

Decision Trees

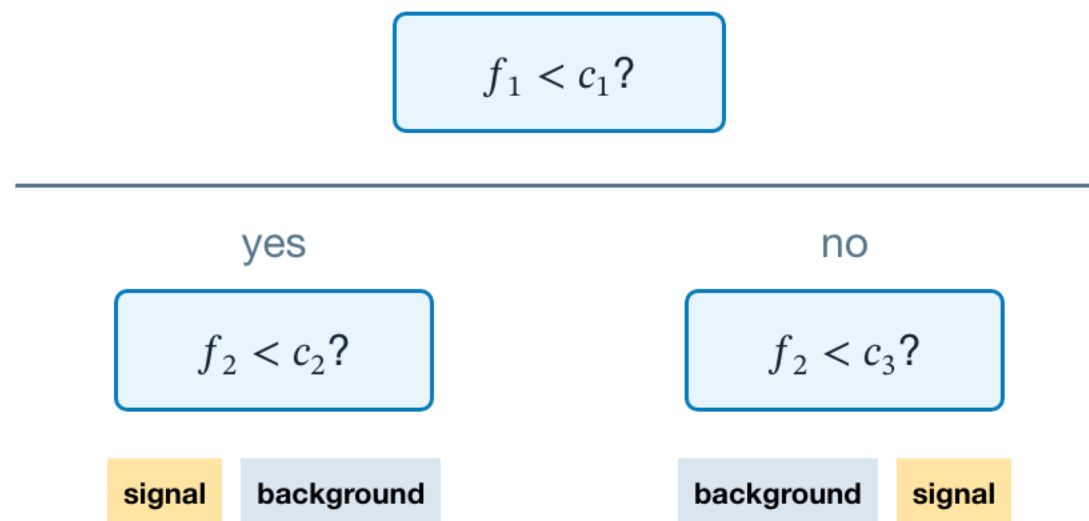
(Not the dating app)

Mikhail Mikhasenko · Lecture 2A

Decision trees

A sequence of simple questions partitions feature space.

- At each node, choose a **feature** and a **cut**.
- Send each event to the left or right child.
- A leaf returns a class, a class fraction, or a regression value.



Reminder: cutting distributions

For a selection $x < c$:

$$\epsilon_S = \frac{S_{\text{pass}}}{S_{\text{all}}}, \quad r_B = 1 - \frac{B_{\text{pass}}}{B_{\text{all}}}$$

$$p = \frac{S_{\text{pass}}}{S_{\text{pass}} + B_{\text{pass}}}$$

Efficiency and **purity** answer different questions.

The ROC curve traces the operating points as the threshold moves.

Here the vertical axis is **background rejection**, $1 - \epsilon_B$.

Purity also depends on the signal fraction **before** selection.

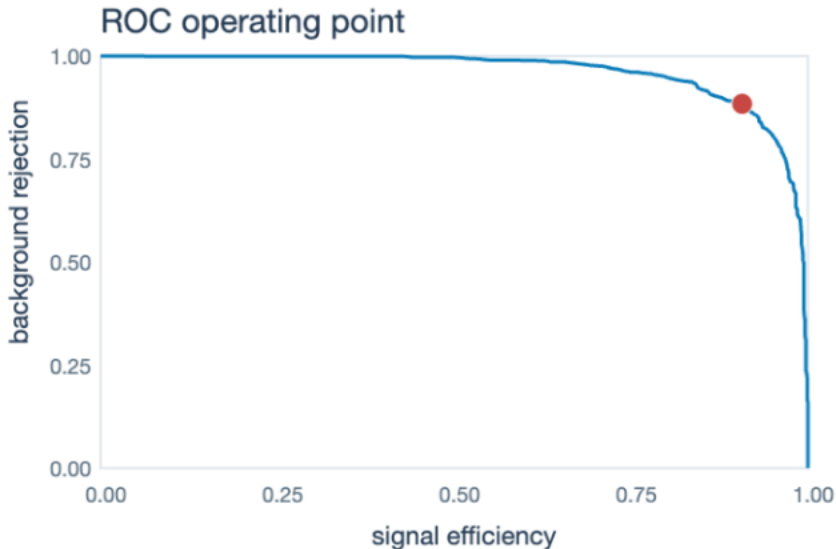
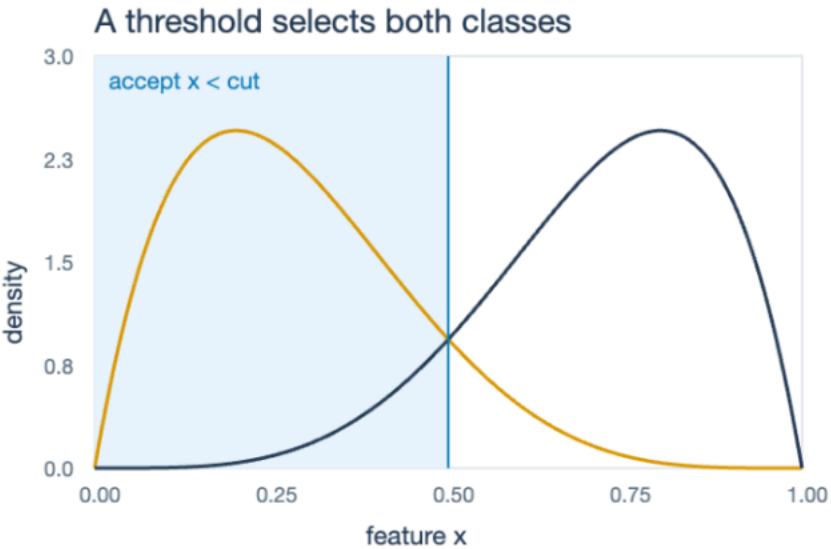
One cut, three competing goals

