Machine Learning

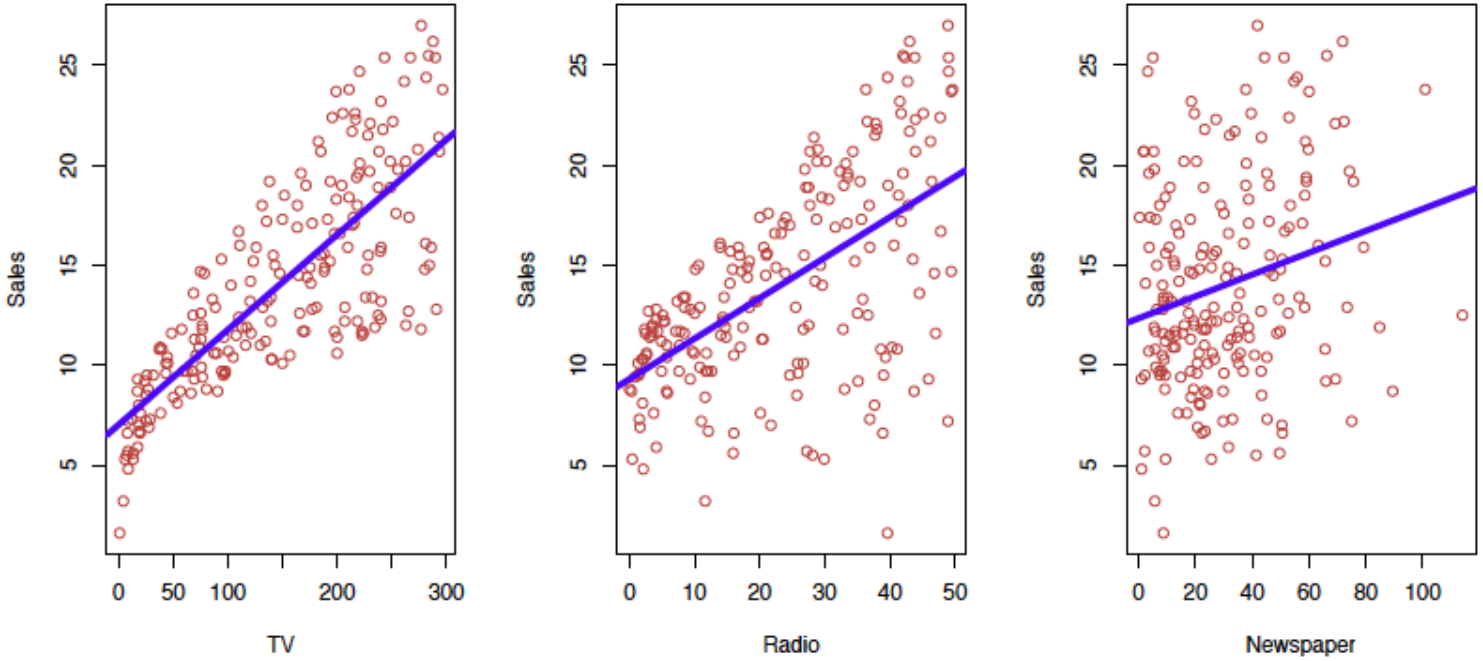
Supervised Machine Learning – Regression

Part 1: Parametric and Non-Parametric Approaches

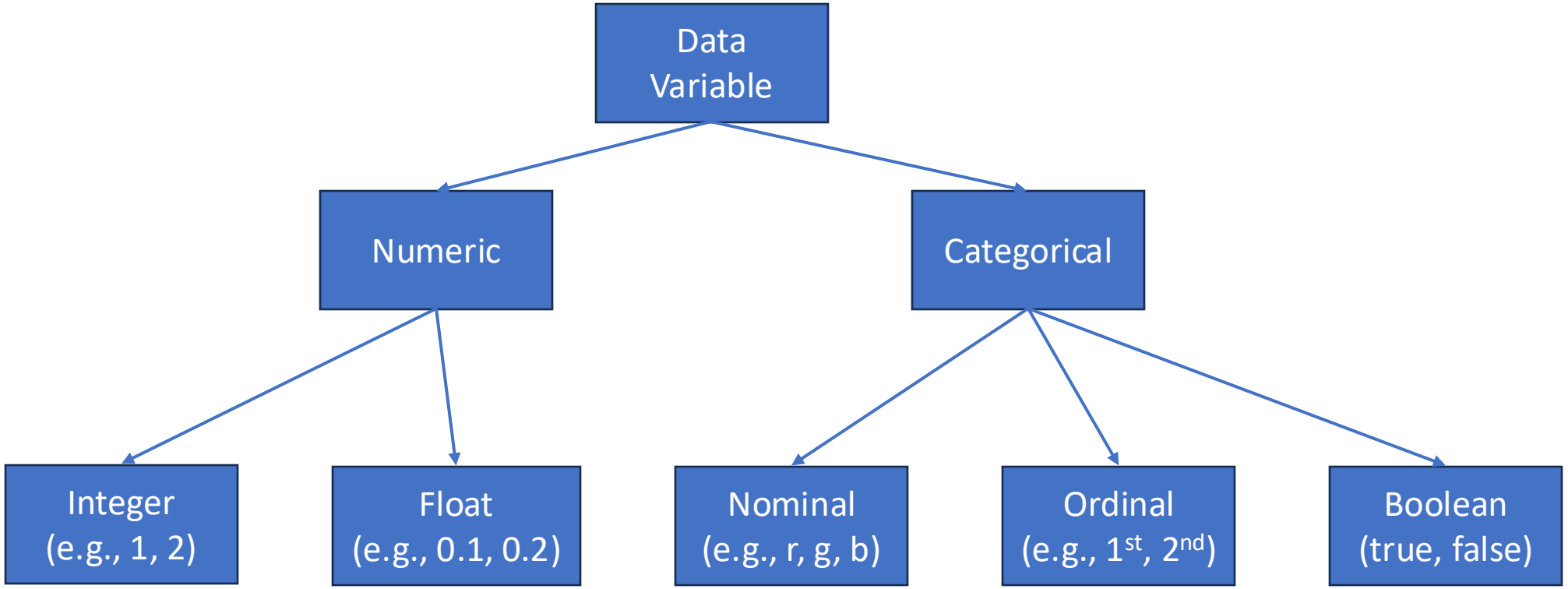
Regression, Data, Error, Performance

Prediction

- Shown are Sales vs amount spent on TV, Radio and Newspaper ads, with a blue linear-regression line fit separately to each.
- Can we predict Sales using these three?
- Perhaps we can do better using a model that combines all three: Sales $\approx f($ TV, Radio, Newspaper $)$



Numerical Data



- In regression, inputs and outputs are all numerical.
- In classification, output can be categorical (but in practice will be mapped to numerical)

Regression - Data Model

- In this example, Sales is a response/target/output; we denote it by y. TV is a feature/input/predictor; we denote it by x_1. Likewise, we denote Radio by x_2, and so on.

- In general, we can refer to the input vector as $\boldsymbol{x} \in I$, containing d input features ($I \subseteq \mathbb{R}^d$ is the "input space").

- We <u>assume</u> that there exists function f so that:

$$y(\boldsymbol{x}) = f(\boldsymbol{x}) + \epsilon \in \mathbb{R}_+$$

where ϵ is a <u>zero-mean **random variable**</u> (independent of $\boldsymbol{x}$) and captures measurement errors and noise.

- $f(\boldsymbol{x})$ is the <u>unknown</u> <u>true model</u> that describes input-output for our problem.

Regression - Goal

What is our goal here?

Our ideal goal is to design model $\hat{f}(\boldsymbol{x})$ that estimates the unknown $f(\boldsymbol{x})$ accurately as possible.

