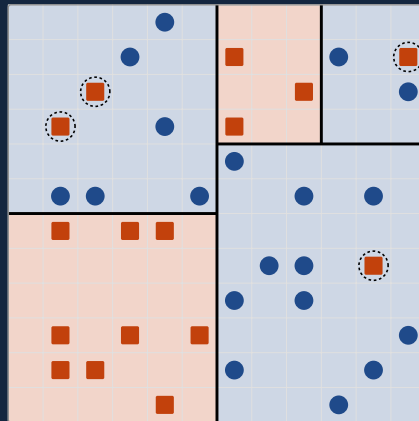




giuseppemuschetta.com

OVERFIT

cut • test • does it generalize?



`train(visible)  $\neq$  test(unseen)`

JavaScript • Canvas • SVG

OverFit: an Interactive Decision-Tree Simulator on Generalization, Overfitting and Underfitting

Project Report

Giuseppe Muschetta

giuseppemuschetta.com/overfit

2026

Abstract

OverFit is an interactive simulator of classification by a decision tree, written for the web in plain JavaScript with no framework, no library, no build step and no server. The player trains a tree by cutting a 12×12 board with straight lines, each cut a split of the kind a tree library makes, $x \leq 4.5$, and every region predicts the class of most of its points. The points on the board come from a hidden rule of straight cuts, and some of them were flipped by noise. When the player presses Test, a new sample the tree has never seen, one point in every empty cell, is scored, and the simulator answers the question every model faces: does the tree generalize, or does it overfit or underfit? Nine experiments look at that question from nine sides: noise, a small real group, a staircase border, little data, heavy noise, three classes, a board where greedy growth starts wrong, pruning a full tree, and noise in two colors. The rules are the real ones: Gini impurity, best-first growth as scikit-learn's `max_leaf_nodes`, cost-complexity pruning, and a test set that is shown once, every further look costing a star. Each board is chosen by an exact search so that accuracy on new data peaks at the rule's own cuts and the full tree clearly loses, and 37 automated tests check that every board teaches what its text says. Seven interactive cards under the simulator explain the model with small boards of their own. The page is 28 files and 230 KB.

Contents

1	Introduction	3
2	The simulation	3
2.1	Board and points	3
2.2	A cut is a split	3
2.3	A region votes	4
2.4	The two scores	5
3	Where the data comes from	5
3.1	The rule	5
3.2	Noise	5
3.3	New data	5
3.4	Seeds and reproducibility	5
4	Generalization, verdicts and stars	5
4.1	One definition	5
4.2	When a level is solved	6
4.3	The verdicts	6
4.4	Stars and the test set	7
5	The overfitting curve and the full tree	7
5.1	Best-first growth	7
5.2	Thresholds in the middle of a gap	8
5.3	The curve	8
5.4	The full tree	8
6	The nine experiments	9
7	Choosing the boards	11
7.1	The checks every board passes	11
7.2	An exact search over trees	12
7.3	How the boards were found	12
8	How a decision tree works: the cards	12
9	Architecture	12
9.1	Modules	12
9.2	Drawing	13

9.3	Input	13
9.4	For every reader	13
9.5	Storage	14
10	Testing	14
11	Results	14
12	Conclusion	15

1 Introduction

Every introduction to machine learning says that a model must generalize: it is judged on data it has never seen, not on the data it was trained on. The sentence is easy to read and hard to feel. OverFit was built so that the reader feels it: the training data is on the board, the test data is hidden until a button is pressed, and the two scores that appear are the reader's own doing. The simulator lives on the author's website next to PolyFit, a game for everyone; OverFit is for readers who already know some machine learning and want to see, on a board they can touch, what overfitting and underfitting are. Five goals guided every decision.

- G1. One question.** Every level asks the same thing, *does it generalize?*, and every text, verdict and star answers it. One definition holds throughout: a tree generalizes well when it does well on new data too, about as well as on its training data.
- G2. The real rules.** A cut is a split on one feature; a region predicts by majority vote; impurity is Gini; the computer grows trees best cut first, as a library does when its leaves are capped; pruning is cost-complexity pruning. Nothing is simplified in a way an expert would object to.
- G3. An honest test set.** New data is fixed by a seed and scored only when the player asks. Looking at it costs: three stars for a level solved on the first test, two on the second, one after that, because a tree chosen on the test data is not tested anymore.
- G4. Boards that teach.** A board is used only if accuracy on new data peaks at the rule's own cuts, the full tree clearly loses, and every tree as small as the rule that fits the training points like the rule does at least as well on new data. The checks run in the test suite.
- G5. Made from scratch, for every reader.** Plain JavaScript modules, the board on a canvas, the tree and the curve in SVG, no library of any kind; mouse, touch and keyboard; shapes as well as colors; light and dark themes; less motion for those who ask for it.

Section 2 describes the simulation and Section 3 where the points come from. Section 4 defines generalization as the simulator uses it, the verdicts and the stars. Section 5 covers the overfitting curve and the full tree, Section 6 the nine experiments, Section 7 how the boards were chosen, and Section 8 the seven cards. Section 9 presents the architecture, Section 10 the tests, Section 11 the measured results, and Section 12 concludes.

2 The simulation

2.1 Board and points

The board is a grid of 12×12 cells, with at most one point per cell. Columns are $x = 0, \dots, 11$ from left to right and rows $y = 0, \dots, 11$ from bottom to top, as on a chart. A point has a class drawn as a color and a shape: blue circles and orange squares, and green triangles on the two boards with three classes. The shape is there for readers who do not tell the colors apart. The points on the board are the training data. The player is told how many of them were flipped by noise, that is, drawn in a class that is not the one of the rule they come from, but never which ones.

2.2 A cut is a split

A cut is a straight line of the grid, vertical or horizontal, inside one region, with at least one point on each side. A vertical cut after column t is the question $x \leq t + 0.5$; a horizontal one after row t is $y \leq t + 0.5$. That is exactly a split of a CART tree on one feature, with the threshold halfway between two values, and the diagram under the board writes it that way. Points that