One cut, three competing goals

● signal ● background

Signal: Beta(2,5). Background: Beta(5,2). The shaded region passes the cut.

Cut 0.5 Signal fraction 0.5



Signal efficiency	Background rejection	Selected purity	Signal fraction before cut
90.7%	88.4%	88.7%	50.0%

Keep the cut fixed and change the signal fraction. Which numbers change, and which stay fixed?

Splitting index

A useful cut depends on the objective.

Objective	Example	What it measures
Select a sample	Purity $S / (S + B)$	Composition after selection
Retain signal	Efficiency S / S_{all}	Signal survival
Estimate sensitivity	$S / \sqrt{S + B}$	A simple counting proxy
Grow a tree	Gini or entropy gain	Improvement in both children

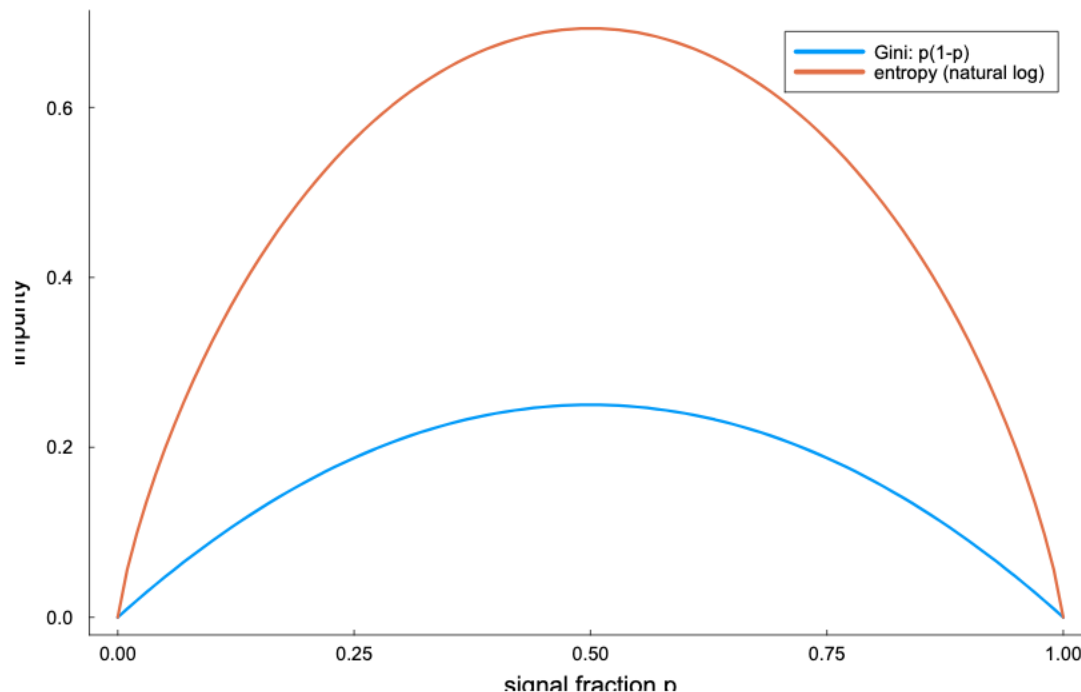
Gini impurity

If p is the weighted signal fraction,

$$p = \frac{\sum_{i \in S} w_i}{\sum_i w_i}, \quad I_G = p(1 - p).$$

A pure node has $I_G = 0$. A balanced node has $I_G = 1/4$.

