Machine Learning

Volker Roth

Department of Mathematics & Computer Science
University of Basel

Section 7

Support Vector Machines and Kernels

Structure on canonical hyperplanes

Theorem (Vapnik, 1982)

Let R be the radius of the smallest ball containing the points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$: $B_R(\boldsymbol{a}) = \{\boldsymbol{x} \in \mathbb{R}^d : \|\boldsymbol{x} - \boldsymbol{a}\| < R, \boldsymbol{a} \in \mathbb{R}^d\}$. The set of canonical hyperplane decision functions $f(\boldsymbol{w}, w_0) = sign\{\boldsymbol{w}^t \boldsymbol{x} + w_0\}$ satisfying $\|\boldsymbol{w}\| \leq A$ has VC dimension h bounded by

$$h \leq R^2 A^2 + 1.$$

Intuitive interpretation: $\text{margin} = 1/\|\boldsymbol{w}\|$

$\rightsquigarrow$ **minimizing capacity($\mathcal{H}$) corresponds to maximizing the margin.**

$$R[f_n] \leq R_{\text{emp}}[f_n] + \sqrt{\frac{a}{n}\left(\text{capacity}(\mathcal{H}) + \ln\frac{b}{\delta}\right)}$$

$\rightsquigarrow$ **Large margin classifiers.**

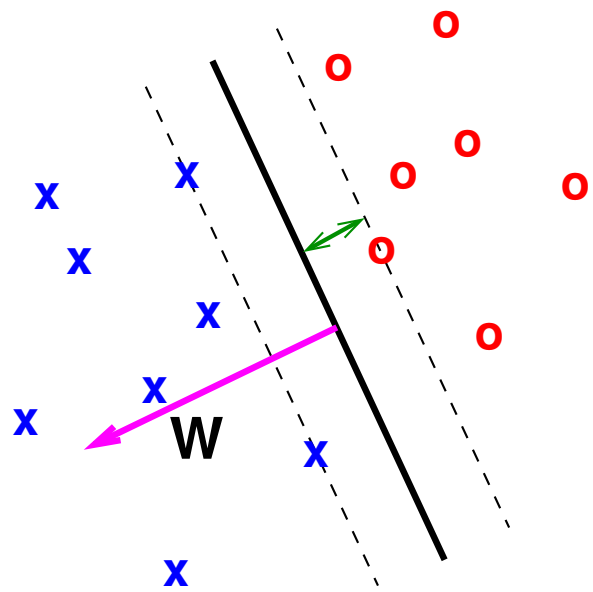
SVMs

- When the training examples are **linearly separable** we can maximize the margin by minimizing the regularization term

$$\|\boldsymbol{w}\|^2/2 = \sum_{i=1}^{d} w_i^2/2$$

subject to the **classification constraints**

$$y_i[\boldsymbol{x}_i^t \boldsymbol{w}] - 1 \geq 0, \ i = 1, \ldots, n.$$



- The solution is defined only on the basis of a subset of examples or **support vectors.**

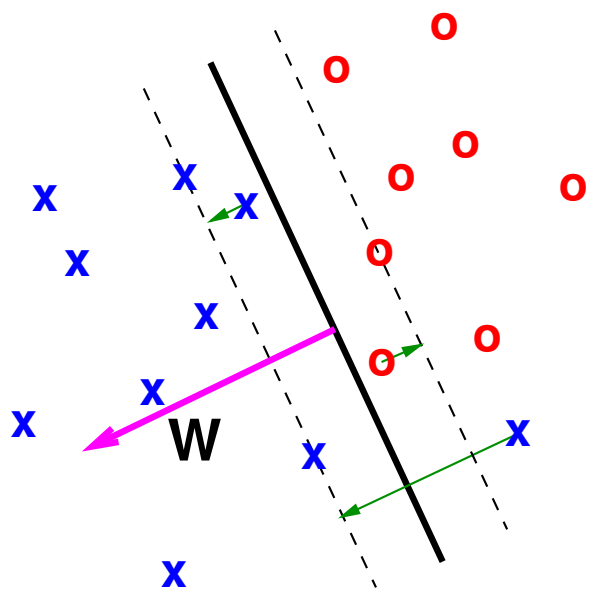
SVMs: nonseparable case

- Modify optimization problem slightly by adding a **penalty for violating the classification constraints:**

$$\text{minimize} \quad \|\boldsymbol{w}\|^2/2 + C\sum_{i=1}^{n}\xi_i$$

subject to **relaxed constraints**

$$y_i[\boldsymbol{x}_i^t\boldsymbol{w}] - 1 + \xi_i \geq 0, \ i = 1, \ldots, n.$$



- The $\xi_i \geq 0$ are called **slack variables.**

SVMs: nonseparable case

- We can also write the SVM optimization problem more compactly as

$$C \sum_{i=1}^{n} \overbrace{(1 - y_i[\boldsymbol{x}_i^t \boldsymbol{w}])^+}^{\xi_i} + \|\boldsymbol{w}\|^2/2,$$

  where $(z)^+ = z$ if $z \geq 0$ and zero otherwise.

- This is equivalent to **regularized empirical loss minimization**

$$\underbrace{\frac{1}{n} \sum_{i=1}^{n} (1 - y_i[\boldsymbol{x}_i^t \boldsymbol{w}])^+}_{R_{\text{emp}}} + \lambda \|\boldsymbol{w}\|^2,$$

  where $\lambda = 1/(2nC)$ is the regularization parameter.

SVMs and LOGREG

- When viewed from the point of view of regularized empirical loss minimization, SVM and logistic regression appear quite similar:

$$\text{SVM:} \quad \frac{1}{n}\sum_{i=1}^{n}(1 - y_i[\boldsymbol{x}_i^t \boldsymbol{w}])^+ + \lambda\|\boldsymbol{w}\|^2$$

$$\text{LOGREG:} \quad \frac{1}{n}\sum_{i=1}^{n} -\log \overbrace{\sigma(y_i[\boldsymbol{x}_i^t \boldsymbol{w}])}^{P(y_i|\boldsymbol{x}_i,\boldsymbol{w})} + \lambda\|\boldsymbol{w}\|^2,$$

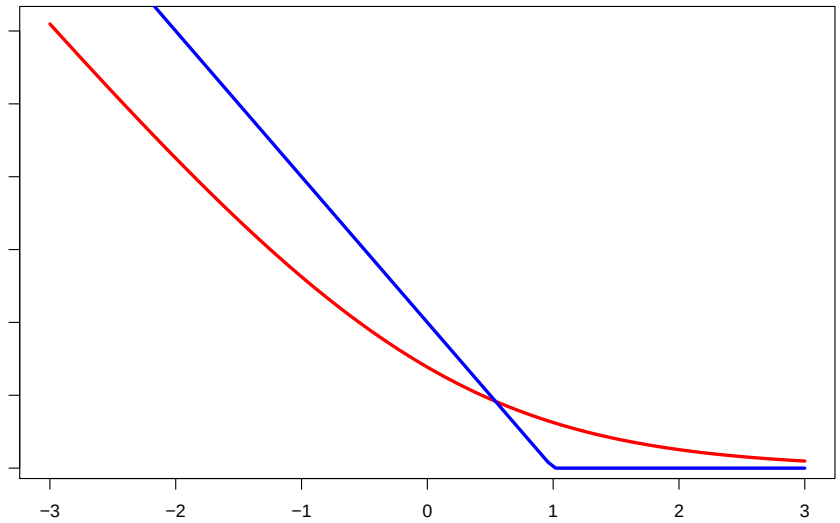
where $\sigma(z) = (1 + e^{-z})^{-1}$ is the logistic function.

SVMs and LOGREG

- The difference comes from how we penalize errors:

$$\text{Both:} \quad \frac{1}{n}\sum_{i=1}^{n} \text{Loss}(\overbrace{y_i[\boldsymbol{x}_i^t \boldsymbol{w}]}^{z}) + \lambda \|\boldsymbol{w}\|^2,$$

- SVM: $\text{Loss}(z) = (1-z)^+$
- LOGREG:
  $\text{Loss}(z) = \log(1+\exp(-z))$



SVMs: solution, Lagrange multipliers

- Back to the separable case: how do we solve

$$\text{minimize}_{\boldsymbol{w}} \quad \|\boldsymbol{w}\|^2/2 \quad \text{s.t.} \quad y_i[\boldsymbol{x}_i^t \boldsymbol{w}] - 1 \geq 0, \; i = 1, \ldots, n.$$

- Represent the constraints as individual loss terms:

$$\sup_{\alpha_i \geq 0} \alpha_i (1 - y_i[\boldsymbol{x}_i^t \boldsymbol{w}]) = \begin{cases} 0, & \text{if } y_i[\boldsymbol{x}_i^t \boldsymbol{w}] - 1 \geq 0, \\ \infty, & \text{otherwise.} \end{cases}$$

- Rewrite the minimization problem:

$$\text{minimize}_{\boldsymbol{w}} \quad \|\boldsymbol{w}\|^2/2 + \sum_{i=1}^{n} \sup_{\alpha_i \geq 0} \alpha_i (1 - y_i[\boldsymbol{x}_i^t \boldsymbol{w}])$$

$$= \text{minimize}_{\boldsymbol{w}} \quad \sup_{\alpha_i \geq 0} \left(\|\boldsymbol{w}\|^2/2 + \sum_{i=1}^{n} \alpha_i (1 - y_i[\boldsymbol{x}_i^t \boldsymbol{w}]) \right)$$

SVMs: solution, Lagrange multipliers

- Swap maximization and minimization (technically this requires that the problem is convex and feasible $\rightsquigarrow$ **Slater's condition**):

$$\begin{aligned} &\text{minimize}_{\boldsymbol{w}} \quad \left[\sup_{\alpha_i \geq 0} \left(\|\boldsymbol{w}\|^2/2 + \sum_{i=1}^{n} \alpha_i (1 - y_i [\boldsymbol{x}_i^t \boldsymbol{w}]) \right) \right] \\ = &\text{maximize}_{\alpha_i \geq 0} \quad \left[\min_{\boldsymbol{w}} \underbrace{\left(\|\boldsymbol{w}\|^2/2 + \sum_{i=1}^{n} \alpha_i (1 - y_i [\boldsymbol{x}_i^t \boldsymbol{w}]) \right)}_{J(\boldsymbol{w};\boldsymbol{\alpha})} \right] \end{aligned}$$

- We have to minimize $J(\boldsymbol{w}; \boldsymbol{\alpha})$ over parameters $\boldsymbol{w}$ for fixed **Lagrange multipliers** $\alpha_i \geq 0$.
  Simple, because $J(\boldsymbol{w})$ is convex $\rightsquigarrow$ set derivative to zero $\rightsquigarrow$ only one stationary point $\rightsquigarrow$ global minimum.

SVMs: solution, Lagrange multipliers

- Find optimal $\boldsymbol{w}$ by setting the derivatives to zero:

$$\frac{\partial}{\partial \boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{\alpha}) = \boldsymbol{w} - \sum_i \alpha_i y_i \boldsymbol{x}_i = 0 \;\Rightarrow\; \hat{\boldsymbol{w}} = \sum_i \alpha_i y_i \boldsymbol{x}_i.$$

- Substitute the solution back into the objective and get (after some re-arrangements of terms):

$$\max_{\alpha_i \geq 0} \min_{\boldsymbol{w}} \left(\|\boldsymbol{w}\|^2/2 + \sum_{i=1}^{n} \alpha_i (1 - y_i [\boldsymbol{x}_i^t \boldsymbol{w}]) \right)$$

$$= \max_{\alpha_i \geq 0} \left(\|\hat{\boldsymbol{w}}\|^2/2 + \sum_{i=1}^{n} \alpha_i (1 - y_i [\boldsymbol{x}_i^t \hat{\boldsymbol{w}}]) \right)$$

$$= \max_{\alpha_i \geq 0} \left(\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \boldsymbol{x}_i^t \boldsymbol{x}_j \right)$$

SVMs: summary

- Find optimal Lagrange multipliers $\hat{\alpha}_i$ by maximizing

$$\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \boldsymbol{x}_i^t \boldsymbol{x}_j \quad \text{subject to } \alpha_i \geq 0.$$

- Only $\hat{\alpha}_i$'s corresponding to **support vectors** will be non-zero.
- Make **predictions** on any new example $\boldsymbol{x}$ according to:

$$\text{sign}(\boldsymbol{x}^t \hat{\boldsymbol{w}}) = \text{sign}(\boldsymbol{x}^t \sum_{i=1}^{n} \hat{\alpha}_i y_i \boldsymbol{x}_i) = \text{sign}(\sum_{i \in SV} \hat{\alpha}_i y_i \boldsymbol{x}^t \boldsymbol{x}_i).$$

- Observation: dependency on input vectors only via **dot products.**
- Later we will introduce the **kernel trick** for efficiently computing these dot products in implicitly defined feature spaces.

SVMs: formal derivation

- **Convex optimization problem:** an optimization problem

$$\begin{aligned} &\text{minimize} && f(\boldsymbol{x}) && (1)\\ &\text{subject to} && g_i(\boldsymbol{x}) \leq 0, \quad i = 1, \ldots, m && (2) \end{aligned}$$

  is convex if the functions $f, g_1 \ldots g_m : \mathbb{R}^n \to \mathbb{R}$ are convex.

- The **Lagrangian function** for the problem is

$$\mathcal{L}(\boldsymbol{x}, \lambda_0, ..., \lambda_m) = \lambda_0 f(\boldsymbol{x}) + \lambda_1 g_1(\boldsymbol{x}) + ... + \lambda_m g_m(\boldsymbol{x}).$$

- **Karush-Kuhn-Tucker (KKT) conditions:** For each point $\hat{\boldsymbol{x}}$ that minimizes f, there exist real numbers $\lambda_0, \ldots, \lambda_m$, called **Lagrange multipliers**, that simultaneously satisfy:
  1. $\hat{\boldsymbol{x}}$ minimizes $\mathcal{L}(\boldsymbol{x}, \lambda_0, \lambda_1, \ldots, \lambda_m)$,
  2. $\lambda_0 \geq 0, \lambda_1 \geq 0, \ldots, \lambda_m \geq 0$, with at least one $\lambda_k > 0$,
  3. Complementary slackness: $g_i(\hat{\boldsymbol{x}}) < 0 \Rightarrow \lambda_i = 0$, $1 \leq i \leq m$.

