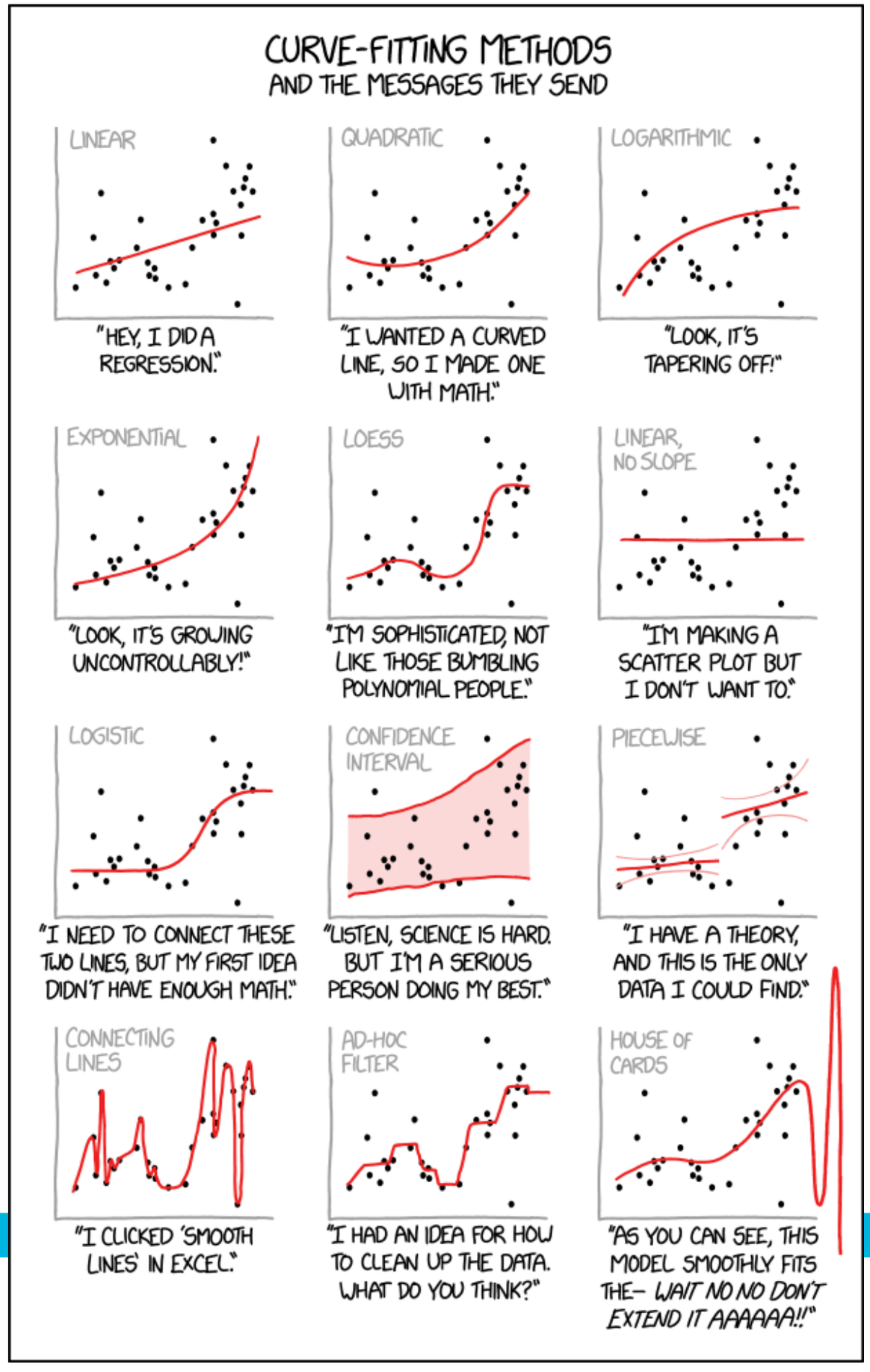
Nonlinear Regression

D.M.J. Tax

xkcd 2048



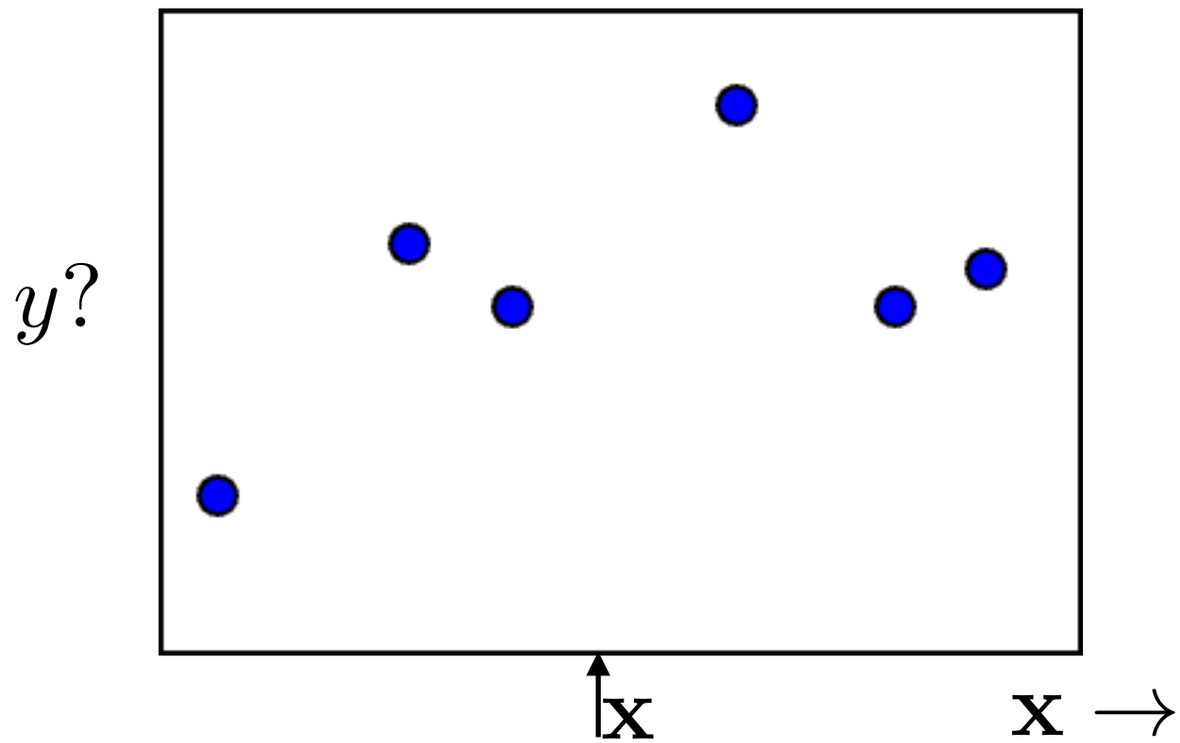
Contents

- Nonlinear 1D regression: (Kernel) Smoothers
- Local linear regression
- Kernel ridge regression
- (Gaussian Processes)
- Neural networks
- (Effective number of parameters)
- Conclusions

- See chapter 6 of 'Pattern Recognition and Machine Learning' by Bishop, up to section 6.4.3.

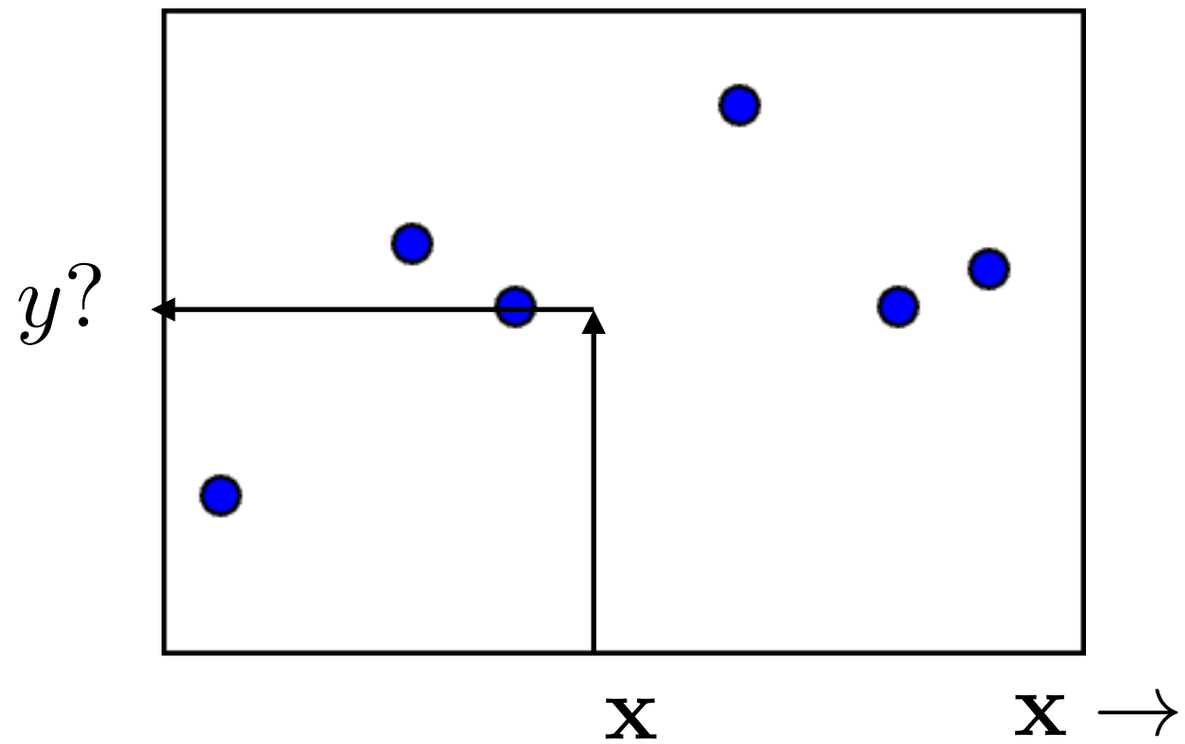
Nonlinear Regression

- Assume you don't think a linear regression is sufficient, what to do?



Nonlinear Regression

- Assume you don't think a linear regression is sufficient, what to do?



Nonlinear regression

- Given training data $(\mathbf{x}_1; y_1), ..., (\mathbf{x}_N; y_N)$ regressors estimate the curve

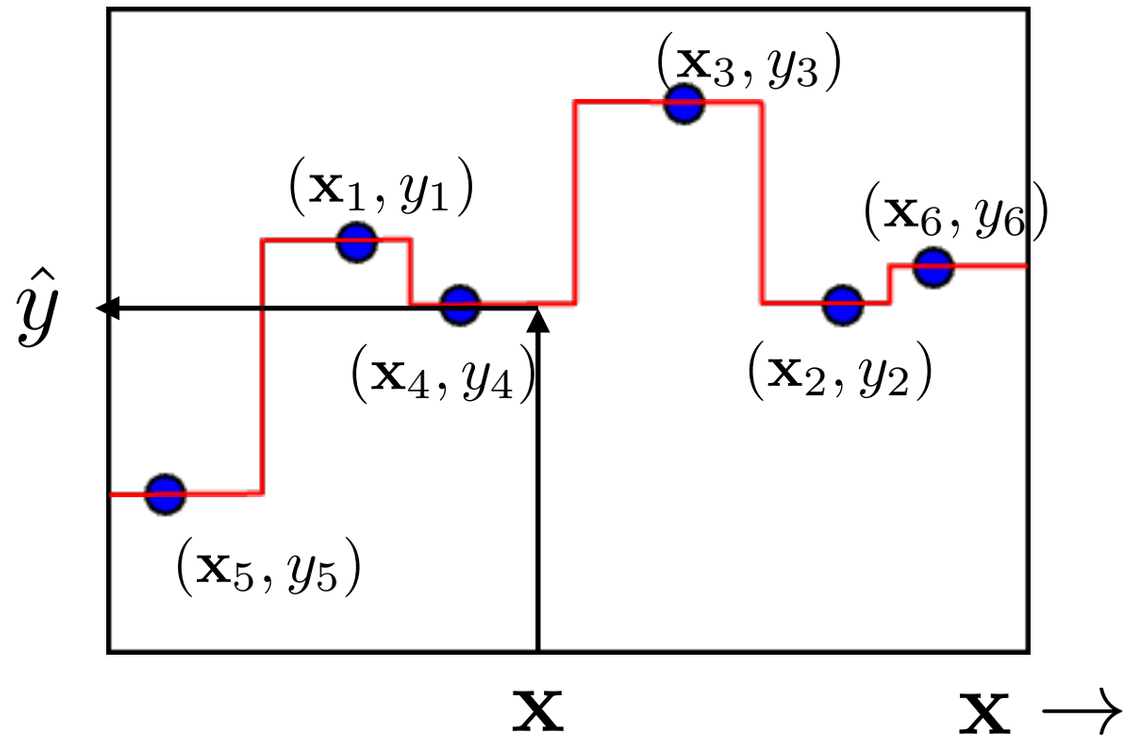
$$\hat{y}(\mathbf{x}) = E[y|\mathbf{x}]$$

- Concentrate on one-dimensional Smoothers: 'simple', 'kernel', 'local regression'

- Smoothers do Lazy learning: remember all training samples, estimate output from scratch for every new test sample

Simple smoothers

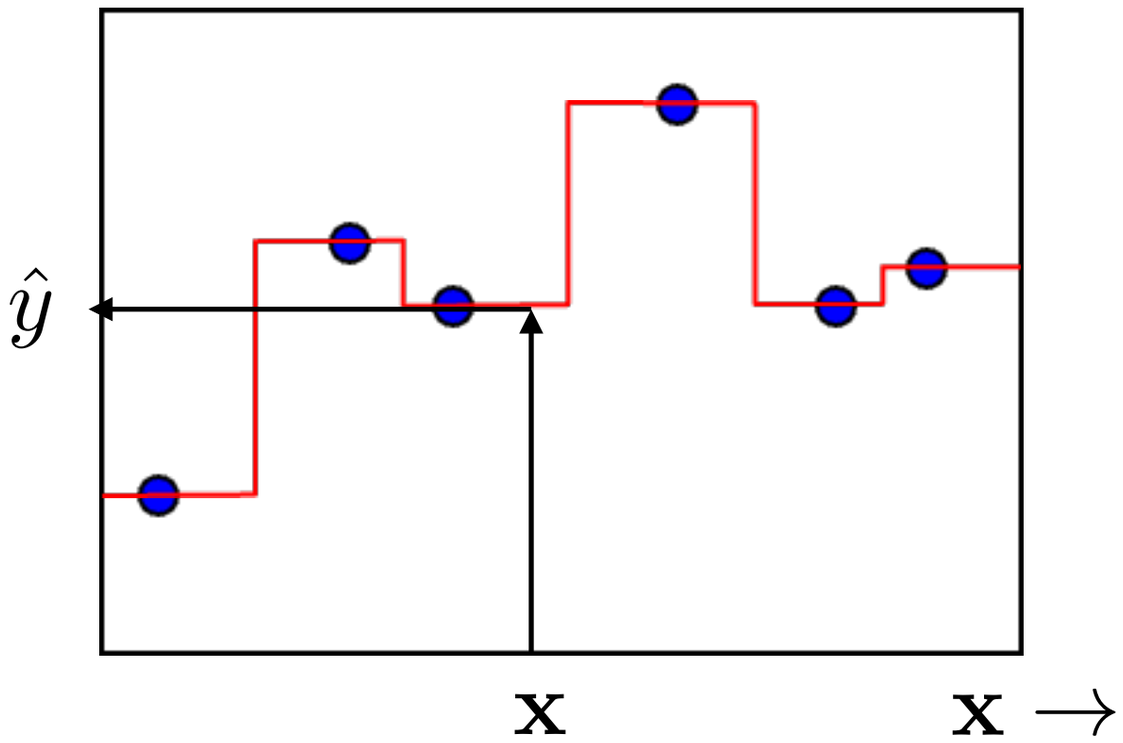
- k-nearest neighbor smoother, k=1



- How to formalize it?

