More complicated models and their pitfalls

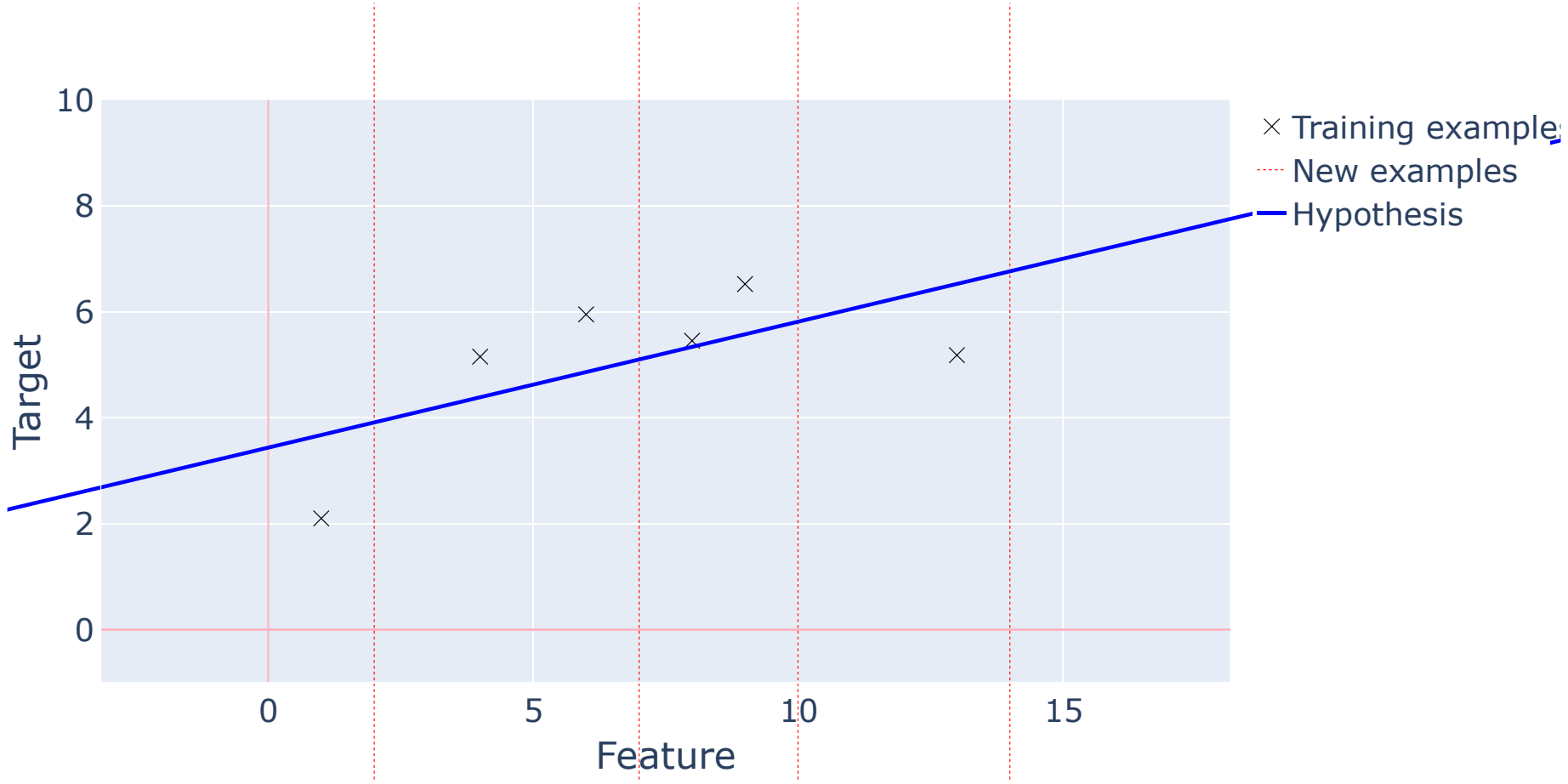
Today’s outline

- More complicated model
- Pitfall of complicated model: Overfitting
- Checking overfitting: validation
- Avoid overfitting: regularisation

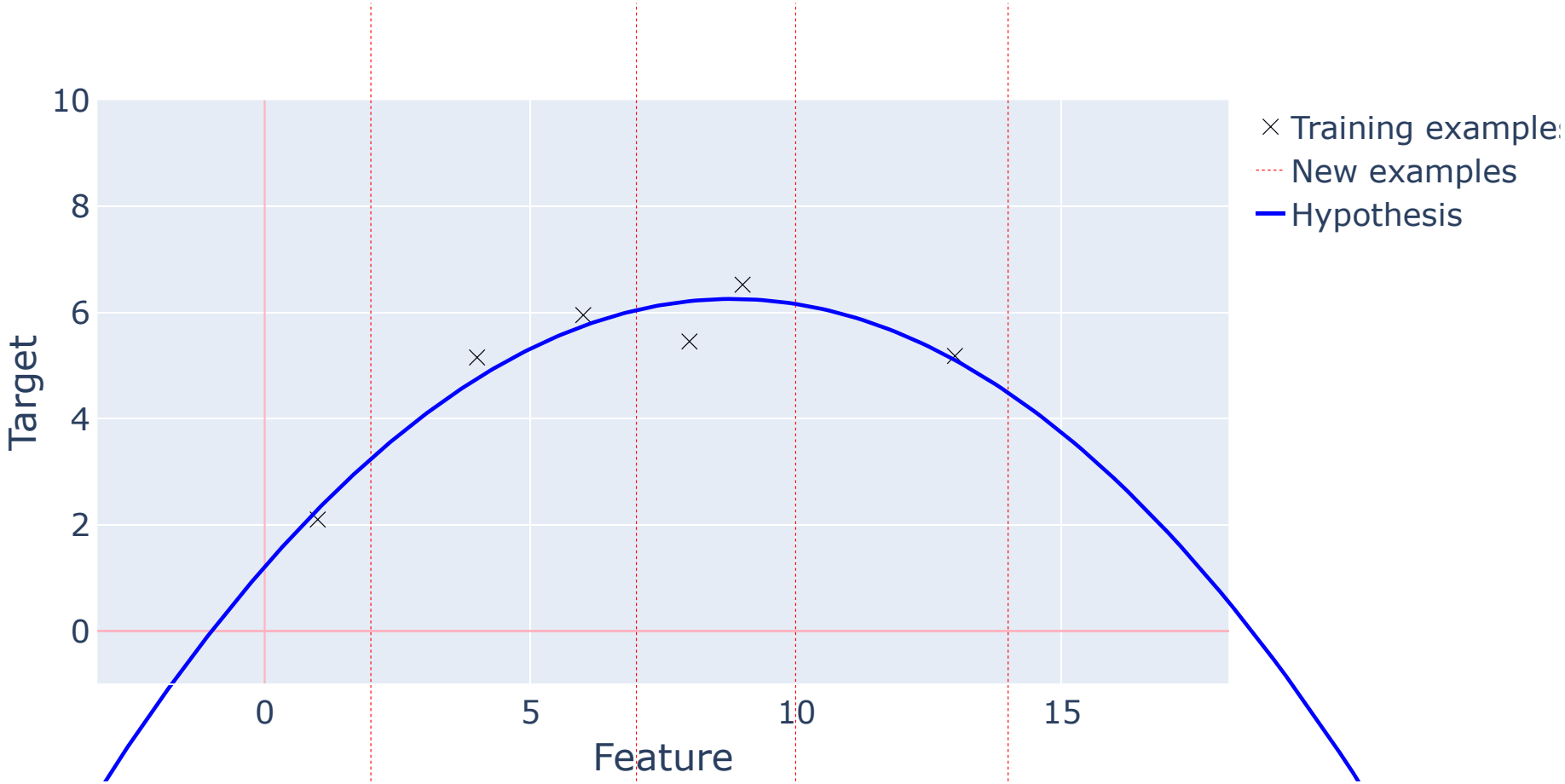
Beyond original features

Linear combination of the original features is not always effective enough. Using more complex features is the first step for NNs.

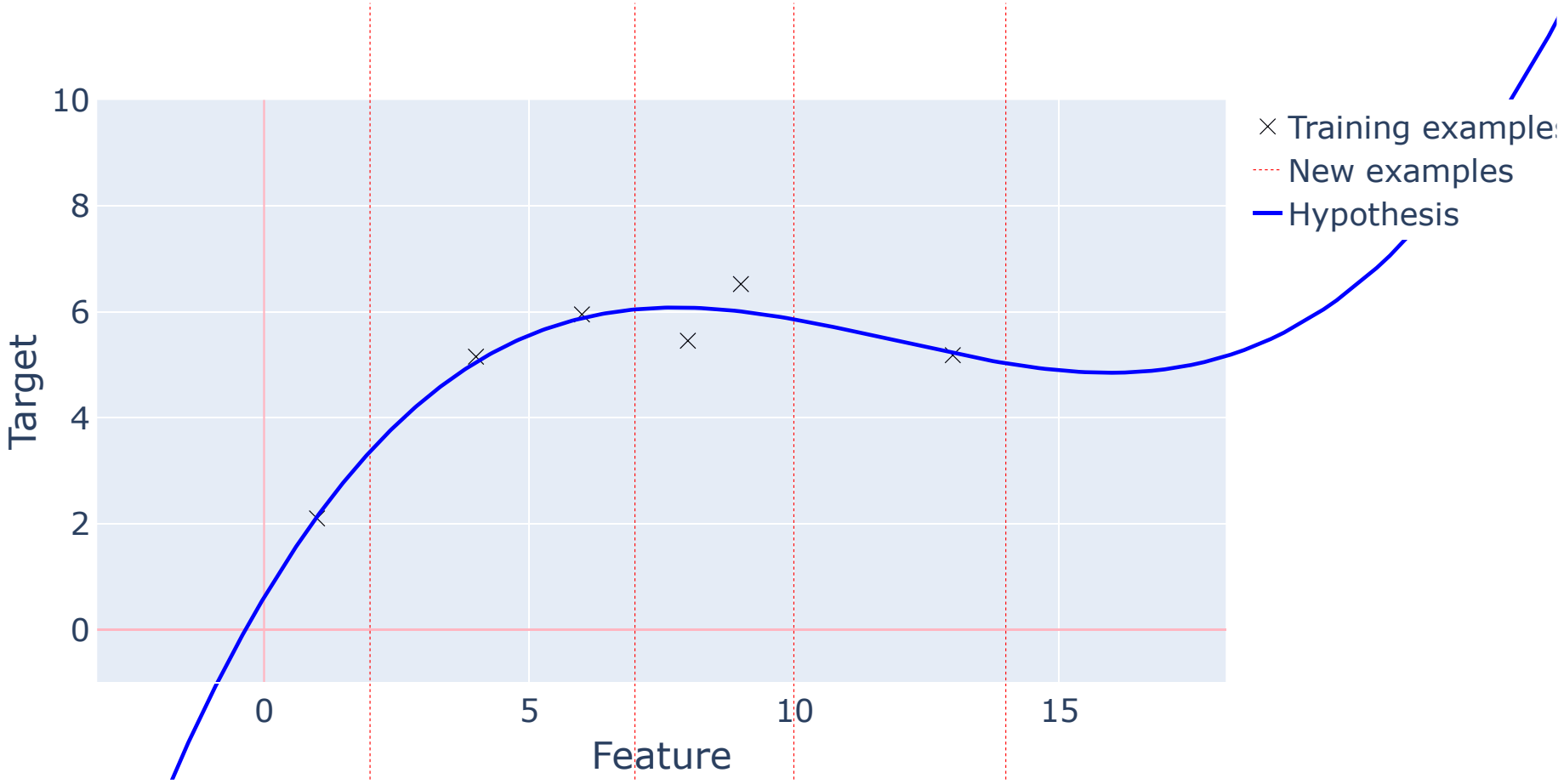
Univariate linear regression



Polynomial regression (degree: 2)



Polynomial regression (degree: 3)



Polynomial regression as a multivariable linear regression

- Original variables

	$x_0^{(i)}$	$x_1^{(i)}$	$x_2^{(i)} = \left(x_1^{(i)}\right)^2$	$x_3^{(i)} = \left(x_1^{(i)}\right)^3$		$y^{(i)}$
$\left(\boldsymbol{x}^{(0)}\right)^{\mathsf{T}}$	1	8.33	$8.33^2 = 69.4$	$8.33^3 = 578$	$y^{(0)}$	4.53
$\left(\boldsymbol{x}^{(1)}\right)^{\mathsf{T}}$	1	3.38	$3.38^2 = 11.4$	$3.38^3 = 38$	$y^{(1)}$	1.84
	⋮	⋮	⋮	⋮	⋮	⋮
$\left(\boldsymbol{x}^{(m-1)}\right)^{\mathsf{T}}$	1	1.72	$1.72^2 = 3.0$	$1.72^3 = 5$	$y^{(m-1)}$	0.95

Polynomial features: Multivariable case

- Original variables: x_1, x_2, x_3, x_4
- 2nd degree:

$$(x_1)^2, x_1x_2, x_1x_3, x_1x_4,$$
$$(x_2)^2, x_2x_3, x_2x_4,$$
$$(x_3)^2, x_3x_4,$$
$$(x_4)^2$$

- 3rd degree:

$$(x_1)^3, (x_1)^2x_2, (x_1)^2x_3, (x_1)^2x_4, x_1(x_2)^2, x_1x_2x_3, x_1x_2x_4, x_1(x_3)^2, x_1x_3x_4, x_1(x_4)^2,$$
$$(x_2)^3, (x_2)^2x_3, (x_2)^2x_4, x_2(x_3)^2, x_2x_3x_4, x_2(x_4)^2,$$
$$(x_3)^3, (x_3)^2x_4, x_3(x_4)^2,$$
$$(x_4)^4$$

- Higher-degree: impractical

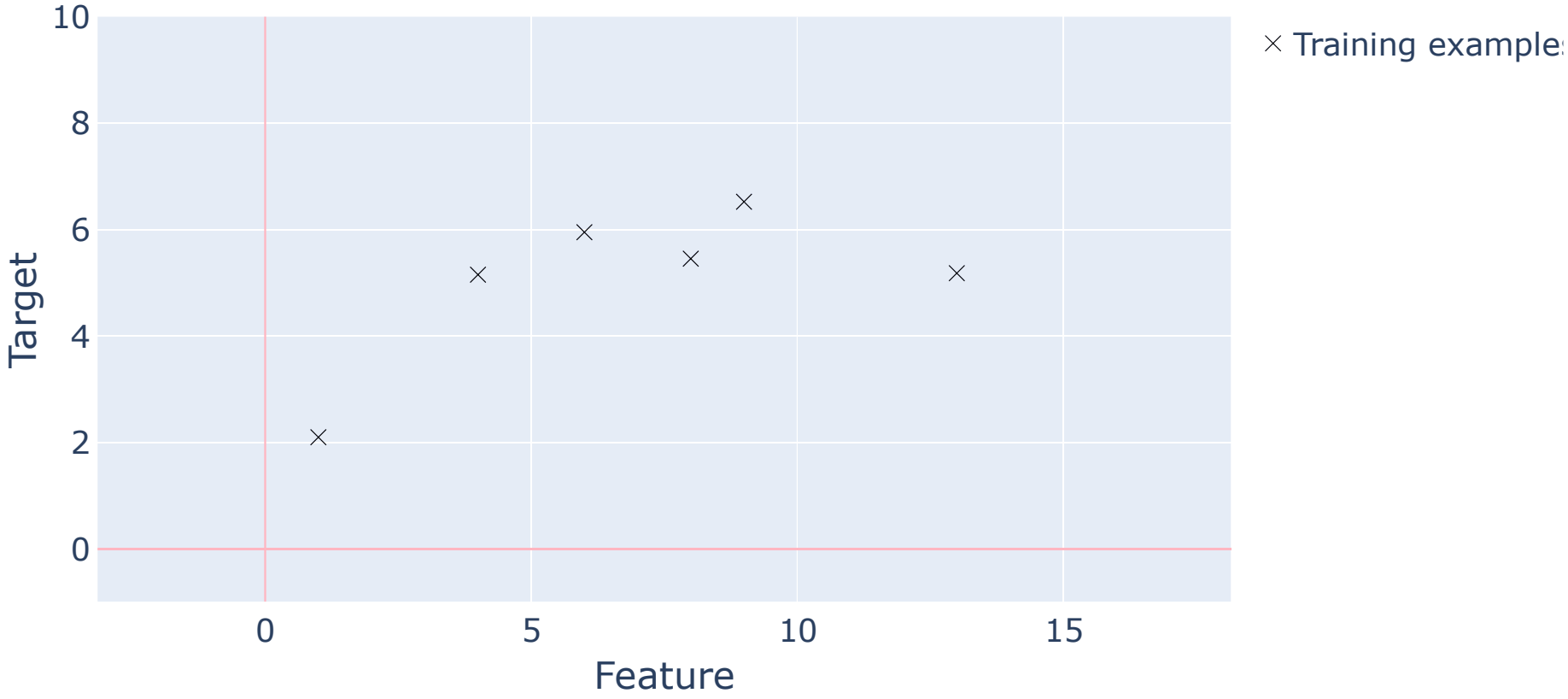
Modern solutions

- Neural network: (automatic feature selection)
- Kernel trick: (computation trick for infinite features)

