ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf | Informatik
Computer Science

Introduction to Machine Learning

Non-linear prediction with kernels (continued)

Prof. Andreas Krause
Learning and Adaptive Systems (las.ethz.ch)

The „Kernel Trick“

- Express problem s.t. it only depends on inner products
- Replace inner products by kernels

$$\mathbf{x}_i^T \mathbf{x}_j \quad \Rightarrow \quad k(\mathbf{x}_i, \mathbf{x}_j)$$

- This „trick“ is very widely applicable!

Definition: kernel functions

- Data space *X*
- A **kernel** is a function $k : X \times X \to \mathbb{R}$ satisfying
- 1) Symmetry: For any $\mathbf{x}, \mathbf{x}' \in X$ it must hold that

$$k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$$

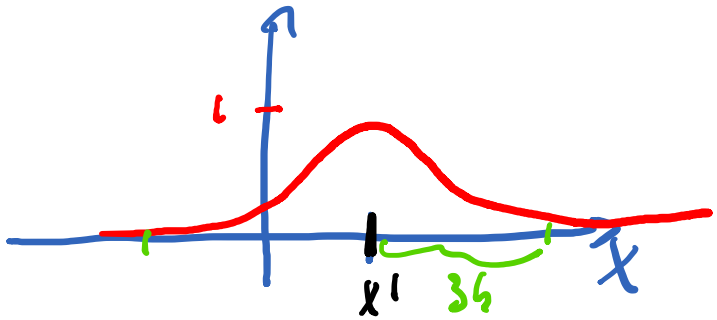
- 2) Positive semi-definiteness: For any *n*, any set $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subseteq X$, the kernel (Gram) matrix

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

must be positive semi-definite

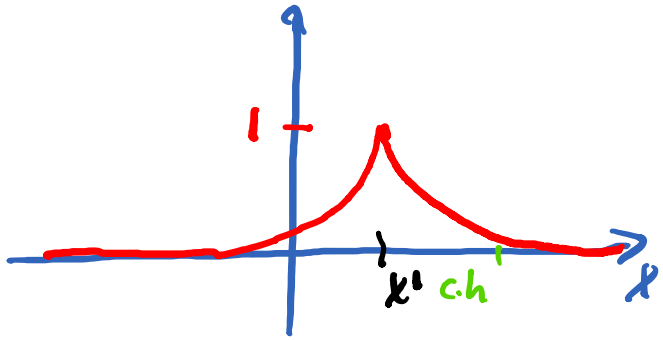
Examples of kernels on $\mathbb{R}^d$

- Linear kernel: $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$
- Polynomial kernel: $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + 1)^d$
- Gaussian (RBF, squared exp. kernel): $k(\mathbf{x}, \mathbf{x}') = \exp(-||\mathbf{x} - \mathbf{x}'||_2^2 / 2h^2)$



Bandwidth / Lengthscale parameter

- Laplacian kernel: $k(\mathbf{x}, \mathbf{x}') = \exp(-||\mathbf{x} - \mathbf{x}'||_1 / h)$



Kernel engineering (composition rules)

- Suppose we have two kernels

$$k_1 : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \qquad k_2 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

  defined on data space *X*

- Then the following functions are valid kernels:

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')$$

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')\, k_2(\mathbf{x}, \mathbf{x}')$$

$$k(\mathbf{x}, \mathbf{x}') = c\, k_1(\mathbf{x}, \mathbf{x}') \text{ for } c > 0$$

$$k(\mathbf{x}, \mathbf{x}') = f(k_1(\mathbf{x}, \mathbf{x}'))$$

Suppose $v: z \to x$

$k(z, z') = k_1(v(z), v(z'))$ is valid kernel.

Pf: $k_1(x, x') = \phi(x)^T \phi(x')$

$k(z, z') = \phi(v(z))^T \phi(v(z'))$

where *f* is a polynomial with positive coefficients or the exponential function

Kernels as *similarity functions*

- Recall Perceptron (and SVM) classification rule:

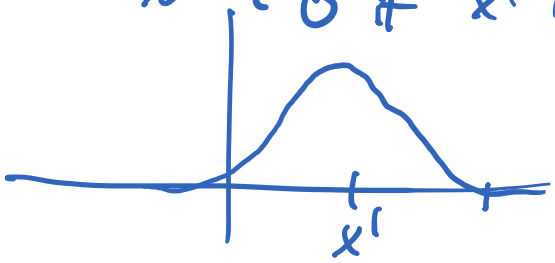
$$y = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x})\right)$$

$\approx \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i [\mathbf{x} \text{ "close" to } \mathbf{x}_i]\right)$

$f(\mathbf{x})$

- Consider Gaussian kernel $k(\mathbf{x}, \mathbf{x}') = \exp(-||\mathbf{x} - \mathbf{x}'||^2/h^2)$

$\approx$ { 1 if $\mathbf{x}'$ is close to $\mathbf{x}$
0 if $\mathbf{x}'$ is far from $\mathbf{x}$



K-NN vs. Kernel Perceptron

- k-Nearest Neighbor:

$$y = \text{sign}\left(\sum_{i=1}^{n} y_i [\mathbf{x}_i \text{ among } k \text{ nearest neighbors of } \mathbf{x}]\right)$$

- Kernel Perceptron:

$$y = \text{sign}\left(\sum_{i=1}^{n} y_i \alpha_i k(\mathbf{x}_i, \mathbf{x})\right)$$

Comparison: k-NN vs Kernelized Perceptron

Method	*k-NN*	*Kernelized Perceptron*
Advantages	No training necessary	Optimized weights can lead to improved performance Can capture „global trends“ with suitable kernels Depends on „wrongly classified“ examples only
Disadvantages	Depends on all data ➔ inefficient	Training requires optimization

Parametric vs nonparametric learning

- Parametric models have finite set of parameters
- **Example**: Linear regression, linear Perceptron, ...

- Nonparametric models grow in complexity with the size of the data
  - Potentially much more expressive
  - But also more computationally complex – Why?
- **Example**: Kernelized Perceptron, k-NN, ...

- Kernels provide a principled way of deriving non-parametric models from parametric ones

Where are we?

- We've seen how to kernelize the perceptron
- Discussed properties of kernels, and seen examples

- Next question:
  - Can we use the kernel trick beyond the perceptron?

