The Model Development and Evaluation Process

Matthew Engelhard

Today

Essentials of Sound Model Development and Evaluation

- Why out-of-sample evaluation (*i.e.,* a held-out test set) is critical
- How do we divide up the data?
- How is each portion is used?
- What can go wrong?

Common Variations in Practice

- Hyperparameter tuning
- Cross-validation

The Napoleon Dynamite Problem (NDP)

The New York Times Magazine

THE SCREENS ISSUE

If You Liked This, You're Sure to Love That

Give this article

By Clive Thompson

Nov. 21, 2008

THE "NAPOLEON DYNAMITE" problem is driving Len Bertoni crazy. Bertoni is a 51-year-old "semiretired" computer scientist who lives an hour outside Pittsburgh. In the spring of 2007, his sister-in-law e-mailed him an intriguing bit of news: Netflix, the Web-based DVD-rental company, was holding a contest to try to improve Cinematch, its "recommendation engine." The prize: $1 million.

Claim: I’ve solved the NDP.

- Randy took headshots of each member of the MMCi class of 2021
- Catherine asked them whether they liked Napoleon Dynamite
- I then studied the data

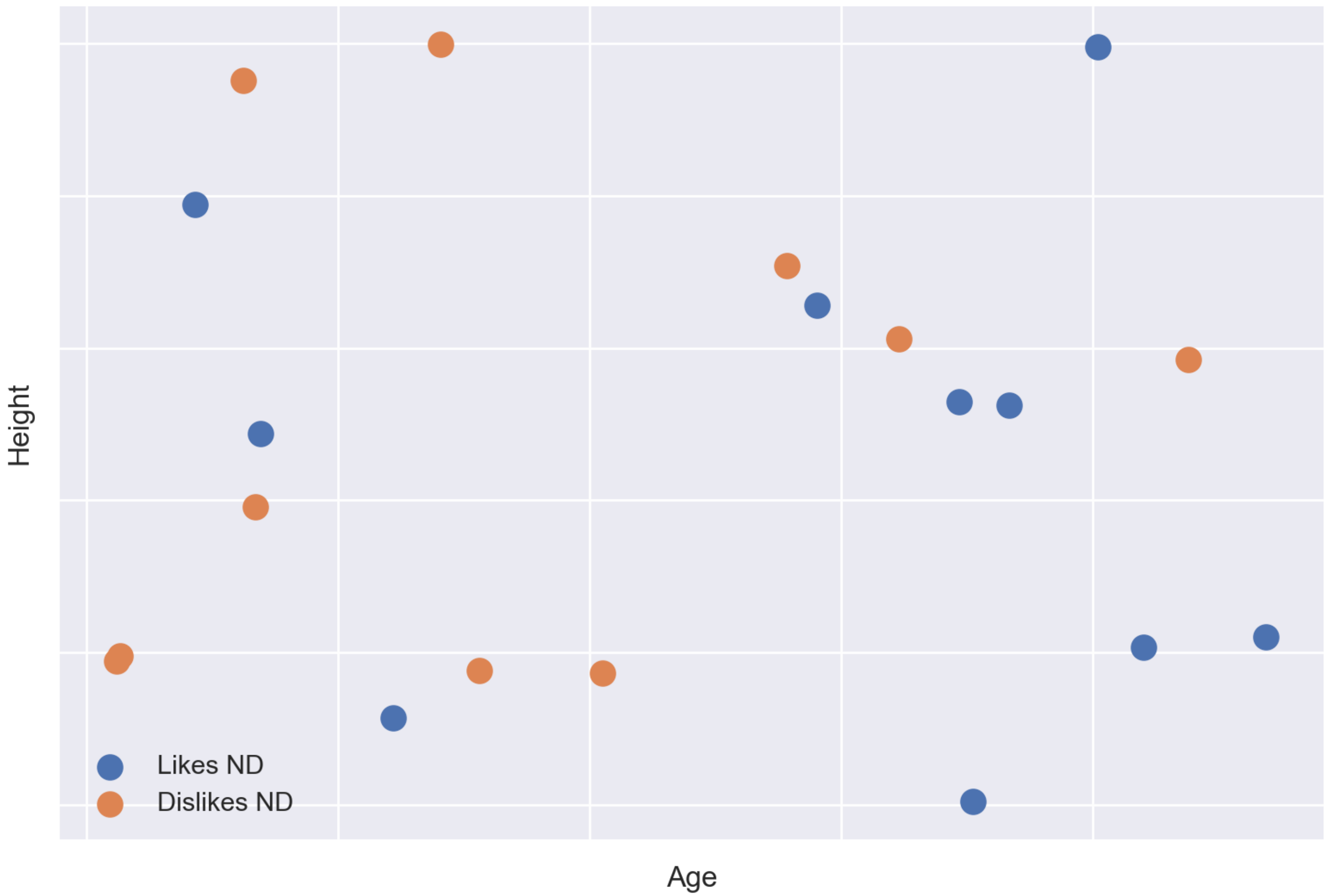
I can predict whether someone likes Napoleon Dynamite from a picture of their face with 100% accuracy, and I can prove it. Show me a picture of anyone in the class and I will predict whether they like Napoleon Dynamite.

Aren’t you impressed?

You got me. I memorized the dataset.

And I can do it again with this dataset.

