Haberman's Survival

08 Sep 2026
Somony Pou

Dataset contains cases from study conducted on the survival of patients who had undergone surgery for breast cancer

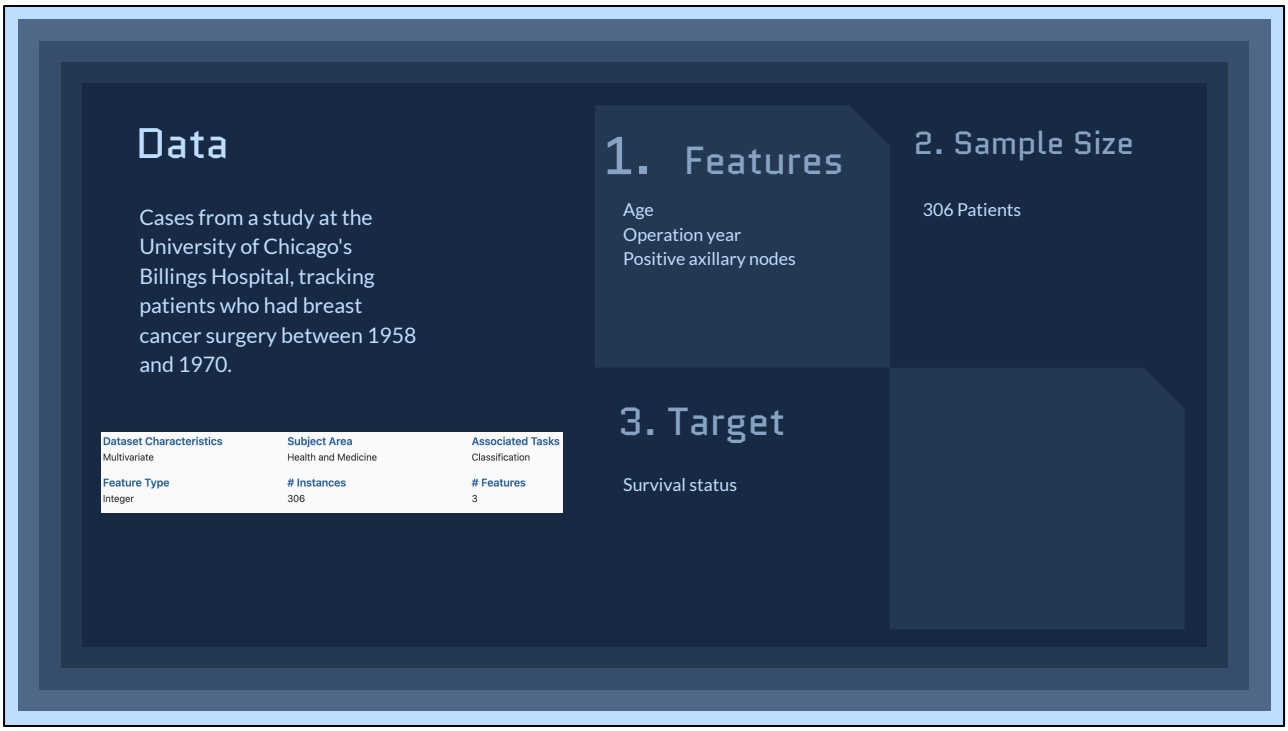
Agenda

Data [01]

Integration [02]

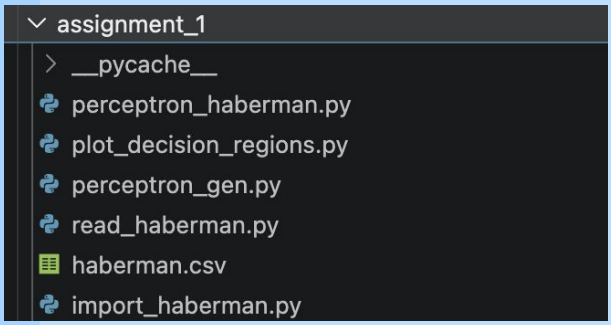
MI and Human Experience [03]

Perspectives [04]



operation_year — the year of surgery (recorded as years since 1900, so "58" means 1958)
positive_auxillary_nodes — the number of positive axillary lymph nodes detected (a key clinical indicator in breast cancer prognosis)
a binary label, 1 means the patient survived 5 years or longer after surgery, 2 means the patient died within 5 years.