Kernelized SVM

- The support vector machine

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\} + \lambda ||\mathbf{w}||_2^2$$

can also be kernelized

How to kernelize the objective?

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\} + \lambda ||\mathbf{w}||_2^2$$

(*)

$$w = \sum_{j=1}^{n} \alpha_j y_j x_j$$

$$(*) = \max\{0, 1 - y_i \left(\sum_{j=1}^{n} \alpha_j y_j x_j\right)^T x_i\}$$

$$= \max\{0, 1 - y_i \sum_{j=1}^{n} \alpha_j y_j (x_j^T x_i)\}$$

$k(x_j, x_i)$

$$= \max\{0, \alpha^T k_i\} \quad \text{where } k_i^T = [y_i k(x_i, x_1), \ldots y_n k(x_i, x_n)]$$

How to kernelize the regularizer?

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\} + \underbrace{\lambda ||\mathbf{w}||_2^2}_{(*)}$$

$$(*) = \lambda\, w^T w = \lambda \Big(\sum_{i=1}^{n} \alpha_i y_i x_i \Big)^T \Big(\sum_{j=1}^{n} \alpha_j y_j x_j \Big)$$

$$= \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \underbrace{x_i^T x_j}_{k(x_i, x_j)}$$

$$= \lambda\, \alpha^T D_y K D_y \alpha$$

$$\alpha^T = [\alpha_1 \; \dots \; \alpha_n]$$

$$D_y = \begin{pmatrix} y_1 & & 0 \\ & \ddots & \\ 0 & & y_n \end{pmatrix}$$

$$K = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{pmatrix}$$

Learning & prediction with kernel classifier

- Learning: Solve the problem

Per-ceptron:
$$\arg\min_{\alpha} \frac{1}{n}\sum_{i=1}^{n} \max\{0, -y_i \alpha^T \mathbf{k}_i\}$$
Or:

SVM:
$$\arg\min_{\alpha} \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i \alpha^T \mathbf{k}_i\} + \lambda \alpha^T \mathbf{D_y K D_y} \alpha$$

$$\mathbf{k}_i = [y_1 k(\mathbf{x}_i, \mathbf{x}_1), \ldots, y_n k(\mathbf{x}_i, \mathbf{x}_n)]$$

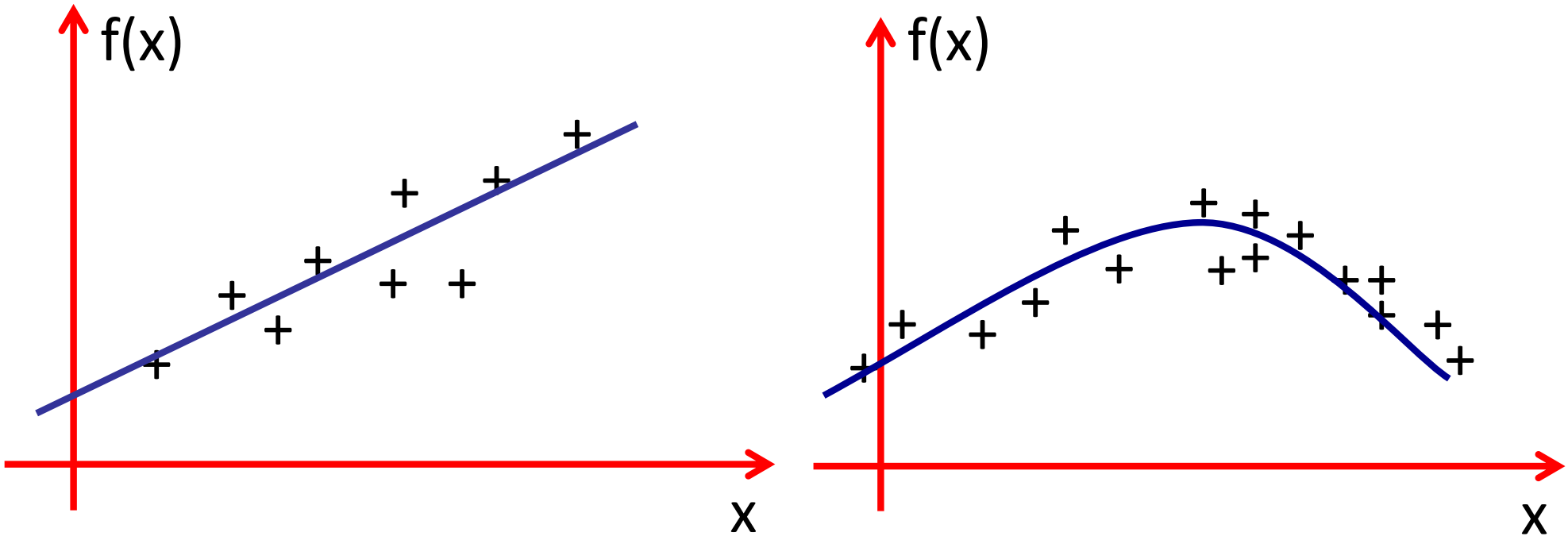
- Prediction: For data point ***x*** predict label *y* as

$$\hat{y} = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x})\right)$$

Demo: Kernelized SVM

Kernelized Linear Regression

- From linear to nonlinear regression:



- Can also kernelize linear regression
- Predictor has the form

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x})$$

Example: Kernelized linear regression

- Original (parametric) linear optimization problem

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \left(\mathbf{w}^T \mathbf{x}_i - y_i \right)^2 + \lambda ||\mathbf{w}||_2^2$$

- Similar as in perceptron, optimal lies in span of data:

$$\hat{\mathbf{w}} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i$$

Kernelizing linear regression

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \left(\mathbf{w}^T \mathbf{x}_i - y_i \right)^2 + \lambda ||\mathbf{w}||_2^2 \qquad \hat{\mathbf{w}} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i$$

