Model diagnostics

ML workflow

Problem scoping

Experimentation

- Choose architecture (data, model)
- Train model
- Evaluate model

Deployment

Debugging your ML model

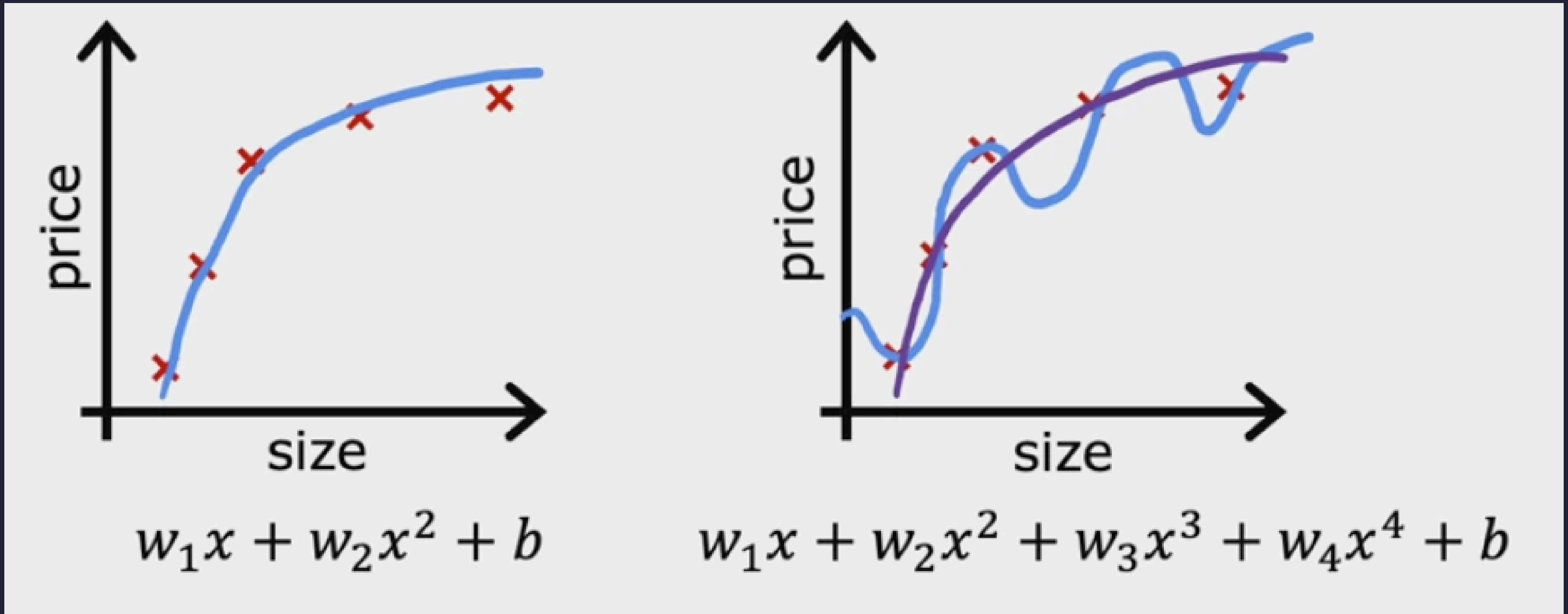
$$J = \frac{1}{m}\sum_{i=1}^{m} L(y^{(i)}, \hat{y}^{(i)}) + \lambda \sum_{j=1}^{n} w_j^2$$

Got large prediction errors. What do you do?

- Get more training examples
- Try smaller sets of features
- Try getting additional features
- Try adding polynomial features (e.g., $x_1^2, x_1 x_2, x_2^2$)
- Try decreasing λ
- Try increasing λ

Model evaluation/selection

Evaluating your model: plotting



Wiggly shape ➡️ overfit ➡️ poor generalization

Plotting for evaluation is not scalable for high-dimensional data

Evaluating your model: Training/test sets

Split the data into ***two*** sets:

- **Training set**: the data on which the model is trained
- **Test set**: the data reserved for evaluation. **New to the model**

e.g., 80/20 split: 80% training, 20% test

Computing training and test errors: linear regression example

Fit the model parameters by minimizing the cost function:

$$J = \underbrace{\frac{1}{m_{train}} \sum_{i=1}^{m_{train}} (y - f(x))^2}_{\mathrm{MSE}_{train}} + \lambda \sum_{j=1}^{n} w_j^2$$

