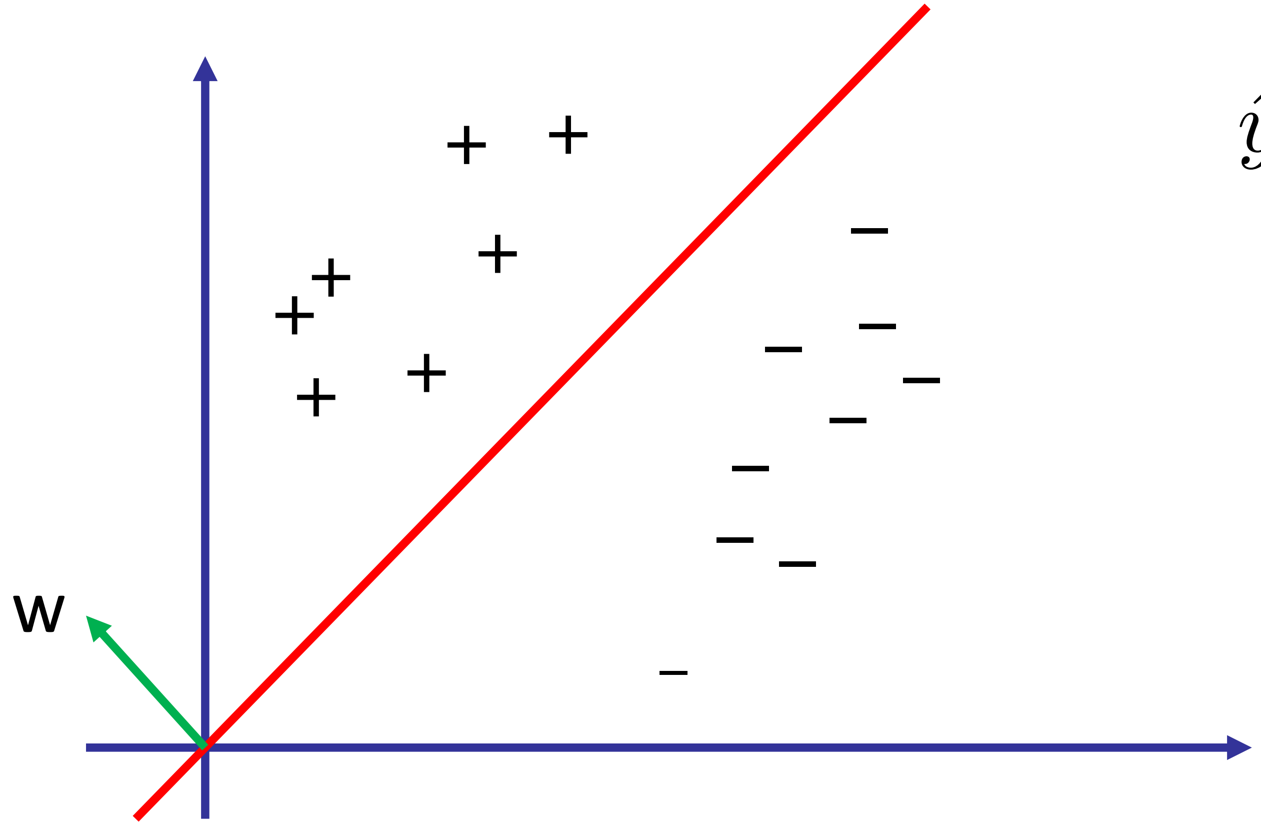
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Introduction to Machine Learning

Non-linear prediction with kernels

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

Recall: Linear classifiers



$$\hat{y} = \text{sign}(\mathbf{w}^T \mathbf{x})$$

Recall: The Perceptron problem

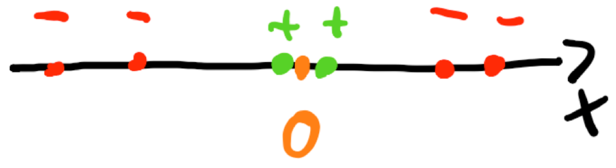
- Solve
$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \ell_P(\mathbf{w}; \mathbf{x}_i, y_i)$$
  where
$$\ell_P(\mathbf{w}; y_i, \mathbf{x}_i) = \max(0, -y_i \mathbf{w}^T \mathbf{x}_i)$$

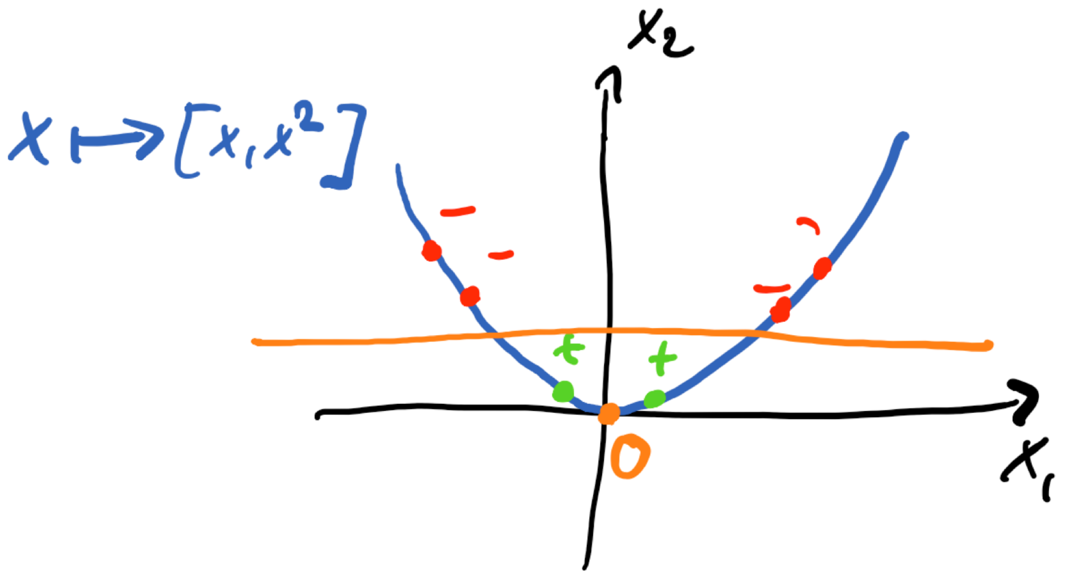
- Optimize via Stochastic Gradient Descent

Solving non-linear classification tasks

- How can we find nonlinear classification boundaries?
- Similar as in regression, can use non-linear transformations of the feature vectors, followed by linear classification

$x_1 = x, \ x_2 = x^2$





Recall: linear regression for polynomials

- We can fit non-linear functions via linear regression, using nonlinear features of our data (basis functions)

$$f(\mathbf{x}) = \sum_{i=1}^{d} w_i \phi_i(\mathbf{x})$$

- For example: polynomials (in 1-D)

$$f(x) = \sum_{i=0}^{m} w_i x^i$$

Polynomials in higher dimensions

- Suppose we wish to use polynomial features, but our input is higher-dimensional
- Can still use monomial features
- **Example**: Monomials in 2 variables, degree = 2

Avoiding the feature explosion

- Need O(d^k) dimensions to represent (multivariate) polynomials of degree k on d features
- **Example**: *d=10000, k=2* → Need ~*100M* dimensions

- In the following, we can see how we can efficiently implicitly operate in such high-dimensional feature spaces (i.e., without ever explicitly computing the transformation)

Revisiting the Perceptron/SVM

- **Fundamental insight**: Optimal hyperplane lies in the span of the data

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

- **(Handwavy) proof**: (Stochastic) gradient descent starting from 0 constructs such a representation

Perceptron: $\mathbf{w}_{t+1} = \mathbf{w}_t + \eta_t y_t \mathbf{x}_t \; [y_t \mathbf{w}_t^T \mathbf{x}_t < 0]$