Code used




```
weights: [0.05924345 0.04288244]
bias: -0.08400000000000006
sum errors_ = 3728
last epoch = 74
final training mistakes = 225
Unique predictions: [1]
Predicted 0 (survived): 0
Predicted 1 (died within 5 yrs): 306
Actual 0 (survived): 225
Actual 1 (died within 5 yrs): 81
```

import_haberman.py — one-time setup, pulls the raw data from UCI and saves it as haberman.csv with proper column headers.haberman.csv — the actual dataset, 306 patient records, everything else reads from this.

read_haberman.py — quick inspection step, loads the CSV and checks the data looks right before trusting it.

perceptron_gen.py — defines the Perceptron model itself, the learning algorithm (fit, predict, etc.). No plotting, just the "brain."

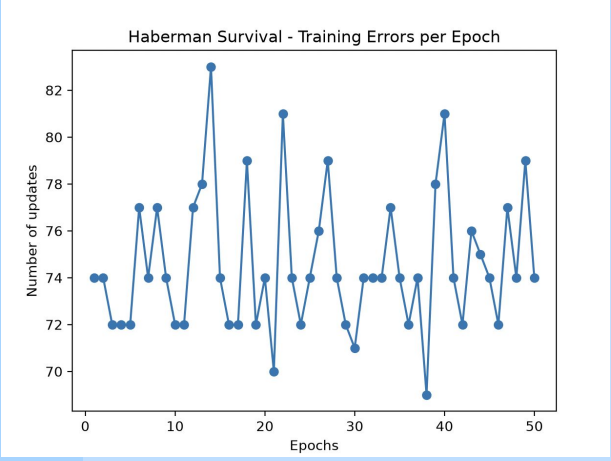
plot_decision_regions.py — a reusable graphing tool. Takes a trained model, predicts across a full grid of points, and colors the background by class.

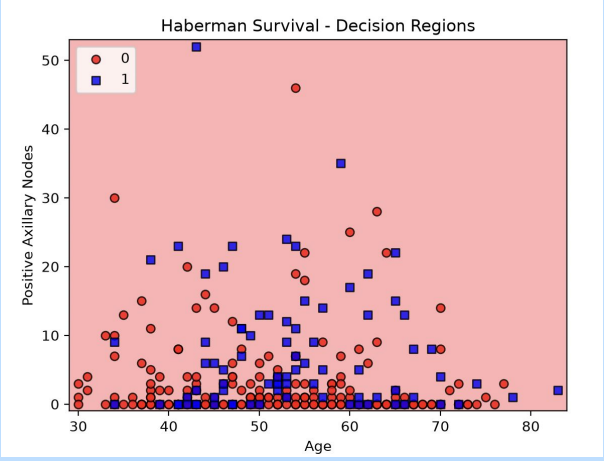
perceptron_haberman.py — the script that ties it all together: loads the CSV, trains the perceptron, then produces both graphs, the error-per-epoch line plot and the decision regions plot, using the two files above.

Bottom line: data → model → visualization tool → one script that runs all three and produces the graphs.

Initial plots and graphs

```
def __init__(self, eta=0.01, n_iter=50,
random_state=1):
```