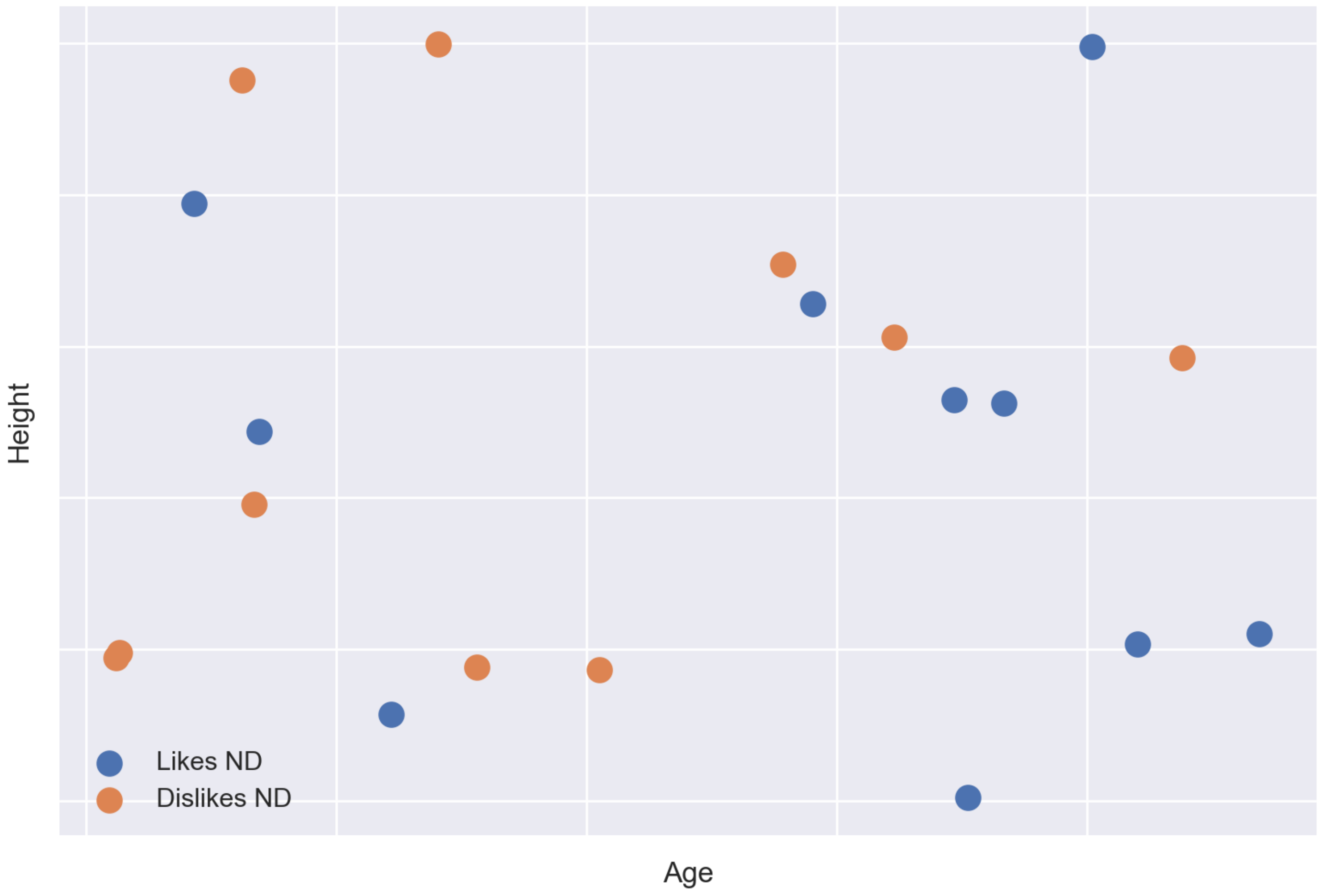
I can predict whether people like Napoleon Dynamite from their age and height.



You got me. I memorized the dataset.

And I can do it again with this dataset.

<u>Question</u>: Can logistic regression memorize this dataset (*i.e.*, make perfect predictions)?

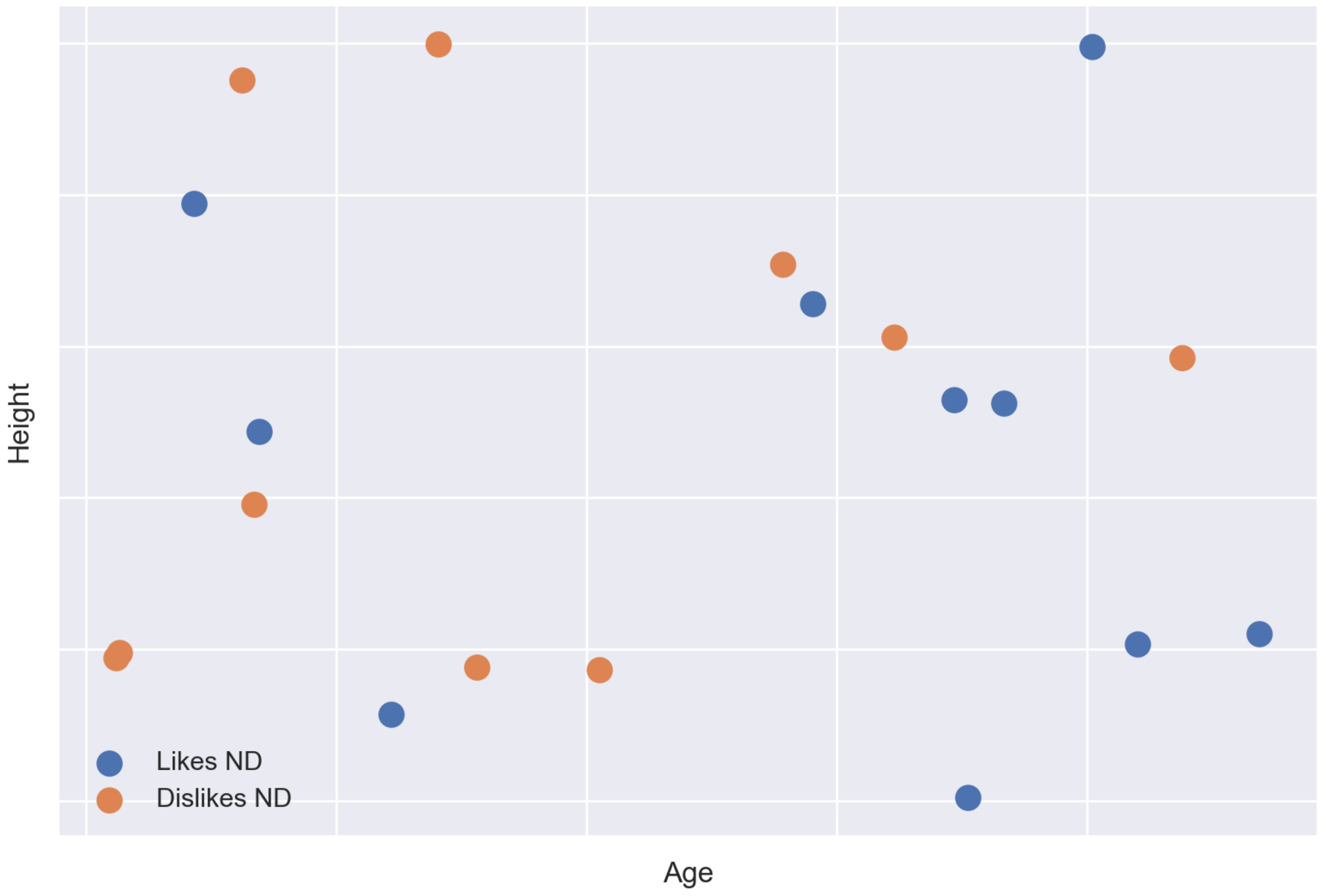


You got me. I memorized the dataset.

And I can do it again with this dataset.

<u>Question</u>: Can logistic regression memorize this dataset (*i.e.*, make perfect predictions)?