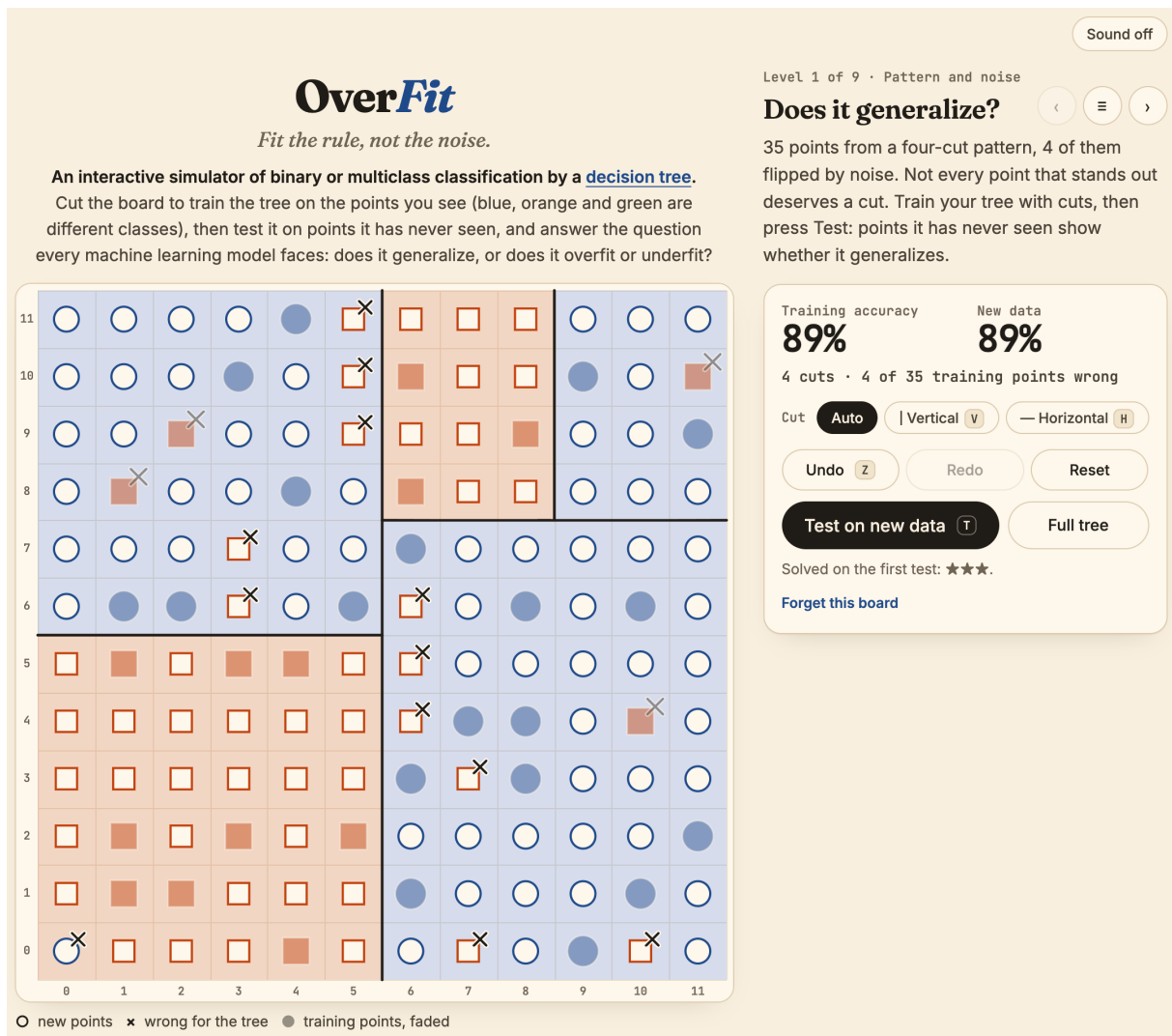


Figure 1: Level 1 after a test. The board holds the 35 training points (filled) and the 109 new points (hollow); the four cuts of the player’s tree tint every region with its class; a × marks a point the tree gets wrong. Training accuracy and accuracy on new data are both 89%: the tree generalizes, and the level is solved on the first test.

answer yes go to the left child, the others to the right. The board starts as one region, the root; each cut turns one region into two, and the regions are the leaves of the tree. Cuts can be undone, redone, and removed: removing a cut removes every cut below it, as pruning a node removes its subtree.

2.3 A region votes

Every region predicts the class of most of its training points. On a tie, or with no points at all, it keeps the class of the region it was cut from. This is a rule of the simulator, stated on the page; libraries break ties their own way (scikit-learn takes the lowest class index). A training point of another class inside a region is a mistake and carries a small ×. The tree diagram grows with every cut and shows, under each leaf, how many points it gets right and wrong; pointing at a leaf highlights its region, and pointing at a point shows its path, the questions from the root to its leaf.

2.4 The two scores

Two numbers stand next to the board. *Training accuracy* is the share of the visible points the tree gets right. *New data* is the same share on points the tree has never seen, and it reads “?” until the player presses Test. Once tested, a tree shows its score; if the tree changes, the score goes back to “?” and the verdict reads “Your tree changed: does it still generalize?”. The hover preview of a cut shows how training accuracy would change, never how the new data would: the tree is built on the training points only.

3 Where the data comes from

3.1 The rule

Every board comes from a rule: a decision tree written as data, `[axis, t, low, high]`, with class numbers as leaves. The rule of level 1, for example, is `['x', 5, ['y', 5, 1, 0], ['y', 7, 0, ['x', 8, 1, 0]]]`: four cuts and five regions. The rule is the pattern a tree should learn, and the simulator shows it, dashed, only on request, at the cost of a star. A tree is written the same way, so a rule can be put on the board, compared with the player’s tree, and saved and restored in the undo history.

3.2 Noise

Some training points are flipped: their class is not the one the rule gives their cell. A flipped point on a two-class board takes the other class; with three classes it takes one of the other two. Flipped points are placed at least two cells away from every edge of their rule region and never in a corner cell, so that no single cut, and no pair of cuts in a corner, can isolate one of them by accident; the test suite checks both conditions on every board. The share of flipped points goes from one in 31 (level 2) to seven in 36 (level 5).

3.3 New data

When the player tests, the tree is scored on new data: one point in every empty cell of the board, classed by the rule, with the same share of them flipped as on the board. Which ones are flipped is drawn with a seeded generator, so the new data of a board is always the same: the same tree always gets the same score, on every device, and every number in this report can be reproduced. Table 1 gives the sample of each level. Since the rule gets every clean point right and every flipped point wrong, its accuracy on new data is exactly the share of clean new points, and no classifier can do better on average: it is the Bayes rate of the board. A tree can beat it on the fixed sample only by luck, by being right on a flipped point, and the verdict says so.

3.4 Seeds and reproducibility

Random numbers come from Mulberry32, a 32-bit generator of a few lines, seeded with the FNV-1a hash of a text such as `pattern-noise-test:3116`. The seed of each level is part of its definition. The boards of the cards are made the same way from a date text, so they are the same for every visitor and never equal to a level: the cards do not reveal a solution.

4 Generalization, verdicts and stars

4.1 One definition

The page uses one definition everywhere: a tree generalizes well when it does well on new data too, about as well as on its training data. Overfitting is high on training and lower on new data:

#	Level	New points	Flipped	Share	Share on board	Rule on new data	Bayes error
1	Pattern and noise	109	12	11%	11%	89%	11%
2	Three together	113	4	4%	3%	96%	4%
3	Too simple	106	8	8%	8%	92%	8%
4	Few points	131	10	8%	8%	92%	8%
5	Heavy noise	108	21	19%	19%	81%	19%
6	Three classes	110	10	9%	9%	91%	9%
7	Beat the library	106	8	8%	8%	92%	8%
8	Prune it	107	12	11%	11%	89%	11%
9	Two kinds of noise	106	11	10%	11%	90%	10%

Table 1: The new data of every level: one point per empty cell, with the board’s own share of noise (rounded to a whole number of points). The rule’s accuracy on new data is the share of clean points, the best any classifier can expect.

Verdict	When
Your tree changed: does it still generalize?	the tree differs from the one tested
It generalizes, with some luck.	solved, and above the rule on new data: right on flipped points by luck
Better than the rule, by luck.	above the rule on new data, but not the rule’s mistakes on training: not solved
It generalizes.	solved: the rule’s mistakes, and as good as the rule on new data
Right score, wrong tree.	the rule’s score on new data, but not its mistakes on the training points
Overfitting: it generalizes worse.	above the rule on training, below it on new data
Pruned too far: too simple to generalize well.	level 8 only: a cut the pattern needs was removed
Underfitting: too simple to generalize well.	fewer cuts than the rule
Cuts in the wrong place: it generalizes worse.	enough cuts, below the rule on training
Close, but it generalizes worse.	the rule’s mistakes, or as many, but a border runs elsewhere through empty cells

Table 2: The verdicts, in the order the simulator tries them.

the tree memorized the points, the noise included. Underfitting is low on both: the tree learned too little. The level title is always “Does it generalize?”; the level name is the line above it.

4.2 When a level is solved

A level is solved when two things hold at once: the tree is wrong on exactly the training points flipped by noise, as the rule is, and nowhere else; and on new data it scores at least as much as the rule. The second condition rules out a tree that matches the training mistakes but draws a border through empty cells in a different place than the rule. The simulator can judge this because it knows the rule the points came from, and the page says so: a real project only has the score. That is also why the verdict “Right score, wrong tree” exists.