SVM: $\mathbf{w}_{t+1} = \mathbf{w}_t (1 - 2\lambda\eta_t) + \eta_t y_t \mathbf{x}_t \; [y_t \mathbf{w}_t^T \mathbf{x}_t < 1]$

- **More abstract proof**: Follows from the „representer theorem" (not discussed here)

Reformulating the Perceptron

Advantage of reformulation

$$\hat{\alpha} = \arg\min_{\alpha_{1:n}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, -\sum_{j=1}^{n} \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j\}$$

- **Key observation**: Objective only depends on inner products of pairs of data points
- Thus, we can implicitly work in high-dimensional spaces, as long as we can do inner products efficiently

$$\mathbf{x} \mapsto \phi(\mathbf{x})$$

$$\mathbf{x}^T\mathbf{x}' \mapsto \phi(\mathbf{x})^T\phi(\mathbf{x}') =: k(\mathbf{x}, \mathbf{x}')$$

„Kernels = *efficient* inner products“

- Often, $k(\mathbf{x}, \mathbf{x}')$ can be computed much more efficiently than $\phi(\mathbf{x})^T \phi(\mathbf{x}')$
- Simple example: Polynomial kernel in degree 2

Polynomial kernels (degree 2)

- Suppose $\mathbf{x} = [x_1, \ldots, x_d]^T$ and $\mathbf{x}' = [x'_1, \ldots, x'_d]^T$
- Then $(\mathbf{x}^T \mathbf{x}')^2 = \left(\sum_{i=1}^{d} x_i x'_i \right)^2$

Polynomial kernels: Fixed degree

- The kernel $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}')^m$
  implicitly represents all monomials of degree *m*

- How can we get monomials up to order *m*?

Polynomial kernels

- The polynomial kernel $k(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^T \mathbf{x}')^m$ implicitly represents all monomials of up to degree *m*

- Representing the monomials (and computing inner product explicitly) is *exponential* in *m*!!

The „Kernel Trick“

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

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

- This „trick“ is very widely applicable!

The „Kernel Trick“

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

- Example: Perceptron

The „Kernel Trick“

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

- Example: Perceptron

$$\hat{\alpha} = \arg\min_{\alpha_{1:n}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, -\sum_{j=1}^{n} \alpha_j y_i y_j (\mathbf{x}_i^T \mathbf{x}_j)\}$$

$$\Downarrow$$

$$\hat{\alpha} = \arg\min_{\alpha_{1:n}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, -\sum_{j=1}^{n} \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)\}$$

- Will see further examples later

Derivation: Kernelized Perceptron

Kernelized Perceptron

Training

- Initialize $\alpha_1 = \cdots = \alpha_n = 0$
- For *t=1,2,...*
  - Pick data point (***x****i,* ,*y*i) uniformly at random
  - Predict
    $$\hat{y} = \text{sign}\Big(\sum_{j=1}^{n} \alpha_j y_j k(\mathbf{x}_j, \mathbf{x}_i)\Big)$$
  - If $\hat{y} \neq y_i$ set $\alpha_i \leftarrow \alpha_i + \eta_t$

Prediction

- For new point x, predict
  $$\hat{y} = \text{sign}\Big(\sum_{j=1}^{n} \alpha_j y_j k(\mathbf{x}_j, \mathbf{x})\Big)$$

Demo: Kernelized Perceptron

Questions

- What are valid kernels?
- How can we select a good kernel for our problem?
- Can we use kernels beyond the perceptron?
- Kernels work in very high-dimensional spaces. Doesn‘t this lead to overfitting?

Properties of kernel functions

- Data space *X*
- A kernel is a function $k : X \times X \to \mathbb{R}$
- Can we use any function?

- *k* must be an inner product in a suitable space

→ *k* must be symmetric!

→ Are there other properties that it must satisfy?

Positive semi-definite matrices

Symmetric matrix $M \in \mathbb{R}^{n \times n}$ is positive semidefinite iff

Kernels ➔ semi-definite matrices

- Data space X (possibly infinite)
- Kernel function $k : X \times X \to \mathbb{R}$
- Take any finite subset of data $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subseteq X$
- Then the kernel (gram) matrix

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

is positive semidefinite

Semi-definite matrices → kernels

- Suppose the data space *X={1,...,n}* is finite, and we are given a positive semidefinite matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$
- Then we can always construct a feature map

$$\phi : X \to \mathbb{R}^n$$

such that $\mathbf{K}_{i,j} = \phi(i)^T \phi(j)$

Outlook: Mercer‘s Theorem

Let *X* be a compact subset of $\mathbb{R}^n$ and $k : X \times X \to \mathbb{R}^n$ a kernel function

Then one can expand *k* in a uniformly convergent series of bounded functions ϕ_i s.t.

$$k(x, x') = \sum_{i=1}^{\infty} \lambda_i \, \phi_i(x)\phi_i(x')$$

Can be generalized even further

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): $k(\mathbf{x},\mathbf{x}') = \exp(-||\mathbf{x}-\mathbf{x}'||_2^2/h^2)$
- Laplacian kernel: $k(\mathbf{x},\mathbf{x}') = \exp(-||\mathbf{x}-\mathbf{x}'||_1/h)$

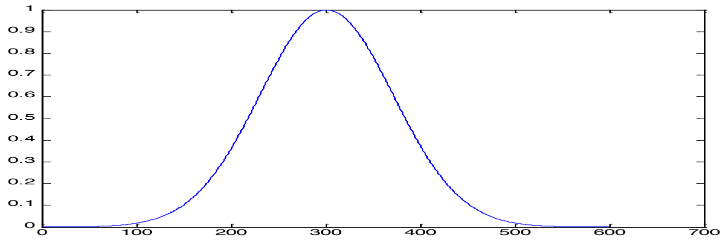
Effect of kernel on function class

- Given kernel k, predictors (for kernelized classification) have the form

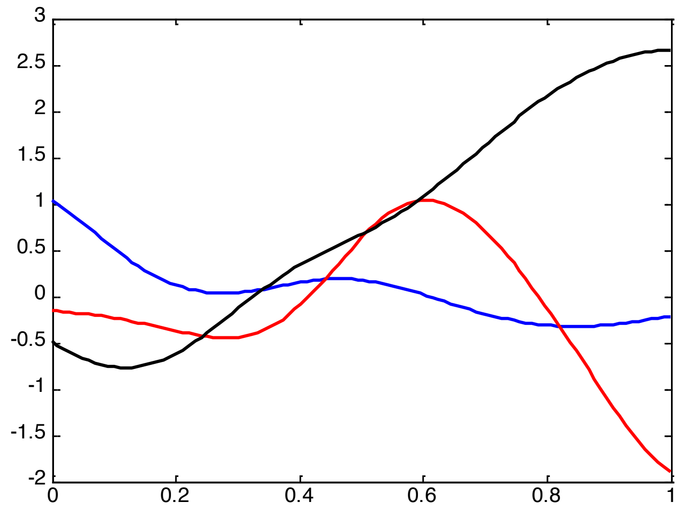
$$\hat{y} = \operatorname{sign}\Big(\sum_{j=1}^{n} \alpha_j y_j k(\mathbf{x}_j, \mathbf{x})\Big)$$

