COMP 551 – Applied Machine Learning
Lecture 20: Gaussian processes

Instructor: Ryan Lowe (ryan.lowe@cs.mcgill.ca)

Slides mostly by: Herke van Hoof (herke.vanhoof@mcgill.ca)

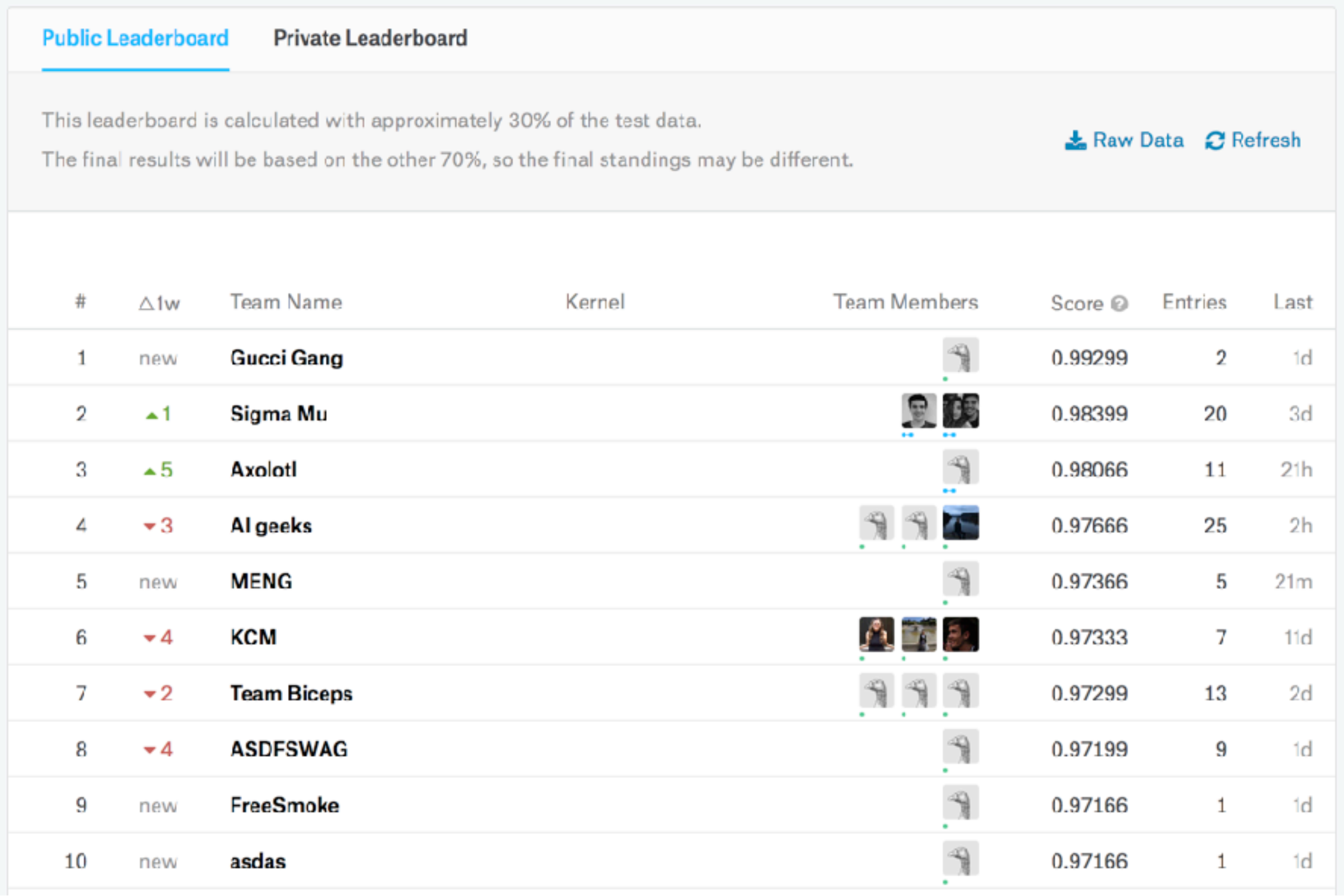
Class web page: *www.cs.mcgill.ca/~hvanho2/comp551*

Announcements

- Change in office hours next week: Wednesday from 11am-12pm, MC 232

- Project 4 Kaggle submission due today!
  - Written report due tomorrow
  - **No hard-copy needs to be submitted!** Just submit on MyCourses
  - # Kaggle submissions increased to 4/day

Announcements



This leaderboard is calculated with approximately 30% of the test data.

The final results will be based on the other 70%, so the final standings may be different.

#	△1w	Team Name	Kernel	Team Members	Score	Entries	Last
1	new	**Gucci Gang**			0.99299	2	1d
2	▲1	**Sigma Mu**			0.98399	20	3d
3	▲5	**Axolotl**			0.98066	11	21h
4	▼3	**AI geeks**			0.97666	25	2h
5	new	**MENG**			0.97366	5	21m
6	▼4	**KCM**			0.97333	7	11d
7	▼2	**Team Biceps**			0.97299	13	2d
8	▼4	**ASDFSWAG**			0.97199	9	1d
9	new	**FreeSmoke**			0.97166	1	1d
10	new	**asdas**			0.97166	1	1d

Beyond linear regression

- Relying on features can be problematic
- We tried to avoid using features before…
  - Lecture 8, instance based learning. Use distances!
  - Lecture 12, support vector machines. Use kernels!

- **<u>This class:</u>** extend regression to nonparametric models
  - *Gaussian processes!*

Recall: Kernels

- A **kernel** is a function of two arguments which corresponds to a dot product in some feature space

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

- Advantage of using kernels:
  - Sometimes evaluating *k()* is cheaper than evaluating features and taking the dot product $k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j)^d$
  - Sometimes *k()* corresponds to an inner product in a feature space with infinite dimensions $k(\mathbf{x}_i, \mathbf{x}_j) = \exp -\|\mathbf{x}_i - \mathbf{x}_j\|^2$

Recall: Kernels

- Kernelize algorithm:
  - Try to formulate algorithm so feature vectors only ever occur in inner products
  - Replace inner products by kernel evaluations **(kernel trick)**

Recall: kernel regression

- Given dataset, how do we calculate *y* value for new input?
- Regression: learn weighted function of features *y* = ***w***^*T x*
- Kernel regression: **don't learn any parameters**!
- Instead, **use *y*'s of neighbouring data points**!!

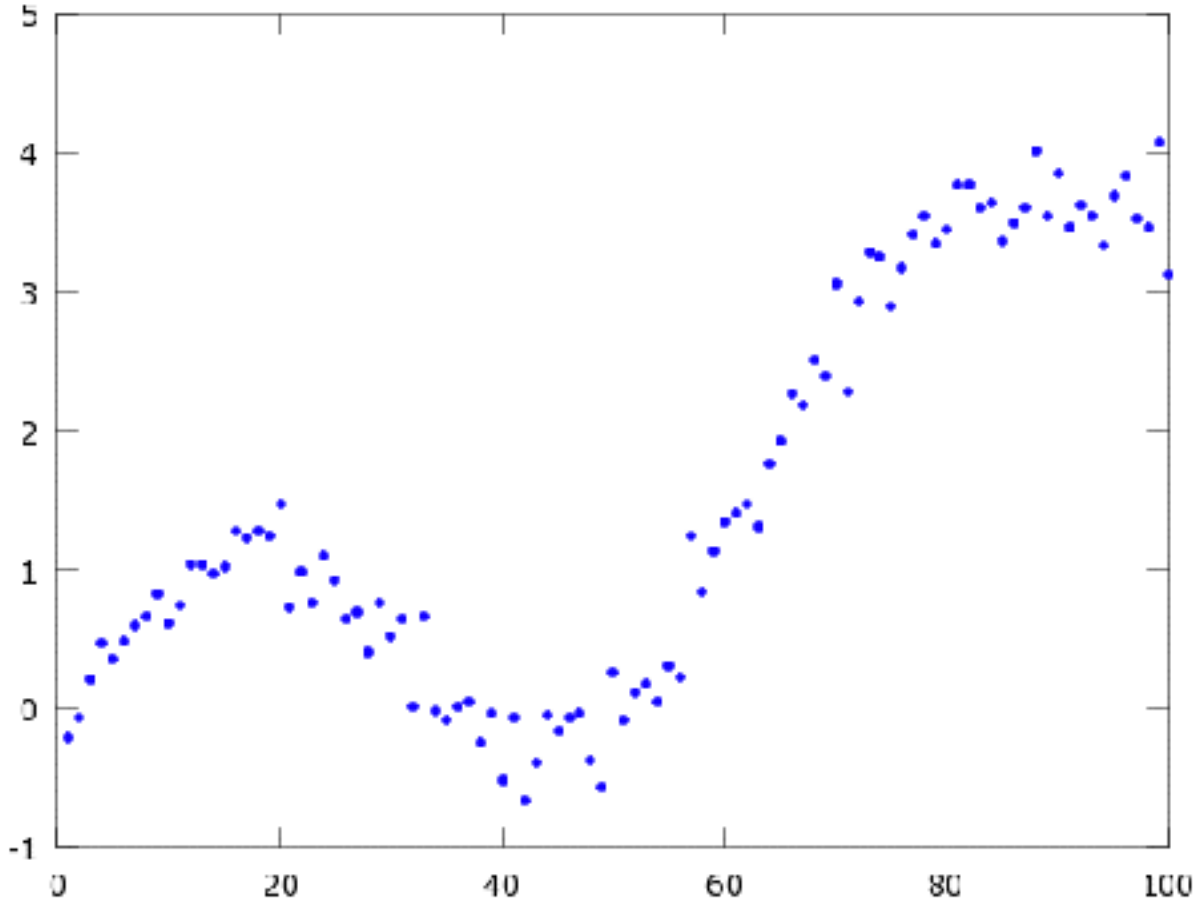


Image source: http://mccormickml.com/

Recall: kernel regression

- What *y* should we predict for *x*=60?
- Could e.g. take an average of surrounding y values
- <u>In general:</u> calculate a **weight** for how much each data point contributes, take *weighted average*
- The **kernel is just a weighting function**

