Kernel

Sewoong Oh

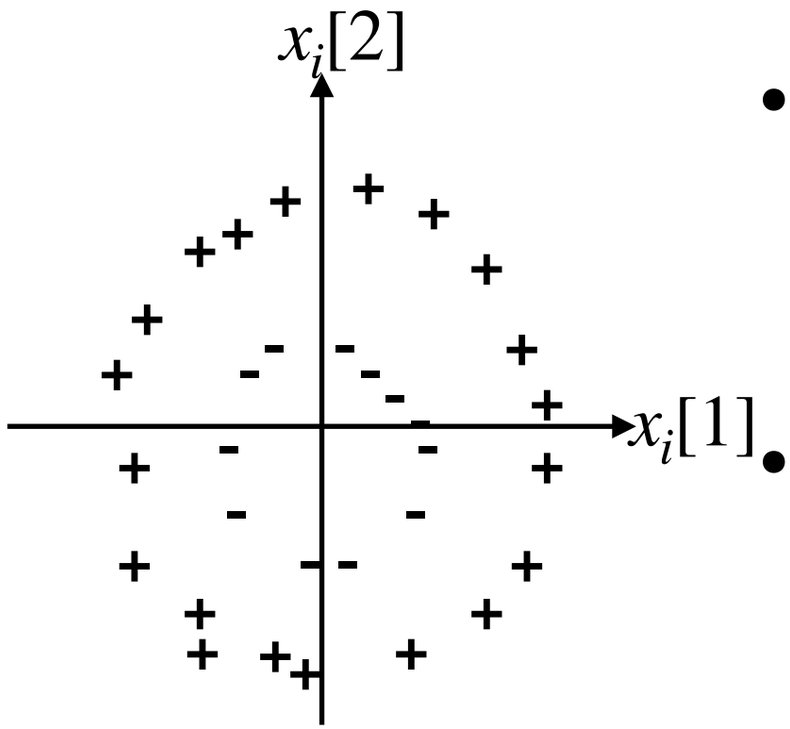
CSE446
University of Washington

Kernel trick:

machine learning for non linearly separable data

Why do we need high-dimensional feature maps?

- consider a classification problem with data $x \in \mathbb{R}^d$ with $d = 2$

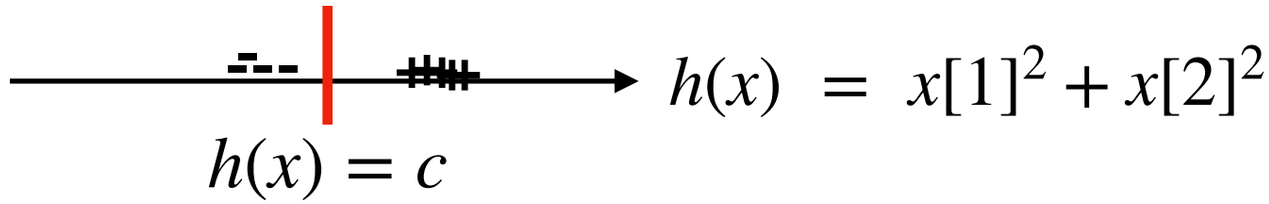


- this is not linearly separable, but a human could engineer a perfect feature map, which

$$h(x) = x[1]^2 + x[2]^2 \in \mathbb{R}^k,$$

with $k = 1$

- the resulting data can be perfectly separated with a linear classifier



- however, it is a priori hard to know what feature map works for the given data
- so the rule of thumb is to use lots of features with very large k, and hope the linear regression/classification algorithm picks the right feature

Feature mapping can be expensive

- recall that when we apply linear regression to model non-linear functions, we used feature maps

$$h : \mathbb{R}^d \to \mathbb{R}^k$$
$$x \mapsto h(x)$$

- examples include
  - sinusoids
  - polynomials
- recall that in linear least squares regression, for example, we want to solve

$$\text{minimize}_{w \in \mathbb{R}^k} \quad \sum_{i=1}^{n} \left(y_i - w^T h(x_i) \right)^2$$

