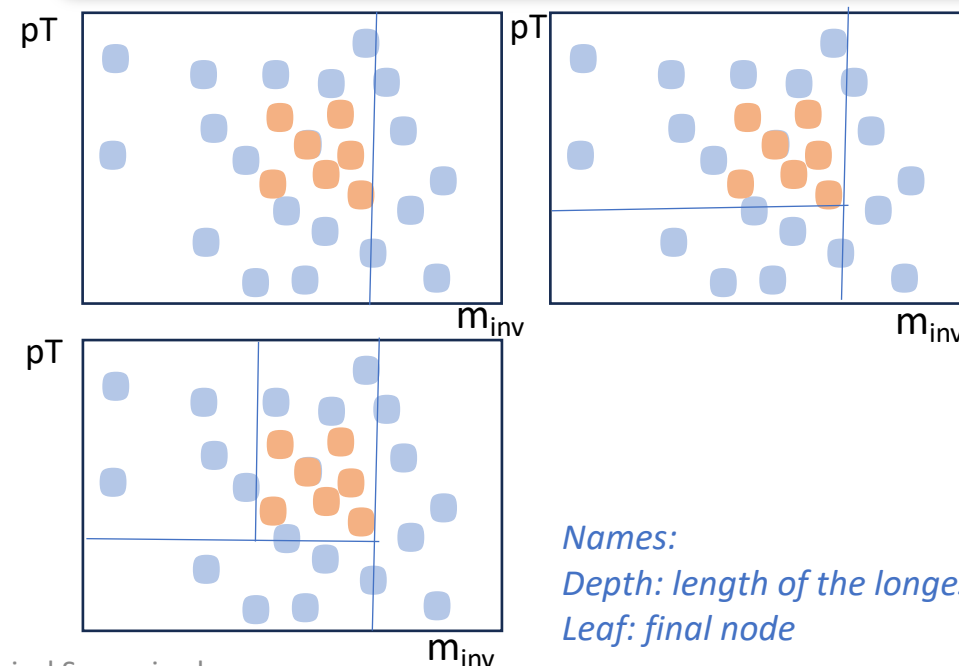
At each node pick the feature and threshold that most reduces impurity.

- **Axis-aligned**

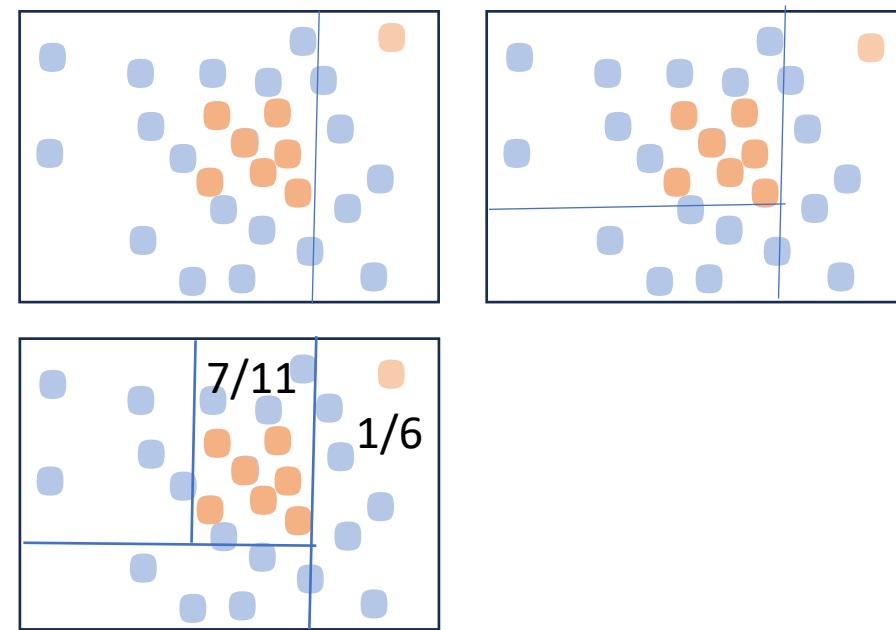
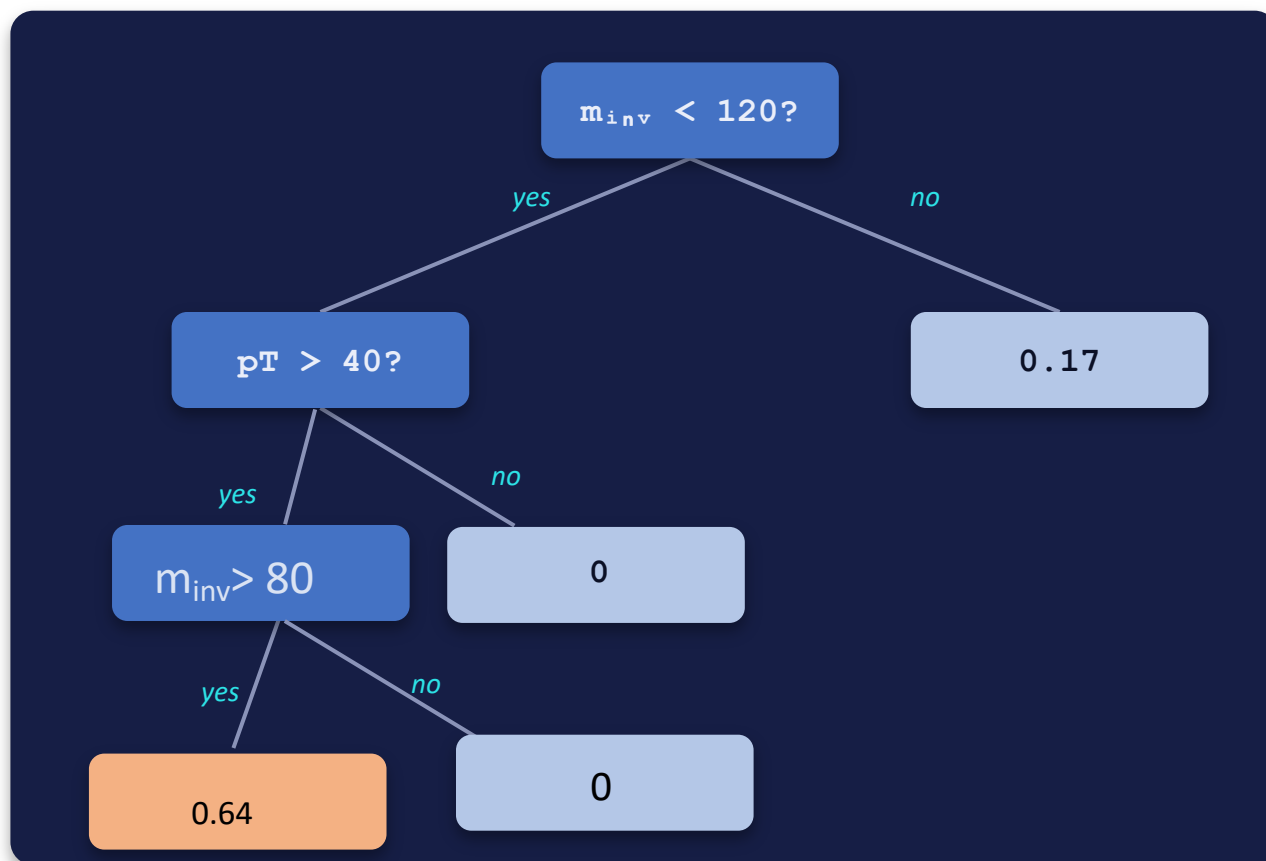
Each cut is on one variable. Rectangular regions in feature space; no rescaling needed.