$$y^* = \frac{\sum_{i=1}^{m}(K(x^*, x_i)y_i)}{\sum_{i=1}^{m} K(x^*, x_i)}$$

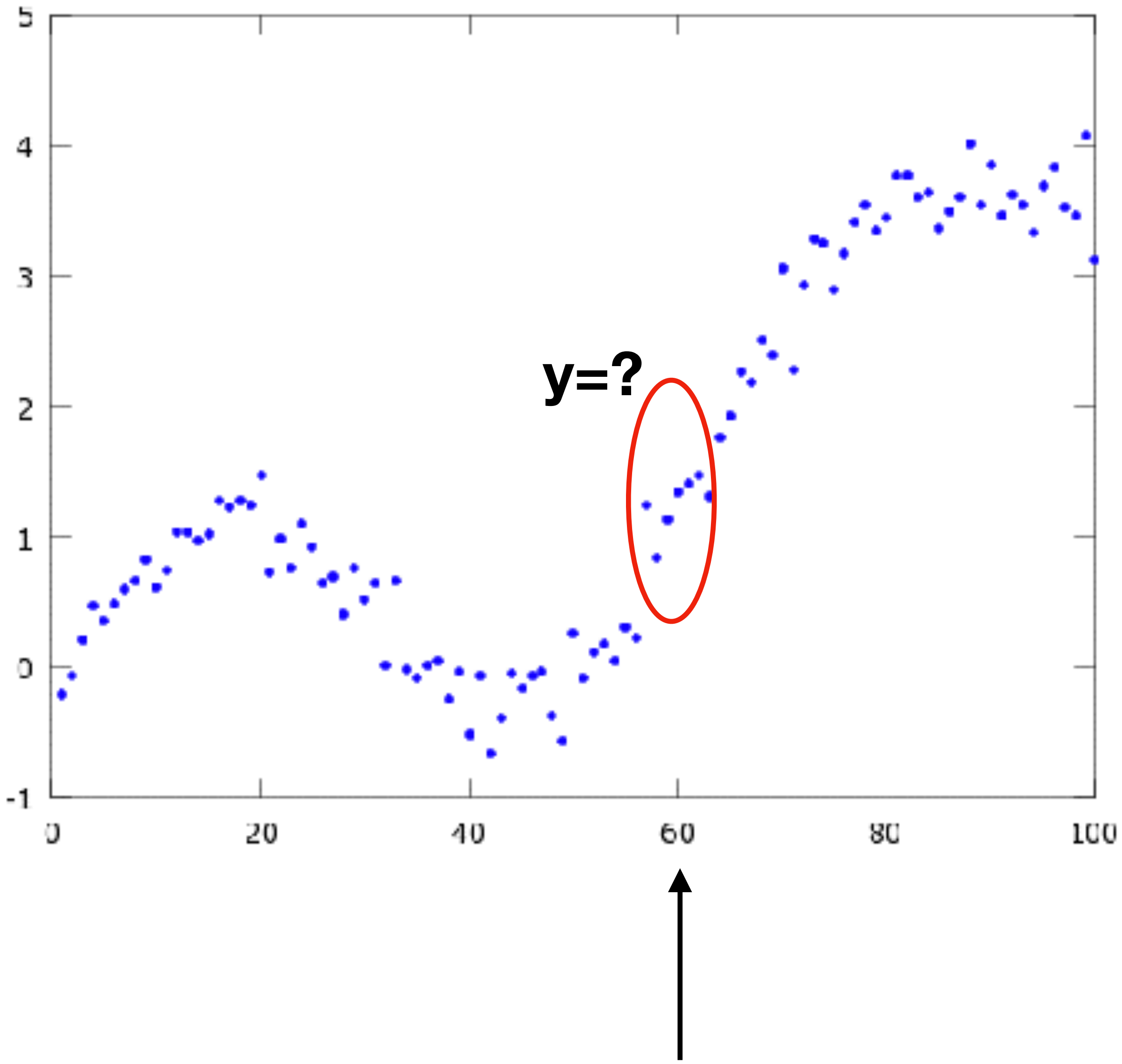
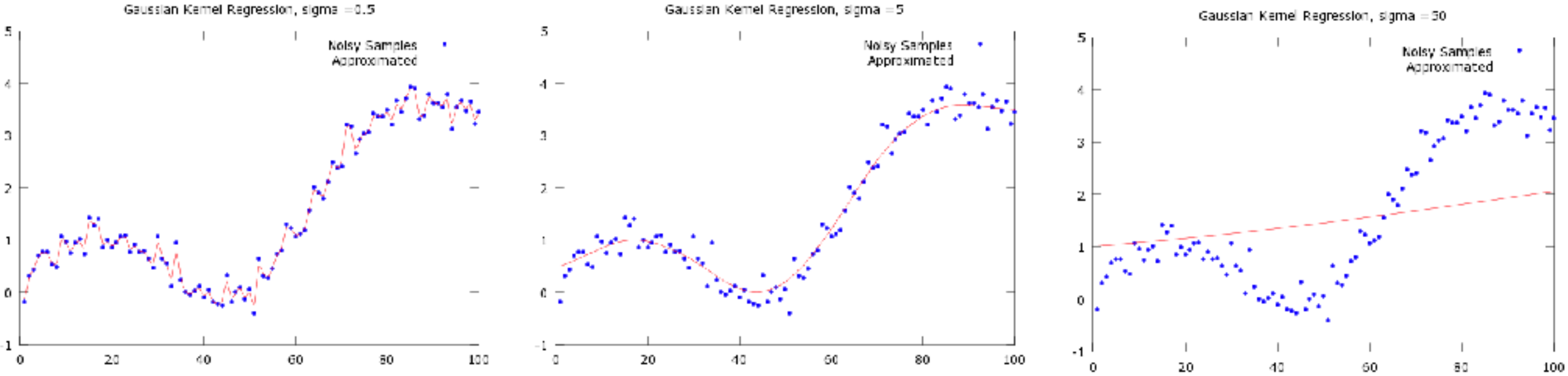


Image source: http://mccormickml.com/

Recall: kernel regression

- Common kernel is Gaussian: $K(x^*, x_i) = e^{-\frac{(x_i - x^*)^2}{2\sigma^2}}$
- *Points nearby contribute more, points further away contribute less*
- Variance controls how many neighbouring points are used



- Higher sigma -> smoother function

Image source: http://mccormickml.com/

Recall: Kernel regression

- Kernel regression is **non-parametric**: no parameters are explicitly learned, just use nearby datapoint to make predictions

- Kernel can be thought of as a *‘distance measure’*, defining which points are considered ‘nearby’ for each input

- We kernelized linear regression — can we kernelize Bayesian linear regression?
  - **Start with just the mean**

Kernelizing the mean function

- Inspect solution mean from Bayesian linear regression

(from last class)

$$p(y^*|\mathcal{D}) = \mathcal{N}(\boxed{\sigma^{-2}\mathbf{x}^{*T}\mathbf{S}_N\mathbf{X}^T\mathbf{y}}, \sigma^2 + \mathbf{x}^T\mathbf{S}_N\mathbf{x}) \tag{1}$$

$$\mathbf{S}_N = (\alpha\mathbf{I} + \sigma^{-2}\mathbf{X}^T\mathbf{X})^{-1} \tag{2}$$

- Vector $\mathbf{y}$ concatenates training outputs
- Matrix $\mathbf{X}$ has one column for each feature (length N)

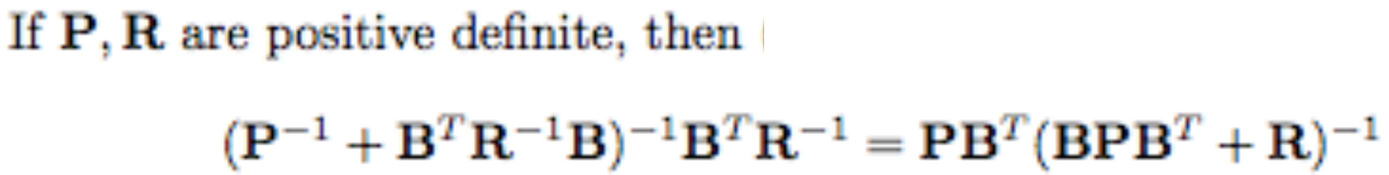
  one row for each datapoint (length M)
- Mean prediction is mean of the Gaussian:

$$y^* = \sigma^{-2}\mathbf{x}^{*T}(\alpha\mathbf{I} + \sigma^{-2}\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Kernelizing the mean function