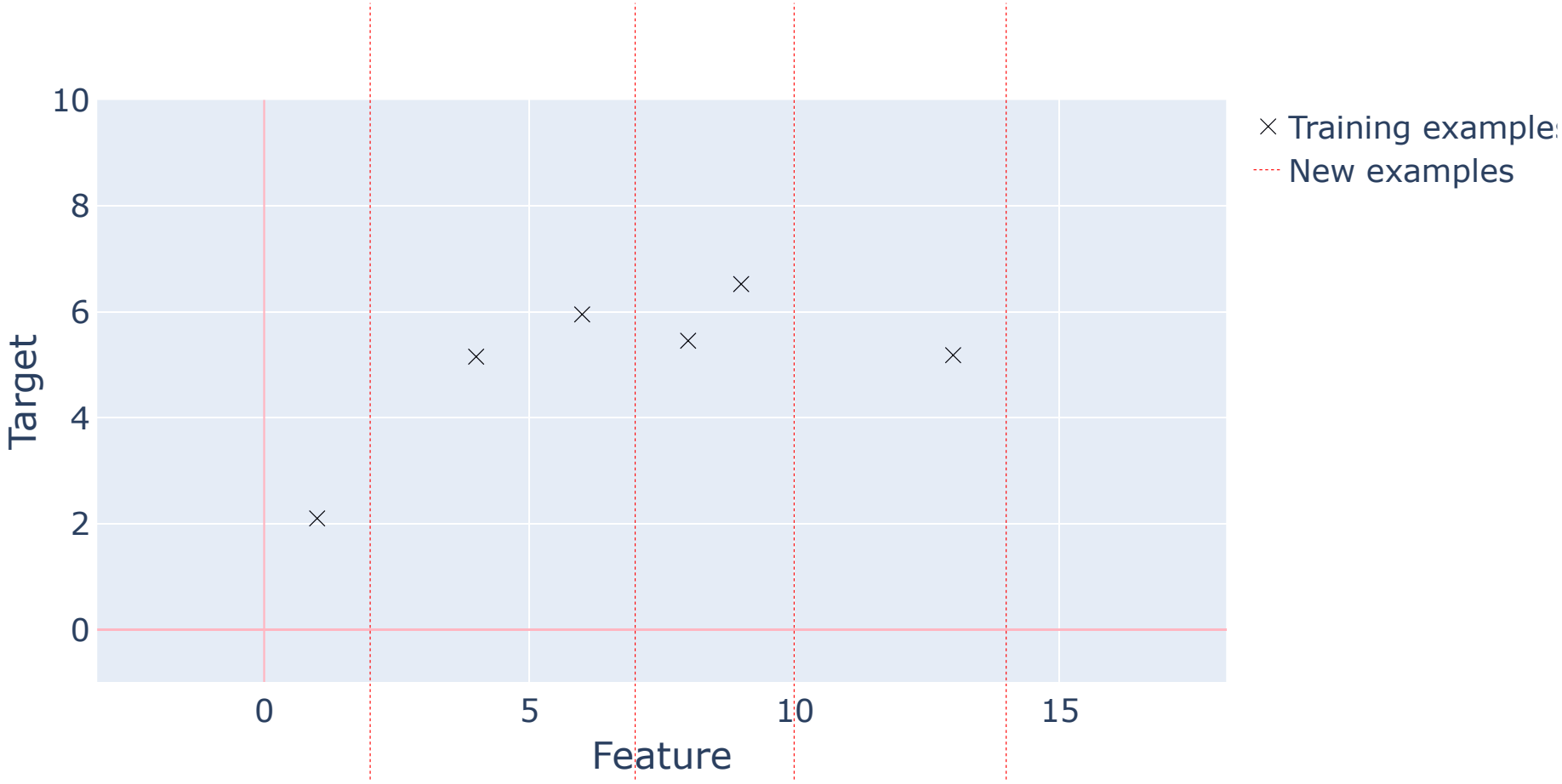
Overfitting

Minimising a loss function is not the goal

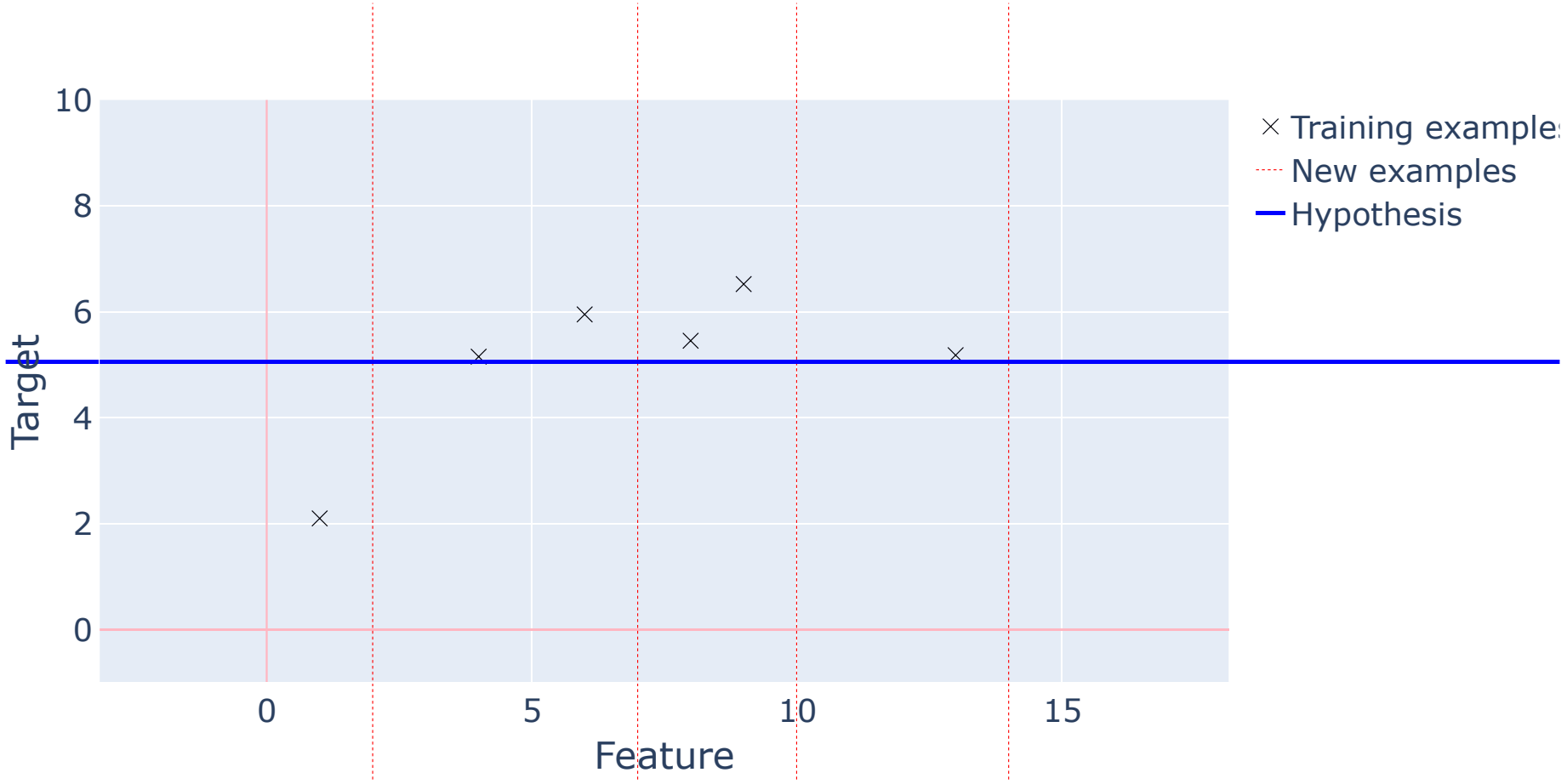
What features should we use?



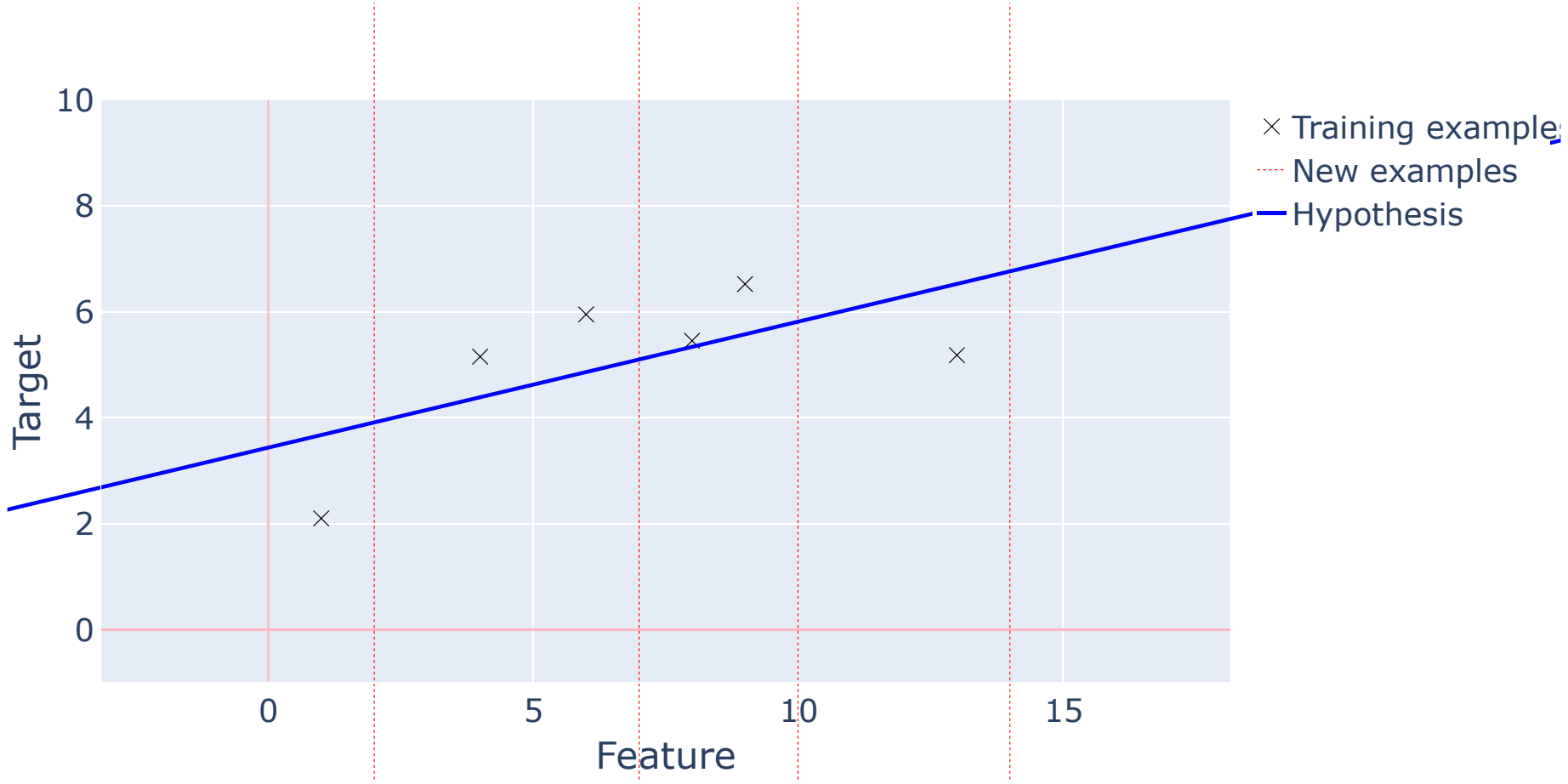
What features should we use?