To do that, we will use multiple input/output examples.

The prediction of $y(\boldsymbol{x}) = f(\boldsymbol{x}) + \epsilon$ by $\hat{f}(\boldsymbol{x})$ is called **<u>regression</u>**.

Regression - Prediction Error

How good is my prediction?

True output: $y(\boldsymbol{x}) = f(\boldsymbol{x}) + \epsilon$

Prediction: $\hat{f}(\boldsymbol{x}) \approx f(\boldsymbol{x})$

Prediction error: $\mathrm{e} = y(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}) = \left(f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})\right) + \epsilon$

Model estimation error (reducible)

Noise error (irreducible)

Regression - Prediction Error (cont'd)

<u>Prediction error:</u> $\mathrm{e} = y(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}) = \left(f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})\right) + \epsilon$

<u>Model estimation error:</u> $f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})$

- This is reducible.
- Our mission in ML is to make this as low as possible. There are many ways that we'll discuss.

<u>Noise error:</u> ϵ

- Irreducible. Nothing we can do about it.
- *The performance of any model is limited by the quality of the data it operates on.*

Regression - Model Performance

Prediction error $\mathrm{e} = y(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}) = \left(f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})\right) + \epsilon$ is a random variable.

- Input $\boldsymbol{x}$ is random.
- Error $\boldsymbol{\epsilon}$ is random
- We assume that f is a deterministic function (same input gives same output). The same for $\hat{f}$.

Even for given input $\boldsymbol{x}$, e is random due to ϵ. We need to calculate the statistics of e.

Regression - Model Performance

For any given input, the error is a random variable with some mean and variance.

Zero mean and zero variance would be ideal (deterministic error zero). But what is feasible?

Mean: $E[e|\boldsymbol{x}] = \left(f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})\right)$

Variance: $Var\ [e|\boldsymbol{x}] = E[\ (e - E[e|\boldsymbol{x}])^2|\boldsymbol{x}] = E[\epsilon^2|\boldsymbol{x}] = Var[\epsilon] =: \sigma_\epsilon^2$

Mean Squared Error: $E[e^2|\boldsymbol{x}] = E\left[\left(y - \hat{f}(\boldsymbol{x})\right)^2|\boldsymbol{x}\right]$

$$MSE = E\left[\left(\left(f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})\right) - \epsilon\right)^2|\boldsymbol{x}\right] = E\left[\left(f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})\right)^2|\boldsymbol{x}\right] + \boldsymbol{E}[\boldsymbol{\epsilon}^2|\boldsymbol{x}] - \boldsymbol{E}\left[\boldsymbol{\epsilon}\left(f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})\right)|\boldsymbol{x}\right]$$

$$= \boxed{\left(f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})\right)^2} \boxed{+ \sigma_\epsilon^2}$$

Reducible

Irreducible (given data quality)

If you find these derivations challenging, you should review the provided notes on probability and random variables.

Parametric Modeling

Parametric Modeling

Goal: Train model $\hat{f}$ such that

$$f(\boldsymbol{x}) \approx \hat{f}(\boldsymbol{x}) \Leftrightarrow \left(f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})\right)^2 \approx 0 \quad \forall \boldsymbol{x} \in I$$

Basic assumption: True function f has a **parametric** form $f(\boldsymbol{x}) = m(\boldsymbol{x}; \boldsymbol{a})$ for some finite-size parameter vector $\boldsymbol{a}$ in parameter space H.

Example:

$$f(\boldsymbol{x}) = a_5 x_1^2 + a_4\, x_2^2 + a_3\, x_1 x_2 + a_2\, x_1 + a_1 x_2 + a_0$$

- The parametric model m is the bivariate 2nd-degree polynomial.
- The parameters are in parameter vector $\boldsymbol{a} = [a_0, a_1, a_2, a_3, a_4, a_5]^T \in H = R^6$.

Training the Parametric Model

Assumption: $f(\boldsymbol{x}) = m(\boldsymbol{x}; \boldsymbol{a})$

Training parametric model $\hat{f} \approx f$ breaks in two steps:

1) Identify $\boldsymbol{m}$

2) Identify $\boldsymbol{a}$

Training the Parametric Model

Assume the simplified case that true m is known.

To train $\hat{f}$ is to train the parameters of the model $\boldsymbol{a} \in H$.

To do that, we rely on a set of training input/output data

$$S = \{(y_n, \boldsymbol{x}_n)\}_{n=1}^{N}$$

We train $\hat{f}(\boldsymbol{x}) = m(\boldsymbol{x}; \hat{\boldsymbol{a}})$, by solving

$$\hat{\boldsymbol{a}} = \text{argmin}_{\boldsymbol{b} \in H} \; L(\boldsymbol{b}; S)$$

L is the <u>loss function</u>: It measures how well $m(\boldsymbol{x}; \hat{\boldsymbol{a}})$ represents the available training data.

MSE Training

Recall performance metric:

$$\text{MSE} = E\left[\left(y(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})\right)^2 |\boldsymbol{x}\right] = E[(y - m(\boldsymbol{x}; \hat{\boldsymbol{a}}))^2 |\boldsymbol{x}]$$

Mean can be estimated by sample-average.

In this case, we can use the training data: $\widehat{\text{MSE}} = \frac{1}{N}\sum_{n=1}^{N}(y_n - m(\boldsymbol{x}_n; \hat{\boldsymbol{a}}))^2$

<u>MSE training</u>:

$$\hat{\boldsymbol{a}} = \text{argmin}_{\boldsymbol{b} \in H}\ L(\boldsymbol{b}; S) = \text{argmin}_{\boldsymbol{b} \in H} \frac{1}{N}\sum_{n=1}^{N}(y_n - m(\boldsymbol{x}_n; \boldsymbol{b}))^2$$

Parameter Training Considerations

The quality of my parameter training is determined by 3 variables.

- Number of parameters.
- Number of available training data N.
- Noise variance in training data σ_ϵ^2.

Good training for more model parameters in m needs more training data of better quality.

If m is known, better parameter estimates $\hat{\boldsymbol{a}}$ means better model $\hat{f}$.