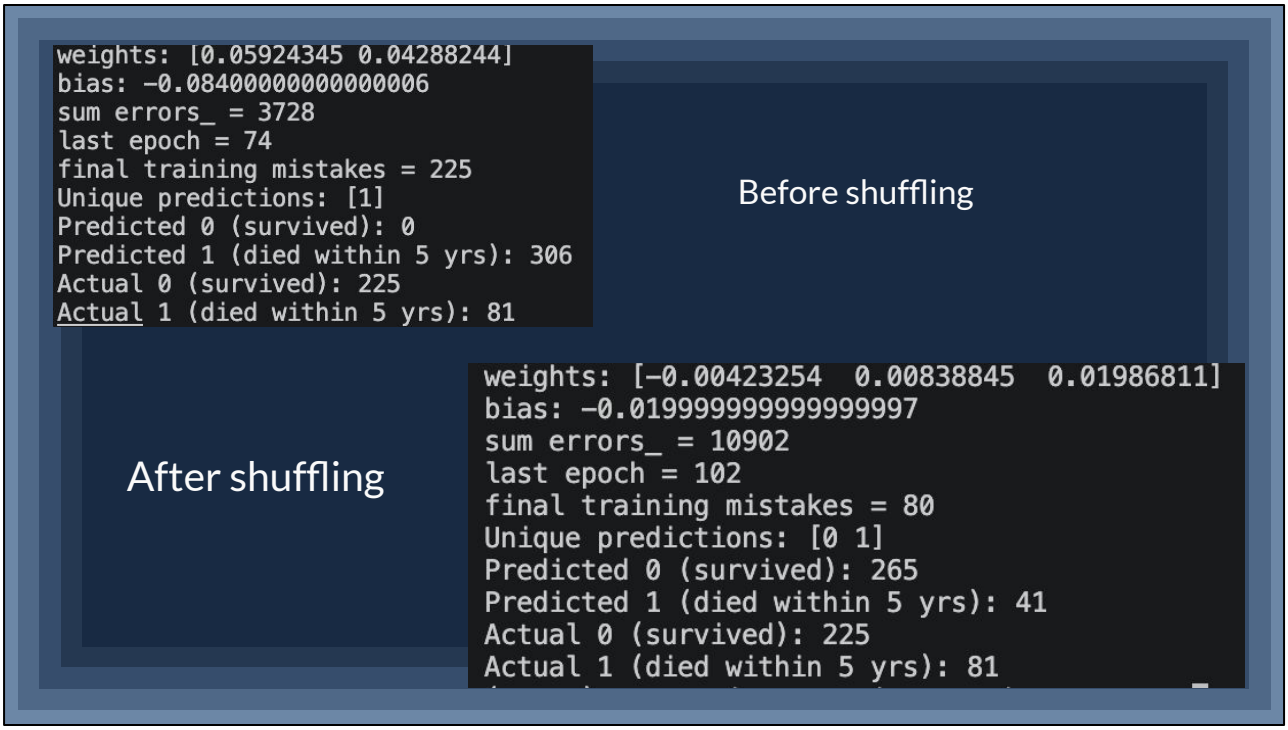
The error graph never drops to zero, it just bounces between about 69 and 83 mistakes for all 50 epochs. Compare that to Iris, where errors dropped fast and flattened at zero. Here, no line position works better, so it just keeps making roughly the same number of mistakes forever, that's the sign that age and positive nodes can't cleanly separate survival outcomes with a straight line.

The decision regions plot confirms it, the whole background is one solid color, meaning the model gave up finding a real boundary and just predicts one class everywhere. The red and blue dots are heavily mixed together, especially near the bottom, so there was never a clean split for it to find.

The takeaway: Iris gave a clean, separable line; Haberman doesn't, because real medical outcomes are far messier than flower measurements. This is the perceptron hitting its core limitation, it can only draw straight lines, and this data just isn't linearly separable, which is exactly why more advanced classifiers exist.

How can we improve this?

Lets try shuffling the data.



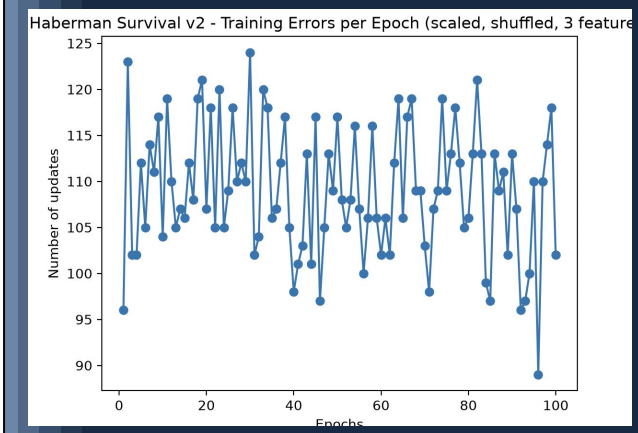
Final training mistakes dropped from 225 to 80 (out of 306), roughly 74% accuracy instead of the model just always guessing one class. It's still not linearly separable (this is real medical data, a perfect split was never realistic), but the model is now genuinely discriminating between the two classes instead of collapsing into one answer.