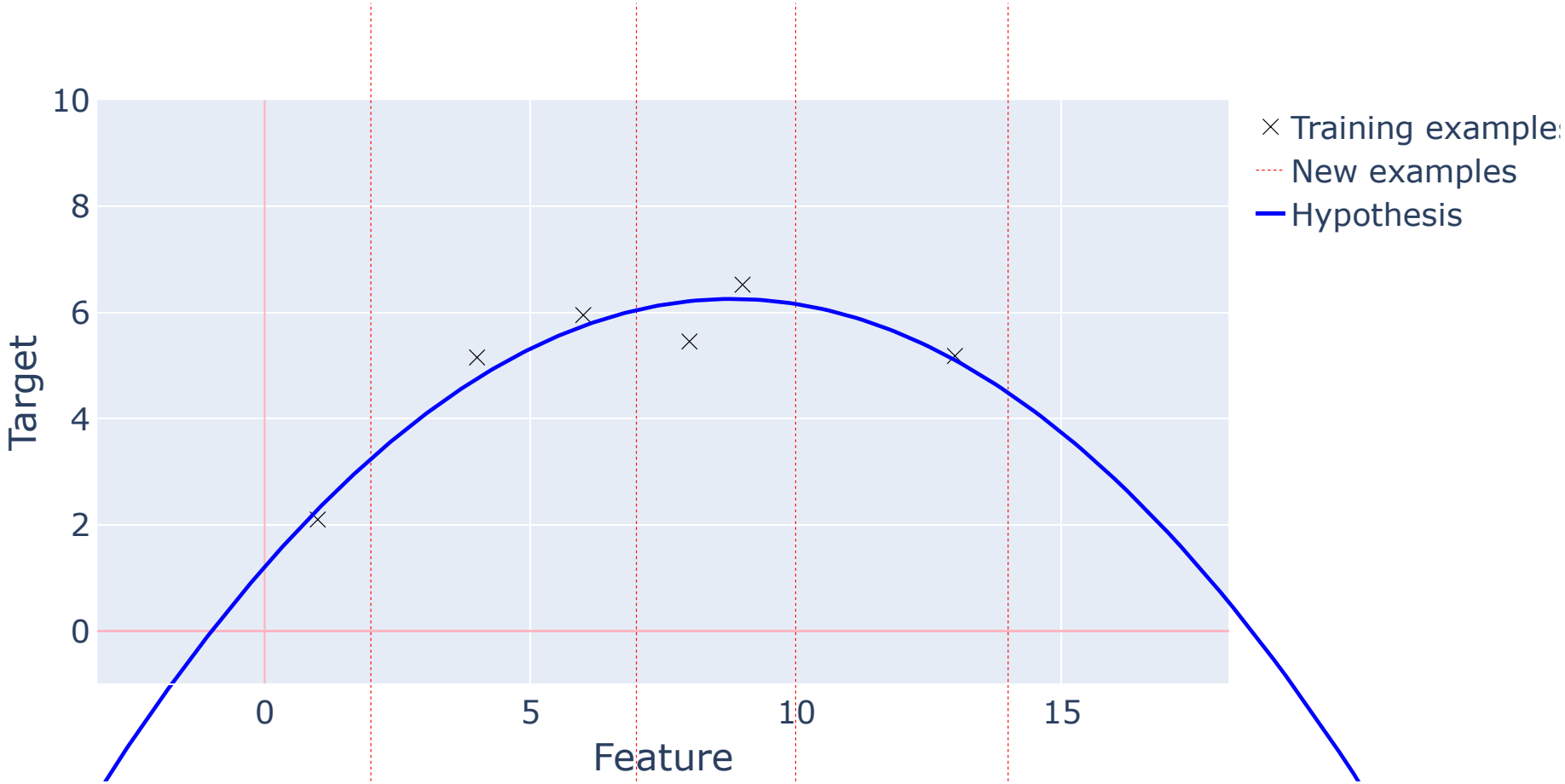
Using 1


10
8
6
4
2
0
Target
0
5
10
15
Feature
Training example
New examples
Hypothesis

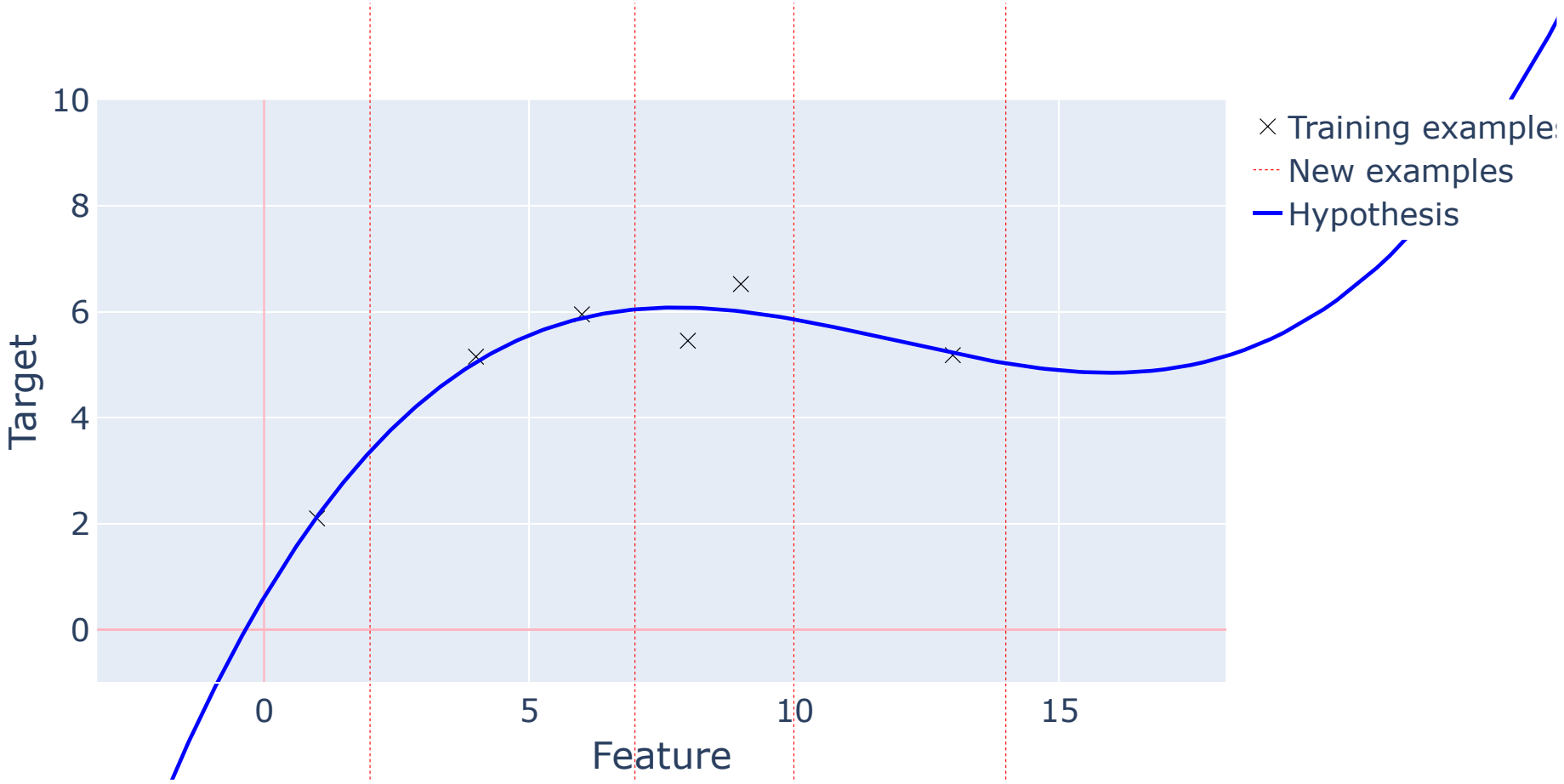
Using $1, x$


10
8
6
4
2
0
Target
0
5
10
15
Feature
Training example
New examples
Hypothesis

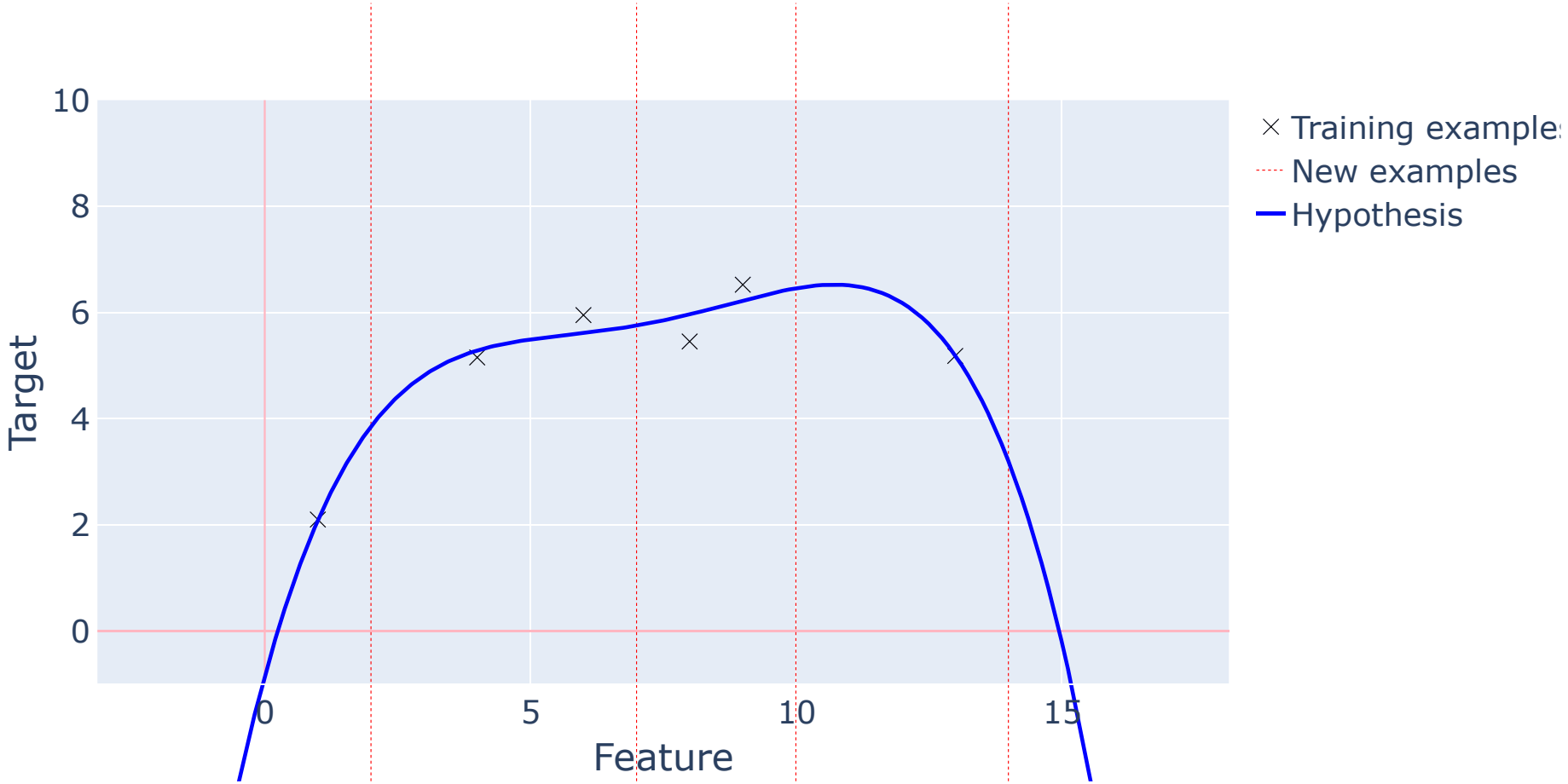
Using $1, x, x^2$


10
8
6
4
2
0
Target
0
5
10
15
Feature
Training example
New examples
Hypothesis

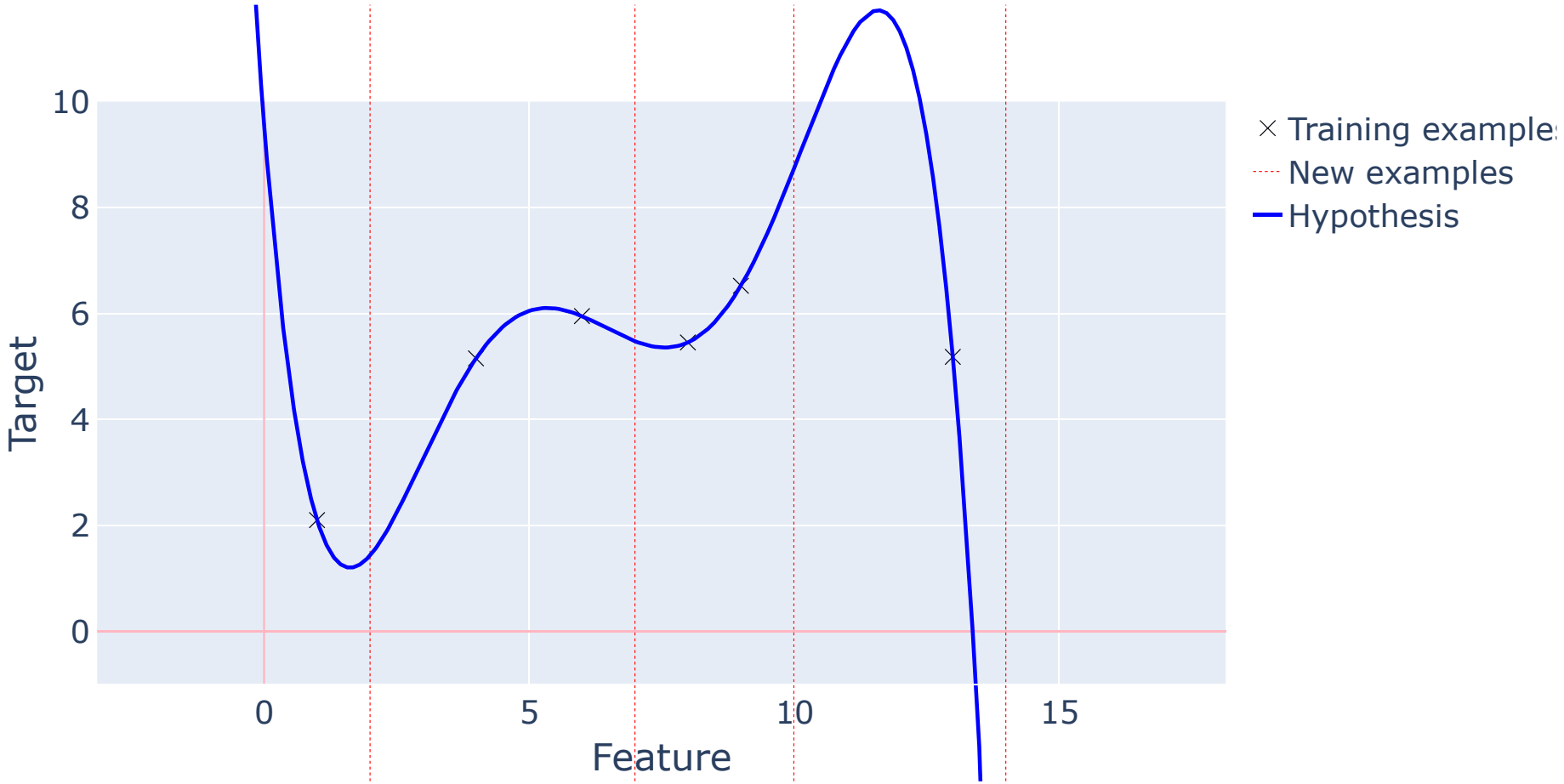
Using $1, x, x^2, x^3$



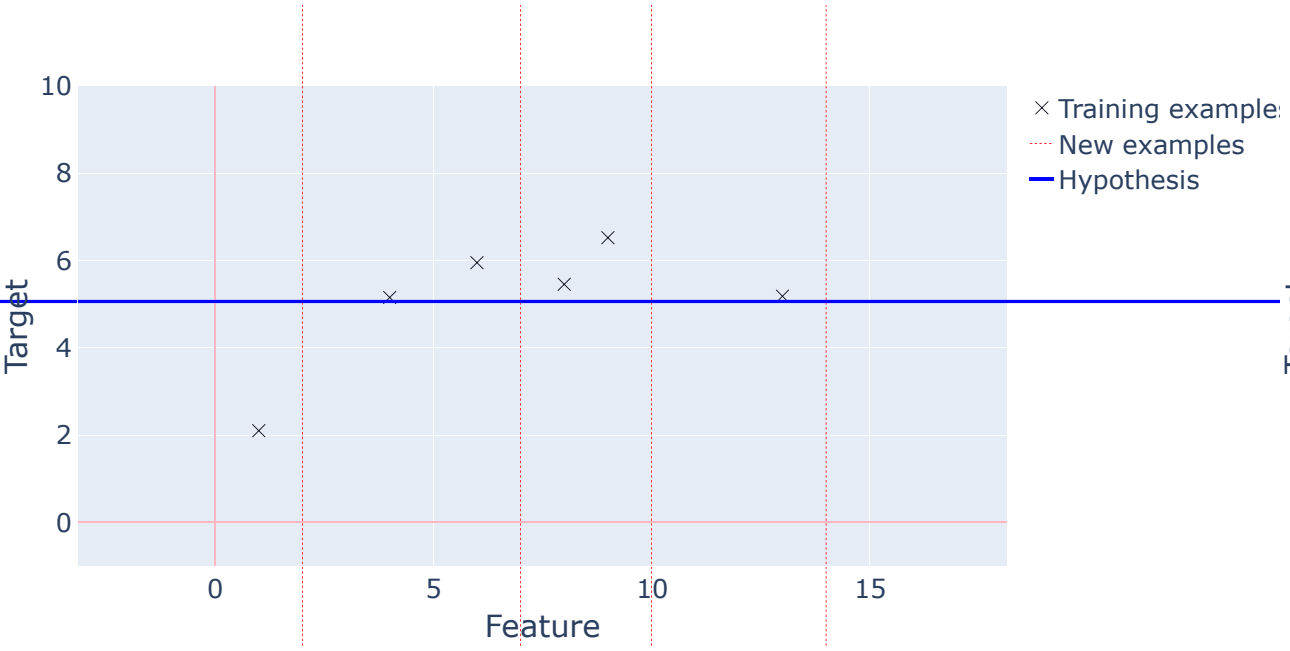
Using $1, x, x^2, x^3, x^4$

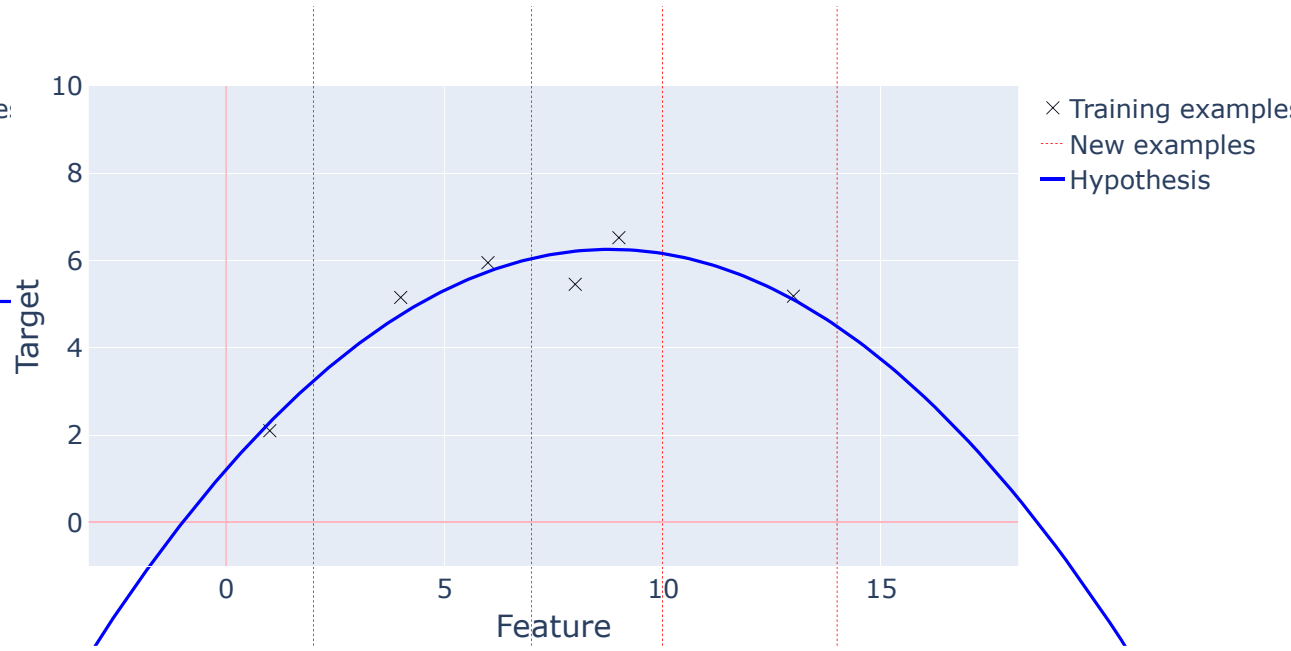


Using $1, x, x^2, x^3, x^4, x^5$

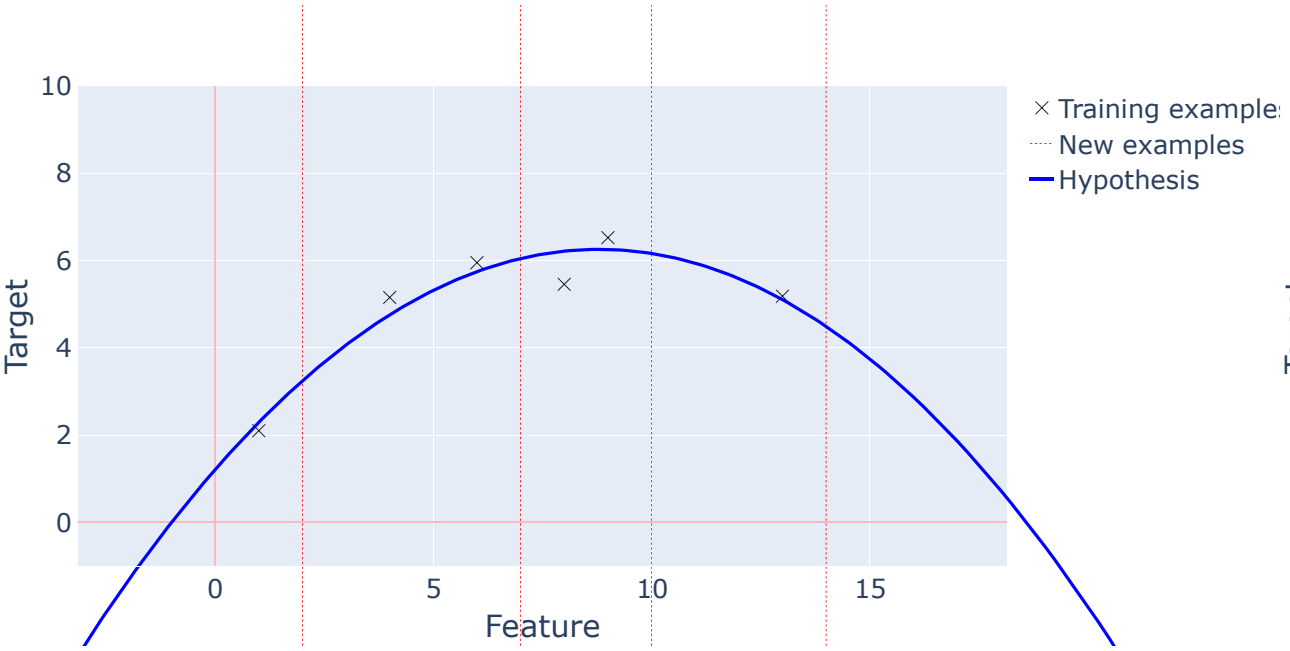


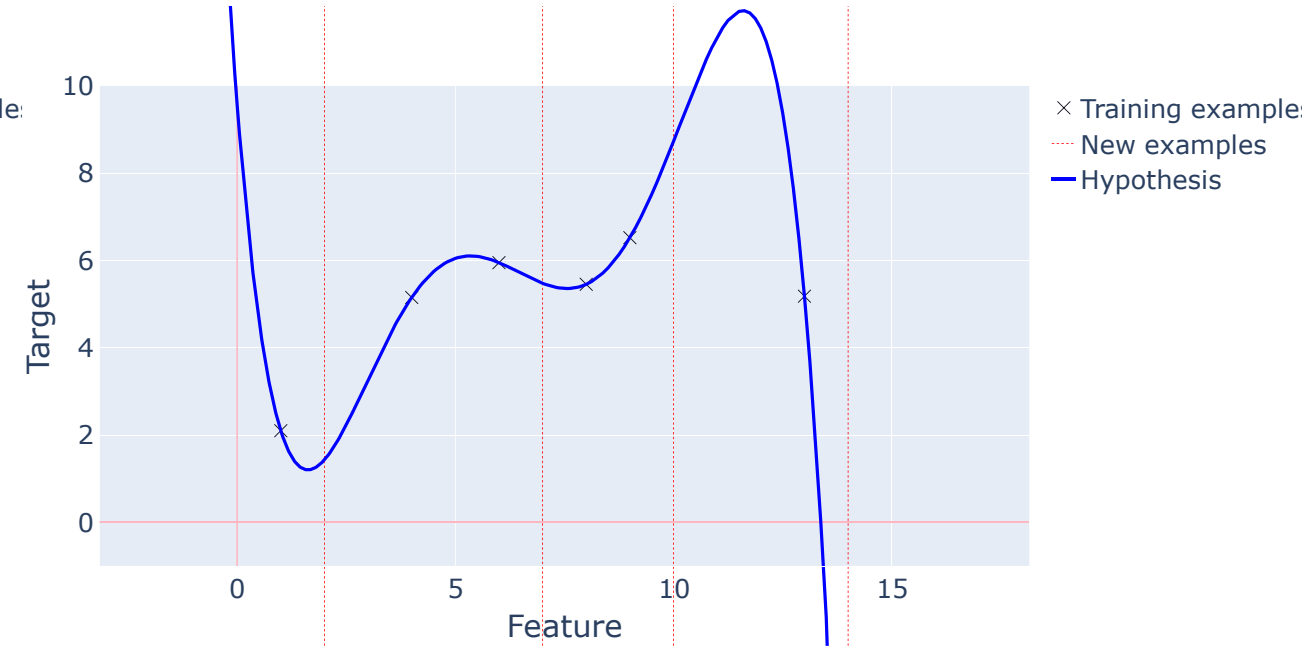
Which is better?