Example: Univariate Polynomial Model

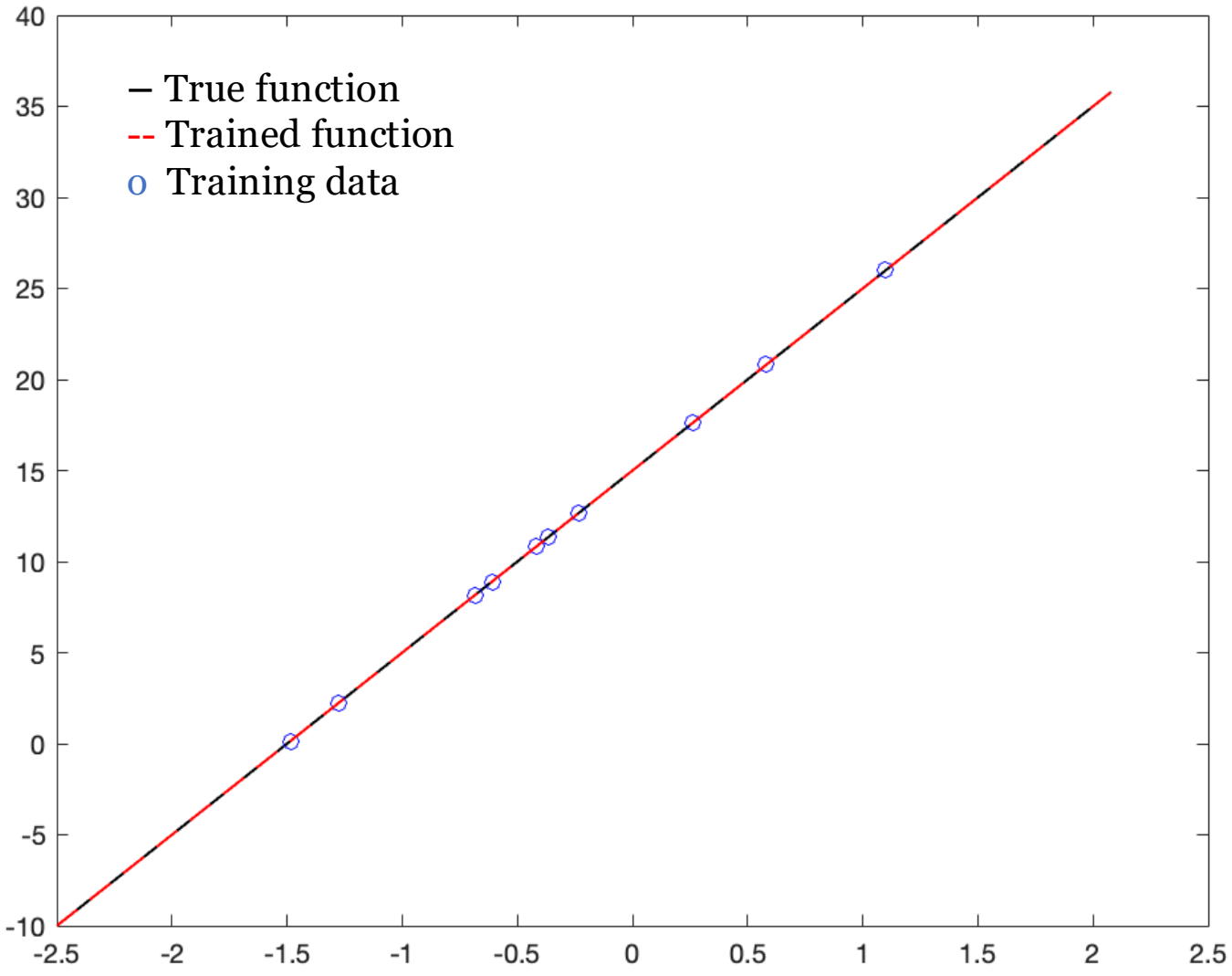
For simplicity, assume 1-D input and polynomial f.

- Constant (degree 0): $f(\mathrm{x}) = \mathrm{a}_0$ (no interest)
- Linear (degree 1): $f(\mathrm{x}) = a_1 x + a_0$
- Quadratic (degree 2): $f(\mathrm{x}) = a_2 x^2 + a_1 x + a_0$
- Cubic (degree 3): $f(\mathrm{x}) = a_3 x^3 + a_2 x^2 + a_1 x + a_0$

For degree-k polynomial, we need to estimate $k + 1$ parameters $\boldsymbol{a} = [a_0, a_1, \dots, a_k]^T$.