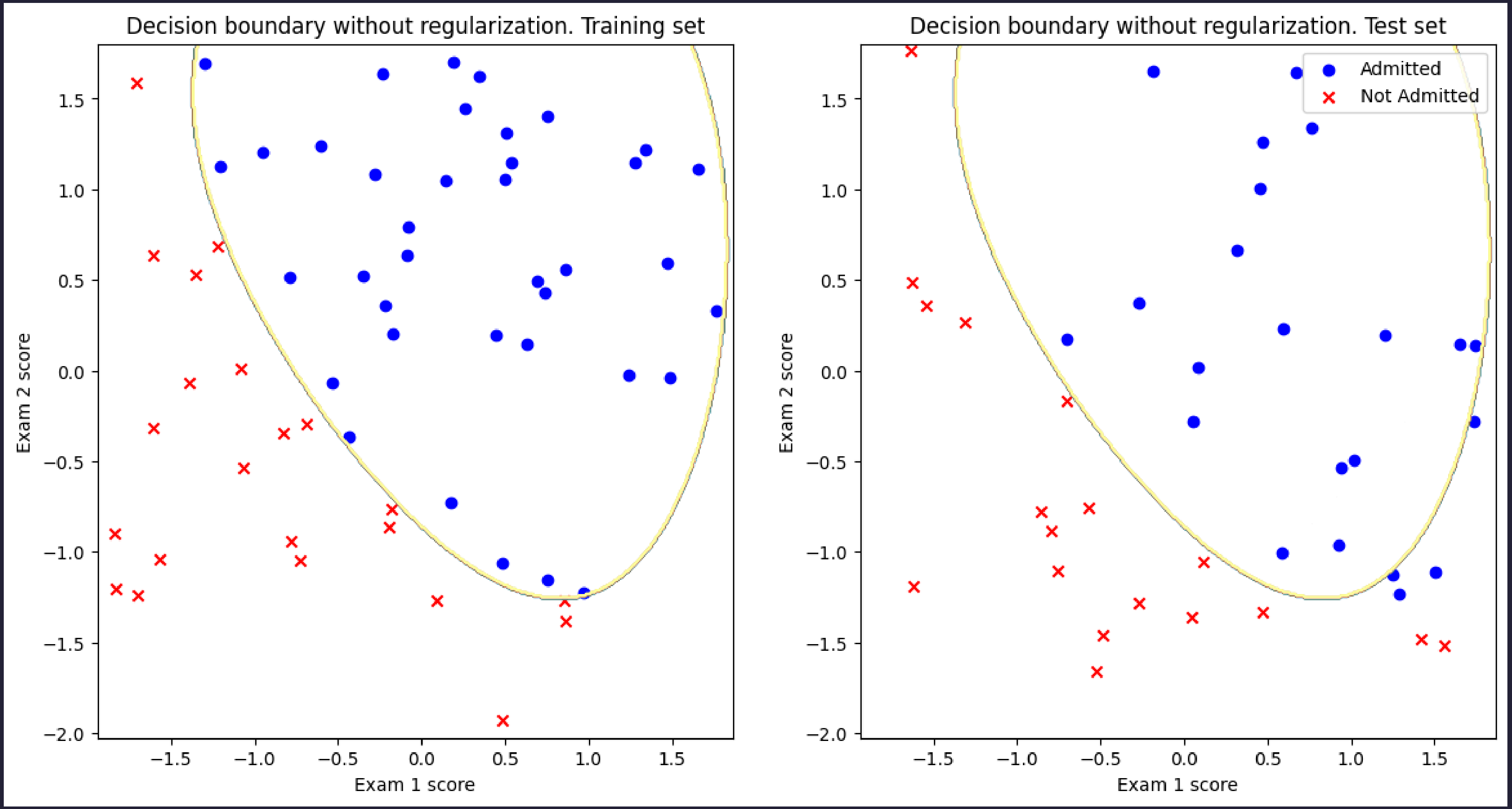
Compute training error:

$$J_{train} = \mathrm{MSE}_{train}$$

Compute test error:

$$J_{test} = \mathrm{MSE}_{test}$$

Overfitting: exam dataset example



J_{train} is low, J_{test} is high

Model selection: Choosing model based on test error

d: degree of polynomial

$d = 1, f(x) = w_1x + b$
$d = 2, f(x) = w_1x + w_2x^2 + b$
$d = 3, f(x) = w_1x + w_2x^2 + w_3x^3 + b$
...
$d = 10, f(x) = w_1x + w_2x^2 + \ldots + w_{10}x^{10} + b$

Calculate J_{test} for each model (i.e., each d)