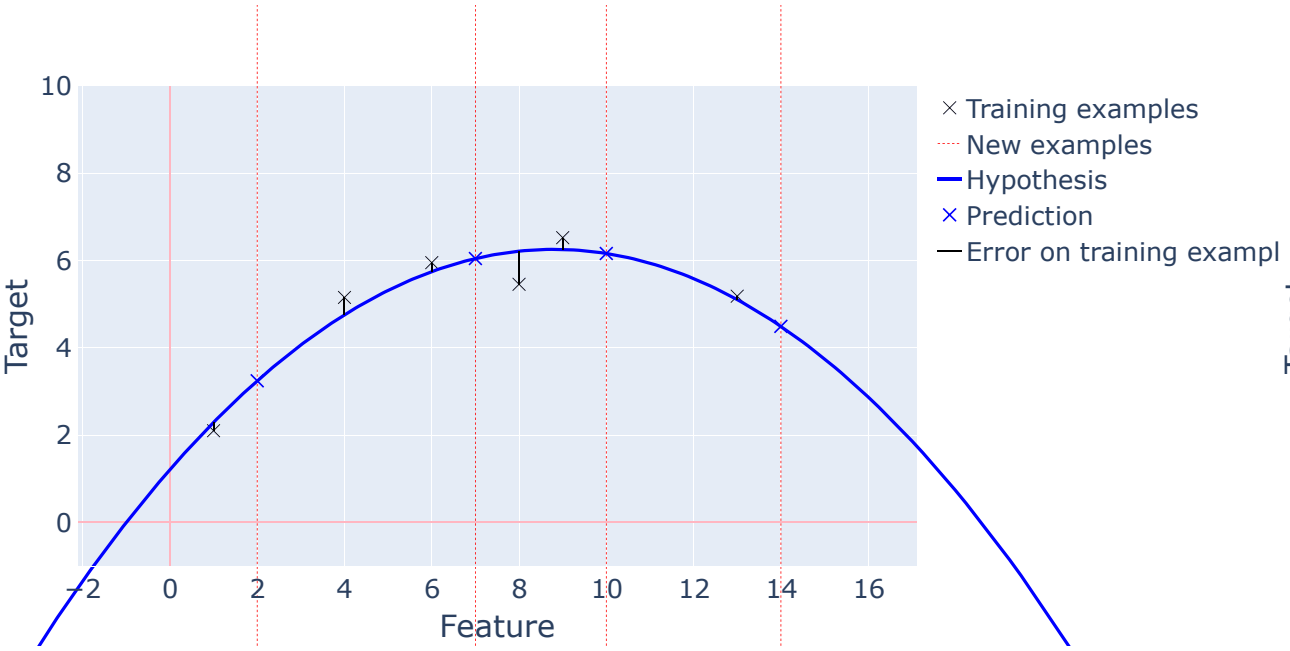


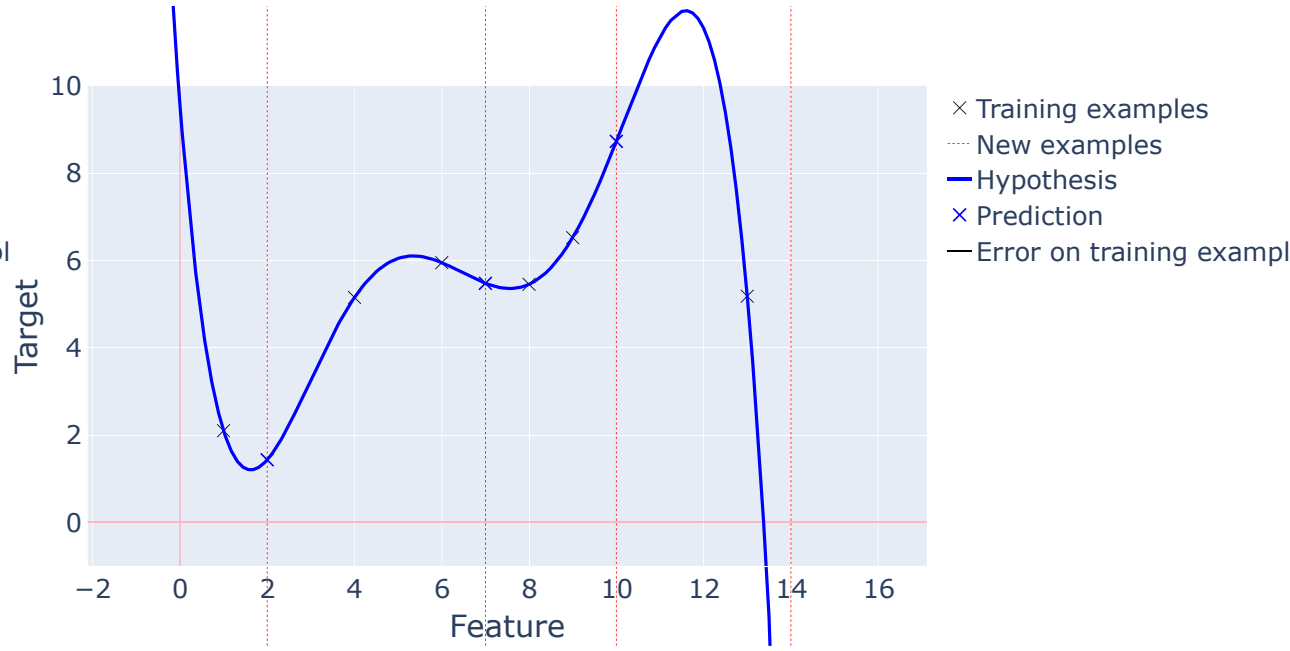
Which is better?