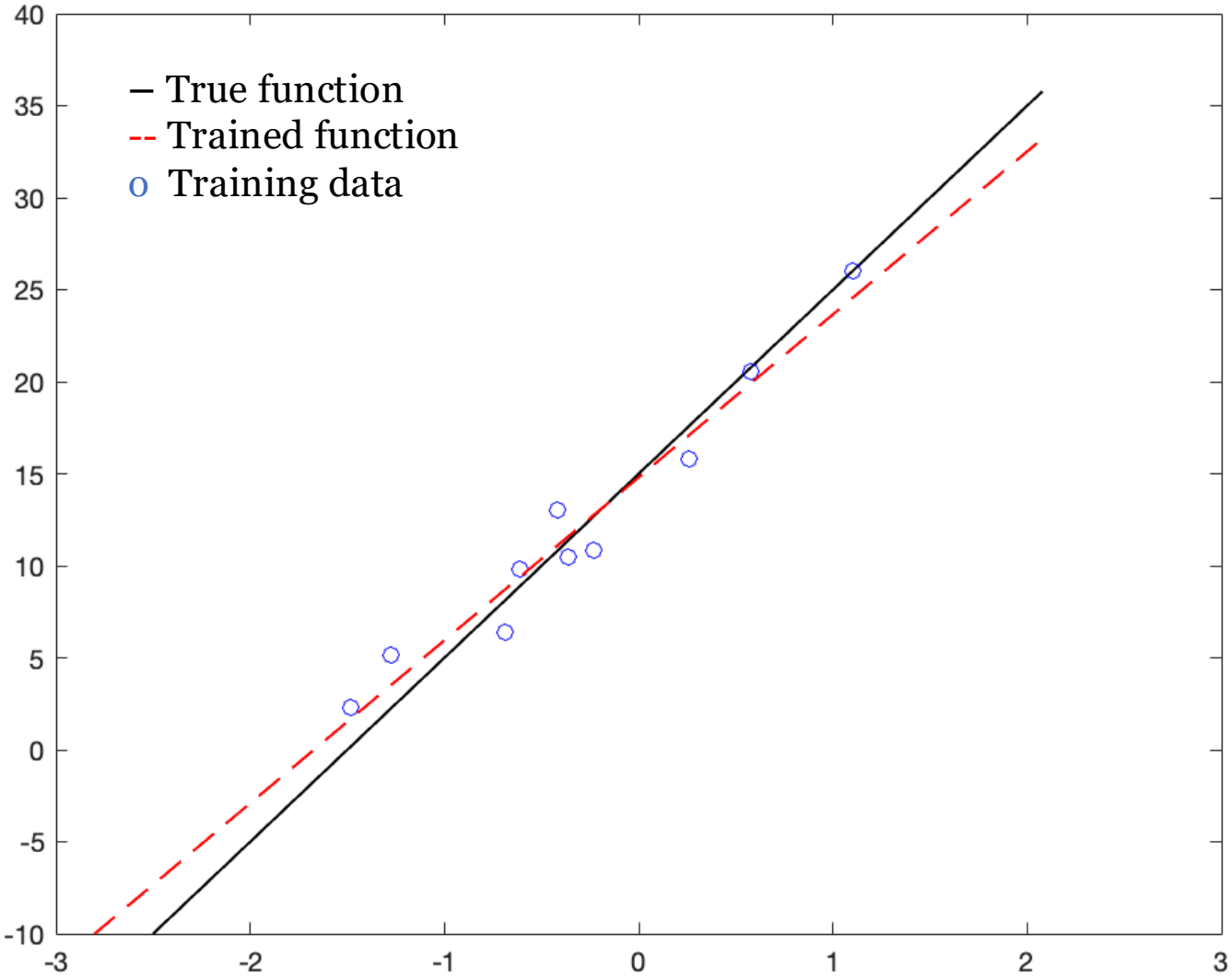
Example: Varying k, N, σ_ϵ^2

- $k = 1$
- $N = 10$
- $\sigma_\epsilon^2 = 0$



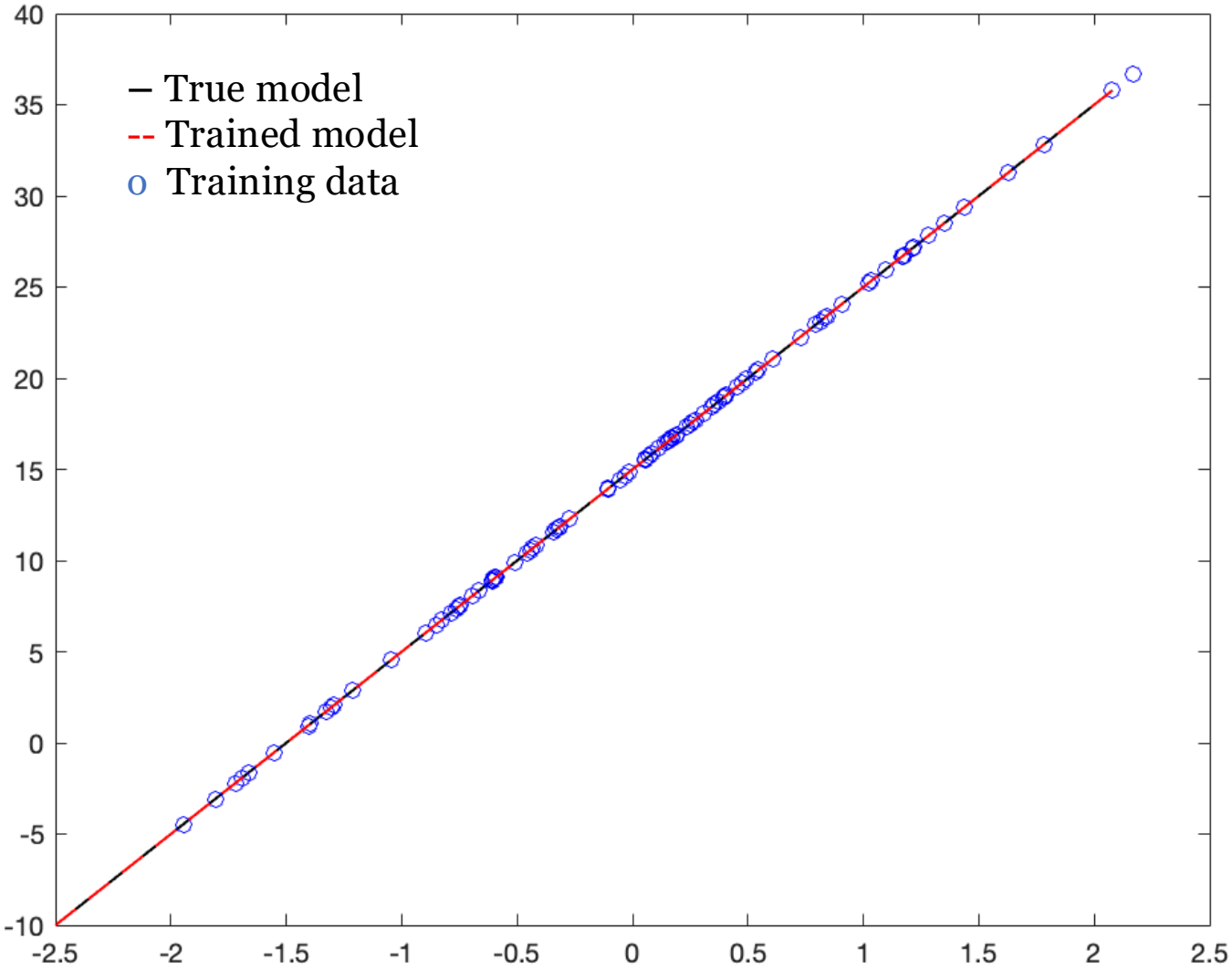
Example: Varying k, N, σ_ϵ^2 (cont'd)

- $k = 1$
- $N = 10$
- $\sigma_\epsilon^2 = 4$



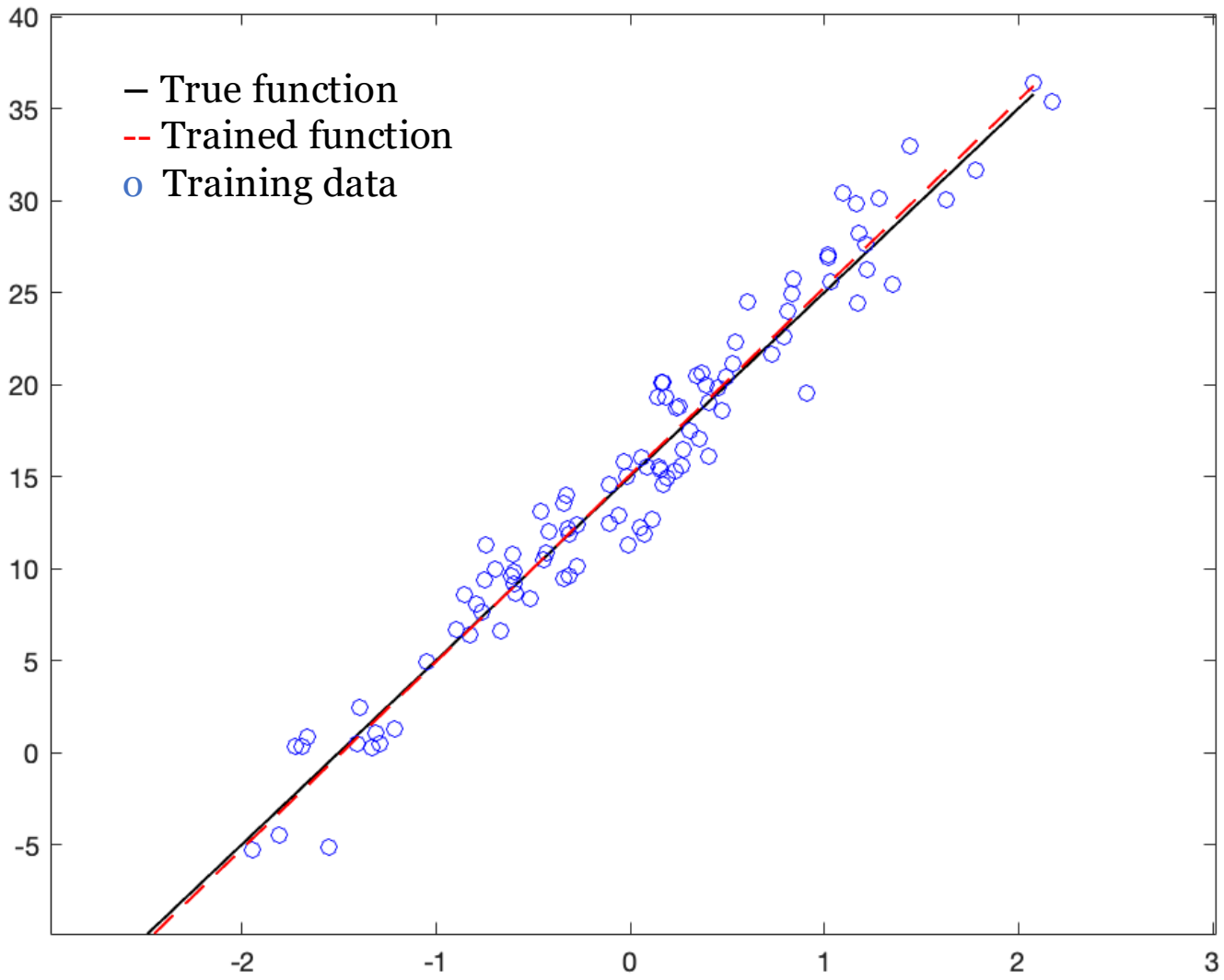
Example: Varying k, N, σ_ϵ^2 (cont'd)