- Step 2: Reformulate to only have inner products of features

$$y^* = \sigma^{-2}\mathbf{x}^{*T}(\alpha\mathbf{I} + \sigma^{-2}\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$



If $\mathbf{P}, \mathbf{R}$ are positive definite, then

$$(\mathbf{P}^{-1} + \mathbf{B}^T\mathbf{R}^{-1}\mathbf{B})^{-1}\mathbf{B}^T\mathbf{R}^{-1} = \mathbf{P}\mathbf{B}^T(\mathbf{B}\mathbf{P}\mathbf{B}^T + \mathbf{R})^{-1}$$

$$y^* = \underbrace{\sigma^{-2}\mathbf{x}^{*T}\mathbf{X}^T}_{\mathbf{k}(\mathbf{x}^*)^T}(\alpha\mathbf{I} + \underbrace{\sigma^{-2}\mathbf{X}\mathbf{X}^T}_{\mathbf{K}})^{-1}\mathbf{y}$$

element i,j of this matrix is $\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$

element i of this vector is $\phi(\mathbf{x}_i)^T\phi(\mathbf{x}^*)$

Kernelizing the mean function

- Step 2: Reformulate to only have inner products of features

$$y^* = \sigma^{-2}\mathbf{x}^{*T}(\alpha\mathbf{I} + \sigma^{-2}\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

# features x #features

# datapoints x #datapoints

$$y^* = \underbrace{\sigma^{-2}\mathbf{x}^{*T}\mathbf{X}^T}_{\mathbf{k}(\mathbf{x}^*)^T}(\alpha\mathbf{I} + \underbrace{\sigma^{-2}\mathbf{X}\mathbf{X}^T}_{\mathbf{K}})^{-1}\mathbf{y}$$

element i,j of this matrix is $\phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$

element i of this vector is $\phi(\mathbf{x}_i)^T\phi(\mathbf{x}^*)$

Kernelizing the mean function

- Step 3: Replace inner products by kernel evaluations

$$y^* = \mathbf{k}(\mathbf{x}^*)^T(\alpha\mathbf{I} + \mathbf{K})^{-1}\mathbf{y}$$

element i,j of this matrix is $k(\mathbf{x}_i, \mathbf{x}_j)$

element i of this vector is $k(\mathbf{x}_i, \mathbf{x}^*)$

- Remember: Mean function is same as ridge regression
- This is **kernel** ridge regression

Recall: Ridge regression

Bayesian linear regression:

$$\max -\frac{\sigma^{-2}}{2}\sum_{n=1}^{N}(y_n - \mathbf{w}^T\mathbf{x}_n)^2 - \frac{\alpha}{2}\mathbf{w}^T\mathbf{w} + \text{const.}$$

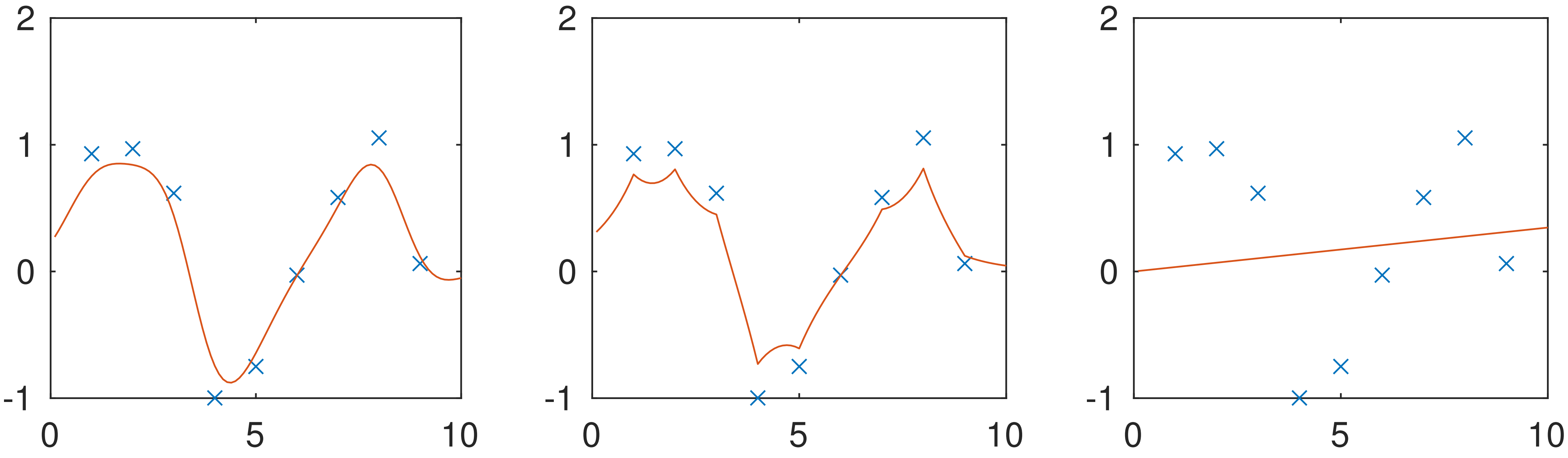
Ridge regression:

$$\min \sum_{n=1}^{N}(y_n - \mathbf{w}^T\mathbf{x}_n)^2 + \lambda\mathbf{w}^T\mathbf{w}$$

- Difference between the two? Bayesian linear regression *learns a distribution over parameters*
- So kernelized mean prediction with Bayesian linear regression <=> kernel ridge regression, $\lambda = \alpha\sigma^2$

Kernel ridge regression

- Choosing a kernel:



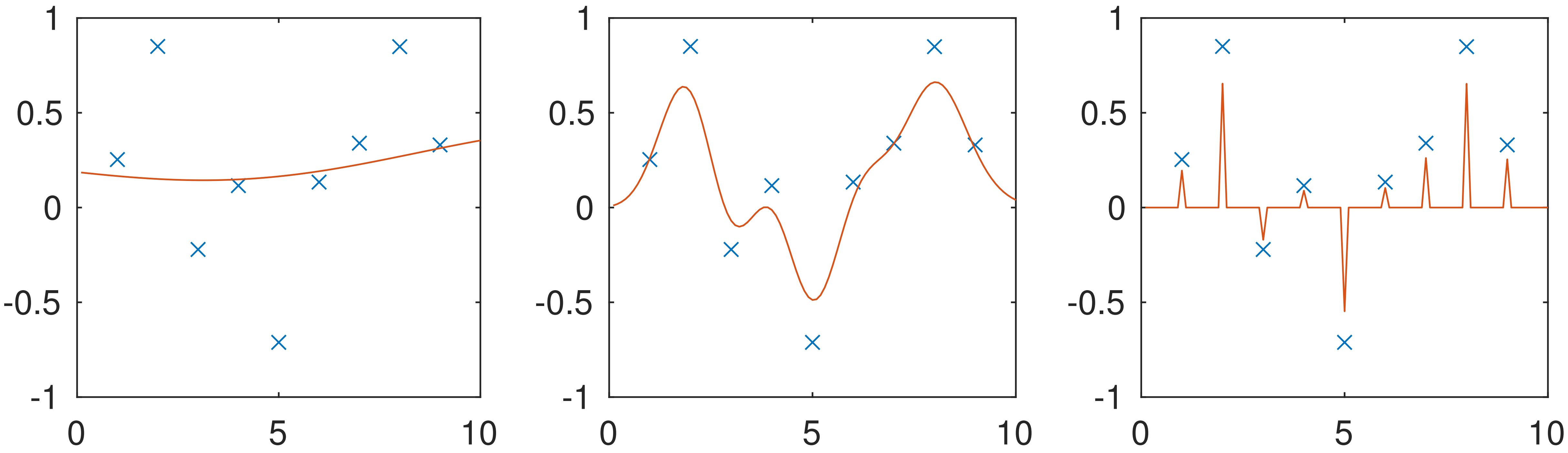
$$k(x_i, x_j) = \exp - \frac{\|x_i - x_j\|^2}{2\sigma^2}$$

$$k(x_i, x_j) = x_i x_j$$

$$k(x_i, x_j) = \exp - |x_i - x_j|$$

Kernel ridge regression

- Setting parameters: sigma controls what data points are ‘close’