4.3 The verdicts

After a test the simulator writes one of the verdicts of Table 2, in this order of precedence, with a lesson of a few lines that quotes the numbers of the board.

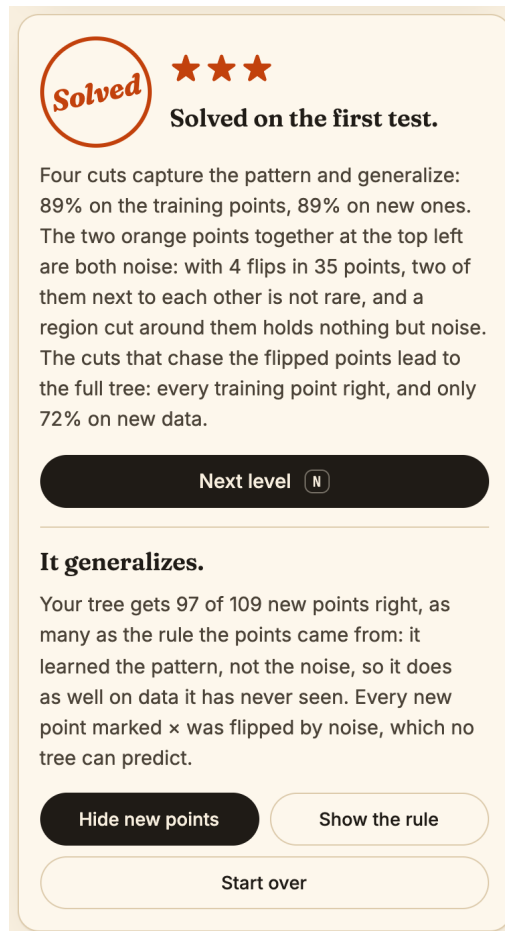


Figure 2: The result card of level 1: the stamp, the stars, the lesson of the level with its numbers, and the verdict with the count of new points right, 97 of 109, as many as the rule.

4.4 Stars and the test set

A level solved on the first test earns three stars, on the second two, later one. A tree already tested is free to test again, also after a reload: it shows nothing new. Showing the rule, or moving the slider that lets the computer grow the tree, caps the level at one star. Starting over keeps the count of tests: the player has seen the new data, and a fair test uses it once. The reason is written under the curve: once the new data has been seen, every change made to score better on it uses it to choose the tree, and its score stops being a fair measure of generalization. Real projects tune on a validation set or by cross-validation and open the test set once, at the end. Stars, tests and the trees tested are kept in the browser’s storage under a key made of the level’s name and a fingerprint of its board, so that a board that changes starts again from no stars, and “Forget this board” clears one level.

5 The overfitting curve and the full tree

5.1 Best-first growth

After the first test, a chart under the tree shows the overfitting curve of the board: training accuracy and accuracy on new data for the trees a library grows as its cap on leaves is raised, from no cut to a tree whose every leaf is pure. The growth is best-first, as scikit-learn grows a tree with `max_leaf_nodes`: at each step, among all the leaves, the cut with the largest drop in Gini impurity is taken, the drop being weighted by the number of points the leaf holds,

$n(G_{\text{before}} - G_{\text{after}})$, with G_{after} the average of the two sides weighted by their sizes. Gini impurity is $1 - \sum_c p_c^2$, the chance that two points drawn at random from the region, with replacement, have different classes.

```
export function growthPath(points, classes) {
  const tree = new Tree(points, classes);
  const path = [tree.snapshot()];
  for (;;) {
    let best = null;
    for (const leaf of tree.leaves()) {
      if (Math.max(...leaf.counts) === leaf.total) continue; // pure
      for (const { axis, t } of gapCuts(tree.counter, leaf.box)) {
        const p = tree.preview(leaf, axis, t);
        const drop = leaf.total * (p.giniBefore - p.giniAfter);
        if (!best || drop > best.drop + 1e-12) best = { leaf, axis, t, drop };
      }
    }
    if (!best) return path;
    tree.split(best.leaf, best.axis, best.t);
    path.push(tree.snapshot());
  }
}
```

5.2 Thresholds in the middle of a gap

Two lines that split the points of a region the same way, because they cross the same empty gap, give the same tree; so each gap is tried once, at its middle line, halfway between the neighboring values, where a library places its thresholds. On level 1 the first cut of the growth is $x \leq 5.5$, the same as the rule's. Counting the points of a class inside any rectangle takes constant time, from one table of two-dimensional prefix sums per class, so the whole growth of a board takes a fraction of a millisecond (Section 11).

5.3 The curve

Training accuracy never falls along the curve. On every board of the simulator but one, accuracy on new data rises to its best at the rule's own number of cuts, where it equals the rule's score, and never rises again after that: the boards are chosen so that this holds (Section 7), and the page says so. The exception is level 7, where the library's first cut is wrong on purpose. The page also says what the chart does not show: on real data the curve is noisier, it can wobble and rise again after a dip, and nobody says how many cuts the rule has. The player's own tests are drawn on the chart as dots, and a slider lets the computer grow the tree step by step, at the cost of a star.

5.4 The full tree

The button "Full tree" grows the tree to the end of the curve: every leaf pure, 100% on the training points. A callout explains why that is the trap of the simulator. The full tree looks perfect on the data it has seen because it memorized the noise along with the pattern, and the small regions cut around flipped points are wrong on most new points that fall in them: on level 1 it has 11 cuts and scores 72% on new data, against 89% for the four cuts of the rule. The gap between the two scores is the sign of overfitting, and the tree that fits the training data best generalizes worst.

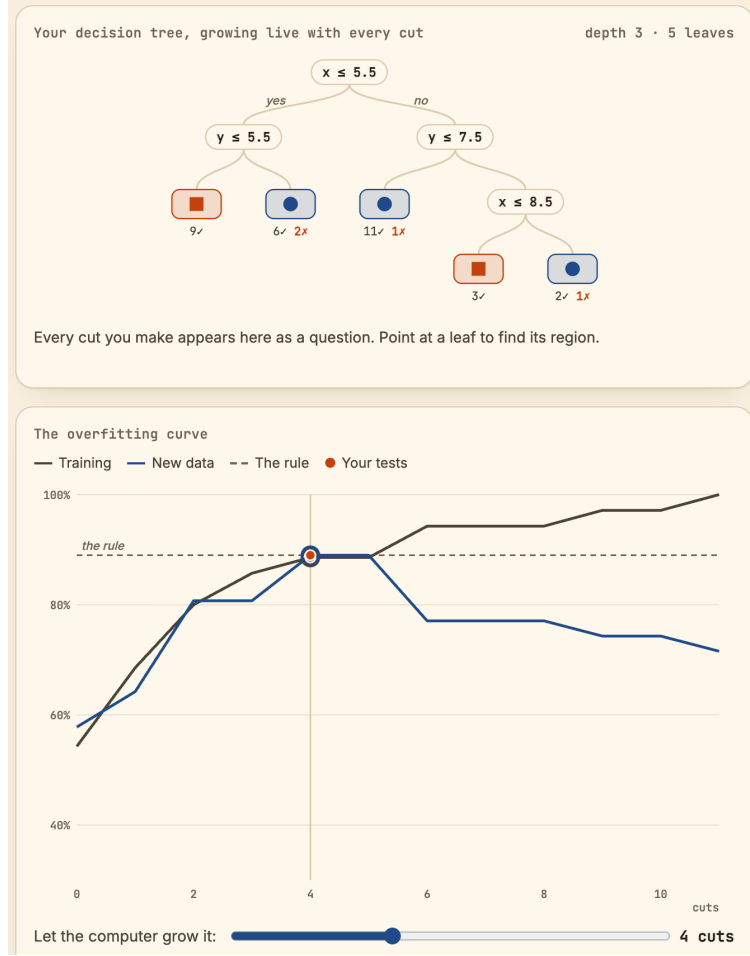


Figure 3: The tree diagram and the overfitting curve of level 1. Each leaf shows its points right and wrong; the curve shows training accuracy and accuracy on new data for the trees the library grows, with the rule dashed and the player’s test as a dot at four cuts.