Choose the model with the lowest J_{test}

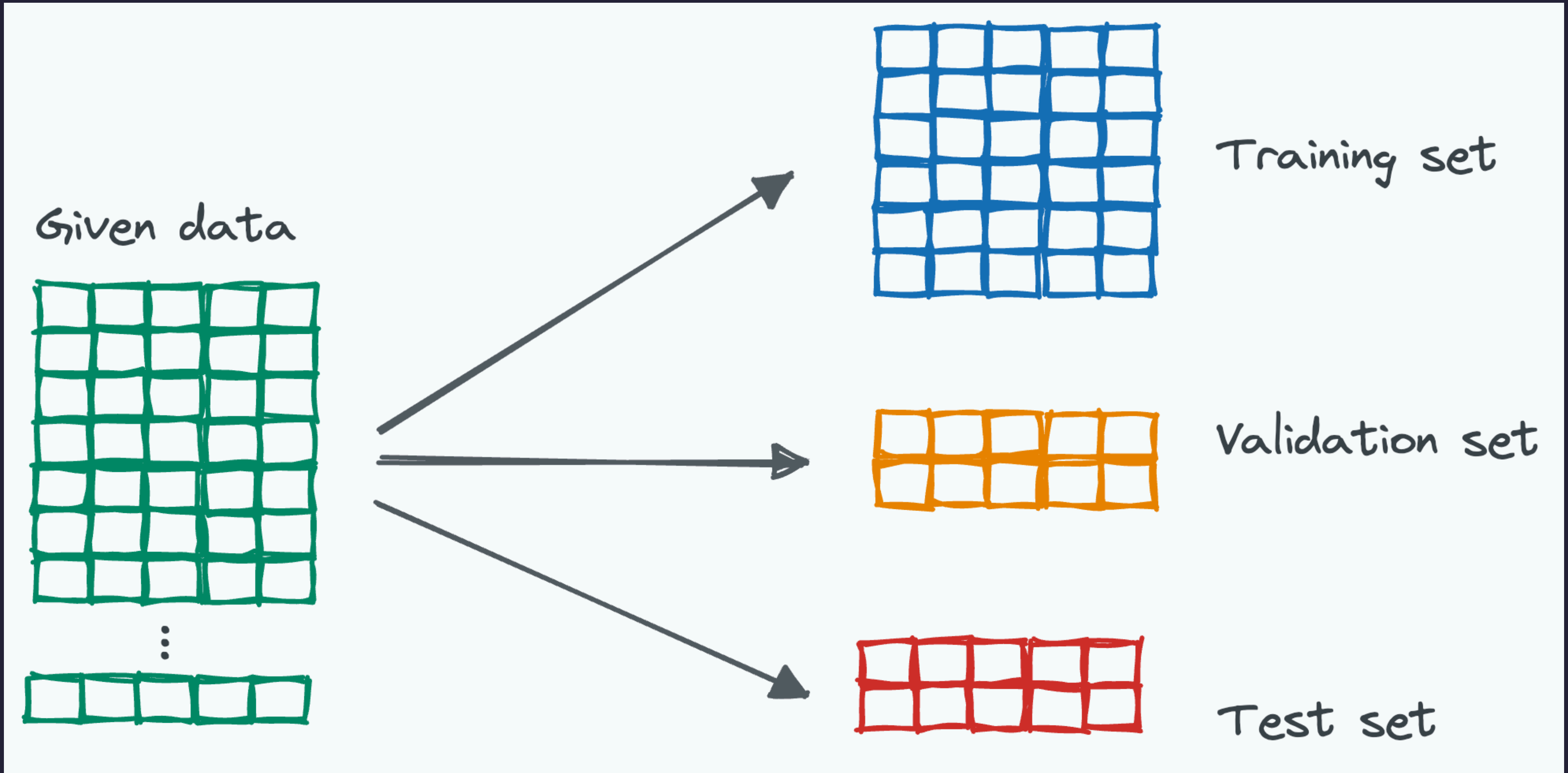
> J_{test} is likely to be an optimistic estimate of generalization error because an extra parameter d was chosen using the test set

Training/validation/test sets

Split the data into ***three*** sets:

- **Training set**: the data on which the model is trained
- **Test set**: the data reserved for evaluation. New to the model
- **(Cross) Validation set**: reserve another set of data just for cross-validating across different models

e.g., 60/20/20 split: 60% training, 20% validation, 20% test



Training set ➡️ fit the model parameters

Validation set ➡️ choose among different models

Test set ➡️ evaluate the performance of the final model

Computing training, validation, and test errors: linear regression example

Fit the model parameters by minimizing the cost function:

$$J = \underbrace{\frac{1}{m_{train}} \sum_{i=1}^{m_{train}} (y - f(x))^2}_{\text{MSE}_{train}} + \lambda \sum_{j=1}^{n} w_j^2$$

Training error:

$$J_{train} = \text{MSE}_{train}$$

(Cross) validation error (CV) for model selection:

$$J_{cv} = \text{MSE}_{cv}$$

Test error for model evaluation:

$$J_{test} = \text{MSE}_{test}$$

Model selection: Choosing model based on validation error

d: degree of polynomial

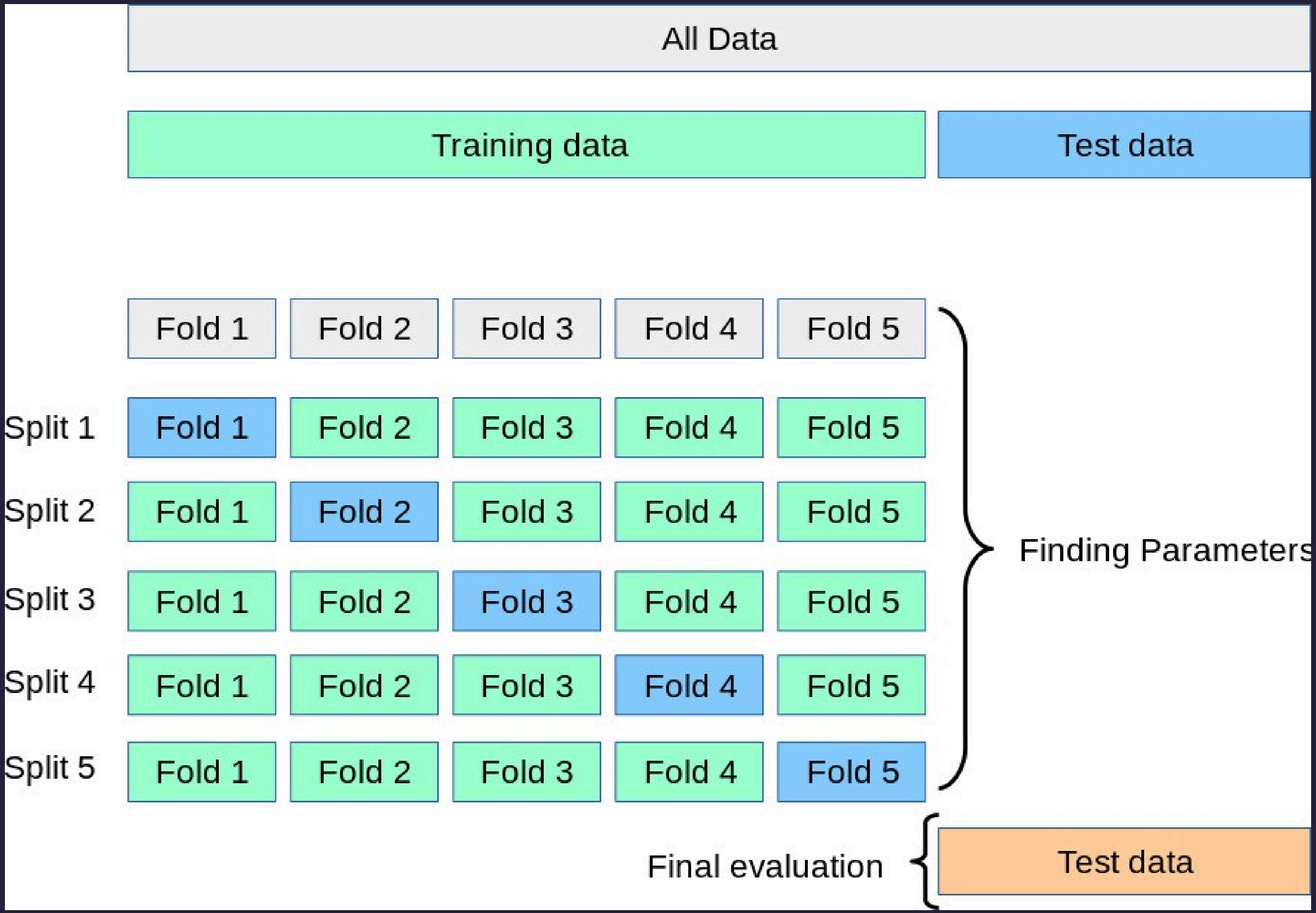
$d = 1, f(x) = w_1x + b$
$d = 2, f(x) = w_1x + w_2x^2 + b$
$d = 3, f(x) = w_1x + w_2x^2 + w_3x^3 + b$
...
$d = 10, f(x) = w_1x + w_2x^2 + \ldots + w_{10}x^{10} + b$

Calculate J_{cv} for each model (i.e., each d)

Choose the model with the lowest J_{cv}

Evaluate the final model using J_{test}

More efficient approach: k-fold cross-validation



k-fold cross validation

1. Split the data into k folds
2. Train the model on k-1 folds and validate on the remaining fold (compute validation error). Repeat k times, each time choosing a different fold as the validation set
3. Compute the average validation error (J_{cv})
4. Choose the configuration with the lowest J_{cv}
5. Retrain the model with the best configuration on the entire training dataset (m_{train})
6. Evaluate the final model using the test set (J_{test})

Hyperparameter tuning: choosing the optimal configuration based on J_{cv}

degree of polynomial (d = 1, 2, 3)

regularization (λ = 0.01, 0.1, 1)

learning rate

initial weights

number of layers in a neural network

number of hidden units in a neural network

...