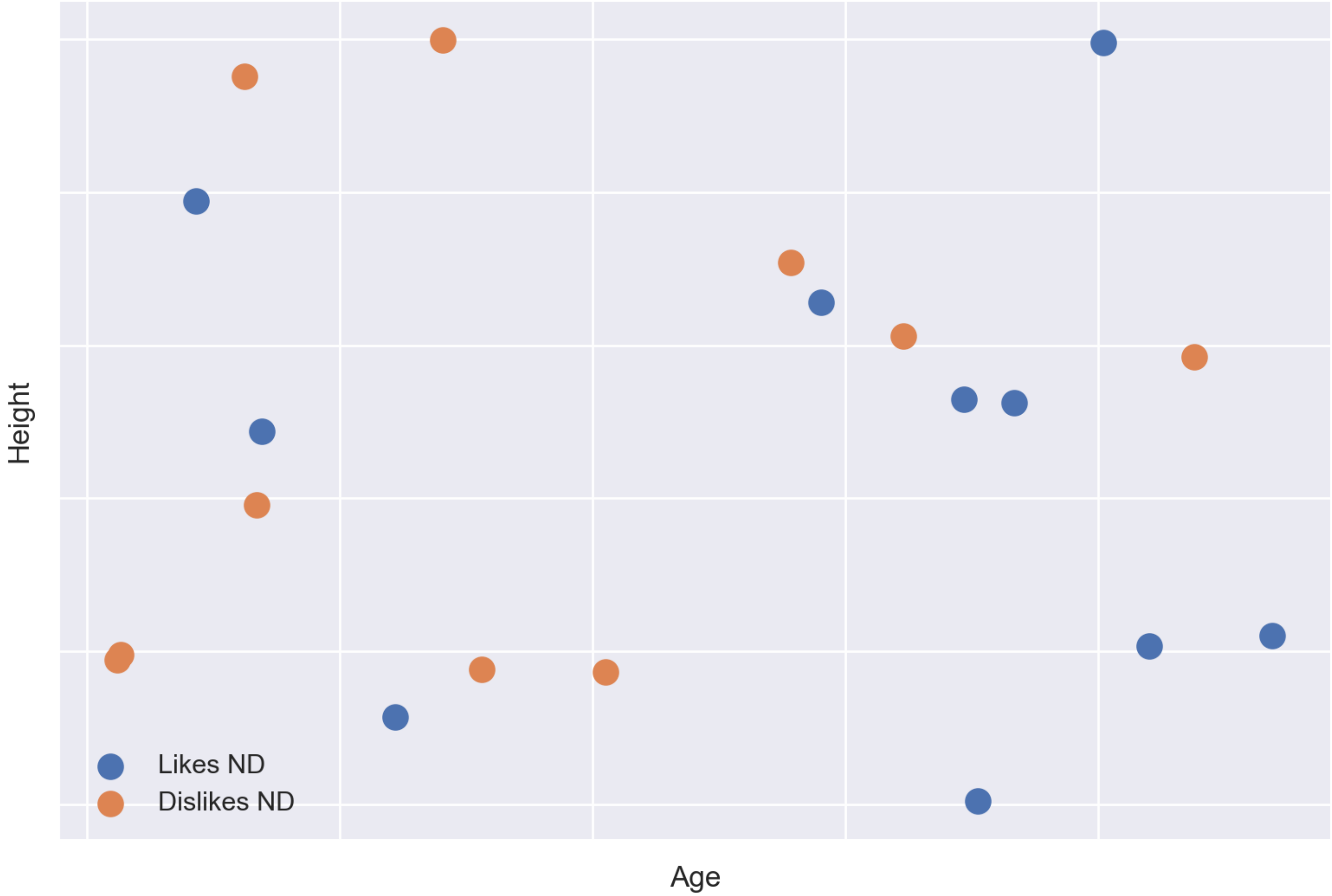
<u>Answer:</u> No. There is no line that perfectly separates the blue and orange points.



You got me. I memorized the dataset.

And I can do it again with this dataset.

<u>Question</u>: Can a neural network memorize this dataset (*i.e.*, make perfect predictions)?

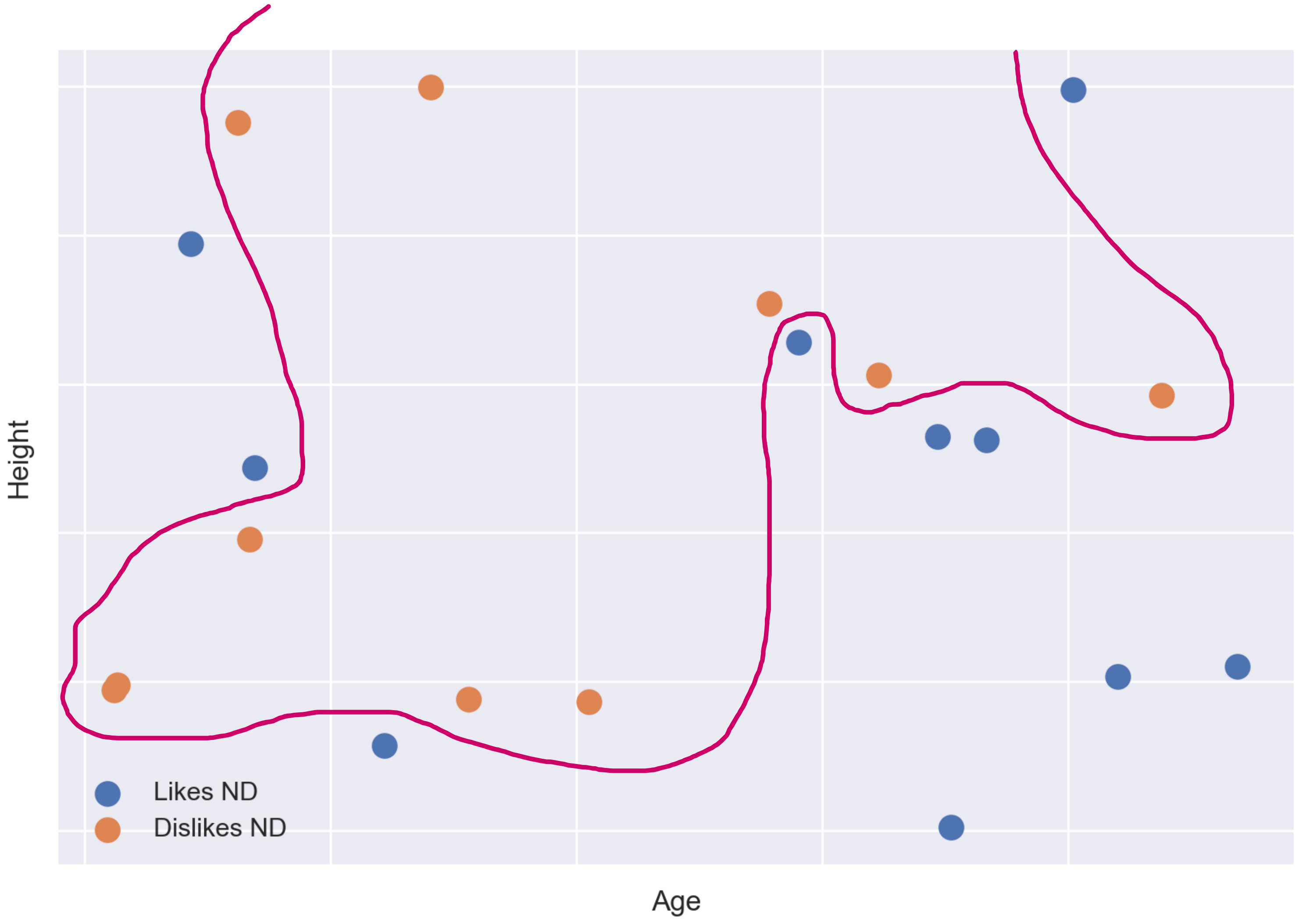


You got me. I memorized the dataset.

And I can do it again with this dataset.