- $k = 1$
- $N = 100$
- $\sigma_\epsilon^2 = 0$



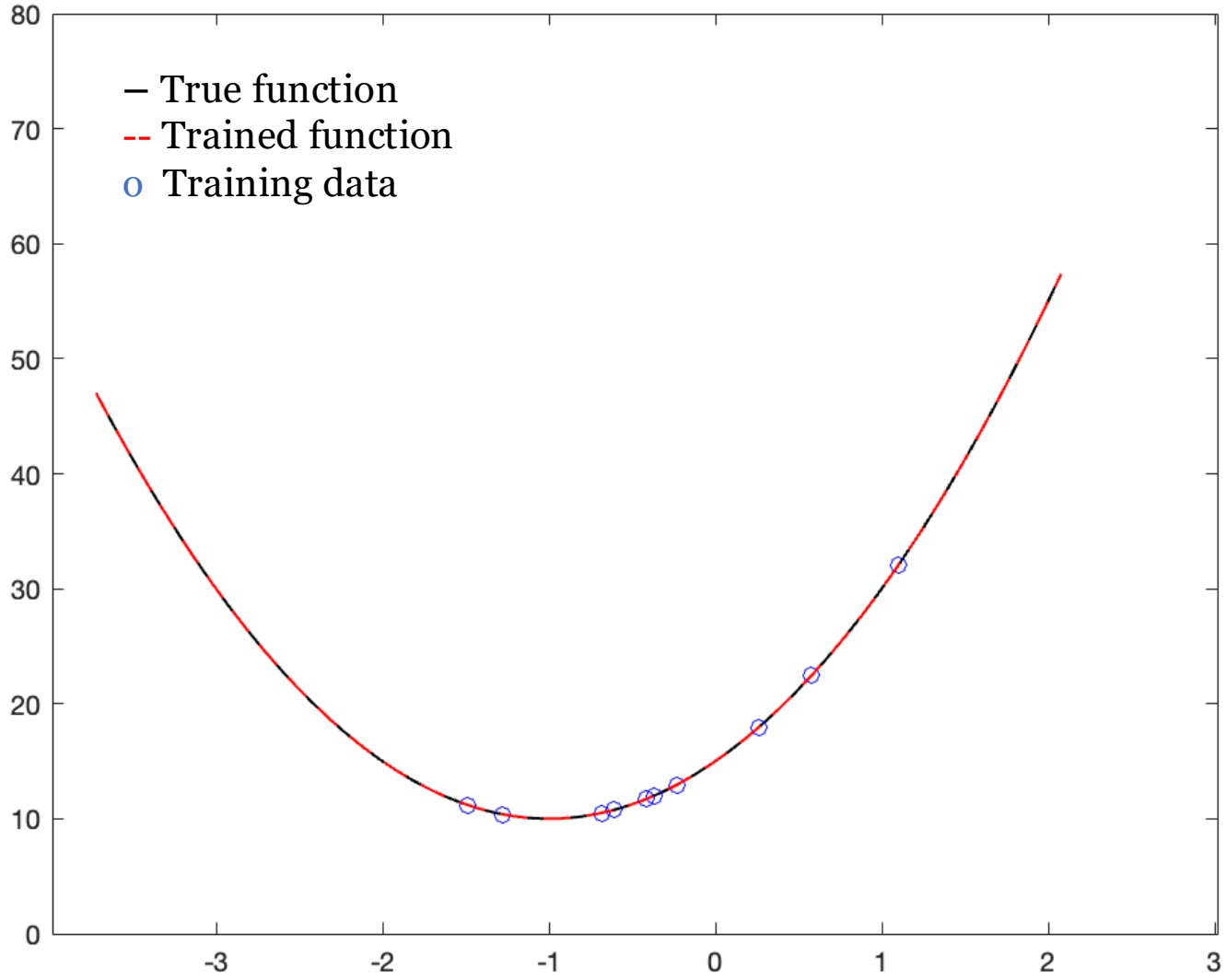
Example: Varying k, N, σ_ϵ^2 (cont'd)

- $k = 1$
- $N = 100$
- $\sigma_\epsilon^2 = 4$



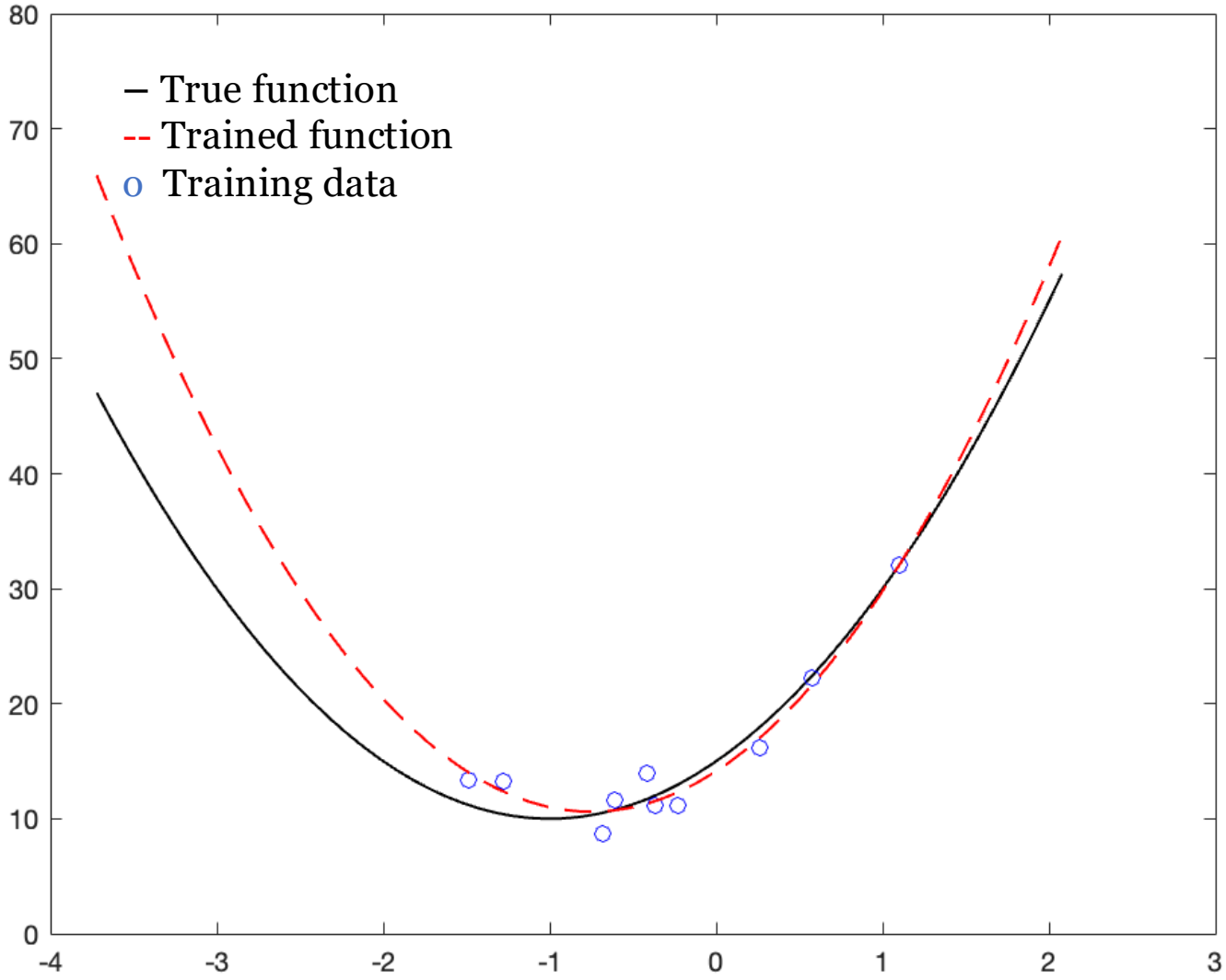
Example: Varying k, N, σ_ϵ^2 (cont'd)