SVMs: formal derivation

- **Slater's condition:** If there exists a **strictly feasible point** z satisfying $g_1(z) < 0, \ldots, g_m(z) < 0$, then one can set $\lambda_0 = 1$.
- Assume that Slater's condition holds. Minimizing the supremum $\mathcal{L}^*(\boldsymbol{x}) = \sup_{\lambda \geq 0} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda})$, is the **primal problem P:**

$$\hat{\boldsymbol{x}} = \operatorname*{argmin}_{\boldsymbol{x}} \mathcal{L}^*(\boldsymbol{x}).$$

Note that

$$\mathcal{L}^*(\boldsymbol{x}) = \sup_{\lambda \geq 0} \left(f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i g_i(\boldsymbol{x}) \right) = \begin{cases} f(\boldsymbol{x}) & \text{, if } g_i(\boldsymbol{x}) \leq 0 \, \forall i \\ \infty & \text{, else.} \end{cases}$$

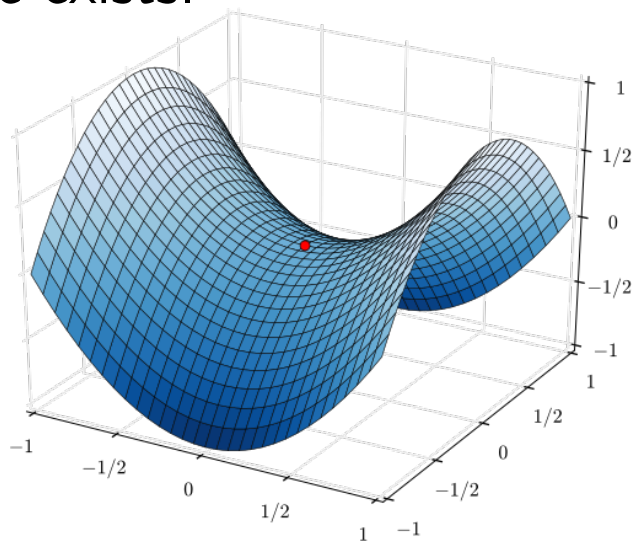
$\rightsquigarrow$ **Minimizing $\mathcal{L}^*(\boldsymbol{x})$ is equivalent to minimizing $f(\boldsymbol{x})$.**

- The maximizer of the **dual problem D** is

$$\hat{\boldsymbol{\lambda}} = \operatorname*{argmax}_{\boldsymbol{\lambda}} \mathcal{L}_*(\boldsymbol{\lambda}), \text{ where } \mathcal{L}_*(\boldsymbol{\lambda}) = \inf_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}).$$

SVMs: formal derivation

- The non-negative number min(P) – max(D) is the **duality gap.**
- **Convexity and Slater's condition imply strong duality:**
  1. The optimal solution $(\hat{\boldsymbol{x}}, \hat{\boldsymbol{\lambda}})$ is a saddle point of $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda})$
  2. The **duality gap is zero.**
- Discussion: For any real function $f(a, b)$
  $\min_a[\max_b f(a, b)] \geq \max_b[\min_a f(a, b)]$.
  Equality $\rightsquigarrow$ saddle value exists.



By Nicoguaro - Own work, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=20570051

Kernel functions

- A **kernel function** is a real-valued function of two arguments, $k(\boldsymbol{x}, \boldsymbol{x}') \in \mathbb{R}$, for $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$.
- Typically the function is **symmetric**, and sometimes non-negative.
- In the latter case, it might be interpreted as a measure of similarity.
- Example: **isotropic Gaussian kernel:**

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|^2}{2\sigma^2}\right)$$

Here, σ^2 is the bandwidth. This is an example of a **radial basis function (RBF) kernel** (only a function of $\|\boldsymbol{x} - \boldsymbol{x}'\|^2$).

Mercer kernels

- A symmetric kernel is a **Mercer kernel**, iff the Gram matrix

$$K = \begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ & \vdots & \\ k(\boldsymbol{x}_n, \boldsymbol{x}_1) & \dots & k(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{pmatrix}$$

  is **positive semidefinite** for any set of inputs $\{\boldsymbol{x}_i, \dots, \boldsymbol{x}_n\}$.

- **Mercer's theorem:** Eigenvector decomposition

$$K = V \Lambda V^t = (V \Lambda^{1/2})(V \Lambda^{1/2})^t =: \Phi \Phi^t.$$

  Eigenvectors: columns of V. Eigenvalues: entries of diagonal matrix $\Lambda = \mathrm{diag}(\lambda_1, \dots, \lambda_n)$. Note that $\lambda_i \in \mathbb{R}$ and $\lambda_i \geq 0$.
  Define $\phi(\boldsymbol{x}_i)^t = i$-th row of $\Phi = V_{[i\bullet]} \Lambda^{1/2}$
  $\rightsquigarrow k(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) = \phi(\boldsymbol{x}_i)^t \phi(\boldsymbol{x}_{i'})$.

- Entries of K: **inner product of some feature vectors**, implicitly defined by eigenvectors V.

Mercer kernels

- If the kernel is **Mercer**, then there exists $\phi : \boldsymbol{x} \to \mathbb{R}^d$ such that

  $$k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^t \phi(\boldsymbol{x}'),$$

  where ϕ depends on the eigenfunctions of k (d might be infinite).
- Example: **Polynomial kernel**

  $$k(\boldsymbol{x}, \boldsymbol{x}') = (1 + \boldsymbol{x}^t \boldsymbol{x}')^m.$$

  Corresponding feature vector contains terms up to degree m.
  Example: $m = 2,\ x \in \mathbb{R}^2$:

  $$(1 + \boldsymbol{x}^t \boldsymbol{x}')^2 = 1 + 2x_1 x_1' + 2x_2 x_2' + (x_1 x_1')^2 + (x_2 x_2')^2 + 2x_1 x_1' x_2 x_2'.$$

  Thus,

  $$\phi(\boldsymbol{x}) = [1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2]^t.$$

  Equivalent to working in a 6-dim feature space.
- **Gaussian kernel:** feature map lives in an infinite dimensional space.

Kernels for documents

- In document classification or retrieval, we want to compare two documents, $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i'}$.
- **Bag of words** representation:
  $\boldsymbol{x}_{ij}$ is the number of times word j occurs in document i.
- One possible choice: **Cosine similarity:**

$$k(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) = \frac{\boldsymbol{x}_i^t \boldsymbol{x}_{i'}}{\|\boldsymbol{x}_i\| \|\boldsymbol{x}_{i'}\|} =: \phi(\boldsymbol{x}_i)^t \phi(\boldsymbol{x}_{i'}).$$

- Problems:
  - Popular words (like "the" or "and") are not discriminative
    $\rightsquigarrow$ remove these **stop words**.
  - Bias: once a word is used in a document,
    it is very likely to be **used again.**
- Solution: Replace word counts with "normalized" representation.

Kernels for documents

- TF-IDF "term frequency inverse document frequency":
  Term frequency is log-transform of the count:

  $$\text{tf}(x_{ij}) = \log(1 + x_{ij})$$

  Inverse document frequency:

  $$\text{idf}(j) = \log \frac{\#(\text{documents})}{\#(\text{documents containing term } j)} = \log \frac{1}{\hat{p}_j}.$$

  $\rightsquigarrow$ Shannon information content:

  idf is a measure of how much information a word provides

- Combine with tf $\rightsquigarrow$ counts weighted by information content:

  $$\text{tf-idf}(\boldsymbol{x}_i) = [\text{tf}(\boldsymbol{x}_{ij}) \cdot \text{idf}(j)]_{j=1}^{V}, \quad \text{where } V = \text{size of vocabulary.}$$

- We then use this inside the **cosine similarity measure.**
  With $\phi(\boldsymbol{x}) = \text{tf-idf}(\boldsymbol{x})$:

  $$k(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) = \frac{\phi(\boldsymbol{x}_i)^t \phi(\boldsymbol{x}_{i'})}{\|\phi(\boldsymbol{x}_i)\| \|\phi(\boldsymbol{x}_{i'})\|}.$$

String kernels

- Real power of kernels arises for **structured input objects.**
- Consider two strings x, and x' of lengths d, d', over alphabet $\mathcal{A}$.
  Idea: define similarity as the **number of common substrings.**
- If s is a substring of $x \rightsquigarrow \phi_s(x) =$ number of times s appears in x.
- **String kernel**

$$k(x, x') = \sum_{s \in \mathcal{A}^*} w_s \phi_s(x) \phi_s(x'),$$

  where $w_s \geq 0$ and $\mathcal{A}^* =$ set of all strings (any length) from $\mathcal{A}$.
- One can show: Mercer kernel, can be computed in $O(|x| + |x'|)$ time using suffix trees (Shawe-Taylor and Cristianini, 2004).
- Special case: $w_s = 0$ for $|s| > 1$: **bag-of-characters kernel:**
  $\phi(x)$ is the number of times each character in $\mathcal{A}$ occurs in x.

The kernel trick

- Idea: modify algorithm so that it **replaces all inner products $x^t x'$** with a call to the **kernel function $k(x, x')$.**
- **Kernelized ridge regression:** $\hat{w} = (X^t X + \lambda I)^{-1} X^t y$.
  Matrix inversion lemma:
  $$(I + UV)^{-1} U = U(I + VU)^{-1}$$
  Define new variables α_i:
  $$\hat{w} = (X^t X + \lambda I)^{-1} X^t y$$
  $$= X^t \underbrace{(XX^t + \lambda I)^{-1} y}_{\hat{\alpha}} = \sum_{i=1}^{n} \hat{\alpha}_i x_i.$$
  $\rightsquigarrow$ **solution is linear sum of the n training vectors.**

The kernel trick

- Use this and the kernel trick to make **predictions for $\boldsymbol{x}$:**

$$\hat{f}(\boldsymbol{x}) = \hat{\boldsymbol{w}}^t \boldsymbol{x} = \sum_{i=1}^{n} \hat{\alpha}_i \boldsymbol{x}_i^t \boldsymbol{x} = \sum_{i=1}^{n} \hat{\alpha}_i k(\boldsymbol{x}_i, \boldsymbol{x}).$$

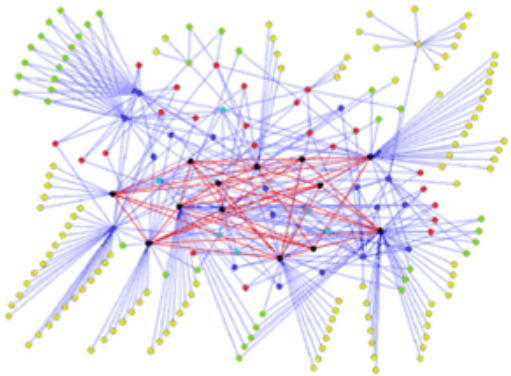
- Same for SVMs:

$$\hat{\boldsymbol{w}}^t \boldsymbol{x} = \sum_{i \in SV} \hat{\alpha}_i y_i \boldsymbol{x}_i^t \boldsymbol{x} = \sum_{i \in SV} \hat{\alpha}_i' k(\boldsymbol{x}_i, \boldsymbol{x})$$

- ...and for most other classical algorithms in ML!

Some applications in bioinformatics

- Bioinformatics: often **non-vectorial** data-types:
  - interaction graphs



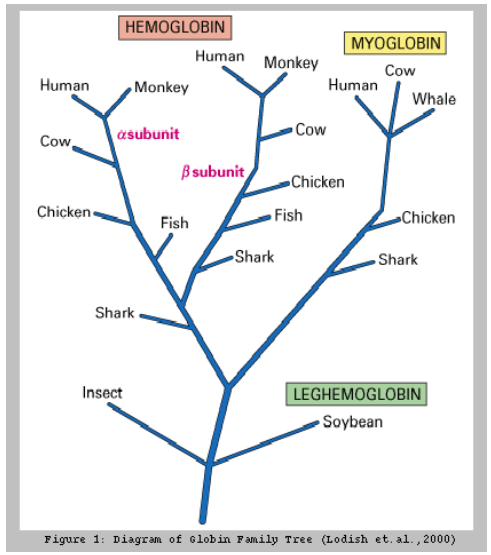
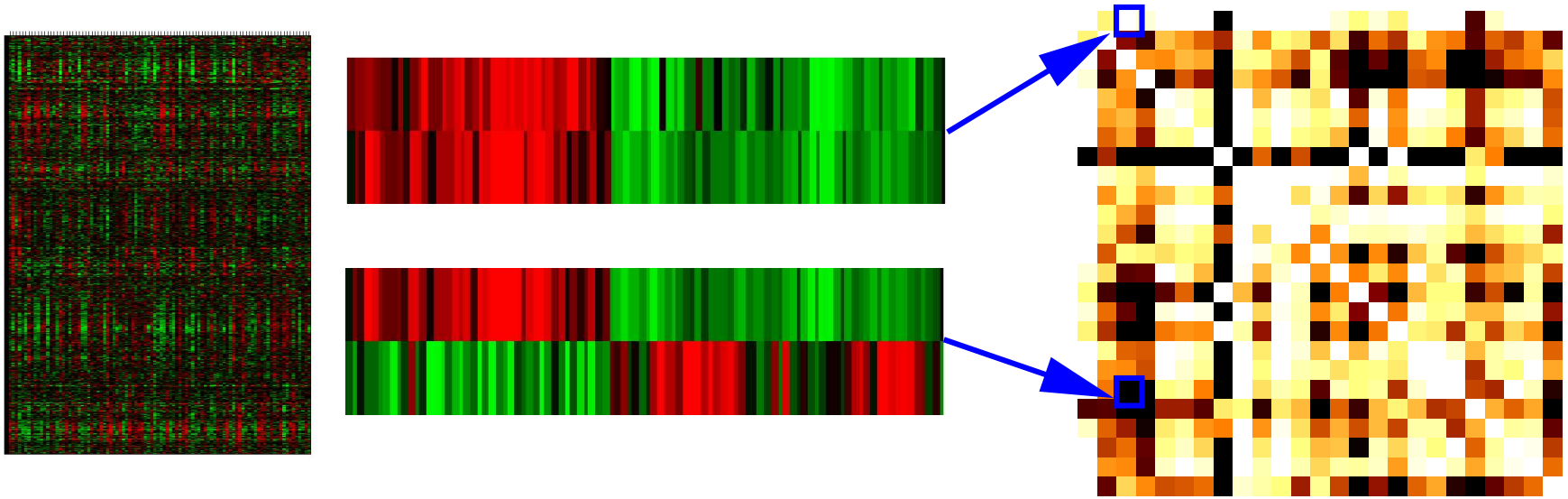


Figure 1: Diagram of Globin Family Tree (Lodish et.al.,2000)

  - phylogenetic trees
  - strings GSAQVKGHGKKVADALTNAVAHV
- **Data fusion:** convert data of each type into kernel matrix
  $\Rightarrow$ **fuse kernel matrices**
  $\Rightarrow$ **"common language" for heterogeneous data.**

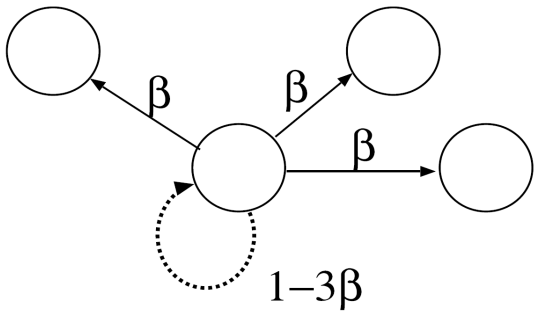
RBF kernels from expression data

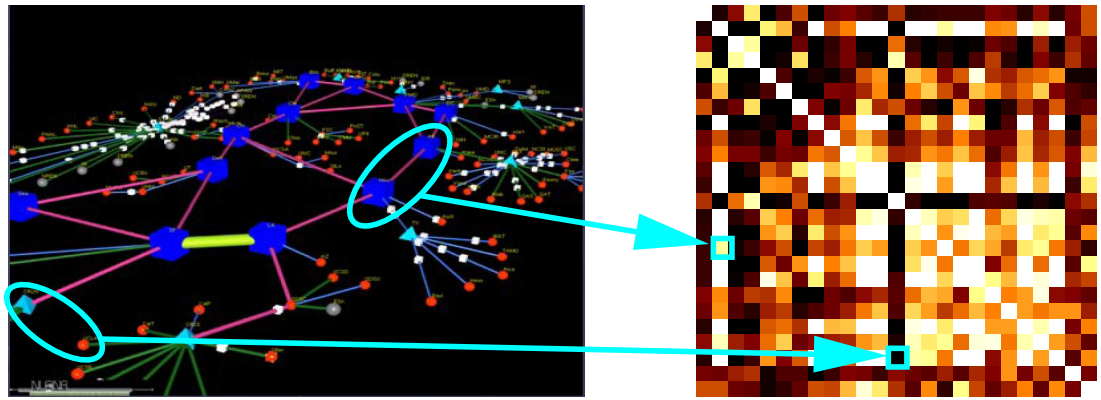
- **Measurements** (for each gene): vector of expression values under different experimental conditions
- "classical" RBF kernel $k(x_1, x_2) = \exp(-\sigma \|x_1 - x_2\|^2)$



Diffusion kernels from interaction-graphs

- A: Adjacency matrix, D: node degrees, $L = D - A$.
- $K := \frac{1}{Z(\beta)} \exp(-\beta L)$ with transition probabilities β.
- **Physical interpretation (*random walk*):**
  randomly choose next node among neighbors.
- Self-transition occurs with prob. $1 - d_i\beta$





- K_{ij}: prob. for walk from i to j.