Simple smoothers

- k-nearest neighbor smoother, k=1

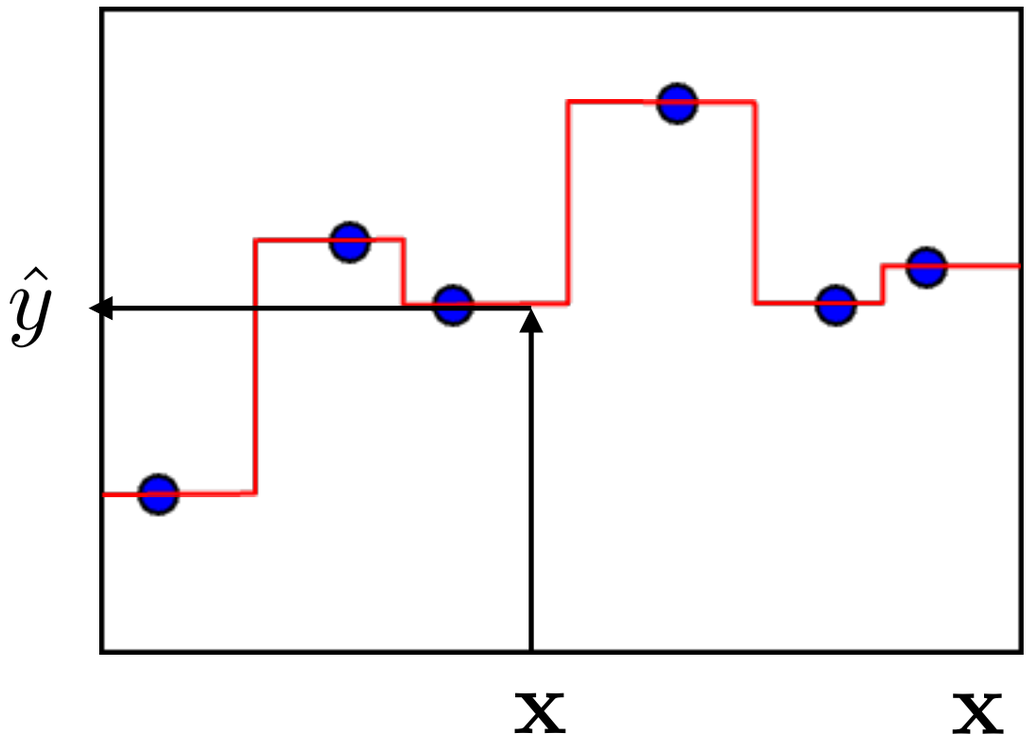


$$\hat{y}(\mathbf{x}) = \frac{1}{k} \sum_{i \in \mathcal{N}_k(\mathbf{x})} y_i$$

- where $\mathcal{N}_k(\mathbf{x})$ is the set of the k-nearest neighbors of **x** in the training set

Simple smoothers

- k-nearest neighbor smoother, k=1

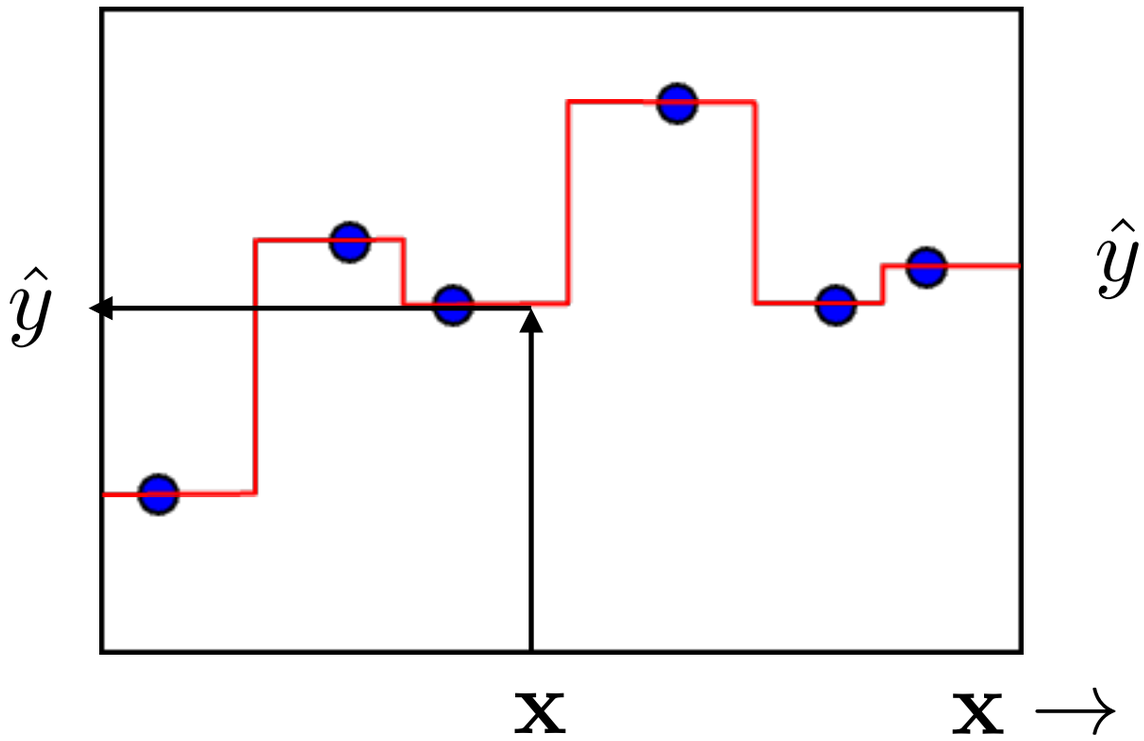


$$\hat{y}(\mathbf{x}) = \frac{1}{k} \sum_{i=1}^{N} \mathbb{I}(\mathbf{x}_i \in \mathcal{N}_k(\mathbf{x})) y_i$$

- where $\mathcal{N}_k(\mathbf{x})$ is the set of the k-nearest neighbors of **x** in the training set

Simple smoothers

- k-nearest neighbor smoother, k=1



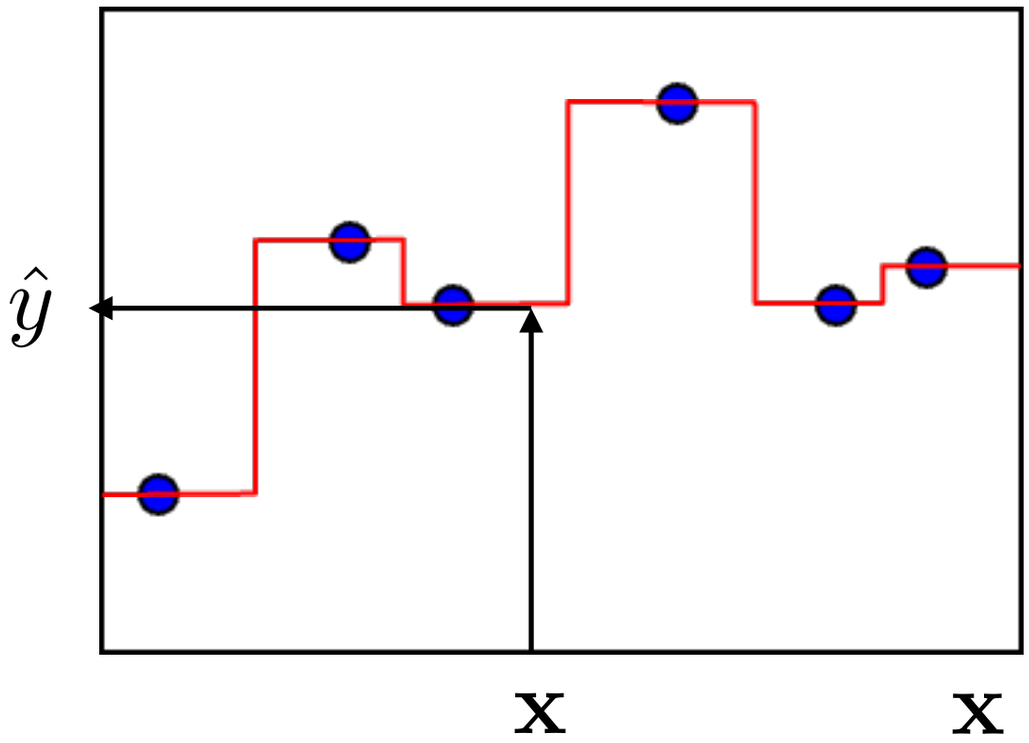
$$\hat{y}(\mathbf{x}) = \frac{1}{k}\sum_{i=1}^{N} \mathbb{I}(\mathbf{x}_i \in \mathcal{N}_k(\mathbf{x}))y_i$$

$$= \frac{\sum_{i=1}^{N} \mathbb{I}(\mathbf{x}_i \in \mathcal{N}_k(\mathbf{x}))y_i}{\sum_j \mathbb{I}(\mathbf{x}_j \in \mathcal{N}_k(\mathbf{x}))}$$

- where $\mathcal{N}_k(\mathbf{x})$ is the set of the k-nearest neighbors of **x** in the training set

Simple smoothers

- 'Nadaraya-Watson model', or 'kernel regression'



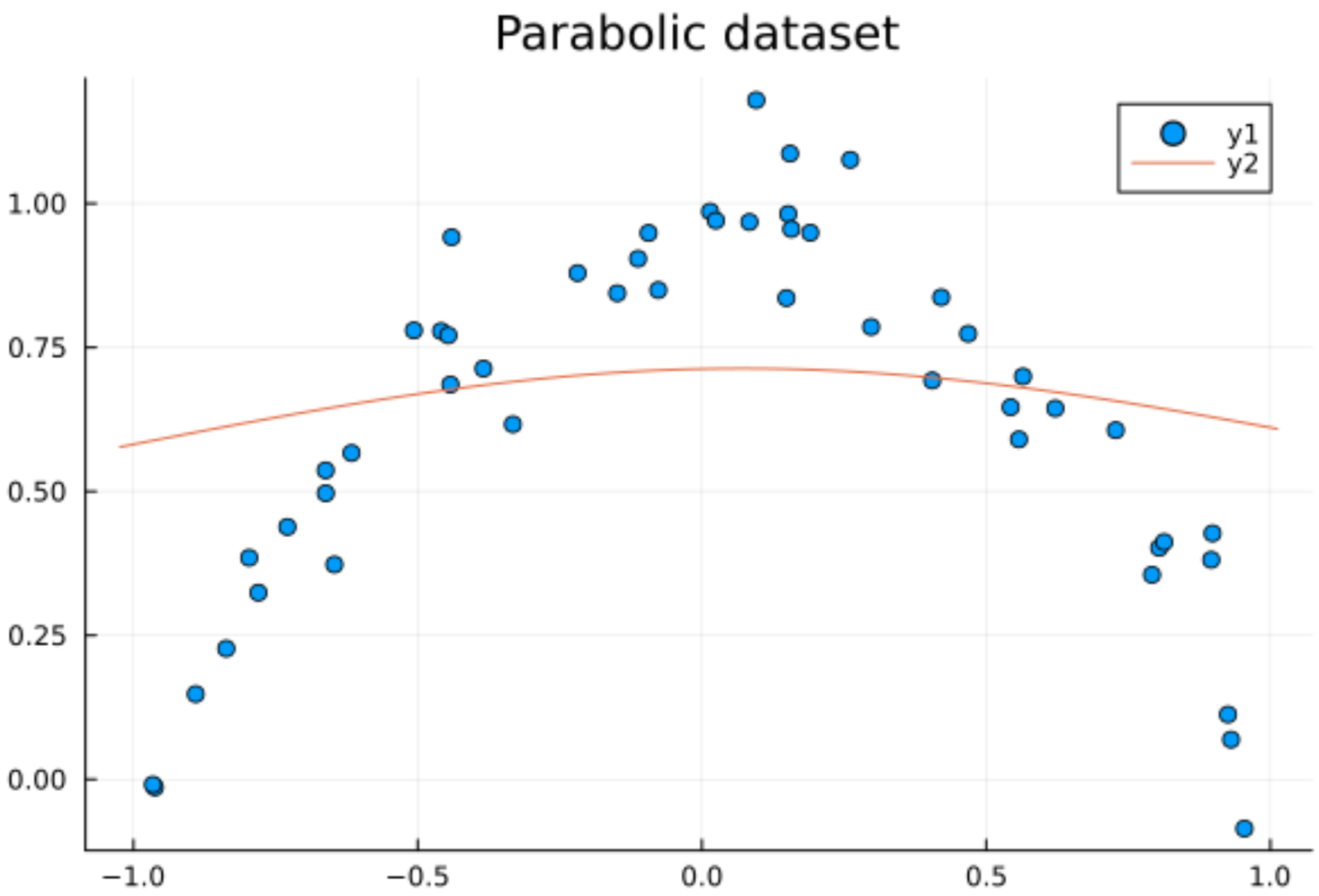
$$\hat{y}(\mathbf{x}) = \frac{\sum_{i=1}^{N} K(\mathbf{x}, \mathbf{x}_i) y_i}{\sum_{i=1}^{N} K(\mathbf{x}, \mathbf{x}_i)}$$

- where $K(\mathbf{x}, \mathbf{x}_i)$ is any kernel function (for instance, a Gaussian function)

Kernel smoother

- Use Gaussian kernel

$$K(\mathbf{x}, \mathbf{x}_i) = \frac{1}{Z} \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{2h^2}\right)$$