Grid Search for hyperparameter tuning

Search a grid of hyperparameters

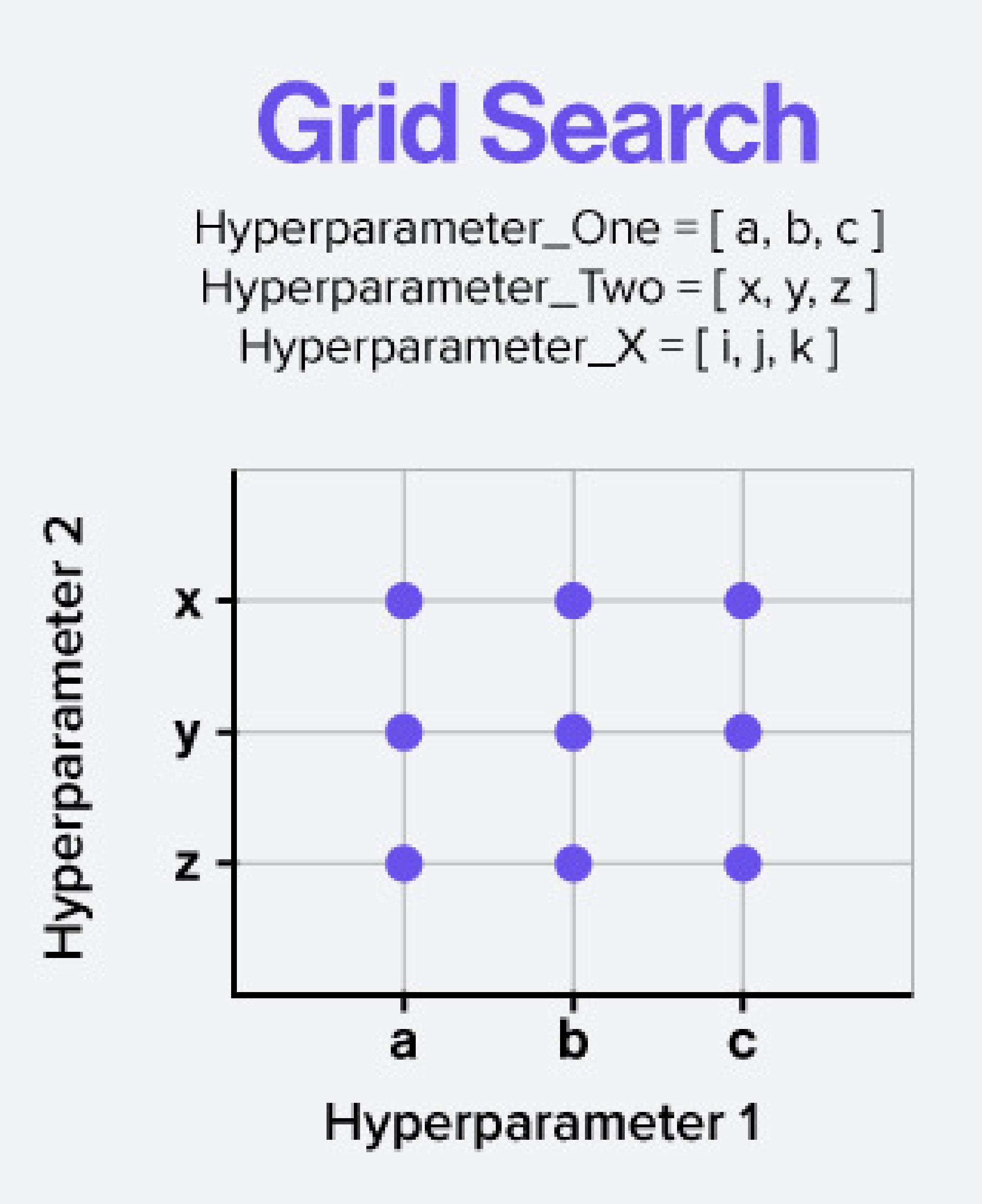
- all possible combinations

Lowest J_{cv} ➡️ best hyperparameters

Pros: Systematic search

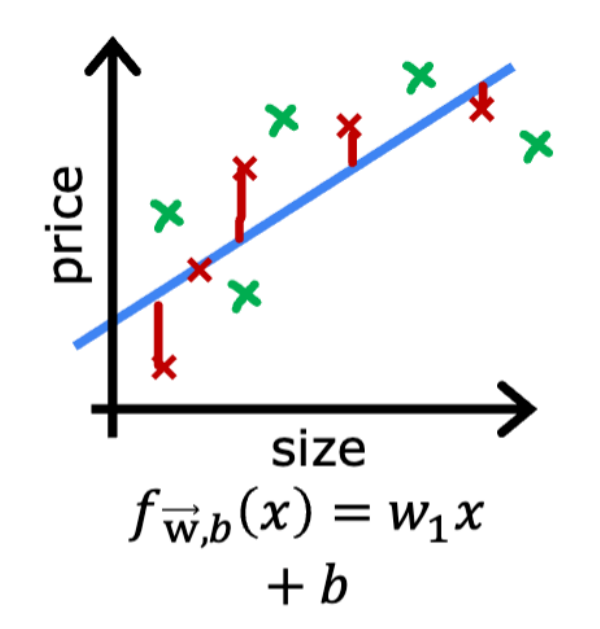
Cons: Computationally expensive

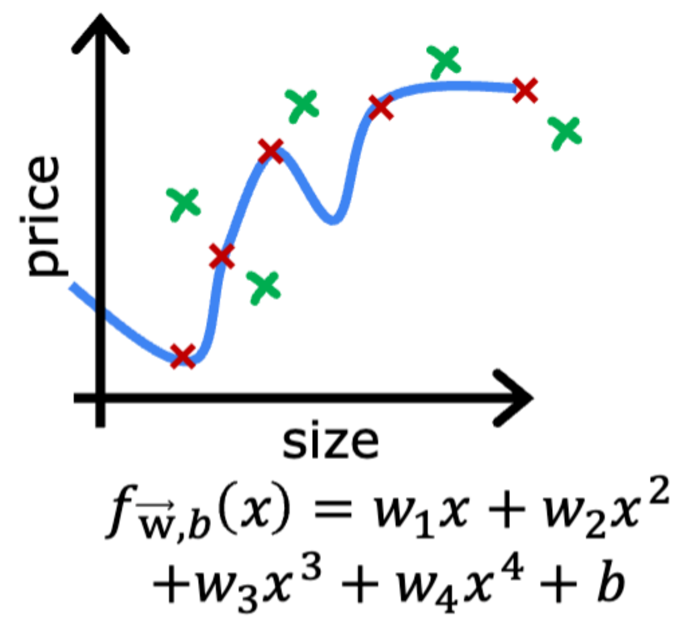
- More efficient search algorithms: Random search, Bayesian optimization, etc.