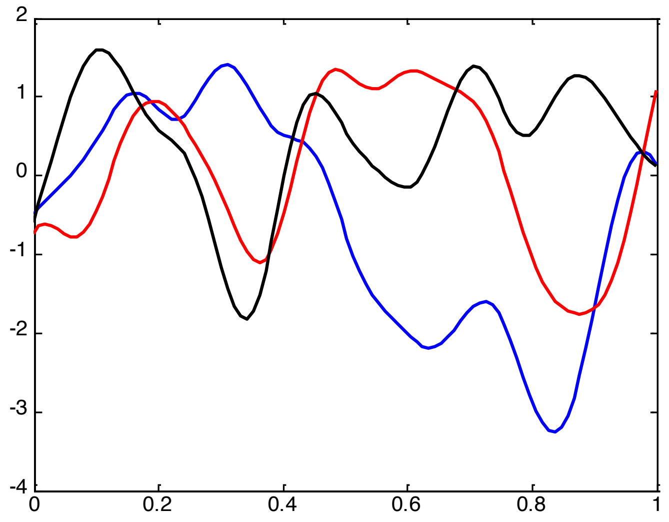
Example: Gaussian kernel

$$k(\mathbf{x}, \mathbf{x}') = \exp(-||\mathbf{x} - \mathbf{x}'||_2^2/h^2)$$



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





Bandwidth h=.3

Bandwidth h=.1

Examples of (non)-kernels

$$k(x, x') = \sin(x)\cos(x')$$

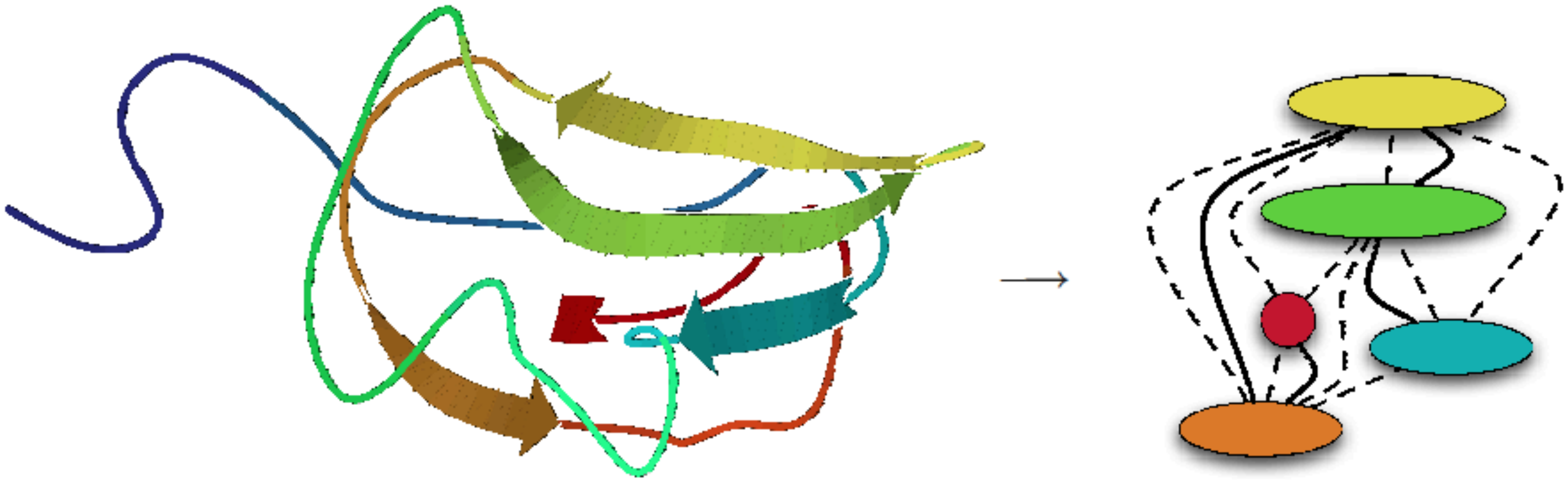
$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T M \mathbf{x}'$$

Kernels beyond $\mathbb{R}^d$

- Can define kernels on a variety of objects:

- Sequence kernels
- Graph kernels
- Diffusion kernels
- Kernels on probability distributions
- ...

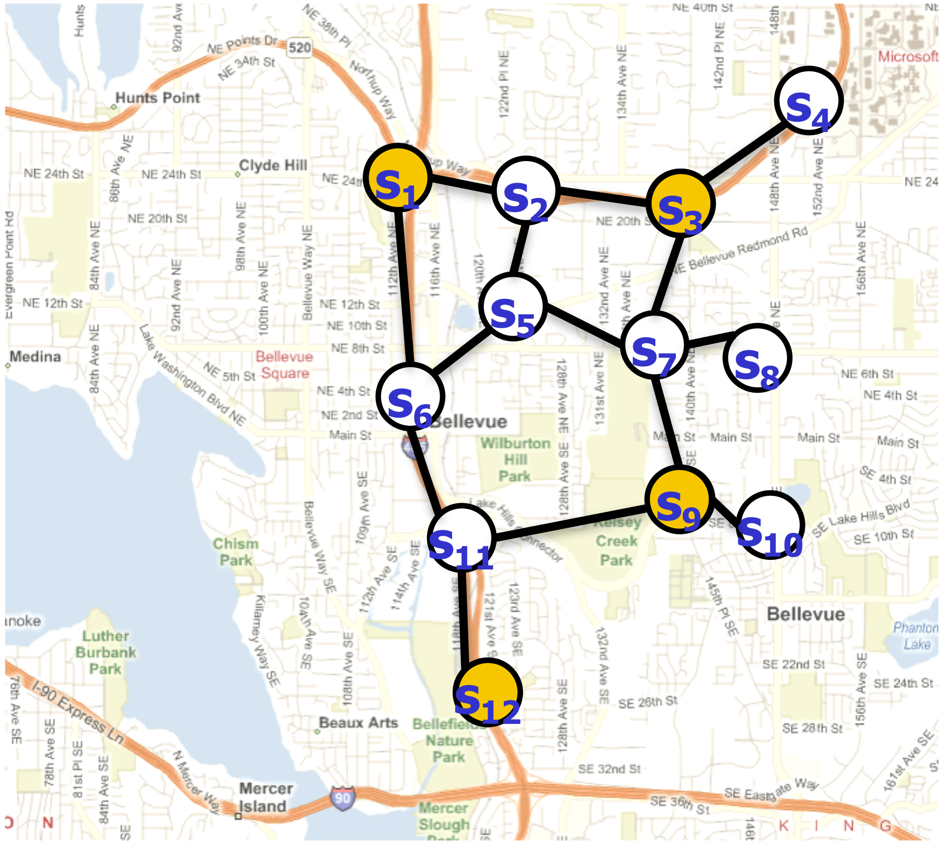
Example: Graph kernels



[Borgwardt et al.]

- Can define a kernel for measuring similarity between graphs by comparing random walks on both graphs (not further defined here)

Example: Diffusion kernels on graphs



$$\mathbf{K} = \exp(-\beta \mathbf{L})$$

- Can measure similarity among nodes in a graph via diffusion kernels (not defined here)

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:

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

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

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

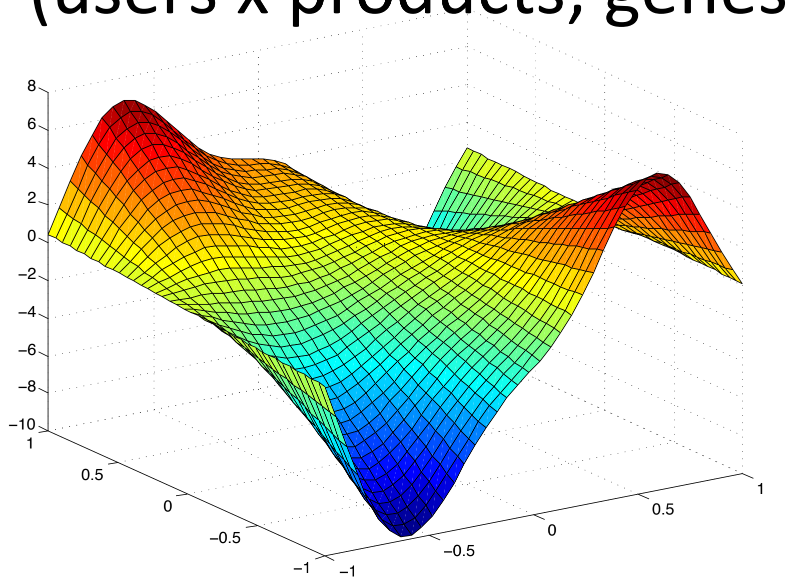
$$k(\mathbf{x}, \mathbf{x}') = f(k_1(\mathbf{x}, \mathbf{x}'))$$