- gradient update rule for gradient descent is

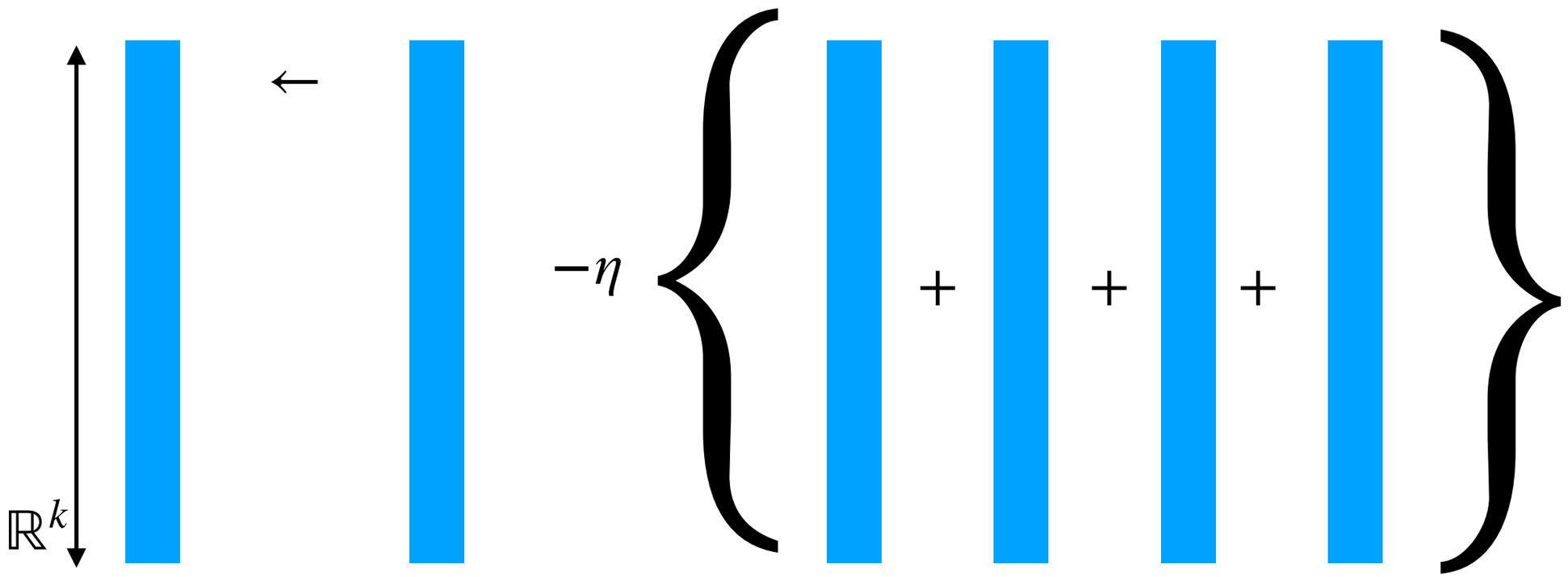
$$w^{(t)} \leftarrow w^{(t-1)} - \eta \sum_{i=1}^{n} \left((w^{(t-1)})^T h(x_i) - y_i \right) h(x_i)$$

- this can be prohibitively high-dimensional, for example $d = 1000$ and cubic functions require $k = 1000^3$
- at a first glance, it seems inevitable to keep k-dimensional memory (for $w^{(t)}$'s) and computation to solve such an optimization

Kernel trick

- however, if the sample size $n \ll k$, then we do not need to track all k-dimensions, as the degree of freedom of the problem is much less

$$w^{(t)} \leftarrow w^{(t-1)} - \eta \sum_{i=1}^{n} \left((w^{(t-1)})^T h(x_i) - y_i \right) h(x_i)$$



- suppose, $w^{(0)} = 0$, then

  $w^{(1)} = \eta \sum_{i=1}^{n} y_i h(x_i)$ is a linear combination of n vectors $\{h(x_1), \cdots, h(x_n)\}$

- so we can compactly write is as $w^{(1)} = \mathbf{H}^T \alpha^{(1)} \in \mathbb{R}^k$, where $\mathbf{H}^T = \begin{bmatrix} h(x_1) & h(x_2) & \cdots & h(x_n) \end{bmatrix} \in \mathbb{R}^{k \times n}$, and $\alpha^{(t)} \in \mathbb{R}^n$