The standard binary Gini impurity is $2p(1 - p)$. The factor of two does not change the best split.



Making a single split

$$\Delta I = I(p_0) - \frac{W_L}{W_0} I(p_L) - \frac{W_R}{W_0} I(p_R)$$

- $W_0 = W_L + W_R$ is the total weight at the parent.
- The two children contribute in proportion to their weights.
- Search candidate features and thresholds. Choose the largest gain.

For unweighted data, $W = N$. Multiplying by N_0 gives the original lecture's $N_0 I_0 - N_L I_L - N_R I_R$.

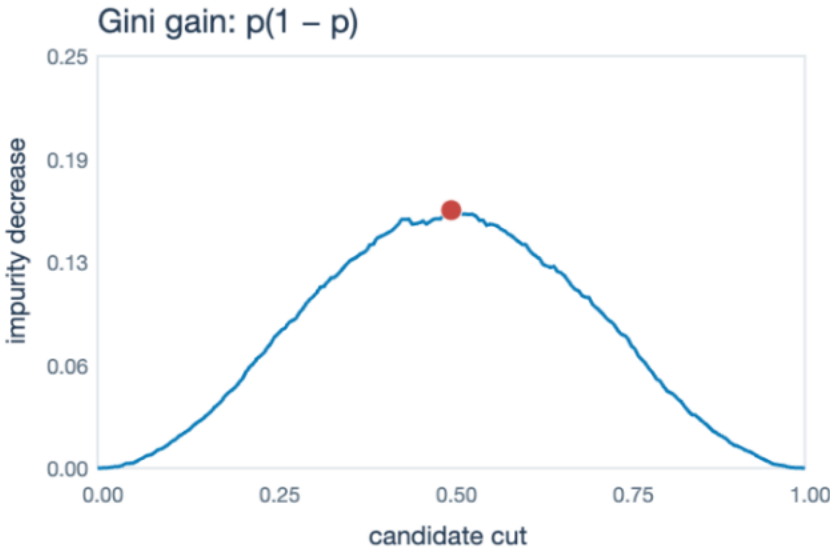
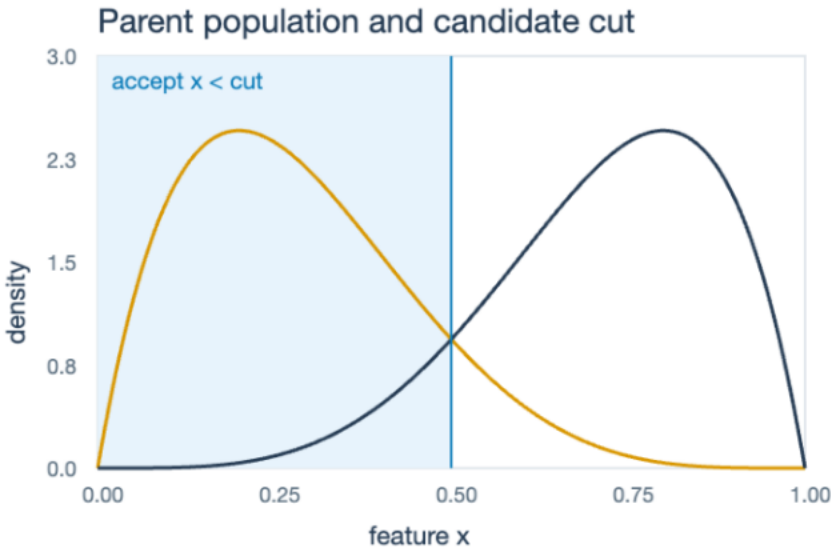
Where should the tree split?

Where should the tree split?

● signal ● background

Compare the weighted impurity of the children with the impurity of the parent.

Cut 0.5 Criterion Gini Find best cut Reset



Gain at selected cut	Best cut on 0.005 grid	Best gain	Parent impurity
0.1565	0.500	0.1565	0.2500