- **Readable**

You can print the path an event took. Rare and valuable in a physics result.

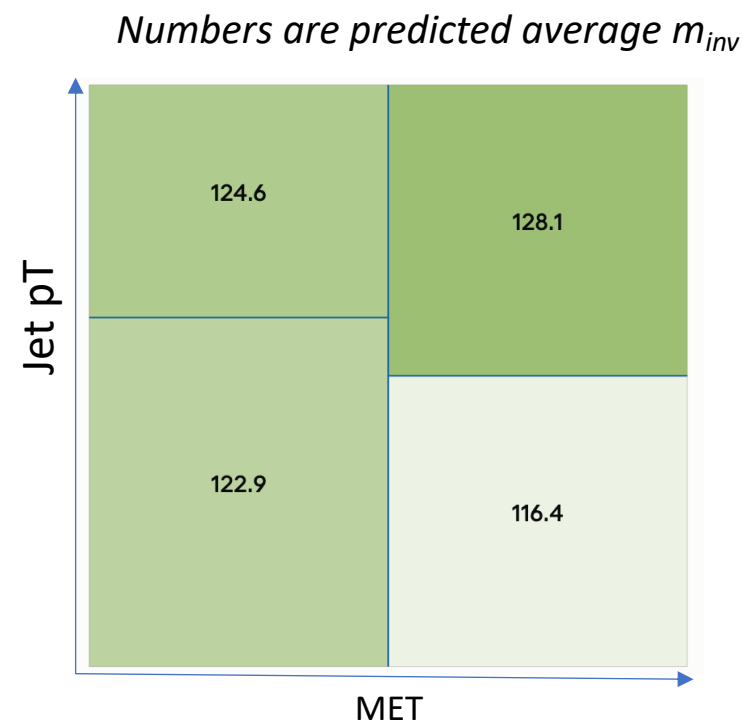
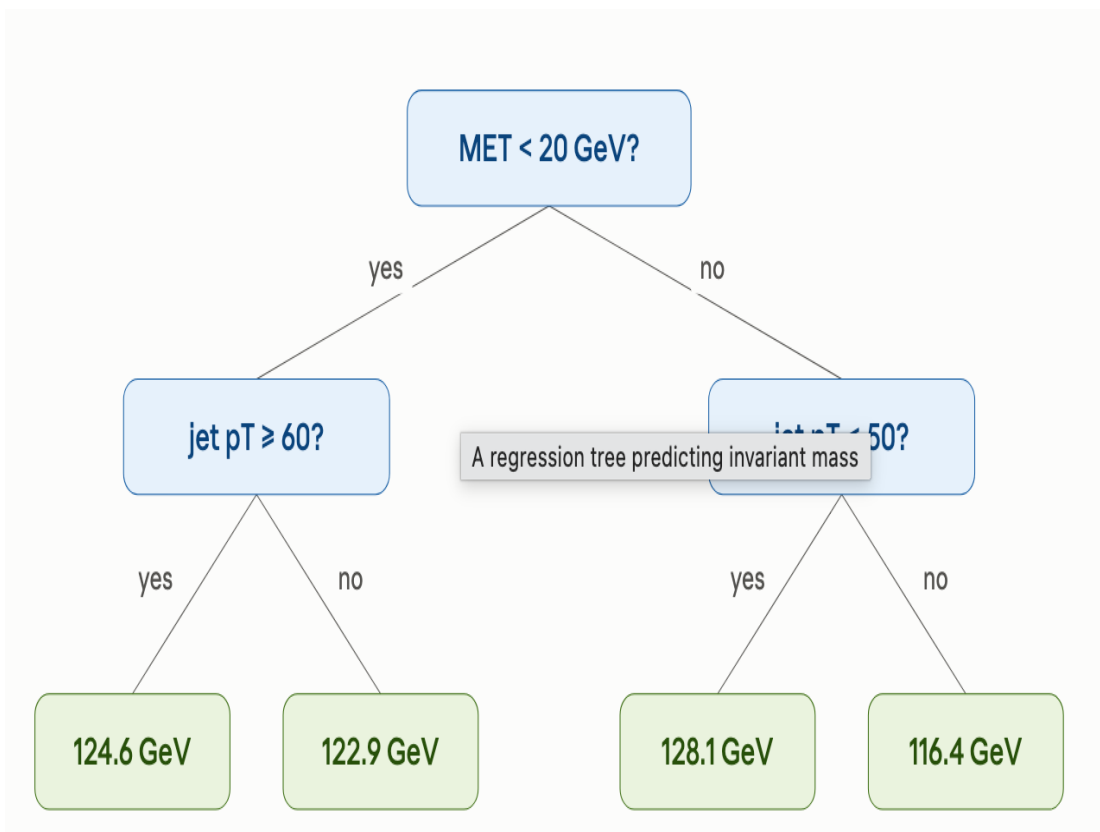


A leaf is a number, not a verdict



The leaf isn't really "signal," it's the fraction $\text{signal}/(\text{signal}+\text{background})$ in that box — a continuous number between 0 and 1 representing the **probability**.

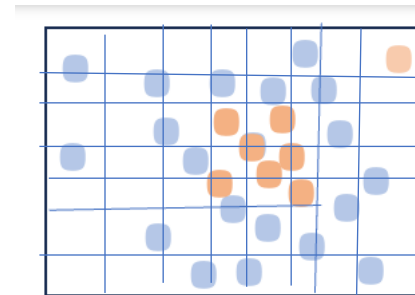
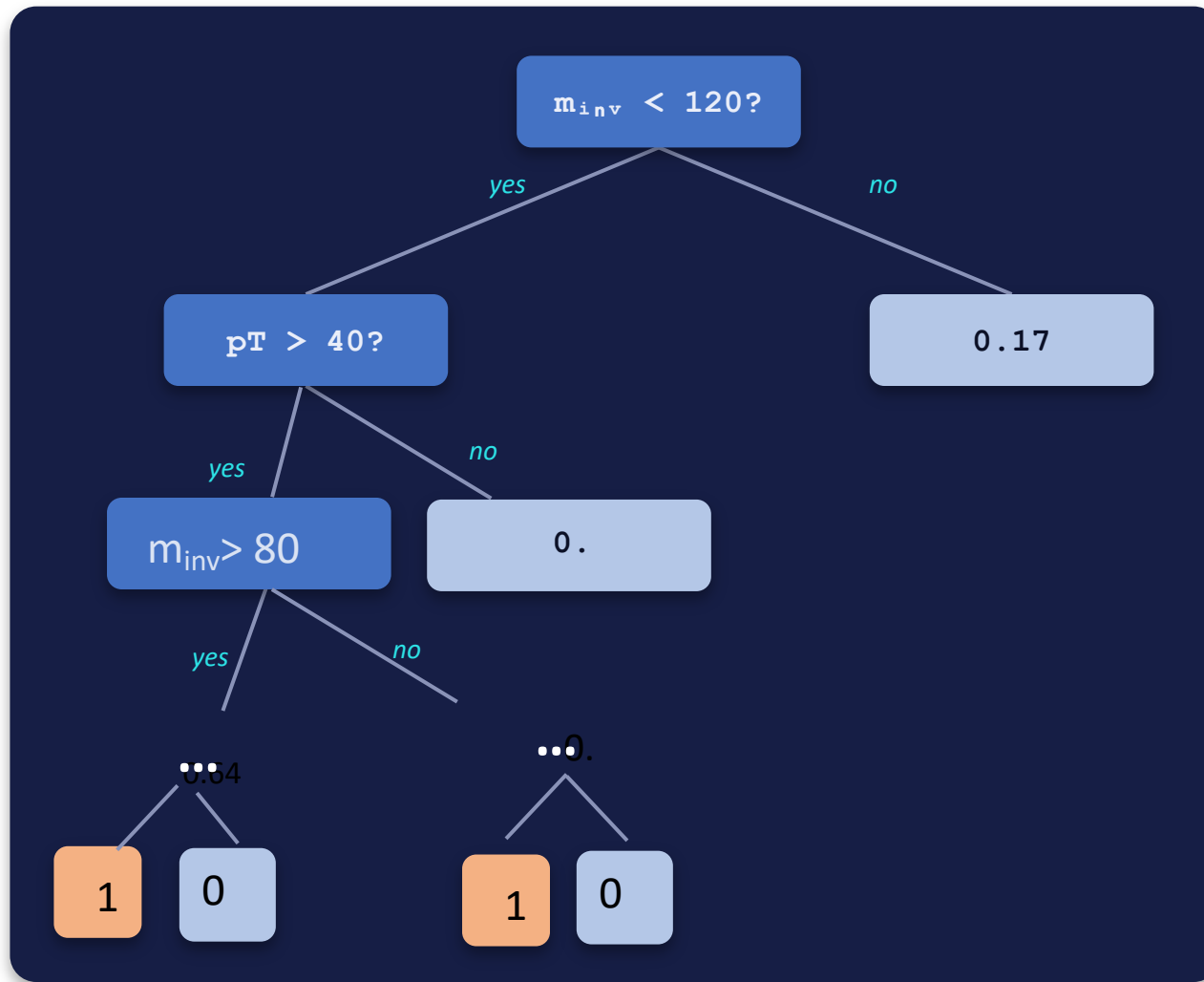
Tree also works for regression