Kernel trick when $k \gg n$

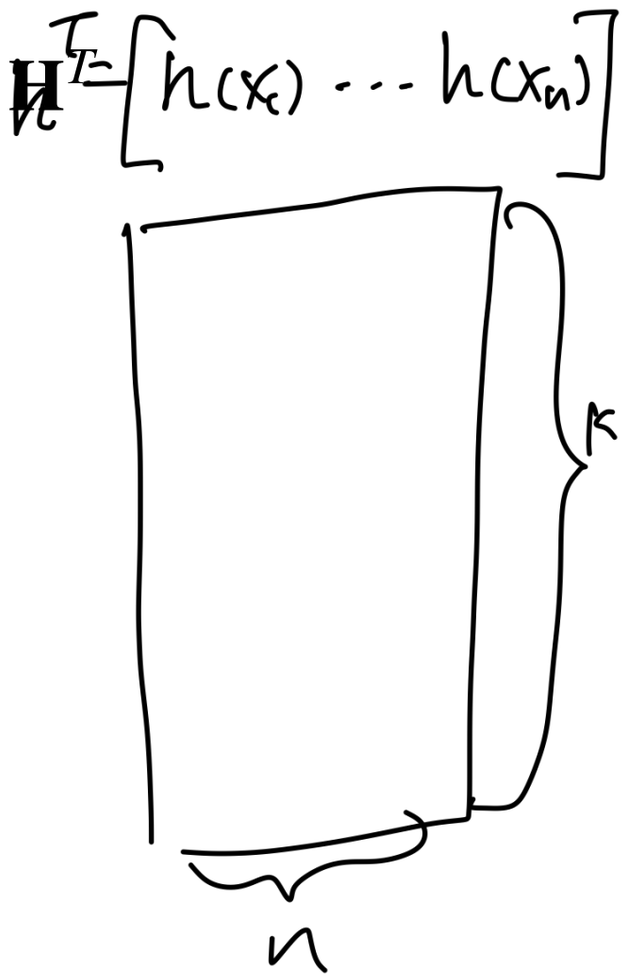
- as the update rule only adds linear combination of the columns of $\mathbf{H}^T$, the entire gradient updates can be replaced from those of $w^{(t)} \in \mathbb{R}^k$ to those of $\alpha^{(t)} \in \mathbb{R}^n$
- suppose $w^{(t-1)}$ is in the span of $\mathbf{H}^T$, i.e. $w^{(t-1)} = \mathbf{H}^T \alpha^{(t-1)}$

$$w^{(t)} = w^{(t-1)} - \eta \sum_{i=1}^{n} \left((w^{(t-1)})^T h(x_i) - y_i \right) h(x_i)$$

$$= \mathbf{H}^T \alpha^{(t-1)} - \eta \mathbf{H}^T \begin{bmatrix} (w^{(t-1)})^T h(x_1) - y_1 \\ (w^{(t-1)})^T h(x_2) - y_2 \\ \vdots \\ (w^{(t-1)})^T h(x_n) - y_n \end{bmatrix}$$

$$= \mathbf{H}^T \left\{ \alpha^{(t-1)} - \eta \begin{bmatrix} (w^{(t-1)})^T h(x_1) - y_1 \\ (w^{(t-1)})^T h(x_2) - y_2 \\ \vdots \\ (w^{(t-1)})^T h(x_n) - y_n \end{bmatrix} \right\}$$

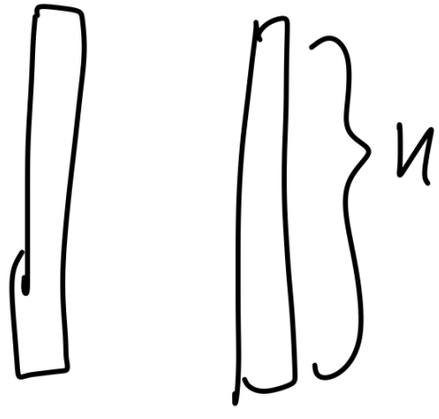
- and hence $w^{(t)}$ is also in the span of $\mathbf{H}^T$