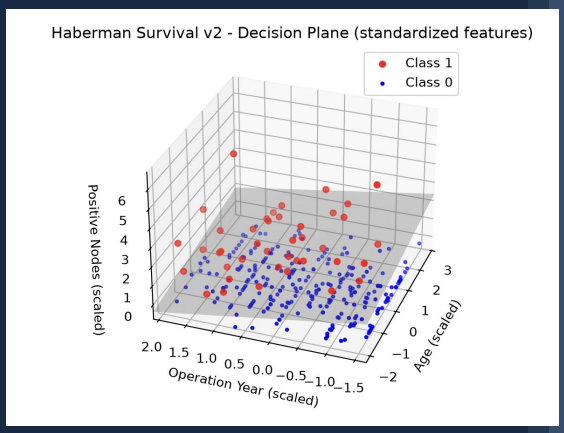
Change 1 — Shuffled training data each epoch (Perceptron_w_s), so a fixed example order doesn't bias learning.

Change 2 — Used all three features (age, operation_year, positive_nodes) instead of two, enabling a full 3D decision plane.

Change 3 — Standardized the features. This mattered most: age and positive_nodes were on wildly different scales, so raw magnitude was skewing the weighted sum regardless of actual predictive value. Scaling puts all features on equal footing.

Change 4 — Increased epochs (50→100) and eta (0.001→0.01) to match the new scaled data.





Age: mean ≈ 52.5, std ≈ 10.8. Formula: real age = (scaled value × 10.8) + 52.5

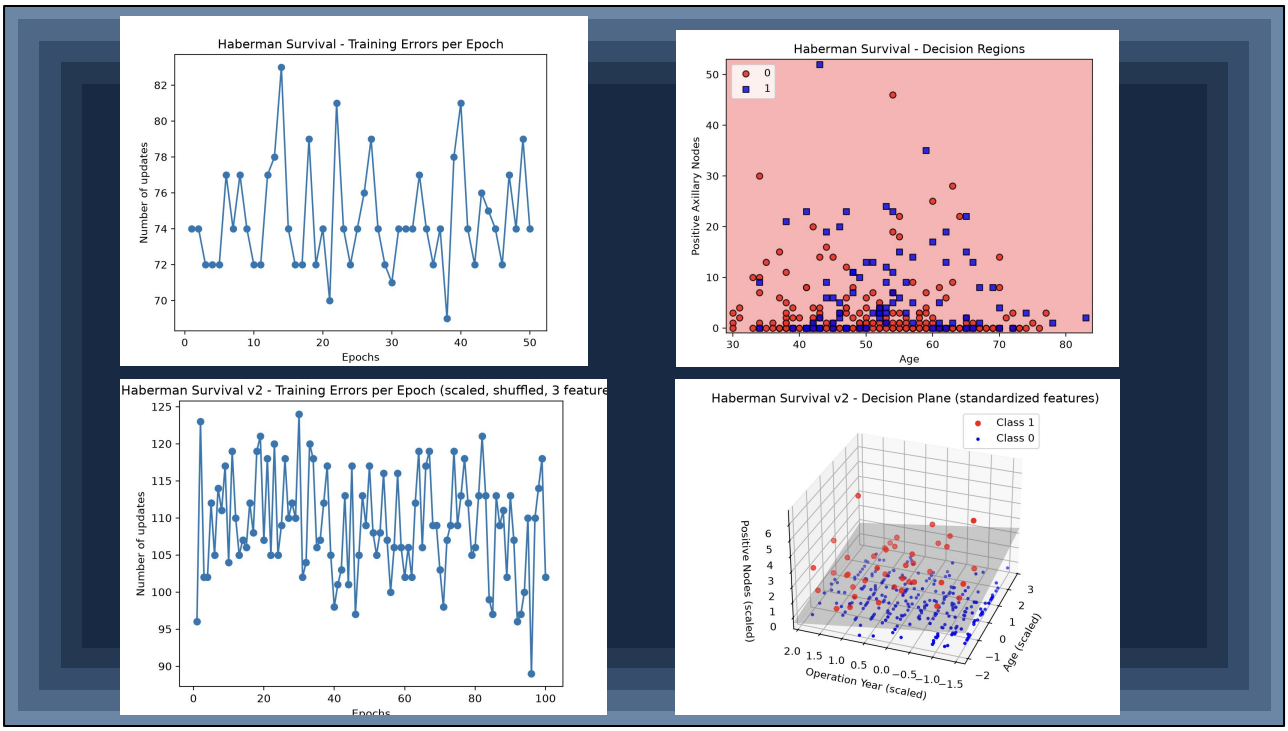
- scaled -2 → ~31 years old
- scaled -1 → ~42 years old
- scaled 0 → ~52.5 years old (the average)
- scaled 1 → ~63 years old
- scaled 2 → ~74 years old
- scaled 3 → ~85 years old