#	Level	Points	Flipped	Classes	The rule			The full tree	
					Cuts	Training	New data	Cuts	New data
1	Pattern and noise	35	4	2	4	89%	89%	11	72%
2	Three together	31	1	2	3	97%	96%	5	87%
3	Too simple	38	3	2	5	92%	92%	10	75%
4	Few points	13	1	2	3	92%	92%	4	70%
5	Heavy noise	36	7	2	2	81%	81%	12	61%
6	Three classes	34	3	3	4	91%	91%	11	83%
7	Beat the library	38	3	2	3	92%	92%	13	70%
8	Prune it	37	4	2	3	89%	89%	10	76%
9	Two kinds of noise	38	4	3	5	89%	90%	10	73%

Table 3: The nine boards. “The rule” is the tree the points came from; “the full tree” is the end of the best-first growth, 100% on training on every board.

6 The nine experiments

The levels open in order: a level opens when every level before it is solved (a button opens them all). Table 3 gives the numbers of each board, all computed by the simulator’s own modules; Figure 5 draws the nine curves.

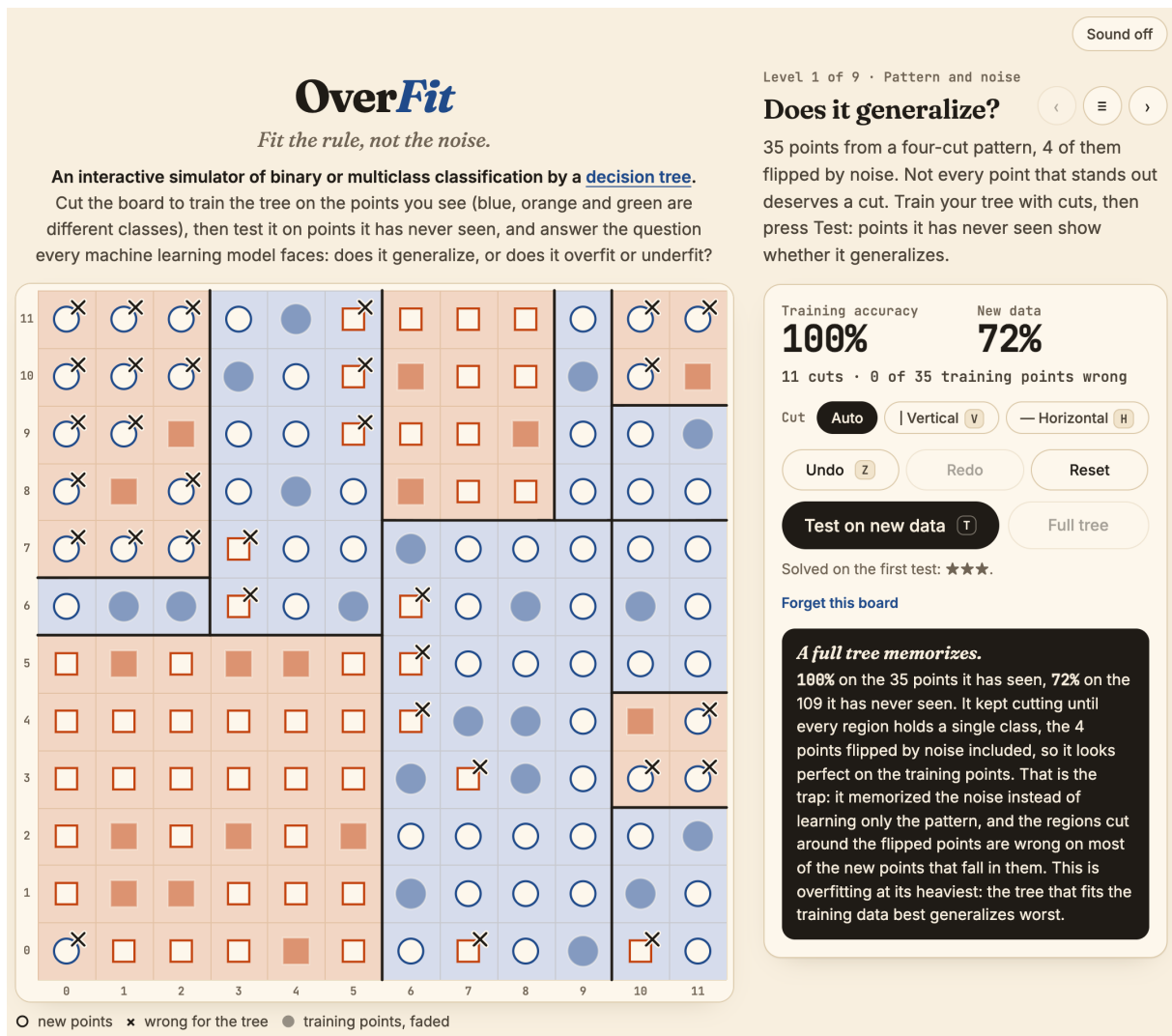


Figure 4: The full tree of level 1 after a test: 11 cuts, every training point right, 72% on new data. The × marks on the hollow points cluster around the regions cut around flipped points.

1. Pattern and noise. 35 points from a four-cut rule, 4 of them flipped. The trap is at the top left: two flipped orange points side by side, at (1, 8) and (2, 9). With 4 flips in 35 points, two of them next to each other is not rare, and a region cut around them holds nothing but noise. Four cuts score 89% on training and 89% on new data; the full tree, 11 cuts, falls to 72%. Figure 6a draws the board with its rule.

2. Three together. 31 points, one flip. The point of the level is the opposite one: the three orange points in a corner are not noise, since three flips close together would be very unlikely with one flip on the board. They are a real part of the pattern, the new points there are orange too, and ignoring them costs at least three points of accuracy on new data (a test checks it). A single point that disagrees with its neighbors is the flip.

3. Too simple. The border steps down the board like a staircase of five cuts, with a middle step only three columns wide. Every tree with three or four cuts is clearly worse on new data, which is what underfitting looks like: part of the border is missed, so both scores are low. The level also says what the board cannot show: a truly diagonal border has no right number of steps, and the more data, the more steps a tree can afford.

4. Few points. 13 points, one flip. Two orange points in the top right corner cannot both be noise, so there is an orange region there; the single orange point among the blue ones is the flip. A region cut around it is large, since there are few points to bound it, and almost every new point in it is wrong: the full tree, only four cuts, falls to 70% while the rule keeps 92%. The less data, the simpler the tree must be.

5. Heavy noise. 36 points, 7 flipped, about one in five. The rule itself scores 81% on new data and on average no tree can do better: that is the Bayes error of the board. With this much noise, flips can sit close together, as three blue points do in one column, and the full tree, 12 cuts, memorizes every one of them and falls to 61%.

6. Three classes. Green triangles join the board. Nothing changes in the model: each region still predicts its most common class, Gini still measures how mixed it is, and the four cuts of the rule generalize at 91%. Each flipped point took one of the other two colors, so one stray point of a third color inside a region is noise, not a region of its own.

7. Beat the library. Four quadrants, blue on one diagonal and orange on the other: the XOR pattern. A cut down or across the middle leaves both sides half blue and half orange, so no first cut lowers the Gini impurity, and the greedy growth takes the small drop the noise offers, in the wrong place: its first cut is $y \leq 4.5$. With three cuts it scores 50% on new data, the best it ever reaches is 75% at six cuts, and the full tree, 13 cuts, ends at 70%; the three cuts of the rule generalize at 92% (Figures 6b and 7). A greedy algorithm looks one cut ahead, and this pattern only shows after two. This is the one board whose curve does not peak at the rule's cuts, by design, and the level is the only one where the player is invited to grow the library's tree and compare.