- $k = 2$
- $N = 10$
- $\sigma_\epsilon^2 = 0$



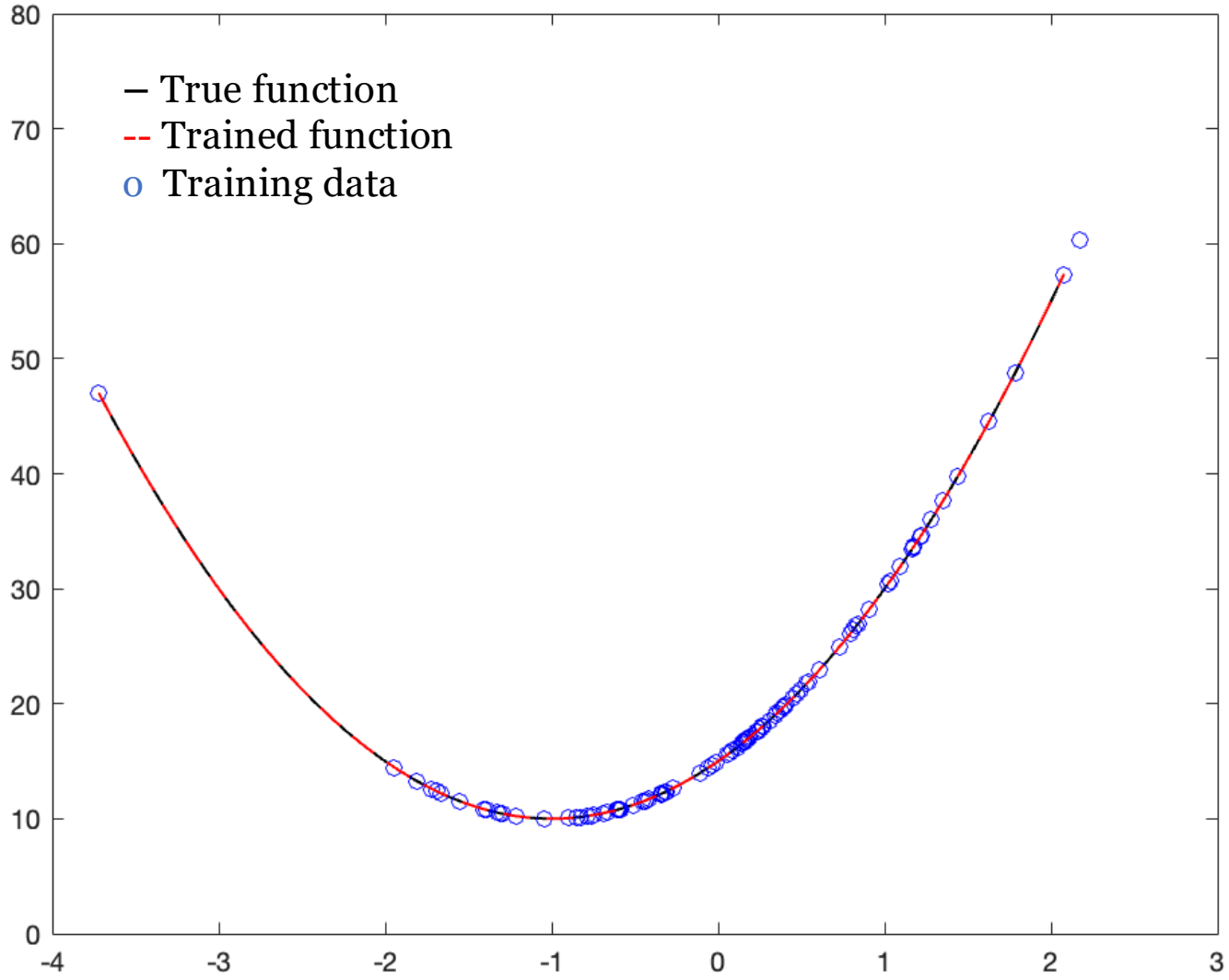
Example: Varying k, N, σ_ϵ^2 (cont'd)

- $k = 2$
- $N = 10$
- $\sigma_\epsilon^2 = 4$



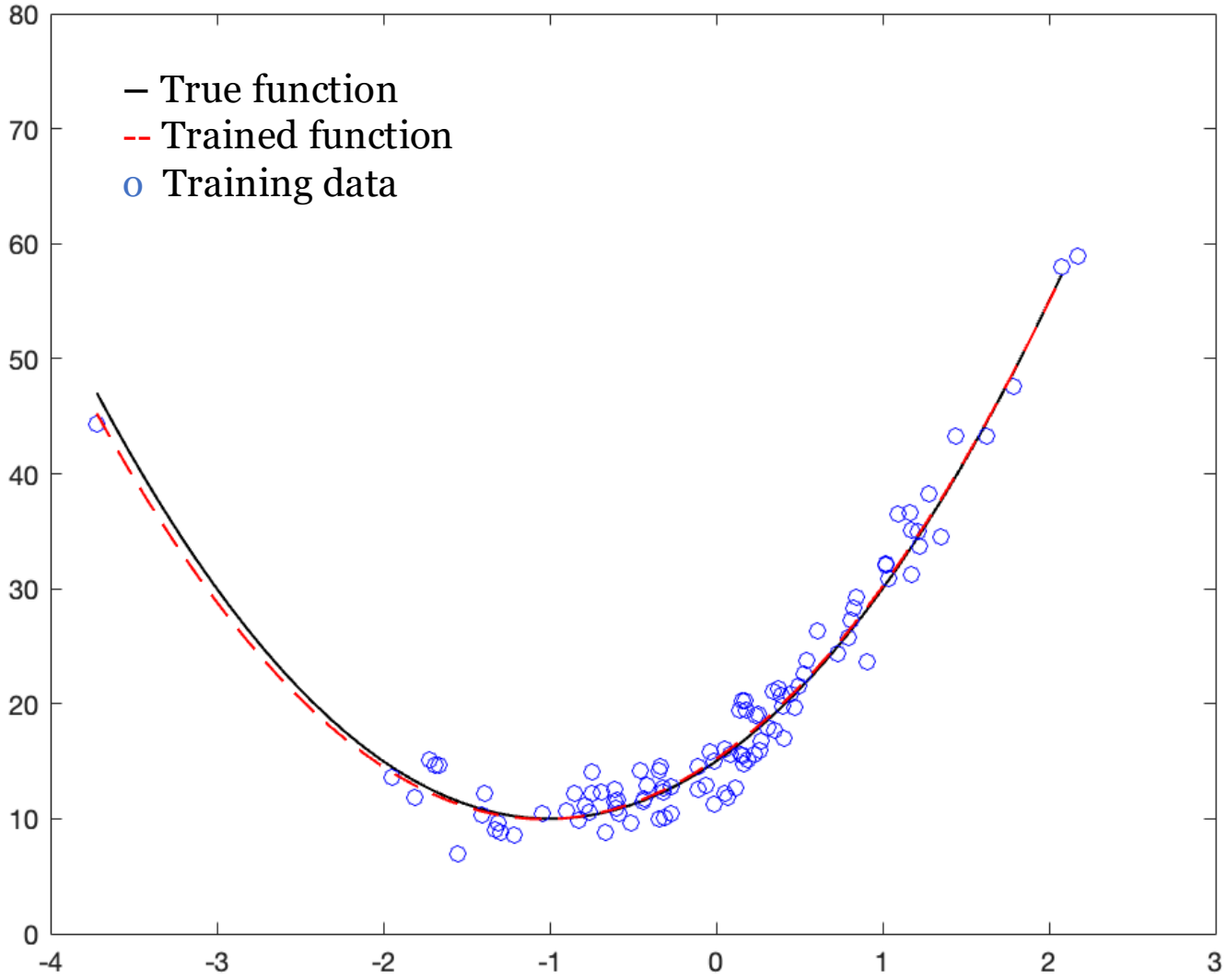
Example: Varying k, N, σ_ϵ^2 (cont'd)

- $k = 2$
- $N = 100$
- $\sigma_\epsilon^2 = 0$



Example: Varying k, N, σ_ϵ^2 (cont'd)

- $k = 2$
- $N = 100$
- $\sigma_\epsilon^2 = 4$



Example: Varying k, N, σ_ϵ^2 (cont'd)

- In all cases, $\sigma_\epsilon^2 = 0$ resulted $\hat{f} = f$, for all tested values of k and N.
- For $\sigma_\epsilon^2 > 0$, for all k, higher N resulted in better training.
- For $\sigma_\epsilon^2 > 0$, for any given value of N, we attained better training for lower k.