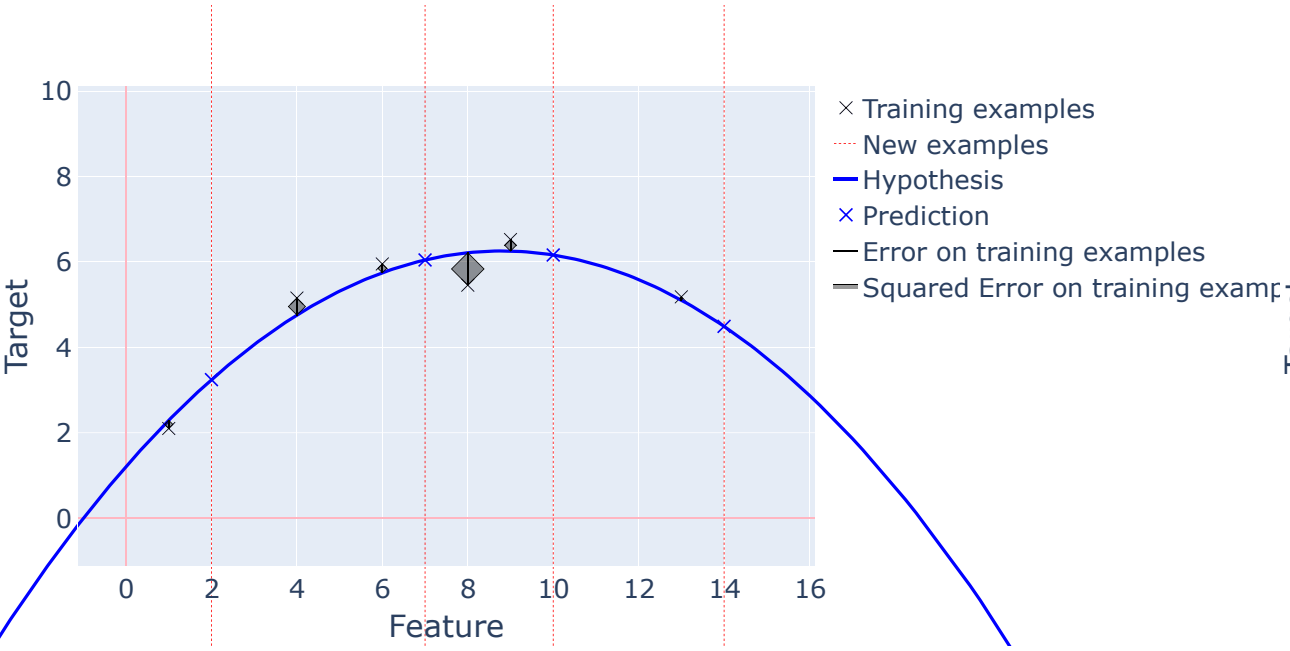


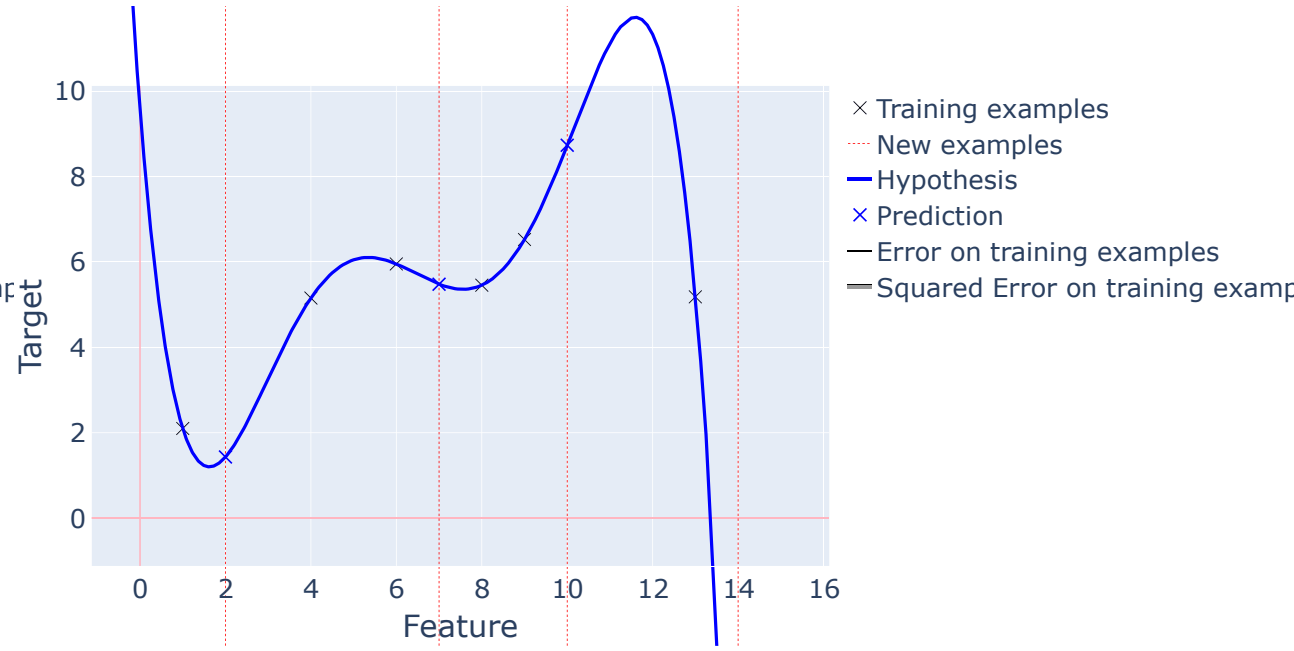
According to training error…



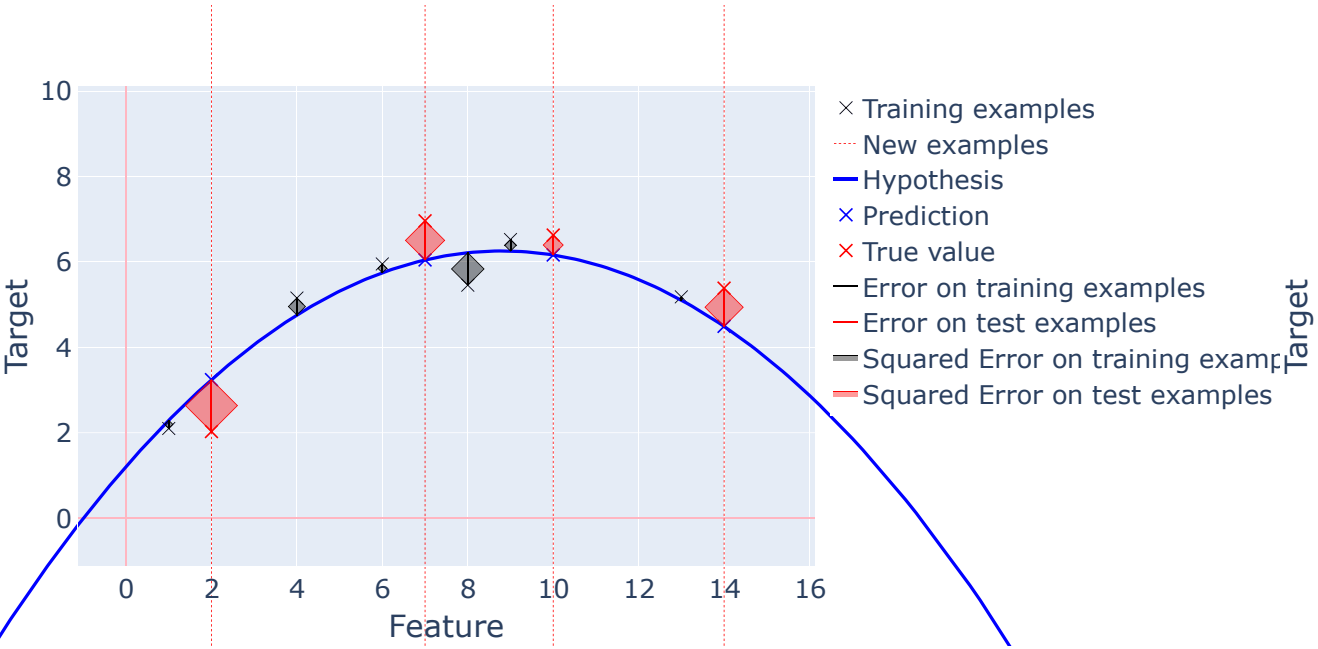


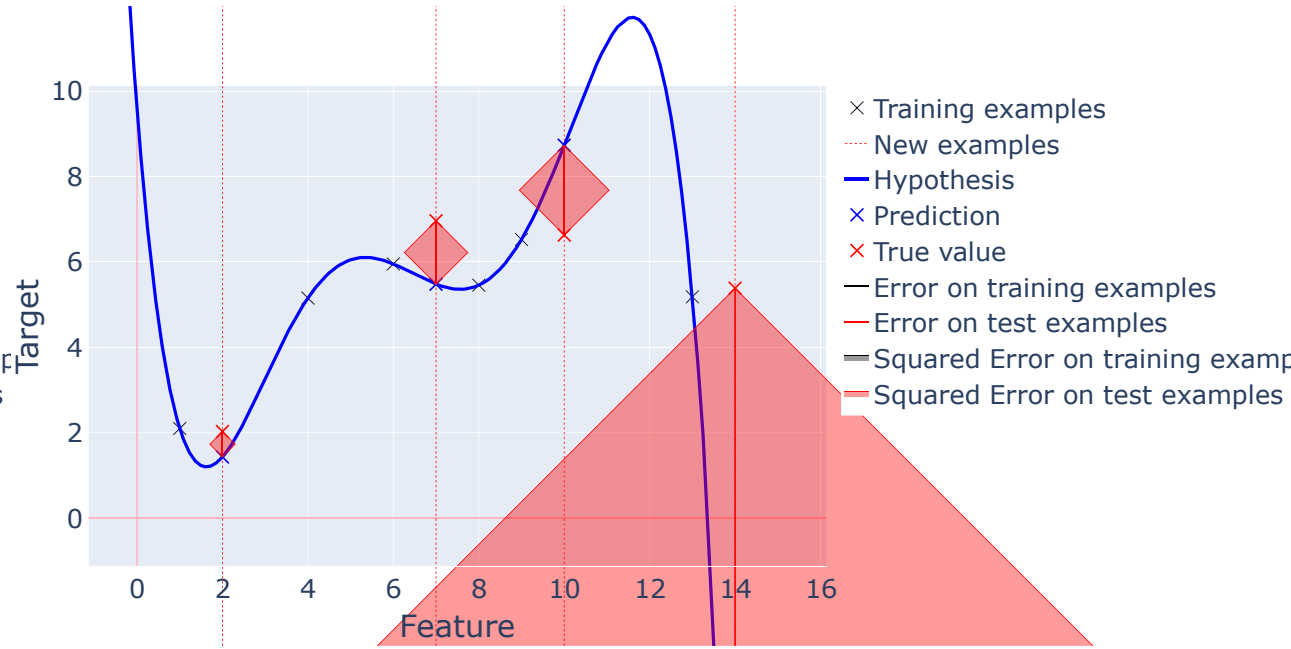
According to training error…





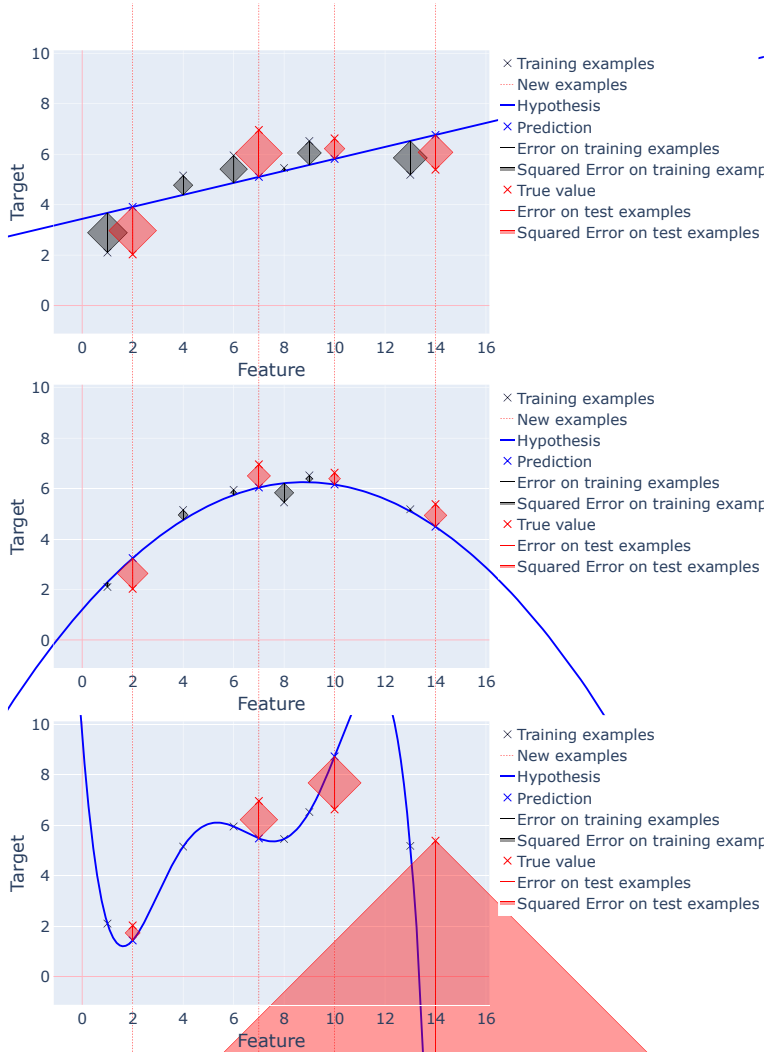
Error for new data





Underfitting and overfitting

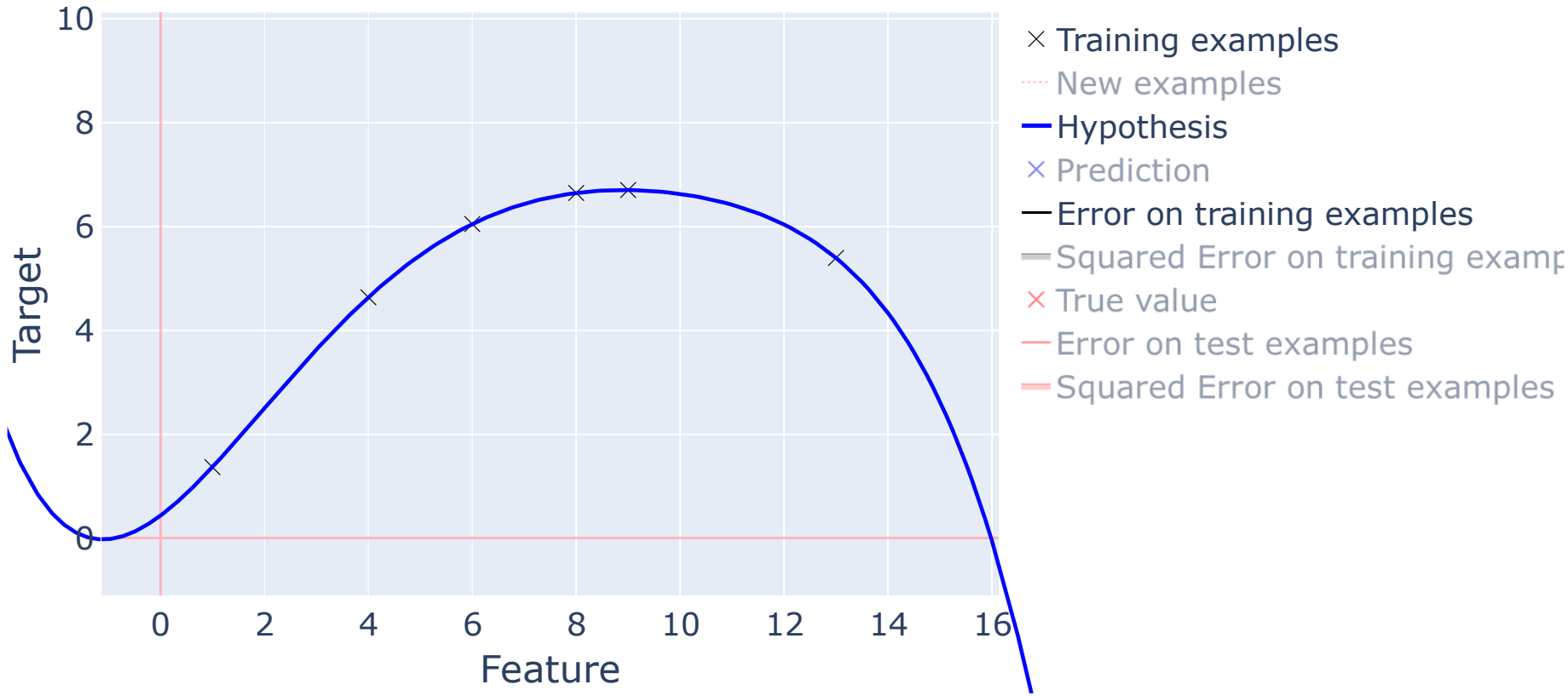
- Underfitting: # features – too few
  - The learned hypothesis…
    - Does not fit the training examples well
    - Fails to predict for new examples well
- Ideal: # features – appropriate
  - The learned hypothesis…
    - Fits the training examples well
    - predicts for new examples well
- Overfitting: # features – too many
  - The learned hypothesis…
    - Fits the training examples extremely well
    - Fails to predicts for new examples well



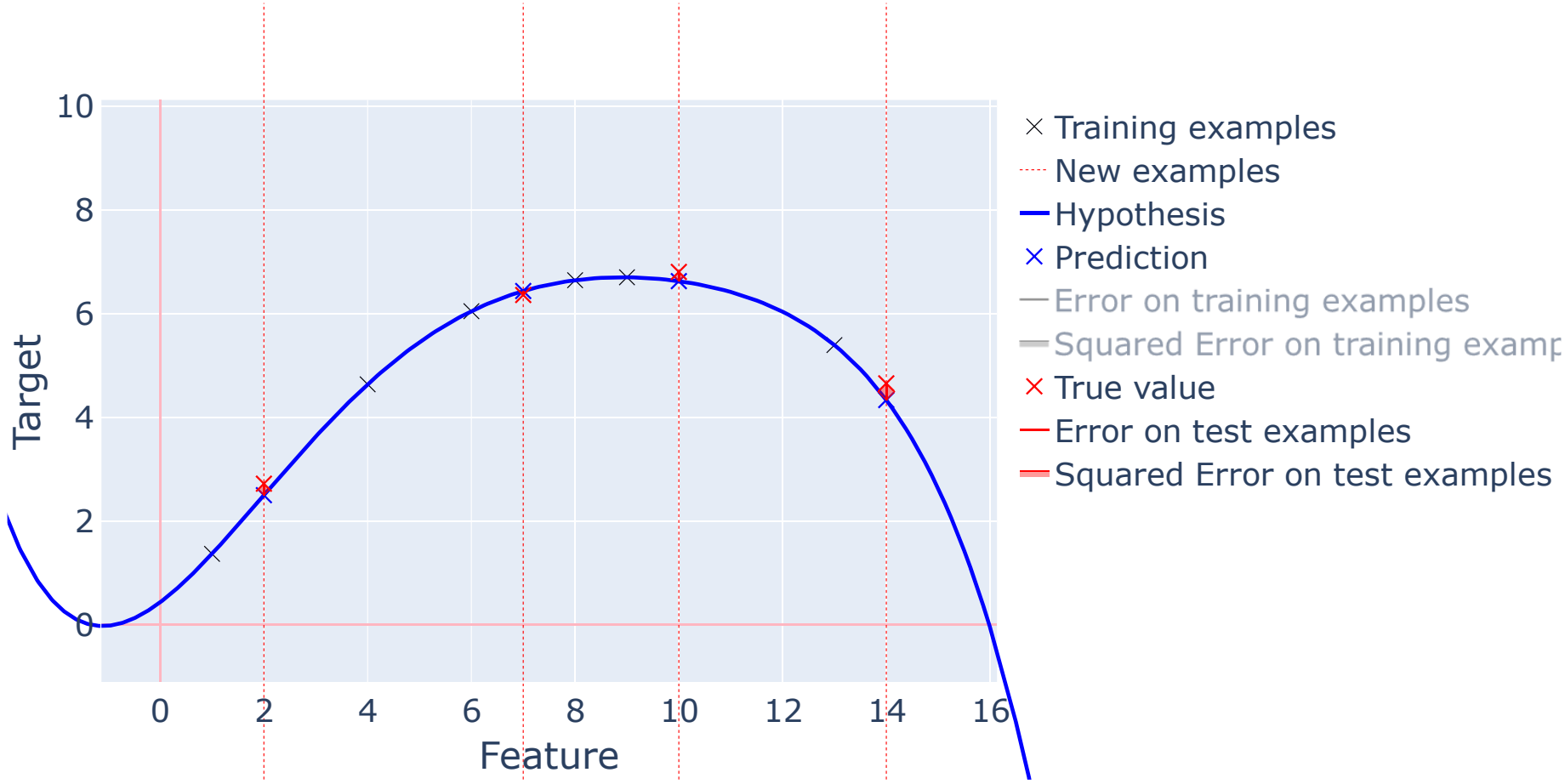
Why overfitting happens?

- Data have some randomness and noise
- Overfitting is caused by naïve fit to the noise

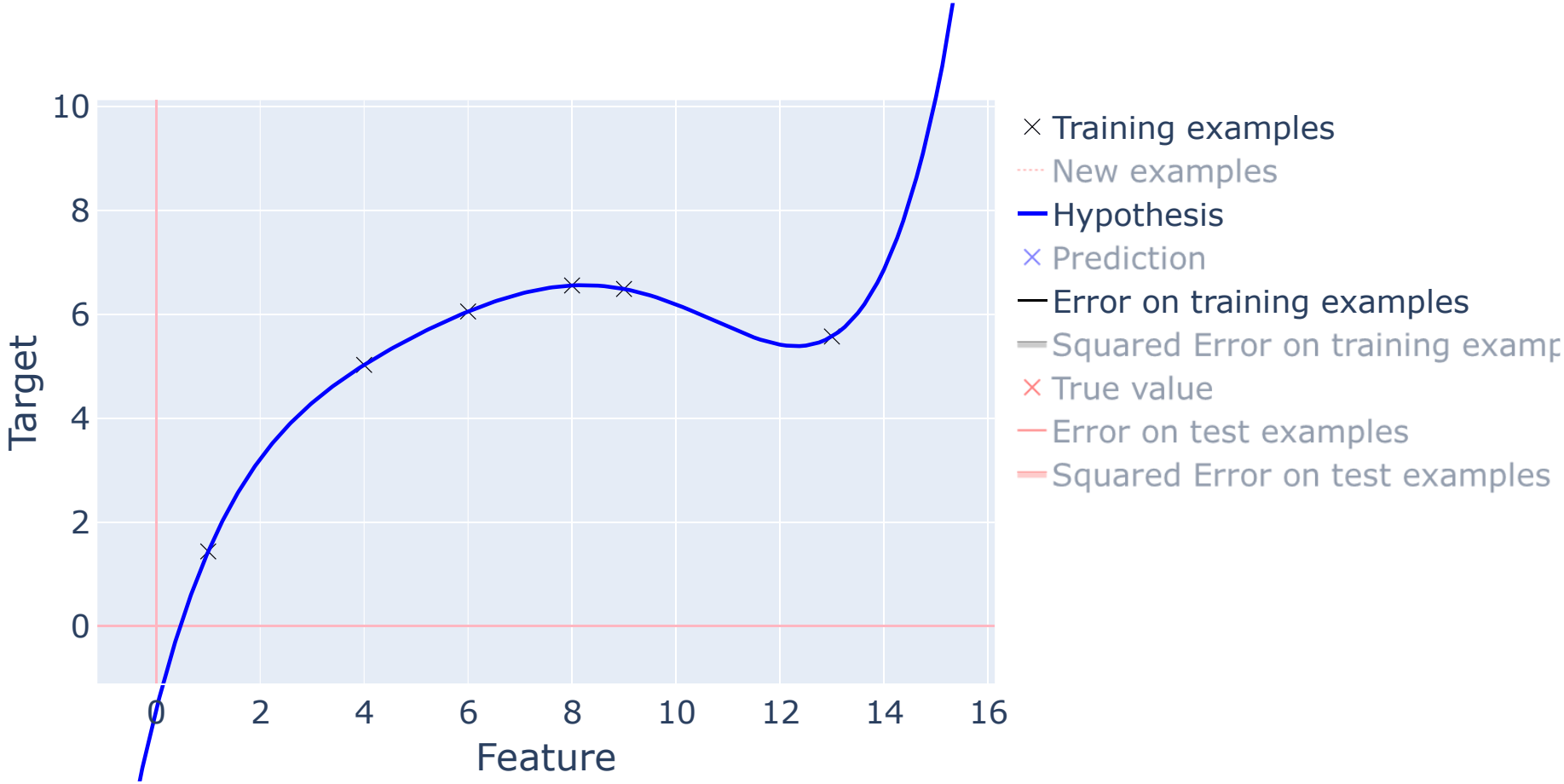
Noise causes overfitting
Example: a small noise case



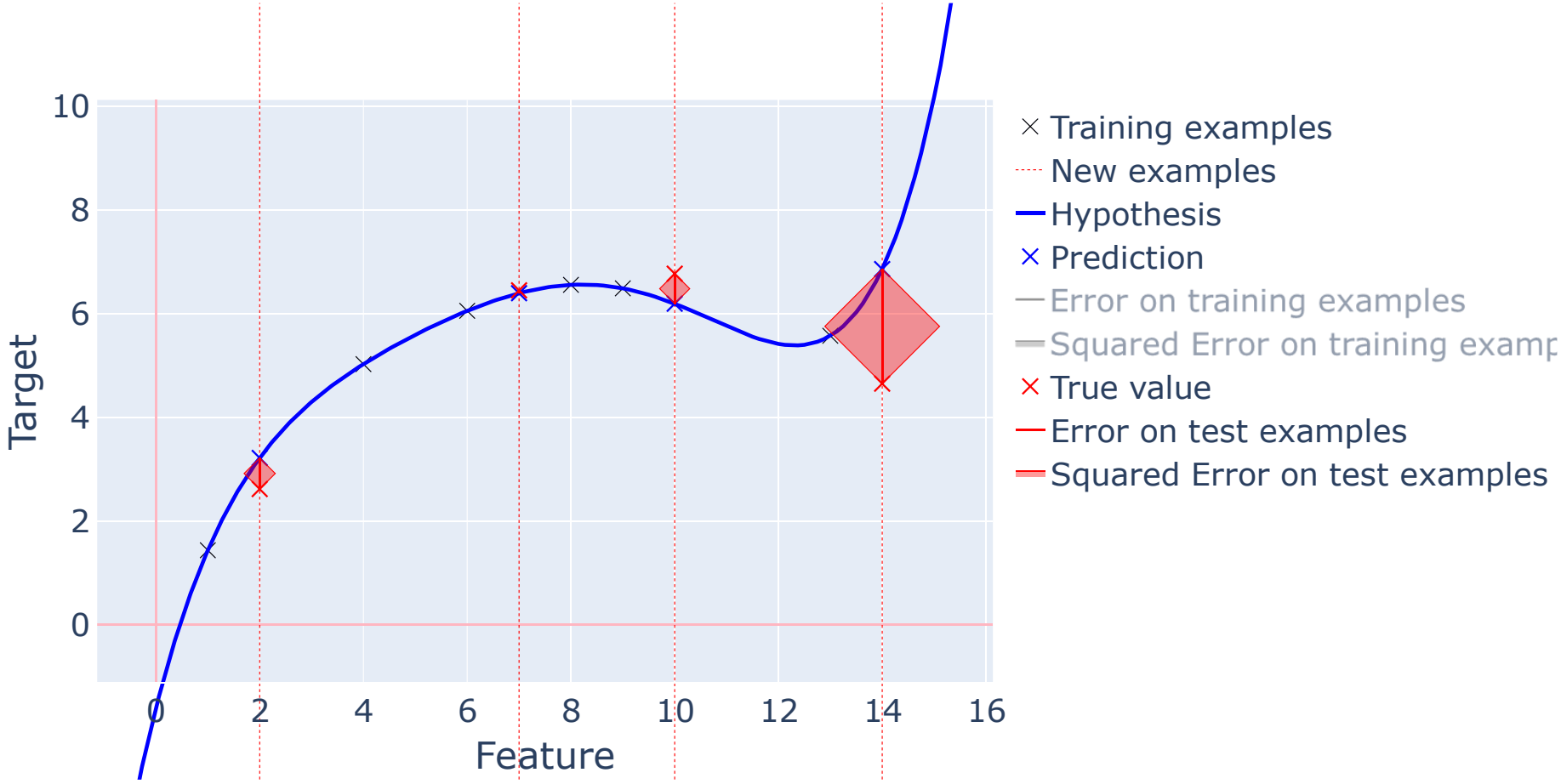
Noise causes overfitting
Example: a small noise case