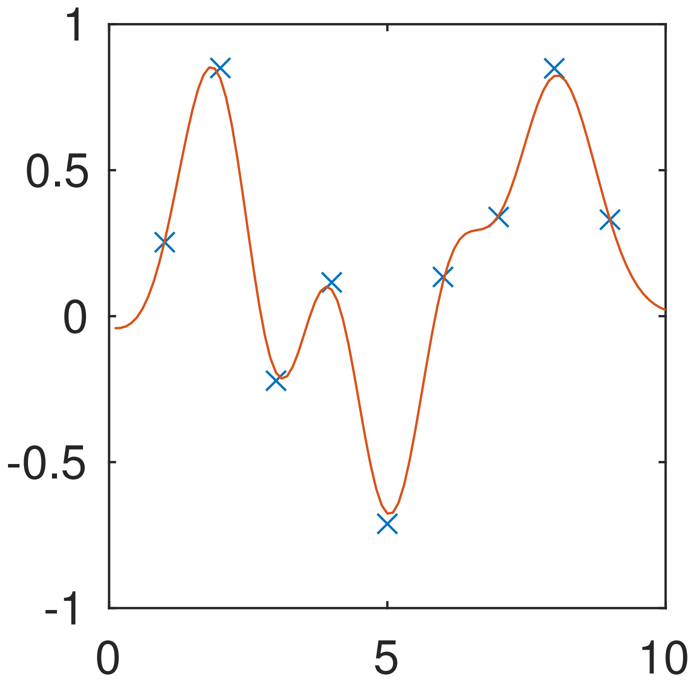


$$\sigma = 10 \qquad \sigma = 1 \qquad \sigma = 0.1$$

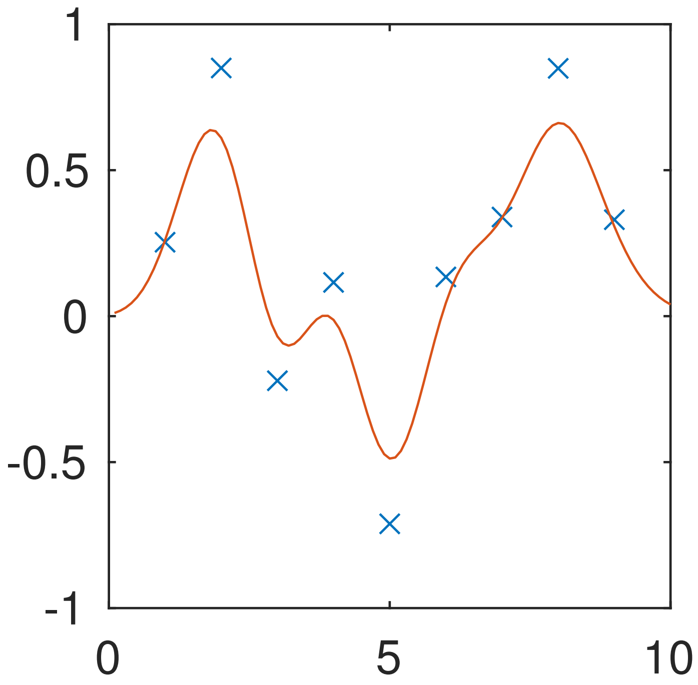
$$k(x_i, x_j) = \exp - \frac{\|x_i - x_j\|^2}{2\sigma^2}$$

Kernel ridge regression

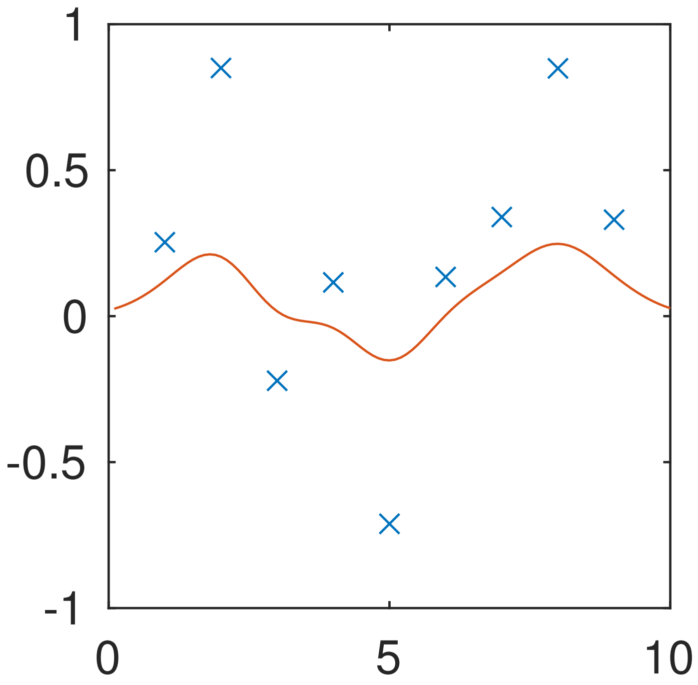
- Setting parameters: alpha controls ‘smoothness’



Small alpha



Medium alpha

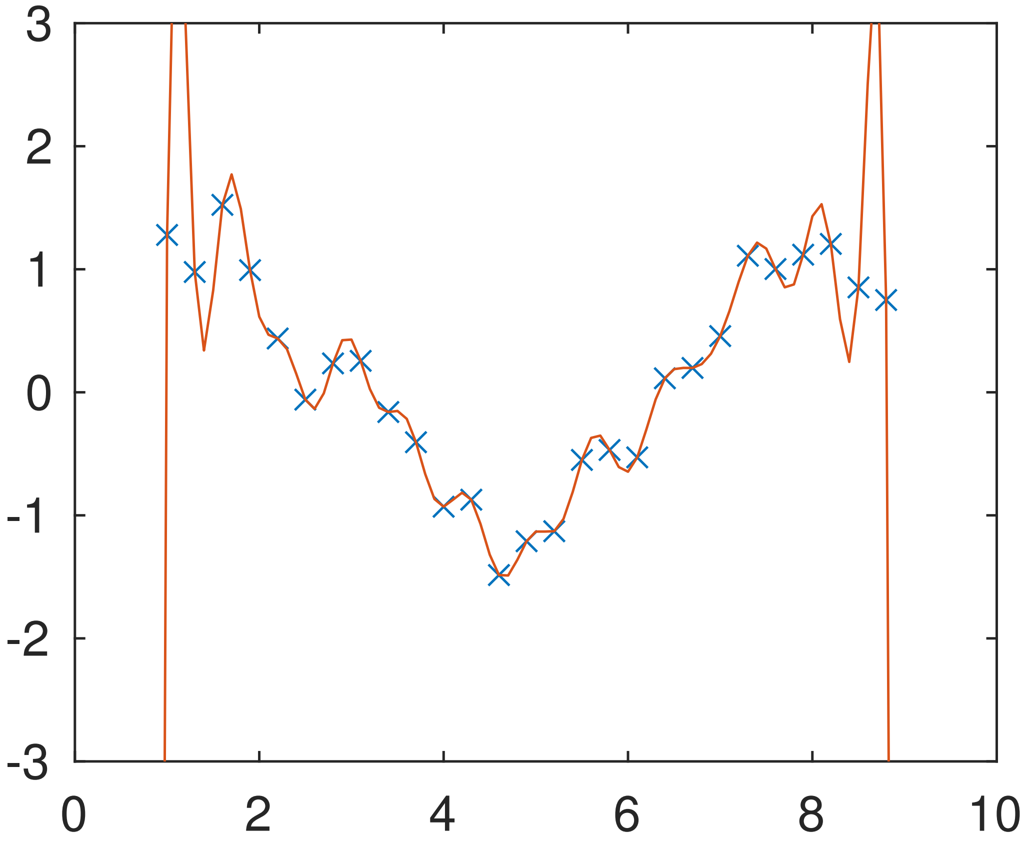


Large alpha

$$y^* = \mathbf{k}(\mathbf{x}^*)^T(\alpha\mathbf{I} + \mathbf{K})^{-1}\mathbf{y}$$

Why add the 'ridge'?

- As before, kernel regression can easily overfit: *regularisation is critical!*



alpha=0 *(aka kernel regression)*

Kernel regression: Practical issues

- Compare ridge regression: $\mathbf{w} = (\lambda \mathbf{I} + \mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$

  inverse $O(d^3)$ matrix-vector product $O(d^2 N)$

  prediction $O(d)$

  memory $O(d)$

- Kernel ridge regression: $y^* = \mathbf{k}(\mathbf{x}^*)^T (\alpha \mathbf{I} + \mathbf{K})^{-1} \mathbf{y}$