Kernel trick when $k \gg n$

- further, the gradient update can be compactly computed
- by representing $w^{(t)} = \mathbf{H}^T \alpha^{(t)}$

$$\nabla_w \mathcal{L} = h^T \cdot (hh^T \alpha^{(t-1)} - y)$$



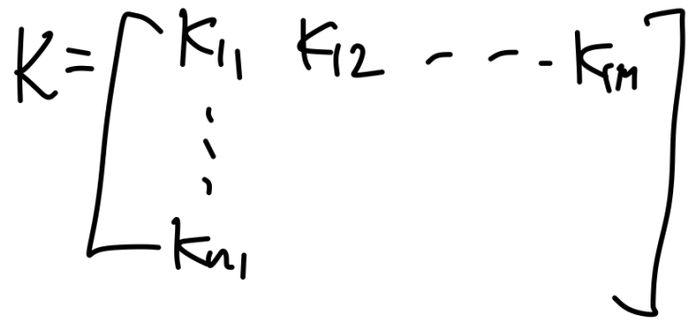
$$\mathbf{H}^T \alpha^{(t)} = \mathbf{H}^T \left\{ \alpha^{(t-1)} - \eta \begin{bmatrix} h(x_1)^T (w^{(t-1)}) - y_1 \\ h(x_2)^T (w^{(t-1)}) - y_2 \\ \vdots \\ h(x_n)^T (w^{(t-1)}) - y_n \end{bmatrix} \right\}$$

$$\underbrace{h(x_n)}_{h} \quad \underbrace{(w^{(t-1)})}_{h^T \alpha^{(t-1)}} \quad \underbrace{- y_n}_{\mathbf{y}}$$

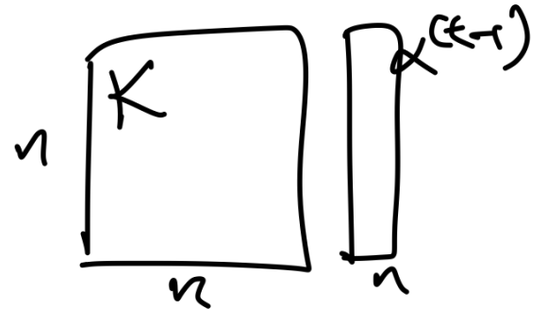
$$= \mathbf{H}^T \left\{ \alpha^{(t-1)} - \eta(\mathbf{H}\mathbf{H}^T \alpha^{(t-1)} - \mathbf{y}) \right\}$$

Kernel trick when $k \gg n$

- the **kernel** with respect to a feature map $h : \mathbb{R}^d \to \mathbb{R}^k$ is defined as
  $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$
  $(x_i, x_j) \mapsto K(x_i, x_j) = h(x_i)^T h(x_j)$