Operation Year: mean ≈ 62.9 (i.e., 1962.9), std ≈ 3.2. Formula: real year = (scaled value × 3.2) + 62.9, then add 1900

- scaled -1.5 → ~1958
- scaled 0 → ~1963 (the average)
- scaled 1 → ~1966
- scaled 2 → ~1969

Positive Nodes: mean ≈ 4.0, std ≈ 7.2. Formula: real nodes = (scaled value × 7.2) + 4.0

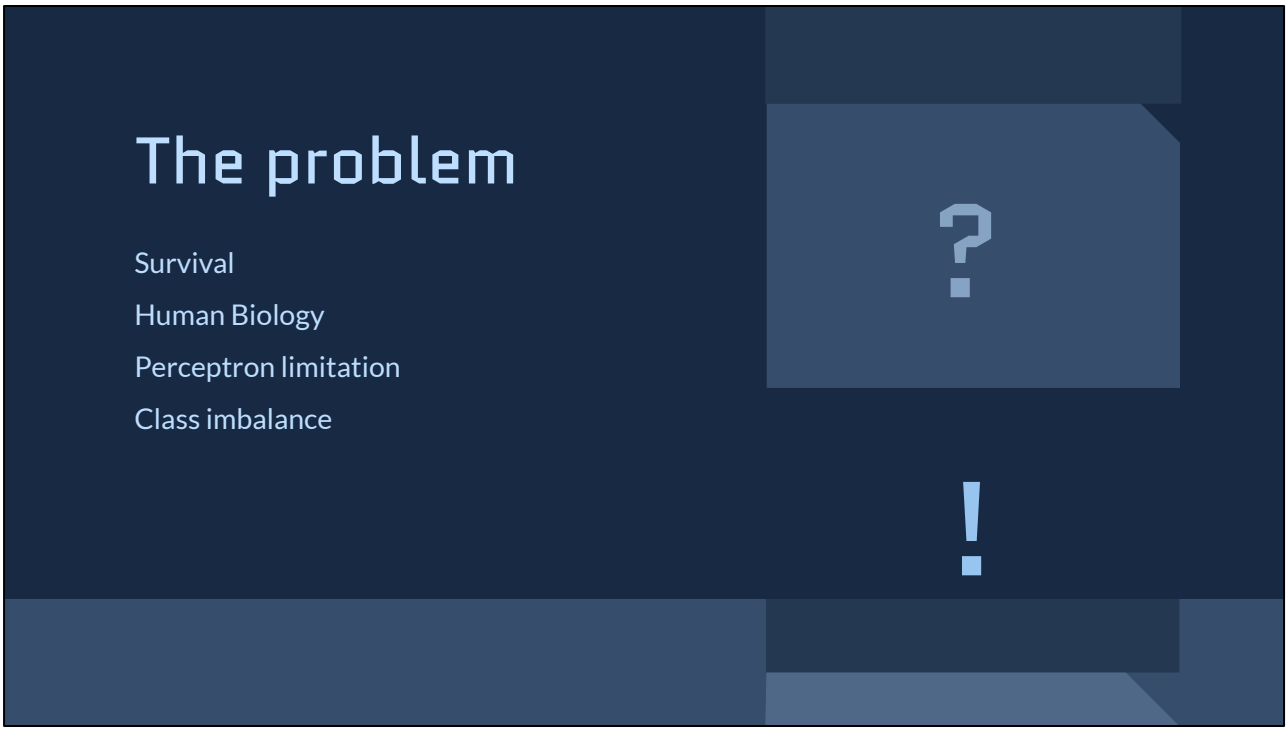
- scaled 0 → ~4 nodes (the average)
- scaled 1 → ~11 nodes
- scaled 3 → ~26 nodes
- scaled 6 → ~47 nodes



Error graph: still noisy, bouncing 90-125 mistakes per epoch, no downward trend. That's expected, since this data isn't linearly separable, the perceptron never locks into a stable solution, so weights keep oscillating instead of flattening like Iris did.