(***) (*) (**)

$$(*) = \sum_{j=1}^{n} \alpha_j \underbrace{(\mathbf{x}_j^T \mathbf{x}_i)}_{k(x_i, x_j)} = \alpha^T k_i$$

$$k_i := \begin{pmatrix} k(x_i, x_1) \\ \vdots \\ k(x_i, x_n) \end{pmatrix}$$

$$(**) = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \mathbf{x}_i^T \mathbf{x}_j = \alpha^T K \alpha$$

$$K := \begin{pmatrix} | & & | \\ k_1 & \cdots & k_n \\ | & & | \end{pmatrix}$$

$$(***) \equiv \hat{\alpha} = \arg\min_{\alpha} \frac{1}{n} \underbrace{\sum_{i=1}^{n} (\alpha^T k_i - y_i)^2}_{\|\alpha^T K - y\|_2^2} + \lambda \alpha^T K \alpha$$

Kernelized linear regression

$$\hat{\alpha} = \arg\min_{\alpha_{1:n}} \frac{1}{n} \sum_{i=1}^{n} \Big(\sum_{j=1}^{n} \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) - y_i \Big)^2 + \lambda \alpha^T \mathbf{K} \alpha \qquad \mathbf{K} = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \dots & k(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix}$$

Learning & Predicting with KLR

- Learning: Solve least squares problem

$$\hat{\alpha} = \arg\min_{\alpha} \frac{1}{n} ||\alpha^T \mathbf{K} - \mathbf{y}||_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

Closed-form solution: $\hat{\alpha} = (\mathbf{K} + n\lambda \mathbf{I})^{-1} \mathbf{y}$

- Prediction: For data point ***x*** predict response *y* as

$$\hat{y} = \sum_{i=1}^{n} \hat{\alpha}_i k(\mathbf{x}_i, \mathbf{x})$$

Demo: Kernelized linear regression

KLR for the linear kernel

- What if $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$?

$$f(x) = \sum_{i=1}^{n} \alpha_i \, k(x_i, x) = \underbrace{\left[\sum_{i=1}^{n} \alpha_i \, x_i^T\right]}_{w^T} x = w^T x$$

Application: semi-parametric regression

- Often, parametric models are too „rigid“, and non-parametric models fail to extrapolate
- **Solution**: Use additive combination of linear and non-linear kernel function

$$k(\mathbf{x}, \mathbf{x}') = c_1 \exp(-||\mathbf{x} - \mathbf{x}'||_2^2 / h^2) + c_2 \mathbf{x}^T \mathbf{x}'$$

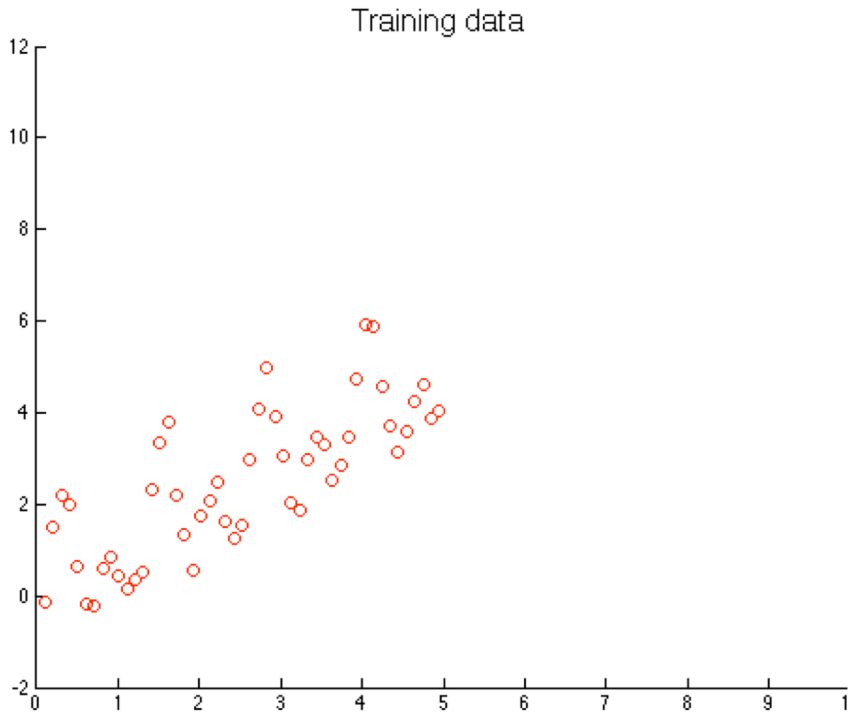
$$\hookrightarrow f(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x) = \sum_{i=1}^{n} \left(\alpha_i \left(c_1 \exp(-\|x_i - x\|_2^2 / h^2) \right) + c_2 x_i^T x \right)$$

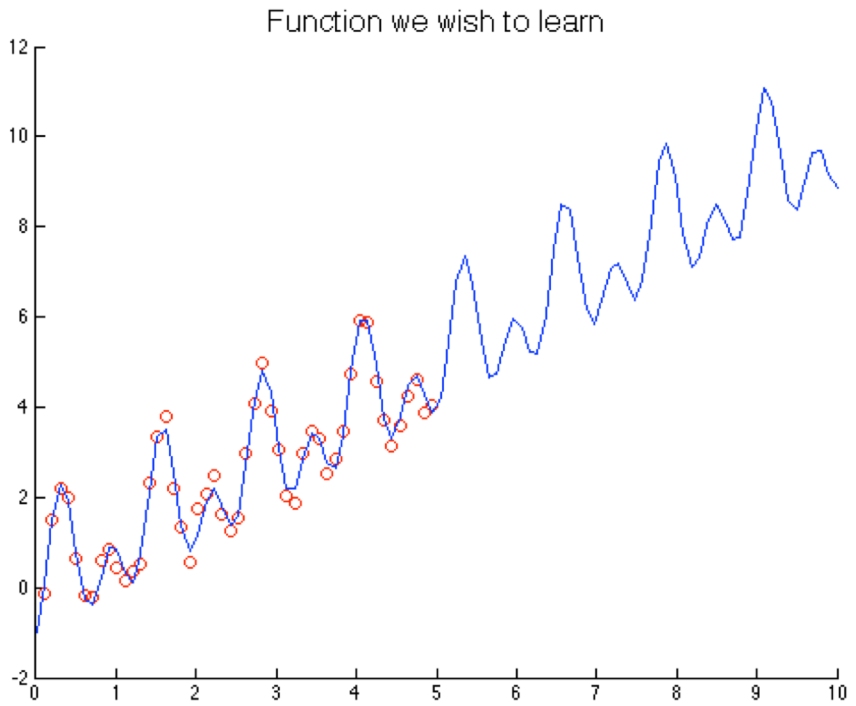
$$= c_1 \sum_{i=1}^{n} \alpha_i \exp(-\|x_i - x\|_2^2 / h^2) + \left(c_2 \sum_{i=1}^{n} \alpha_i x_i \right)^T x$$