(Kondor and Lafferty, 2002)

Alignment kernels from sequences

Alignment with **Pair HMMs**
$\rightsquigarrow$ Mercer kernel (Watkins, 2000).

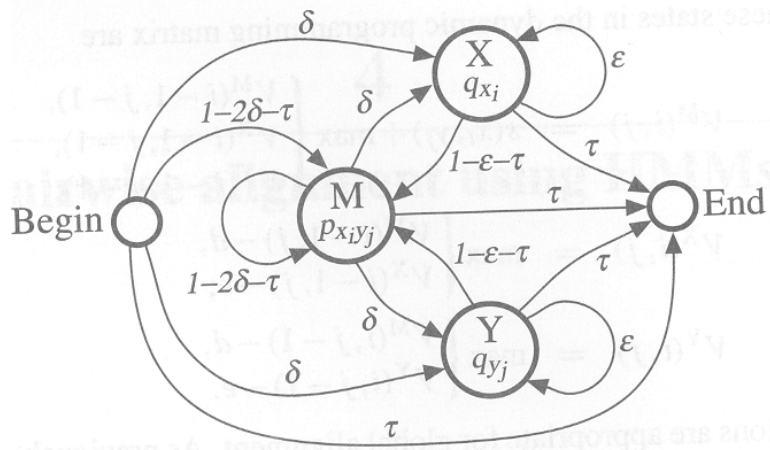
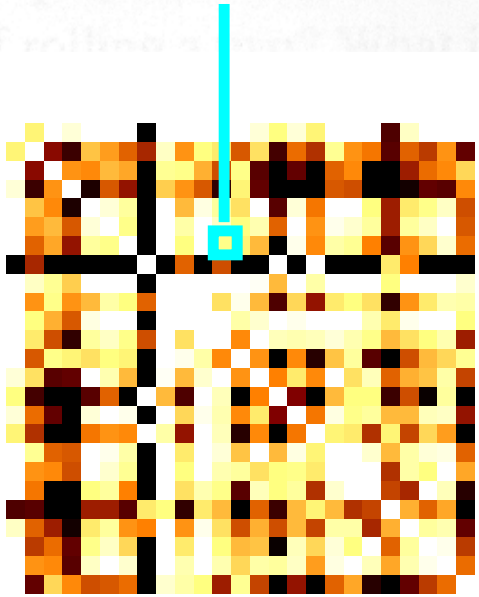


Image source: Durbin, Eddy, Krogh, Mitchison. Biological Sequence Alignment. Cambridge.

```
HBA_HUMAN   GSAQVKGHGKKVADALTNAVAHV---D--DMPNALSALSDLHAHKL
            ++ ++++H+ KV    + +A   ++              +L+ L+++H+ K
LGB2_LUPLU NNPELQAHAGKVFKLVYEAAIQLQVTGVVVTDATLKNLGSVHVSKG
```

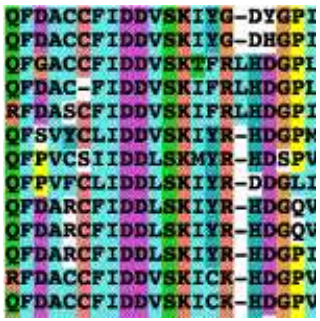


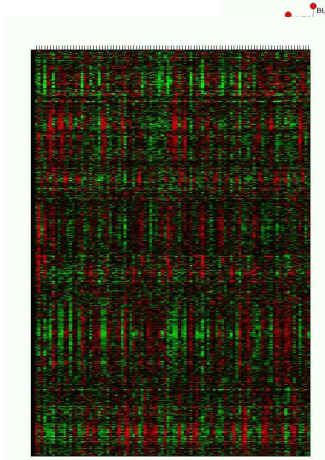
Combination of heterogeneous data

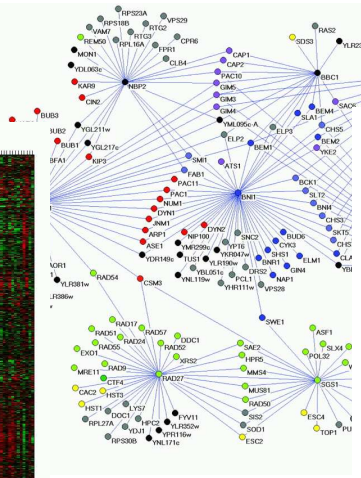
Adding kernels $\Rightarrow$ new kernel:

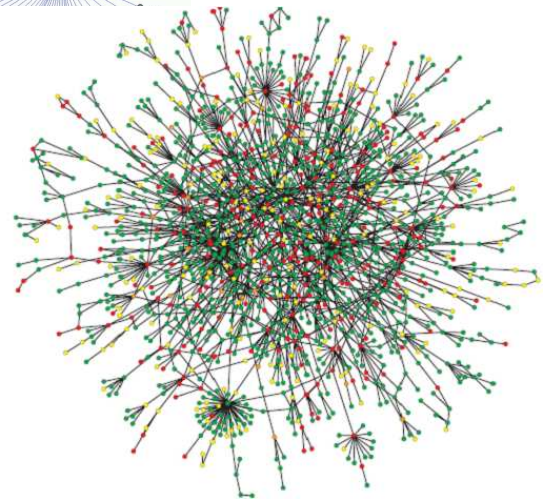
$$k_1(x,y) = \phi_1(x) \cdot \phi_1(y),$$
$$k_2(x,y) = \phi_2(x) \cdot \phi_2(y)$$
$$\Rightarrow k' = k_1 + k_2 = \begin{pmatrix}\phi_1(x)\\ \phi_2(x)\end{pmatrix} \cdot \begin{pmatrix}\phi_1(y)\\ \phi_2(y)\end{pmatrix}$$

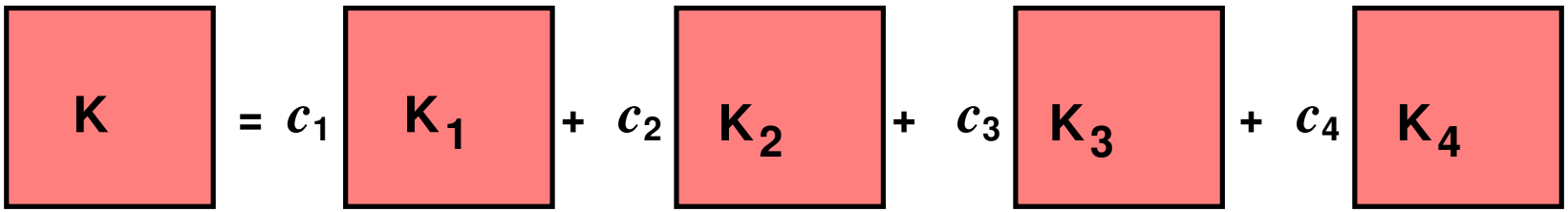
Fusion & relevance determination: kernel-combinations











Section 8

Gaussian Processes: probabilistic kernel models

Overview

- The use of the Gaussian distribution in ML
  - Properties of the **multivariate Gaussian distribution**
  - Random variables $\rightarrow$ random vectors $\rightarrow$ **stochastic processes**
  - **Gaussian processes for regression**
  - **Model Selection**
  - **Gaussian processes for classification**
- Relation to **kernel models** (e.g. SVMs)
- Relation to **neural networks.**

Kernel Ridge Regression

- **Kernelized ridge regression:** $\hat{\boldsymbol{w}} = (X^t X + \lambda I)^{-1} X^t \boldsymbol{y}$.
- Matrix inversion lemma: $(I + UV)^{-1} U = U(I + VU)^{-1}$
- Define new variables α_i:

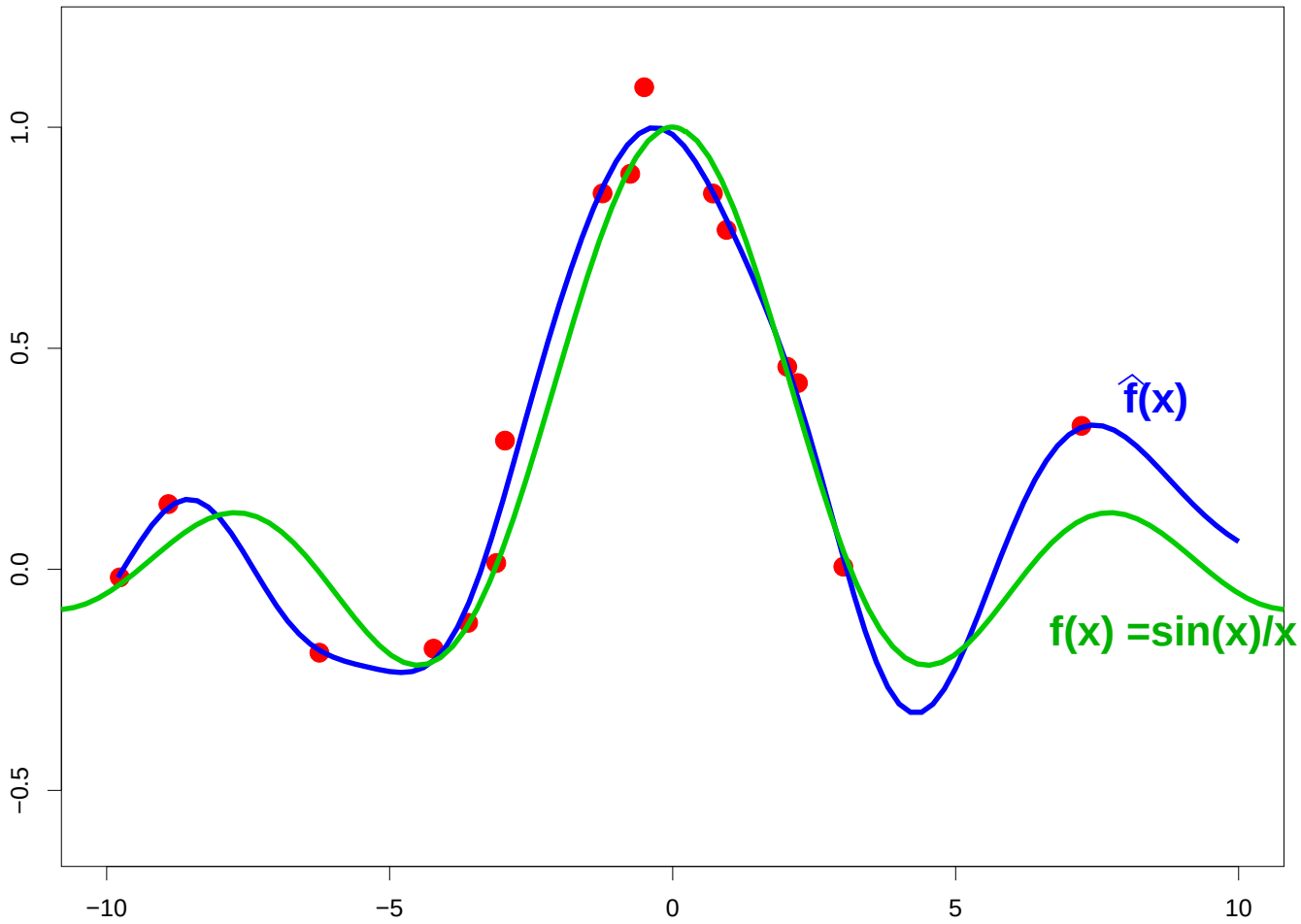
$$\hat{\boldsymbol{w}} = (X^t X + \lambda I)^{-1} X^t \boldsymbol{y}$$

$$= X^t \underbrace{(XX^t + \lambda I)^{-1} \boldsymbol{y}}_{\hat{\boldsymbol{\alpha}}} = \sum_{i=1}^{n} \hat{\alpha}_i \boldsymbol{x}_i.$$

- **Predictions for new $\boldsymbol{x}_*$:**

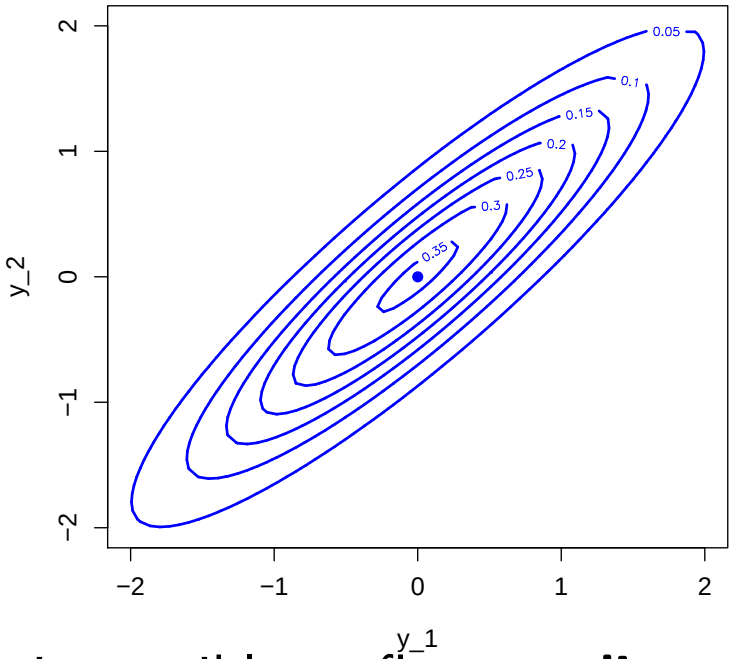
$$\hat{f}(\boldsymbol{x}_*) = \hat{\boldsymbol{w}}^t \boldsymbol{x}_* = \sum_{i=1}^{n} \hat{\alpha}_i \boldsymbol{x}_i^t \boldsymbol{x}_* = \sum_{i=1}^{n} \hat{\alpha}_i k(\boldsymbol{x}_i, \boldsymbol{x}_*).$$

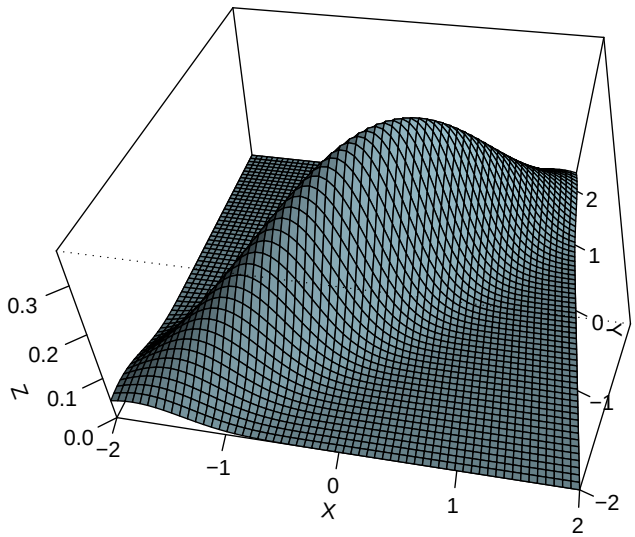
Kernel Ridge Regression



Kernel function: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\frac{1}{2l^2}\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2)$

How can we make use of the Gaussian distribution?