Question: Can a neural network memorize this dataset (*i.e.*, make perfect predictions)?

Answer: Yes… like so -->

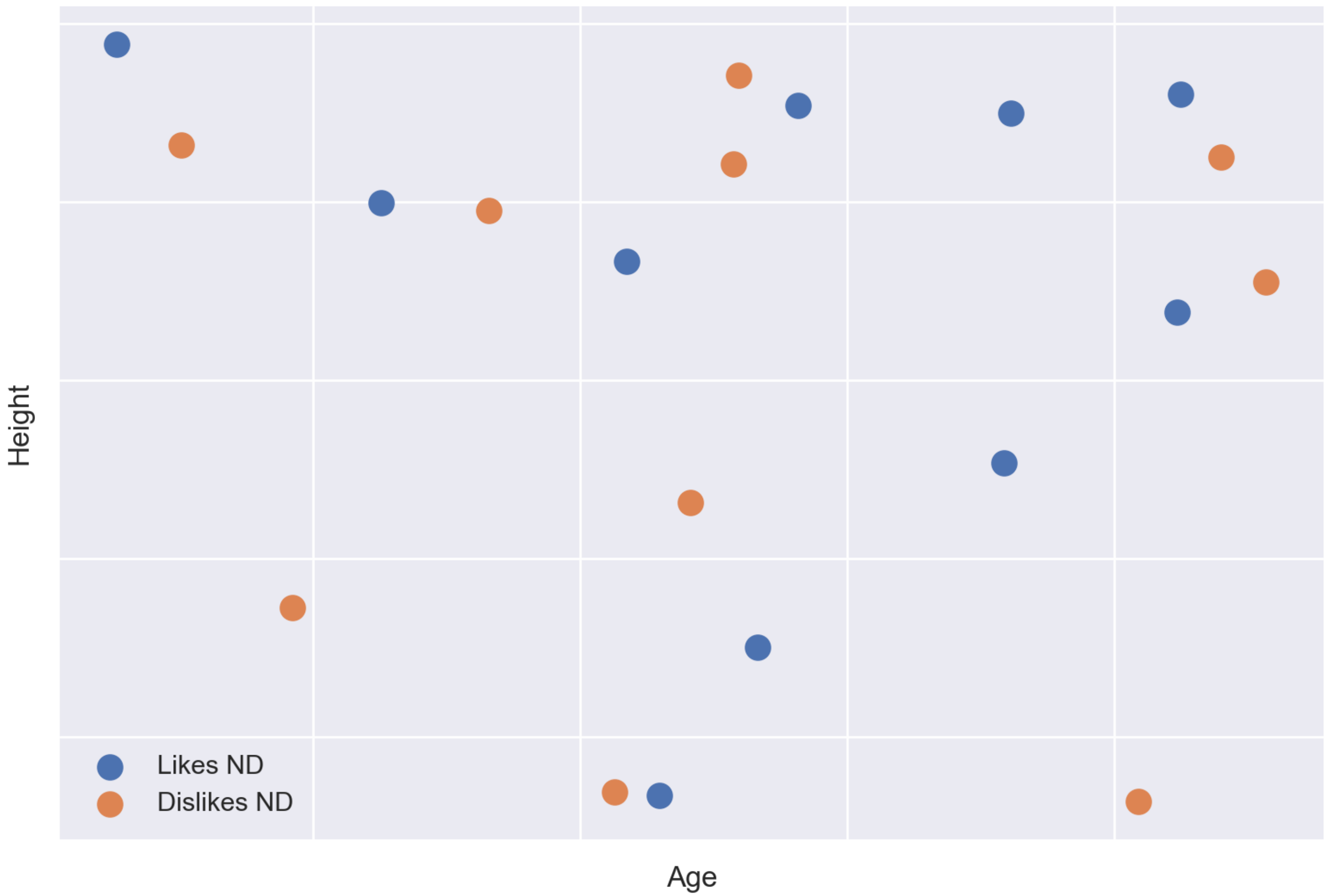


You got me. I memorized the dataset.

And I can do it again with this dataset.

Question: Can a neural network memorize this dataset (*i.e.*, make perfect predictions)?

Answer: Yes… like so -->

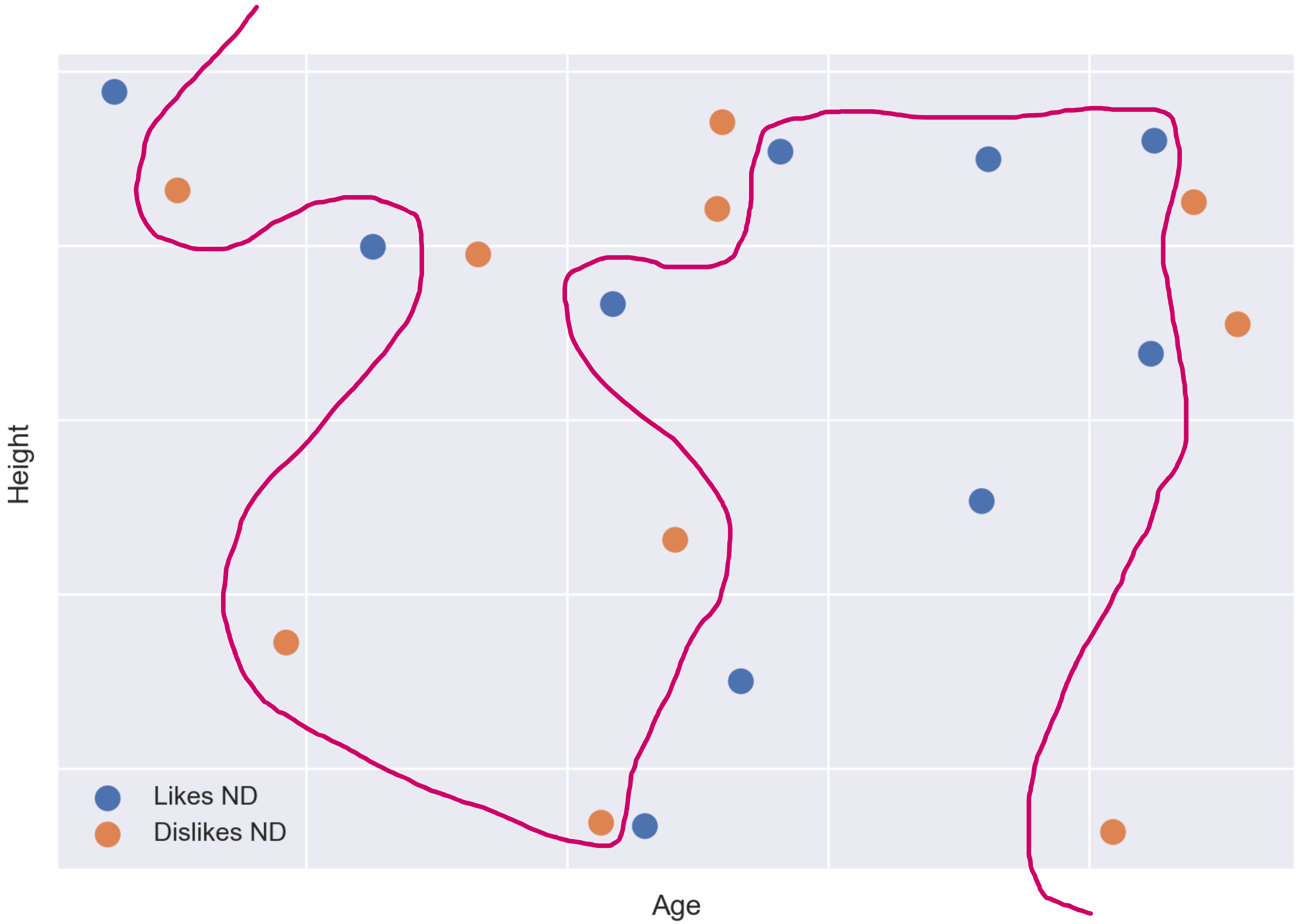


You got me. I memorized the dataset.