$$\text{MSE} = (1/N) \sum_{\text{events}} (m_{inv}(\text{event}) - \text{mean of its leaf})^2$$

The leaf isn't a probability — it's the **mean m_{inv} (GeV)** of the events in that box. Splits are chosen to shrink the spread of m_{inv} within each box, and the tree partitions the other features, never the mass itself.

One tree is not enough



● High variance

A single deep tree overfits badly — resample the data and it changes wholesale. This is the problem ensembles fix.

Random forests: “bagging” (bootstrap aggregating)



Two sources of randomness

● Bootstrap rows

Each tree trains on a resampled copy of the data.

● Random features

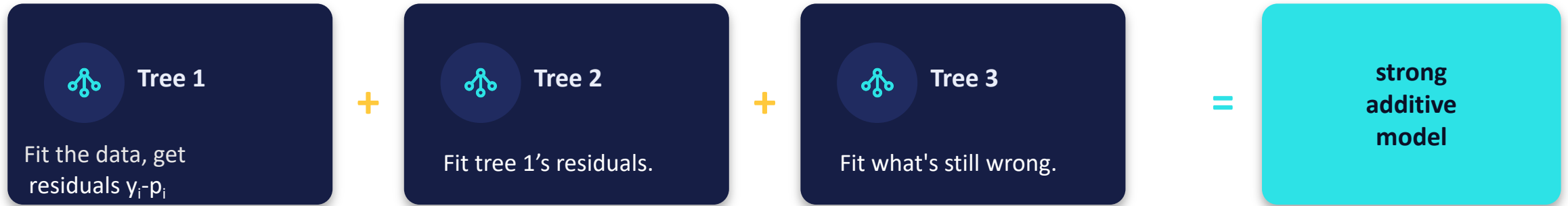
Each split sees only a random subset of variables — this decorrelates the trees.

● Then average

Averaging decorrelated, low-bias trees cuts variance without adding bias.

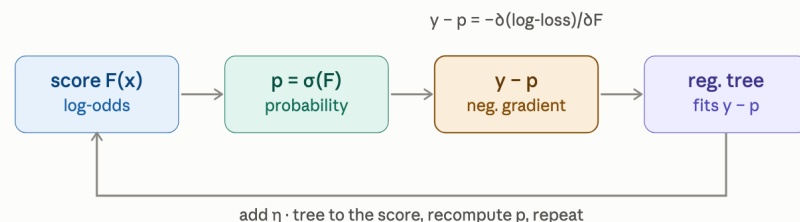
Trees vote in parallel and independently — trivially parallelizable, hard to overfit.

Gradient boosting



$$F(x) = F_0 + \eta \cdot \text{tree}_1(x) + \eta \cdot \text{tree}_2(x) + \eta \cdot \text{tree}_3(x) + \dots$$

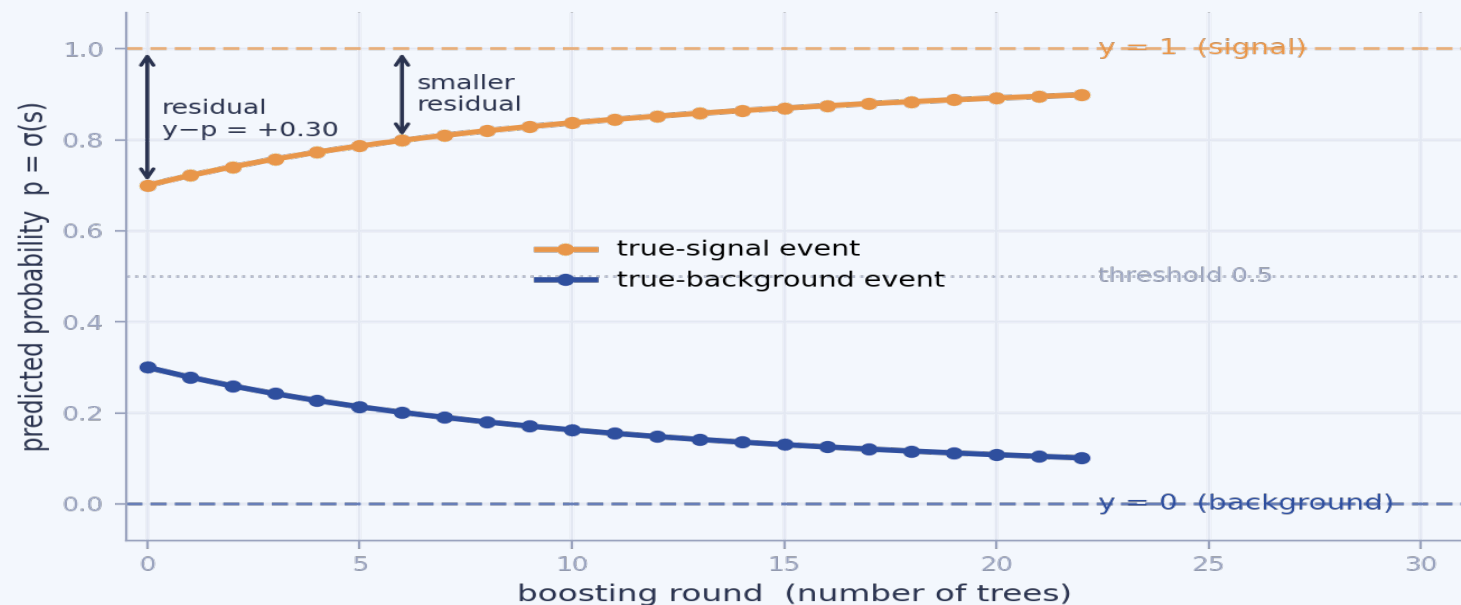
$$F_0 = \log \frac{\bar{y}}{1 - \bar{y}}, \quad \bar{y} = \frac{\# \text{signal}}{\# \text{total}}.$$



The idea

Build the model additively, one small tree at a time. Each **new tree is fit to the gradient of the loss** — the direction that most reduces current error. A learning rate shrinks each step, so the ensemble improves gradually rather than overfitting in one leap.