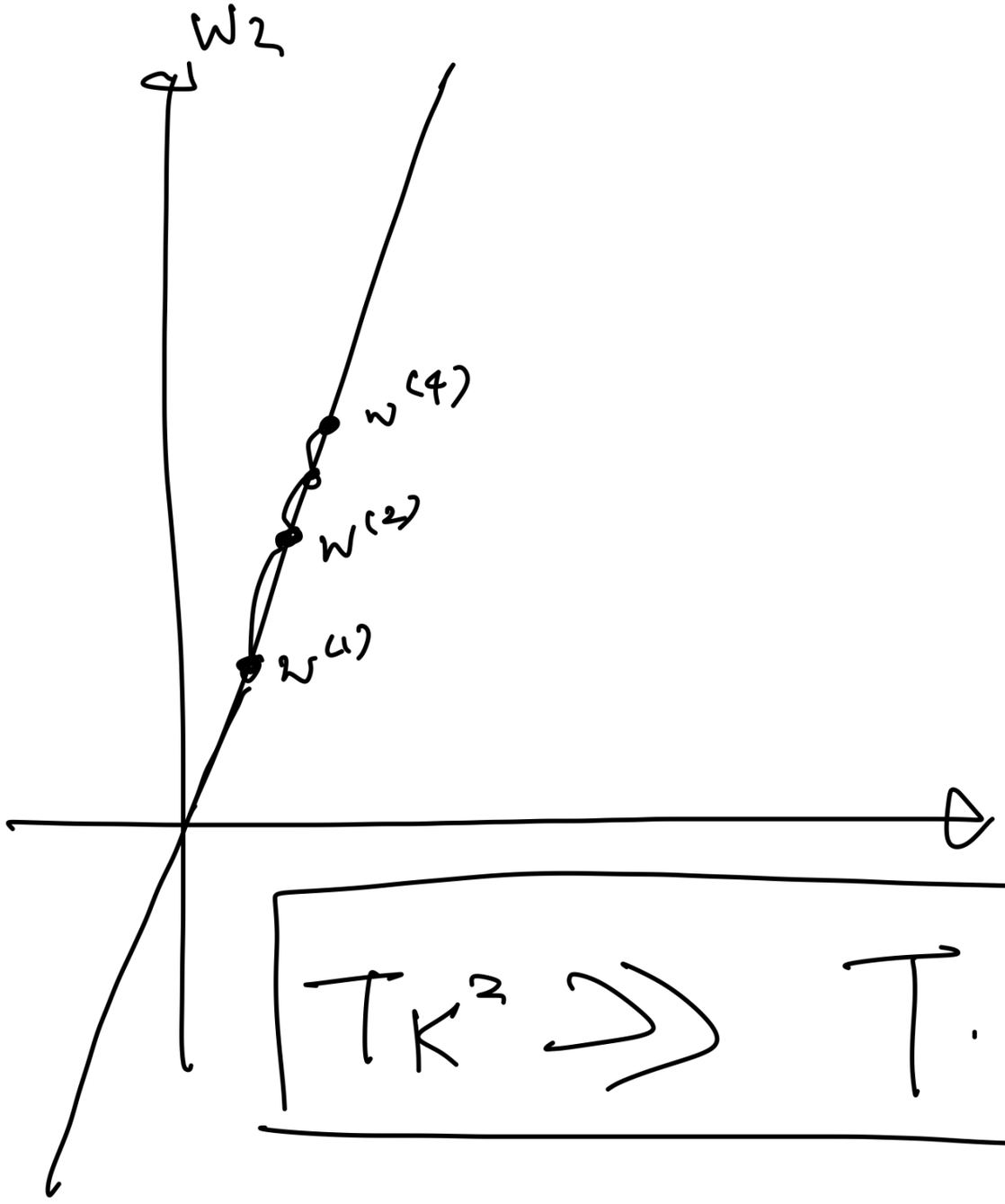


- the **kernel trick** for gradient update can be written as
  - compute the kernel matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$ as $K_{ij} = K(x_i, x_j)$
  - for $t = 1, \ldots, T$
    - $\alpha^{(t)} \leftarrow \alpha^{(t-1)} - \eta(\mathbf{K}\alpha^{(t-1)} - \mathbf{y})$



- this is much more efficient requiring memory of size n and per iteration computational complexity of n^2

- fundamentally, all we need to know about the feature map $h(x_i)'s$ is captured in a much more compact matrix $\mathbf{K}$