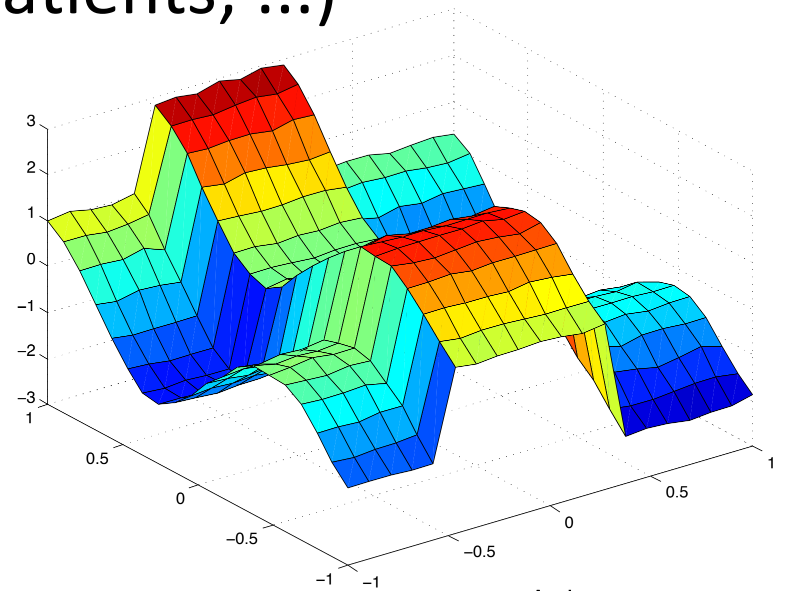
where *f* is a polynomial with positive coefficients or the exponential function

Example: ANOVA kernel

Example: Modeling pairwise data

- May want to use kernels to model pairwise data (users x products; genes x patients; ...)





Where are we?

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

- Next questions:
  - What kind of predictors / decision boundaries do kernel methods entail?
  - Can we use the kernel trick beyond the perceptron?

Kernels as *similarity functions*

- Recall Perceptron (and SVM) classification rule:

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

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

Side note: Nearest-neighbor classifiers

- For data point ***x***, predict majority of labels of *k* nearest neighbors

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

Demo: k-NN

Nearest-neighbor classifiers

- For data point ***x***, predict majority of labels of *k* nearest neighbors

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

- How to choose *k*?
  - Cross-validation! ☺

K-NN vs. Kernel Perceptron

- k-Nearest Neighbor:

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

- Kernel Perceptron:

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

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?

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

How to kernelize the regularizer?

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

Learning & prediction with kernel classifier

- Learning: Solve the problem

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

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

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

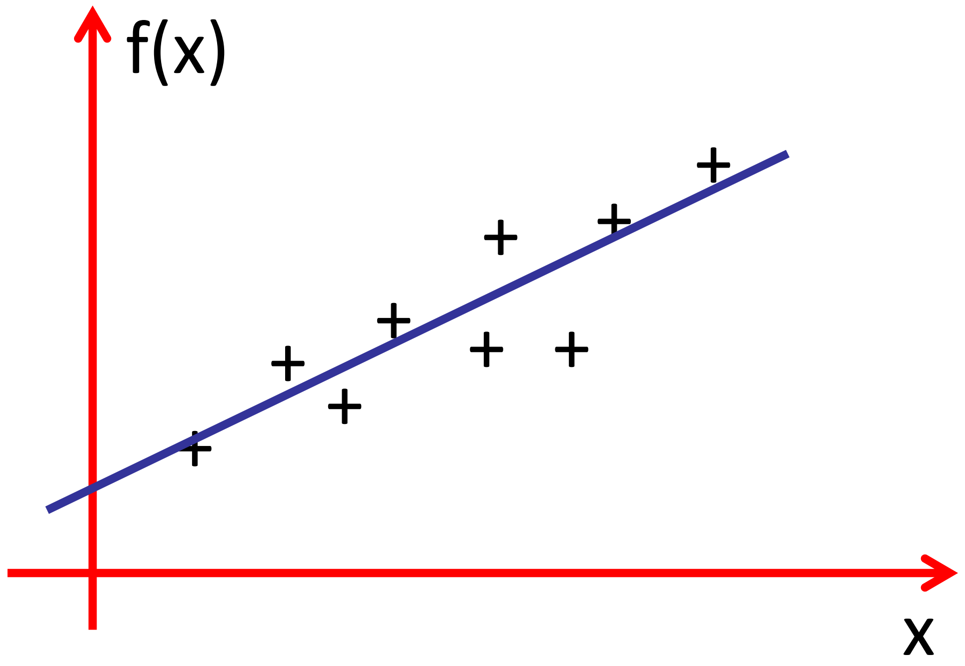
- Prediction: For data point ***x*** predict label *y* as

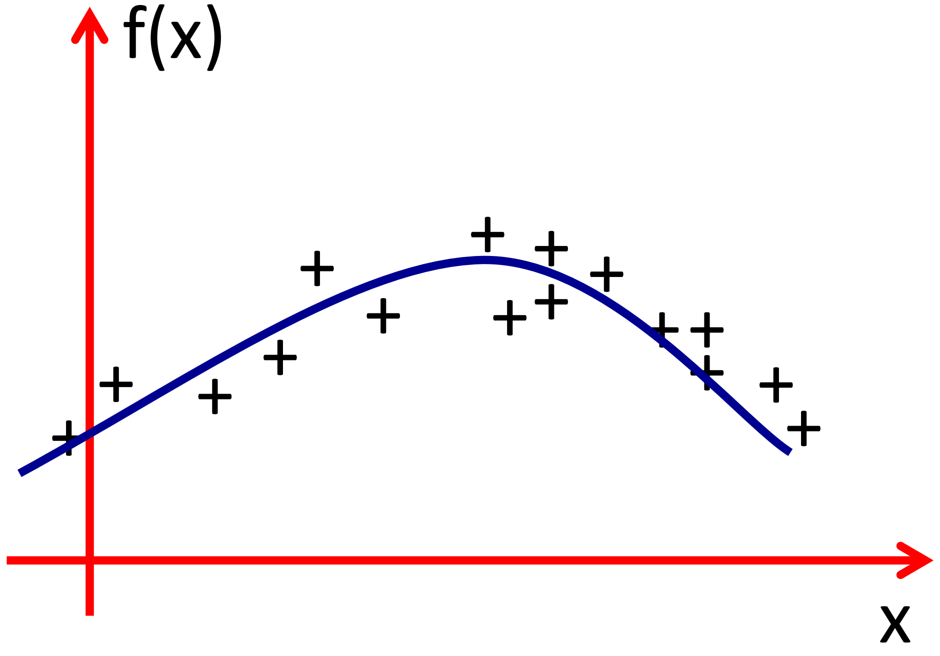
$$\hat{y} = \operatorname{sign}\left(\sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) \right)$$

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:

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

Kernelizing linear regression

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

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

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

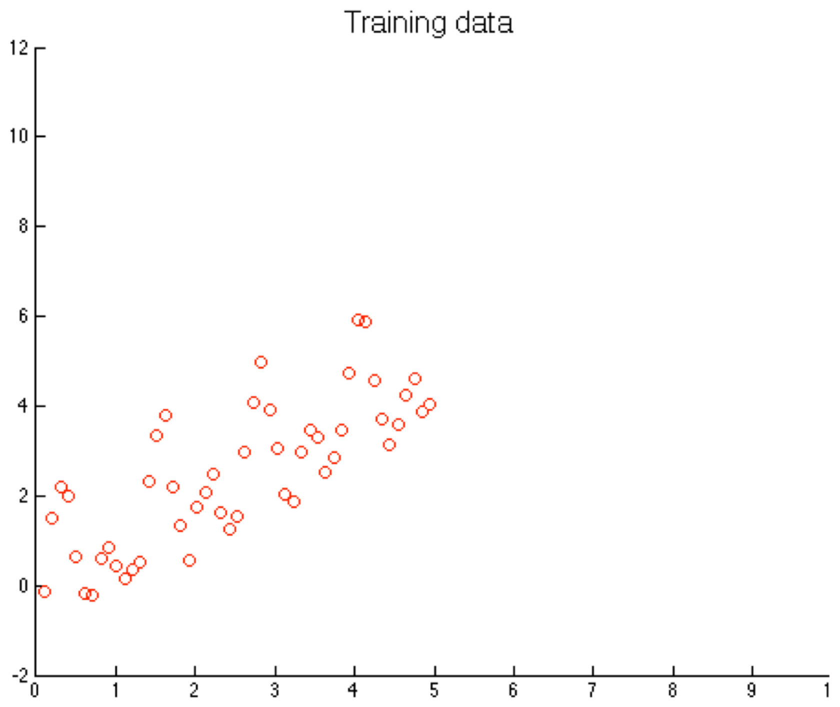
Application: semi-parametric regression

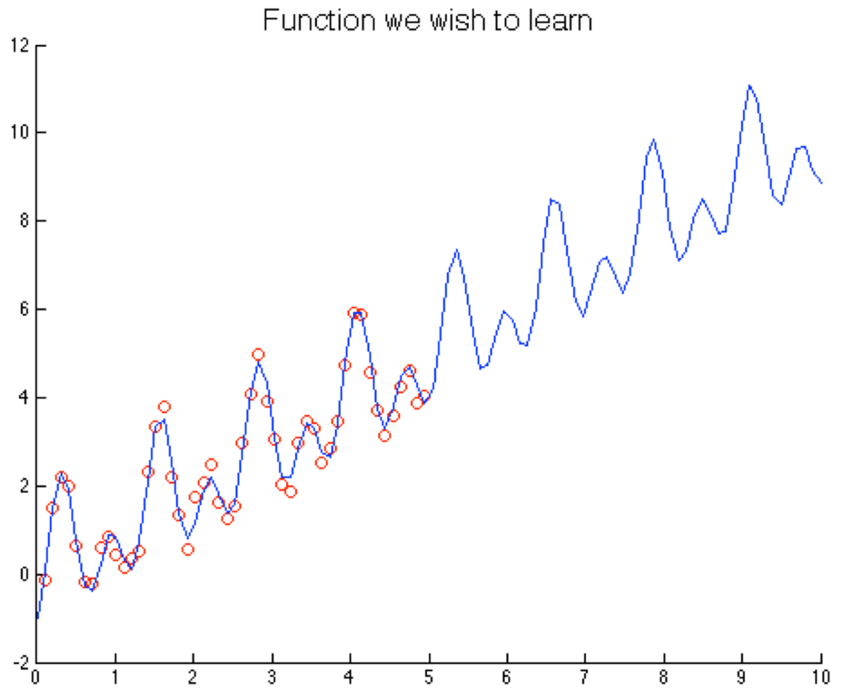
- Often, parametric models are too „rigid“, and non-parametric models fail to extrapolate
- **Solution**: Use additive combination of linear and non-linear kernel function