8. Prune it. The level opens on the full tree, 10 cuts, every training point right, the 4 flipped included. The player removes the cuts that only chase noise, on the board or in the diagram, and tests. Cost-complexity pruning (CART; `ccp_alpha` in scikit-learn) works the same way: grow the tree in full, then remove first the cuts that lower the impurity of the training points the least for each leaf they add, misclassification error in the original CART, impurity in scikit-learn. A test of the suite runs that weakest-link pruning on this board and checks that the sequence of trees passes through the three-cut pattern. Pruned too far, the verdict says which cut the pattern needed.

9. Two kinds of noise. Three classes and a five-cut rule. With three colors a flipped point can take either of the other two, so two stray points side by side can look like two different things: a blue point and a green point at the bottom of the orange strip, an orange point and a green point among the blue ones. All four are noise. The five cuts of the rule score 89% on training and 90% on new data; the full tree falls to 73%.

7 Choosing the boards

7.1 The checks every board passes

A board teaches only if its numbers agree with its text, so each board is accepted only if it passes three checks, which run in the test suite on every level and on twenty random boards.

- **The rule explains the points.** Every training point has the class of its cell under the rule, except the flipped ones, which have another class, sit at least two cells from every edge of their rule region, and are not in a corner.

- **The curve has the right shape.** Along the best-first growth, training accuracy never falls; accuracy on new data reaches its best at the rule’s own number of cuts, where it equals the rule’s score, and never rises again after that; the full tree gets every training point right and loses at least eight points of accuracy on new data. Level 7 is exempted from the peak and the rise only.
- **The board is fair.** Among all trees with at most as many cuts as the rule that are wrong on exactly the flipped training points, the worst score on new data is at least the rule’s. A player who finds the rule’s mistakes with the rule’s number of cuts can therefore never be told the tree generalizes worse because of a border that runs through empty cells; the solved condition can only fail on a tree that is larger or differently wrong.

7.2 An exact search over trees

The fairness check is an exact search: for a region, a fallback class and a budget of k cuts, the best (or worst) count of new points right is the best over every cut line and every split of the remaining budget between the two sides, and a leaf counts the points of its majority class. The counts come from the prefix-sum tables of Section 5, and the search is memoized on (region, fallback, budget), so it runs in milliseconds on a board; a test compares it with trying every tree one by one on small budgets. The same search gives the *most* new points right among all trees of a given size, which the tests of levels 3 and 4 use to state that every tree with fewer cuts than the rule is clearly worse.

7.3 How the boards were found

The rule of each level was written by hand for what the level should show. Its points were then drawn in thousands of candidate boards, generated in a headless browser with the simulator’s own modules, each candidate placing points region by region with a minimum number per region and a share of points close to the rule’s edges, and only candidates that passed the three checks were kept; among those, the board was chosen by eye for how it reads. The share of points near the edges matters: a staircase border, or a sparse board, is never reproduced by the greedy growth unless some points sit against its steps. A two-column step was never reproduced at all, which is why the middle step of level 3 is three columns wide.

8 How a decision tree works: the cards

Under the levels, seven cards explain the model, each with a short text and a small interactive board drawn like the main one (Figure 9). Four of them share one board, 30 points with 3 flipped, from a three-cut rule that scores 90% on training and 90% on new data; the underfitting card has a board of its own with no noise; none equals a level.

9 Architecture

9.1 Modules

The simulator is a set of JavaScript modules loaded directly by the browser, with no bundler and no library (Table 5). The engine knows nothing about the page: `board.js` parses a board drawn as twelve lines of text and counts points in rectangles with prefix sums; `tree.js` is the player’s model, with cuts, votes, previews, undo and snapshots; `rule.js` classifies by a rule and measures distances to its edges; `growth.js` is the best-first growth; `newdata.js` draws the new data and scores a tree; `random.js` is the generator. `experiment.js` prepares a level, points, new data and curve, and holds the three checks of Section 7; `levels.js` is the nine boards as data.

Card	What the board does
One question at a time	click a grid line to cut; the cut becomes a question of the diagram, and a point shows its path
A leaf votes	click a point to switch its class; every leaf votes again; “Show a tie” shows the tie rule
Generalization	“3 cuts” or “Full tree”, then “Test on new data”: 90% / 90% against 100% / 75%
Overfitting	a slider grows the tree best cut first; the curve under it opens the gap between the two scores
Underfitting	a frame of blue around an orange square, no noise: with 1, 2 and 3 cuts both scores stay low and close, 61%, 78%, 89%; with 4 cuts both are 100%
Greedy growth	“Make this cut” takes the cut with the largest drop in Gini impurity; the best three of the cuts tried are ranked under the board, with their drops
Stopping in time	a <code>max_depth</code> slider from 1 to None on a tree grown depth first, as CART grows it; with no limit, 12 leaves at depth 7, 100% on training and 75% on new data

Table 4: The seven cards.

`main.js` is the page: drawing, input, scores, verdicts, stars and storage. Each card is a module of its own on a shared mini board, `demos/mini.js`.

Part	Files	Lines
Engine	<code>board.js</code> , <code>tree.js</code> , <code>rule.js</code> , <code>growth.js</code> , <code>newdata.js</code> , <code>random.js</code>	482
Experiments	<code>experiment.js</code> , <code>levels.js</code> , <code>randomboard.js</code>	452
The page	<code>main.js</code> , <code>explain.js</code> , <code>sound.js</code> , <code>storage.js</code> , <code>theme-init.js</code> , <code>analytics.js</code>	1 576
Cards	<code>demos/</code> (8 modules)	1 299
Markup and style	<code>index.html</code> , <code>overfit.css</code>	655
Tests	<code>harness.js</code> , three test files, <code>run.js</code>	541
Total		5 005

Table 5: The source, by part.

9.2 Drawing

The board is a canvas, redrawn on every change at the device’s pixel density: regions tinted with their class, grid lines, points as circles, squares and triangles, the $\times$ marks, the dashed preview of the next cut, the new points hollow after a test and the training points faded. The tree diagram and the overfitting curve are SVG, built from the tree and the curve data, so their text stays sharp and selectable. Cuts animate briefly as they are made; nothing on the page blinks or pulses, and readers who ask their system for less motion get none.

9.3 Input

With a mouse, moving over a grid line previews the cut and shows how training accuracy would change; a click makes it; a key, V or H, keeps cuts vertical or horizontal; hovering a point shows its path. On touch screens a drag along the grid cuts, a tap on a point shows its path, and a tap on a cut selects it for removal. With the keyboard, the arrows move a cursor on the grid, Enter cuts, T tests, Z undoes, R resets, Delete removes the selected cut and N goes to the next level; keys work only while the board is on screen. Every control is also a button on the page.

9.4 For every reader

Classes have shapes as well as colors. The page has a light and a dark theme, the site’s own. Screen readers get the board as an application with a description, the two scores as text and

the verdicts in a live region. Fonts are the site's, served from the site. The page has no inline script, so the site's content security policy, which allows scripts from the site only, applies to it unchanged; the copy on the site is made by a build script that checks that, copies the modules, adds the site's icons and fonts, and refuses to run if a referenced file is missing.

9.5 Storage

The browser's storage keeps, per board, the best stars, the number of tests made, the trees already tested (as the set of their predictions, so an equivalent tree is not charged again), whether help was used, and whether the board was ever solved, which keeps the later levels open even if its stars are forgotten. A board's key is its name plus a hash of its rows, its rule and its seed.

10 Testing

The tests run in the browser, on the very modules of the simulator, with a harness of 49 lines written for the project: `test(name, fn)` registers a test and `assert.ok`, `equal` and `deepEqual` throw on failure. There are 37 tests in three files.