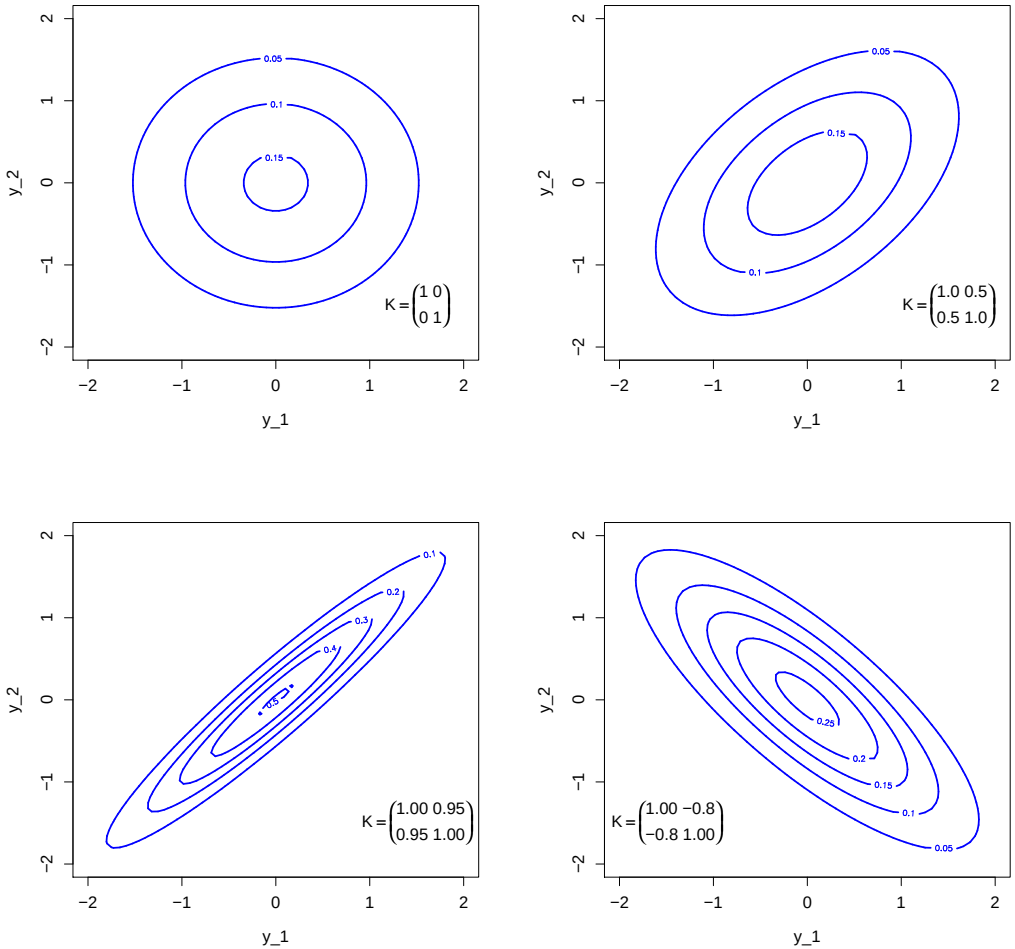
- Is it possible to fit a **nonlinear regression line** with the "boring" Gaussian distribution?
- **Yes**, but we need to introduce the concept of **Gaussian Processes**!

The 2D Gaussian distribution

2D Gaussian: $P(\boldsymbol{y}; \boldsymbol{\mu} = \mathbf{0}, \Sigma = K) = \frac{1}{\sqrt{2\pi|K|}} \exp(-\frac{1}{2}\boldsymbol{y}^t K^{-1}\boldsymbol{y})$

Covariance
(also written "*co*-variance")
is a measure of how much **two random variables vary together**:

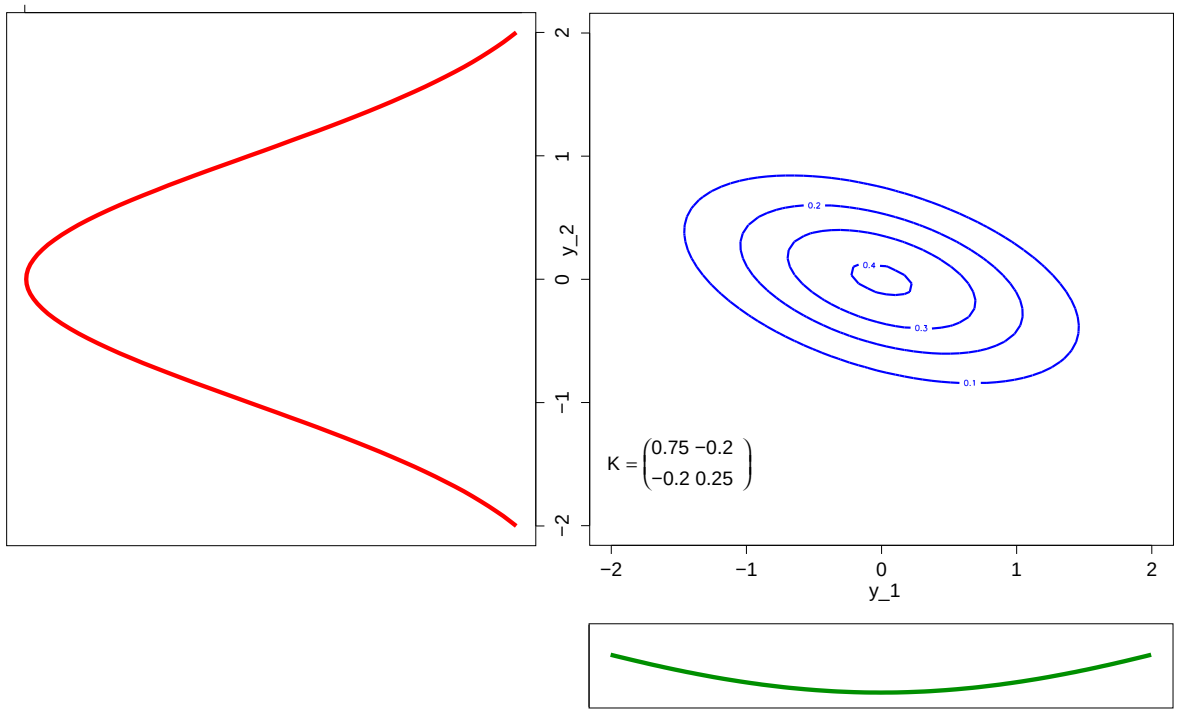
- **+1**: perfect linear coherence,
- **-1**: perfect negative linear coherence,
- **0**: no linear coherence.



Properties of the Multivariate Gaussian distribution

$\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, K)$. Let $\boldsymbol{y} = \begin{pmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \end{pmatrix}$ and $K = \begin{pmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{pmatrix}$.

Then $\boldsymbol{y}_1 \sim \mathcal{N}(\boldsymbol{\mu}_1, K_{11})$ and $\boldsymbol{y}_2 \sim \mathcal{N}(\boldsymbol{\mu}_2, K_{22})$.

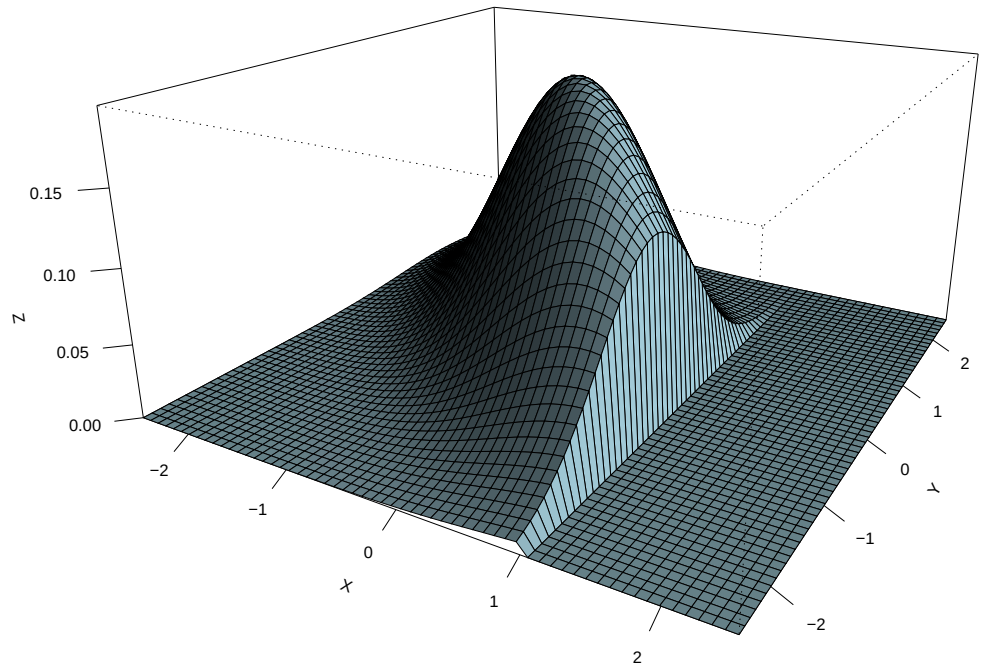


Marginals of Gaussians are again Gaussian!

Properties of the Multivariate Gaussian distribution (2)

$\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, K)$. Let $\boldsymbol{y} = \begin{pmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \end{pmatrix}$ and $K = \begin{pmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{pmatrix}$.

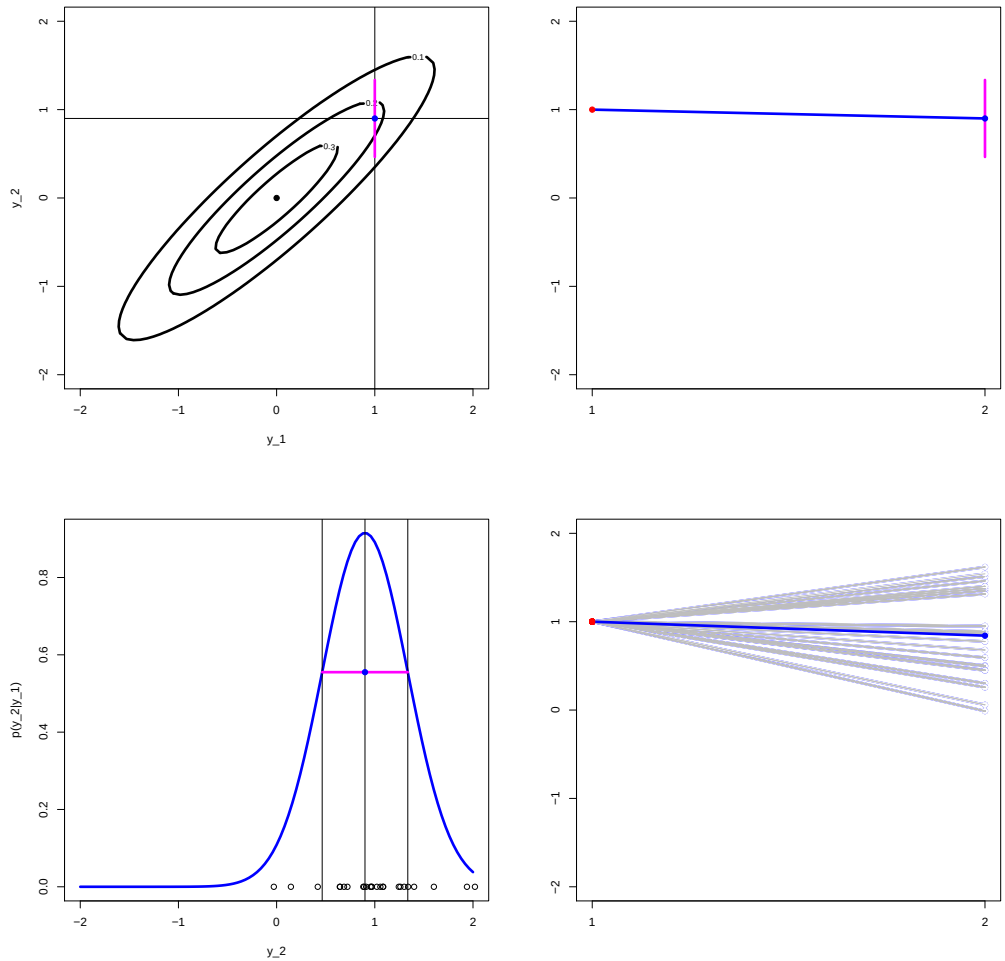
Then $\boldsymbol{y}_2 | \boldsymbol{y}_1 \sim \mathcal{N}(\boldsymbol{\mu}_2 + K_{21} K_{11}^{-1}(\boldsymbol{y}_1 - \boldsymbol{\mu}_1), K_{22} - K_{21} K_{11}^{-1} K_{12})$.



Conditionals of Gaussians are again Gaussian!