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

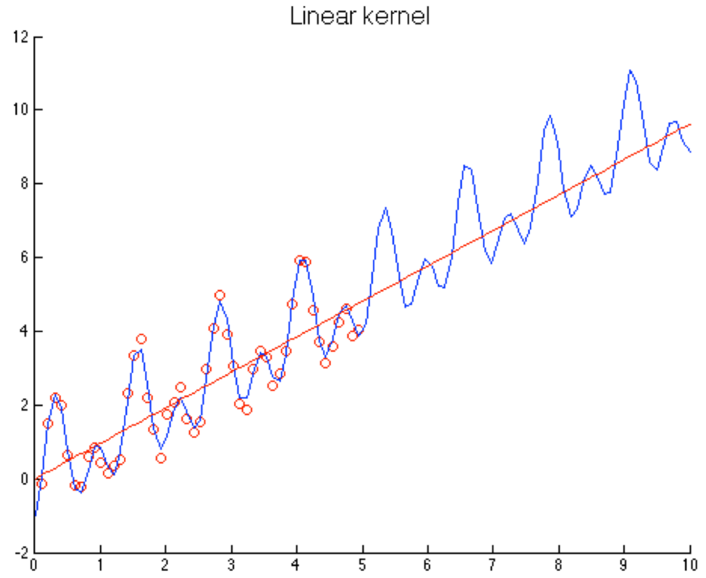
Demo: Semi-parametric KLR

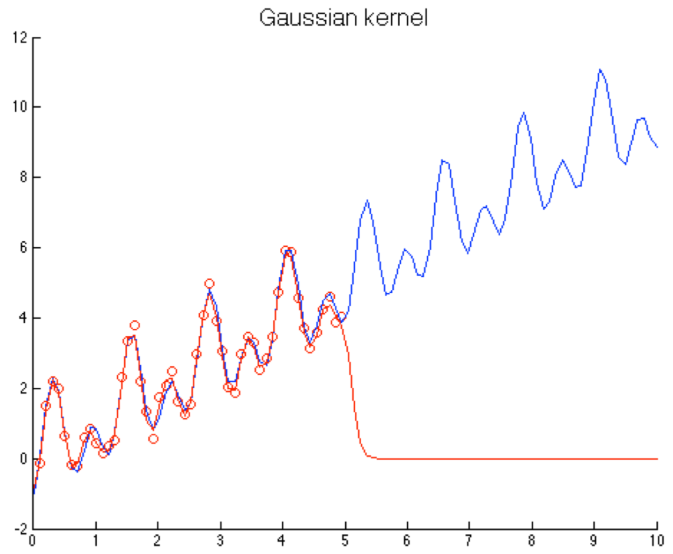
Example

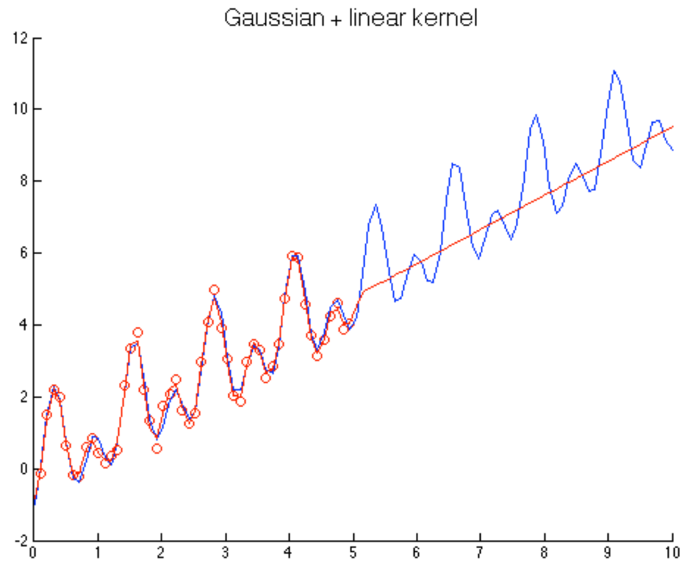

Training data
12
10
8
6
4
2
0
-2
0
1
2
3
4
5
6
7
8
9
1

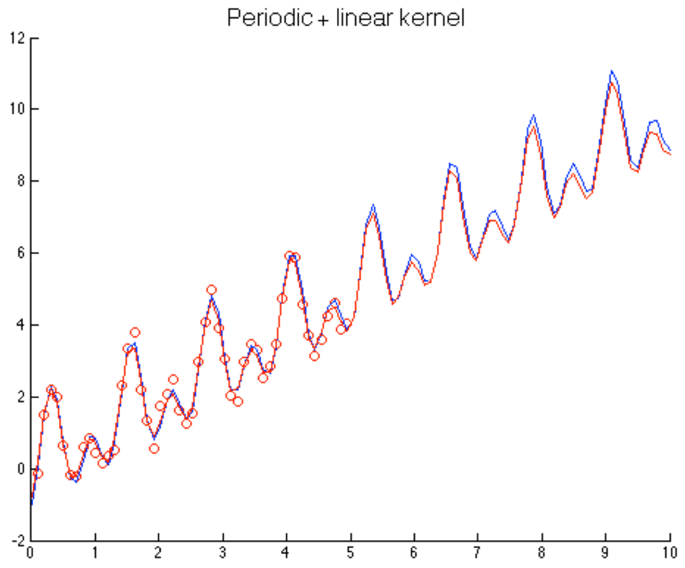