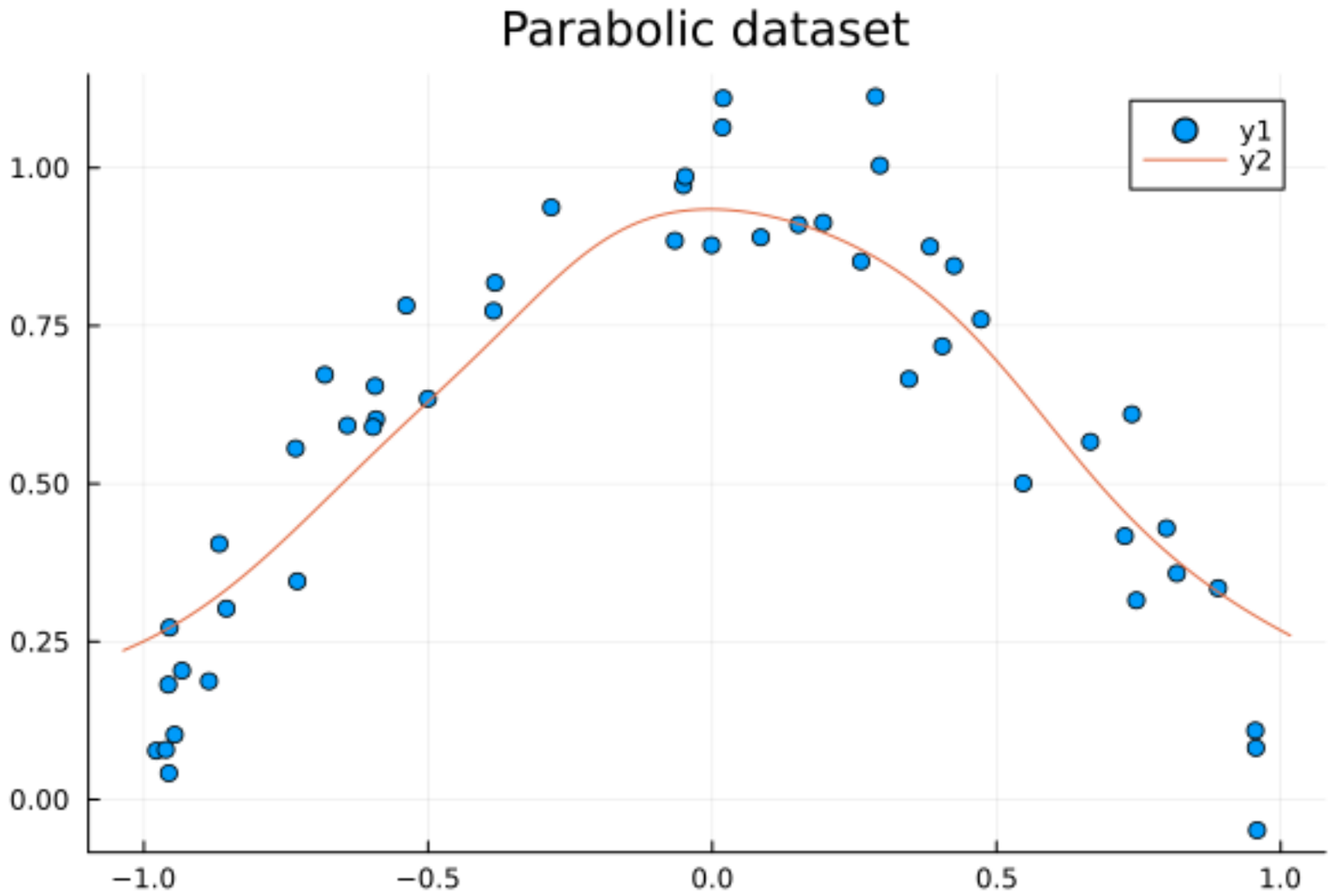


$h = 1$

Kernel smoother

- Use Gaussian kernel

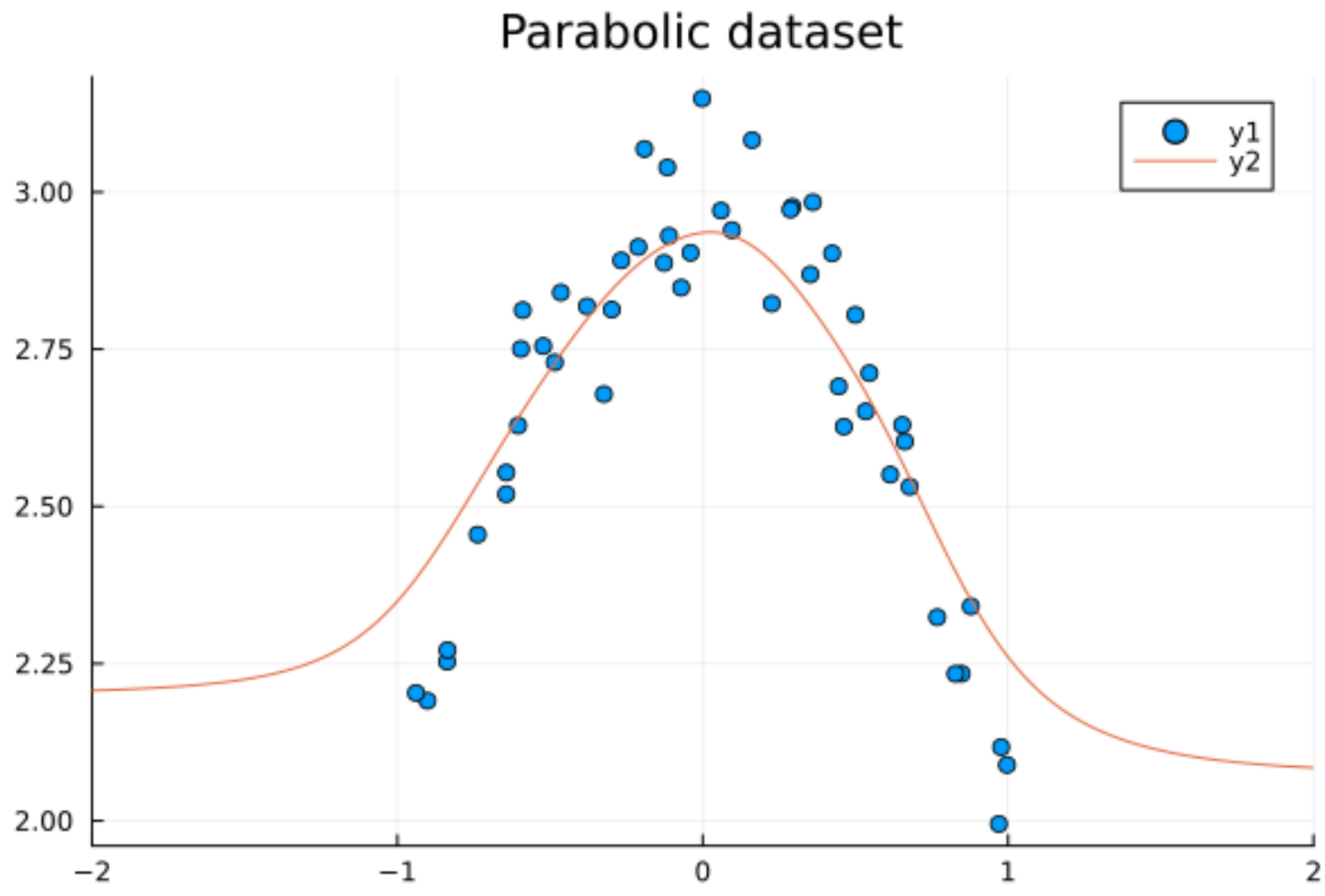
$$K(\mathbf{x}, \mathbf{x}_i) = \frac{1}{Z} \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_i\|^2}{2h^2}\right)$$



$h = 0.3$

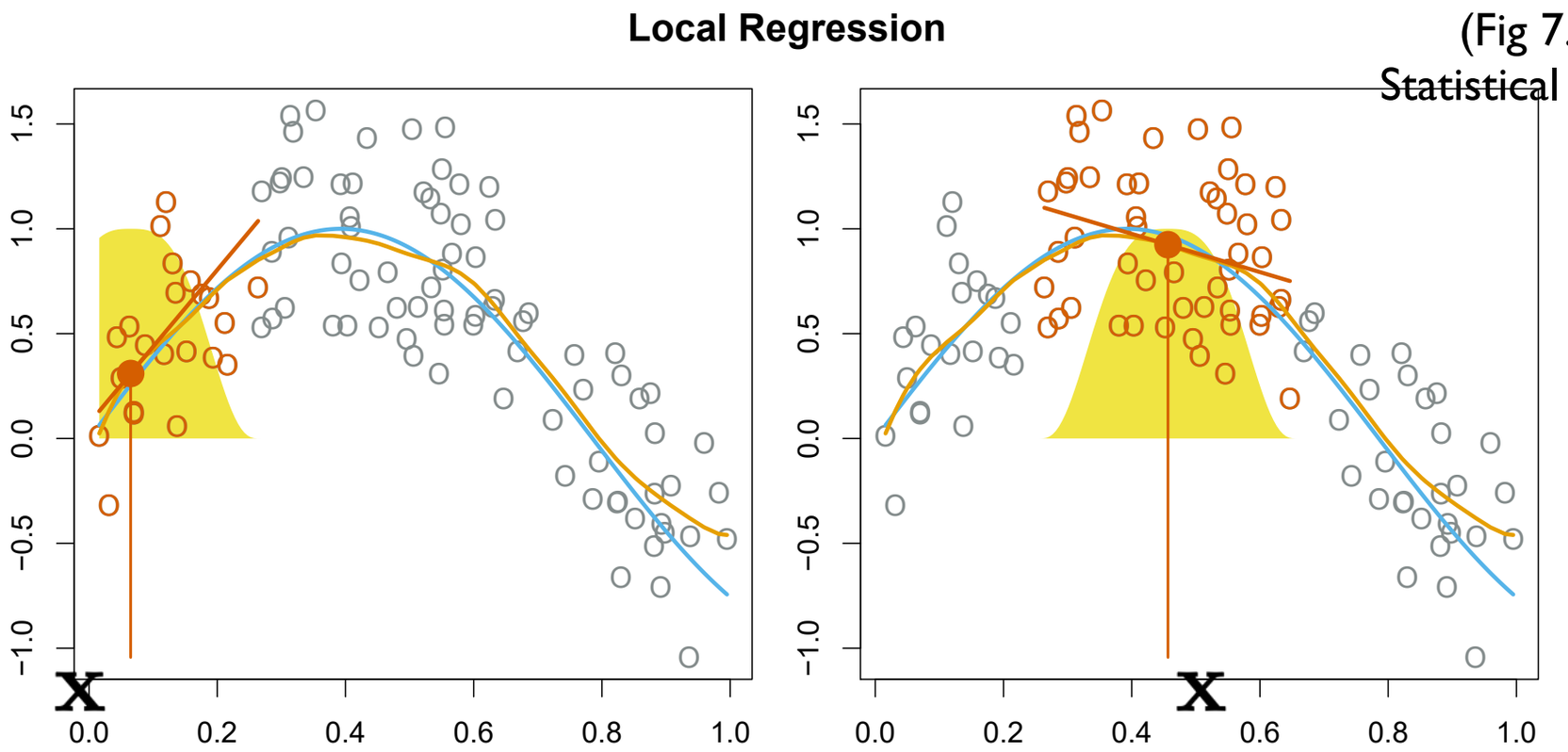
Kernel smoother

- The Gaussian kernel does extrapolate to flat line



$$h = 0.3$$

Local linear regression

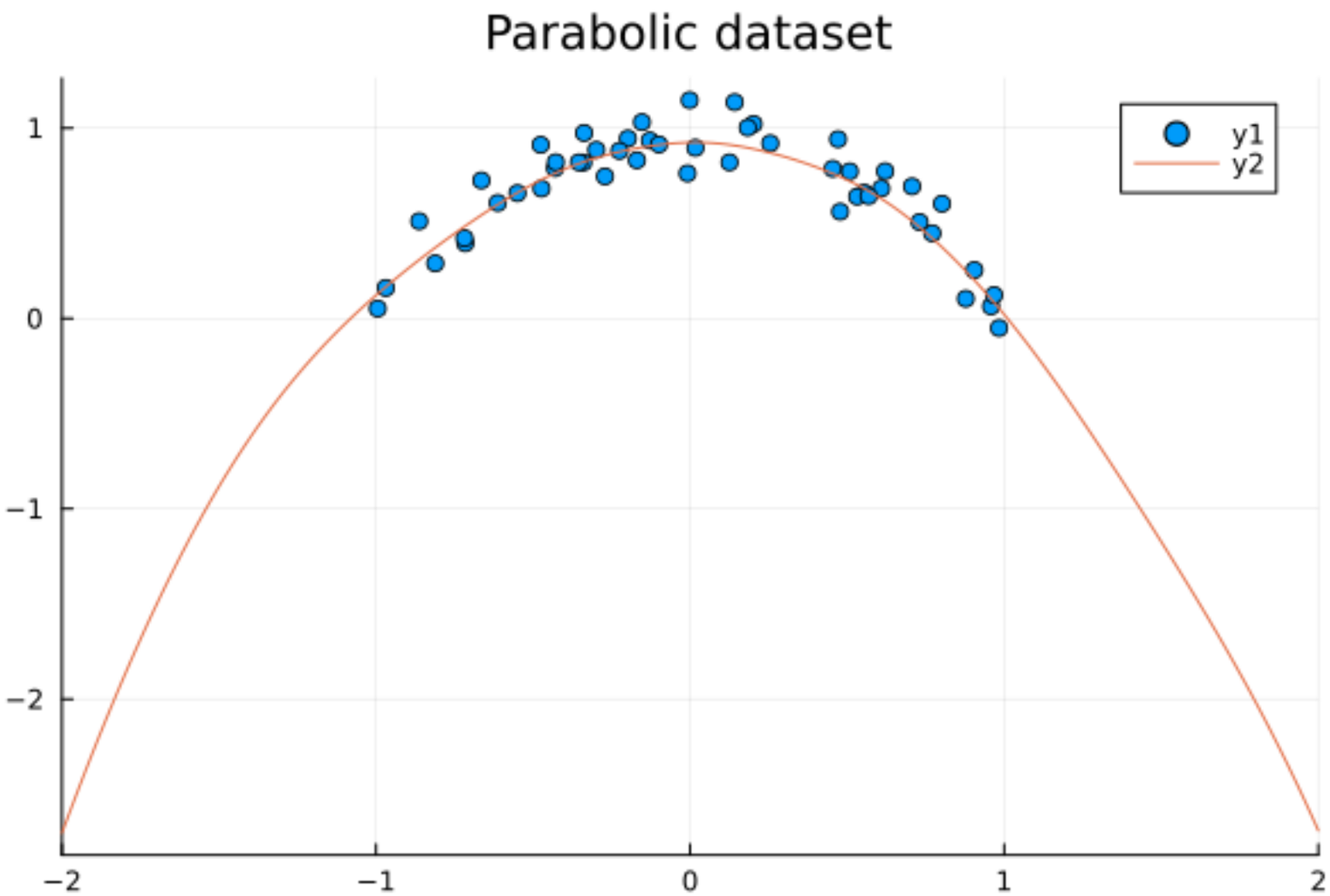


(Fig 7.9 Introduction to Statistical Learning, James et al.)