Decision plane: this is where the real improvement shows. Unlike v1's flat, single-color background, there's now a visible tilted plane with a real pattern, red dots (died) cluster toward higher positive_nodes, blue dots (survived) dominate lower nodes and younger age. Not a clean split, but a genuine, medically-sensible signal instead of one collapsed guess.

Key takeaway: a noisy training graph doesn't mean the model failed, it means the data resists a clean solution. The real proof of improvement is the final mistake count (225 → 80) and the interpretable plane, not the per-epoch noise.



Survival isn't really determined by just age and node count, real outcomes depend on tumor stage, treatment, genetics, and overall health, none of which this dataset includes. So these two features are an incomplete proxy for a much more complex reality.

Human biology also doesn't split cleanly like flower measurements do. You can see real overlap directly in the graph, patients with similar age and node counts had different outcomes, because of factors the model can't see. That's genuine biological variability, not noise you can clean away.

On top of that, a perceptron can only draw a straight line or flat plane. Even if there were a real pattern, if it's more complex than a straight cutoff, the perceptron simply can't represent it. Add in the class imbalance (far more survivors than non-survivors), and there's less signal available to define the minority class clearly.

Bottom line: Iris worked because species differences are clean and linearly separable. Human survival is multi-causal and messy, two features and a straight line were never going to fully capture it.

Integration

Dropped operation year.

Lets balance the data.

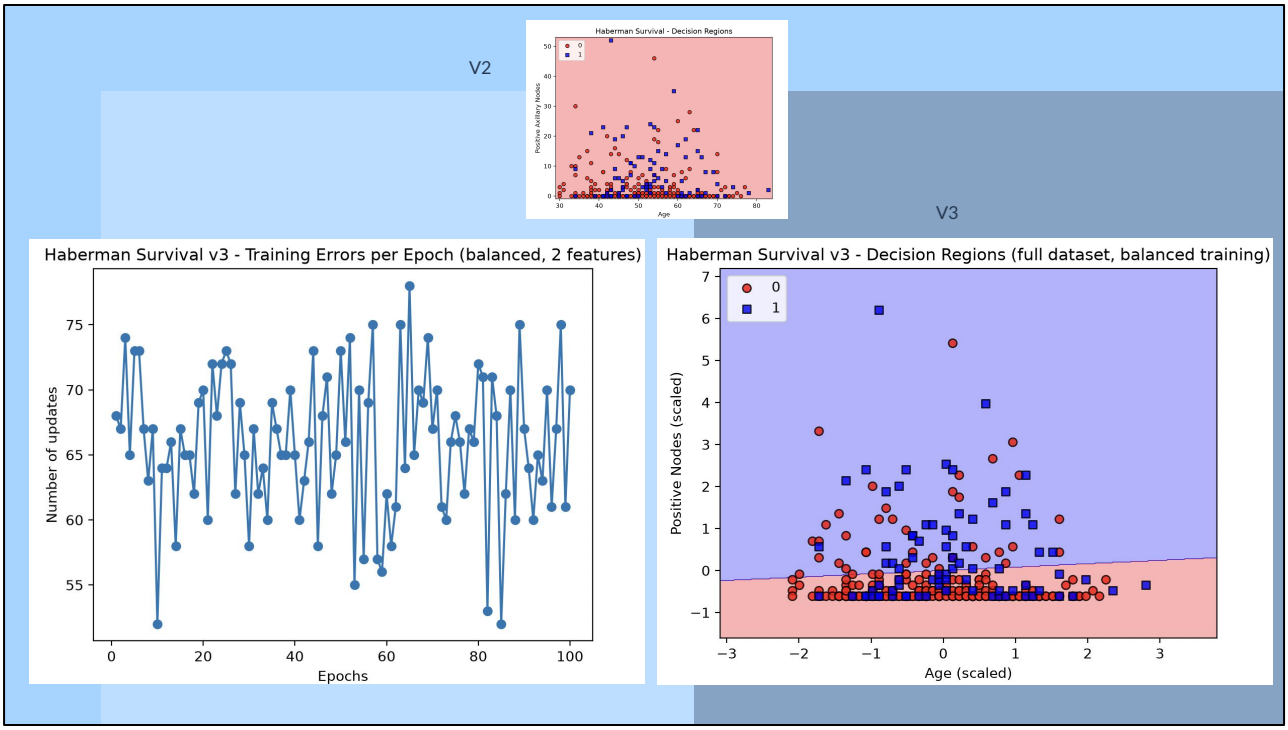
```
Balanced training set size: 162
Died in training set: 81
Survived in training set: 81
weights: [-0.00223412  0.02793288]
bias: -3.469446951953614e-18
Mistakes on balanced training set: 52 out of 162
Mistakes on FULL dataset: 82 out of 306
Accuracy on FULL dataset: 0.7320261437908497
Predicted 0 (survived) on full data: 229
Predicted 1 (died) on full data: 77
Actual 0 (survived): 225
Actual 1 (died): 81
```