Bias and variance

Bias and variance





High bias (underfit)

J_{train} is high (red)
J_{cv} is high (green)
$J_{cv} \approx J_{train}$

High variance (overfit)

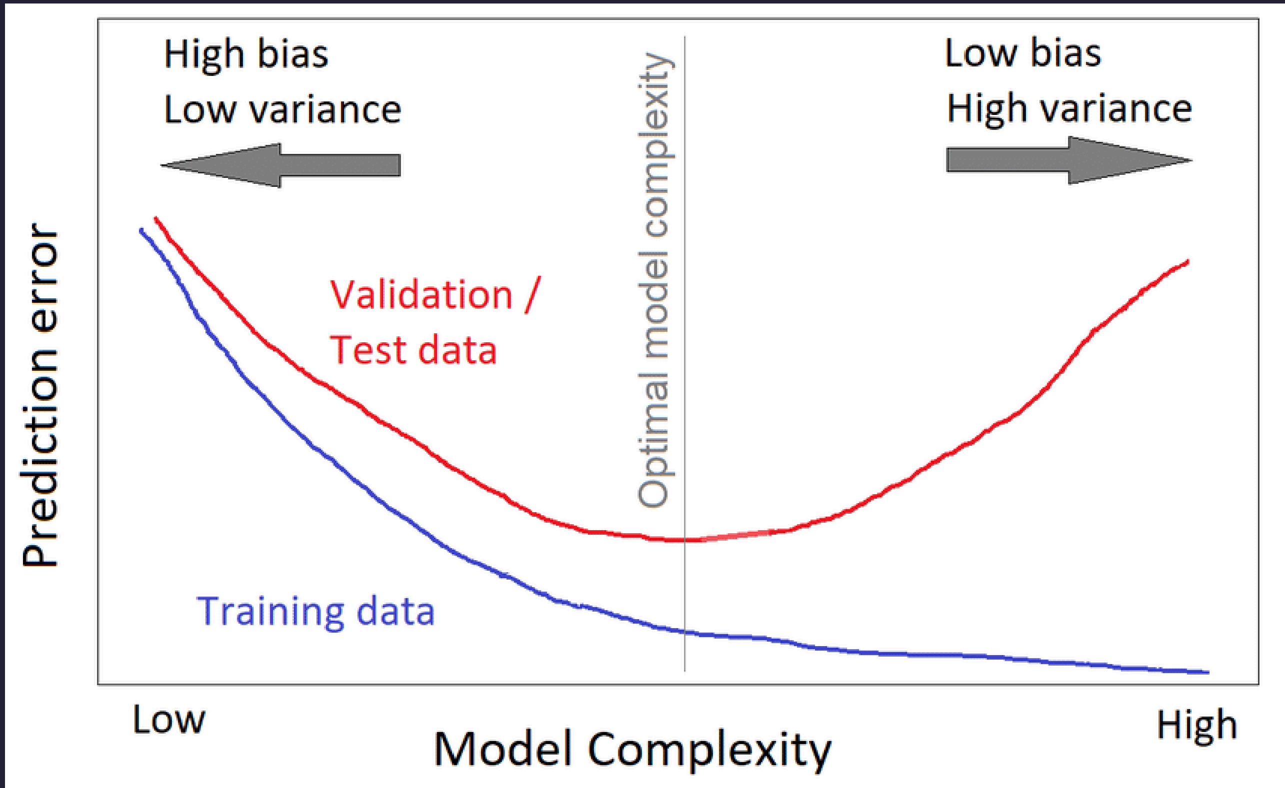
J_{train} is low (red)
J_{cv} is high (green)
$J_{cv} > J_{train}$