- Gather N local points around **x**
- Define a weight $w_i = K(\mathbf{x}, \mathbf{x}_i)$
- Do linear fit $L = \sum_{i=1}^{N} w_i (y_i - \beta_0 - \beta^T \mathbf{x}_i)^2$
- Predict $\hat{f}(\mathbf{x}) = \hat{\beta}_0 + \hat{\beta}^T \mathbf{x}$

Local linear regression

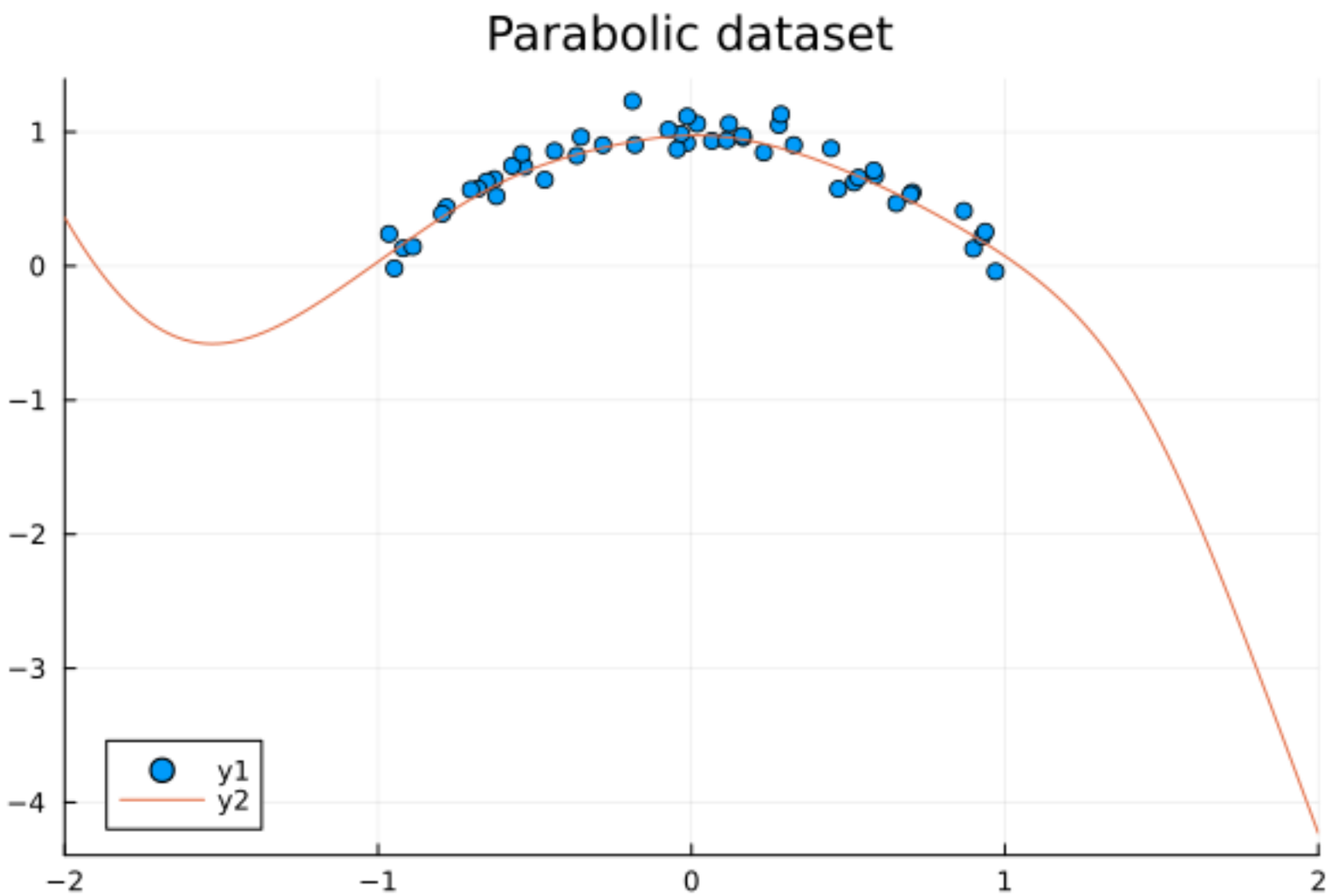
- Fit a line on the local neighborhood



$h = 0.3$

Local linear regression

- What is happening here?



$h = 0.3$

(Dis-)Advantages?

- + Relatively simple to implement
- + Hardly any training (lazy learning)
- + Relatively easy interpretation of hyperparameters
- (+ Some mathematical analysis is possible)

- - Very expensive during evaluation
- - Need to optimize the hyperparameters

Ridge regression

- Start with the classical linear regression, with some regularization:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)^2 + \lambda\|\mathbf{w}\|^2$$

- What was the solution again?

$$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$

- Fine. Now what if we want to kernelize?

Ridge regression

- Start with the classical linear regression, with some regularization:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)^2 + \lambda\|\mathbf{w}\|^2$$

- What was the solution again?

$$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$

$(d \times 1)$ $(d \times N)$ $(N \times d)$ $(d \times d)$ $(d \times N)$ $(N \times 1)$

- Fine. Now what if we want to kernelize?

Ridge regression

- Start with the classical linear regression, with some regularization:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)^2 + \lambda\|\mathbf{w}\|^2$$

- Now kernelize, or assume a feature mapping

$$\tilde{\mathbf{x}} = \Phi(\mathbf{x})$$

- So minimizing

$$\mathcal{L}(\tilde{\mathbf{w}}) = \frac{1}{2}\sum_{i=1}^{N}(\tilde{\mathbf{w}}^T\Phi(\mathbf{x}_i) - y_i)^2 + \lambda\|\tilde{\mathbf{w}}\|^2$$

gives

$$\hat{\mathbf{w}} = (\Phi(\mathbf{X})^T\Phi(\mathbf{X}) + \lambda\mathbf{I})^{-1}\Phi(\mathbf{X})^T\mathbf{y}$$

Kernel Ridge regression

- Applying this linear regression to a test point **z**:

$$\hat{y} = \Phi(\mathbf{z})^T \hat{\mathbf{w}} = \Phi(\mathbf{z})^T (\Phi(\mathbf{X})^T \Phi(\mathbf{X}) + \lambda \mathbf{I})^{-1} \Phi(\mathbf{X})^T \mathbf{y}$$

$(D \times 1)$ $(1 \times D)$ $(D \times N)$ $(N \times D)$ $(D \times D)$ $(D \times N)$ $(N \times 1)$

- In order to do the kernel trick, we need

$$K(\mathbf{X}, \mathbf{X}) = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & & \\ k(\mathbf{x}_N, \mathbf{x}_1) & \dots & k(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix} = \Phi(\mathbf{X}) \Phi(\mathbf{X})^T$$

$(N \times N)$

Need a trick

- How to convert $(\mathbf{AB}+\mathbf{I})^{-1} \rightarrow (\mathbf{BA}+\mathbf{I})^{-1}$??????

- It appears:

$$(\mathbf{AB}+\mathbf{I}_D)^{-1}\mathbf{A} = \mathbf{A}(\mathbf{BA}+\mathbf{I}_N)^{-1}$$

- (Proof:

$$\begin{aligned}(\mathbf{AB}+\mathbf{I}_D)^{-1}\mathbf{A} &= (\mathbf{AB}+\mathbf{I}_D)^{-1}\mathbf{A}(\mathbf{BA}+\mathbf{I}_N)(\mathbf{BA}+\mathbf{I}_N)^{-1} \\ &= (\mathbf{AB}+\mathbf{I}_D)^{-1}(\mathbf{ABA}+\mathbf{A})(\mathbf{BA}+\mathbf{I}_N)^{-1} \\ &= (\mathbf{AB}+\mathbf{I}_D)^{-1}(\mathbf{AB}+\mathbf{I}_D)\mathbf{A}(\mathbf{BA}+\mathbf{I}_N)^{-1} \\ &= \mathbf{A}(\mathbf{BA}+\mathbf{I}_N)^{-1} \qquad)\end{aligned}$$

Kernel Ridge regression

- Applying this linear regression to a test point **z**:

$$\hat{y} = \Phi(\mathbf{z})^T \hat{\mathbf{w}} = \Phi(\mathbf{z})^T (\underbrace{\Phi(\mathbf{X})^T}_{\mathbf{A}} \underbrace{\Phi(\mathbf{X})}_{\mathbf{B}} + \lambda \mathbf{I})^{-1} \underbrace{\Phi(\mathbf{X})^T}_{\mathbf{A}} \mathbf{y}$$

$$= \Phi(\mathbf{z})^T \Phi(\mathbf{X})^T (\Phi(\mathbf{X}) \Phi(\mathbf{X})^T + \lambda \mathbf{I})^{-1} \mathbf{y}$$

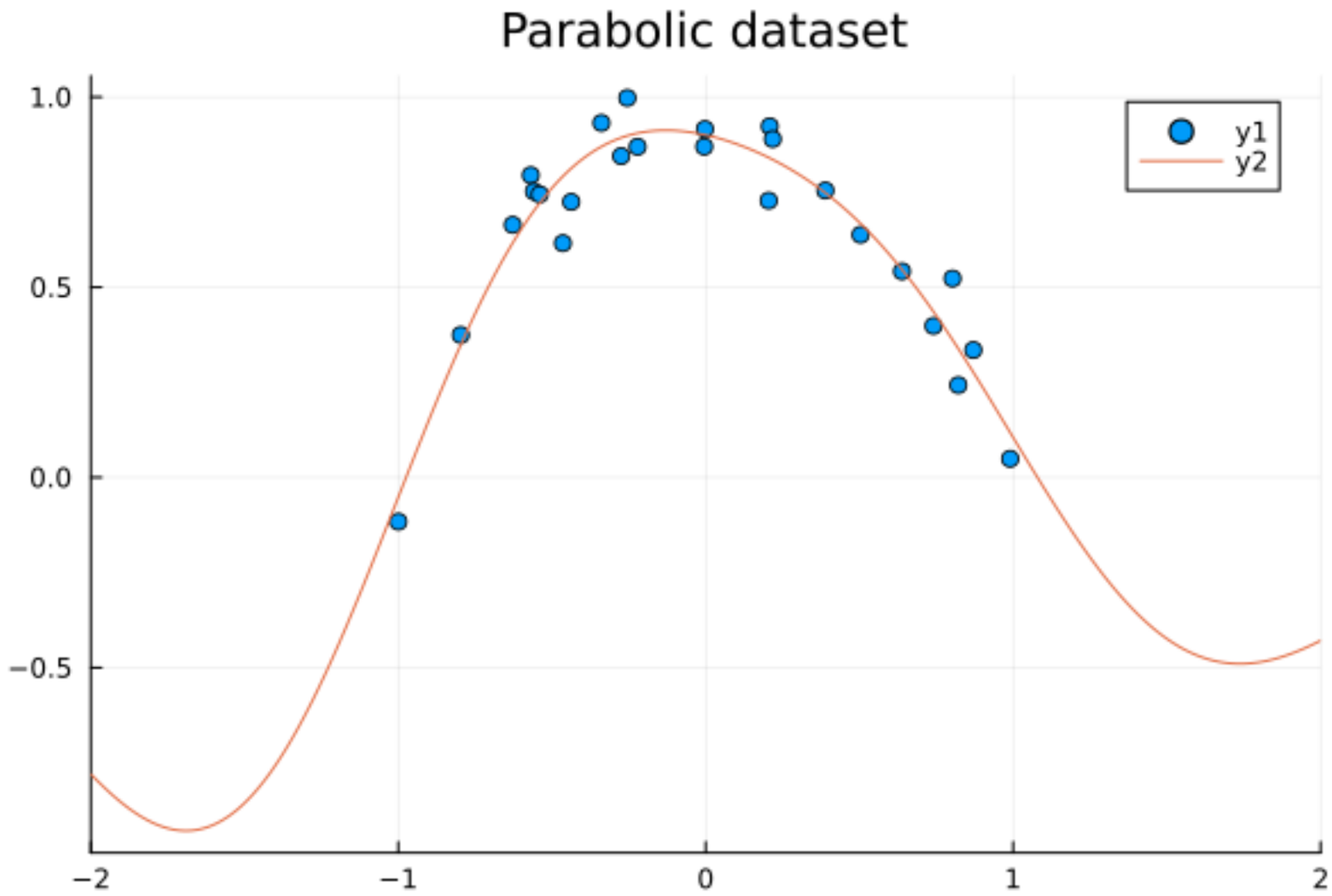
$$= K(\mathbf{z}, \mathbf{X})^T (K(\mathbf{X}, \mathbf{X}) + \lambda \mathbf{I})^{-1} \mathbf{y}$$

$$= \mathbf{y}^T (K(\mathbf{X}, \mathbf{X}) + \lambda \mathbf{I})^{-1} \mathbf{k}(\mathbf{z})$$

- where:

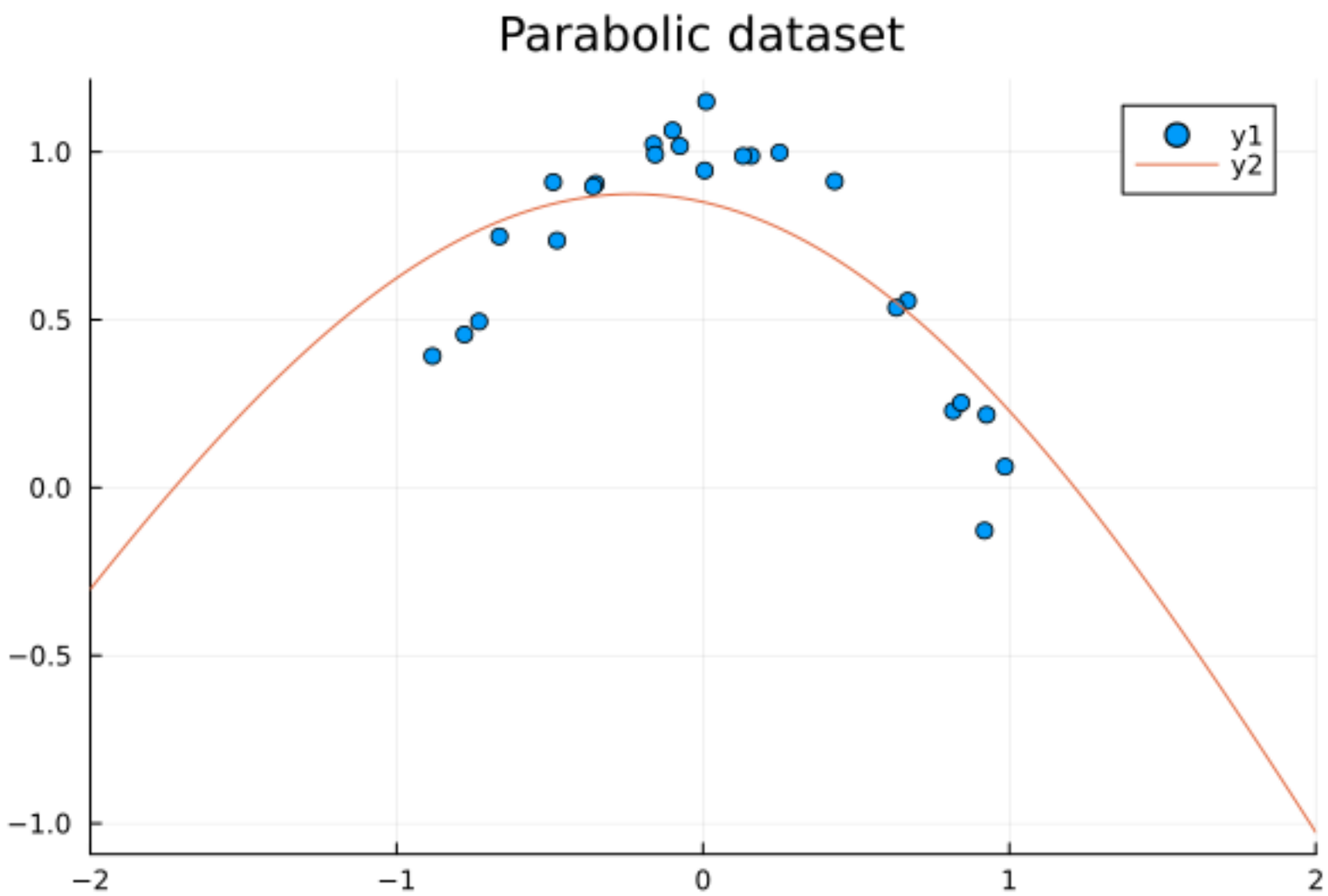
$$\mathbf{k}(\mathbf{z}) = [K(\mathbf{z}, \mathbf{x}_1), ..., K(\mathbf{z}, \mathbf{x}_N)]^T$$