$$= f_1(x) + w^T x$$

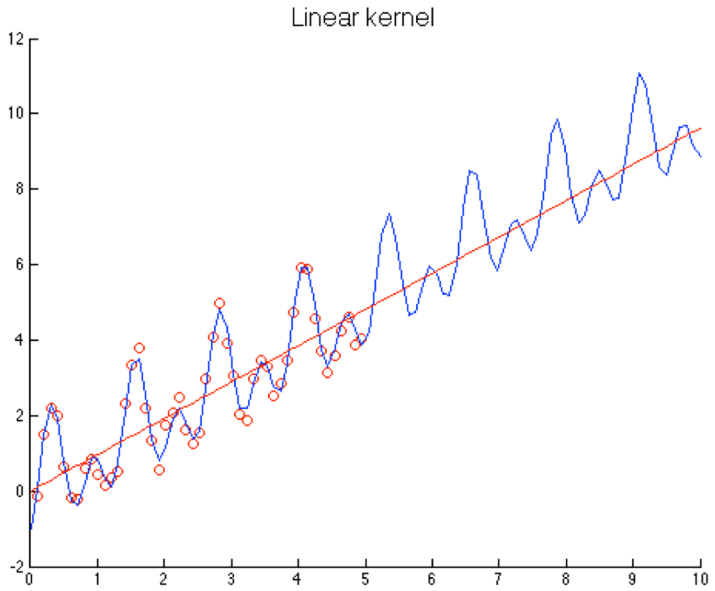
Demo: Semi-parametric KLR

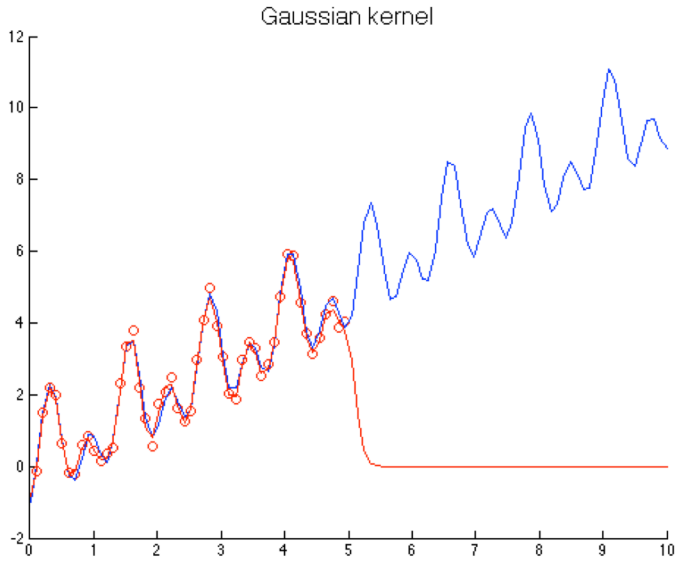
Example

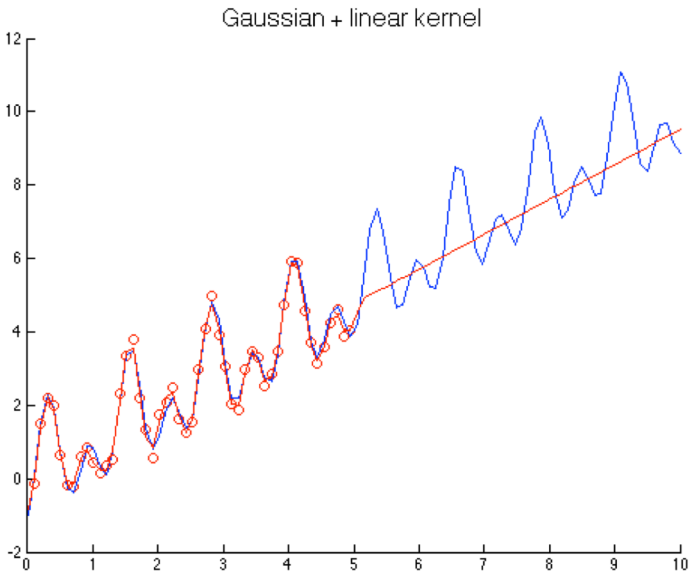

Training data
12
10
8
6
4
2
0
-2
0
1
2
3
4
5
6
7
8
9
1

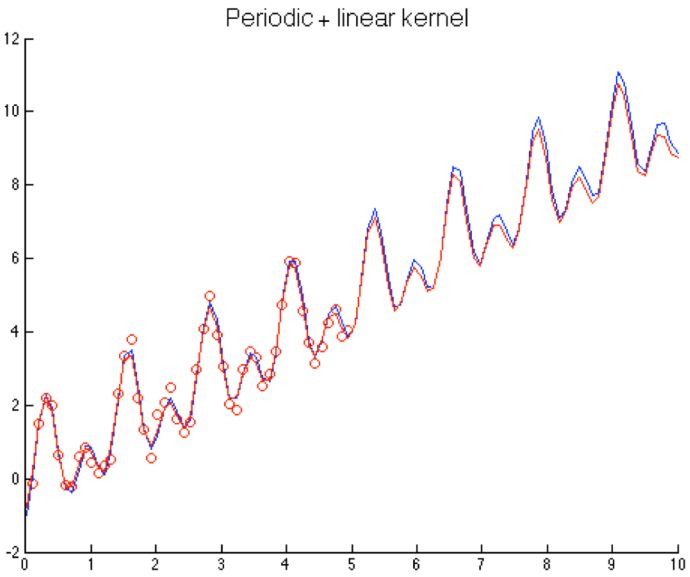