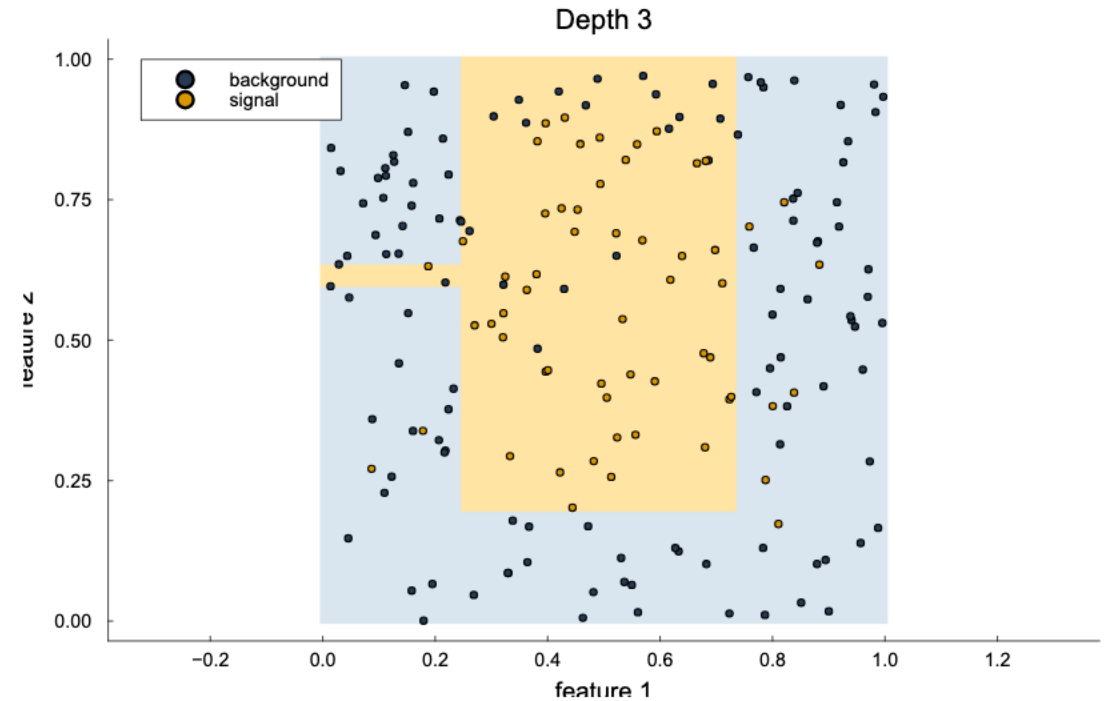
A pure but tiny child may give little gain. Both children contribute in proportion to their size.

Growing a tree

Repeat the split search **inside each child**.

Stop when:

- the node is pure,
- the depth limit is reached,
- too few events remain, or
- no allowed split improves the criterion.



How well does it do?

$$\text{accuracy} = \frac{N_{\text{correct}}}{N_{\text{events}}}$$

Training: fit the splits and leaf predictions.

Validation: choose depth and other settings.