Kernel Ridge regression



- With standard Gaussian kernel $\sigma = 1,\ \lambda = 0.01$

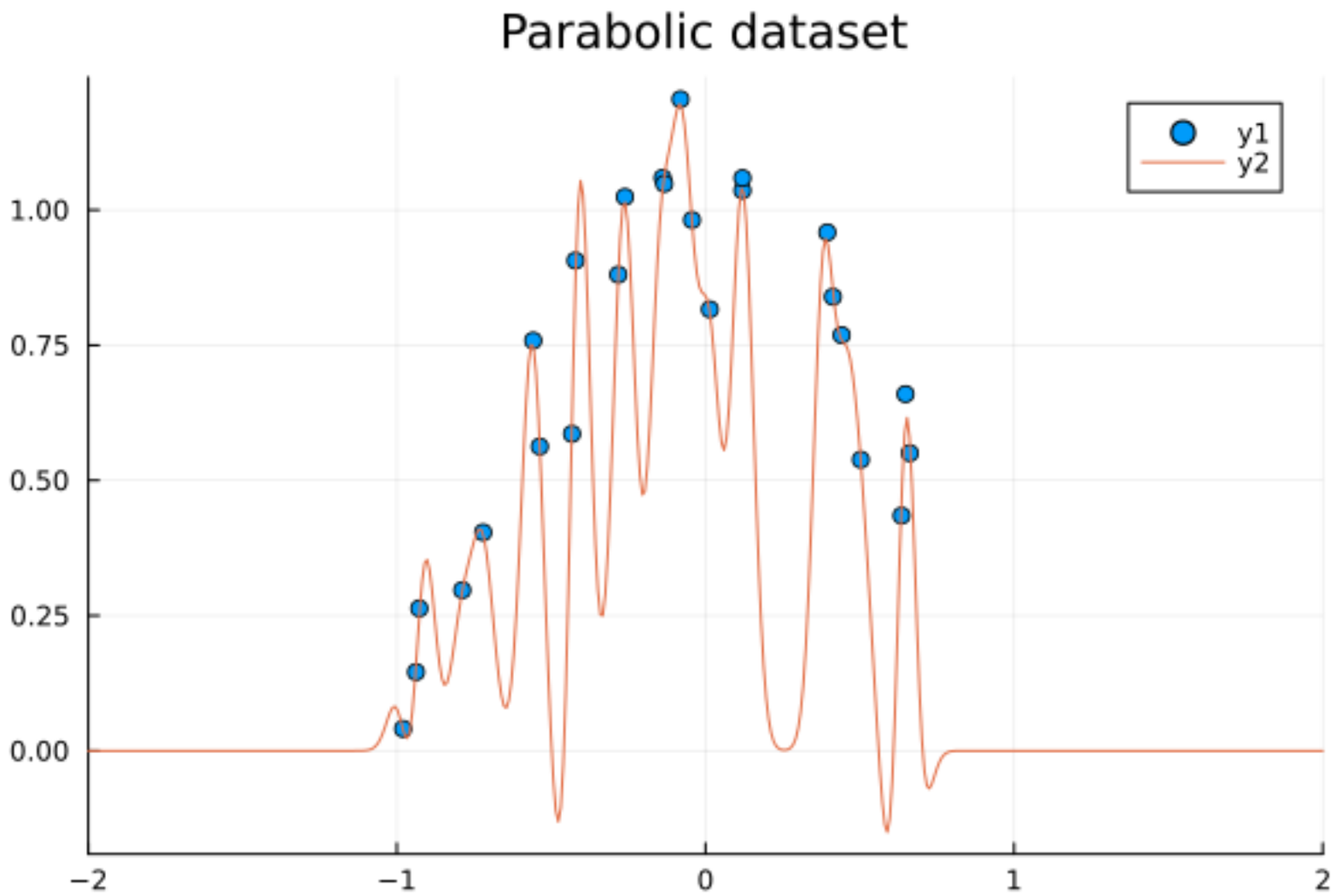
Kernel Ridge regression



much smoother function

- With standard Gaussian kernel $\sigma = 5,\ \lambda = 0.01$

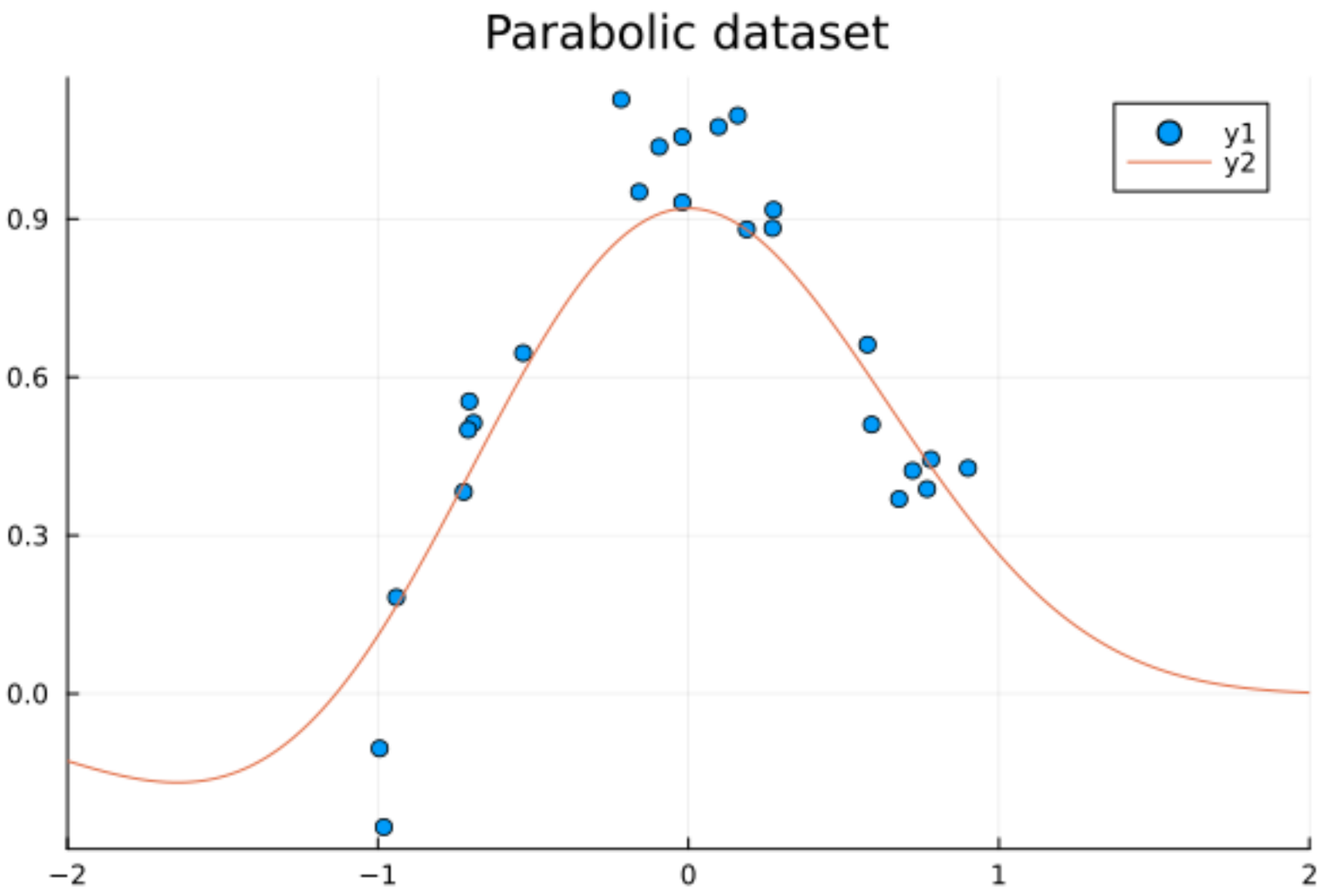
Kernel Ridge regression



much rougher function

- With standard Gaussian kernel $\sigma = 0.05,\ \lambda = 0.01$

Kernel Ridge regression



function stays closer to 0

- With standard Gaussian kernel $\sigma = 1.0,\ \lambda = 0.9$

(Dis-)Advantages

- + Relative easy to implement
- + Only need to store one (Nx1) weight vector
- + Relatively easy interpretation of hyperparameters

- - Need to invert the (NxN) kernel matrix
- - Need to optimize the hyperparameters

- BONUS: it is possible to get an additional estimate of the uncertainty: Gaussian Processes

- Lecture went up to here
- If you’re interested, look at the rest of the slides,
- If you’re not interested, just skip all and move to the conclusions

Bayesian Approach

- If we are fitting a model

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$$

on some dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$,
is it now logical to commit to one 'optimal' solution?

- You typically assume the data has additional noise, like $y_i = \mathbf{w}^T \phi(\mathbf{x}_i) + \varepsilon_i$, where ε_i may be Gaussian $\varepsilon_i \sim \mathcal{N}(\varepsilon|, 0, \sigma_\varepsilon^2)$
- In effect, you defined

$$p(y|\mathbf{x}, \mathbf{w}) = \mathcal{N}(y; \mathbf{w}^T \phi(\mathbf{x}), \sigma_\varepsilon)$$

Bayesian Approach

- If we are fitting a model

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$$

on some dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$,
is it now logical to commit to one 'optimal' solution?

- Bayesian approach:
  - Start with an prior belief in solutions $p(\mathbf{w})$
  - Update your believe with new incoming data

$$p(\mathbf{w}|\mathcal{D})$$

  - Use that all for doing prediction on **x**

$$p(y|\mathbf{x}, \mathcal{D}) = \int_{\mathbf{w}} p(y|\mathbf{x}, \mathbf{w}) p(\mathbf{w}|\mathcal{D}) d\mathbf{w}$$

Bayesian Approach

- How do you learn the parameter distribution form data?
- Bayes rule and the assumption of iid data

$$p(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})} \quad \text{(Bayes)} \quad = \frac{(\prod_{i=1}^{N} p(\mathbf{x}_i, y_i|\mathbf{w}))p(\mathbf{w})}{p(\mathcal{D})} \quad \text{(iid)}$$

$$= \frac{(\prod_{i=1}^{N} p(y_i|\mathbf{x}_i, \mathbf{w})p(\mathbf{x}_i|\mathbf{w}))p(\mathbf{w})}{p(\mathcal{D})} \quad \text{(cond.prob.)}$$

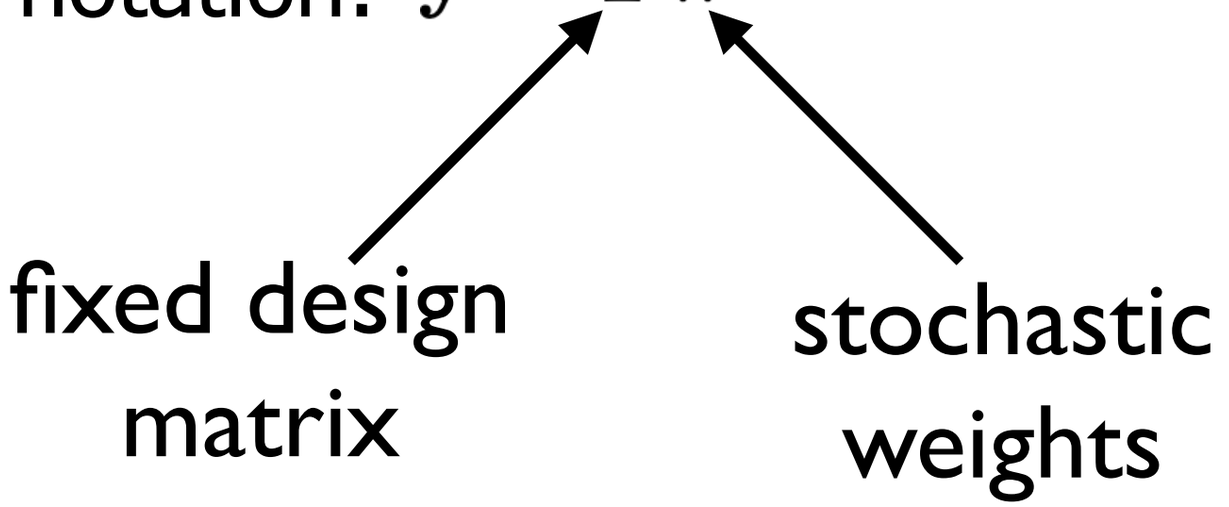
$$= \frac{(\prod_{i=1}^{N} p(y_i|\mathbf{x}_i, \mathbf{w})p(\mathbf{x}_i))p(\mathbf{w})}{p(\mathcal{D})} \quad \text{(x ind. of w)}$$

$$= \frac{(\prod_{i=1}^{N} p(y_i|\mathbf{x}_i, \mathbf{w}))p(\mathbf{w})}{Z} \quad \text{(move constants)}$$

product of all Gaussians!

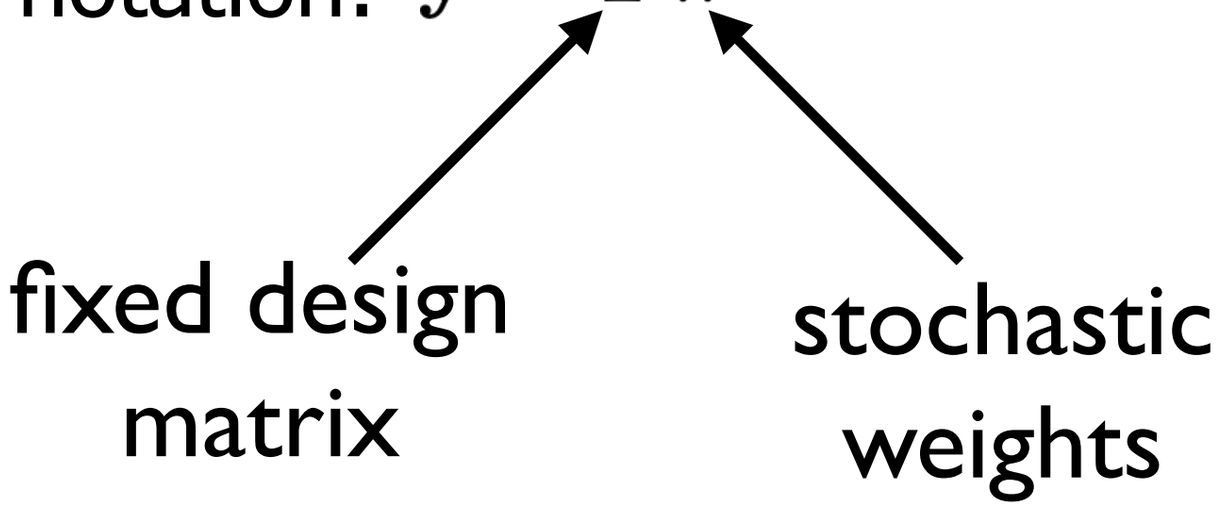
Gaussian processes

- Assume f is a linear combination of M basis functions $y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$
- In vector notation: $\mathbf{y} = \mathbf{\Phi w}$



Gaussian processes

- Assume f is a linear combination of M basis functions $y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$
- In vector notation: $\mathbf{y} = \mathbf{\Phi w}$



- Now use a Bayesian approach and define a prior over weights $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, h^2\mathbb{I})$
- Assume we have a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ what is now the likelihood for the labels y_i ?