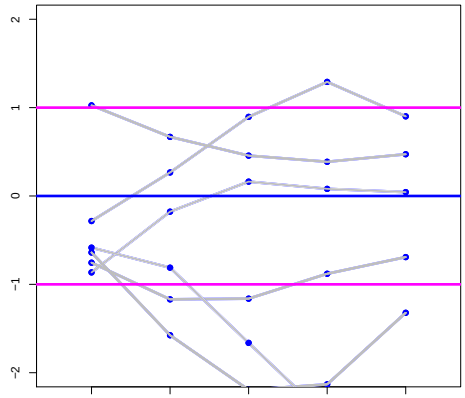
2D Gaussians: a new visualization

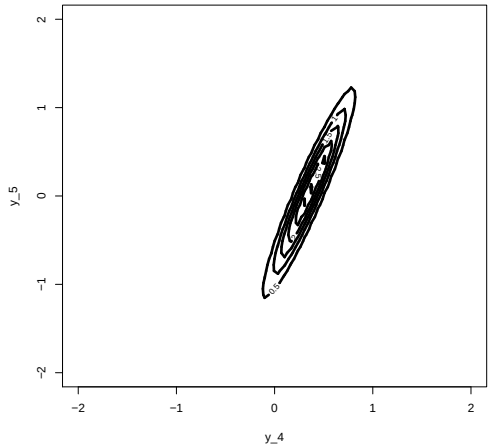
- **top left:** mean and $\pm$std.dev. of $p(y_2|y_1 = 1)$.
- **bottom left:** $p(y_2|y_1 = 1)$ and samples drawn from it.
- **top right:** x-axis: indices $(1, 2)$ of dimensions, y-axis: density in each component. Shown are $y_1 = 1$ and the conditional mean $\bar{p}(y_2|y_1 = 1)$ and std.dev.
- **bottom right:** samples drawn from above model.

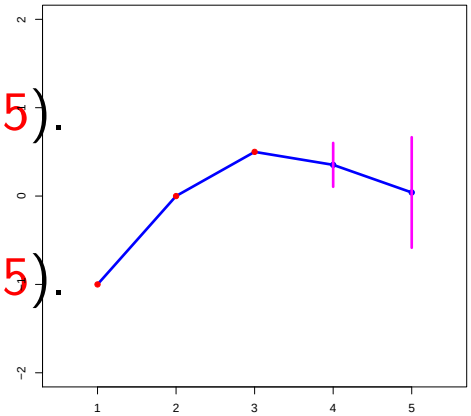


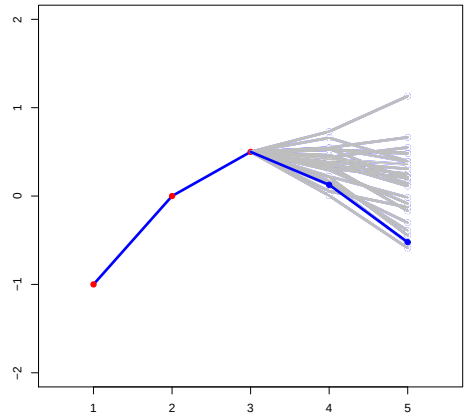
Visualizing high-dimensional Gaussians

- **top left:** 6 samples drawn from 5-dimensional Gaussian with zero mean (indicated by blue line). $\sigma = 1$ (magenta line).
- **bottom left:** Conditional mean and std.dev of $p(y_4, y_5 | y_1 = -1, y_2 = 0, y_3 = 0.5)$.
- **top right:** contour lines of $p(y_4, y_5 | y_1 = -1, y_2 = 0, y_3 = 0.5)$.
- **bottom right:** samples drawn from above model.



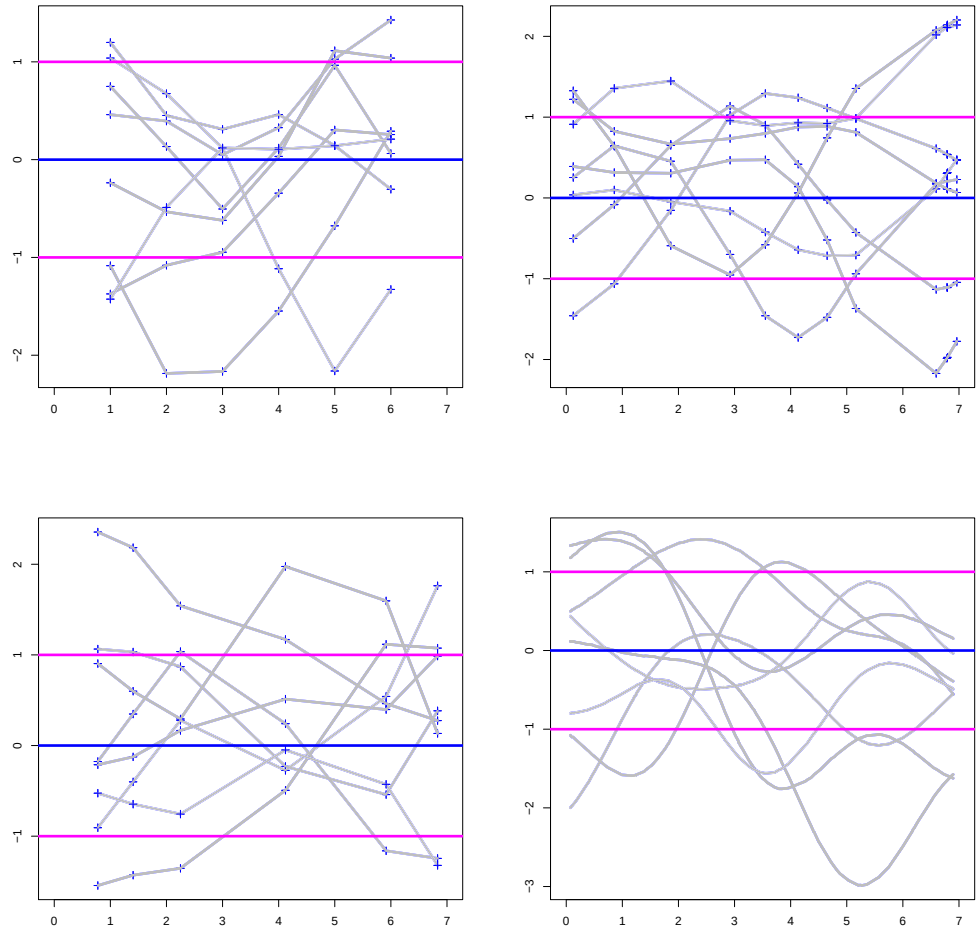




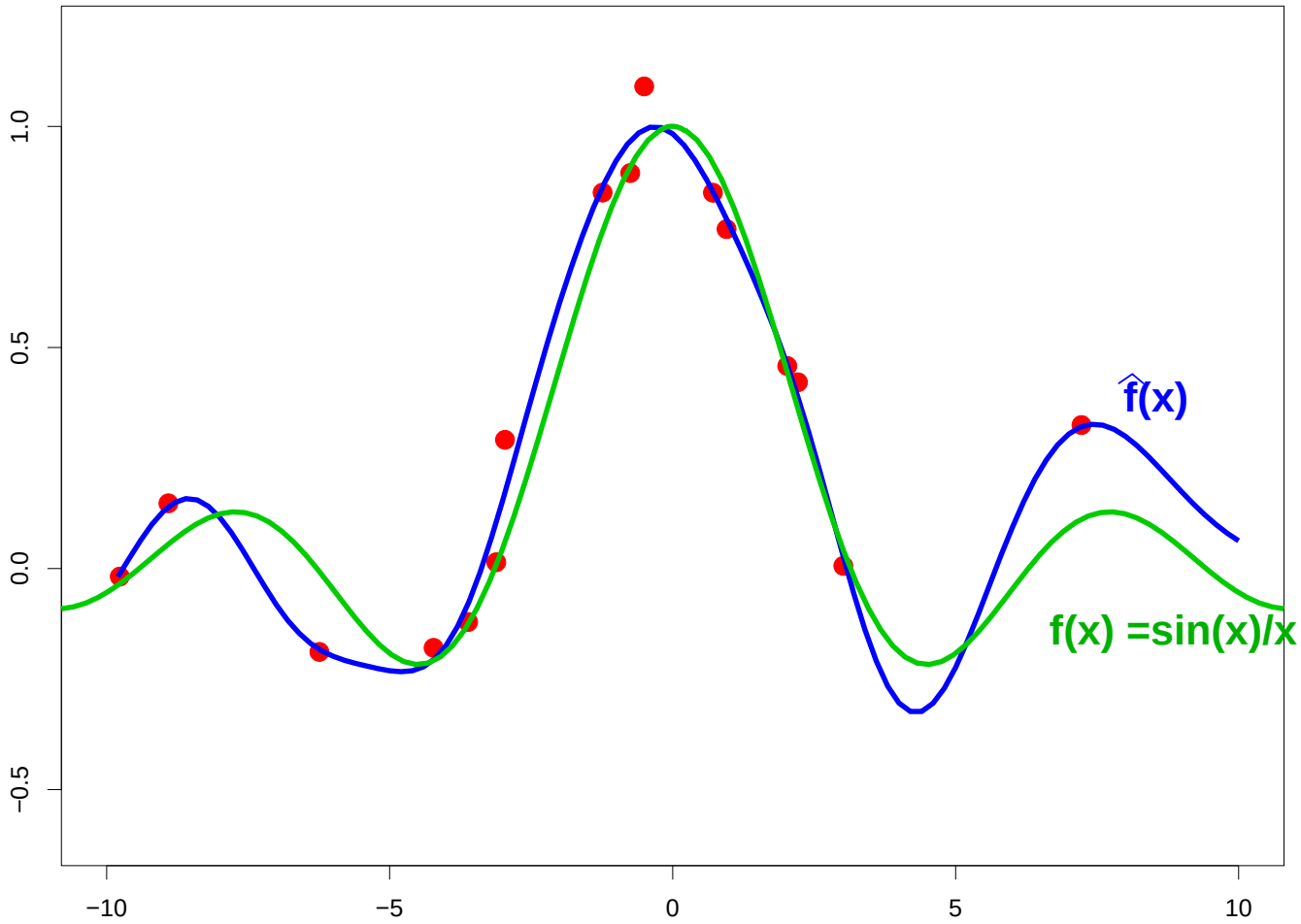


From covariance matrices to Gaussian processes

- **top left:** 8 samples, 6 dim.
  x-axis: dimension-indices.
- **bottom left:** 8 samples,
  viewed as values $y = f(\boldsymbol{x})$.
  Construction: choose 6
  input points $\boldsymbol{x}_i$ at random
  $\rightsquigarrow$ build covariance matrix K
  with **covariance function**
  $k(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\frac{1}{2l^2}\|\boldsymbol{x}-\boldsymbol{x}'\|^2)$
  $\rightsquigarrow$ draw $\boldsymbol{f} \sim \mathcal{N}(\boldsymbol{0}, K)$
  $\rightsquigarrow$ plot as function of inputs.
- **top right:** same for 12 inputs
- **bottom right:** 100 inputs



This looks similar to Kernel Regression...



Gaussian Processes

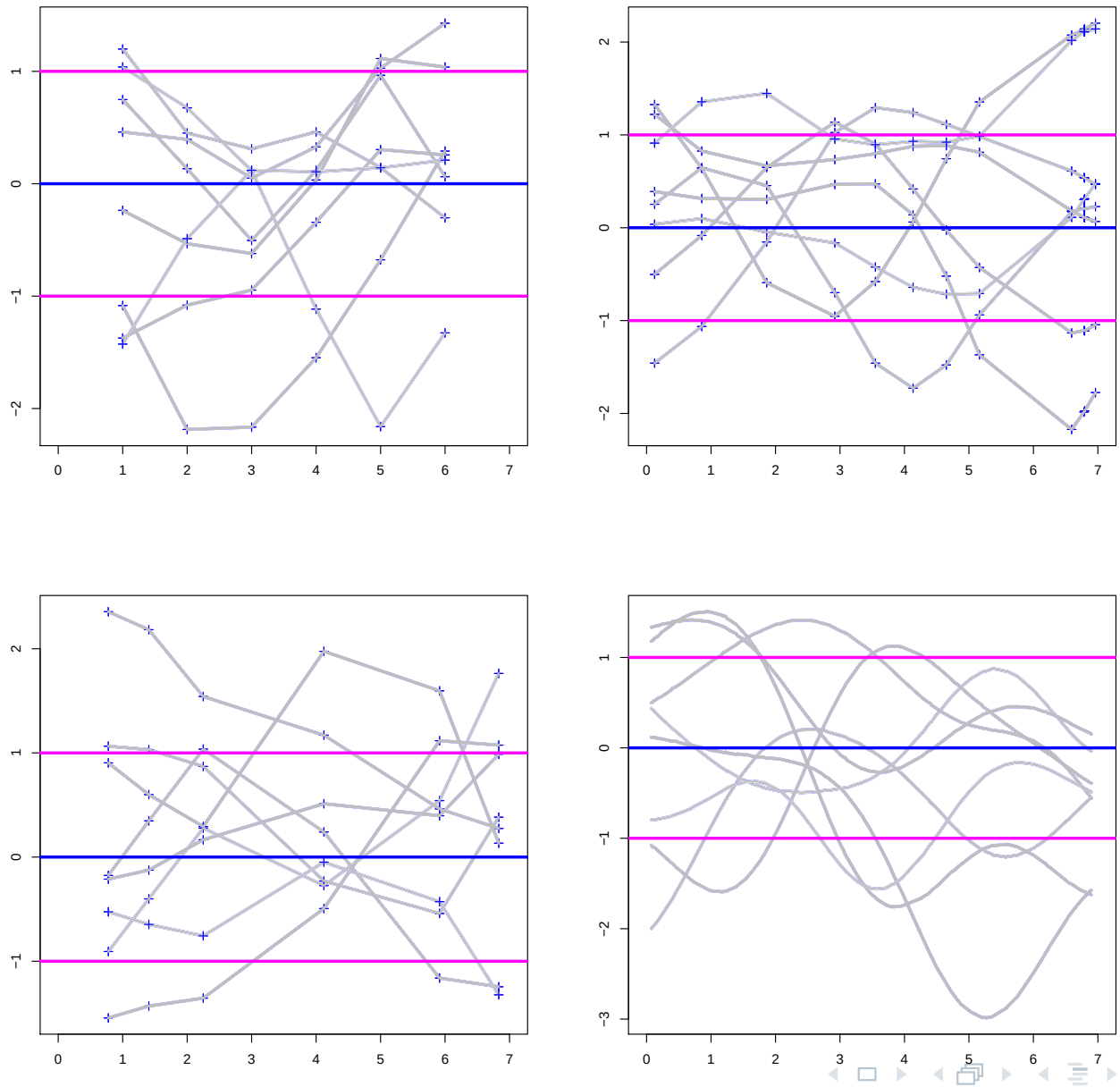
- Gaussian **Random Variable** (RV): $f \sim \mathcal{N}(\mu, \sigma^2)$.
- Gaussian **Random Vector**: Collection of n RVs, characterized by mean vector and covariance matrix: $\boldsymbol{f} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$
- Gaussian **Process**: infinite Gaussian random vector, every finite subset of which is jointly Gaussian distributed
  Continuous index, e.g. time $t \rightsquigarrow$ **function** $f(t)$**.**
  Fully specified by **mean function** $m(t) = \mathbb{E}[f(t)]$
  and **covariance function** $k(t, t') = \mathbb{E}[(f(t) - m(t))(f(t') - m(t'))]$.
- In ML, we will focus on more general index sets $\boldsymbol{x} \in \mathbb{R}^d$ with **mean function** $m(\boldsymbol{x})$ and **covariance function** $k(\boldsymbol{x}, \boldsymbol{x}')$:

$$f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')).$$

Visualizing Gaussian Processes: Sampling

- **Problem:** working with infinite vectors and covariance matrices is not very intuitive...
- **Solution:** evaluate the GP at set of n discrete times (or input vectors $\boldsymbol{x} \in \mathbb{R}^d$):
  - Choose n **input points** $\boldsymbol{x}_i$ at random $\rightsquigarrow$ matrix X
  - build **covariance matrix** $K(X, X)$ with **covariance function** $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$
  - **sample** realizations of the Gaussian random vector $\boldsymbol{f} \sim \mathcal{N}(\boldsymbol{0}, K(X, X))$
  - plot $\boldsymbol{f}$ as **function of inputs**.