Change 1 — Dropped operation_year. In v2, the trained weights showed operation_year barely mattered (weight ~0.008, next to nothing compared to positive_nodes' ~0.02). If a feature isn't pulling its weight, it's just adding noise, so v3 drops it and keeps only age and positive_nodes, the two features that actually mattered.

Change 2 — Balanced the training data. The real dataset has 225 "survived" patients but only 81 "died" patients. Training on that imbalance can make the model lazy, it can get a decent chunk of predictions "right" just by leaning toward the bigger group without actually learning the real pattern. The fix: build a training set using all 81 "died" patients plus a random sample of 81 "survived" patients, so the model sees an equal number of both outcomes. After training on that balanced set, we still test the final model against the full, real 306-patient dataset, so we get an honest read on real-world performance, not just performance on the artificially balanced sample.



The training errors are still similar but the graphs have a more define line.

This is much easier to read than v2's 3D plane, back to a simple 2D boundary with two clear regions instead of a plane you had to rotate to interpret.

The boundary is almost flat, meaning the model is basically deciding based on positive_nodes alone, age barely matters. That confirms what the weights already showed: nodes carry the real signal.

You can now directly see hits and misses: dots matching their background color are correct, dots clashing with it are mistakes, mostly clustered right around the boundary where node counts are low but outcomes still vary.

Bottom line: v2 proved a pattern existed but needed 3D to show it. v3 makes that same pattern clear in simple 2D, and reveals it's really just one feature doing the work.



The Limits of Clean Separation — The perceptron separated flowers cleanly but not human survival, not a bug, just honesty: survival depends on far more than three numbers.

The Data-Reality Gap — A model only knows what it's given. It has no concept of tumor stage or resilience, just the numbers handed to it, that gap between data and real life is where the risk lives.

The Power of Scale — The real value is scale, not replacing judgment. A model can learn from millions of cases, surfacing patterns no single doctor could ever notice (e.g., early cancer detection, sepsis prediction before symptoms).

Hidden Bias in "Objective" Numbers — My model was wrong ~25% of the time and defaulted toward the majority outcome unless corrected. Deployed carelessly, that bias hides behind the appearance of an "objective" number.

Where Human and Machine Judgment Diverge — The real question isn't "can AI replace doctors," it's what each one sees that the other structurally can't, and being honest about where the pattern ends and the person begins.



Feature selection is a product decision. Dropping operation_year wasn't about math, it was deciding what actually matters, the same judgment a PM makes prioritizing what data a product should collect.

Accuracy can hide the real story. v2 and v3 had similar accuracy, but v3 matched the real class distribution far better. A PM can't trust one number, you have to ask what mistakes the model is making.

Iteration is the job. v1 → v2 → v3, testing against the full dataset instead of the convenient sample, mirrors the discipline of shipping and interrogating an AI feature rather than trusting the first result.

The real risk is what the model's blind to. Watching it default toward the majority class taught me a PM's job with AI is surfacing that blind spot, not hiding it behind a confident output.