And I can do it again with this dataset.

Question: Can a neural network memorize this dataset (*i.e.*, make perfect predictions)?

Answer: Yes… like so -->



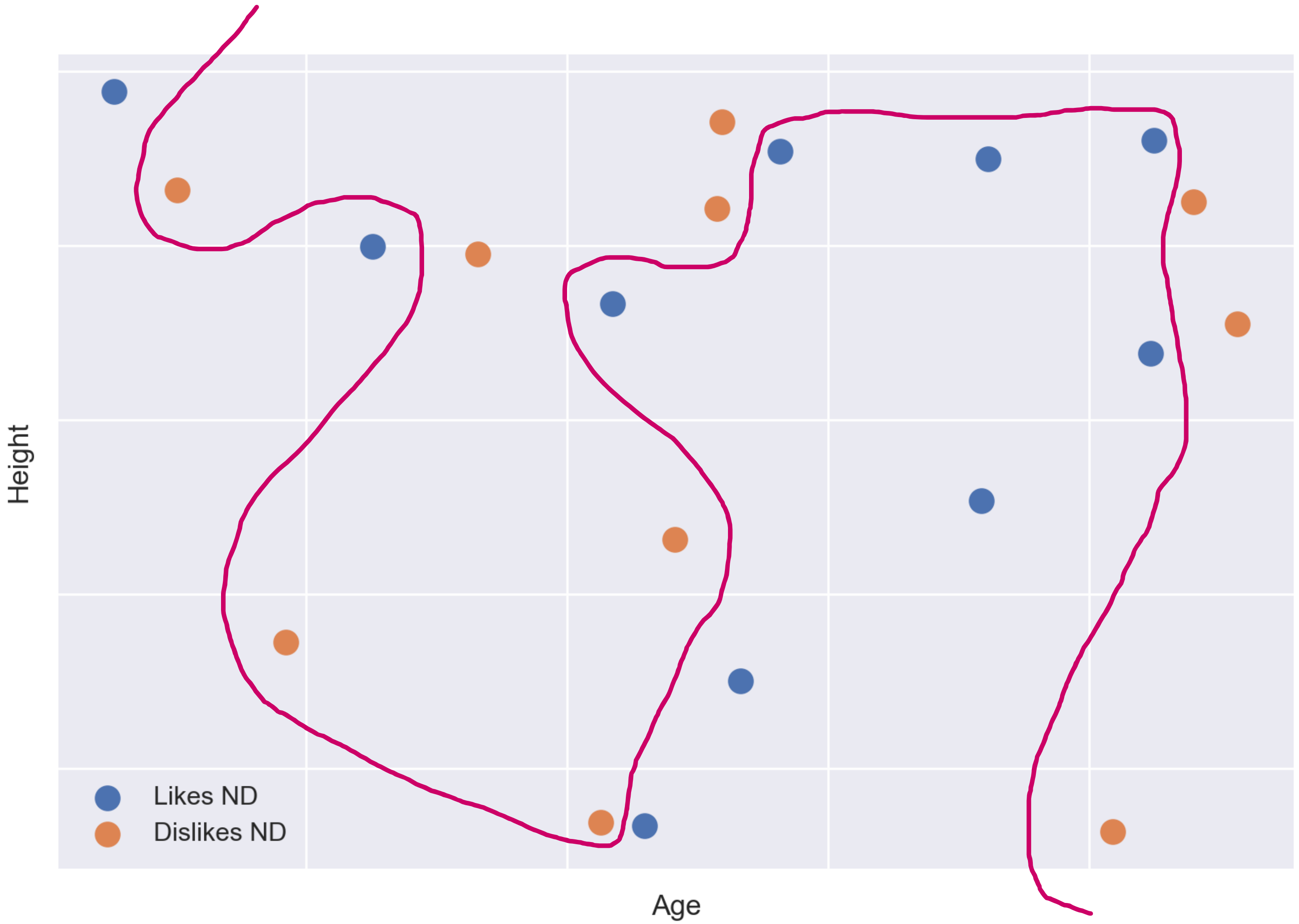
You got me. I memorized the dataset.

And I can do it again with this dataset.

<u>Question</u>: Can a neural network memorize this dataset (*i.e.*, make perfect predictions)?