  inverse, product $O(N^3)$

  prediction $O(N)$

  memory $O(N)$

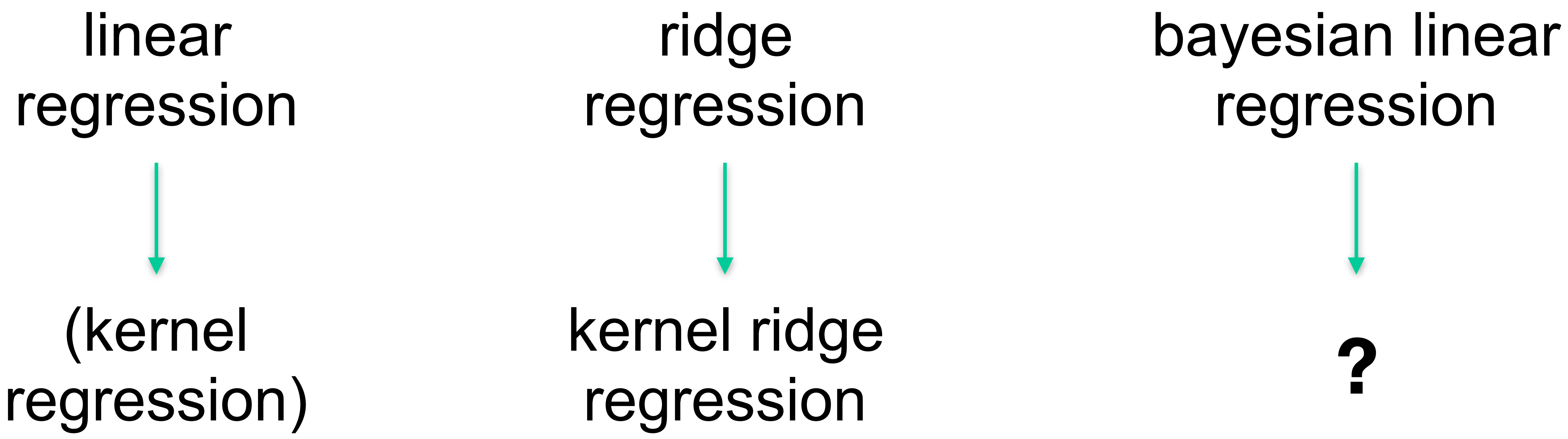
d = feature dimension
N = # datapoints

Kernel regression: Practical issues

- If we have a **small set** of **good features** it’s faster to do regression in feature space
- However, if no good features are available (or we need a very big set of features), kernel regression might yield better results
- Often, it is easier to pick a kernel than to choose a good set of features

Kernelizing Bayesian linear regression

- We have now kernelized ridge regression
- Can we kernelize Bayesian linear regression, too?
  - i.e. **can we kernelize the covariance / uncertainty?**



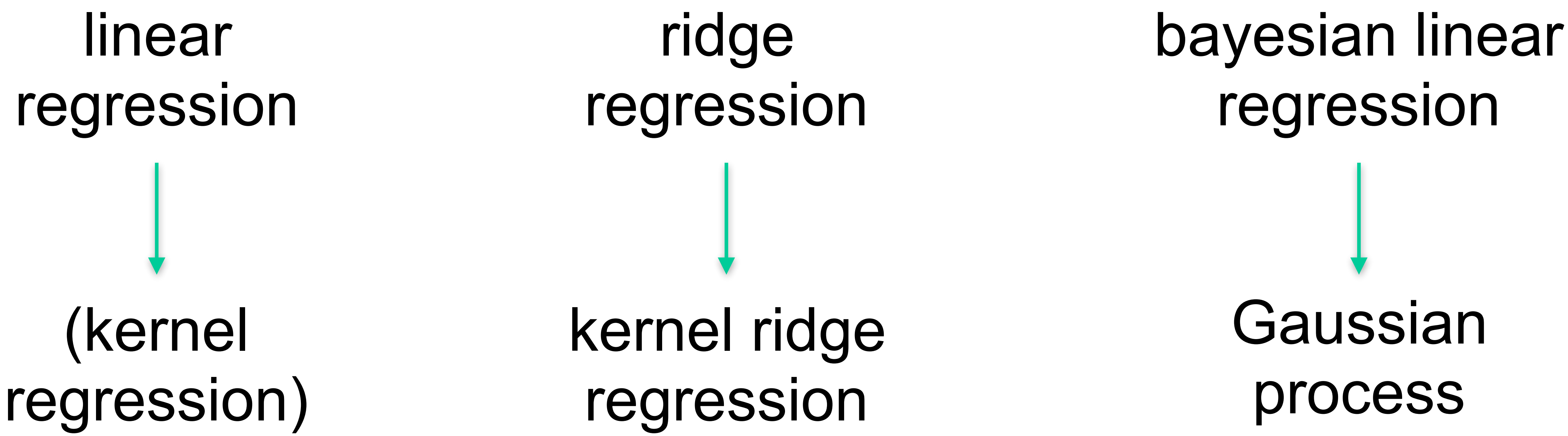
linear regression → (kernel regression)

ridge regression → kernel ridge regression

bayesian linear regression → **?**

Kernelizing Bayesian linear regression

- We have now kernelized ridge regression
- Can we kernelize Bayesian linear regression, too?
  - i.e. can we kernelize the covariance / uncertainty?
- Yes, and this is called **Gaussian process regression** (GPR)



linear regression → (kernel regression)

ridge regression → kernel ridge regression

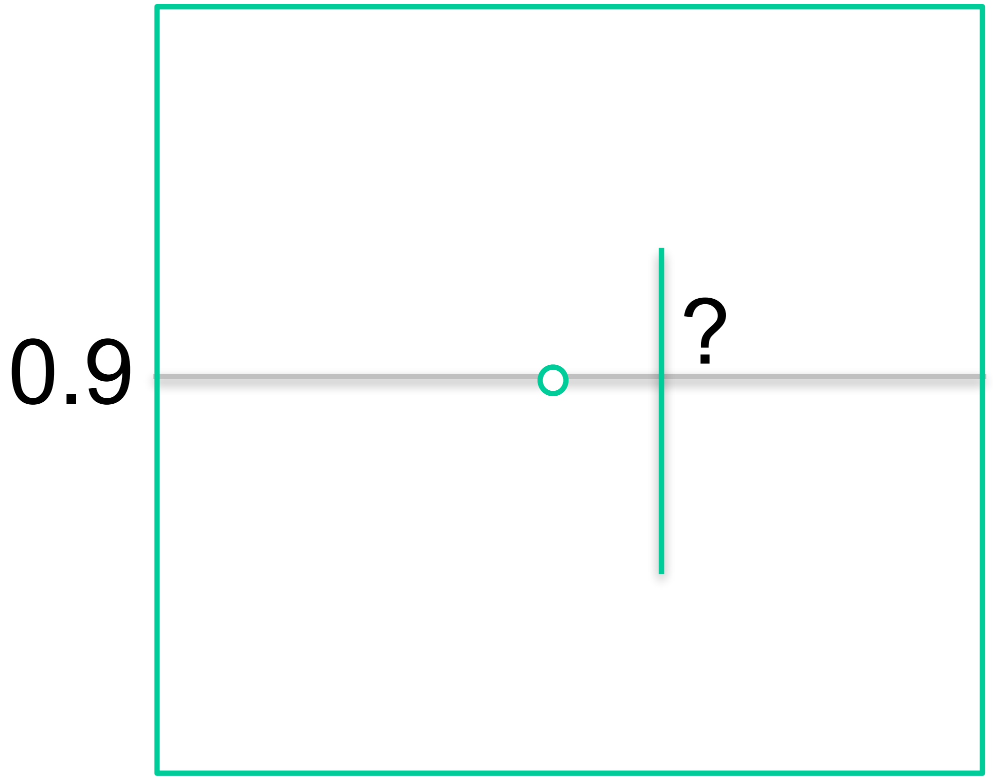
bayesian linear regression → Gaussian process

Gaussian processes: high level

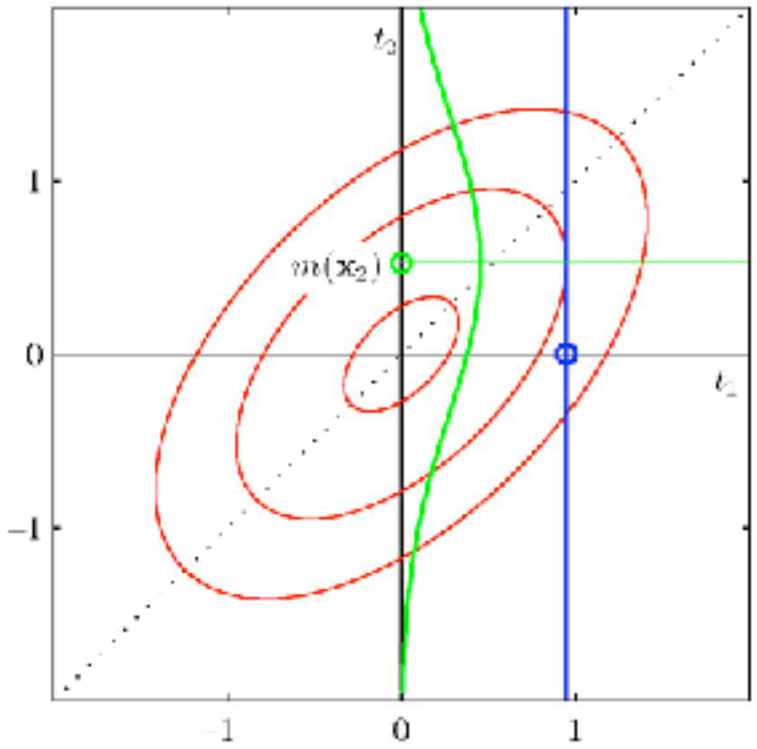
- GPs are defined by a mean function, *and* a covariance function
- Mean function derived in the same way as kernel ridge regression (based on surrounding data points)
- Covariance defined by the kernel: Cov$(f(x), f(x')) = k(x,x')$
- Bayesian method — need to specify prior distribution