How one event's probability moves



The signal event, round by round

round	p	$y - p$
0	0.70	+0.30
1	0.72	+0.28
2	0.74	+0.26
5	0.79	+0.21
10	0.84	+0.16
22	0.90	+0.10
∞	1.00	0.00

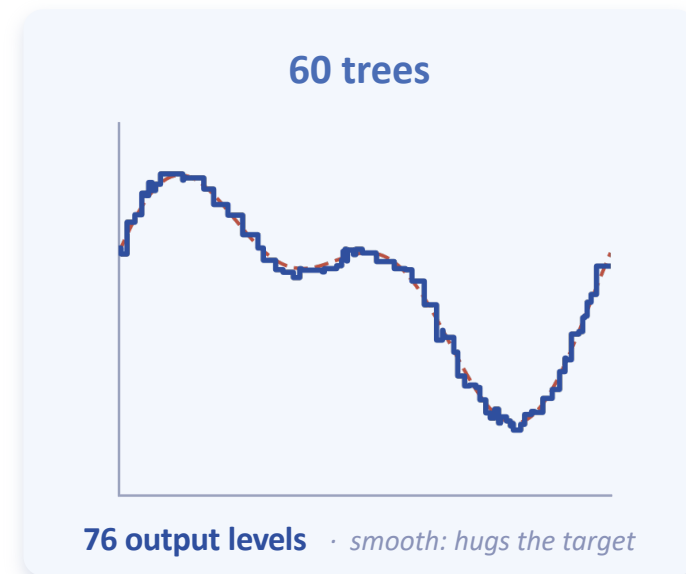
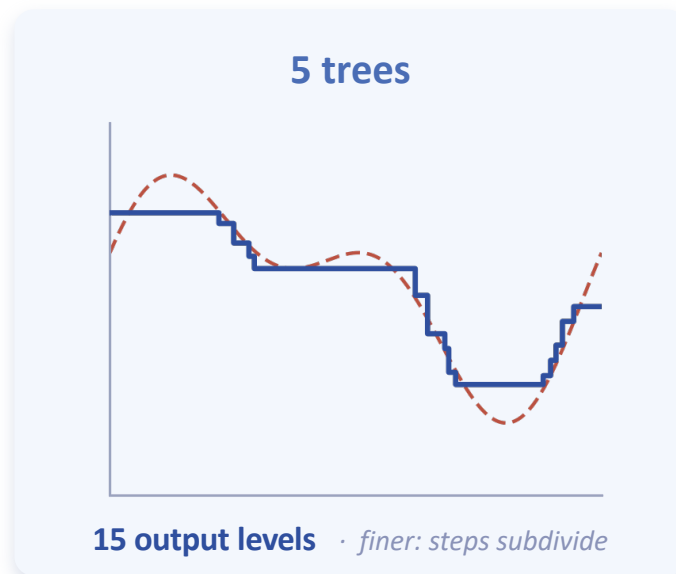
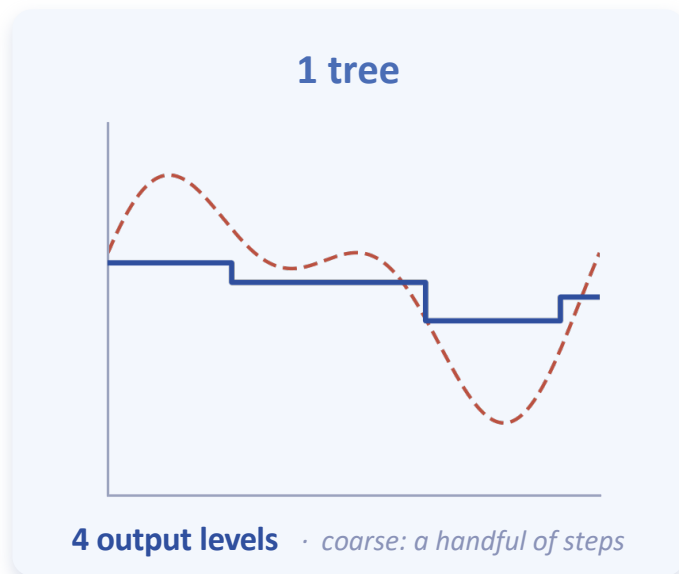
p climbs, the residual shrinks — the next tree has less to do each round.



Each tree adds to the score s ; $p = \sigma(s)$ approaches the label but never reaches 0 or 1.

One tree steps; the ensemble curves

— ensemble (sum of trees)
- - - true function $g(x)$



One tree's precision is capped by its leaves. The ensemble's is set by its **number of trees** — so build precision by stacking, not by deepening.



Shallow trees

Depth 3–6. Each tree is a weak, coarse learner that cannot overfit on its own.



Many rounds

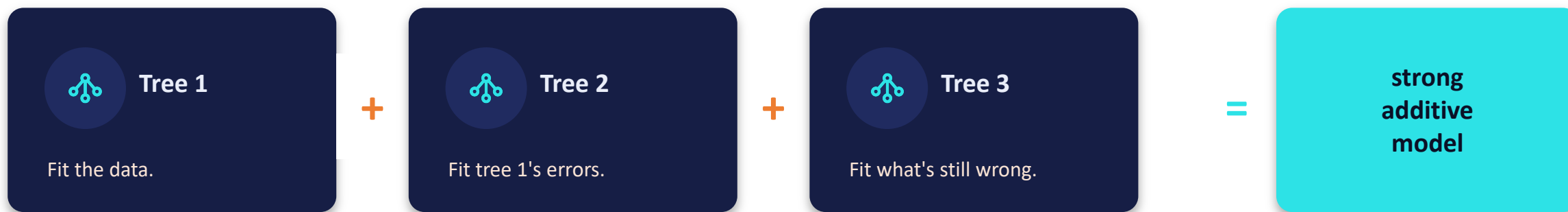
Hundreds to thousands. Resolution accumulates as the coarse steps overlay.



Small learning rate

η shrinks every step, so the sum settles smoothly instead of lurching (shrinkage).

Gradient boosting



The idea

Build the model additively, one small tree at a time. Each new tree is fit to the gradient of the loss — the direction that most reduces current error. A learning rate shrinks each step, so the ensemble improves gradually rather than overfitting in one leap.

In practice

● XGBoost · LightGBM · CatBoost

Fast, regularized, GPU-ready. The default first thing to try on tabular data.

● Key knobs

Tree depth, learning rate, number of trees, subsampling — tuned on the validation set.

Why boosted trees are frequently used in physics analysis



Tabular by nature

Analysis data is a table of engineered kinematic features — exactly what trees eat. No spatial or sequential structure for a net to exploit.



Strong out of the box

Robust to feature scaling, monotone transforms, and irrelevant variables. A good result with little tuning.