Gaussian processes

- Assume f is a linear combination of M basis functions $y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$
- In vector notation: $\mathbf{y} = \mathbf{\Phi w}$

(α^{-1} in Bishop)

- Now use a Bayesian approach and define a prior over weights $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, h^2\mathbb{I})$
- Assume we have a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ what is now the likelihood for the labels y_i ?

Gaussian processes

$$\mathbf{y} = \mathbf{\Phi w}$$

- **y** is a linear combination of Gaussians (also becomes a Gaussian)

$$E[\mathbf{y}] = E[\mathbf{\Phi w}] = \mathbf{\Phi} E[\mathbf{w}] = \mathbf{\Phi 0} = \mathbf{0}$$

$$Cov[\mathbf{y}] = E[\mathbf{y}\mathbf{y}^Y] = \mathbf{\Phi} E[\mathbf{w}\mathbf{w}^T]\mathbf{\Phi}^T = h^2 \mathbf{\Phi}\mathbf{\Phi}^T = \mathbf{K}$$

  where $\mathbf{K}_{nm} = k(\mathbf{x}_n, \mathbf{x}_m) = h^2 \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m)$

- So $p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K})$

$$\varepsilon_i \sim \mathcal{N}(\varepsilon|, 0, \sigma_\varepsilon^2)$$

- If you have additional noise $\mathbf{y} = \mathbf{\Phi w} + \varepsilon$

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K} + \sigma_\varepsilon^2 \mathbb{I})$$

Gaussian Process

- Now I would like to predict for a new object $\mathbf{x}_{N+1}$
- Still the joint distribution of the labels is Gaussian

$$p(\mathbf{y}_{N+1}) = \mathcal{N}(\mathbf{y}_{N+1}|\mathbf{0}, \mathbf{C})$$

where

$$\mathbf{C} = \begin{pmatrix} \mathbf{K} + \sigma_\varepsilon^2 & \mathbf{k} \\ \mathbf{k}^T & c \end{pmatrix}$$

(extend the original covariance matrix with the new object)

$$\mathbf{k} = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_{N+1}) \\ \dots \\ k(\mathbf{x}_N, \mathbf{x}_{N+1}) \end{pmatrix}$$

$$c = k(\mathbf{x}_{N+1}, \mathbf{x}_{N+1}) + \sigma_\varepsilon^2$$

Partitioned Gaussians

- Given joint distribution $\mathcal{N}(\mathbf{x}|\mu, \Sigma), \Lambda = \Sigma^{-1}$

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{pmatrix} \qquad \mu = \begin{pmatrix} \mu_a \\ \mu_b \end{pmatrix}$$

$$\mathbf{\Sigma} = \begin{pmatrix} \mathbf{\Sigma}_{aa} & \mathbf{\Sigma}_{ab} \\ \mathbf{\Sigma}_{ba} & \mathbf{\Sigma}_{bb} \end{pmatrix} \quad \mathbf{\Lambda} = \begin{pmatrix} \mathbf{\Lambda}_{aa} & \mathbf{\Lambda}_{ab} \\ \mathbf{\Lambda}_{ba} & \mathbf{\Lambda}_{bb} \end{pmatrix}$$

then the conditional distribution becomes:

$$p(\mathbf{x}_a|\mathbf{x}_b) = \mathcal{N}(\mathbf{x}|\mu_{a|b}, \Sigma_{a|b})$$

$$\mu_{a|b} = \mu_a - \Sigma_{ab}\Sigma_{bb}^{-1}(\mathbf{x}_b - \mu_b)$$

$$\Sigma_{a|b} = \Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}$$

Gaussian Processes

- So we condition on the training data:

$$p(y_{N+1}|\mathcal{D}) = \mathcal{N}(y_{N+1}|m(\mathbf{x}_{N+1}), s^2(\mathbf{x}_{N+1}))$$

$$m(\mathbf{x}_{N+1}) = \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{y}$$

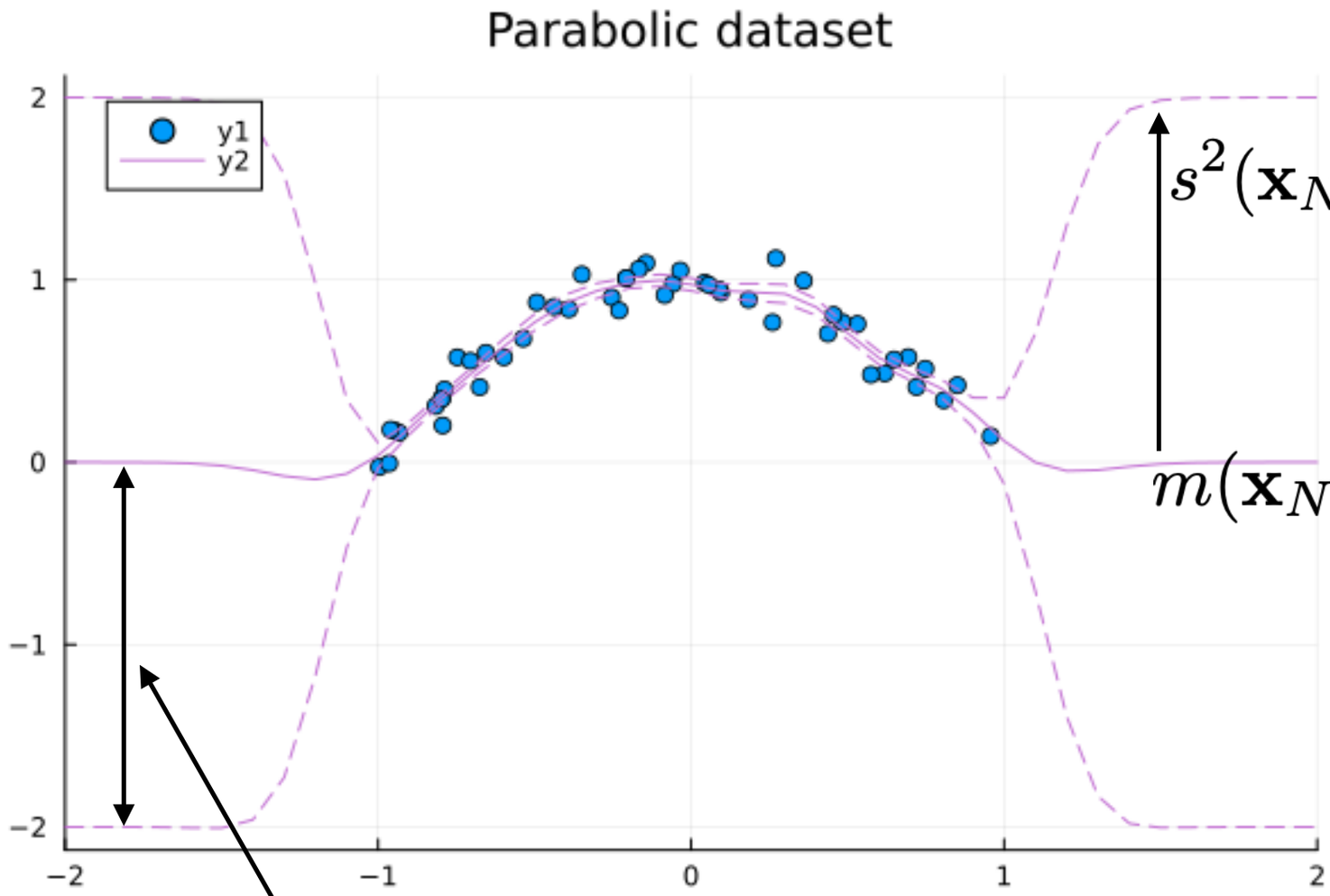
$$s^2(\mathbf{x}_{N+1}) = c - \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{k}$$

- When we finally assume that

$$\mathbf{K}_{nm} = k(\mathbf{x}_n, \mathbf{x}_m) = h^2 \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m) = \frac{h^2}{Z} \exp\left(-\gamma \|\mathbf{x}_n - \mathbf{x}_m\|^2\right)$$

we obtained a **Gaussian Process**

Gaussian Process

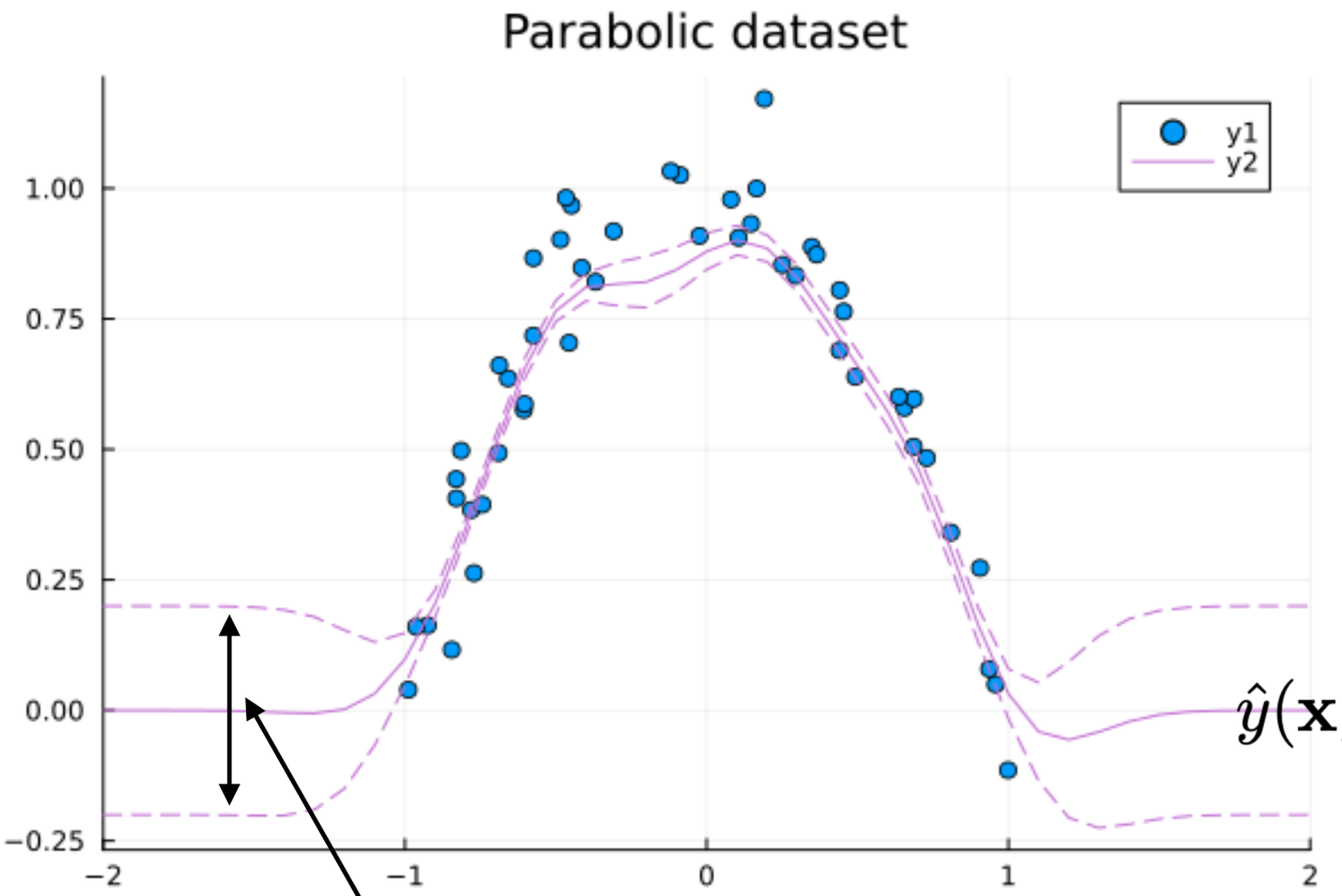


$$s^2(\mathbf{x}_{N+1}) = c - \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{k}$$

$$m(\mathbf{x}_{N+1}) = \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{y}$$

- Uncertainty band outside the data distribution is determined by prior variance h

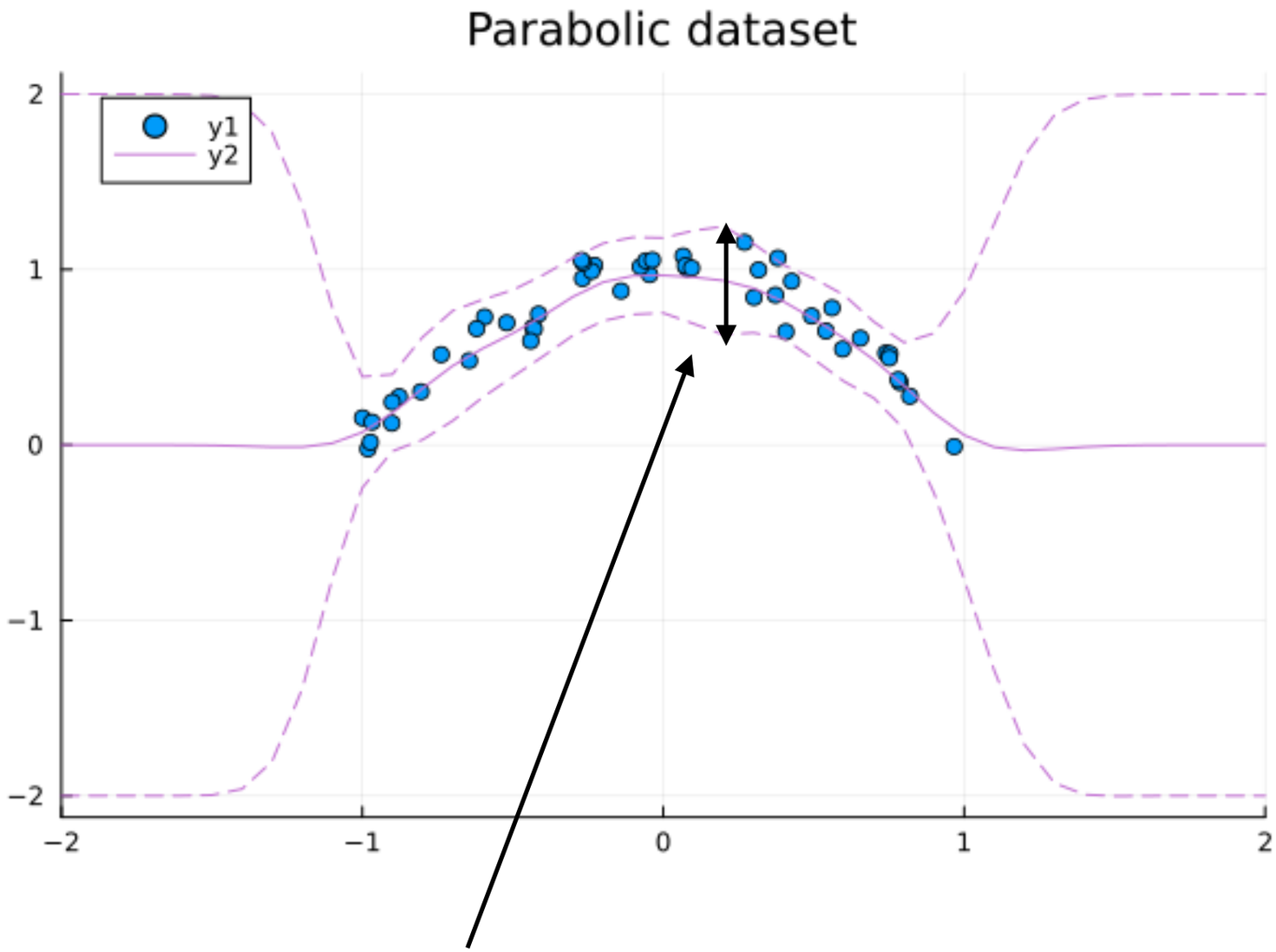
Changing the prior h



$$\hat{y}(\mathbf{x}_{N+1}) = \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{y}$$