This is exactly what we have done here...



From the Prior to the Posterior

GP defines distribution over functions $\rightsquigarrow \boldsymbol{f}$ evaluated at training points X and $\boldsymbol{f}_*$ evaluated at test points X_* are jointly Gaussian:

$$\begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{f}_* \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right)$$

Posterior $p(\boldsymbol{f}_*|X_*, X, \boldsymbol{f}(X))$: conditional of a Gaussian distribution.

Let $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, K)$. Let $\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{pmatrix}$ and $K = \begin{pmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{pmatrix}$.

Then $\boldsymbol{x}_2|\boldsymbol{x}_1 \sim \mathcal{N}(\boldsymbol{\mu}_2 + K_{21}K_{11}^{-1}(\boldsymbol{f}_1 - \boldsymbol{\mu}_1), K_{22} - K_{21}K_{11}^{-1}K_{12})$.

$$\boldsymbol{f}_*|X_*, X, \boldsymbol{f} \sim \mathcal{N}(\quad K(X_*,X)(K(X,X))^{-1}\boldsymbol{f}, \\ K(X_*,X_*) - K(X_*,X)(K(X,X))^{-1}K(X,X_*))$$

For only one test case:

$$f_*|\boldsymbol{x}_*, X, \boldsymbol{f} \sim \mathcal{N}(\boldsymbol{k}_*^t K^{-1}\boldsymbol{f}, k_{**} - \boldsymbol{k}_*^t K^{-1}\boldsymbol{k}_*)$$

A simple extension: noisy observations

- Assume we have access only to noisy versions of function values: $y = f(\boldsymbol{x}) + \eta,\ \eta \sim \mathcal{N}(0, \sigma^2)$ (cf. initial example of **ridge regression**).
- Noise η does not depend on data!
- Covariance of noisy observations y is sum of covariance of f and variance of noise: $cov(\boldsymbol{y}) = K(X, X) + \sigma^2 I$.

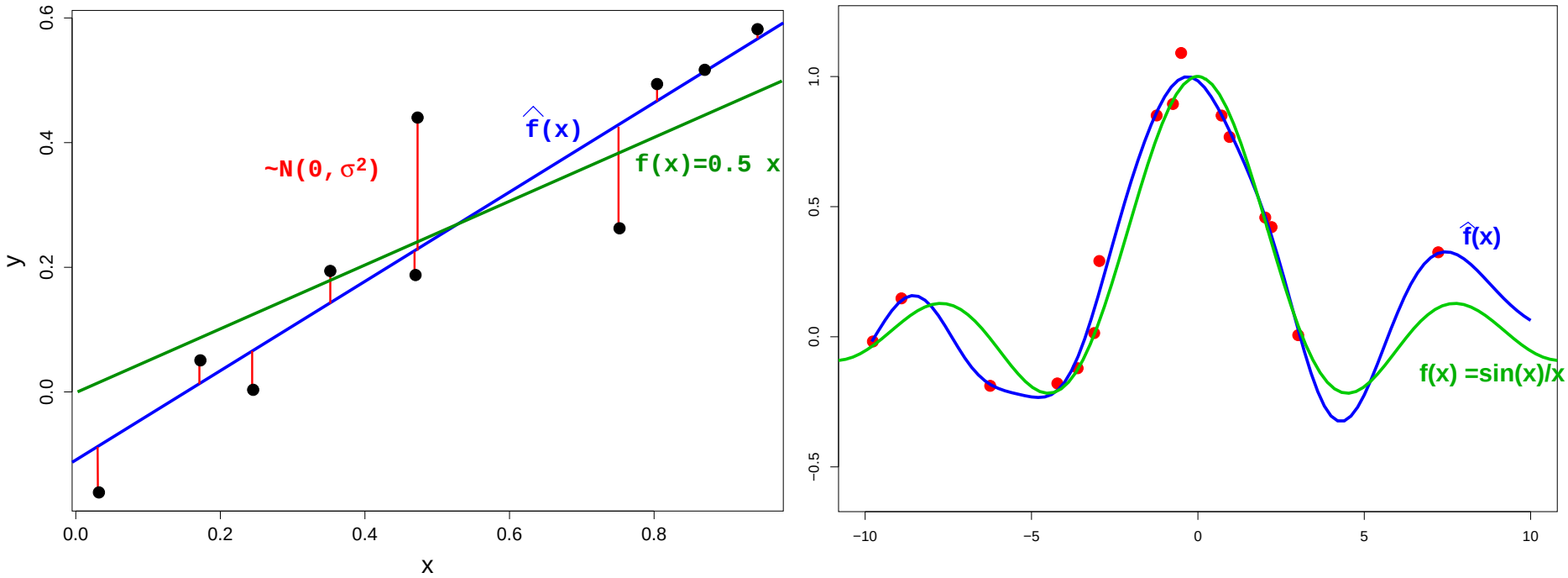
$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{f}_* \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} K(X,X) + \sigma^2 I & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right)$$

$$\begin{aligned} \boldsymbol{f}_* | X_*, X, \boldsymbol{y} \sim \mathcal{N}(& K(X_*, X)(K(X,X) + \sigma^2 I)^{-1} \boldsymbol{y}, \\ & K(X_*, X_*) - K(X_*, X)(K(X,X) + \sigma^2 I)^{-1} K(X, X_*)) \end{aligned}$$

$$f_* | \boldsymbol{x}_*, X, \boldsymbol{f} \sim \mathcal{N}(\boldsymbol{k}_*^t (K + \sigma^2 I)^{-1} \boldsymbol{y}, k_{**} - \boldsymbol{k}_*^t (K + \sigma^2 I)^{-1} \boldsymbol{k}_*)$$

$\Rightarrow$ Posterior mean is solution of **kernel ridge regression**!

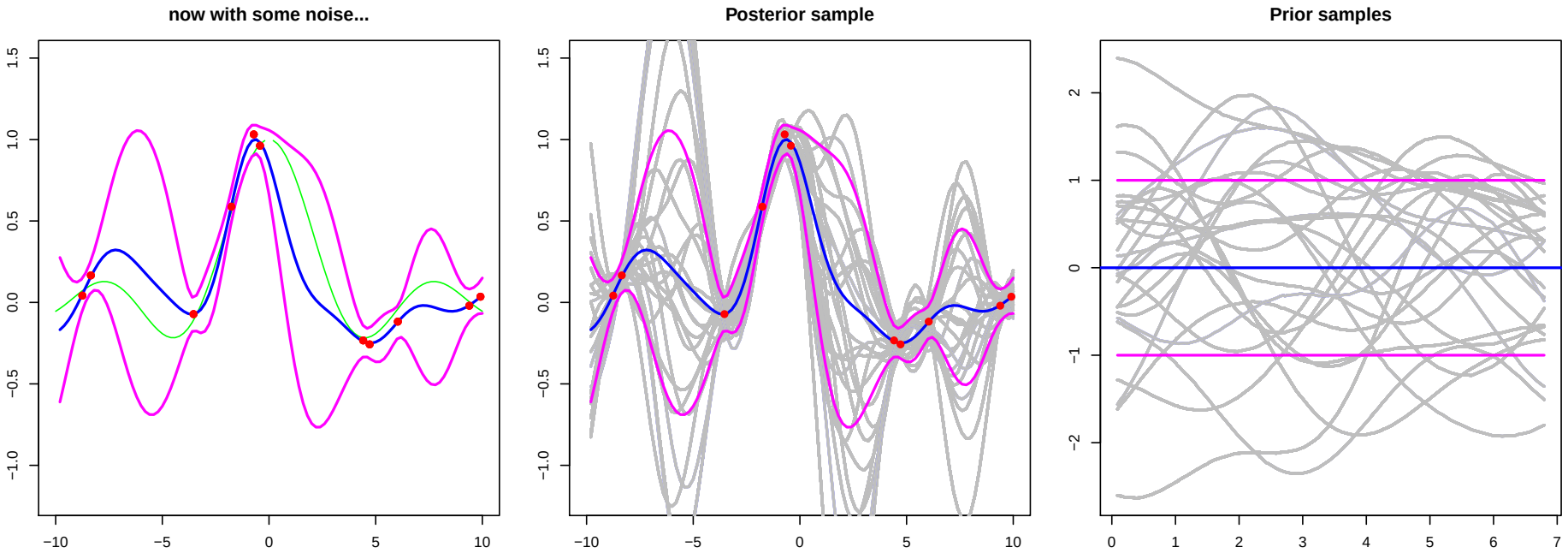
Noisy observations: examples



Noisy observations: $y = f(\boldsymbol{x}) + \eta,\ \eta \sim \mathcal{N}(0, \sigma^2)$

Mean predictions: $\hat{\boldsymbol{f}}_* = K_*(K + \sigma^2 I)^{-1}\boldsymbol{y}$.

Gaussian processes for regression



- **Left:** 11 training points generated as $y = \sin(x)/x + \nu$, $\nu \sim \mathcal{N}(0, 0.01)$
  Covariance $k(\boldsymbol{x}_p, \boldsymbol{x}_q) = \exp(-\frac{1}{2l^2}\|\boldsymbol{x}_p - \boldsymbol{x}_q\|^2) + \sigma^2\delta_{p,q}$.
  100 test points uniformly chosen from $[-10, 10] \rightsquigarrow$ matrix X_*.
  Mean prediction $E[\boldsymbol{f}_*|X_*, X, \boldsymbol{y}]$ and $\pm$std.dev.
- **Middle:** samples drawn from posterior $\boldsymbol{f}_*|X_*, X, \boldsymbol{y}$.
- **Right:** samples drawn from prior $\boldsymbol{f} \sim \mathcal{N}(\boldsymbol{0}, K(X, X))$.

Covariance Functions

- A GP specifies a distribution over functions $f(\boldsymbol{x})$, characterized by mean function $m(\boldsymbol{x})$ and covariance function $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$.
- Finite subset evaluated at n inputs $\rightsquigarrow$ Gaussian distribution:

  $$\boldsymbol{f}(X) = (f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n))^t \sim \mathcal{N}(\boldsymbol{\mu}, K),$$

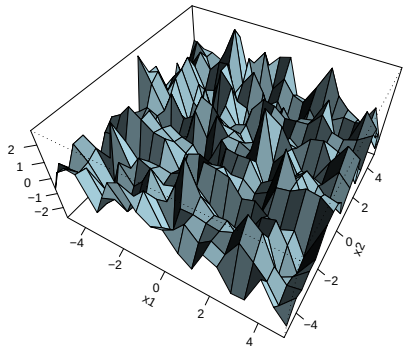
  where K is the covariance matrix with entries $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$.
- Covariance matrices are **symmetric positive semi-definite:**

  $$K_{ij} = K_{ji} \quad \text{and} \quad \boldsymbol{x}^t K \boldsymbol{x} \geq 0, \ \forall \boldsymbol{x}.$$

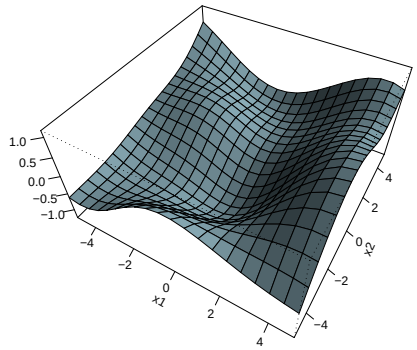
- We already know that **Mercer kernels** have this property
  $\rightsquigarrow$ all Mercer kernels define proper covariance functions in GPs.
- Kernels frequently have **additional parameters.**
- The **noise variance** in the observation model
  $y = f(\boldsymbol{x}) + \eta, \eta \sim \mathcal{N}(0, \sigma^2)$ is another parameter.
- How should we choose these parameters? $\rightsquigarrow$ **model selection.**

Model Selection

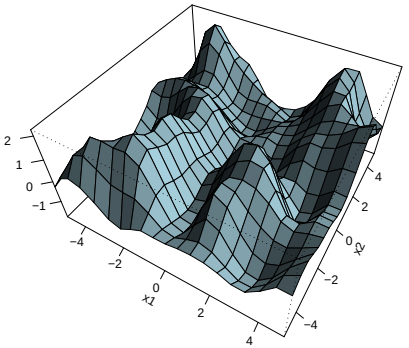
- **top left:** sample function from prior $\boldsymbol{f} \sim \mathcal{N}(\mathbf{0}, K(X, X))$ with **covariance function** $k(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\frac{1}{2l^2}\|\boldsymbol{x} - \boldsymbol{x}'\|^2)$. Length scale $l = 10^{-0.5}$ small $\rightsquigarrow$ highly varying function.
- **bottom left:** same for $l = 10^0$ $\rightsquigarrow$ smoother function
- **top right:** same for $l = 10^{0.5}$ $\rightsquigarrow$ even smoother...
- **bottom right:** almost linear function for $l = 10^1$.



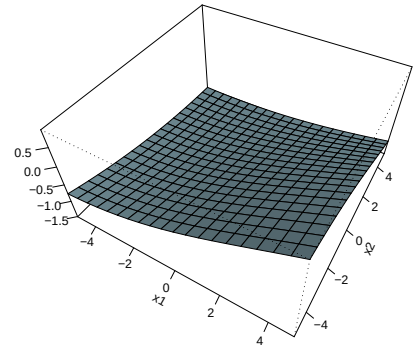
length scale: 10^−0.5, sample no. 1



length scale: 10^0.5, sample no. 1



length scale: 10^0, sample no. 1



length scale: 10^1, sample no. 1

Model Selection (2)

- How to select the parameters?
- One possibility: maximize **marginal likelihood:**

$$p(\boldsymbol{y}|X) = \int p(\boldsymbol{y}|\boldsymbol{f}, X)p(\boldsymbol{f}|X)\, d\boldsymbol{f}.$$

- We do not need to integrate: we know that

$$\boldsymbol{f}|X \sim \mathcal{N}(\boldsymbol{0}, K) \quad \text{and} \quad \boldsymbol{y} = \boldsymbol{f} + \eta,\ \eta \sim \mathcal{N}(0, \sigma^2).$$

  Since η does not depend on X, the variances simply add:

$$\boldsymbol{y}|X \sim \mathcal{N}(\boldsymbol{0}, K + \sigma^2 I).$$

- Possible strategy:
  Select parameters on a grid and choose maximum.
- Or: Compute derivatives of marginal likelihood and use gradient descent.