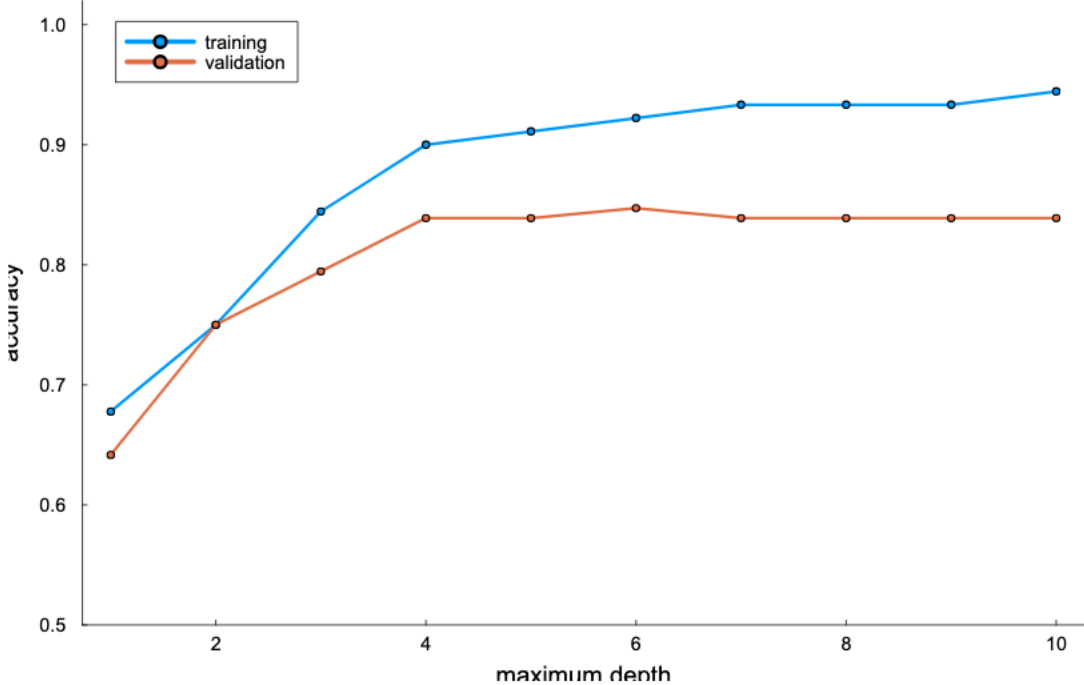
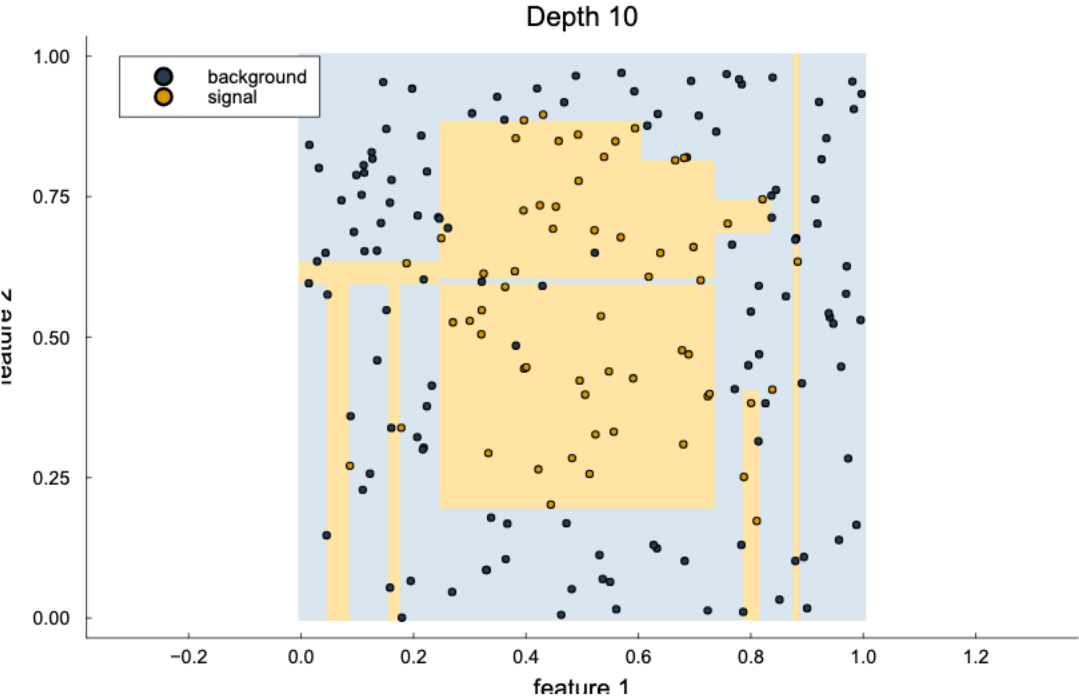
Test: assess the chosen model once.

A model can improve on training data while getting worse on new data.

Keep the data split fixed while comparing settings.

For imbalanced classes, inspect class efficiencies and ROC curves as well as accuracy.