Function we wish to learn
12
10
8
6
4
2
0
-2
0
1
2
3
4
5
6
7
8
9
10

Example fits



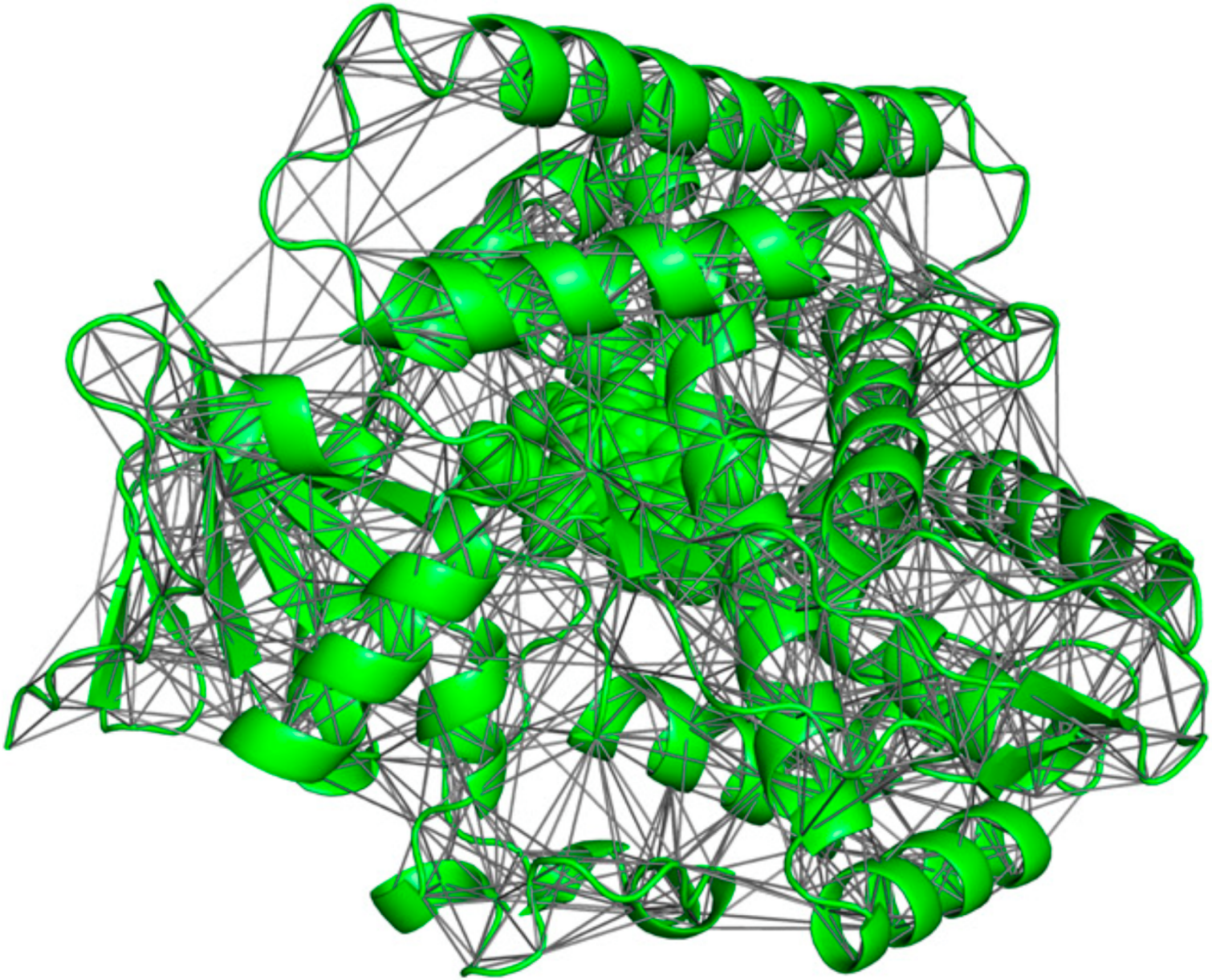






Application: Designing P450s chimeras

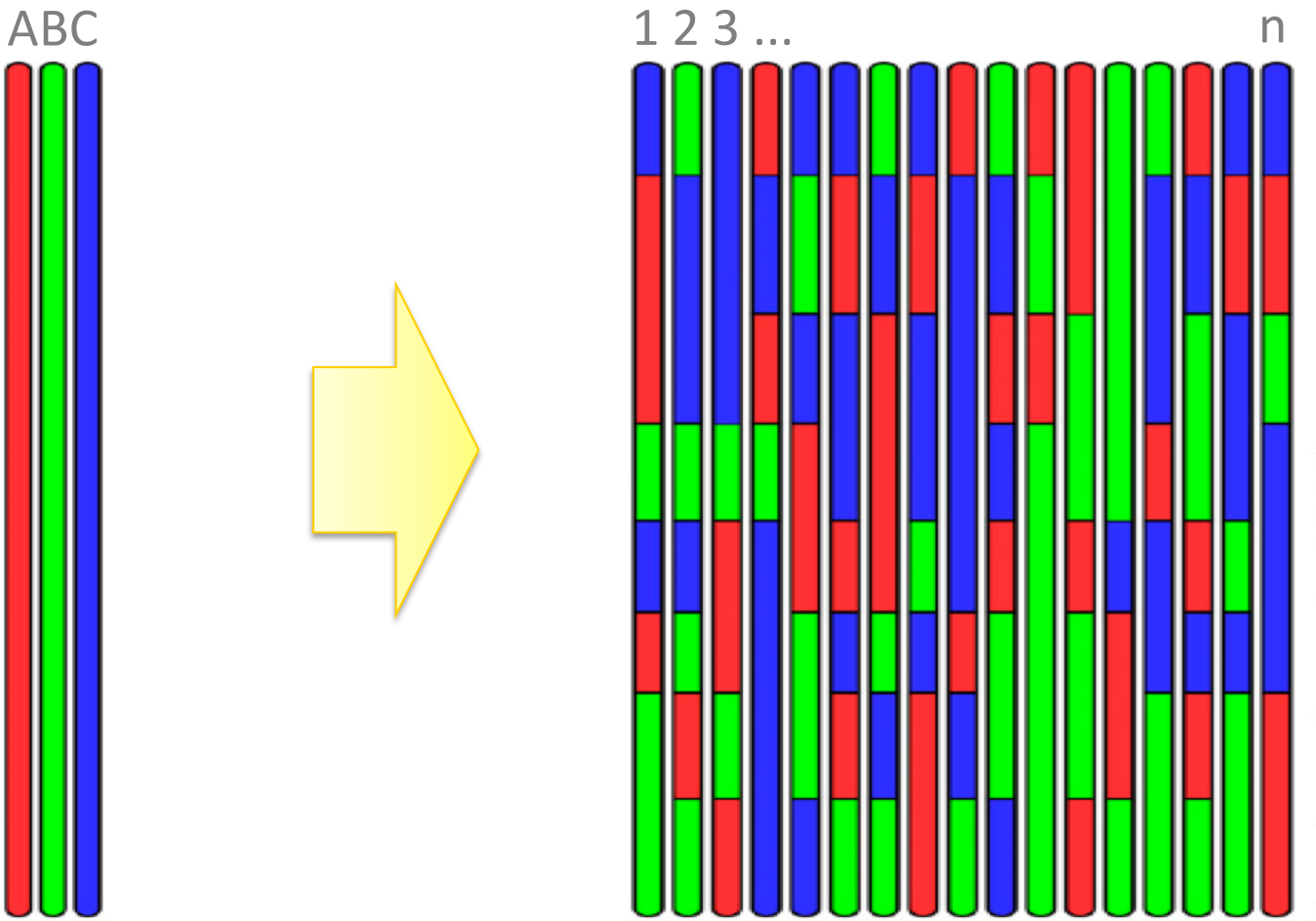
[with Phil Romero, Frances Arnold PNAS‘13]