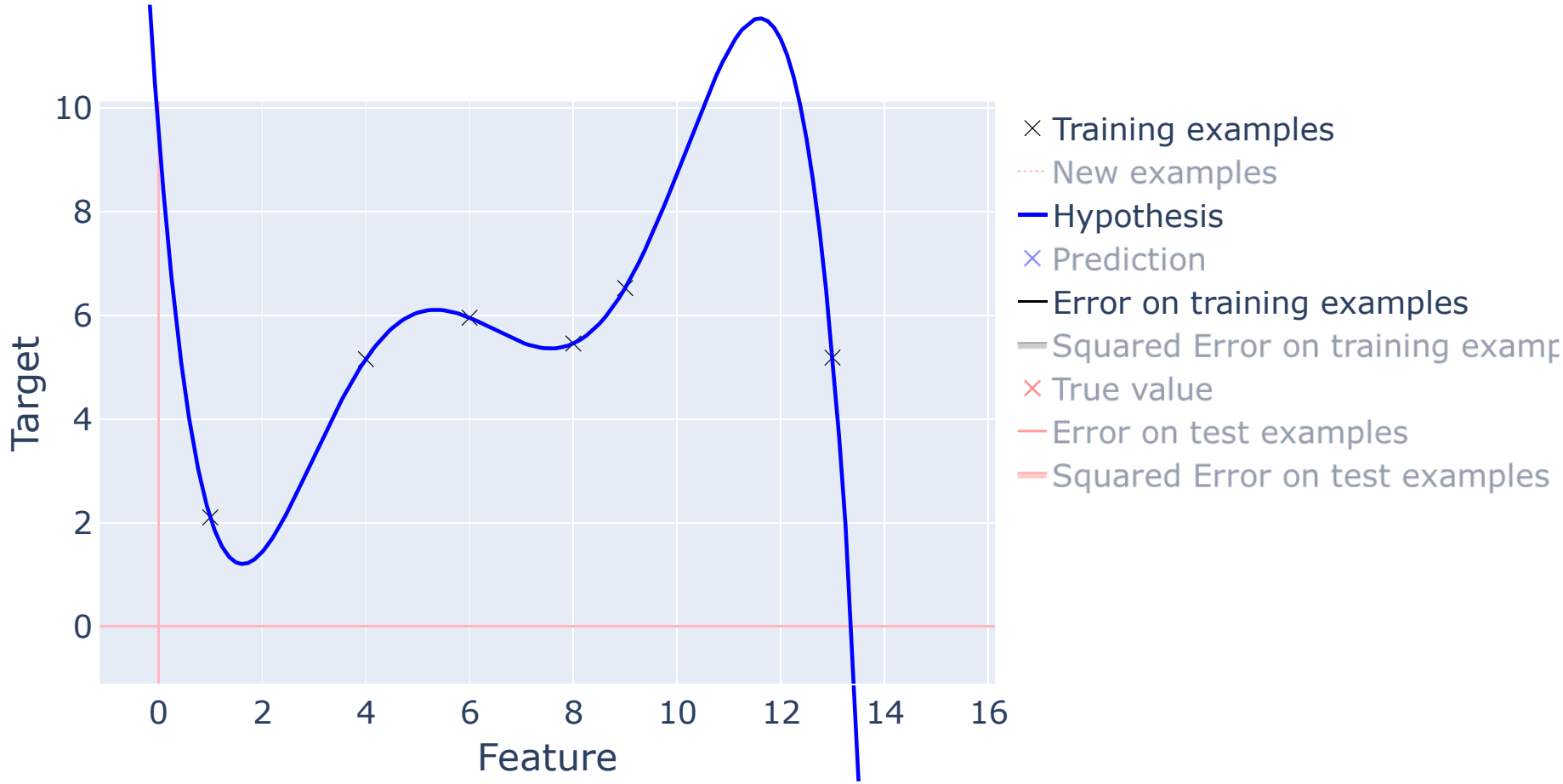
Noise causes overfitting
Example: a medium noise case



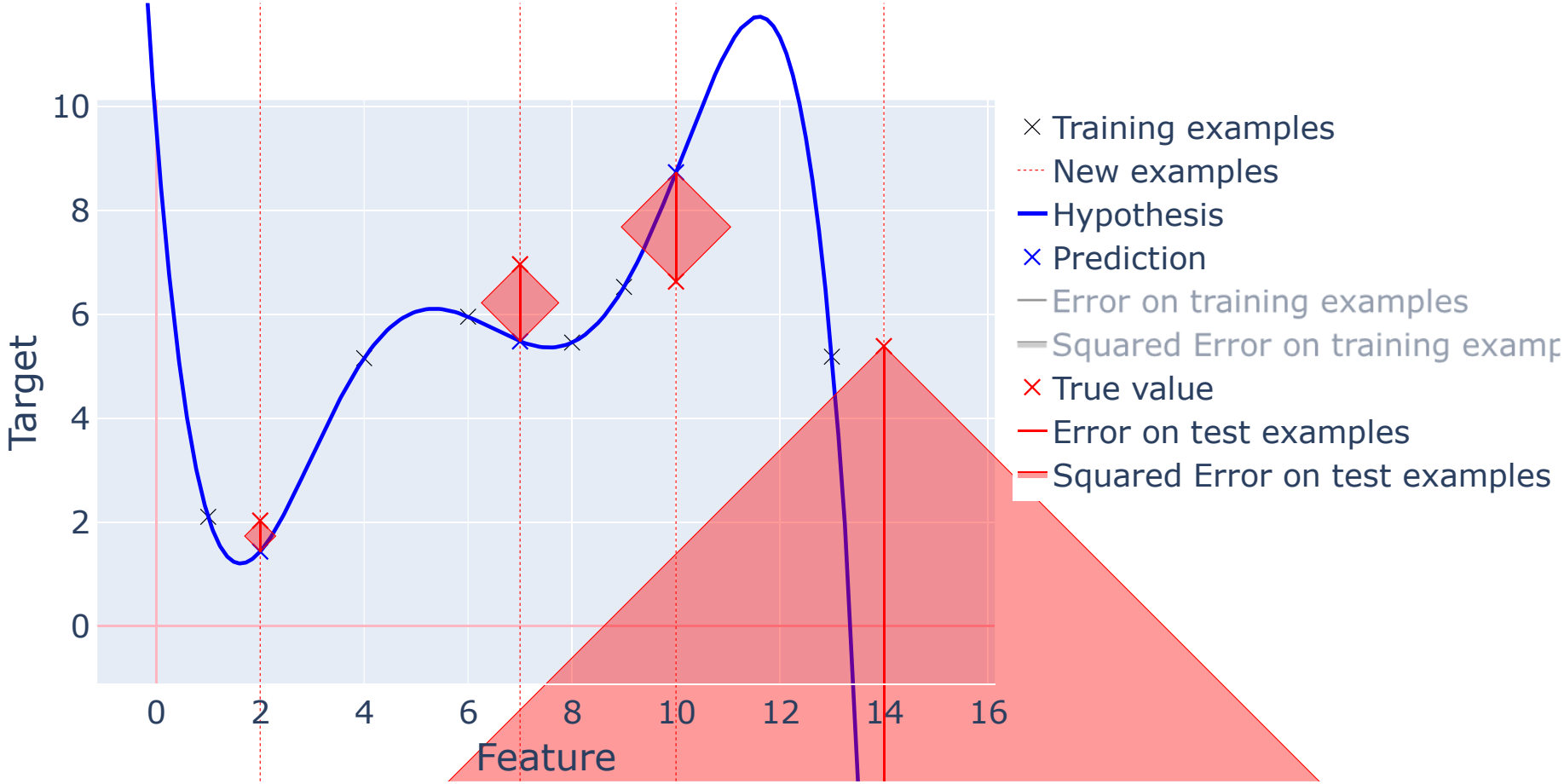
Noise causes overfitting
Example: a medium noise case



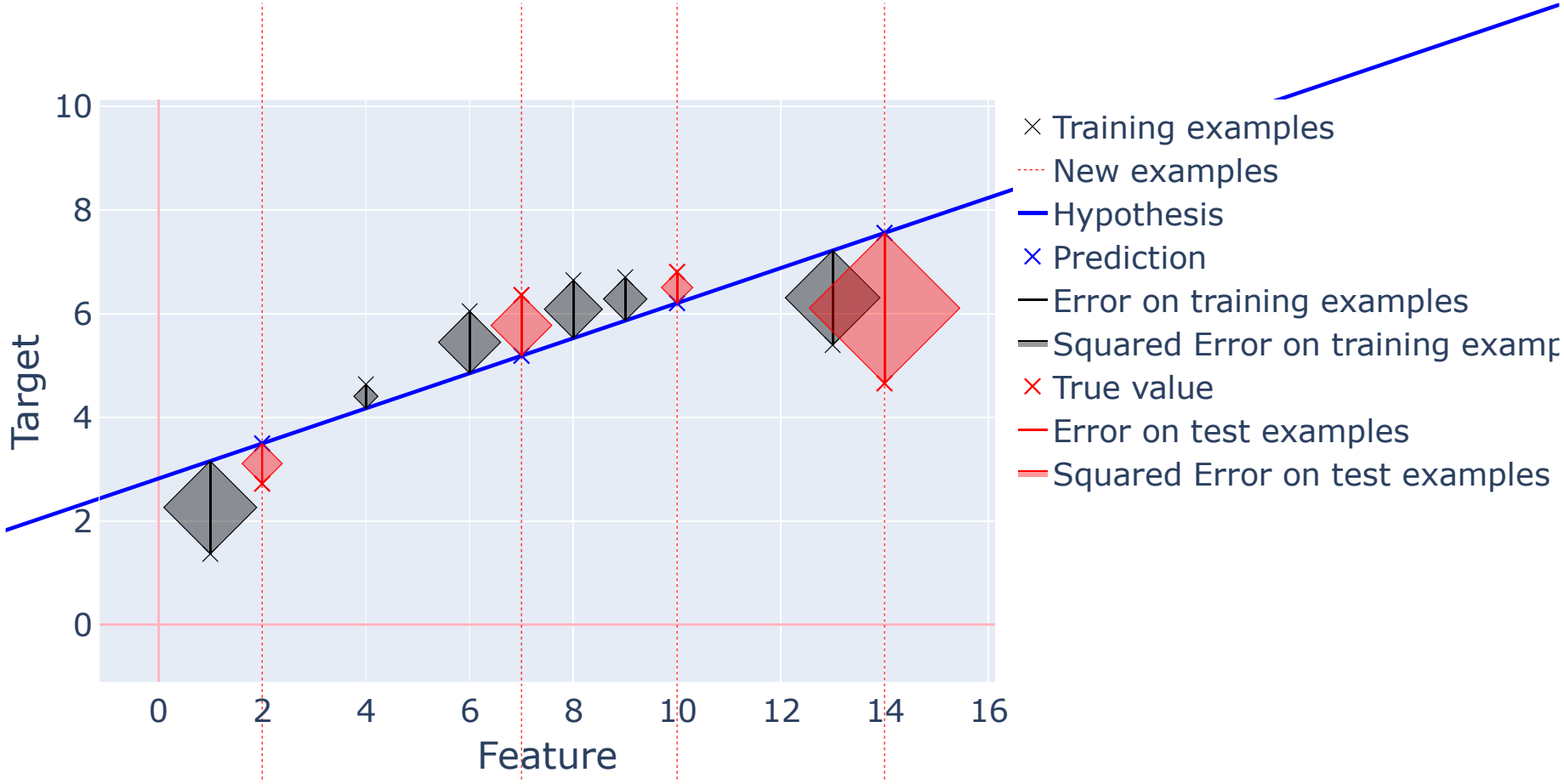
Noise causes overfitting
Example: a large noise case



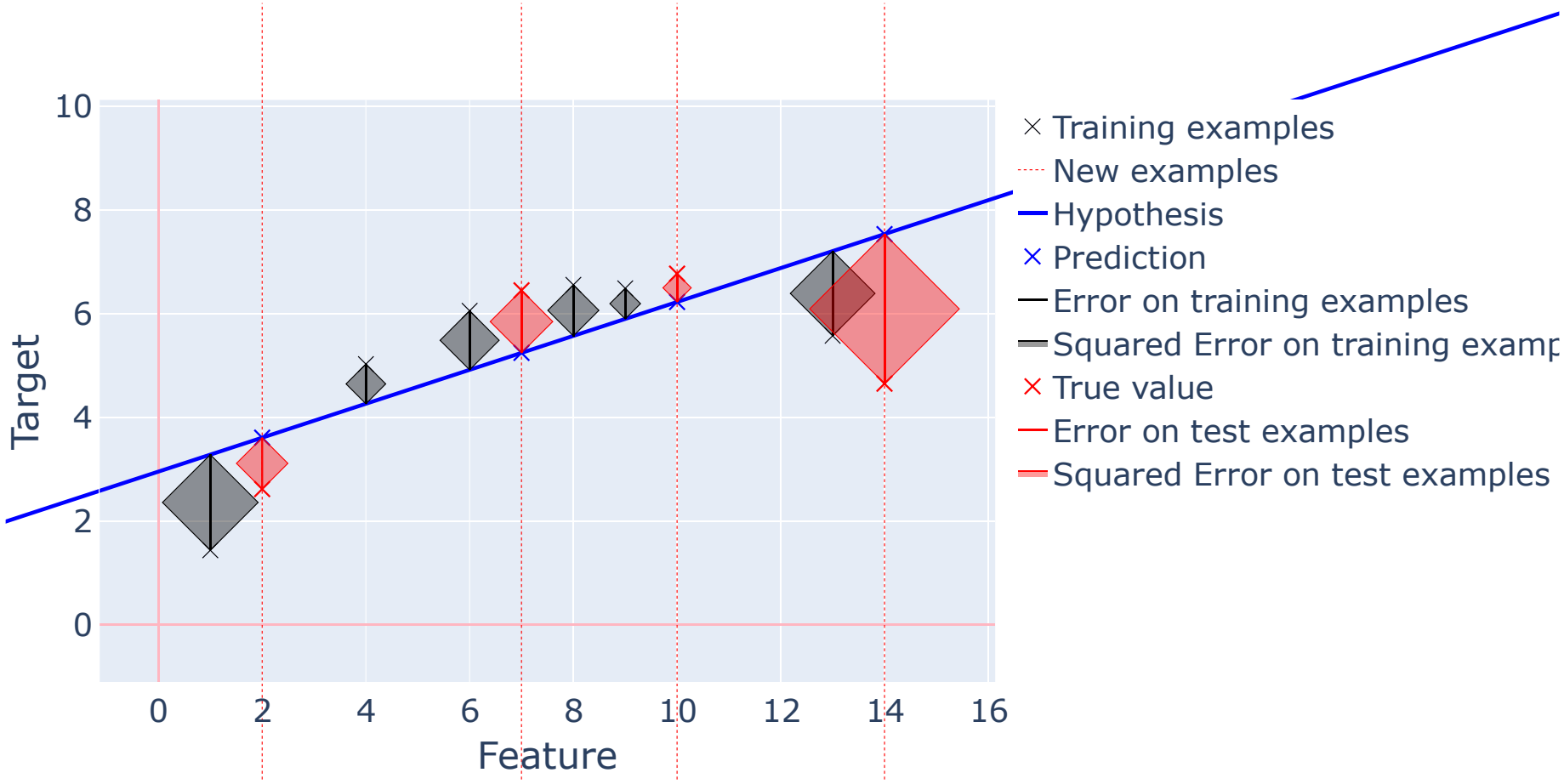
Noise causes overfitting
Example: a large noise case