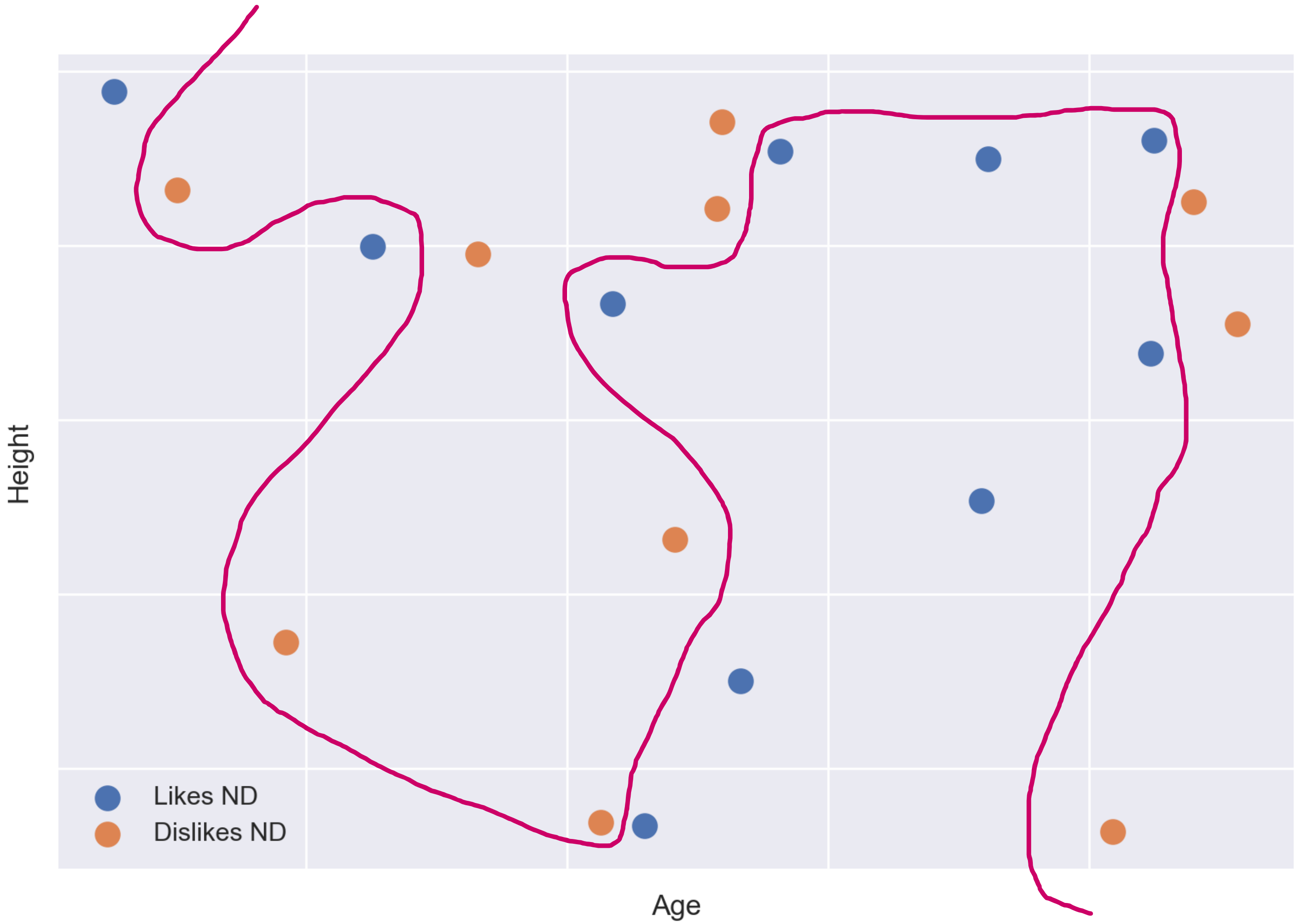
<u>Answer:</u> Yes… like so -->

What is this called?



You got me. I memorized the dataset.

If this were the true relationship between age, height, and Napoleon Dynamite, then this boundary would still work when we gather data from a new sample of participants.

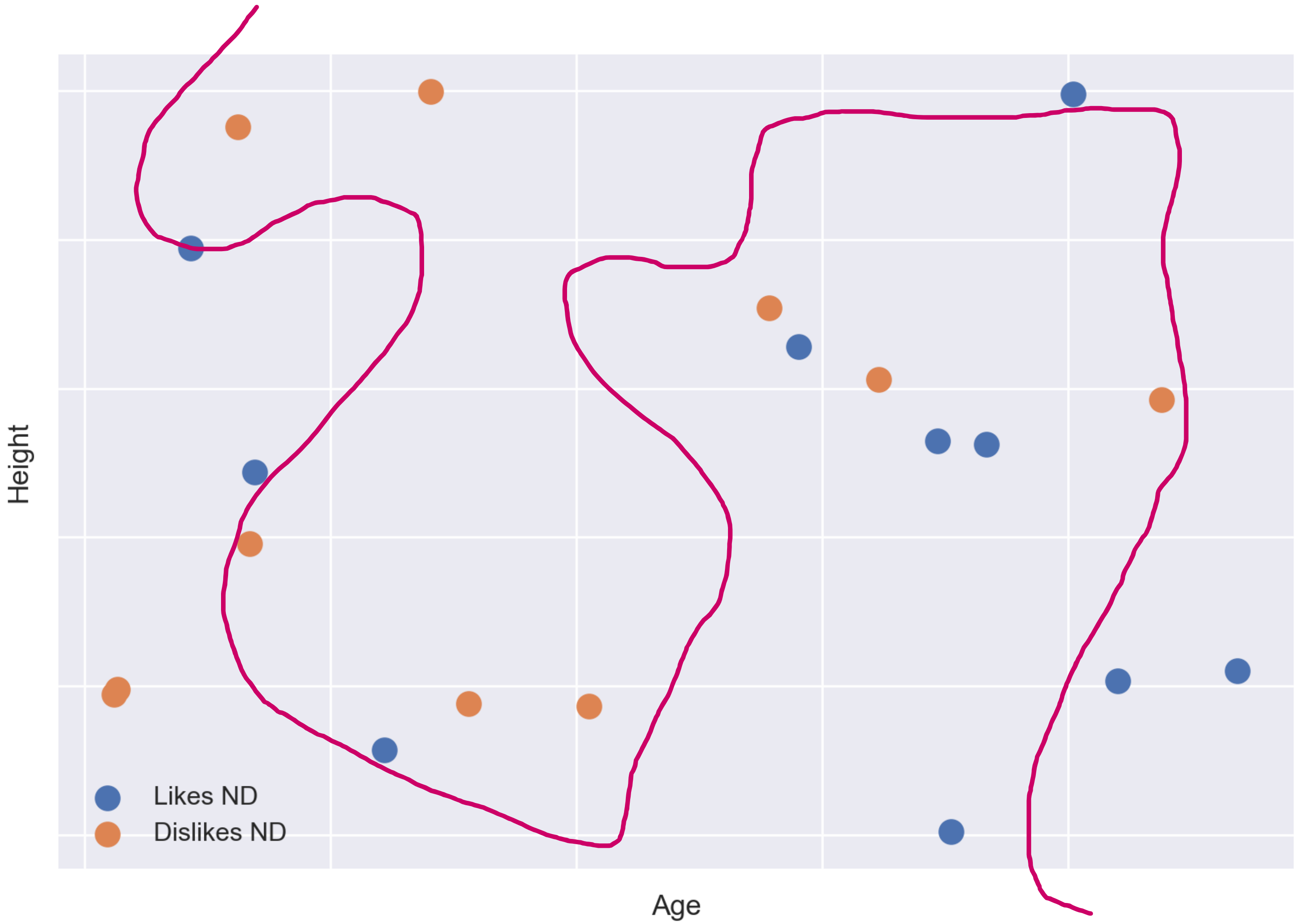


You got me. I memorized the dataset.

If this were the true relationship between age, height, and Napoleon Dynamite, then this boundary would still work when we gather data from a new sample of participants.

But it doesn't.

In this case, there's no relationship – it's just random noise. But the NN can still make perfect predictions on the training data.