Parametric Model Training – Open Questions

Basic assumption: $f(x) = m(x; a)$

Training parametric model $\hat{f}$ breaks in two steps:

1) Identify m

   If unknown (common), estimate $\hat{m}$.

   How?

2) Identify a

   Train $\hat{a}$.

   What loss? How do we solve the fitting problem?

What are the trade-offs between estimation of $\hat{m}$ and training of $\hat{\boldsymbol{a}}$?

We will return to this discussion…

Non-Parametric Modeling

Non-Parametric Regression

Recall, our goal is to train $\hat{f}$ such that, for all $\boldsymbol{x}$ in input space,

$$f(\boldsymbol{x}) \approx \hat{f}(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in I$$

Recall that $y(\boldsymbol{x}) = f(\boldsymbol{x}) + \epsilon$ with $E[\epsilon] = 0$. That is,

$$f(\boldsymbol{x}) = E[y(\boldsymbol{x})] \quad \forall \boldsymbol{x} \in I$$

Therefore, I want to train $\hat{f}$ such as

$$\hat{f}(\boldsymbol{x}) \approx E[y(\boldsymbol{x})] \quad \forall \boldsymbol{x} \in I$$

- We do not have $E[y(\boldsymbol{x})]$ for any $\boldsymbol{x}$.
- We just have $y(\boldsymbol{x})$ for some values of $\boldsymbol{x}$ (in S).

Nearest-Neighbor Regression

<u>Idea:</u> Estimate mean $E[y(\boldsymbol{x})]$ by averaging values of $y(\boldsymbol{x})$.

Coherent samples: For any $\boldsymbol{x} \in I$, to estimate $E[y(\boldsymbol{x})]$, I need to have many instances of $y(\boldsymbol{x}) = f(\boldsymbol{x}) + \epsilon$, or this same $\boldsymbol{x}$.

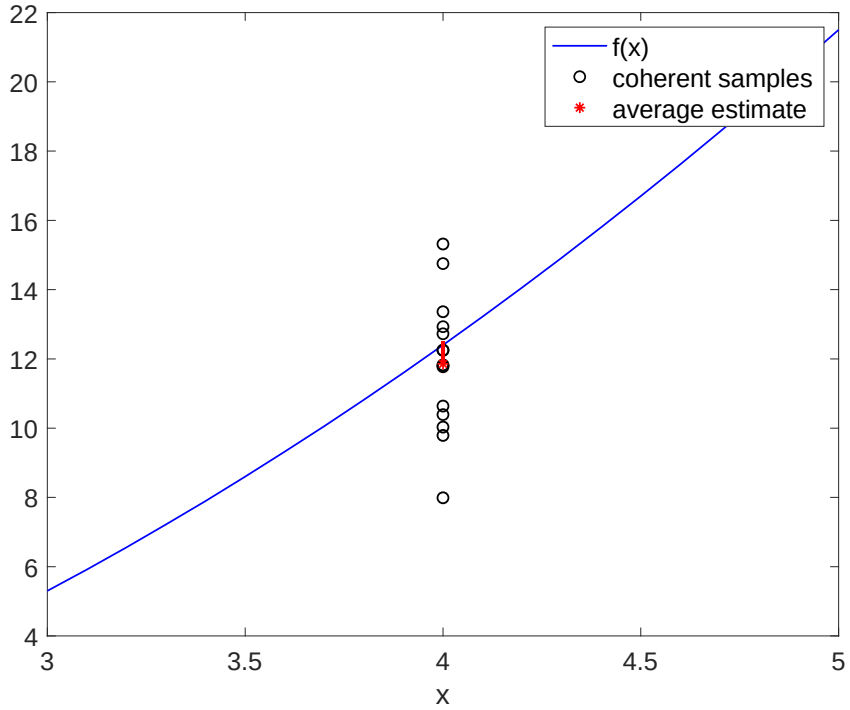
How many distinct instances $(y(\boldsymbol{x}), \boldsymbol{x})$ do we have in S for each $\boldsymbol{x} \in I$?

In S, we only have <u>some/few/one</u> pairs $(y(\boldsymbol{x}), \boldsymbol{x})$ for <u>some</u> values of $\boldsymbol{x}$.

<u>Idea:</u> I will relax coherence constraint. I will estimate $E[y(\boldsymbol{x})]$ by averaging $y(\boldsymbol{z})$ for all available $\boldsymbol{z}$ in "the neighborhood" of $\boldsymbol{x}$.

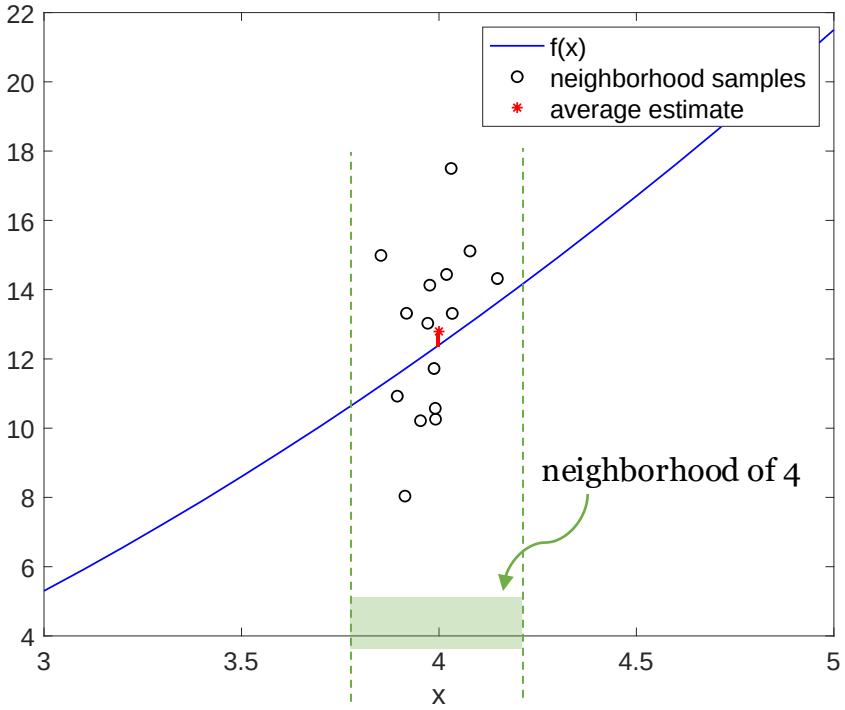
Nearest-Neighbor Regression (cont'd)

Slow varying function. Average estimation on coherent samples $y(4) = f(4) + \epsilon$, $\sigma_\epsilon = 2$.