Gaussian processes

- Mean function derived already, variance can be similarly derived
- Formal definition: a function *f* is a GP if any finite set of values *f(x_1), …, f(x_n)* follows a *multivariate Gaussian distribution*



Assumption: outputs are correlated



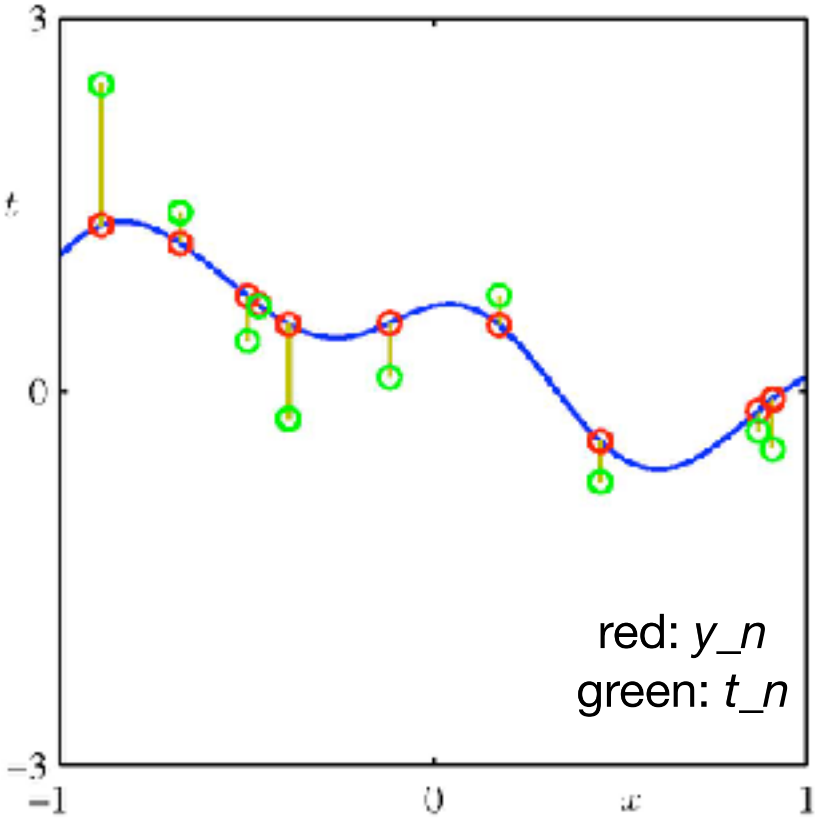
Copyright C.M. Bishop, PRML

Deriving GP equations

- Model:
  - We are interested in the function values $y_1, y_2, . .$, at a set of points $\mathbf{x}_1, \mathbf{x}_2, \ldots.$. We observe target values *t* for the training set, but we assume these are noisy $t_n = y_n + \epsilon$
  - Prior: $\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \mathbf{K})$

    With **y** a vector of function values and **K** the kernel matrix
  - Likelihood (Gaussian noise on output):

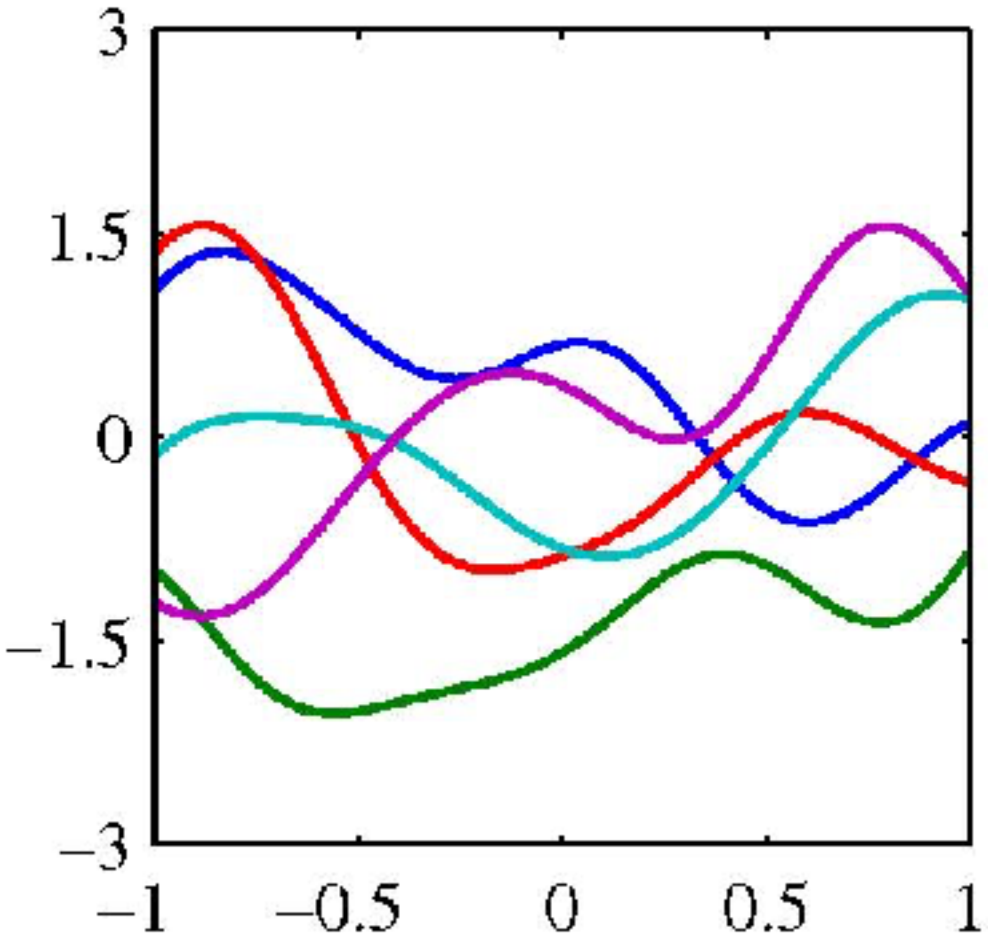
$$\mathbf{t} \sim \mathcal{N}(\mathbf{y}, \beta^{-1}\mathbf{I})$$

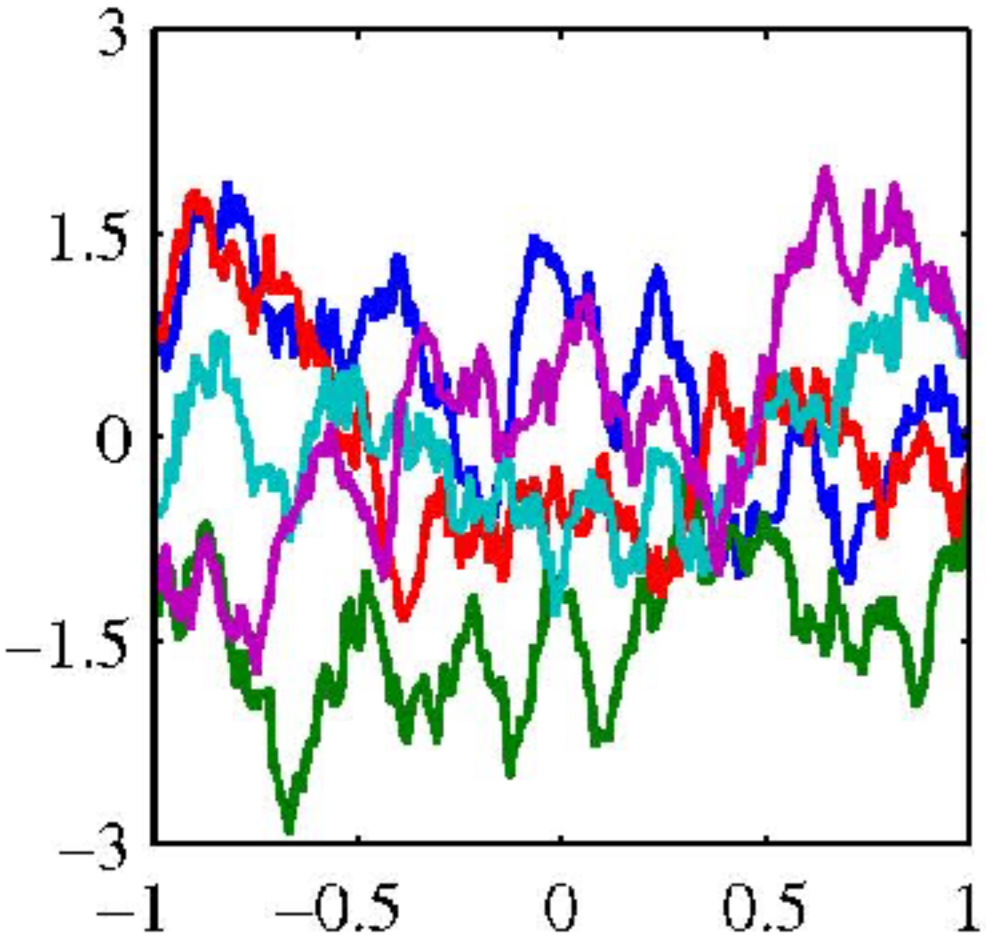


Copyright C.M. Bishop, PRML

Examples from the prior

$$\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \mathbf{K})$$





$$k(x,y) = \exp -(x-y)^2$$

$$k(x,y) = \exp -|x-y|$$

GP Regression

- Prior and likelihood are Gaussian
- Again obtain a closed form solution

$$\mathbb{E}[y^*] = \mathbf{y}^T(\mathbf{K} + \beta^{-1}\mathbf{I})^{-1}\mathbf{k}(\mathbf{x}^*)$$