- **engine** (14): boards are read top line first with y upward and refused when malformed; box counts match a plain count on random boards; the vote and its tie rule; Gini is 0 when pure, 0.5 at 50/50 and 2/3 for an even three-way mix; a cut splits one region and each side votes; a cut must leave a point on each side; undo, redo and reset walk the history; removing a cut prunes everything below it; a snapshot rebuilds the same tree; the preview gives the sides, the points fixed and the impurity; the path of a cell lists its questions; lines are tried once per gap, in its middle.
- **levels** (18): nine levels with unique ids and every field filled; every board passes the three checks of Section 7; the new data has one point per empty cell with the board's share of noise; the fast search agrees with trying every tree one by one; and one test per level states what the level claims: the pair trap of level 1, the corner group of level 2 (ignoring it costs at least three points), the three-column step of level 3 (every three- or four-cut tree is clearly worse), the corner pair of level 4 (the full tree loses the most of all levels, the XOR board excepted), the noise share and the rule's score of level 5, the colors of the flips of level 6, the wrong first cut of level 7, the pruning sequence of level 8, and the two pairs of level 9; every goal and lesson mentions generalizing and quotes the computed numbers.
- **random boards** (5): the generator of the cards' board gives the same board for the same seed, another for another seed, twenty random boards pass the checks, three classes on request, and a board is made in well under a second; the board the cards share has 30 points, 3 flipped, a rule of three cuts and 90% on both scores.

The suite is a page of the project, `tests/index.html`; a script runs it in a headless browser and prints the report: 37 passed, 0 failed, in 343 ms.

11 Results

The boards. Over the nine levels the player sees 300 training points, 30 of them flipped, and tests on 996 new points. The rule's score on new data runs from 81% (level 5, one point in five flipped) to 96% (level 2); the full tree loses between 8 points (level 6, where the three flips are each of a different color and the regions around them are small) and 23 points (level 7) against the rule, and its depth runs from 3 to 7. On level 7 the greedy library scores 50% with the rule's three cuts, 25 points under the rule.

Size and speed. The page on the site is 28 files and 230 KB, fonts excluded. Preparing a level, that is, drawing its new data and growing the whole curve, takes between 0.1 and 0.4 ms in Chrome on an Apple M5 Max; twenty preparations of the largest board take 2.2 ms. The fairness check of a board, the exact search of Section 7, takes under a millisecond with memoization. The whole test suite, with its twenty random boards, runs in a third of a second.

12 Conclusion

OverFit puts the central idea of supervised learning on a board: the data a model is trained on is not the data it is judged on. Everything the player does, a cut, a vote, a test, is what a tree library does, with the same impurity, the same thresholds and the same growth, and everything the player reads, a verdict, a curve, a star, answers one question. The honesty of the simulator rests on three choices: new data fixed by a seed, so that every score can be reproduced; boards chosen by an exact search and held to their claims by the test suite; and a test set that costs something to look at. The trees of the simulator are binary, one question per cut on one feature, as in CART and scikit-learn; the author's thesis built a different kind of tree, with native categorical attributes and multiway splits, and the simulator is the natural companion of that work: the two together cover how a tree is grown and why it must be stopped. The simulator runs entirely in the browser, with no server and no library, and is open to everyone at giuseppemuschetta.com/overfit.

Reference. L. Breiman, J. Friedman, R. Olshen and C. Stone, *Classification and Regression Trees*, Wadsworth, 1984.

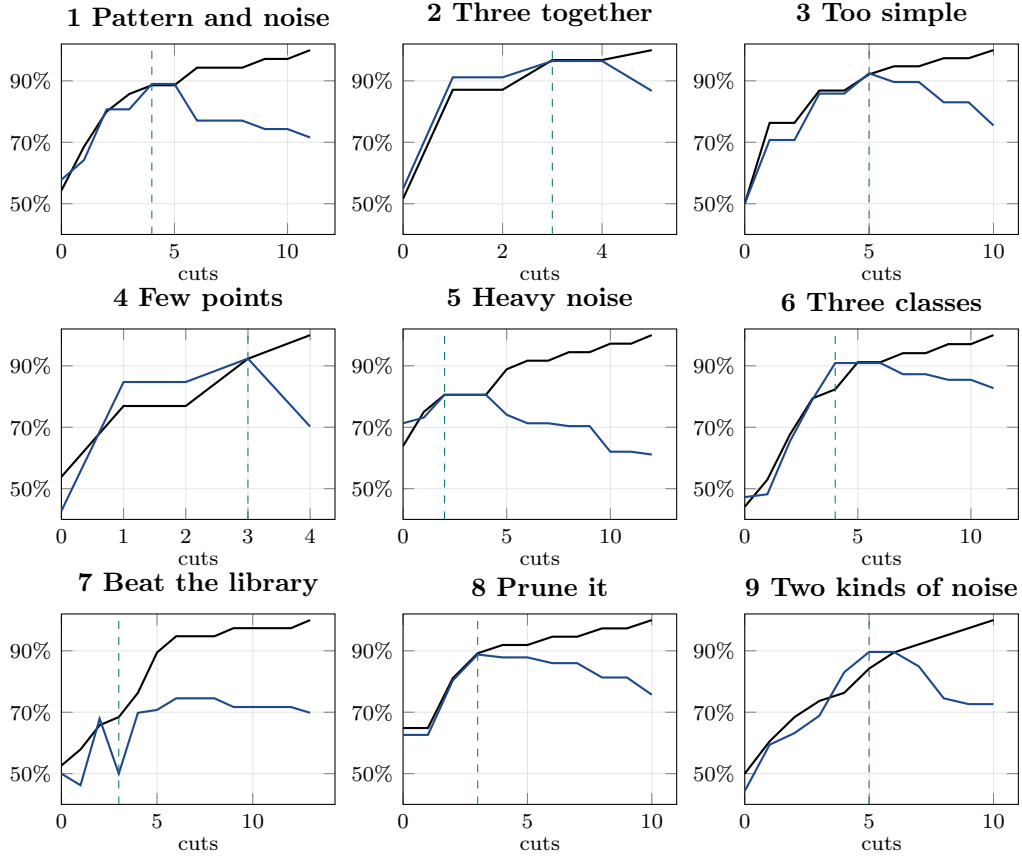
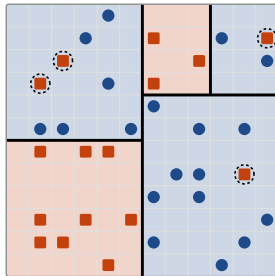
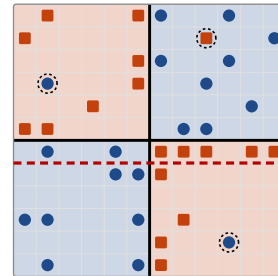


Figure 5: The overfitting curve of every level: training accuracy (black) and accuracy on new data (blue) of the trees grown best cut first, from no cut to the full tree; the dashed line marks the rule's number of cuts. On eight boards the blue curve peaks there and never rises again. On level 7 the library's first cut is wrong, so its curve starts low and never reaches the rule.



(a) Level 1 with the rule's four cuts. The dotted rings mark the four flipped points, which the player never sees marked; two of them sit side by side.



(b) Level 7, the XOR board, with the rule's three cuts. The red dashed line is the first cut of the greedy growth, $y \leq 4.5$: it lowers the impurity a little, thanks to the noise, and leads nowhere.

Figure 6: Two boards drawn from the level data, with the rule's regions tinted.