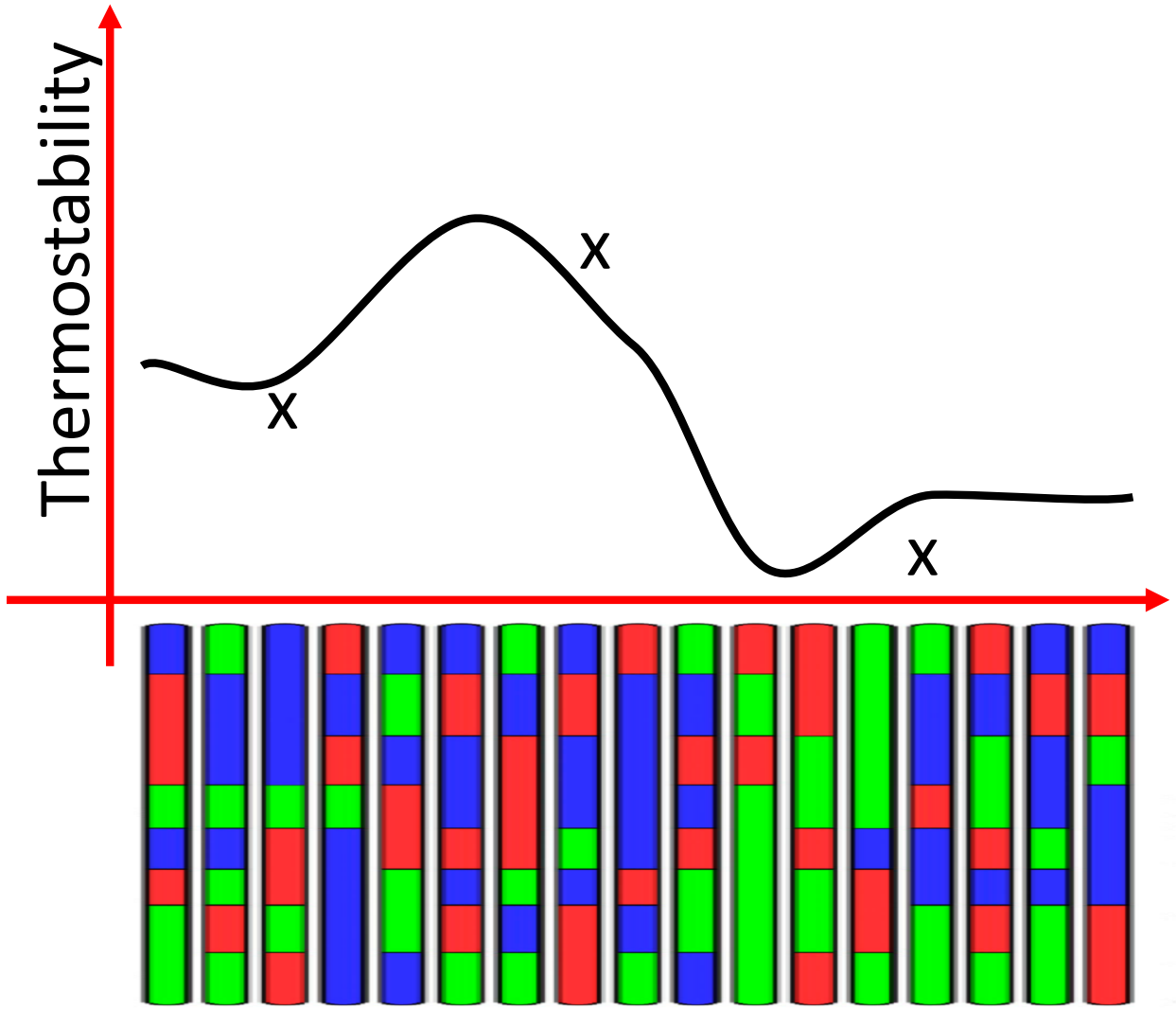
Design space

Parent sequences

Candidate designs

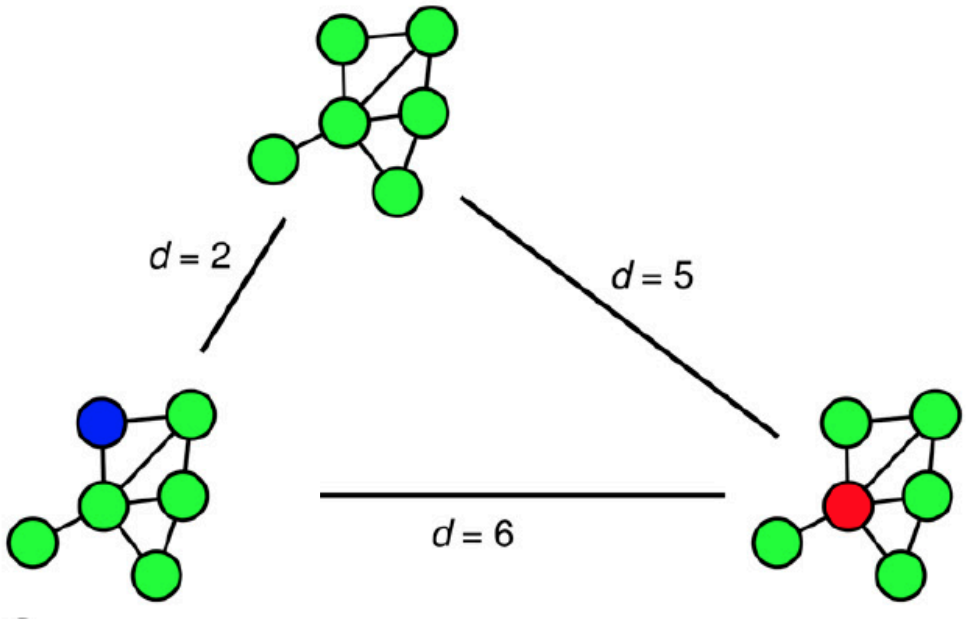


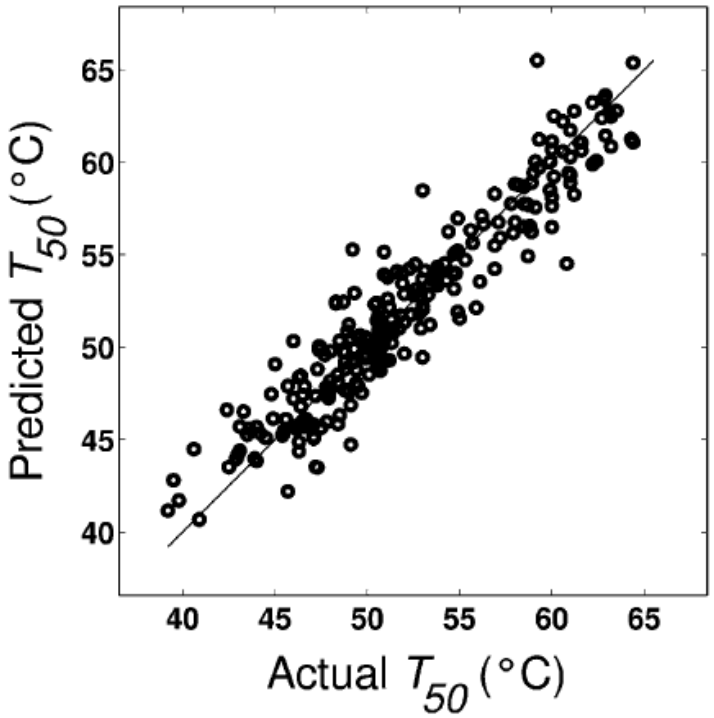
Protein Fitness Landscape



Application: Protein Engineering

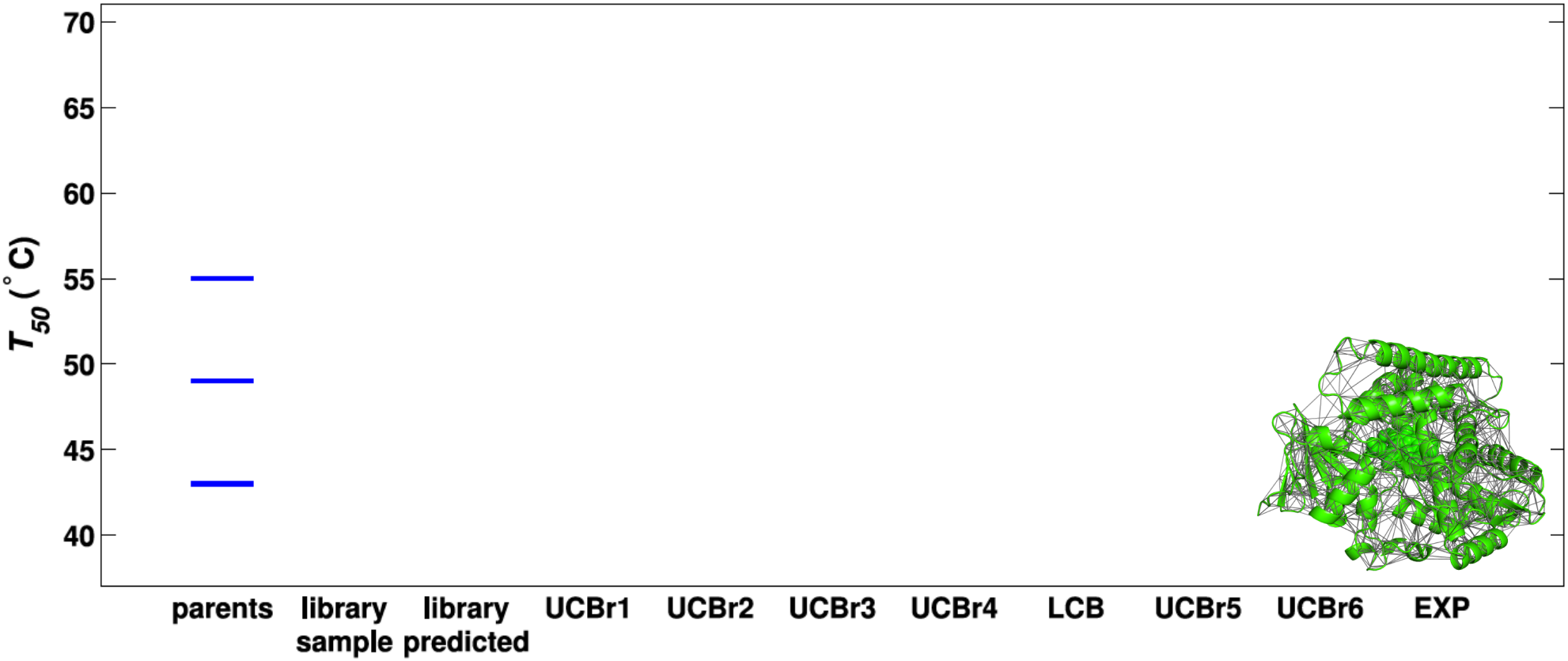
[with Romero, Arnold, PNAS '13]





Wet-lab results

[w Romero, Arnold PNAS ’13]



- Identification of new thermostable P450s chimera
- **5.3C more stable than best published sequence!**

Choosing kernels

- For a given kernel, how should we choose parameters?
  - Cross-validation! ☺
- How should we select suitable kernels?
  - Domain knowledge (dependent on data type)
  - «Brute force» (or heuristic) search
  - Use cross-validation
- **Learning kernels**
  - Much research on automatically selecting good kernels (Multiple Kernel Learning; Hyperkernels; etc.)