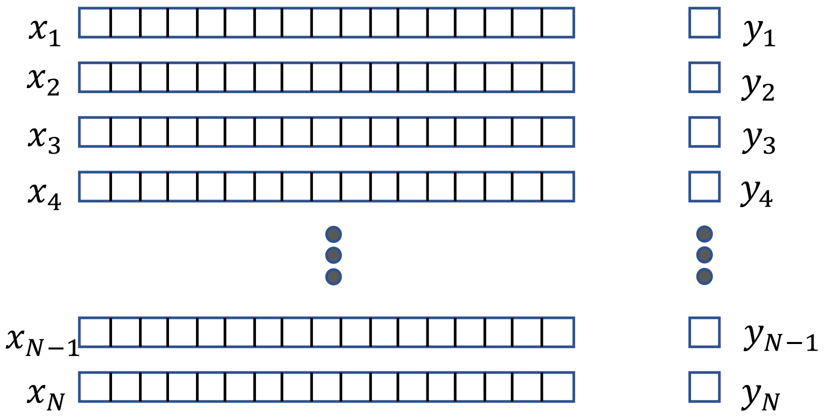
Key Takeaways

- **We must evaluate our models using data <u>they haven't seen</u> before.**
  - The test set should not be used to train
  - The test set should not be used *in any other way* prior to the evaluation

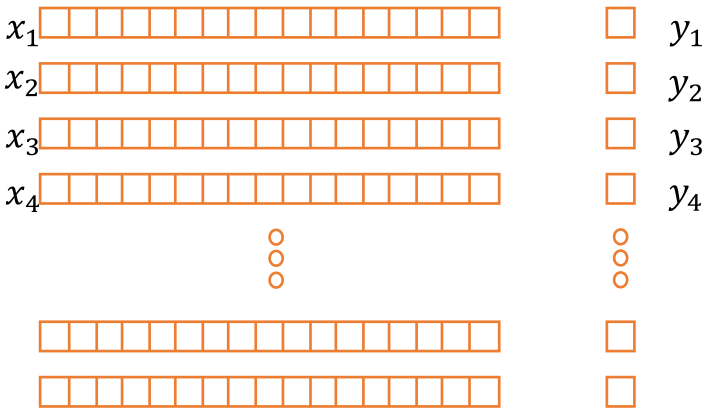
- For complex models – say, an MLP with many layers, or logistic regression with a large number of predictors:
  - Performance on the training set can dramatically overestimate performance on the test set
  - So, evaluating on a held-out test set is critical

- For simple models – say, logistic regression with only a few predictors:
  - Performance on the training set should be close to performance on the test set
  - Nevertheless, evaluating on a held-out test set is good practice

How do we divide up the data? (easy version)



Development Set:
-> used to build the model



Test Set:
-> used to estimate real-world performance

Suppose we receive a new dataset

- Read the data (Excel, SQL)
- Assign to the development set or test set: shuffle, then split
- Divide predictors (*x*) from outcomes (*y*) and unwanted columns (often identifiers)
- Preprocessing
  - Normalize numeric variables
  - Cell code categorical variables
  - Remember, we don't want to use any information from the test set prior to training. This can be a bit tricky to implement.

MORTALITY PREDICTION WORKSHEET				
COVARIATES				OUTCOME
patient	age	female	temp	mortality
0	30.5	0	105.0	1
1	74.0	1	96.7	0
2	27.4	0	96.1	0
3	0.1	1	98.5	0
4	0.7	1	96.5	0
5	49.9	1	97.1	0
6	72.9	1	100.1	1
7	29.1	1	99.6	0
8	83.5	1	100.6	1
9	82.3	1	95.2	1
10	23.7	0	99.4	1
11	12.9	0	96.6	0
12	53.9	1	100.3	0
13	18.8	0	98.6	0
14	51.8	0	98.5	0
15	3.3	0	94.6	0
16	69.7	0	99.1	0
17	60.4	1	104.2	1
18	73.6	1	99.1	1
19	53.3	1	99.1	0

Variations on random assignment

- Why is it a good idea to assign the data at random (*i.e.*, shuffle first)?

- Are there any exceptions or alternatives?

  - Our evaluation should match the claims we wish to make about performance

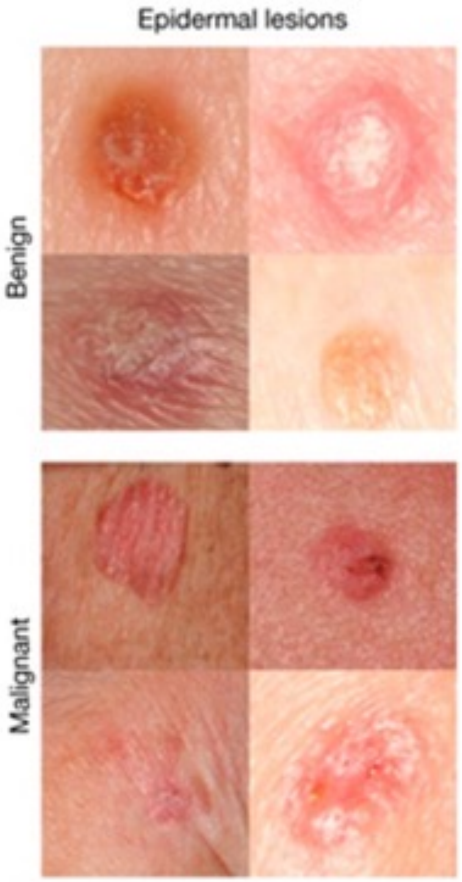
  - We might want to make a specific claim, such as:
    - A model trained on years 2000-2010 generalizes to years 2010-2020
    - A model trained at DUHS generalizes to the UNC Health System

  - We might have repeated measures data
    - Multiple (*x*, *y*) pairs from the same patient *<-- this is a very common source of errors*
    - We typically handle this by assigning repeated measures (*i.e.,* all data for a given patient) to the same fold (*i.e.*, development set, test set)

Claims about performance must match your evaluation.

What's wrong here?

- Dataset: pictures of skin lesions along with label (benign/malignant)
  - 2,000 images of 600 lesions from 400 participants
- Goal: predict whether lesions are benign or malignant
- Evaluation Strategy: Divide the data (image/label pairs) at random between a development set (80%) and a test set (20%)



Claims about performance must match your evaluation.

What's wrong here?

- Dataset: wearables data from 20 patients with arrhythmia and 20 controls
  - 10 sessions during arrhythmia for each patient with arrhythmia
  - 10 sessions at random times for each control
- Goal: predict arrythmia
- Evaluation Strategy: Divide the data (image/label pairs) at random between a development set (70%) and a test set (30%)

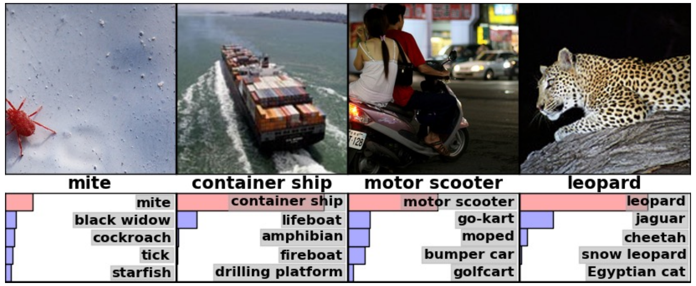
What goes wrong?

- Subtle mistakes like the previous examples are very common.
- Obvious mistakes like evaluating on the training data are also very common.

- <u>Data Leakage</u>: the use of information in the model training process **which would not be expected to be available at prediction time**

- *Think about whether the information you are using would be available at the time of prediction.*

Defenses against improper evaluation



<u>Publicly available data and code:</u>
Facilitates reproducibility

<u>Challenge format</u>:
Test and development sets are fixed by the organizers. Often, test sets aren't even available; entrants submit their model rather than presenting their results.

<u>In healthcare</u>:
There are some good publicly available datasets (MIMIC, *All of Us*, etc.), but we're still way behind the curve.