Overtraining



Depth and bagging

Depth, overtraining, and bagging

● signal ● background

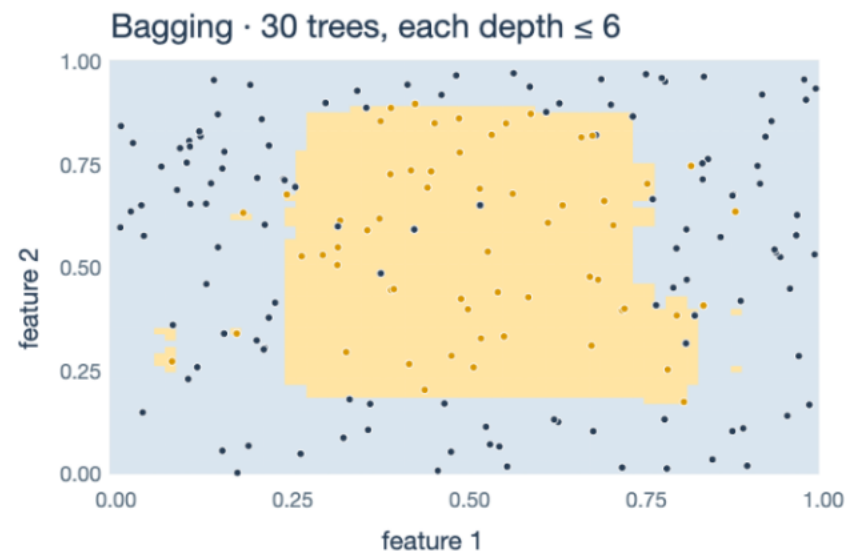
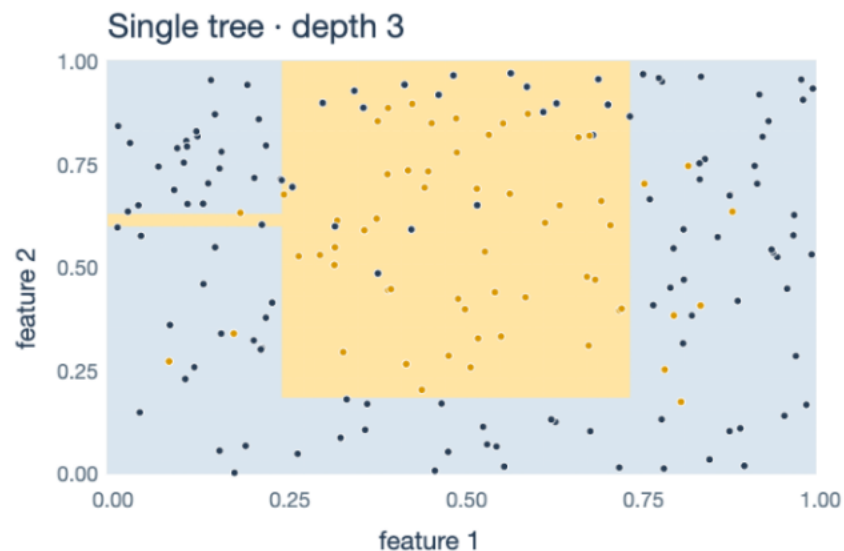
Same training data, separate validation data. Gold = signal; blue = background.

Tree depth 3

Bagged trees 30

Points Training ▾

Reset



Tree training accuracy
84.4%

Tree validation accuracy
79.4%

Bagging training accuracy
95.0%

Bagging validation accuracy
86.4%

Increase depth, then switch to validation points. Bagging uses bootstrap samples and both features at each split.

Bagging

Bootstrap aggregating

1. Draw a training sample with replacement.
2. Train a tree on that sample.
3. Repeat, then average predictions or vote.

Different samples produce different trees.

Averaging can reduce variance.

The original lecture's example settings:

```
1 n_trees = 30
2 max_depth = 6
3 partial_sampling = 0.7
4 n_subfeatures = 2
```

A **random forest** also samples the candidate features at each split. Using both of two features adds no feature randomness.

Julia: regression trees

The same partitioning idea works for continuous targets.

Each leaf predicts its training mean:

$$\hat{y}_R = \frac{1}{N_R} \sum_{i \in R} y_i.$$

Squared-error splits prefer regions with less target variation.

```
1 using DecisionTree
2 X = reshape(x_train, :, 1)
3 model = DecisionTreeRegressor(
4     max_depth=5, rng=123)
5 fit!(model, X, y_train)
6 yhat = predict(model, X_validation)
```

[Open the regression notebook](#) · [Julia source](#)

Julia notebooks

Notebook	Experiment	Lecture connection
Cuts and impurity	Move the cut and vary the mixture	ROC, purity, split gain
Classification trees	Change depth, tree count, boosting rounds	Boundaries, overtraining, ensembles
Regression trees	Fit 1D and 2D functions	Piecewise predictions, forests, losses

The slide figures and widget models come from the same seeded Julia calculations.

Boosting

AdaBoost trains weak classifiers in sequence.

1. Start with equal event weights.
2. Fit a classifier using the current weights.
3. Increase the relative weight of mistakes.
4. Add the classifier to a weighted vote.

Bagging: different resampled datasets, independently trained trees.

Boosting: each fit depends on earlier mistakes.

A weak learner needs an error below $1/2$ in the binary setting.

AdaBoost: weights and votes

For labels and predictions $y_i, h_m(x_i) \in \{-1, +1\}$:

$$\epsilon_m = \frac{\sum_i w_i^{(m)} \mathbf{1}[h_m(x_i) \neq y_i]}{\sum_i w_i^{(m)}}, \quad \alpha_m = \frac{1}{2} \log \frac{1 - \epsilon_m}{\epsilon_m}$$

$$w_i^{(m+1)} \propto w_i^{(m)} e^{-\alpha_m y_i h_m(x_i)}, \quad H(x) = \text{sign} \left(\sum_{m=1}^M \alpha_m h_m(x) \right)$$

Normalize the event weights after each round.