A discovery-grade tool

BDTs did real work in the Higgs-boson analyses and searches and remain a standard classifier across the LHC experiments.



The baseline to beat

On tabular physics data a tuned BDT routinely matches or beats a hastily-built neural net. Earn the net.

In practice:

XGBoost, LightGBM: fast, regularized, GPU-ready. The default first thing to try on tabular data.

Key knobs: tree depth, learning rate, number of trees, subsampling – tuned on the validation set

Takeaways



Linear & logistic are the same core

A weighted sum, fit by gradient descent; logistic just adds a sigmoid and a Bernoulli loss.



Gradient descent is the engine

Introduced on the easiest model, it powers everything through the deep-learning half of the series.



Regularization encodes a prior

L2/L1 penalties are Gaussian/Laplace priors; L1's geometry yields sparsity. λ is prior strength.



Ensembles tame trees

Bagging averages to cut variance; boosting fits errors sequentially to cut bias. Boosting is the workhorse.



Respect the BDT baseline

On tabular physics data a tuned boosted tree is the number a neural net has to beat.

NEXT HOUR

Course 3

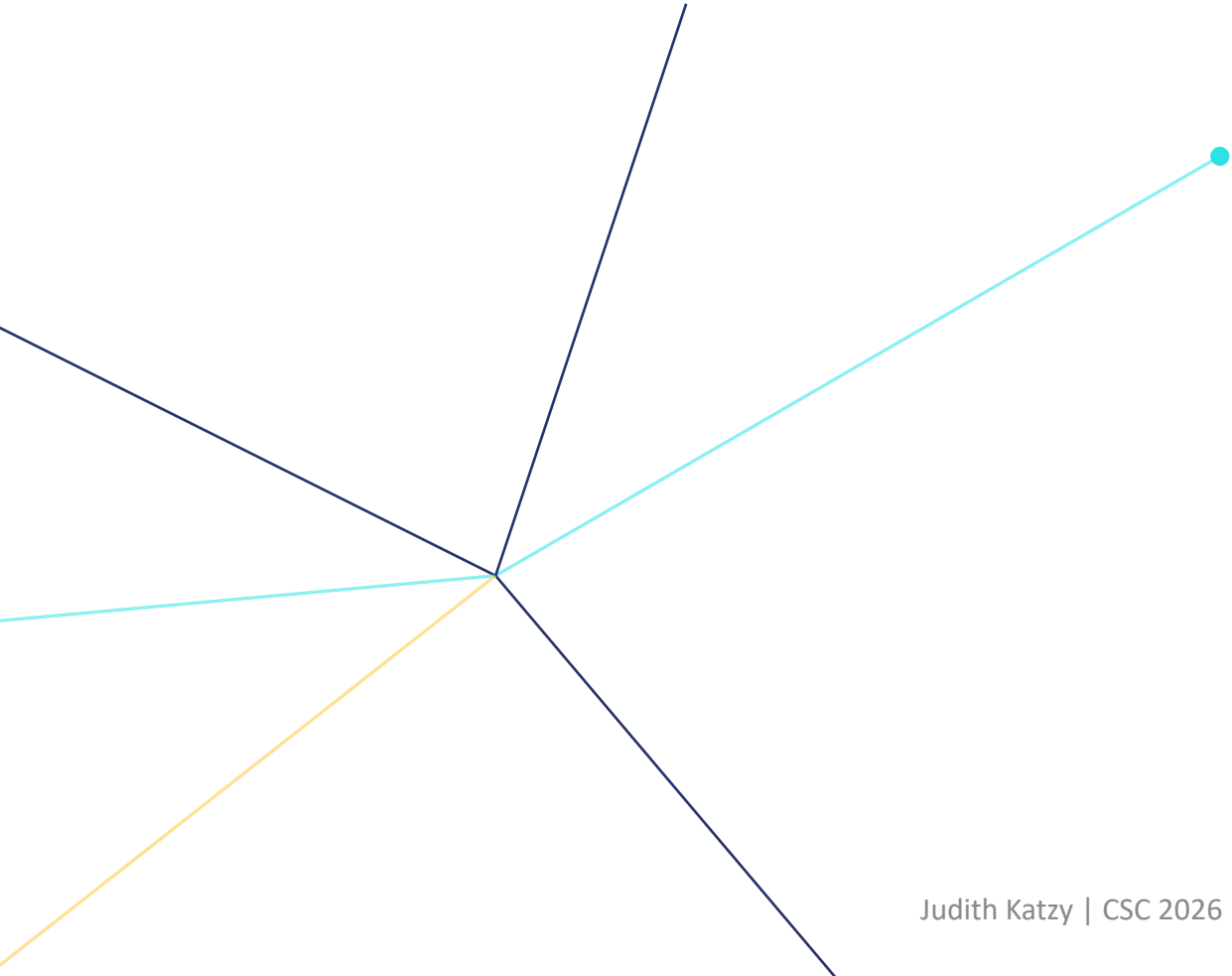
Neural networks

Deep learning foundations & training dynamics

- The MLP and backpropagation as reverse-mode autodiff
- Activations, optimizers, and the regularization toolkit
- Training dynamics: initialization, schedules, vanishing gradients
- Hands-on: build and train an MLP from scratch in PyTorch

The through-line: a neuron is the logistic unit you just met — backprop is only the chain rule, scaled up.

BACK-UP



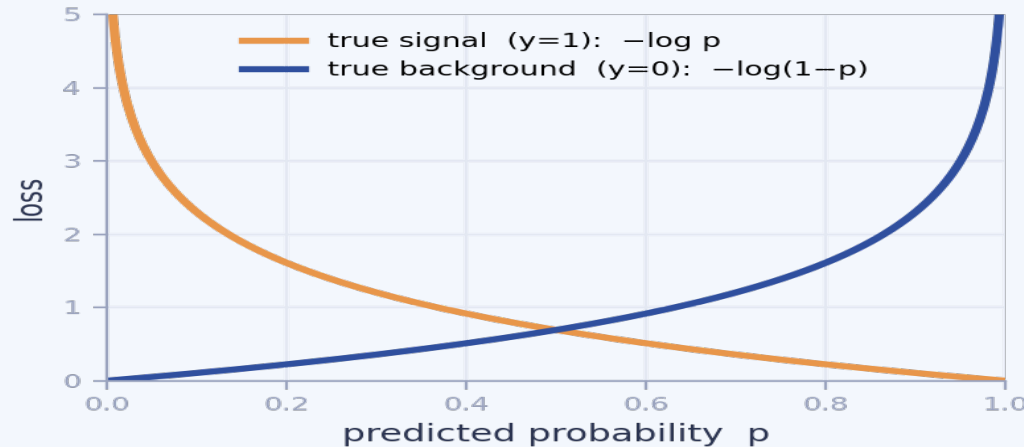
$$\text{score } s_i \rightarrow p_i = \sigma(s_i) \rightarrow \text{loss } L_i$$

THE THING WE DESCEND

How wrong is a predicted probability? Log loss

$$p_i = \underbrace{\sigma(w \cdot x_i + b)}_{s_i} = 1 / (1 + e^{-(w \cdot x_i + b)})$$

$$L_i = - [y_i \cdot \log p_i + (1 - y_i) \cdot \log(1 - p_i)]$$



Confident and wrong is punished without bound — $p \rightarrow 0$ for a true signal sends the loss to infinity.