There's more to model development than training…

What if I want to choose which kind of model is best (*e.g.*, logistic regression, MLP)?

-> this is *model selection*

What if I want to refine or modify my model or the training process in some way (*e.g.*, adjust the number of hidden layers)?

-> this is *hyperparameter tuning*

Definition: a *hyperparameter* is a value or setting that is fixed before training that influences the training process

Parameters versus Hyperparameters

<u>Parameters</u>

- Lengths of the seams

<u>Hyperparameters</u>

- Which seams to adjust
- What ruler to use
- How fast / careless versus careful

Parameters versus Hyperparameters

<u>Parameters</u>

- Lengths of the seams
- Model coefficients (*i.e.,* numeric values in the equation)

<u>Hyperparameters</u>

- Which seams to adjust
- What ruler to use
- How fast / careless versus careful
- Model architecture (*i.e.*, the form of the equation)
- Training algorithm
- Regularization (*more next time)

How do we tune hyperparameters?

- <u>Proposal 1</u>: Figure out which hyperparameters perform best on the training set
- <u>Proposal 2</u>: Figure out which hyperparameters perform best on the test set

- Are either of these a good idea? Why or why not?

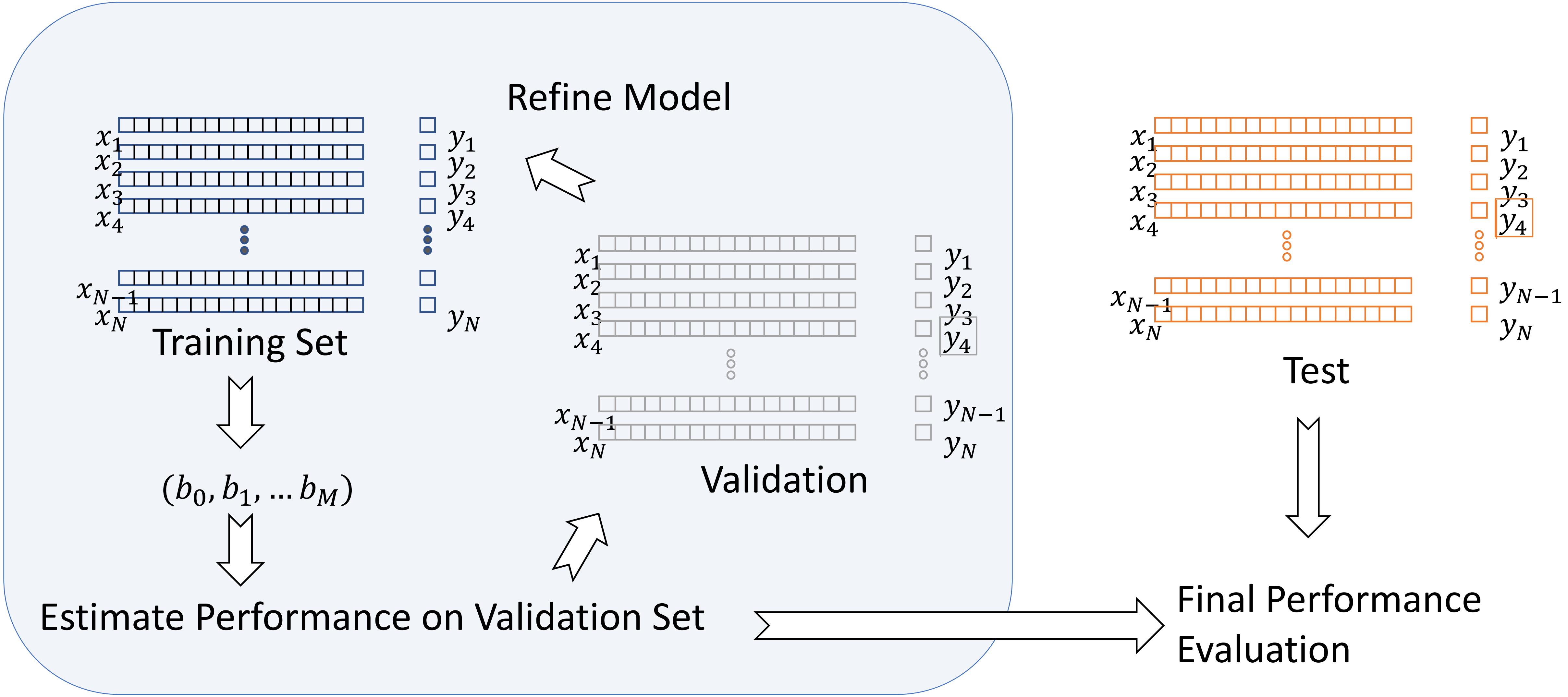
How do we tune hyperparameters?

- Proposal 1: Figure out which hyperparameters perform best on the training set
- Proposal 2: Figure out which hyperparameters perform best on the test set

- Are either of these a good idea? Why or why not?

-> Hyperparameters require their own *tuning* set.

How do we divide up the data?



Cross-Validation

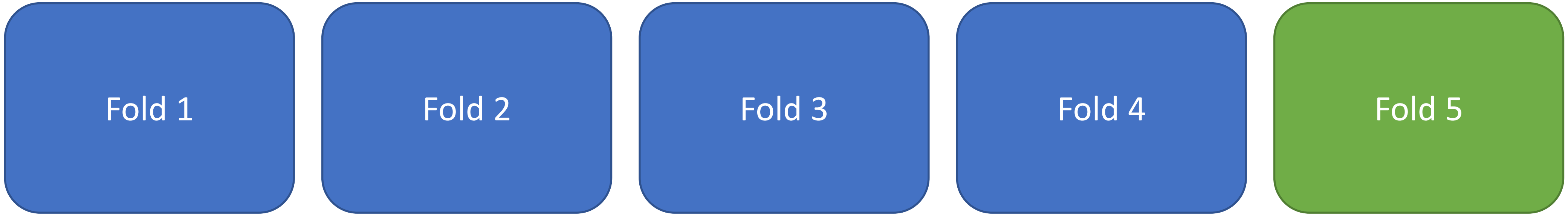
(time permitting)

Traditional “split” for development + eval

Training set | Validation set | Test set

Train parameters | Tune hyperpms | Evaluate performance

Cross validation: rotate the test set



Round 1: Train on 1-4, test on 5
Round 2: Train on all but 4, test on 4
Round 3: Train on all but 3, test on 3
Round 4: Train on all but 2, test on 2
Round 5: Train on 2-5, test on 1

Why would we do this?

- More data for the evaluation
- Better estimate of out-of-sample performance

What happened to the validation set?

- Use “flat” cross-validation (below) versus “nested” cross-validation
- Both give unbiased estimates and flat is easier

Fold 1	Fold 2	Fold 3	Fold 4	Fold 5

Round 1: Train on 1-3, validate on 4, test on 5
Round 2: Train on 2-4, validate on 5, test on 1
Round 3: Train on 3-5, validate on 1, test on 2
Round 4: Train on 4, 5, and 1; validate on 2, test on 3
Round 5: Train on 5, 1, and 2; validate on 3, test on 4