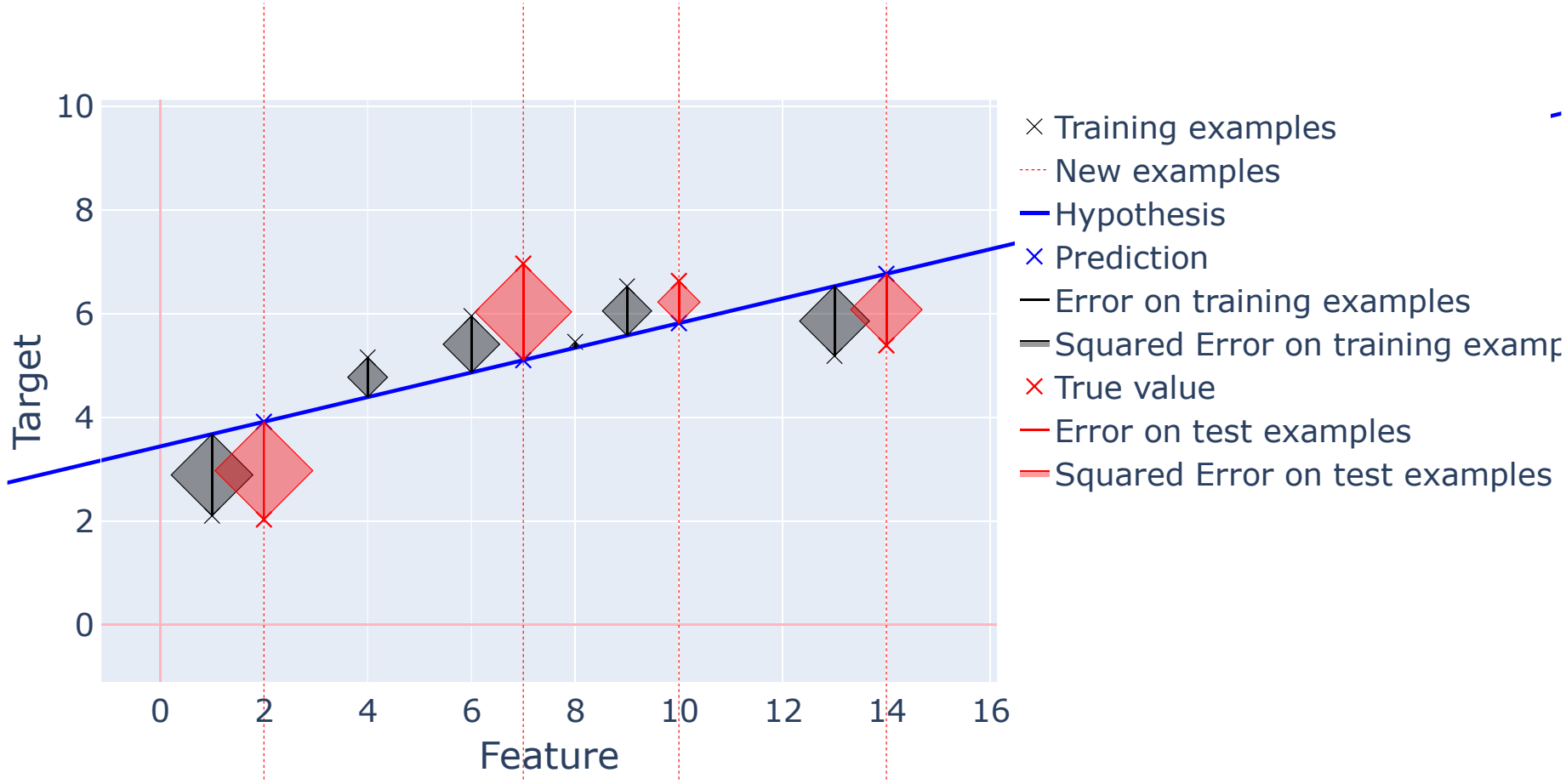
The simpler, the more robust



The simpler, the more robust



The simpler, the more robust



Validation

Overfitting is a fatal problem. First, we need to know whether it is happening or not.

Evaluating true performance

- The loss function on examples used in training does not reflect the true performance of our hypothesis.
- We need to evaluate the performance on examples NOT used in training.
- To calculate loss function, we need the true target values.

Data split: splitting feature-target pairs into
- Examples for training
- Examples for evaluation

Data split

$x_0^{(i)}$	$x_1^{(i)}$: MedInc	$x_2^{(i)}$: HouseAge	$x_3^{(i)}$: AveRooms	$x_4^{(i)}$: AveBedrms	$x_5^{(i)}$: Population	$y^{(i)}$: Price
1	8.33	41.00	6.98	1.02	322.00	4.53
1	3.38	29.00	4.84	1.00	1919.00	1.84
1	2.50	15.00	4.10	1.17	924.00	0.90
1	4.15	10.00	4.79	0.84	447.00	1.03
1	6.80	33.00	7.19	1.04	2041.00	3.90
1	1.72	49.00	4.49	0.99	1861.00	0.95
1	4.04	33.00	5.31	0.97	600.00	1.39
1	3.32	42.00	4.45	1.05	1269.00	1.49
1	4.60	36.00	5.24	1.03	1128.00	2.08
1	4.00	49.00	6.87	1.04	1018.00	1.97

Data split

$x_0^{(i)}$	$x_1^{(i)}$: MedInc	$x_2^{(i)}$: HouseAge	$x_3^{(i)}$: AveRooms	$x_4^{(i)}$: AveBedrms	$x_5^{(i)}$: Population	$y^{(i)}$: Price
1	8.33	41.00	6.98	1.02	322.00	4.53
1	3.38	29.00	4.84	1.00	1919.00	1.84
1	2.50	15.00	4.10	1.17	924.00	0.90
1	4.25	37.00	5.25	1.04	447.00	1.03
1	6.80	33.00	7.19	1.04	2041.00	3.90
1	1.72	49.00	4.49	0.99	1861.00	0.95
1	4.04	33.00	5.31	0.97	600.00	1.39
1	3.32	42.00	4.45	1.05	1269.00	1.49
1	4.00	37.00	5.24	1.03	1128.00	2.08
1	4.00	49.00	6.87	1.04	1018.00	1.97

Training data:

The examples on which we optimise the objective function

Test data:

The examples on which we evaluate the loss function

Evaluation on test examples

- On training examples
  - Get $\boldsymbol{\theta}^* \coloneqq \underset{\boldsymbol{\theta}}{\operatorname{argmin}} L(\boldsymbol{\theta})$
  - i.e. Optimise parameter $\boldsymbol{\theta}$
- On test examples
  - Compute $L(\boldsymbol{\theta}^*)$ as an evaluation score

Model selection

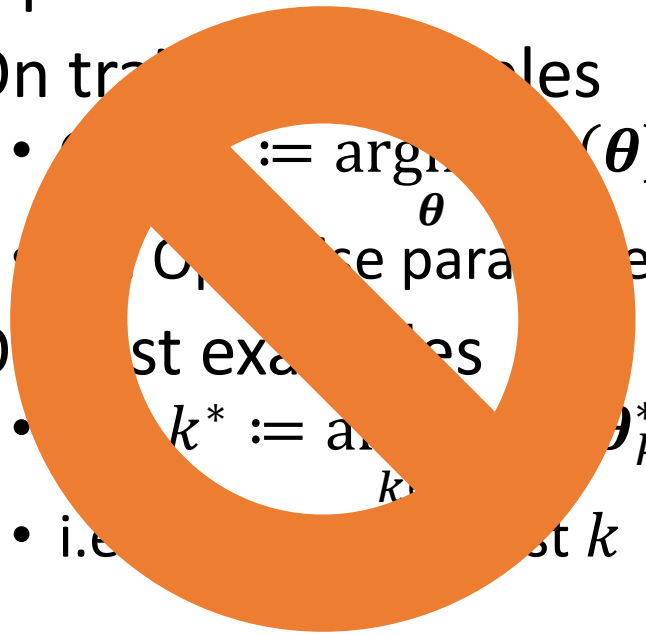
- For example, degree selection in polynomial regression.

- Input: candidate models indexed by $k \in K$ (K is the index set)
- On training examples
  - Get $\boldsymbol{\theta}_k^* \coloneqq \underset{\boldsymbol{\theta}}{\operatorname{argmin}} L(\boldsymbol{\theta})$ for $k \in K$
  - i.e. Optimise parameter $\boldsymbol{\theta}$ for each k
- On test examples
  - Get $k^* \coloneqq \underset{k \in K}{\operatorname{argmin}} L(\boldsymbol{\theta}_k^*)$
  - i.e. Choose the best k

Model selection

- For example, degree selection in polynomial regression.

- Input: candidate models indexed by $k \in K$ (K is the index set)
- On tr[illegible]les
  - [illegible] $:= \arg$[illegible]$(\boldsymbol{\theta})$ for $k \in K$
  - O[illegible]e para[illegible]er $\boldsymbol{\theta}$ for each k
- O[illegible]st exa[illegible]es
  - $k^* :=$ a[illegible]$(\boldsymbol{\theta}_k^*)$
  - i.e[illegible]st k



The selected model may overfit the test examples
We can't evaluate the true performance.

Solution: split data into three.
Training/validation/test

Model selection

- Input: candidate models indexed by $k \in K$ (K is the index set)
- On training examples
  - Get $\boldsymbol{\theta}_k^* \coloneqq \underset{\boldsymbol{\theta}}{\operatorname{argmin}} L(\boldsymbol{\theta})$ for $k \in K$
  - i.e. Optimise parameter $\boldsymbol{\theta}$ for each k
- On validation examples
  - Get $k^* \coloneqq \underset{k \in K}{\operatorname{argmin}} L(\boldsymbol{\theta}_k^*)$
  - i.e. Choose the best k
- On test examples
  - Compute $L(\boldsymbol{\theta}_{k^*}^*)$ as an evaluation score

Addressing overfitting

- Reduce # features
  - Manually remove the features that does not seem to contribute to the prediction
  - Model selection (validation, information criterion)
    - If we have 1000 features and need to select 10 features, $\binom{1000}{10} = 2 \times 10^{23}$
- Regularisation
  - Keep all the features
  - Automatically reduce the magnitude of parameters so that the hypothesis selection is robust to the change of training examples

Regularisation

Basic idea:

- Overfitting is caused by fitting subtle noises in training examples
- Idea: hypothesis selection robust to the change of a training example
- New cost function

$$J(\boldsymbol{\theta}) = \frac{1}{2m}\left[\|h_{\boldsymbol{\theta}}(\boldsymbol{X}) - \tilde{\boldsymbol{y}}\|_2^2 + \right]$$

  - Recall:

$$\|\boldsymbol{v}\|_2 \coloneqq \sqrt{\boldsymbol{v}^\top \boldsymbol{v}} = \sqrt{\sum_i (v_i)^2}, \qquad \|\boldsymbol{v}\|_2^2 \coloneqq \boldsymbol{v}^\top \boldsymbol{v} = \sum_i (v_i)^2$$

Basic idea:

- Overfitting is caused by fitting subtle noises in training examples
- Idea: hypothesis selection robust to the change of a training example
- New cost function

$$J(\boldsymbol{\theta}) = \frac{1}{2m}\left[\|h_{\boldsymbol{\theta}}(\boldsymbol{X}) - \tilde{\boldsymbol{y}}\|_2^2 + 10000\sum_{j=1}^{n}\left(\theta_j\right)^2\right]$$

  - Penalise the magnitude of $\boldsymbol{\theta}$.
  - End up with small $\boldsymbol{\theta}$ regardless of training examples' behaviour

Cost func. w/ regularisation

- Cost function with regularisation

$$J(\boldsymbol{\theta}) = \frac{1}{2m}\left[\|h_{\boldsymbol{\theta}}(\boldsymbol{X}) - \tilde{\boldsymbol{y}}\|_2^2 + \lambda \sum_{j=1}^{n} (\theta_j)^2\right]$$

$$= \frac{1}{2m}\|h_{\boldsymbol{\theta}}(\boldsymbol{X}) - \tilde{\boldsymbol{y}}\|_2^2 + \lambda \frac{1}{2m}\|\boldsymbol{\theta}\|_2^2$$

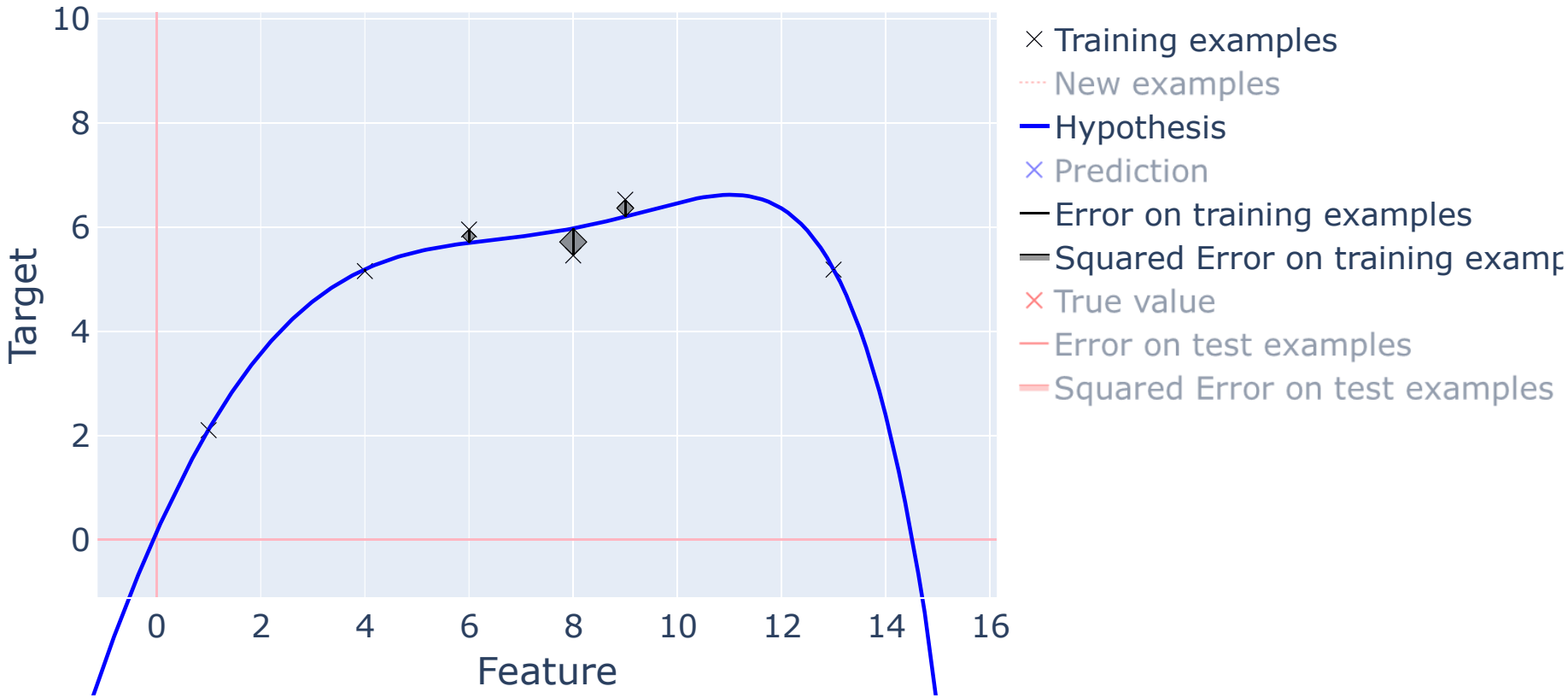
  - $\lambda > 0$: controls robustness of hypothesis selection
  - Feature scaling is important
  - $L(\boldsymbol{\theta}) \coloneqq \frac{1}{2m}\|h_{\boldsymbol{\theta}}(\boldsymbol{X}) - \tilde{\boldsymbol{y}}\|_2^2$: Loss function
  - $R(\boldsymbol{\theta}) \coloneqq \frac{1}{2m}\|\boldsymbol{\theta}\|_2^2$: Regularisation term

Loss function and regularisation term

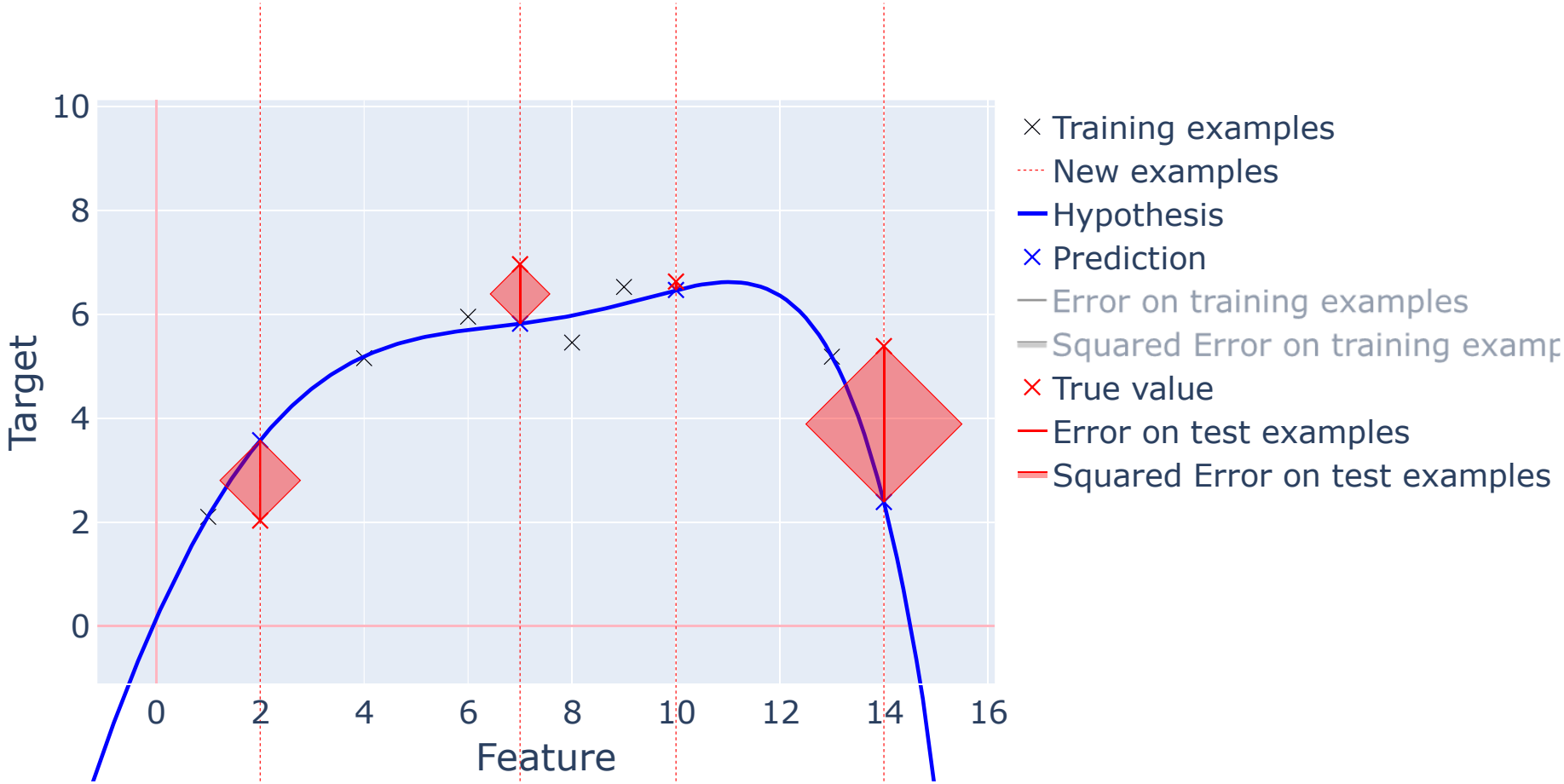
$$J(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \lambda R(\boldsymbol{\theta})$$