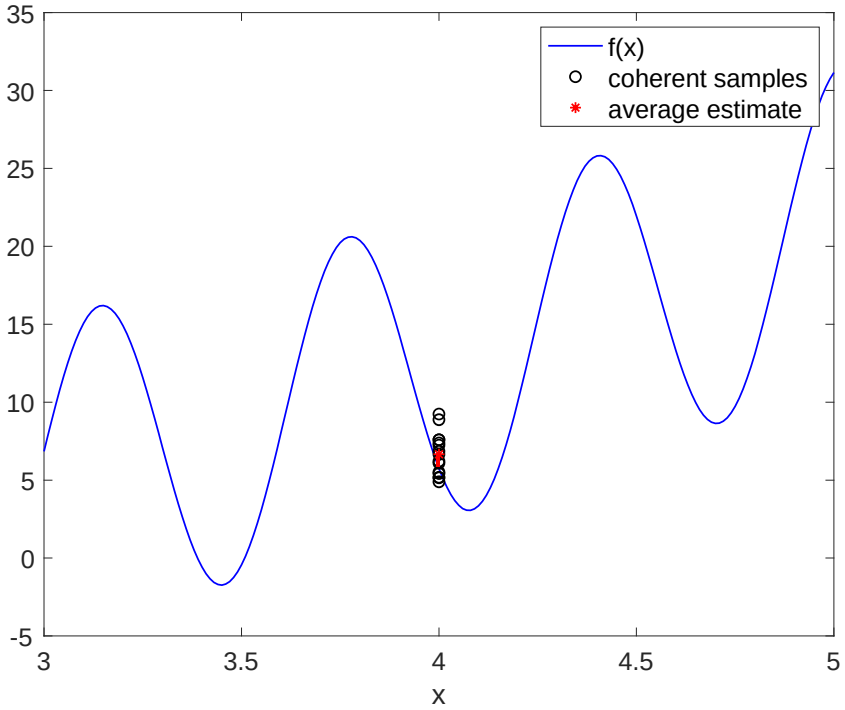
Nearest-Neighbor Regression (cont'd)

<u>Slow</u> varying function. Average estimation on neighborhood samples $y(4+n) = f(4+n) + \epsilon$, $\sigma_\epsilon = 2$.



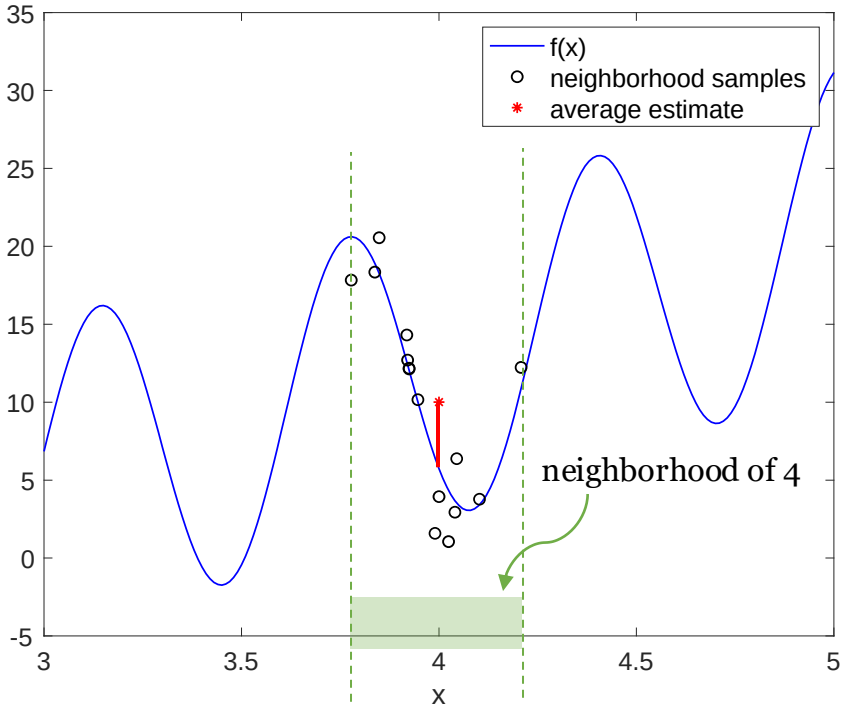
Nearest-Neighbor Regression (cont'd)

<u>Fast</u> varying function. Average estimation on coherent samples $y(4) = f(4) + \epsilon$, $\sigma_\epsilon = 2$.



Nearest-Neighbor Regression (cont'd)

<u>Fast</u> varying function. Average estimation on neighborhood samples $y(4+n) = f(4+n) + \epsilon$, $\sigma_\epsilon = 2$.



Nearest-Neighbor Regression (cont'd)

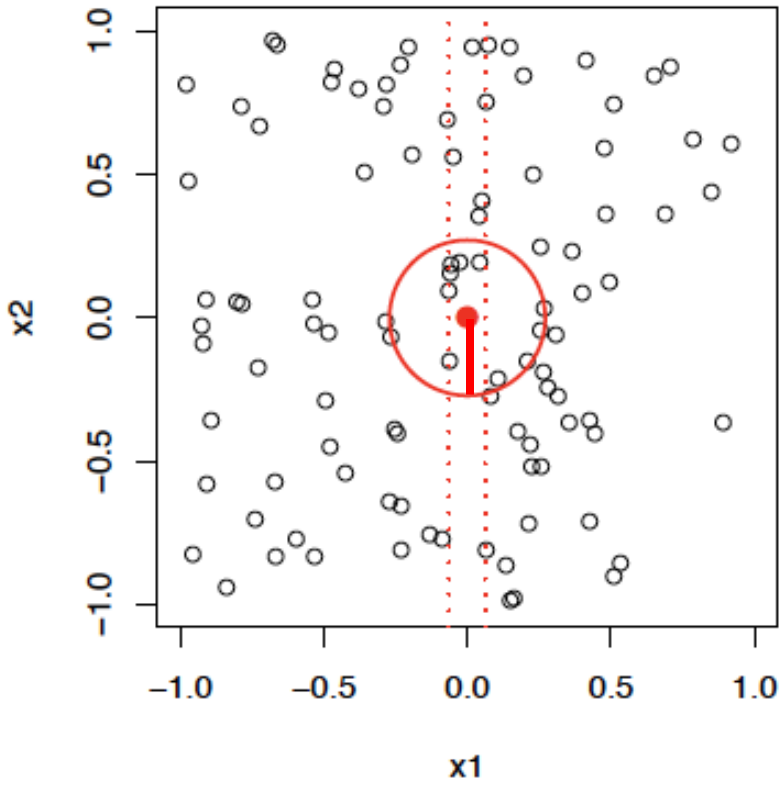
- For each $\boldsymbol{x} \in I$ let $V(\boldsymbol{x})$ be the set of points in S that are the nearest to $\boldsymbol{x}$.
  - nearest in terms of Euclidean distance, or any other "distance" metric
- We call these points the "nearest neighbors" of $\boldsymbol{x}$

- Option 1: Limit $K = |V(\boldsymbol{x})|$

  For example, $K = 10\% \, |S| = 0.1 \, N$

- Option 2: $\max_{\boldsymbol{z} \in V(\boldsymbol{x})} \text{distance}(\boldsymbol{z}, \boldsymbol{x}) \leq C$

  For example, $\|\boldsymbol{z} - \boldsymbol{x}\|_2 \leq C$



Nearest-Neighbor Regression (cont'd)

For every $\boldsymbol{x} \in I$, estimate $f(x) = E[y(\boldsymbol{x})]$ by the nearest neighbor average:

- Identify input neighborhood $V(\boldsymbol{x}) = \{\boldsymbol{z}_k\}_{k=1}^{K}$ and corresponding outputs $\{y_k\}_{k=1}^{K}$.

- Estimate $\hat{f}(\boldsymbol{x}) = \frac{1}{K}\sum_{k=1}^{K} y_k$.

The Curse of Dimensionality

- Nearest-neighbor averaging performance drops as d increases.
- **This is part of "the curse of dimensionality"**
- Nearest neighbors tend to be far away in high dimensions.
- We need to get a reasonable fraction of the N values of y_i to average to bring the variance down – e.g., 10%.
- A 10% neighborhood in high dimensions need no longer be local, so we lose the spirit of estimating $f(\boldsymbol{x})$ by local averaging.

The Curse of Dimensionality (cont'd)