Parameter demo

What about overfitting?

- Kernels map to (very) high-dimensional spaces.
- Why do we hope to be able to learn?
- **First attempt of an answer:**
  (typically) # parameters << # dimensions. Why?

- Number of parameters = number of data points
  („non-parametric learning“)

What about overfitting?

- Kernels map to (very) high-dimensional spaces.
- Why do we hope to be able to learn?
- **Second attempt of an answer:**
- Overfitting can of course happen
  (if we choose poor parameters)
- Can combat overfitting by regularization
  - This is already built into kernelized linear regression (and SVMs), but not the kernelized Perceptron

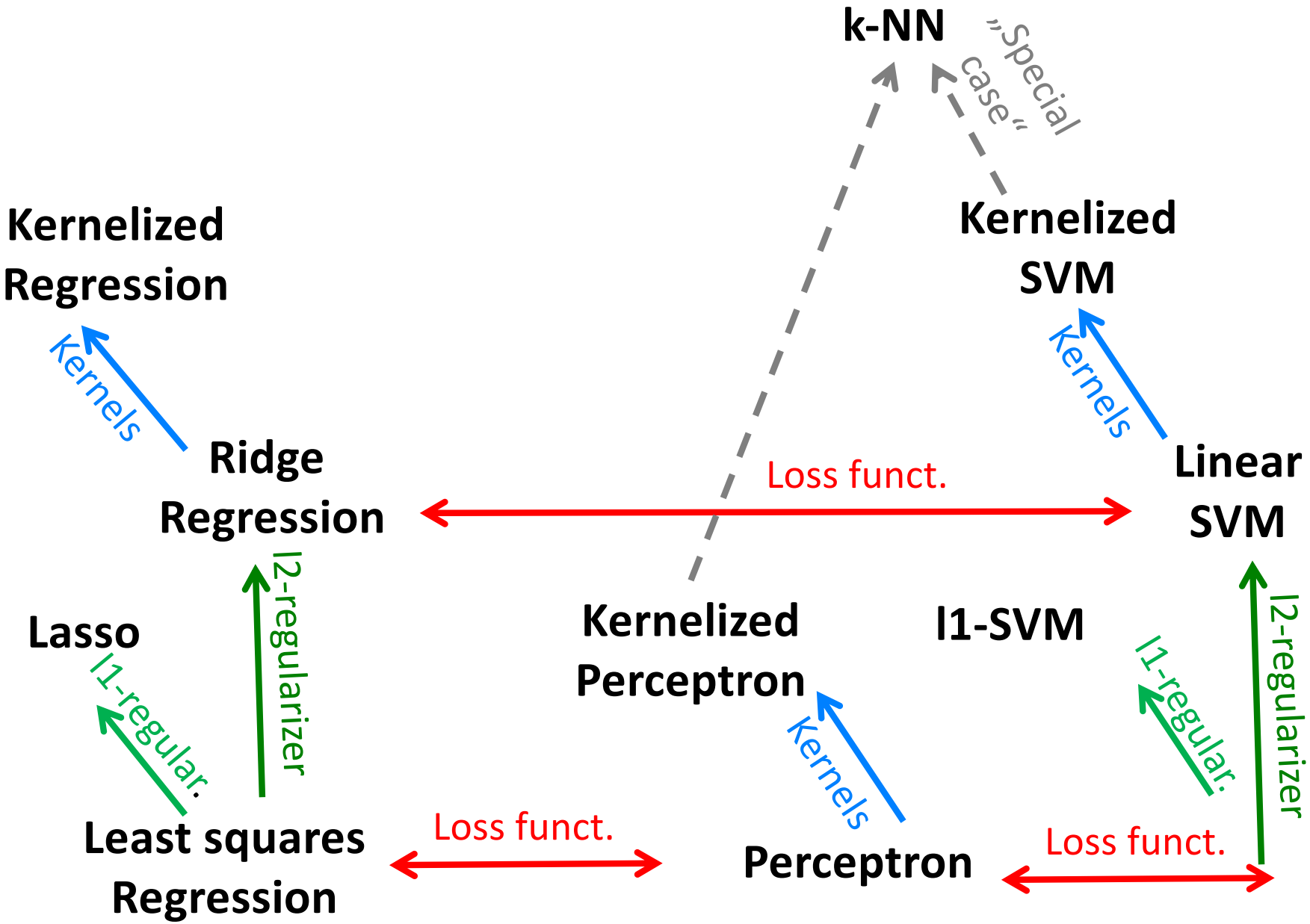
KLR: $$\hat{\alpha} = \arg\min_{\alpha} \frac{1}{n} ||\alpha^T \mathbf{K} - \mathbf{y}||_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

SVM: $$\hat{\alpha} = \arg\min_{\alpha} \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \alpha^T \mathbf{k}_i\} + \lambda \alpha^T \mathbf{D_y K D_y} \alpha$$

What you need to know

- Kernels are
  - (efficient, implicit) inner products
  - Positive (semi-)definite functions
  - Many examples (linear, polynomial, Gaussian/RBF, ...)
- The „Kernel trick“
  - Reformulate learning algorithm so that inner products appear
  - Replace inner products by kernels
- K-Nearest Neighbor classifier (and relation to Perceptron)
- How to choose kernels (kernel engineering etc.)
- **Applications**: Kernelized Perceptron / SVM; kernelized linear regression

Supervised learning big picture so far



Supervised learning summary so far

Representation/ features	Linear hypotheses; nonlinear hypotheses with nonlinear feature transforms; kernels		
Model/ objective:	Loss-function Squared loss, 0/1 loss, Perceptron loss, Hinge loss	+	Regularization L^2 norm, L^1 norm
Method:	Exact solution, Gradient Descent, (mini-batch) SGD, Convex Programming, …		
Evaluation metric:	Mean squared error, Accuracy		
Model selection:	K-fold Cross-Validation, Monte Carlo CV		