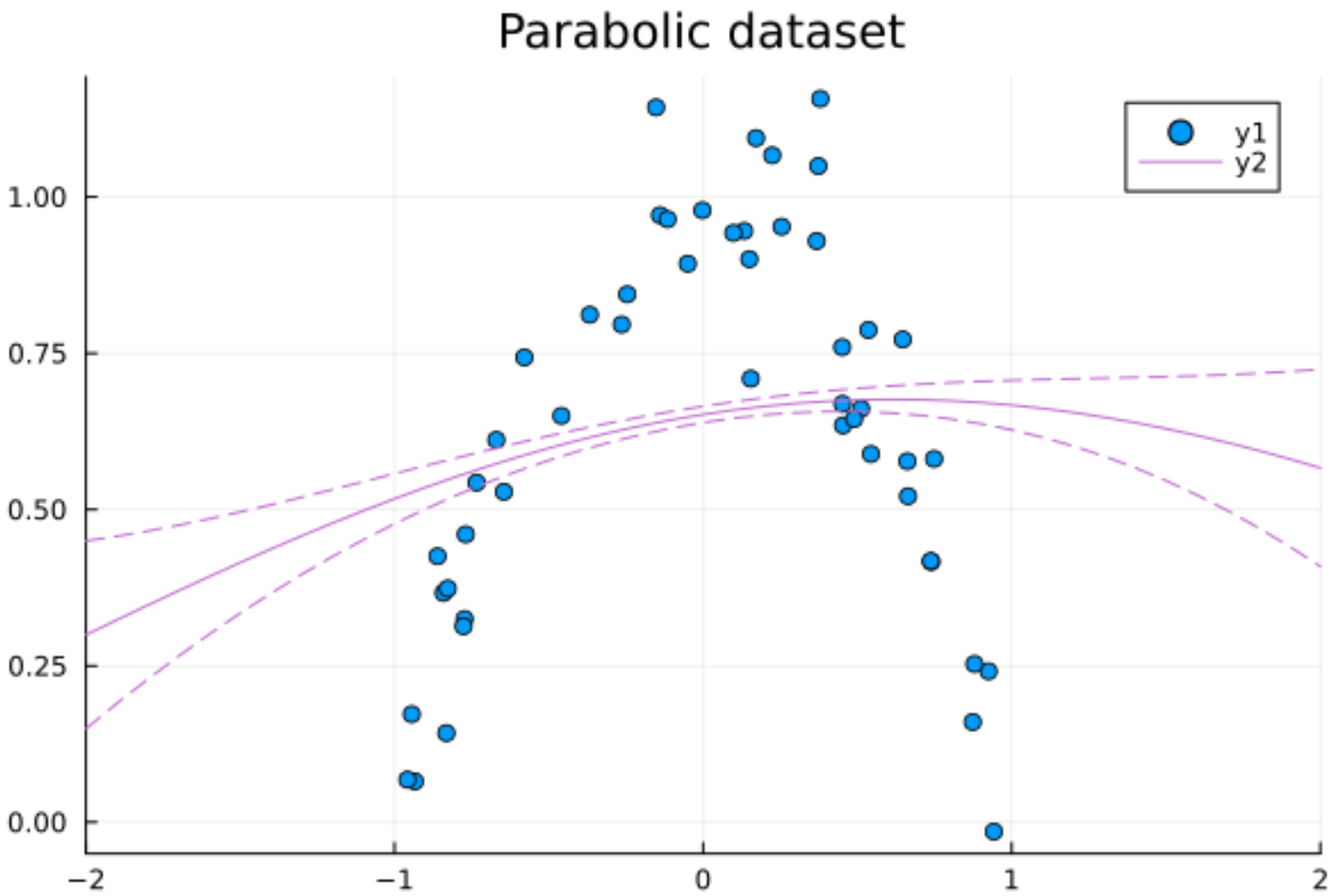
- Prior variance h has been decreased

Changing noise variance



- Increasing noise variance, σ_ε increased the spread in the data as well

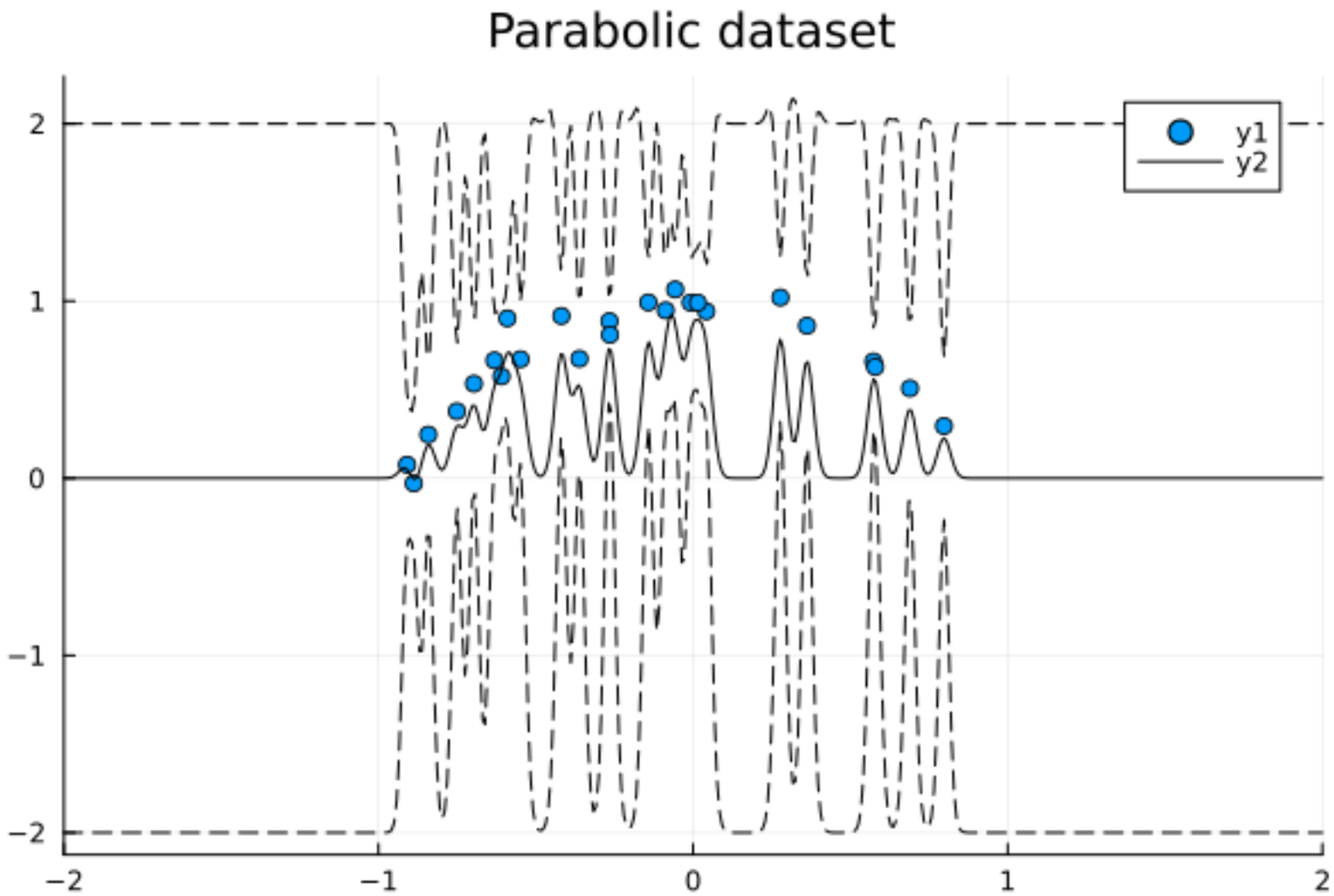
Changing scale of K



γ small

- The scale γ in the kernel K determines how smooth the function becomes

Changing scale of K



γ large

- The scale γ in the kernel K determines how smooth the function becomes

Gaussian processes

- Bayesian approach to regression

- Assumes prior on the weights: wide prior gives large uncertainties (in particular when there is no data)
- Models Gaussian noise on the output: large noise give large uncertainties (in particular where there is data)
- Allows to define a kernel between objects: a wide kernel gives a smooth regression function

Neural networks

- The default 'solution' nowadays are (deep) neural networks, often with MSE

$$\mathcal{L}(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N}(f_{W_L}(...f_{W_1}(\mathbf{x}_i)...) - y_i)^2$$

- No need for (inverse) kernel matrices, or storing the training set
- But training involves many tricks, hyperparameters and tuning, and typically requires large training sets

Back to smoothers

- Note that for Gaussian processes, the mean

$$\hat{y}(\mathbf{x}_{N+1}) = m(\mathbf{x}_{N+1}) = \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{y}$$

also is of the form of

$$\hat{y}(\mathbf{x}) = \mathbf{s}^T(\mathbf{x})\mathbf{y}$$

smoothing function

all the labels of the training data

- Compare with kernel regressor:

$$\hat{y}(\mathbf{x}) = \frac{\sum_{i=1}^{N} K(\mathbf{x}, \mathbf{x}_i) y_i}{\sum_{i=1}^{N} K(\mathbf{x}, \mathbf{x}_i)}$$

Effective nr of parameters

- Assume we have a smoother:

$$\hat{y}(\mathbf{x}) = \mathbf{s}^T(\mathbf{x})\mathbf{y}$$

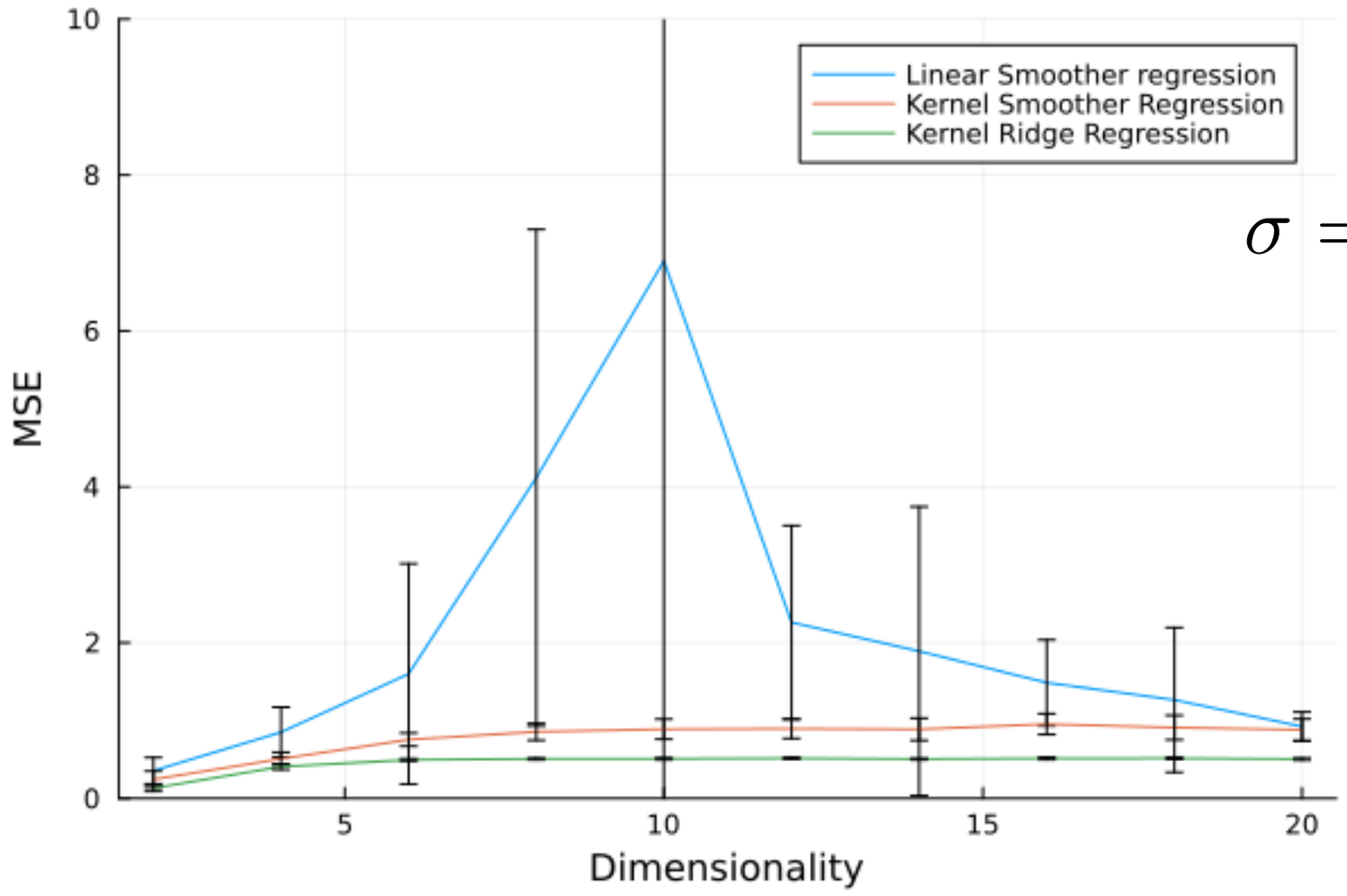
- So, for the training set holds

$$\hat{\mathbf{y}} = \mathbf{S}\mathbf{y}$$

- One definition for the effective nr of parameters is:

$$p_e = \mathrm{trace}(\mathbf{S})$$

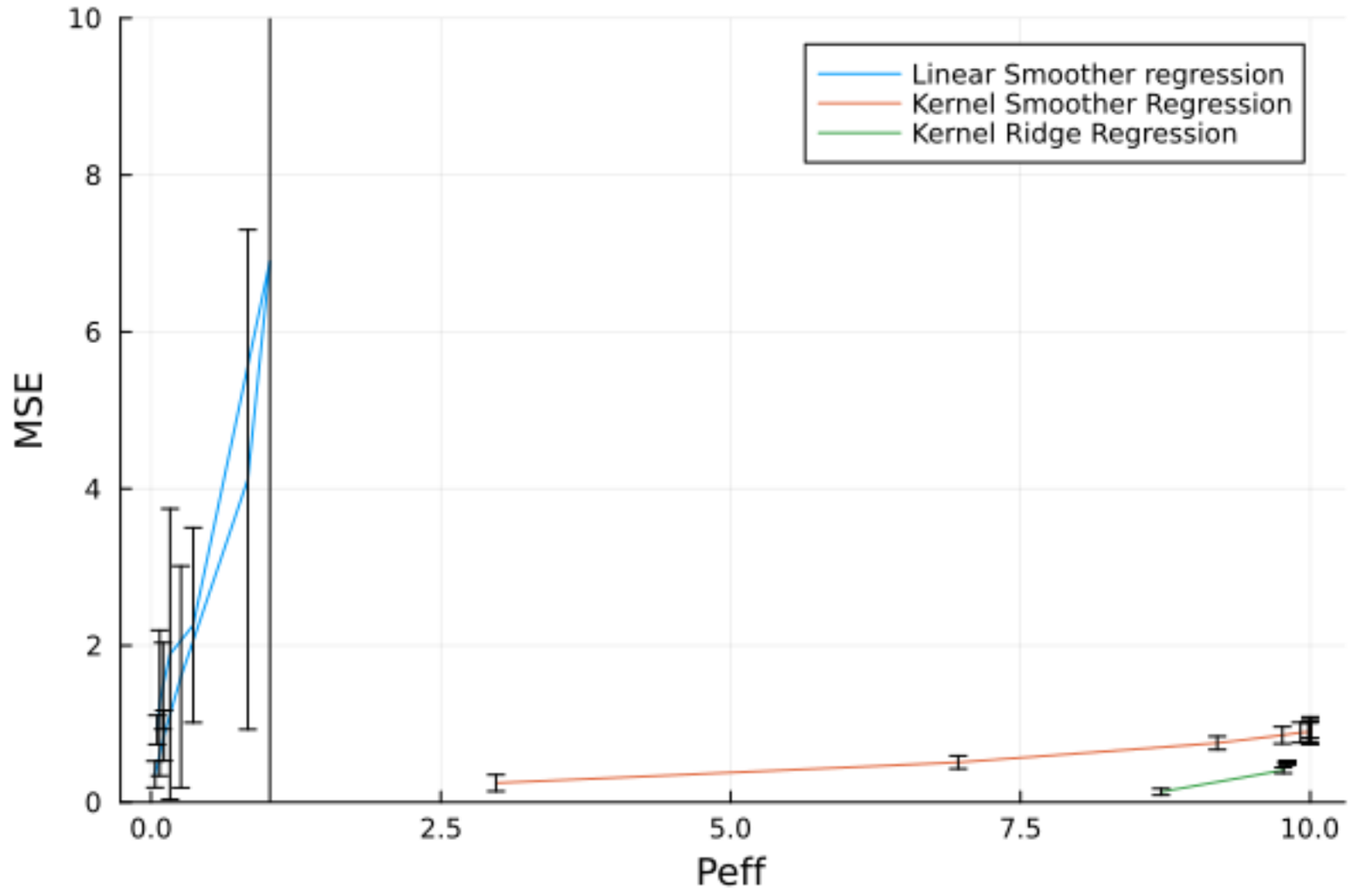
Feature curve



$\sigma = 1,\ \lambda = 0.01$

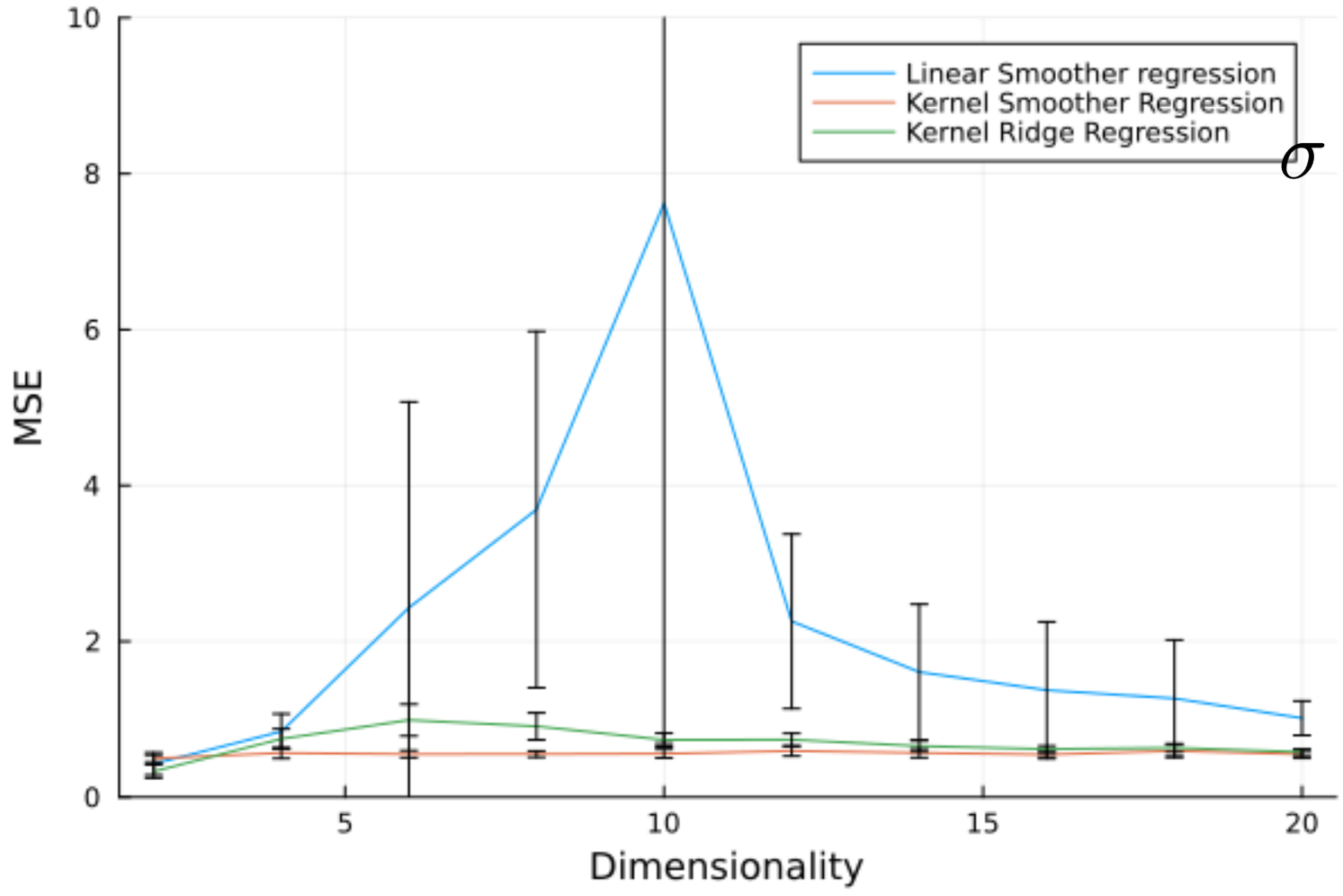
- Nonlinear regression data, N=10 training obj.
- Linear regression shows the peaking phenomenon

Peff as complexity



- Kernel methods have higher effective number of parameters (up to 10)
- Linear regression first increases, then decreases Peff

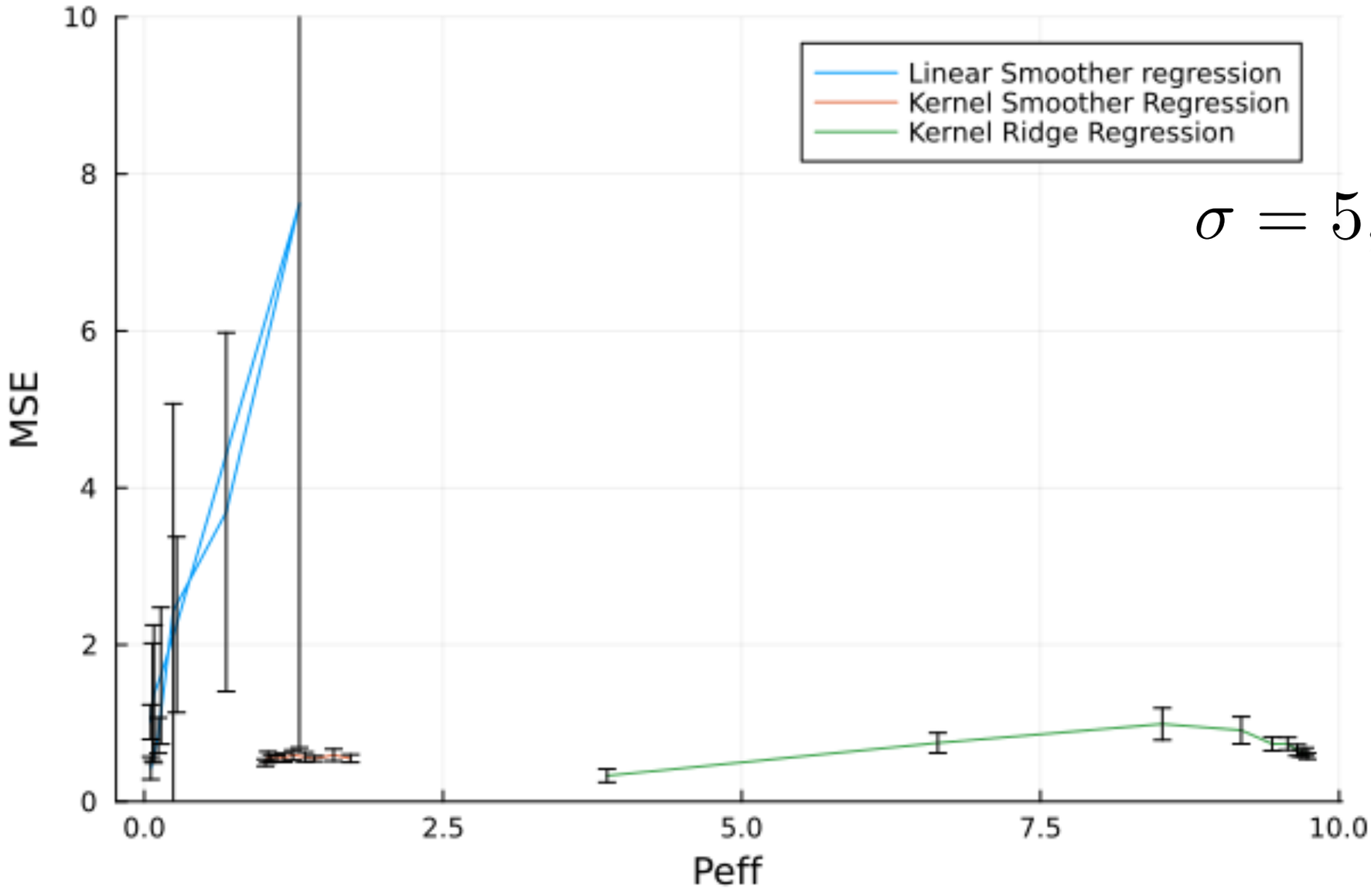
Feature curve



$\sigma = 5,\ \lambda = 0.01$

- Changing the scale influences the performances of kernel methods

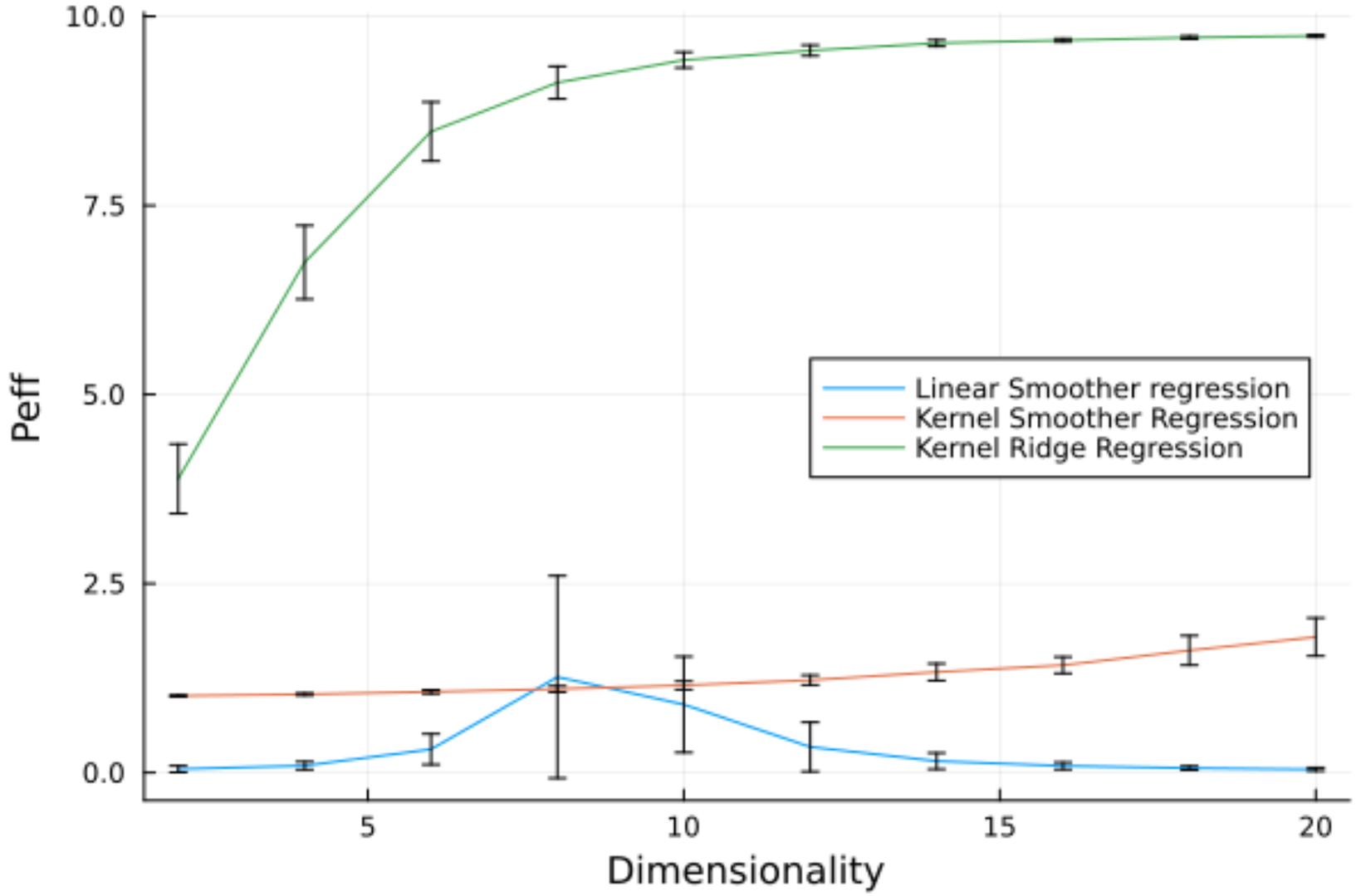
Complexity



$\sigma = 5,\ \lambda = 0.01$

- By increasing sigma, the complexity goes down

Peff vs dimensionality



- Note how Peff increases and decreases for linear regression

Conclusions

- Alternative to Deep Learning, nonlinear Smoothers can be used:
  - Less hyperparameters (typically a scale and a regularization parameter)
  - often interpretable
  - often well performing on smaller datasets
- Drawbacks
  - (Typically) all training data has to be stored
  - Inference time is long or inverse kernel matrix is needed