$K = 2$

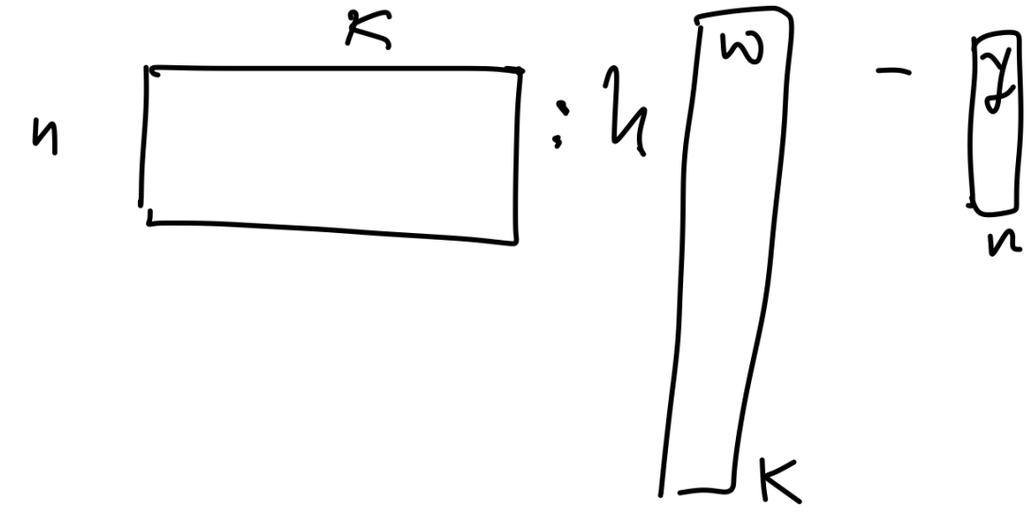
$n = 1$

$\boxed{h^T \cdot \alpha^{(t)} = w^{(t)}}$

$\boxed{T K^2 \gg T \cdot n^2 + O(K)}$

$n \ll K \ll T n^2$

$$L(w) = \|k \cdot w - y\|_2^2 + \lambda \|w\|_2^2$$



$w = k^T \cdot \alpha$, $K = k \cdot k^T \in \mathbb{R}^{n \times n} \longrightarrow K^T = K$

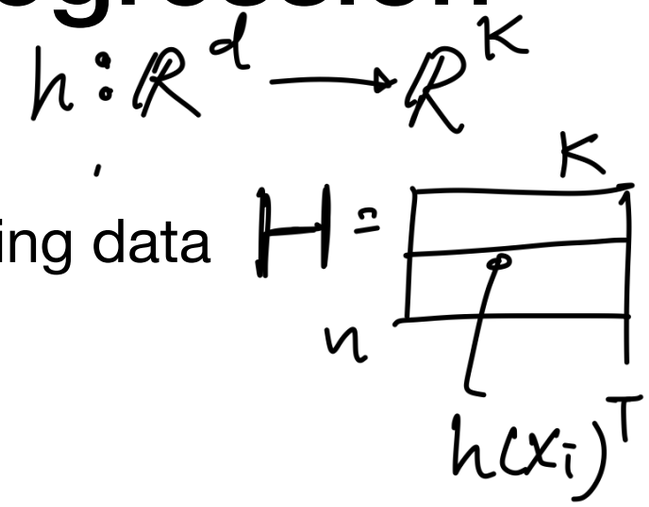
$$L(\alpha) = \|k k^T \alpha - y\|_2^2 + \lambda \|k^T \alpha\|_2^2$$

$$= \|K \cdot \alpha - y\|_2^2 + \lambda \alpha^T K \cdot \alpha$$

$$\nabla_\alpha L(\alpha) = 2K^T(K\alpha - y) + 2\lambda K\alpha = 0$$

$$= 2K(K$$

Closed-form solution to kernel regression



- in practice you first choose a kernel to be used
- and compute the kernel matrix $\mathbf{K} = \mathbf{H}\mathbf{H}^T \in \mathbb{R}^{n \times n}$ for training data
- then the regularized squared loss is

$$\mathscr{L}(w) \;=\; \|\mathbf{H}w - \mathbf{y}\|_2^2 \;+\; \lambda \|w\|_2^2$$

can be written (using $w = \mathbf{H}^T\alpha$) as

$$\begin{aligned}\mathscr{L}(\alpha) \;&=\; \|\mathbf{H}\mathbf{H}^T\alpha - \mathbf{y}\|_2^2 + \lambda \alpha^T \mathbf{H}\mathbf{H}^T\alpha \\ &=\; \|\mathbf{K}\alpha - \mathbf{y}\|_2^2 + \lambda \alpha^T \mathbf{K}\alpha\end{aligned}$$

- as we assume $k \gg n$ and $\mathbf{K}$ is invertible (and note that $\mathbf{K} = \mathbf{K}^T$ by definition), the minimizer is $\hat{\alpha} \;=\; (\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y}$
- this follows from the fact that

$$\nabla_\alpha \mathscr{L}(\alpha) \;=\; 2\mathbf{K}^T(\mathbf{K}\alpha - \mathbf{y}) + 2\lambda \mathbf{K}\alpha$$