- Loss function $L(\boldsymbol{\theta})$:
  How well the hypothesis fits the data
- Regularisation term $R(\boldsymbol{\theta})$:
  How much influence the training examples have
- Regularisation makes hypothesis selection robust to the change of training examples
- Too strong regularisation (too large λ) may discard necessary information in the training examples

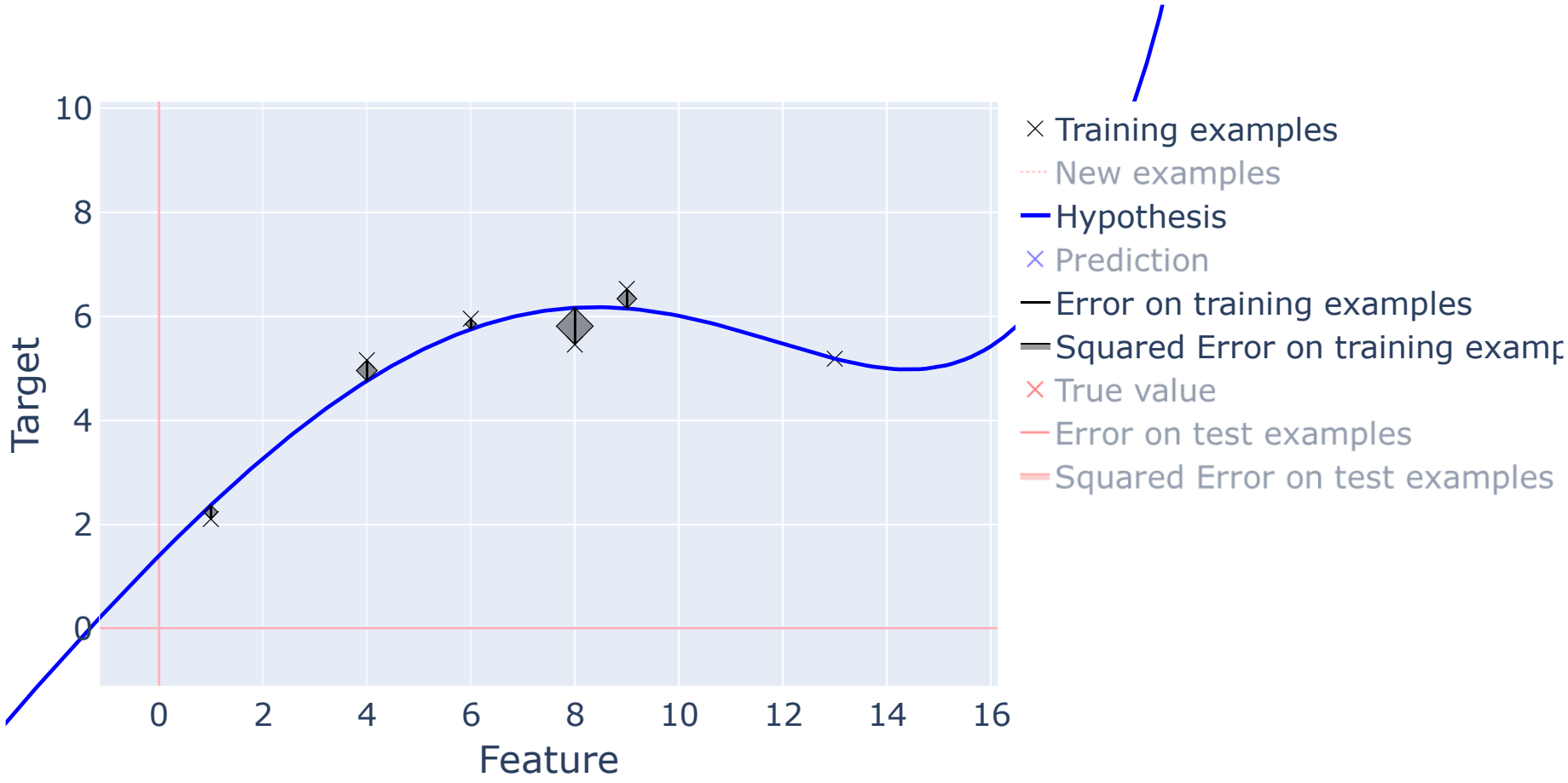
Varying λ: $\lambda = 0.001$



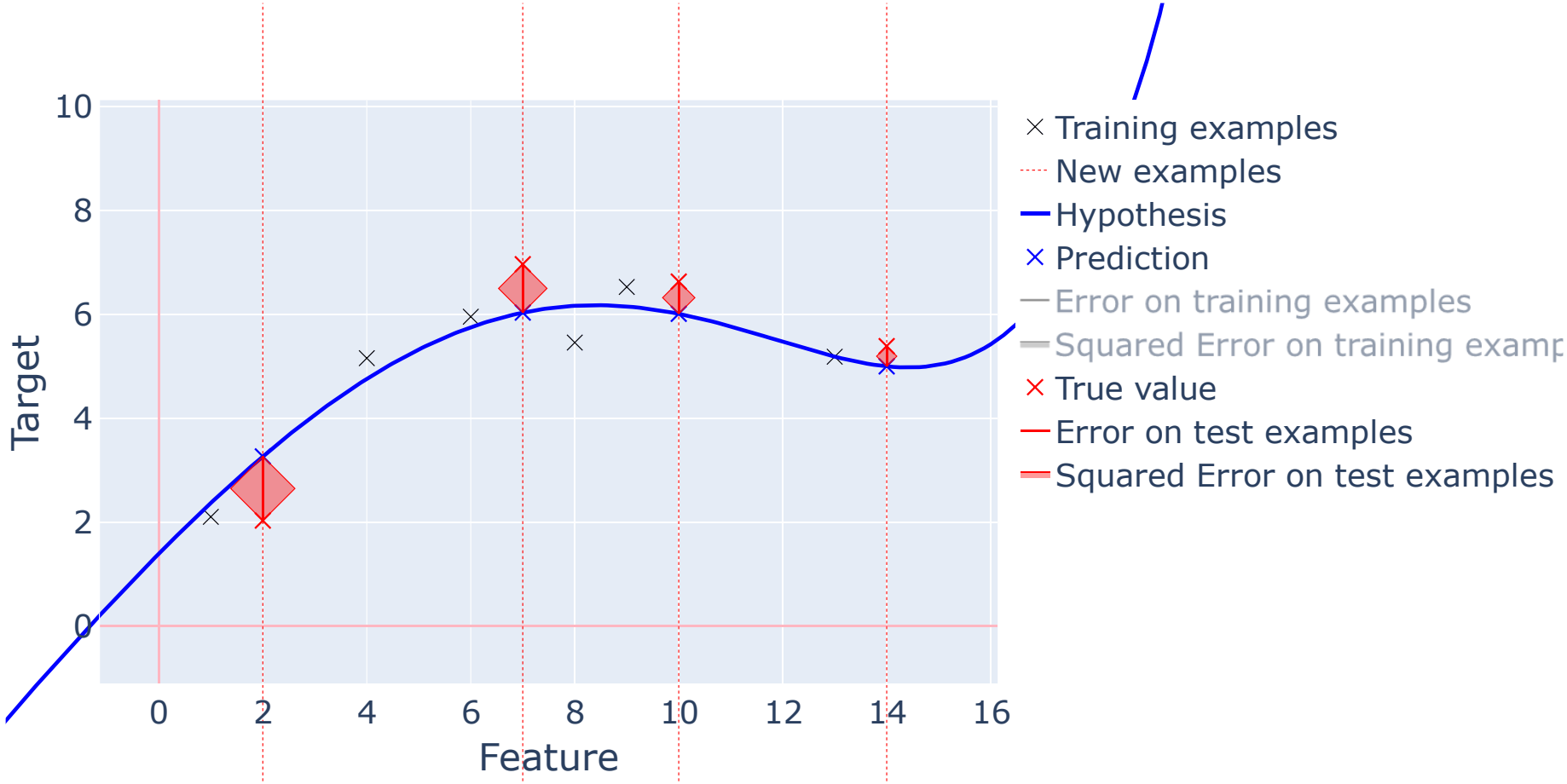
Varying λ: $\lambda = 0.001$



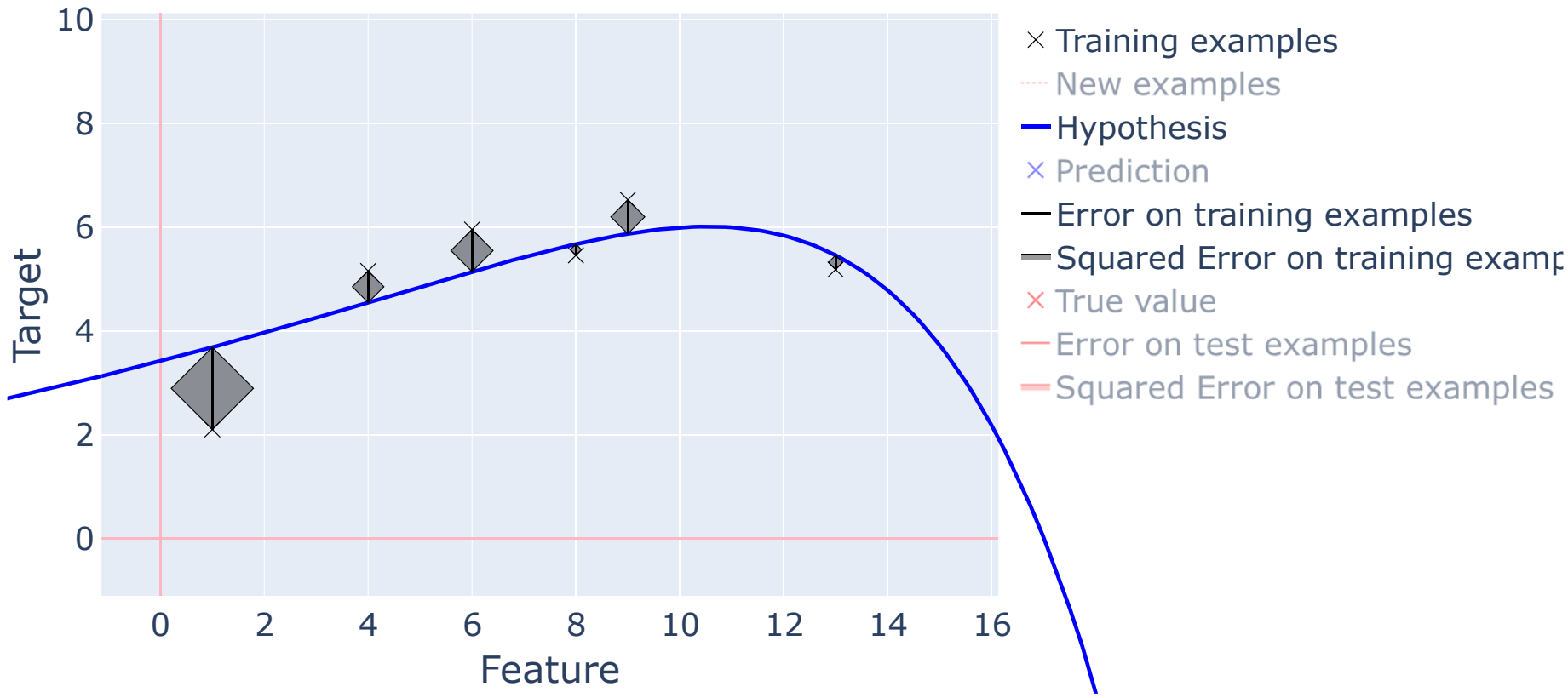
Varying λ: $\lambda = 0.1$



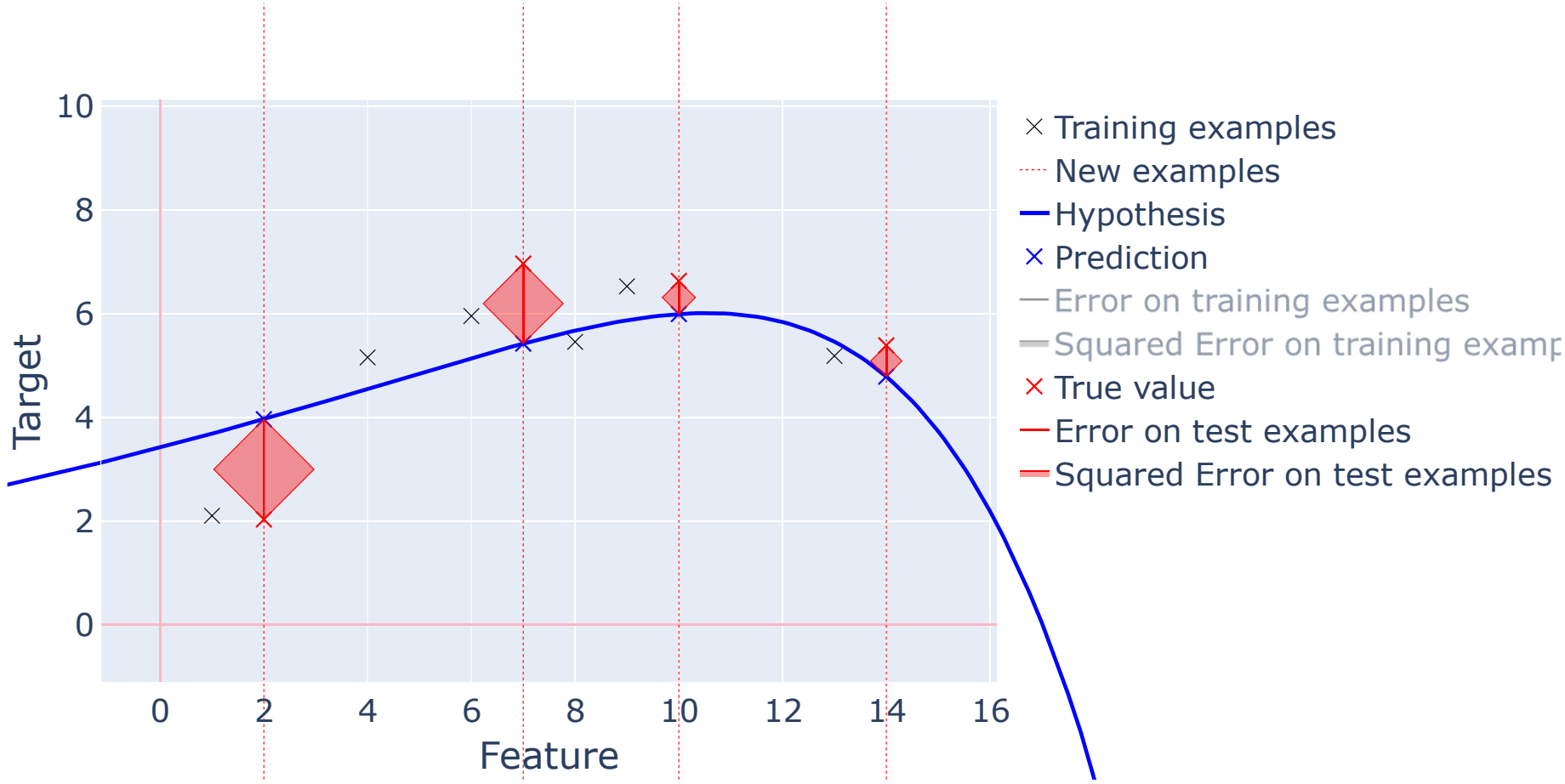
Varying λ: $\lambda = 0.1$



Varying λ: $\lambda = 10.0$



Varying λ: $\lambda = 10.0$



Varying λ
How to detect over/underfitting?

- Underfitting: λ – too large
  - The learned hypothesis…
    - Does not fit the training examples well
    - Fails to predict for new examples well
- Ideal: λ – appropriate
  - The learned hypothesis…
    - Fits the training examples well
    - Predicts for new examples well
- Overfitting: λ – too small
  - The learned hypothesis…
    - Fits the training examples extremely well
    - Fails to predicts for new examples well

Choose the best λ

- Input: candidates Λ (e.g. $\Lambda = (0.01, 0.1, 1.0, 10.0)$) for λ
- On training examples
  - Get $\boldsymbol{\theta}_\lambda^* \coloneqq \underset{\boldsymbol{\theta}}{\operatorname{argmin}} L(\boldsymbol{\theta}) + R(\boldsymbol{\theta}; \lambda)$ for $\lambda \in \Lambda$
  - i.e. Optimise parameter $\boldsymbol{\theta}$ for each λ
- On validation examples
  - Get $\lambda^* \coloneqq \underset{\lambda \in \Lambda}{\operatorname{argmin}} L(\boldsymbol{\theta}_\lambda^*)$
  - i.e. Choose the best λ
- On test examples
  - Compute $L(\boldsymbol{\theta}_{\lambda^*}^*)$ as an evaluation score

Overall process of ML

- Pre-process data
- Repeat:
  - Select model
  - Select features
  - Scale features
  - Set the regularisation parameter
  - Optimise loss function
  - Compare performance on training and validation data
    - If good enough, break the loop
- Evaluate performance on the test set