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial s} \cdot \frac{\partial s}{\partial w}$$

$$\frac{\partial s}{\partial w} = x, \quad \frac{\partial s}{\partial b} = 1$$

$$\partial L / \partial s = p - y$$

$$\frac{\partial L}{\partial w} = (p - y) x, \quad \frac{\partial L}{\partial b} = (p - y)$$

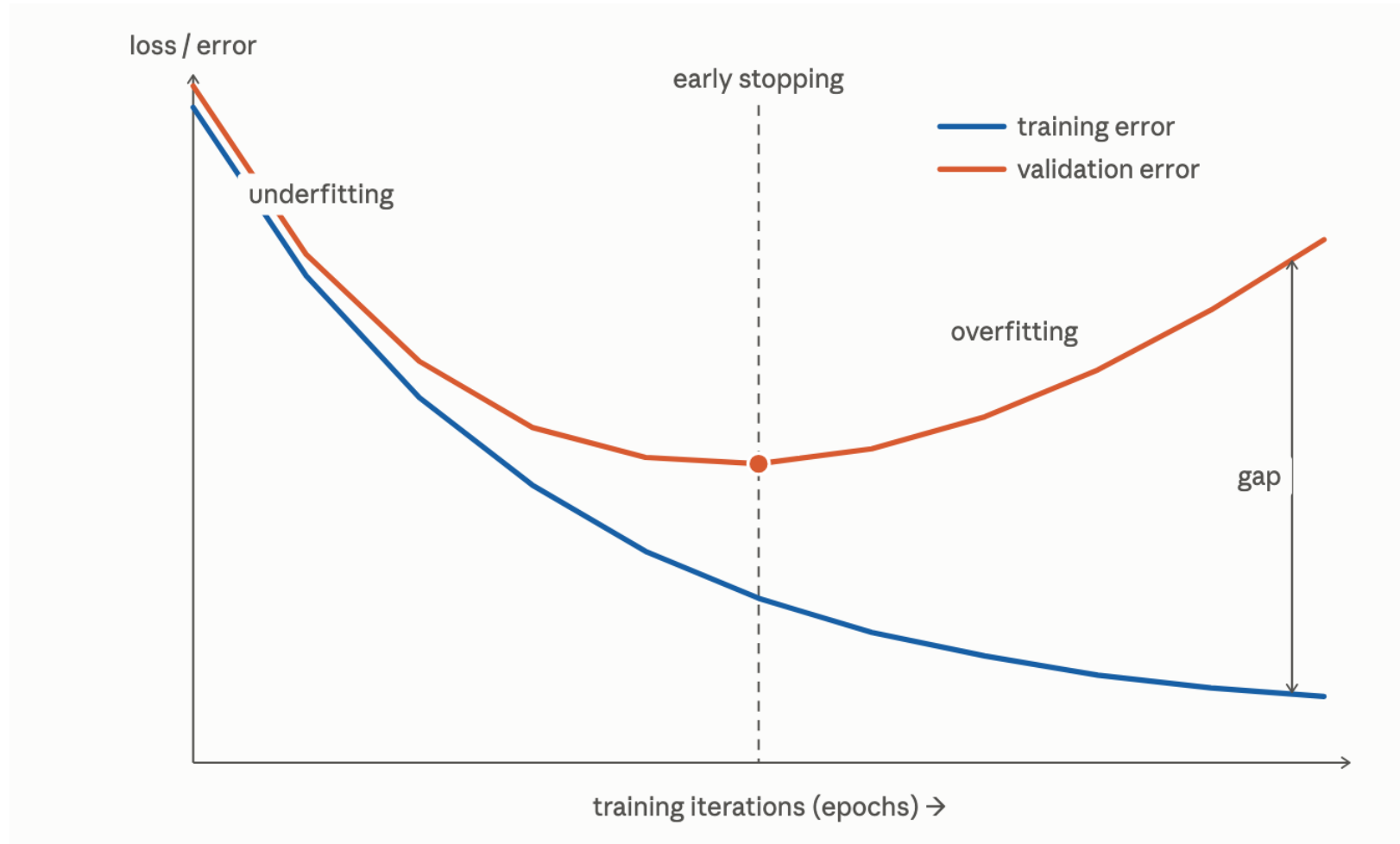
Gradient descent:

$$w \leftarrow w - \eta \partial L / \partial w$$

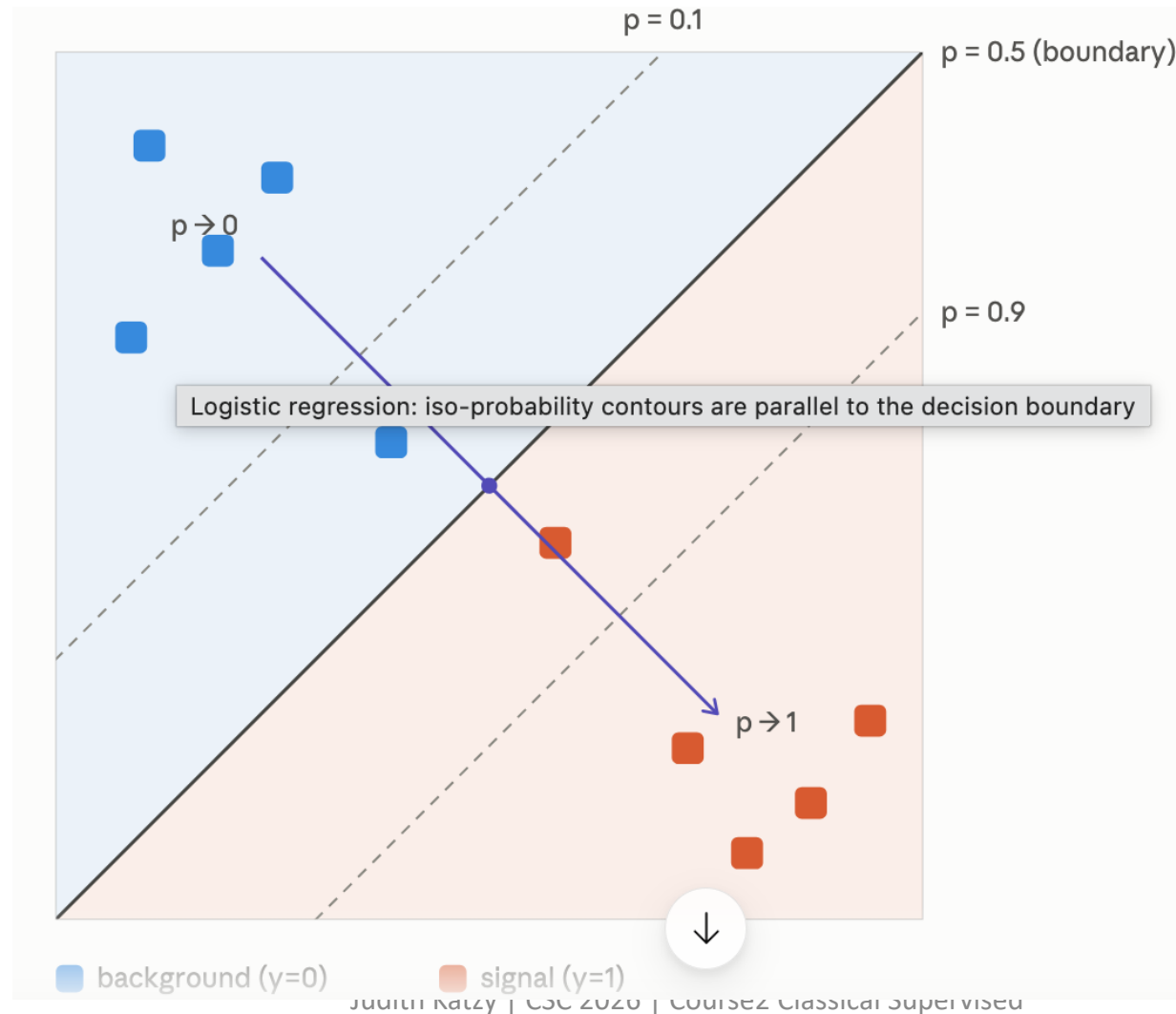
Putting it all in:

$$w \leftarrow w - \eta \underbrace{(p_i - y_i)}_{\text{residuals}} x_i$$

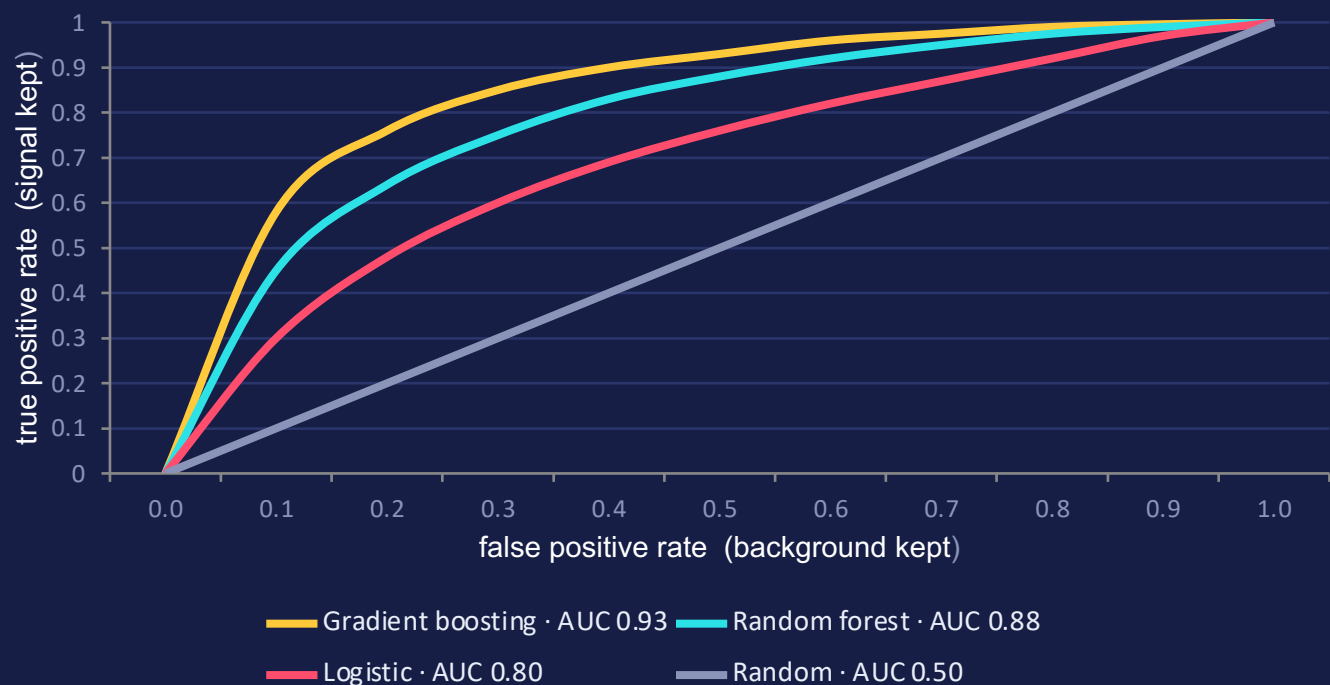
When validation turns around



Probabilities and decision boundaries



ROC curves and AUC



Reading it

The curve

Sweep the threshold; trace signal kept vs background kept. Up and to the left is better.

AUC

Area under the curve: one number, threshold-free. 1.0 is perfect, 0.5 is a coin flip.

Physics dialect

Signal efficiency vs background rejection is the same plot — pick the working point your analysis needs.

Logistic regression

xy

Squash the linear score to (0, 1)

$$p = \sigma(w \cdot x + b) = 1 / (1 + e^{-(w \cdot x + b)})$$

Loss: cross-entropy

$$-\sum [y \log p + (1-y) \log(1-p)]$$

— the Bernoulli negative log-likelihood.

Why it matters here

- **Calibrated output**

The score is a probability, not just a label — essential when you cut on it to define a signal region.

- **Linear boundary**

The decision surface is a hyperplane. Simple, and often not enough — motivating everything that follows.

Regularization: L2 and L1

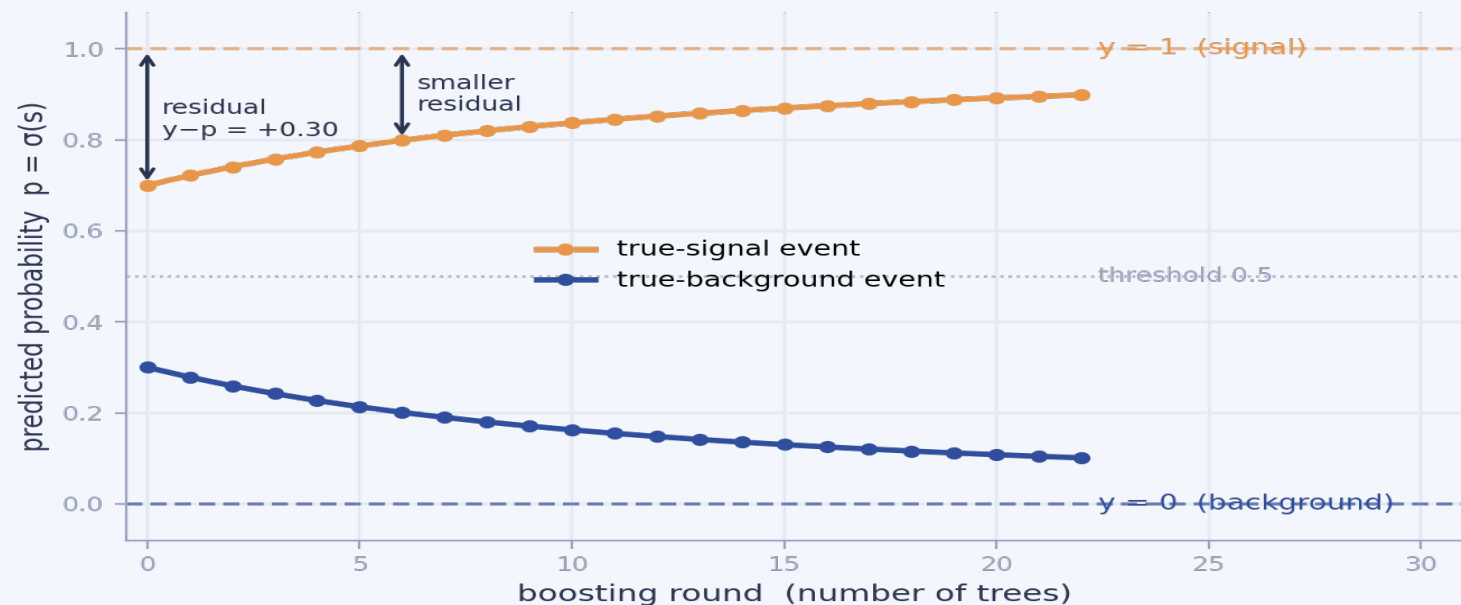
Overfitting. Minimizing the training loss alone lets weights grow to whatever fits the training sample best — including its noise. Large weights make the model very sensitive: a small change in x produces a large change in output (as $w \cdot x_i + b$) so the function wiggles to chase individual training points.