- also it follows that

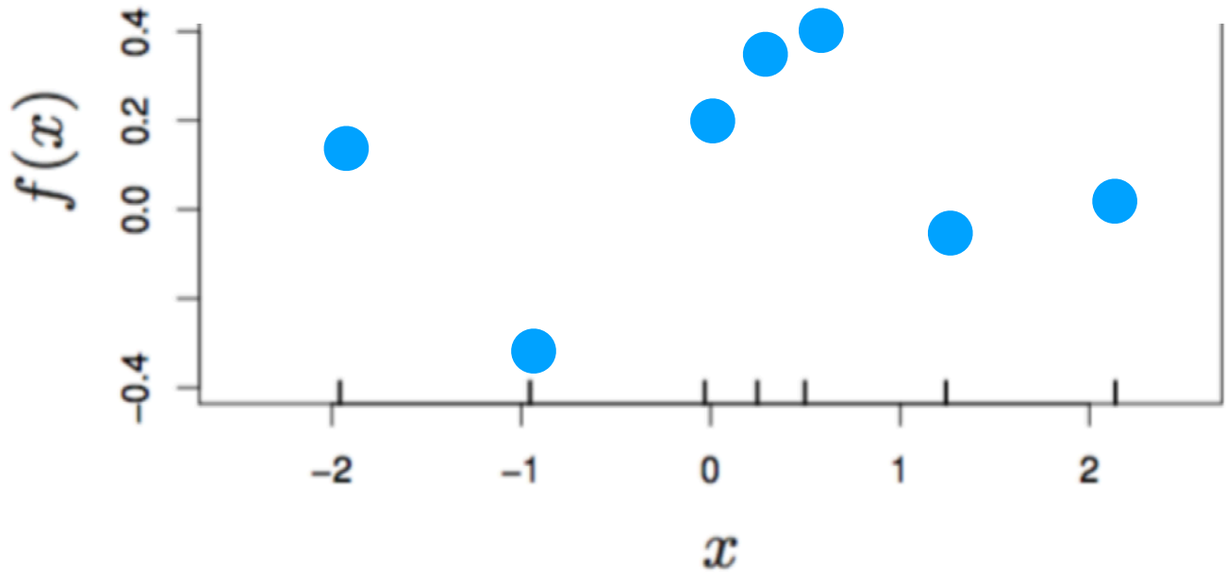
$$\hat{w} \;=\; \mathbf{H}^T\hat{\alpha} \;=\; \mathbf{H}^T(\mathbf{H}\mathbf{H}^T + \lambda \mathbf{I})^{-1}\mathbf{y}$$