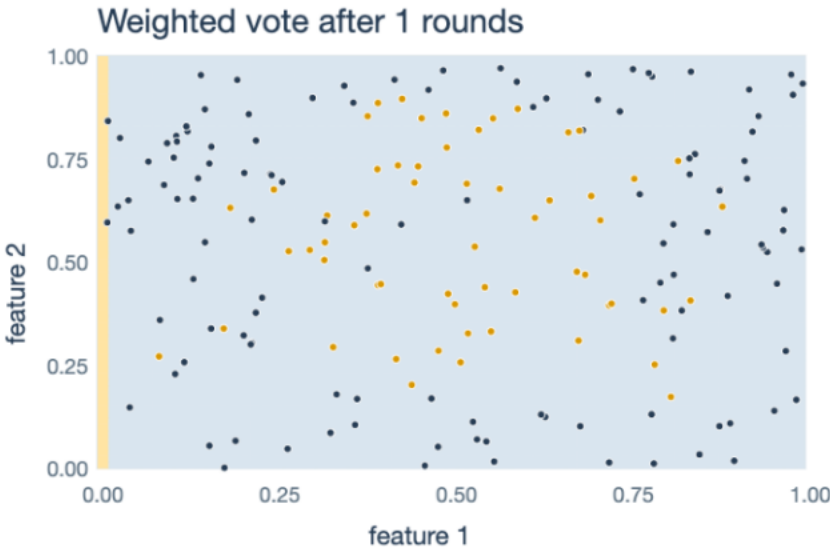
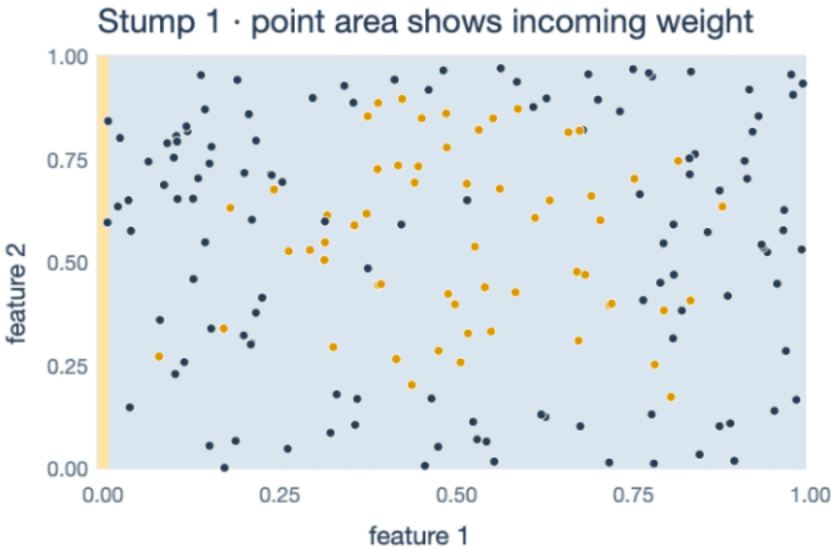
AdaBoost pays attention to mistakes

AdaBoost pays attention to mistakes

● signal ● background

Point areas on the left represent weights BEFORE fitting the selected stump.

Round 1 Next round → Reset



Weighted stump error	Stump vote α	Ensemble training accuracy	Ensemble validation accuracy
32.2%	0.372	67.8%	62.8%