Model Selection (3)

- **Example problem:** $y = \sin(x)/x + \eta,\ \eta \sim \mathcal{N}(0, 0.01)$.
- **Log marg. likeli.** $= \log \mathcal{N}(\mathbf{0}, K + \sigma^2 I) =$

$$\underbrace{-\frac{1}{2}\boldsymbol{y}^t (K + \sigma^2 I)^{-1} \boldsymbol{y}}_{\text{data fit}} \underbrace{- \frac{1}{2} \log |K + \sigma^2 I|}_{\text{complexity penalty}} \underbrace{- \frac{n}{2} \log(2\pi)}_{\text{norm. constant}} .$$

- 2d-Example with Gaussian RBF:

$$(K + \sigma^2 I) = \begin{pmatrix} 1 + \sigma^2 & a \\ a & 1 + \sigma^2 \end{pmatrix} \Rightarrow |K + \sigma^2 I| = (1 + \sigma^2)^2 - a^2 > 0$$

Note that $a \to 0$ if length scale $l \to 0$

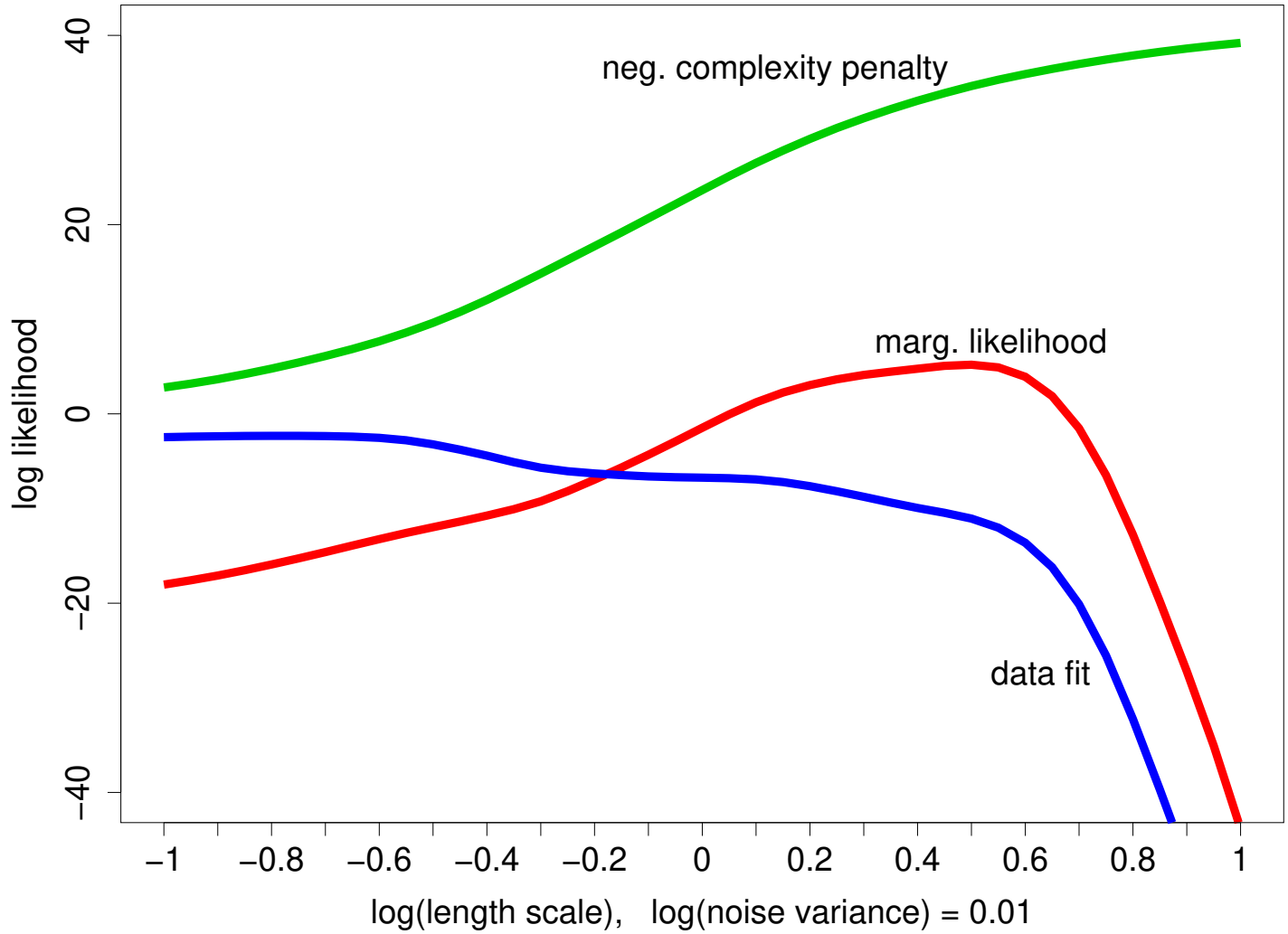
$\rightsquigarrow$ complexity penalty has high values for small length scales.

Matrix inverse includes a dominating factor $|K + \sigma^2 I|^{-1}$

$\rightsquigarrow$ data fit term also high for small l.

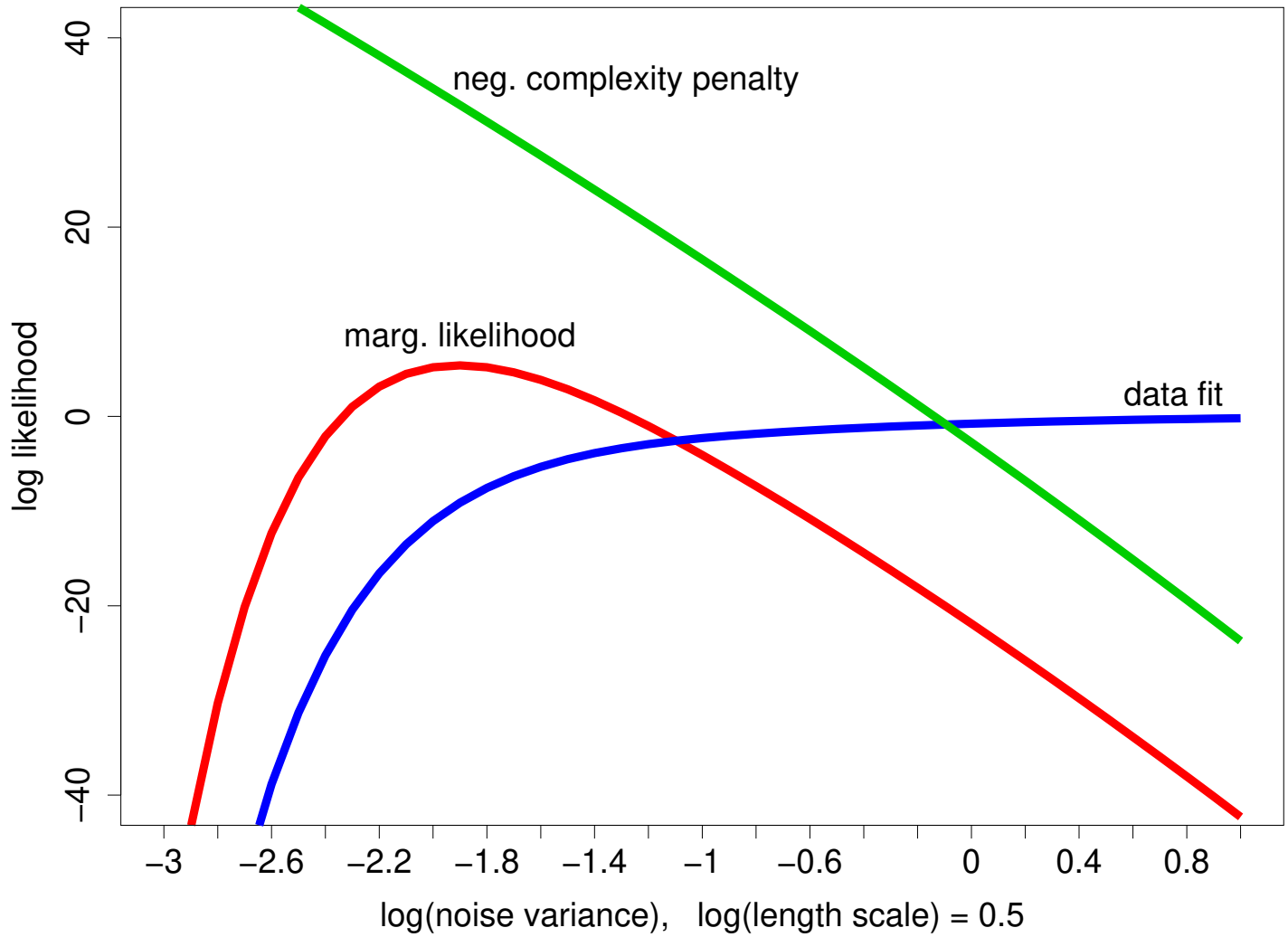
Model Selection (4)

Fixing $\sigma^2 = 0.01$ and **varying length scale** l:



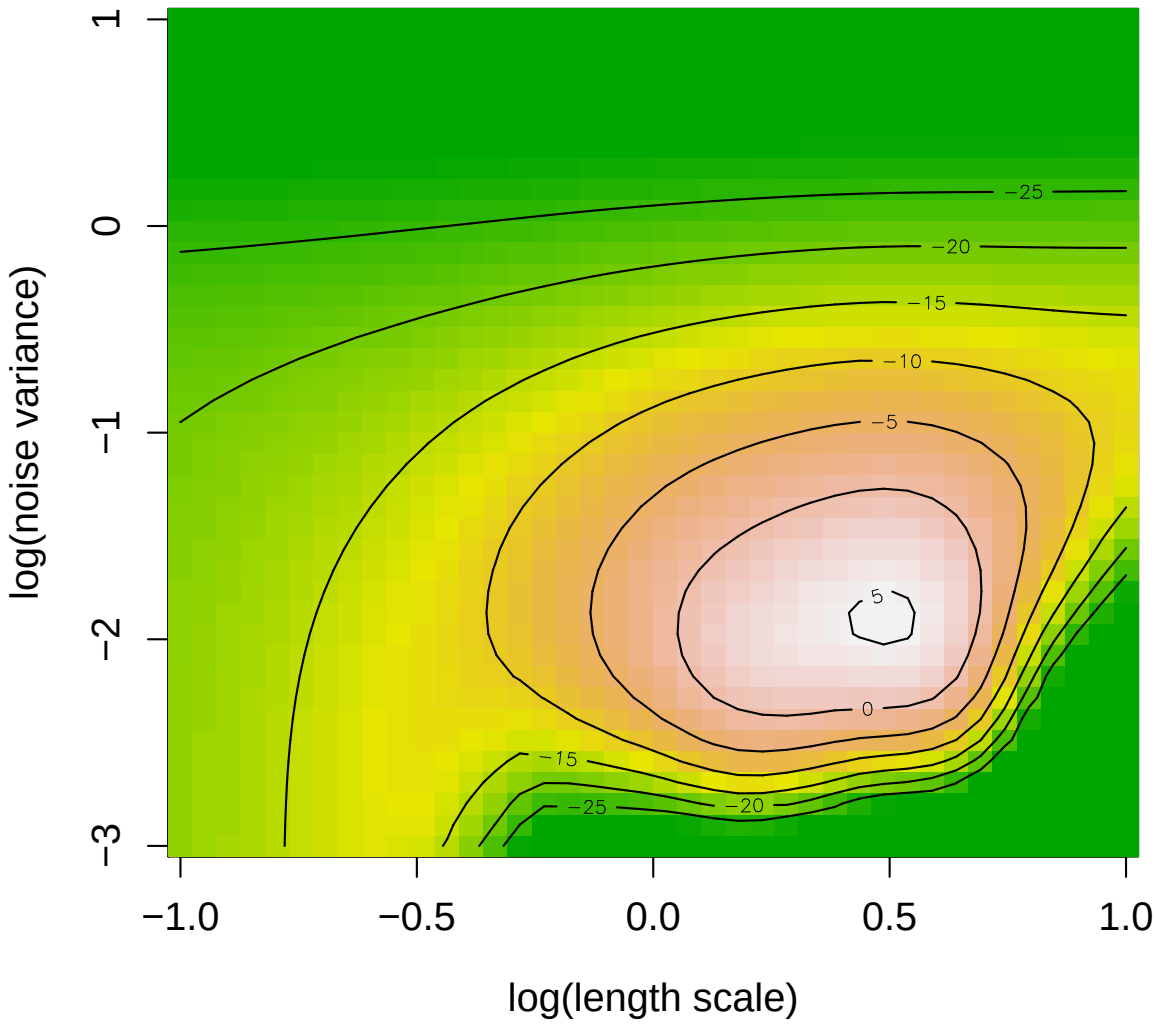
Model Selection (5)

Fixing length scale $l = 0.5$ and **varying the noise level** σ^2:

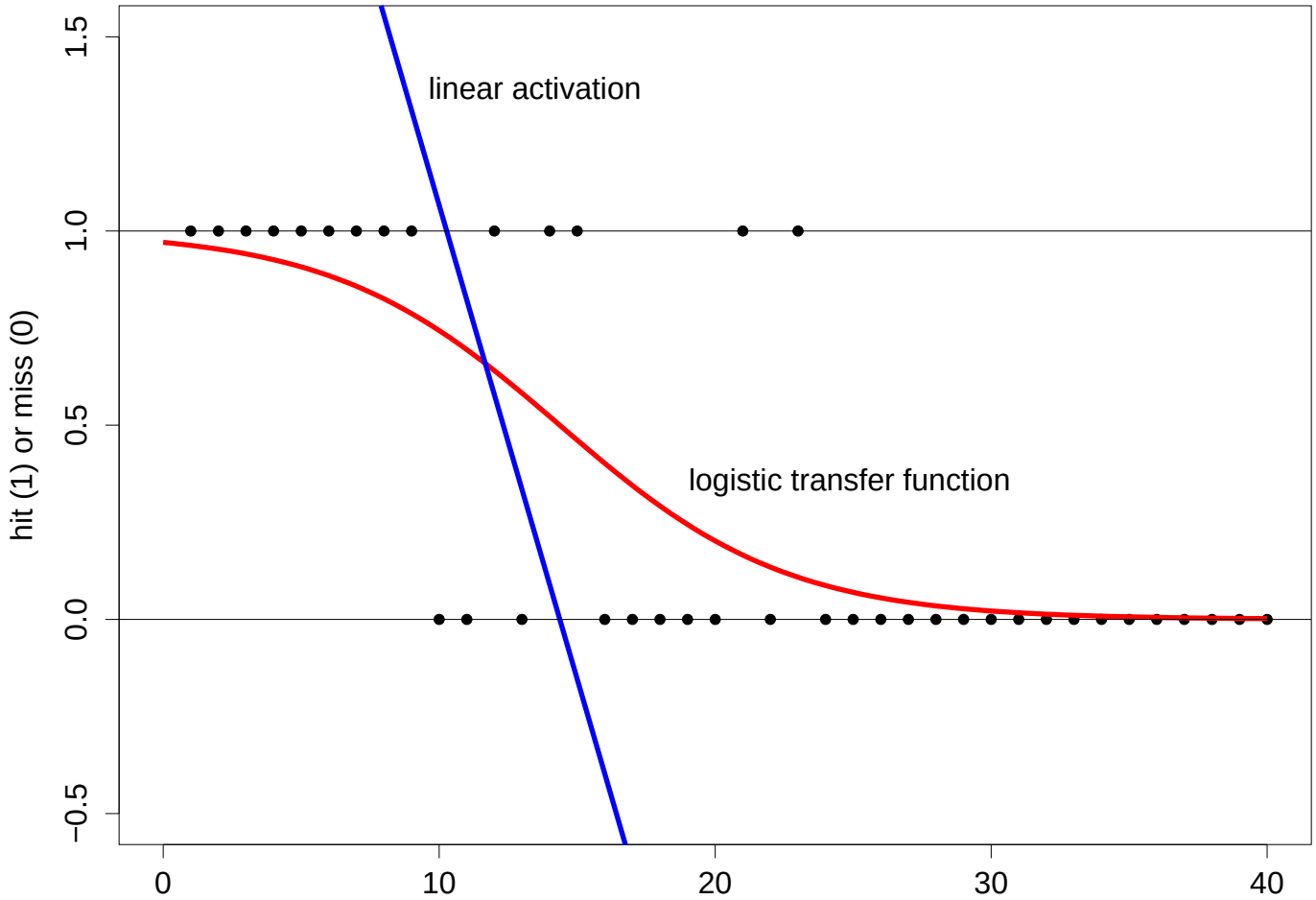


Model Selection (6)

Varying both σ^2 and l:



Classification: Basket Ball Example



Adapted from Fig. 7.5.1 in (B. Flury)

Classical Logistic Regression

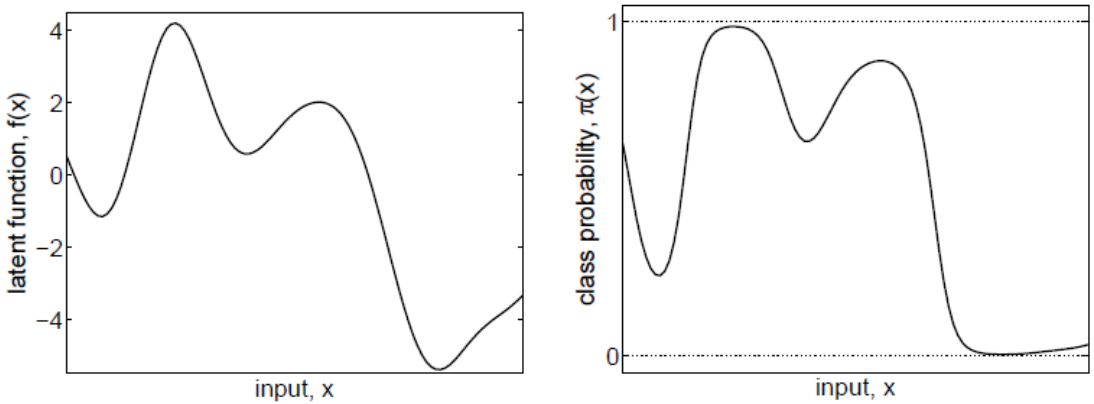
- Targets $y \in \{0, 1\}$
  $\rightsquigarrow$ Bernoulli RV with "success probability" $\pi(\boldsymbol{x}) = P(1|\boldsymbol{x})$.
- Likelihood: $P(\boldsymbol{y}|X, f) = \prod_{i=1}^{n} (\pi_f(\boldsymbol{x}_i))^{y_i} (1 - \pi_f(\boldsymbol{x}_i))^{1-y_i}$
- **Linear logistic regression:** unbounded $f(\boldsymbol{x}) = \boldsymbol{w}^t \boldsymbol{x}$ ("**activation**")
  Bounded estimates: pass $f(\boldsymbol{x})$ through **logistic transfer function**
  $\sigma(f(\boldsymbol{x})) = \frac{e^{f(\boldsymbol{x})}}{1+e^{f(\boldsymbol{x})}} = \frac{1}{1+e^{-f(\boldsymbol{x})}}$ and set $\pi_f(\boldsymbol{x}) = \sigma(f(\boldsymbol{x}))$.
- Newton method for maximizing the log posterior
  $J(\boldsymbol{w}) := \log p(\boldsymbol{y}|X, \boldsymbol{w}) + \log p(\boldsymbol{w})$:

$$\boldsymbol{w}^{(r+1)} = \boldsymbol{w}^{(r)} - \{E[H]\}^{-1} \frac{\partial}{\partial \boldsymbol{w}} J(\boldsymbol{w})$$

- Kernel trick: expand $\boldsymbol{w} = X^t \boldsymbol{\alpha}$, substitute dot products by kernel function $k(\boldsymbol{x}, \boldsymbol{x}') \rightsquigarrow$ **kernel logistic regression**.

GP Classification

- Place GP prior over "latent" function $\boldsymbol{f}(\boldsymbol{x}) \sim \mathcal{GP}(0, k(\boldsymbol{x}, \boldsymbol{x}'))$.
- "Squash" it through logistic function $\rightsquigarrow$ prior on $\pi(\boldsymbol{x}) = \sigma(f(\boldsymbol{x}))$.



(Rasmussen & Williams, 2006)

- **Problem:** Bernoulli likelihood $\rightsquigarrow$ predictive distribution $p(y_* = 1|X, \boldsymbol{y}, \boldsymbol{x}_*)$ cannot be calculated analytically.
- Possible **solution:** use **Laplace approximation**.
- **Observation:** MAP classification boundary is identical with boundary obtained from **kernel logistic regression**.

GP Classification using Laplace's approximation

- Prior $\boldsymbol{f}|X \sim \mathcal{N}(\boldsymbol{0}, K)$. Bernoulli likelihood:

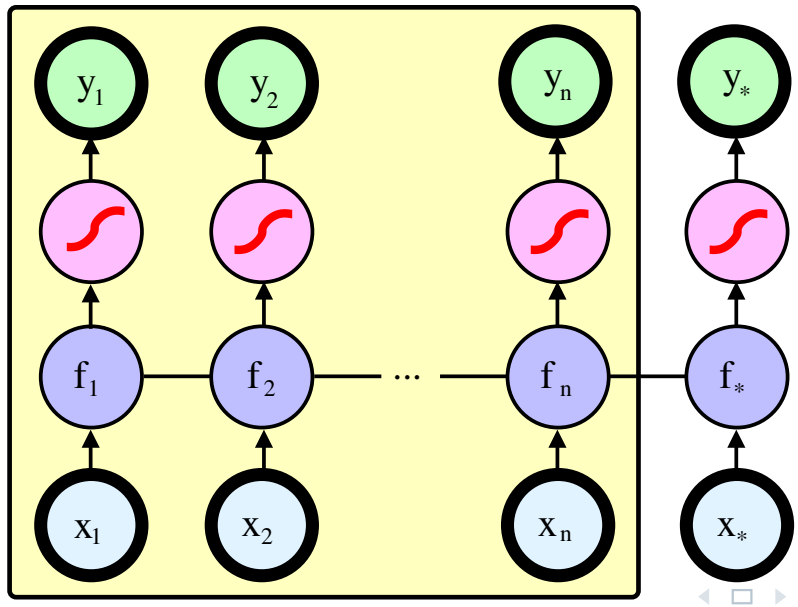
$$p(\boldsymbol{y}|X, \boldsymbol{f}) = \prod_{i=1}^{n} \left(\sigma(f(\boldsymbol{x}_i))\right)^{y_i} \left(1 - \sigma(f(\boldsymbol{x}_i))\right)^{1-y_i}.$$

- Gaussian approximation of posterior:

$$p(\boldsymbol{f}|X, \boldsymbol{y}) \approx \mathcal{N}(\hat{\boldsymbol{f}}, H^{-1}).$$

- Predictions: compute

$$p(y_* = 1|\boldsymbol{y}, \boldsymbol{x}_*, X) = \int \sigma(f_*) p\underbrace{(f_* \mid \boldsymbol{y},\ \boldsymbol{x}_*,\ X)}_{\text{latent function at } \boldsymbol{x}_*} df_* = \mathbb{E}_{p(f_*|\boldsymbol{y},\boldsymbol{x}_*,X)}(\sigma)$$



GP Classification using Laplace's approximation

- First **predict latent function** at test case $\boldsymbol{x}_*$:

$$p(f_*|\boldsymbol{y}, \boldsymbol{x}_*, X) = \int \underbrace{p(f_*|\boldsymbol{f}, \boldsymbol{x}_*, X)}_{\text{Gaussian}} \quad \underbrace{p(\boldsymbol{f}|X, \boldsymbol{y})d\boldsymbol{f}}_{\text{approx. Gaussian } \mathcal{N}(\hat{\boldsymbol{f}}, H^{-1})}$$

$$\approx \mathcal{N}(\mu_*, \sigma_*), \text{ with}$$

$$\mu_* = \boldsymbol{k}_*^t K^{-1} \hat{\boldsymbol{f}},$$

$$\sigma_* = k_{**} - \boldsymbol{k}_*^t \tilde{K}^{-1} \boldsymbol{k}_*$$

- Then use **Monte Carlo approximation**

$$p(y_*|\boldsymbol{y}, \boldsymbol{x}_*, X) = \mathbb{E}_{p(f_*|\boldsymbol{y}, \boldsymbol{x}_*, X)}(\sigma) \approx \frac{1}{S} \sum_{s=1}^{S} \sigma(f_*^s(\boldsymbol{x}_*)),$$

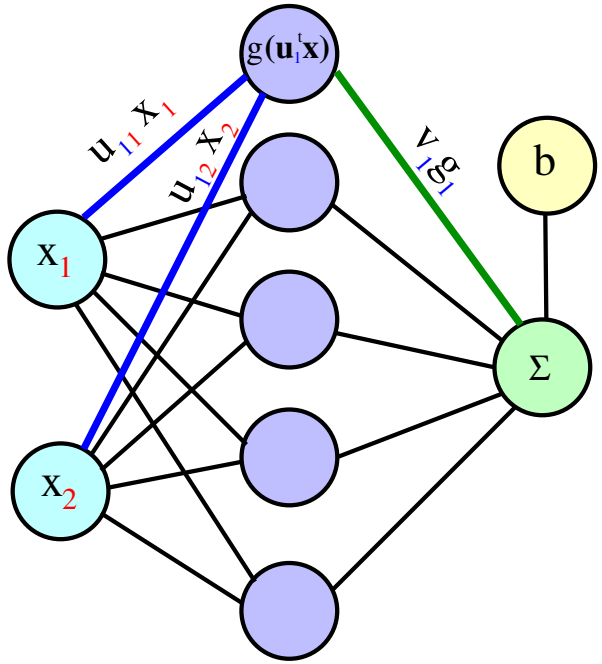
where f_*^s are samples from the (approximated) distribution over latent function values.

GPs and Neural networks

Consider a neural network for regression (square loss) with one hidden layer:

$$p(y|\boldsymbol{x}, \boldsymbol{\theta}) = \mathcal{N}(f(\boldsymbol{x}; \boldsymbol{\theta}), \sigma^2),$$

$$f(\boldsymbol{x}) = b + \sum_{j=1}^{n_H} v_j g(\boldsymbol{x}; \boldsymbol{u}_j).$$



Bayesian treatment: **i.i.d. prior assumptions over weights:** indep. zero-mean Gaussian priors for b and v, with variance σ_b^2 and σ_v^2, and independent (arbitrary) priors for components of the weight vector $\boldsymbol{u}_j$ at the j-th hidden unit.

GPs and Neural networks

- Mean and covariance:

$$m(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{\theta}}[f(\boldsymbol{x})] = \overbrace{\mathbb{E}[b]}^{=0} + \sum_{j=1}^{n_H} \mathbb{E}[v_j g(\boldsymbol{x}; \boldsymbol{u}_j)]$$

$$\overset{(v \text{ indep. of } u)}{=} \sum_{j=1}^{n_H} \underbrace{\mathbb{E}[v_j]}_{=0} \mathbb{E}[g(\boldsymbol{x}; \boldsymbol{u}_j)] = 0.$$

$$k(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}_{\boldsymbol{\theta}}[f(\boldsymbol{x}) f(\boldsymbol{x}')] = \sigma_b^2 + \sum_{j=1}^{n_H} \sigma_v^2 \mathbb{E}_{\boldsymbol{u}}[g(\boldsymbol{x}; \boldsymbol{u}_j) g(\boldsymbol{x}'; \boldsymbol{u}_j)].$$

- All hidden units are identically distributed
  $\rightsquigarrow$ the sum is over n_H i.i.d. RVs. Assume activation g is bounded
  $\rightsquigarrow$ all moments of the distribution will be bounded
  $\rightsquigarrow$ **central limit theorem applicable**

GPs and Neural networks

Suppose $\{X_1, \ldots, X_n\}$ is a sequence of i.i.d. RVs with $\mathbb{E}[X_i] = \mu$ and $\mathrm{Var}[X_i] = \sigma^2 < \infty$. Then $\sqrt{n}\,(S_n - \mu) \xrightarrow{d} \mathcal{N}(0, \sigma^2)$ as $n \to \infty$.

- The covariance between any pair of function values $(f(\boldsymbol{x}), f(\boldsymbol{x}'))$ converges to the covariance of two Gaussian RVs
  $\rightsquigarrow$ Joint distribution of n function values is multivariate Gaussian
  $\rightsquigarrow$ **we get a GP as $n_H \to \infty$.**
- For specific activations, the **neural network covariance function** can be computed analytically (Williams 1998).
- A **three-layer network** with and **infinitely wide hidden layer** can be **interpreted as a GP.**

Summary

- **GPs: fully probabilistic models**
  $\rightsquigarrow$ posterior $p(\boldsymbol{f}_*|X, \boldsymbol{y}, \boldsymbol{x}_*)$.
- Uniquely defined by specifying **covariance function**.
- **Mathematically simple:**
  we only need to calculate **conditionals of Gaussians**!
- **Connections:**
  regression: MAP(GP_r) = kernel ridge reg.
  classification: MAP(GP_c) = kernel logistic reg.
  $\text{GP}_c \approx$ probabilistic version of SVM.

  A three-layer network with an infinitely wide hidden layer can be **interpreted as a GP with the neural network covariance function.**