Penalizing $\|w\|$ forbids that, forcing a smoother function that fits the signal, not the noise. This is the bias–variance trade-off: regularization adds a little bias to cut variance to get better test performance.

$$\text{minimize } \mathcal{L}(w) + \lambda \cdot \mathcal{R}(w)$$

$\mathcal{L}(w)$ = the loss you already descend $\lambda \cdot \mathcal{R}(w)$ = the penalty (this slide)

How one event's probability moves



The signal event, round by round

round	p	$y - p$
0	0.70	+0.30
1	0.72	+0.28
2	0.74	+0.26
5	0.79	+0.21
10	0.84	+0.16
22	0.90	+0.10
∞	1.00	0.00

p climbs, the residual shrinks — the next tree has less to do each round.



Each tree adds to the score s ; $p = \sigma(s)$ approaches the label but never reaches 0 or 1.

$$w = (w_1, w_2, \dots, w_n)$$

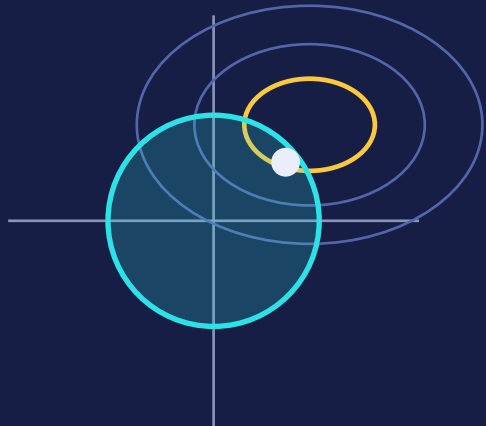
Regularization: L2 and L1

$$\underbrace{L(w)}_{\text{the ellipses}} + \lambda \underbrace{\|w\|}_{\text{the shaded region}}$$

L2 · Ridge

$$\|w\|_2 = \sqrt{w_1^2 + w_2^2 + \dots + w_n^2}$$

$+ \lambda \|w\|^2 \Leftrightarrow$ Gaussian prior on w



Shrinks weights smoothly. Nothing forced to zero.

L1 · Lasso

$$\|w\|_1 = |w_1| + |w_2| + \dots + |w_n|$$

$+ \lambda \|w\|_1 \Leftrightarrow$ Laplace prior on w



Corners sit on axes.
Drives weights to exactly zero $\rightarrow$ feature selection.



The MAP–Bayesian view: a penalty is a prior. Minimizing loss $+ \lambda \cdot$ penalty = maximizing the posterior. λ sets how strongly you believe it.

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial s} \cdot \frac{\partial s}{\partial w}.$$

You already have the first factor from the whole derivation: $\partial L / \partial s = p - y$. For $s = wx + b$,

$$\frac{\partial s}{\partial w} = x, \quad \frac{\partial s}{\partial b} = 1.$$

So

$$\boxed{\frac{\partial L}{\partial w} = (p - y) x, \quad \frac{\partial L}{\partial b} = (p - y)}$$

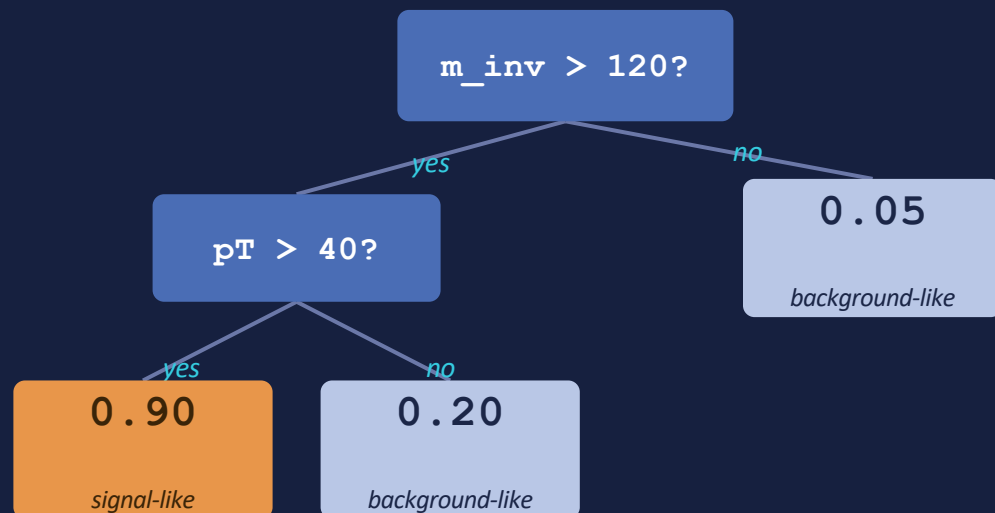
Vector inputs. If $x \in \mathbb{R}^d$ and $s = \mathbf{w}^\top \mathbf{x} + b = \sum_j w_j x_j + b$, then $\partial s / \partial w_j = x_j$, so

$$\frac{\partial L}{\partial w_j} = (p - y) x_j, \quad \nabla_{\mathbf{w}} L = (p - y) \mathbf{x}.$$

Each weight's gradient is the shared scalar residual times its own input feature. The direction of the gradient vector is just $\mathbf{x}$; the residual sets its sign and size.

$$\frac{\partial \mathcal{L}}{\partial w_j} = \sum_i (p_i - y_i) x_{ij}, \quad \nabla_{\mathbf{w}} \mathcal{L} = \sum_i (p_i - y_i) \mathbf{x}_i = X^\top (\mathbf{p} - \mathbf{y}),$$

A leaf is a number, not a verdict



Same tree. The leaves now read as fractions, not words.

The leaf value is a mean



9 signal + 1 background $\rightarrow 9/10 = 0.90$
the average of the 0/1 labels in the leaf

Why one number is enough

$$P(\text{background}) = 1 - P(\text{signal})$$

Two classes, one free parameter — the complement is already there.

Thresholding ($p < 0.5 \rightarrow \text{background}$) is a separate, later choice — and in HEP you set it from the ROC / SVB, not at 0.5.



The leaf is a probability — a soft score. That continuous score, not the hard label, is the thing gradient boosting will correct.

Example regression tree

event	m_inv (feat)	pT (feat)	E = target	lands in leaf
A	130	55	145	m_inv>120, pT>40
B	140	60	138	m_inv>120, pT>40
C	125	50	150	m_inv>120, pT>40
D	135	30	70	m_inv>120, pT≤40
E	90	20	60	m_inv≤120
F	80	45	65	m_inv≤120

Same tree, same cuts. Now the leaves:

- the leaf that said "**signal**" (m_inv>120 and pT>40) catches A, B, C → its value is the mean of their targets, $(145+138+150)/3 = \mathbf{144.3}$
- the "**background**" leaf at (m_inv>120, pT≤40) catches D → value **70**
- the "**background**" leaf at (m_inv≤120) catches E, F → value $(60+65)/2 = \mathbf{62.5}$

Training target is to minimize variance of the target values (here E) in the leaves

Precision is limited by the number of leaves