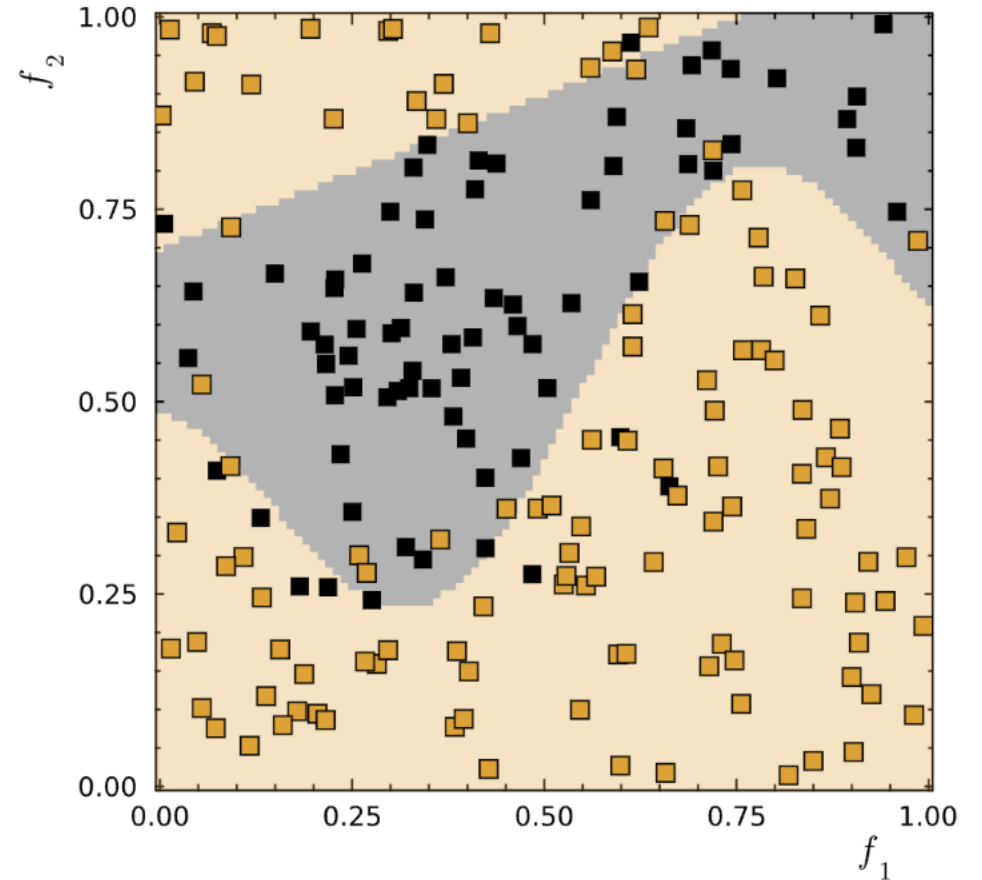
Move to the next round and watch the previously misclassified points grow. The right panel combines all stumps so far.

Original sample

Two measured features, overlapping classes.

A tree approximates the separating boundary with axis-aligned regions.

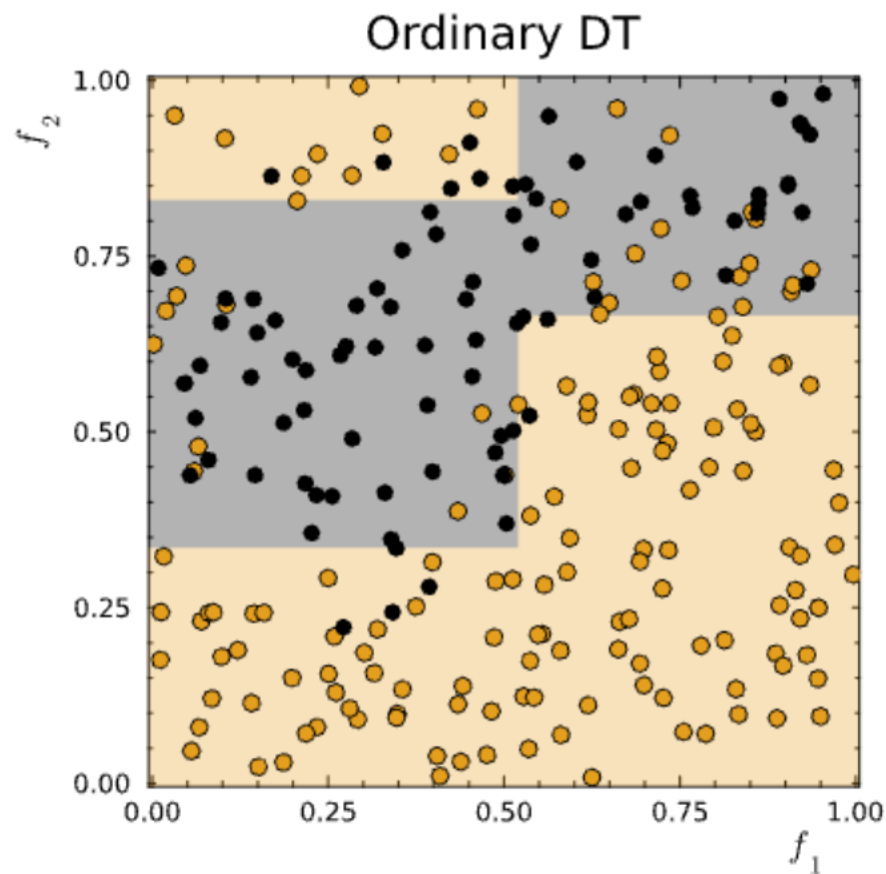
Noise and class overlap limit what any fitted boundary can achieve.



Quiz

Looking at this classification, **draw a possible tree**.

- Which feature could the first split use?
- Which later cuts apply only inside one region?
- Can different trees produce the same partition?



End

Thank you for the attention