kernel ridge regression

$$\mathrm{Cov}[y^*] = k(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}(\mathbf{x}^*)^T(\mathbf{K} + \beta^{-1}\mathbf{I})^{-1}\mathbf{k}(\mathbf{x}^*)$$

prior variance

reduction in variance due to close training points

- Prediction of new observations

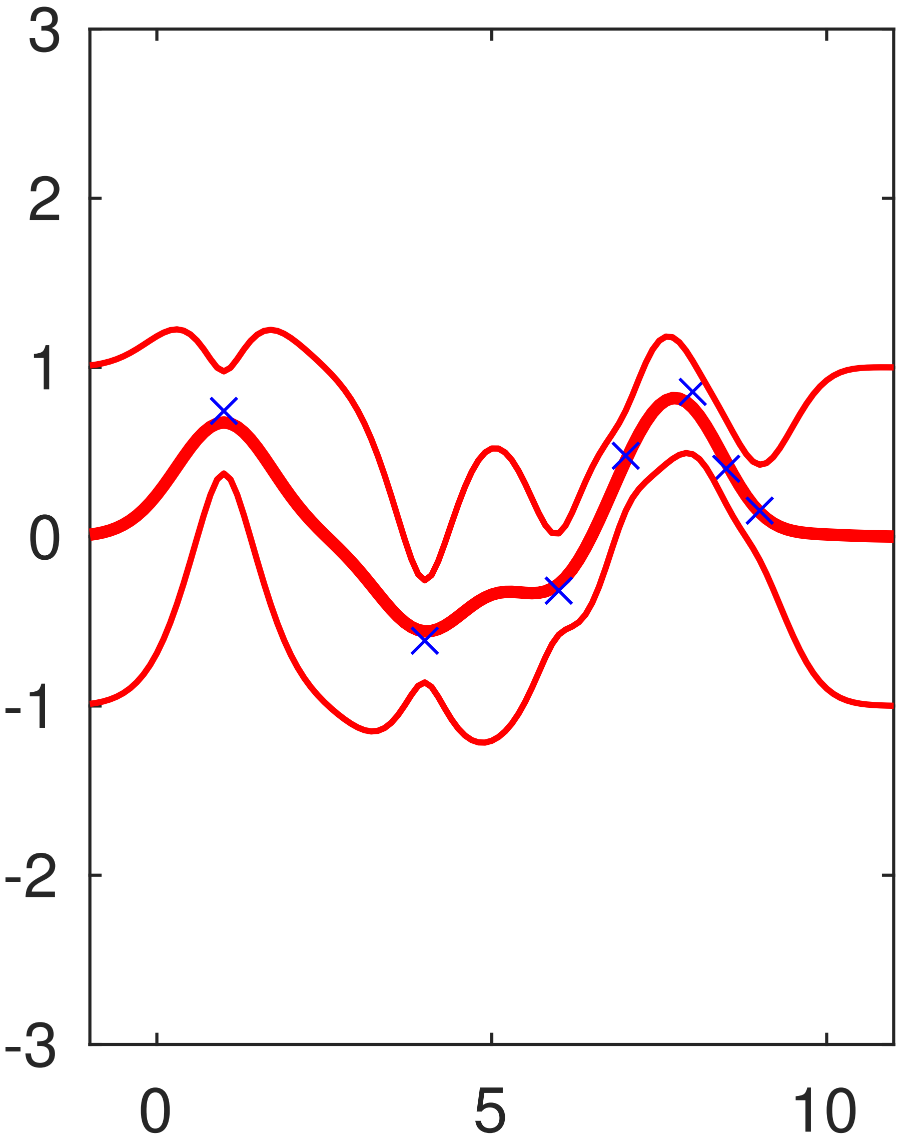
$$\mathrm{Cov}[t^*] = k(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}(\mathbf{x}^*)^T(\mathbf{K} + \beta^{-1}\mathbf{I})^{-1}\mathbf{k}(\mathbf{x}^*) + \beta^{-1}$$

- Easy to implement!

add noise term

GP Regression

- Results of GP regression



$$\mathbb{E}[y^*|\mathbf{t}] + \sqrt{\mathrm{Cov}[y^*|\mathbf{t}]}$$

$$\mathbb{E}[y^*|\mathbf{t}]$$

$$\mathbb{E}[y^*|\mathbf{t}] - \sqrt{\mathrm{Cov}[y^*|\mathbf{t}]}$$

Calculated for many possible y^*

t: set of observed points

GP Regression: hyperparameters

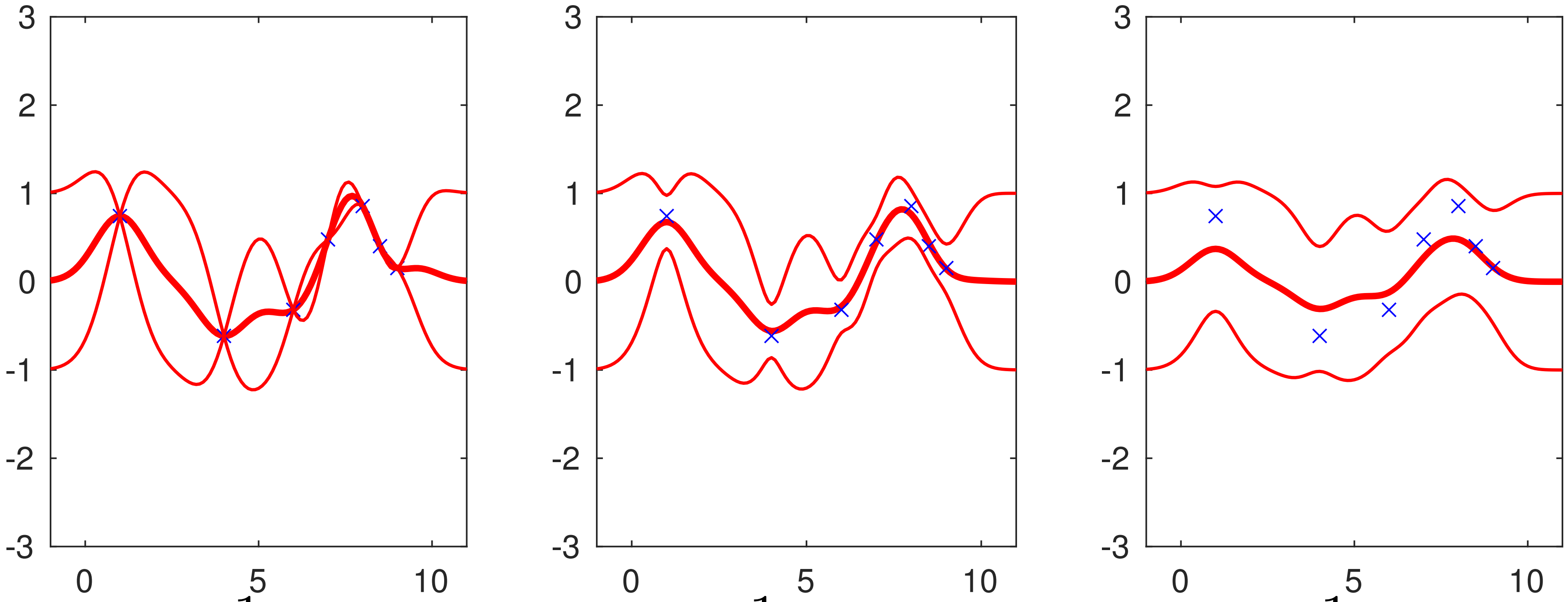
- Hyperparameters
  - Assumed noise (variance of likelihood) $\mathbf{t} \sim \mathcal{N}(\mathbf{y}, \beta^{-1}\mathbf{I})$
  - Any parameters of the kernel
    - Typical kernel:

$$k(\mathbf{x}_i, \mathbf{x}_j) = s^2 \exp - \frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}$$

      - s: scale (standard deviation prior to seeing data)
      - σ : bandwidth (which datapoint are considered close)
  - Effective regularisation: $\beta^{-1} s^{-1}$
  - Knowing the 'meaning' of parameters helps tune them

GP Regression: hyperparameters

- Assumed noise (variance of likelihood) $\mathbf{t} \sim \mathcal{N}(\mathbf{y}, \beta^{-1}\mathbf{I})$
- Effective regularisation: $\beta^{-1}s^{-1}$