Bias-variance tradeoff

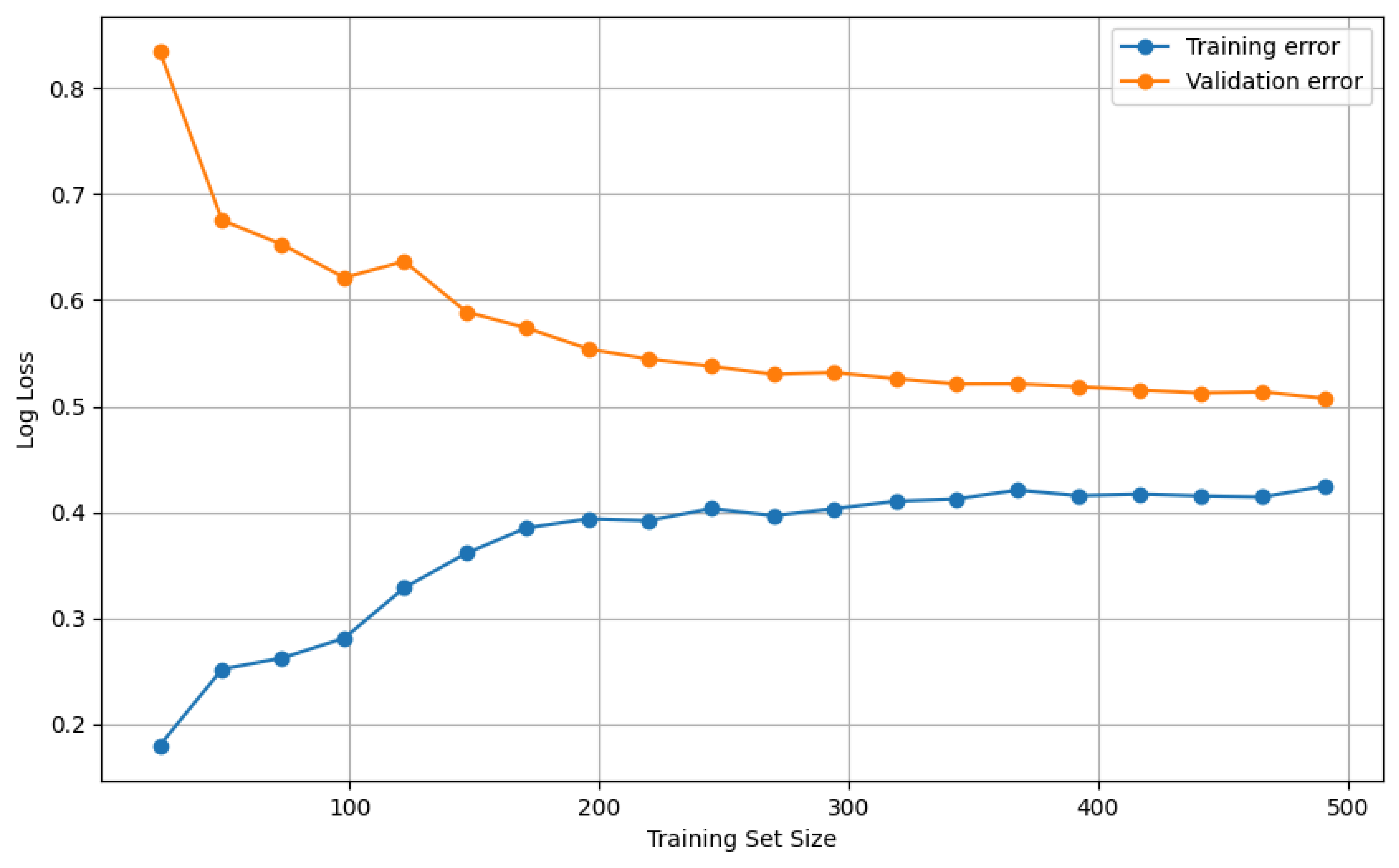
The balance between underfit and overfit

Optimal model complexity

- polynomial degree
- regularization parameter
- number of layers in a neural network
- ...



Learning curves: error vs. training set size

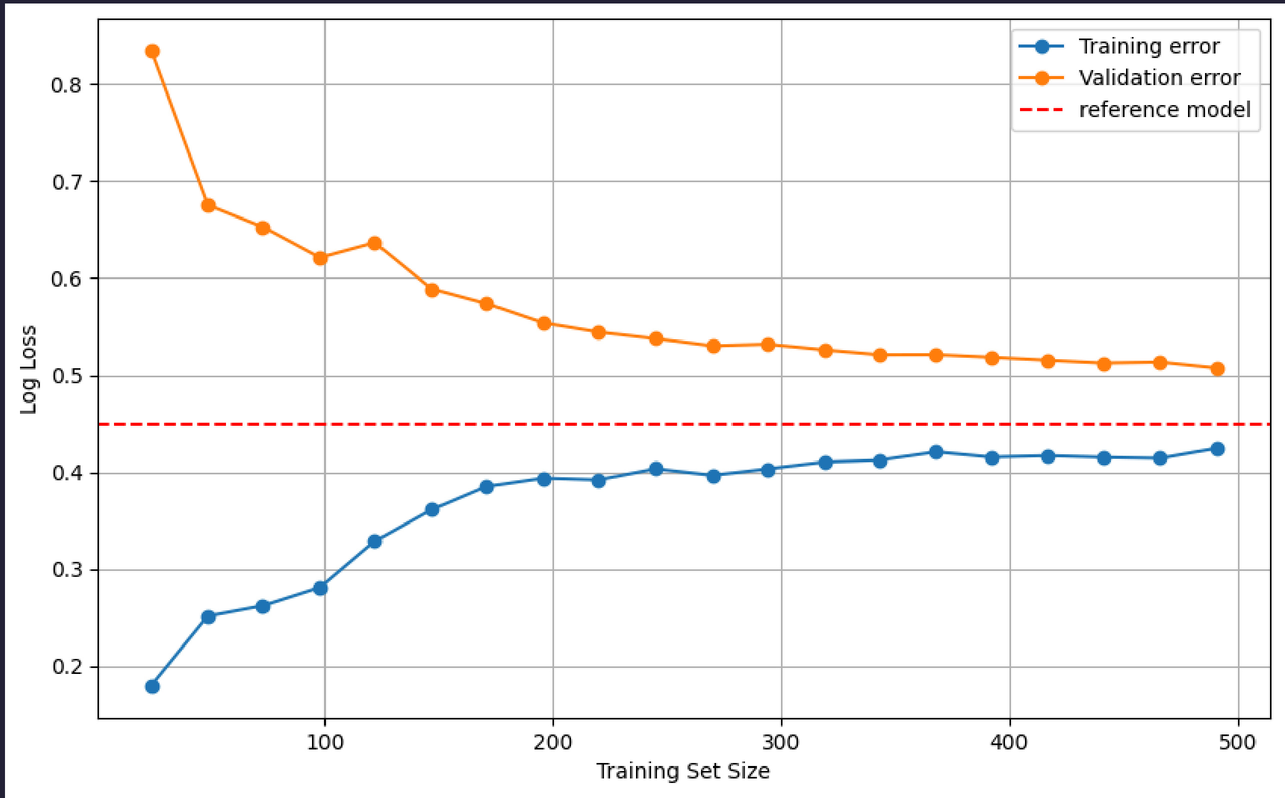


Will collecting more data help?