Function we wish to learn
12
10
8
6
4
2
0
-2
0
1
2
3
4
5
6
7
8
9
10

Example fits

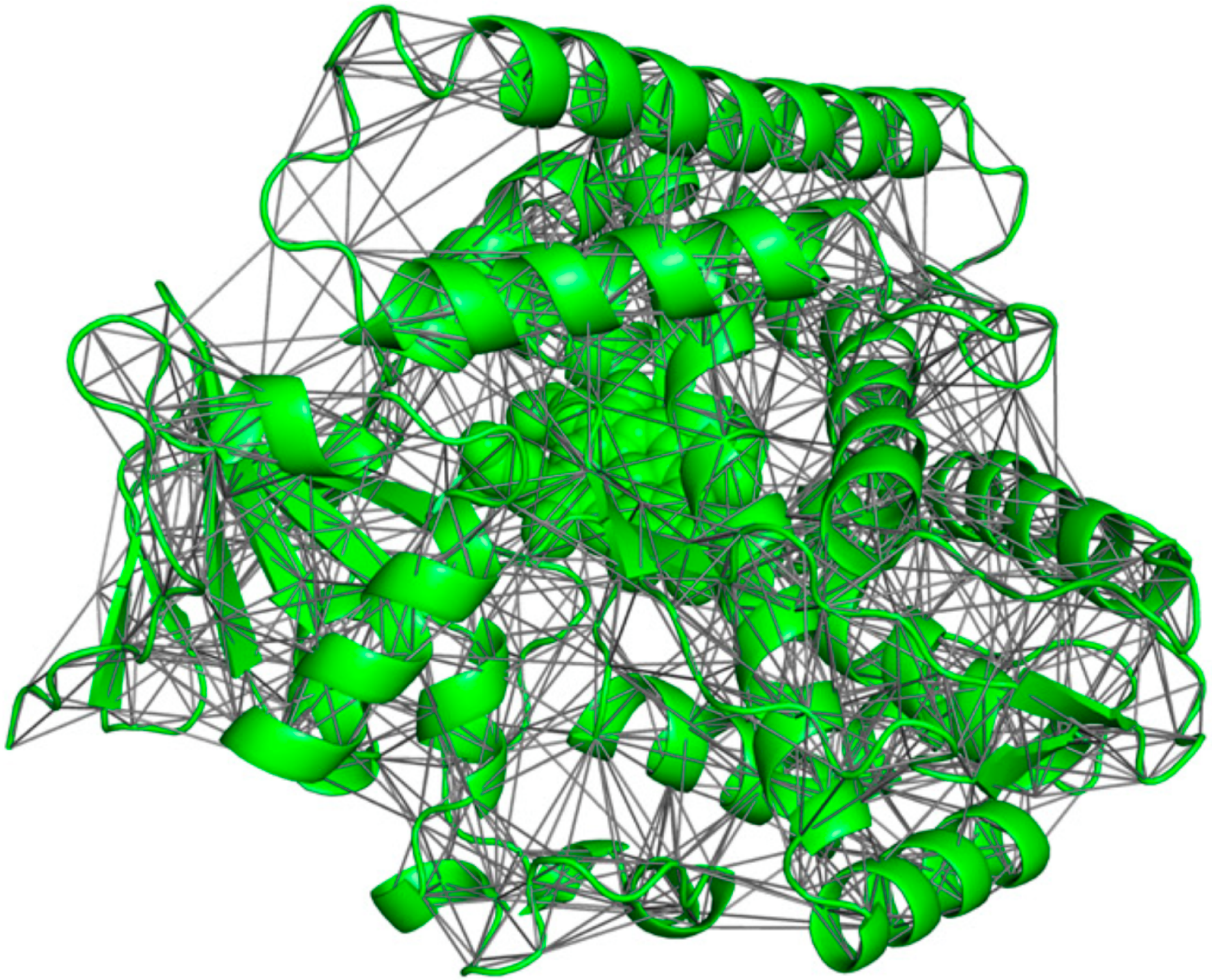




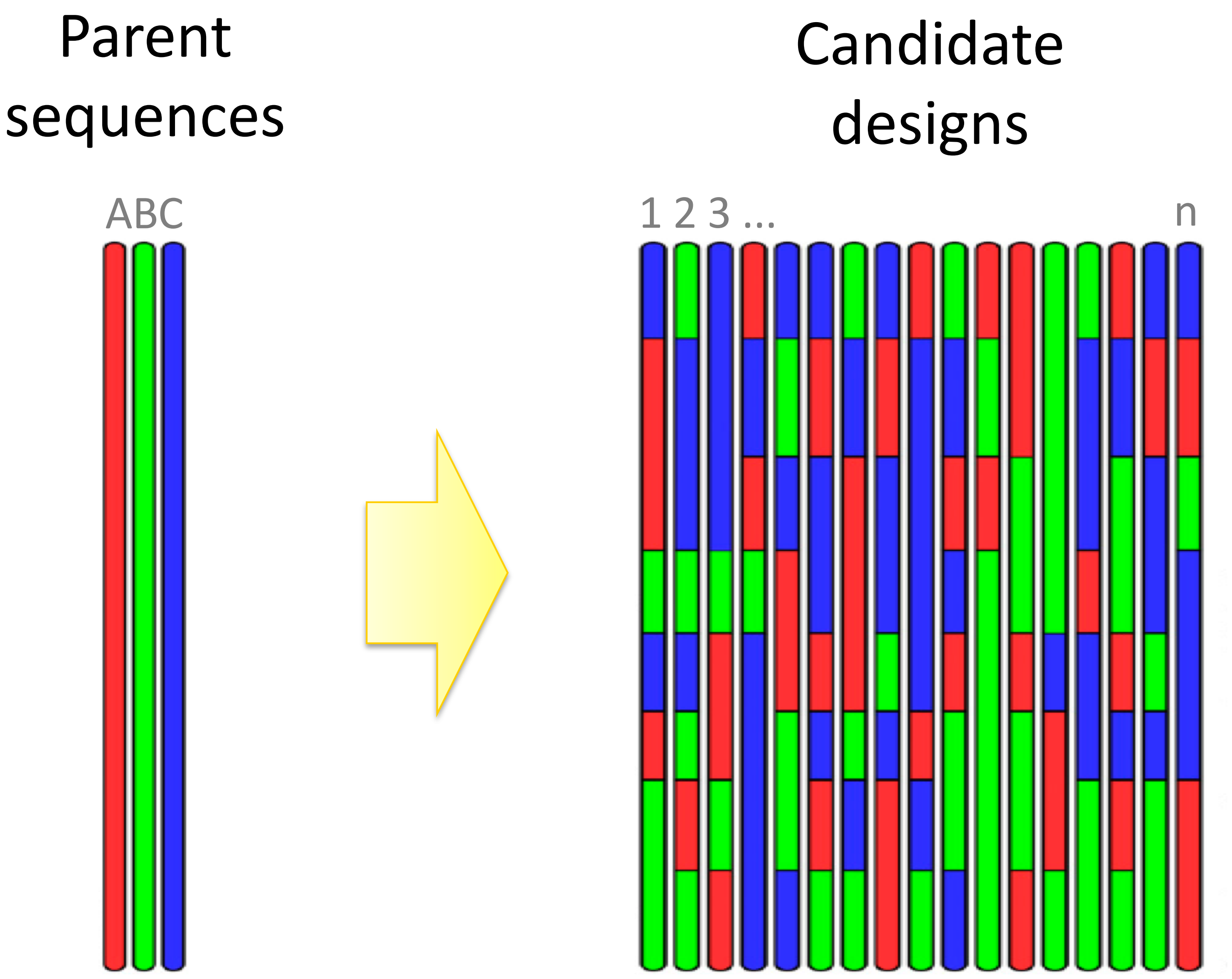




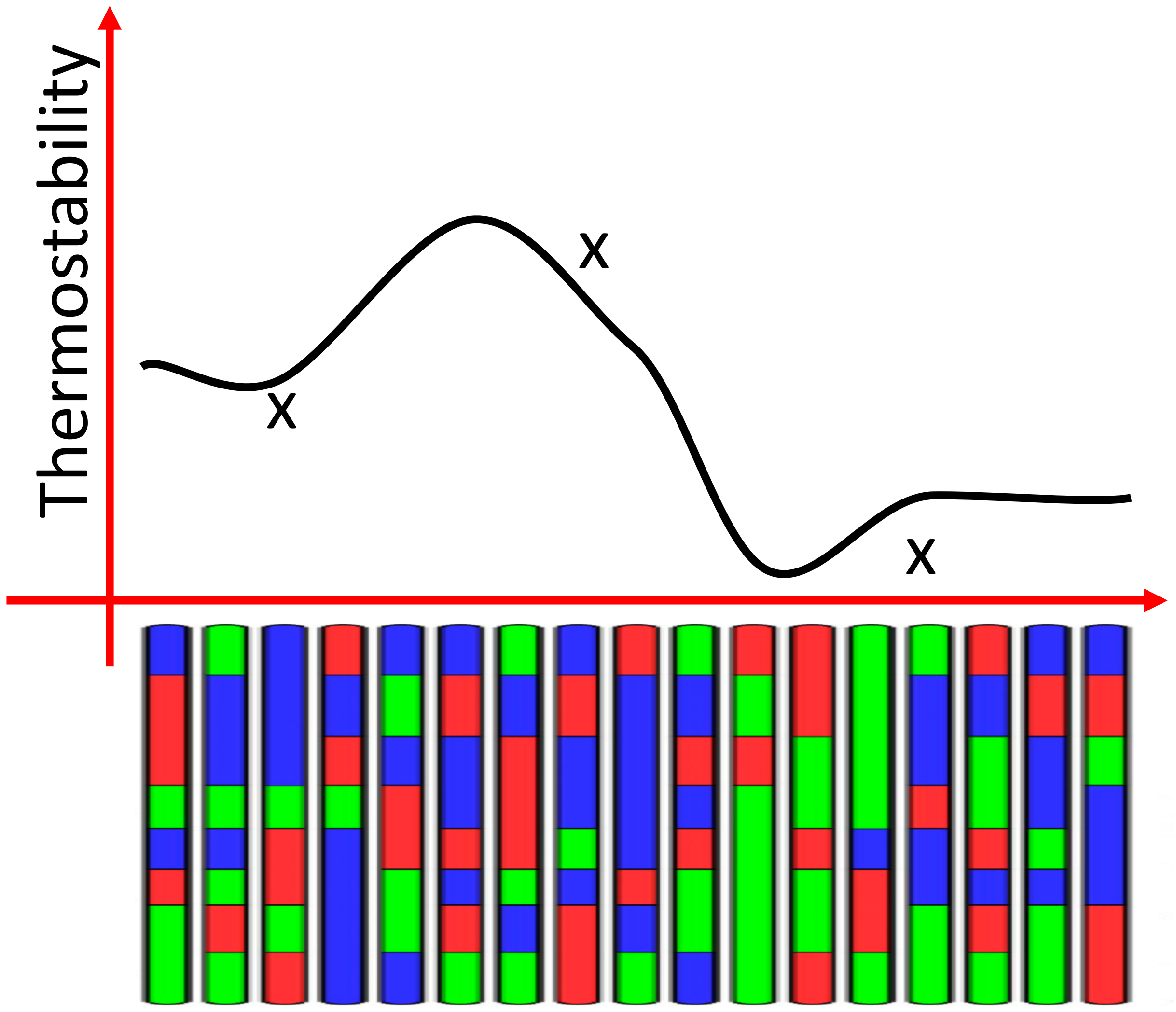
Application: Designing P450s chimeras
[with Phil Romero, Frances Arnold PNAS‘13]