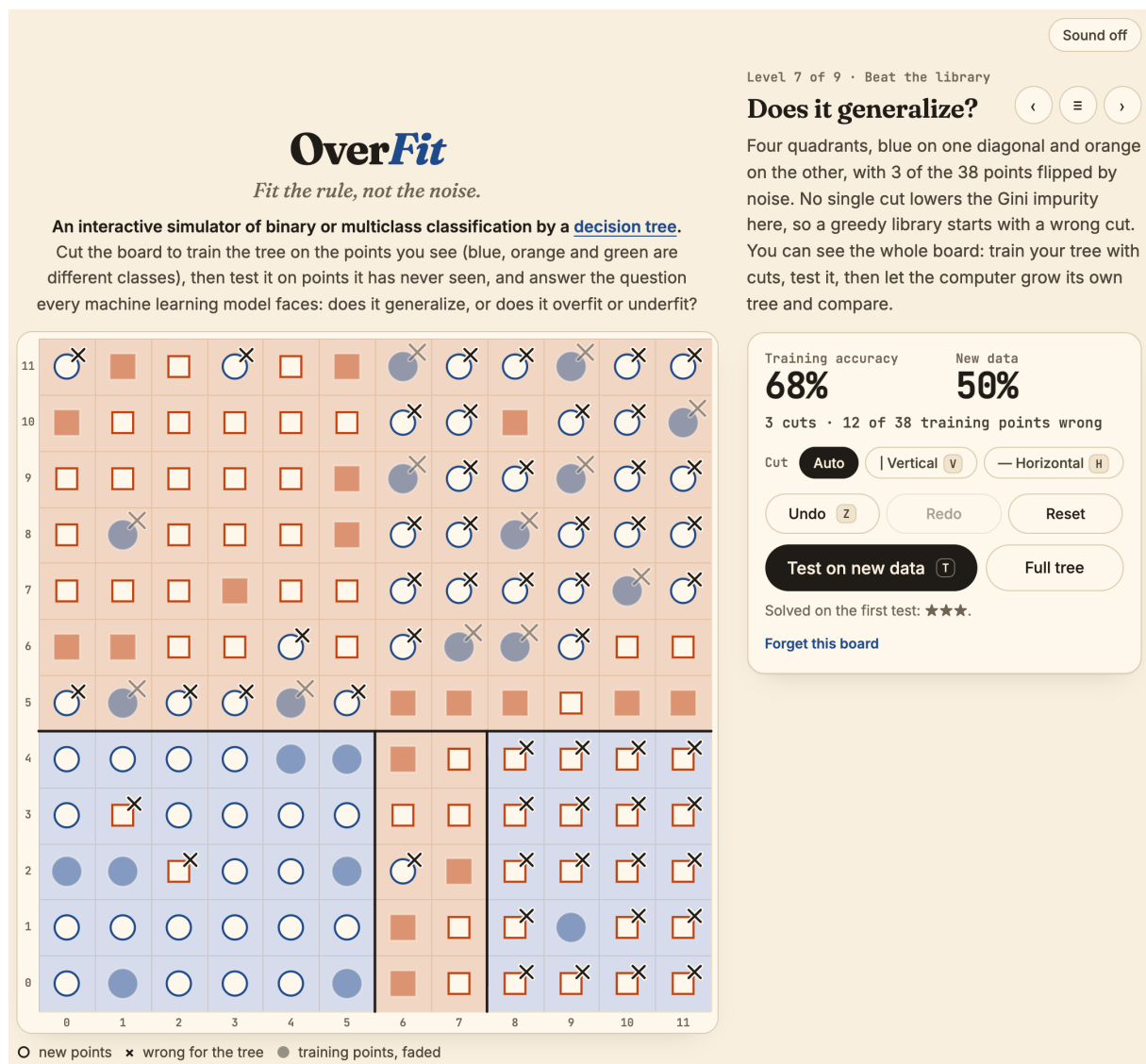


Figure 7: Level 7 with the library's tree at three cuts, put on the board by the slider: 68% on training, 50% on new data, 12 of the 38 training points wrong. The player's three cuts, tested first, had solved the level at 92%.

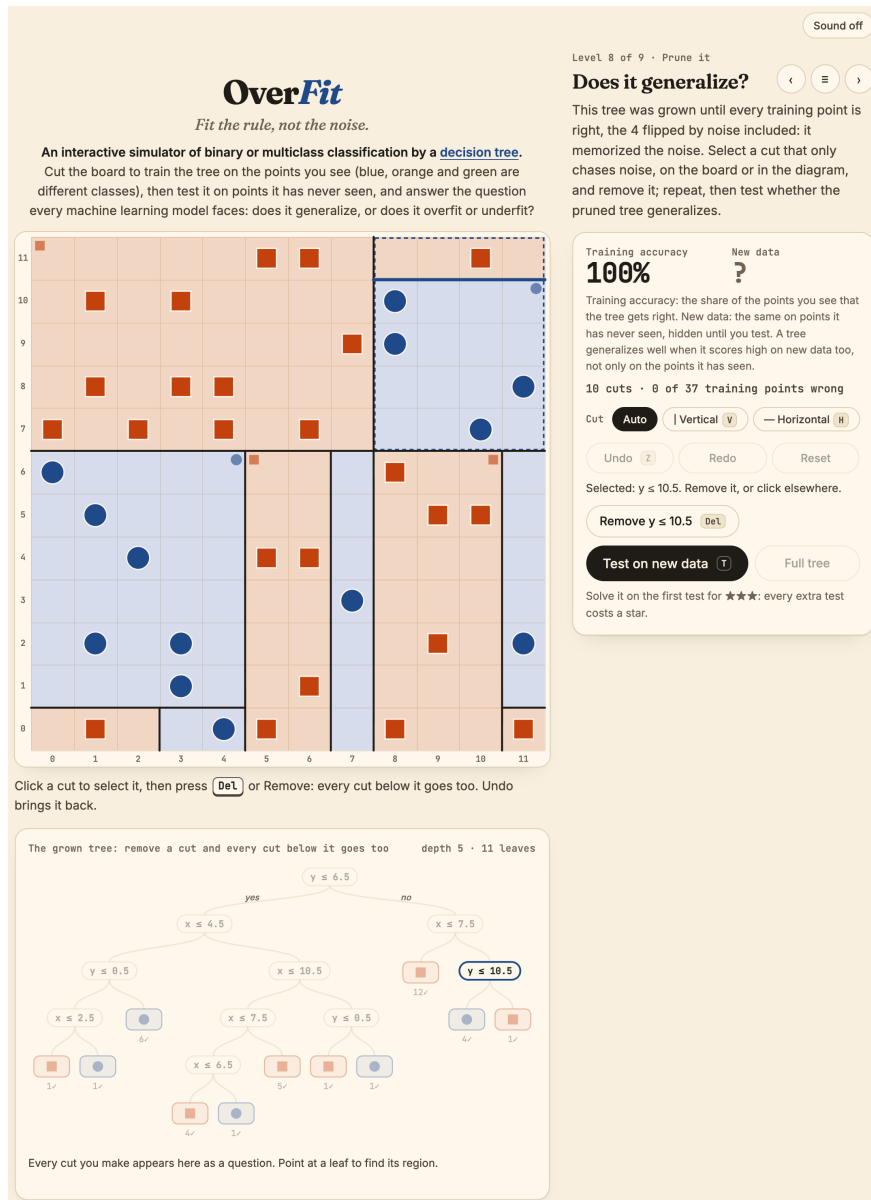
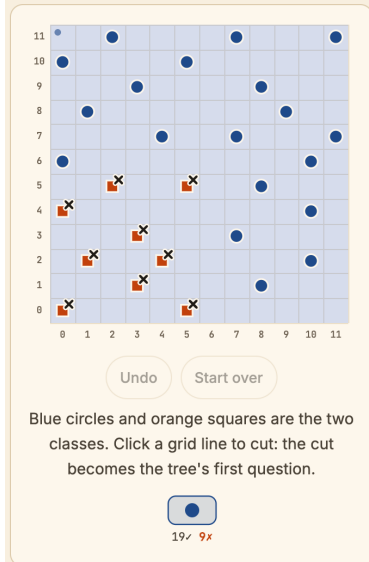
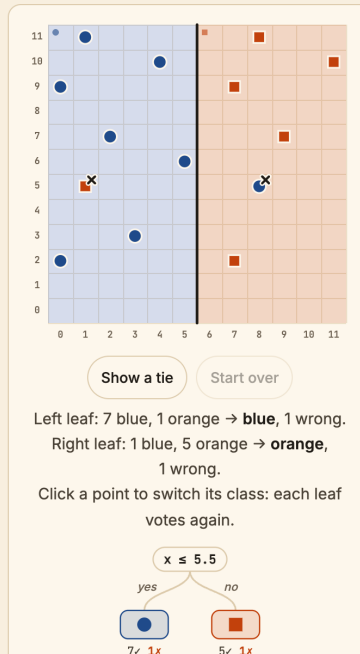