$J_{train} < J_{cv}$

➡️ high variance (overfit)

More data helps

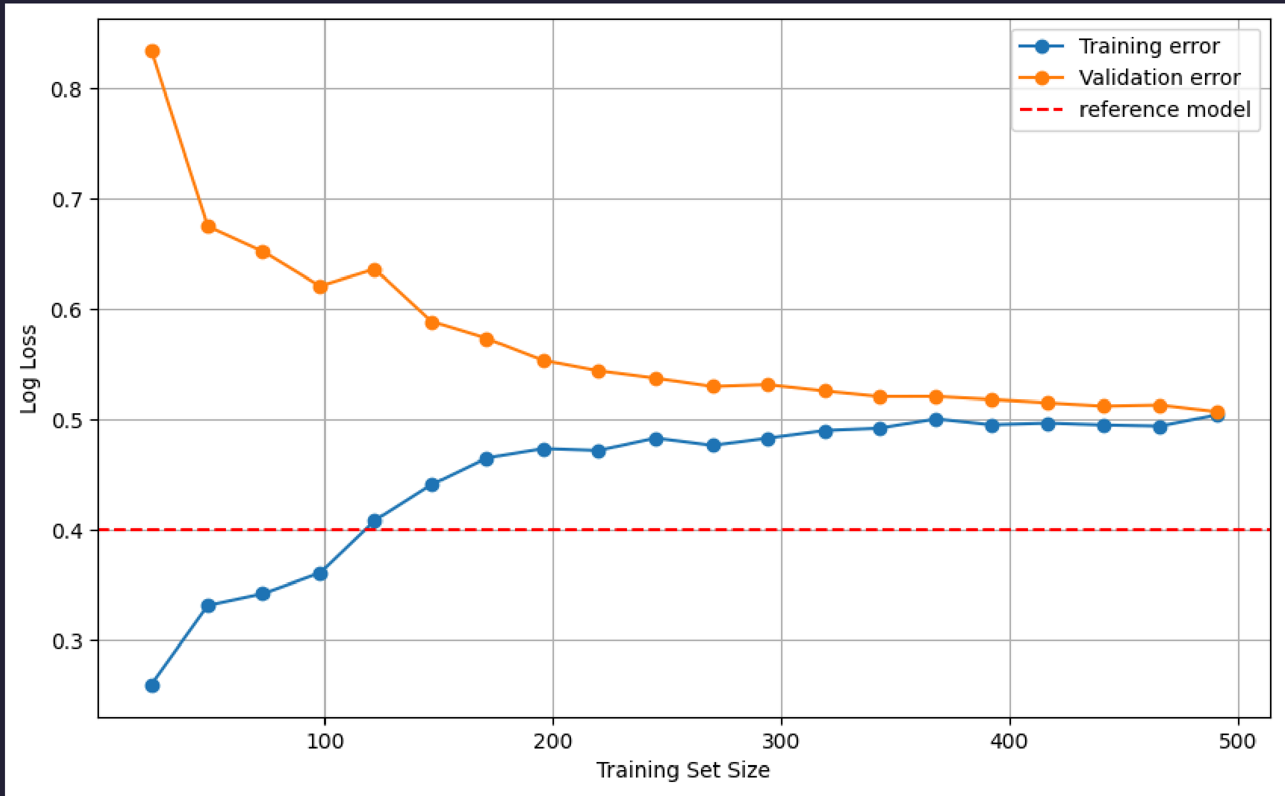


Will collecting more data help?

$J_{train} \approx J_{cv}$
$J_{train} > J_{reference}$
➡️ high bias (underfit)

More data won't help

Fit a more complex model



Debugging your ML model: High bias or high variance?

$$
\begin{tikzcd}
\text{Low training error?} \arrow[r, "Yes"] \arrow[d, "No"', "\text{High bias}"] & \text{Low validation error?} \arrow[r, "Yes"] \arrow[d, "No"', "\text{High variance}"] & \text{Done!} \\
\text{Bigger model} & \text{More data} &
\end{tikzcd}
$$

Does it fix bias or variance problem?

$$J = \frac{1}{m}\sum_{i=1}^{m} L(y^{(i)}, \hat{y}^{(i)}) + \lambda \sum_{j=1}^{n} w_j^2$$

Got large prediction errors. What do you do?

- Get more training examples
- Try smaller sets of features
- Try getting additional features
- Try adding polynomial features (e.g., x_1^2, x_1x_2, x_2^2)
- Try decreasing λ
- Try increasing λ

💻 Grid search for hyperparameter tuning