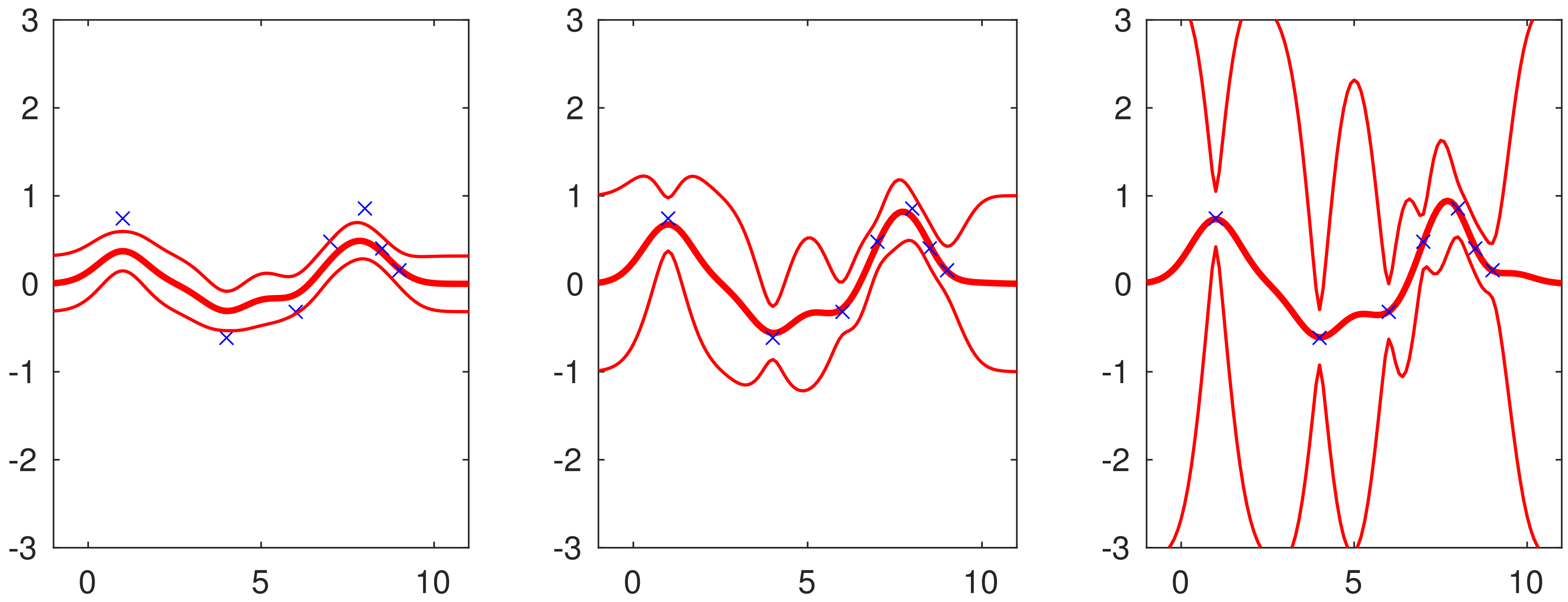


$\beta^{-1} = 0$ $\beta^{-1} = 0.1$ $\beta^{-1} = 1$

- Mostly changes behaviour close to train points

GP Regression: hyperparameters

- Kernel $k(\mathbf{x}_i, \mathbf{x}_j) = s^2 \exp - \frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}$
- Effective regularisation $\beta^{-1} s^{-1}$

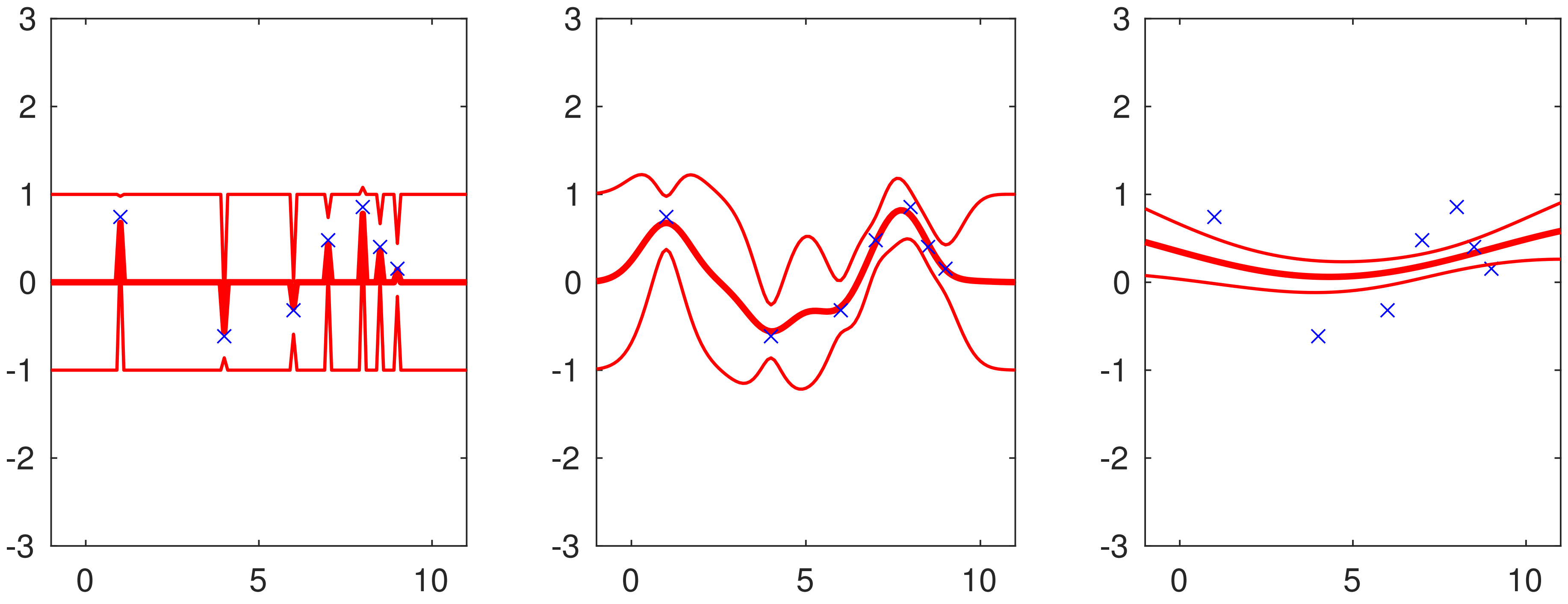


$s = 0.1$ $s = 1$ $s = 10$

- Mostly changes behaviour further away from training points

GP Regression: hyperparameters

- Kernel $$k(\mathbf{x}_i, \mathbf{x}_j) = s^2 \exp - \frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}$$



$\sigma = 0.1$ $\sigma = 1$ $\sigma = 10$

- Changes what is considered ‘close’ or ‘far’

GPs: Practical issues

- Complexity pretty much similar to kernel regression
- Except for calculating predictive variance

$$\mathbb{E}[y^*] = \mathbf{y}^T(\mathbf{K} + \beta^{-1}\mathbf{I})^{-1}\mathbf{k}(\mathbf{x}^*)$$

$$\mathrm{Cov}[y^*] = k(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}(\mathbf{x}^*)^T(\mathbf{K} + \beta^{-1}\mathbf{I})^{-1}\mathbf{k}(\mathbf{x}^*)$$

- inverse, product $O(N^3)$
- prediction $\xcancel{O(N)}$ $O(N^2)$
- memory $O(N)$

GPs: Practical issues

- For small dataset, GPR is a state-of-the-art method!
  - Advantage: provides uncertainty, flexible yet can control overfitting
  - Computational costs acceptable for small datasets (<10 000)
  - Has been applied to robotics & control, hyperparameter optimization, MRI data, weather prediction, …
- For large datasets, uncertainty not as important, GPs are expensive
- Good approximations exist
- Specifying the right prior (kernel!) is important!

More resources on GPs

- Lectures by Nando de Freitas:
  - https://www.youtube.com/watch?v=4vGiHC35j9s&t=0s&index=8&list=PLE6Wd9FR--EdyJ5lbFl8UuGjecvVw66F6
- ‘Gaussian processes for dummies’
  - http://katbailey.github.io/post/gaussian-processes-for-dummies/
- Gaussian processes textbook
  - http://www.gaussianprocess.org/gpml/ (free download)

Bayesian methods in practice

- Time complexity varies compared to frequentist methods
- Memory complexity can be higher
    - e.g. need to store mean + uncertainty : quadratic, not linear
- Lots of data everywhere: posterior close to point estimate
    - (might as well use frequentist methods)
- Little data everywhere
    - Prior information helps bias/variance trade-off
- Some areas with little data, some areas with lots of data
    - Uncertainty helps to decide where predictions are reliable

Inference in more complex models

- We saw some examples with closed-form posterior
- In many complex models, no closed-form representation
- **Variational inference** (deterministic)
  - Consider family of distributions we **can** represent (Gaussian)
  - Use optimisation techniques to find best of these
- **Sampling** (stochastic)
  - Try to directly sample from the posterior
  - Expectations can be approximated using the samples
- **Maximum a posterior** (point estimate)

What you should know

- Previous lectures:
  - What is the Bayesian view of probability?
  - Why can the Bayesian view be beneficial?
  - Role of the following distributions:
    - Likelihood, prior, posterior, posterior predictive
  - Key idea of Bayesian regression and its properties
- This lecture:
  - Key idea of kernel regression and its properties
  - Main idea behind Gaussian process regression