Design space

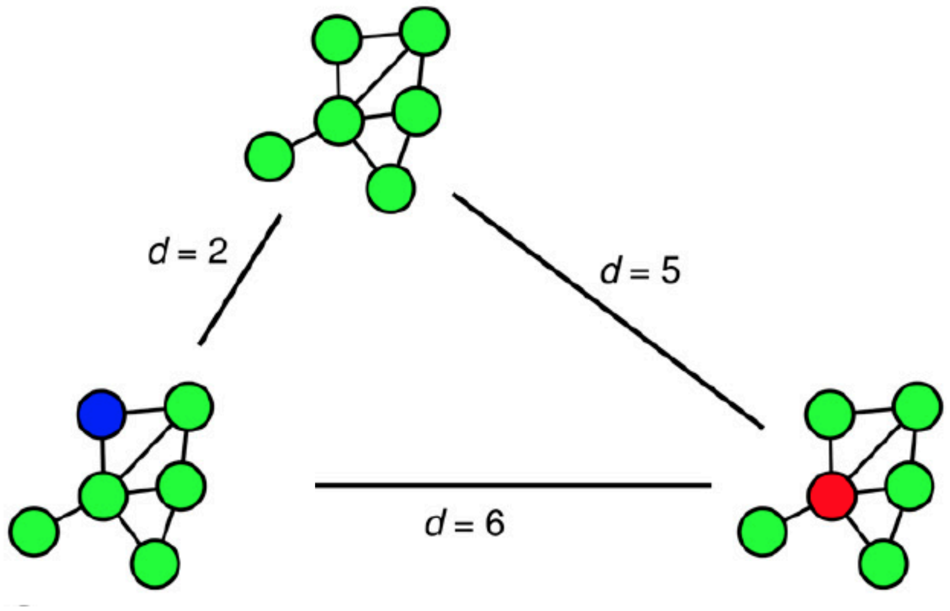


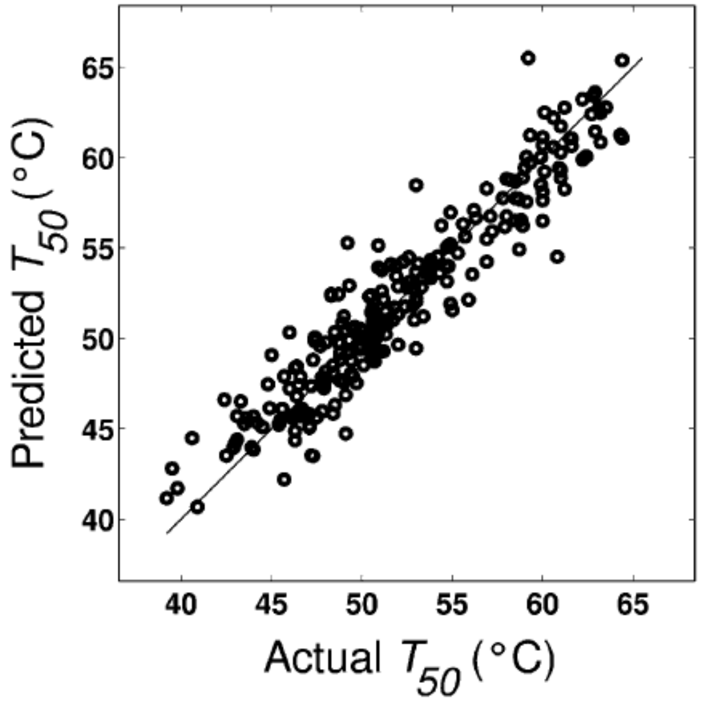
Protein Fitness Landscape



Application: Protein Engineering

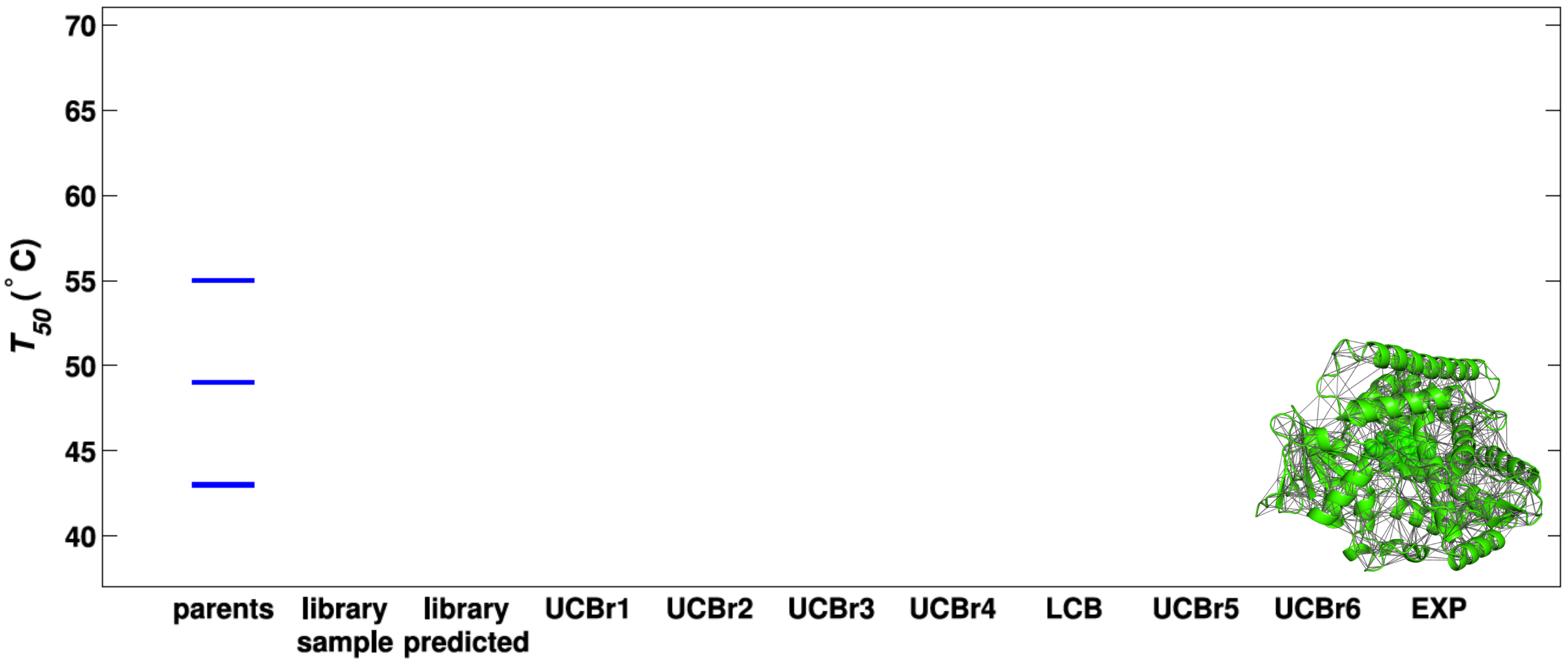
[with Romero, Arnold, PNAS ‘13]





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

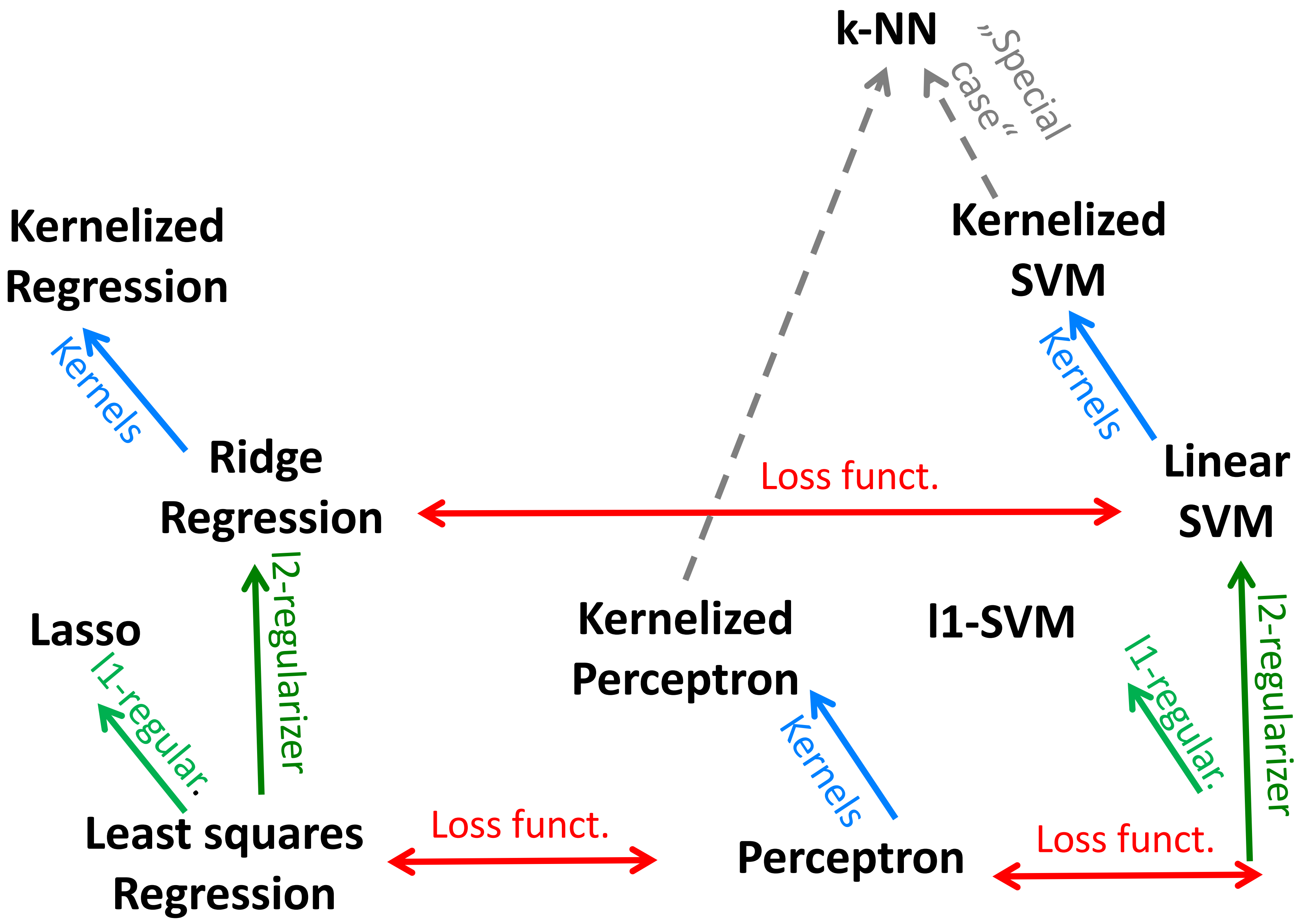
KLR: $$\hat{\alpha} = \arg\min_{\alpha} \frac{1}{n} ||\alpha^T \mathbf{K} - \mathbf{y}||_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

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

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 + Regularization
	Squared loss, 0/1 loss, Perceptron loss, Hinge loss / 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