Figure 8: Level 8 as it opens: the full tree, 10 cuts, 100% on training, with the cut $y \leq 10.5$ selected on the board (dashed) and in the diagram, ready to be removed with its subtree.

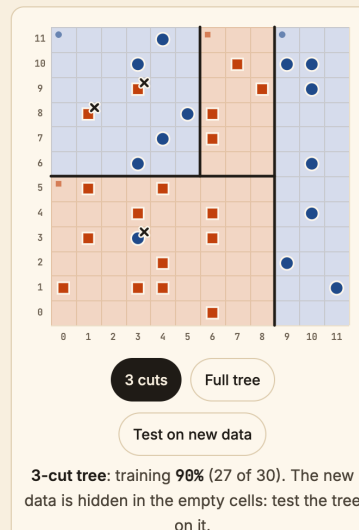
A decision tree classifies a point with yes/no questions, each about one feature: is $x \leq 4.5$? Then is $y \leq 6.5$? The answers lead from the root to a leaf, and the leaf gives the class. On the board, every cut is a question and every region is a leaf.



A leaf predicts the most common class of its training points; a training point of another class in it is a mistake, the small x on the board. Here a tie keeps the color of the region it was cut from, a rule of this game; libraries break ties their own way.

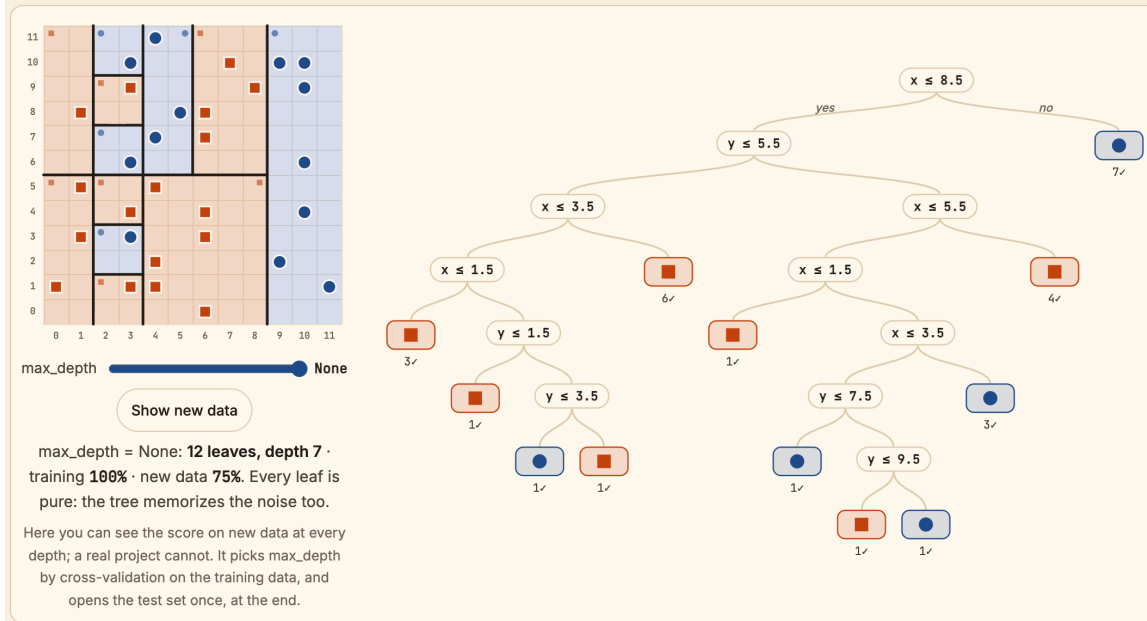


A model is trained on some data and judged on data it has never seen. It generalizes well when it does well on that new data too, about as well as on its training data: it learned the pattern, not the points. A tree that memorizes the points scores high on training and lower on new data (overfitting); a tree too simple scores low on both (underfitting). Here the training points are the ones you cut around; the new data is one point in every empty cell, from the same rule with the same share of noise, hidden until you test.

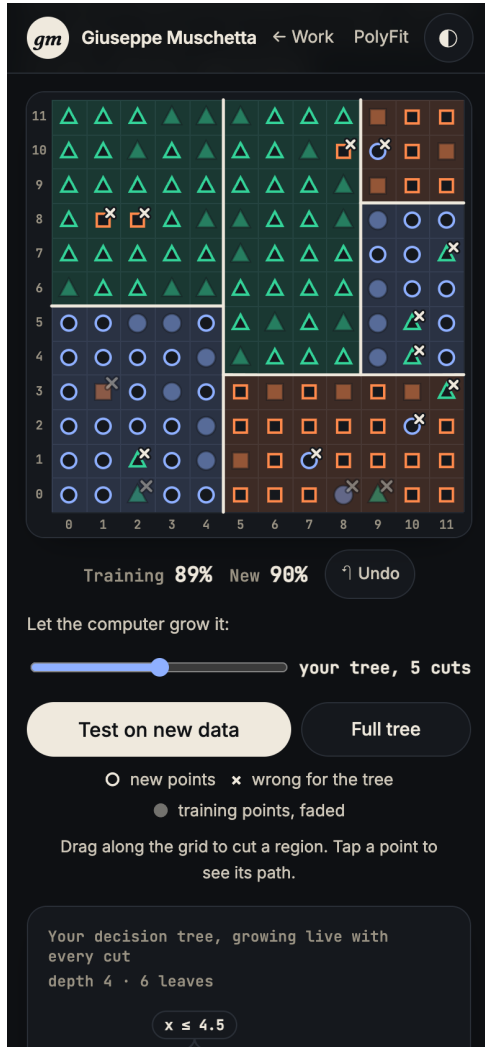


Stopping in time

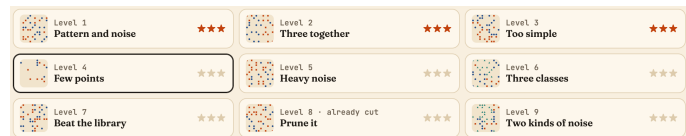
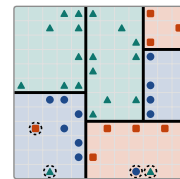
By default scikit-learn grows a tree until every leaf is pure: the Full tree of this page, 100% on training and worse on new data. To keep it from memorizing, you stop it early (a maximum depth, a minimum number of points per leaf, a maximum number of leaves) or grow it in full and prune it (cost-complexity pruning, `ccp_alpha`). These settings are chosen by cross-validation or on a validation set; the test set is opened once, at the end.



19



(a) Level 9 on a phone, in the dark theme, after a test: 89% on training, 90% on new data.



(b) The nine tiles with the first three levels solved, and the rule of level 9: three classes, five cuts, and in two regions two flipped points of different colors side by side.

Figure 11: The simulator on a phone, the level tiles, and the last board.