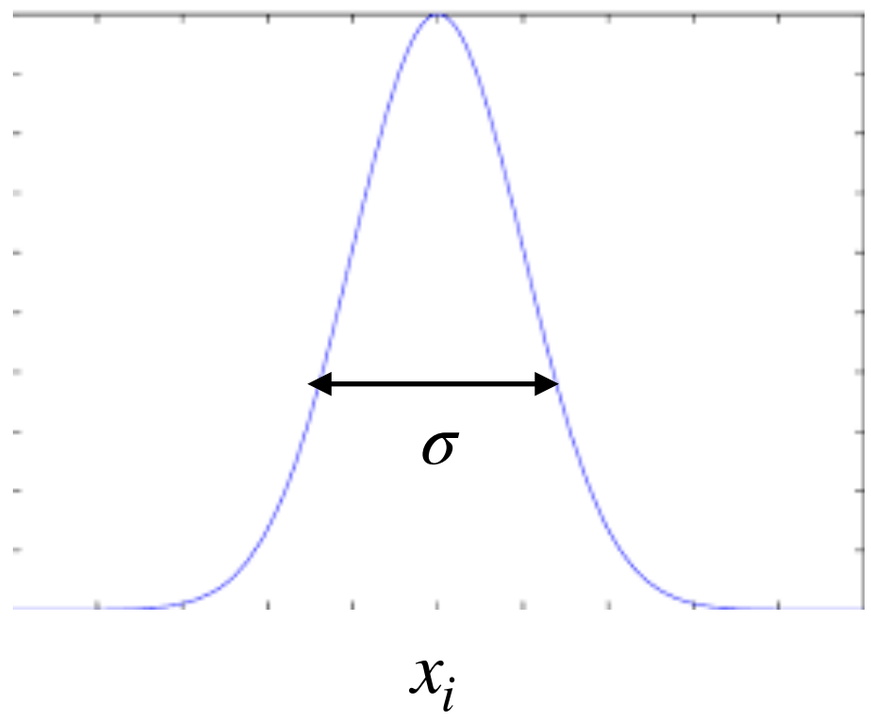
- and the prediction is

$$f(x) \;=\; h(x)^T\hat{w} \;=\; h(x)^T\mathbf{H}^T\hat{\alpha} \;=\; \sum_{i=1}^{n} K(x_i, x)\hat{\alpha}_i$$

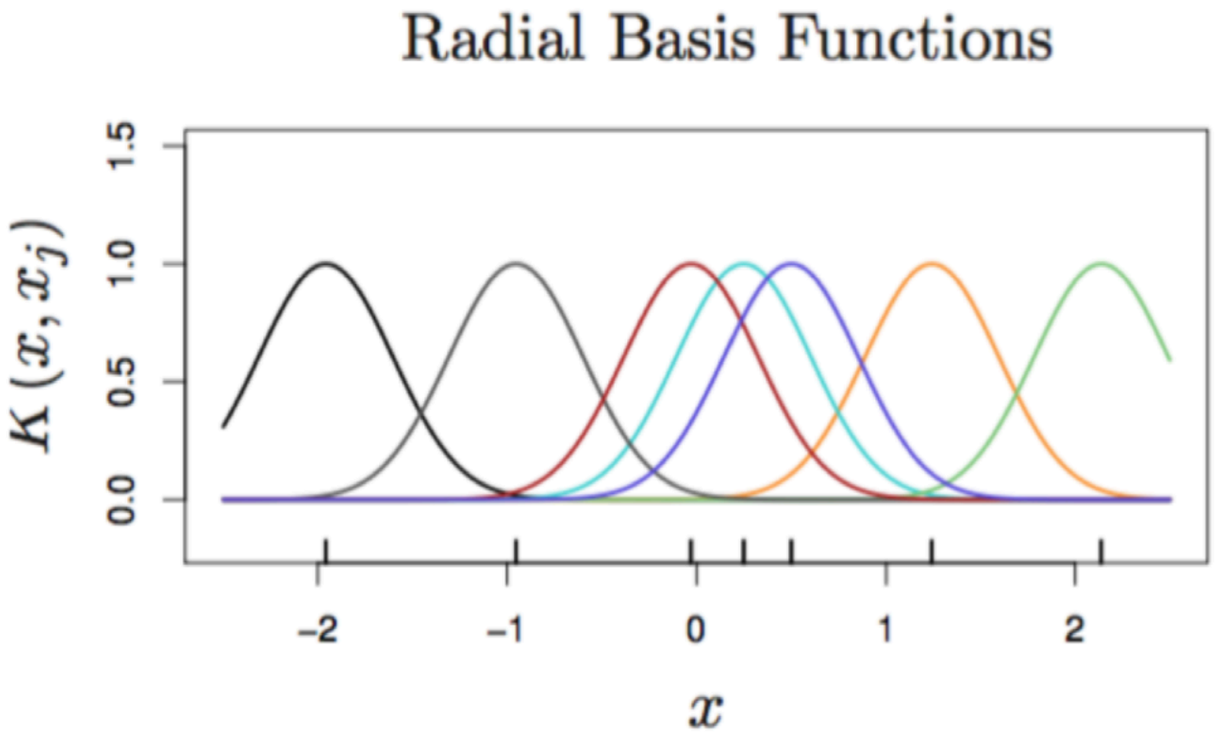
- this is a weighted sum of kernel functions $K(x_i, \cdot\,)$ "centered" at x_i's, weighted by the learned parameter $\hat{\alpha}_i$'s

RBF kernel $k(x_i, x) = \exp\left\{ -\frac{\|x_i - x\|_2^2}{2\sigma^2} \right\}$

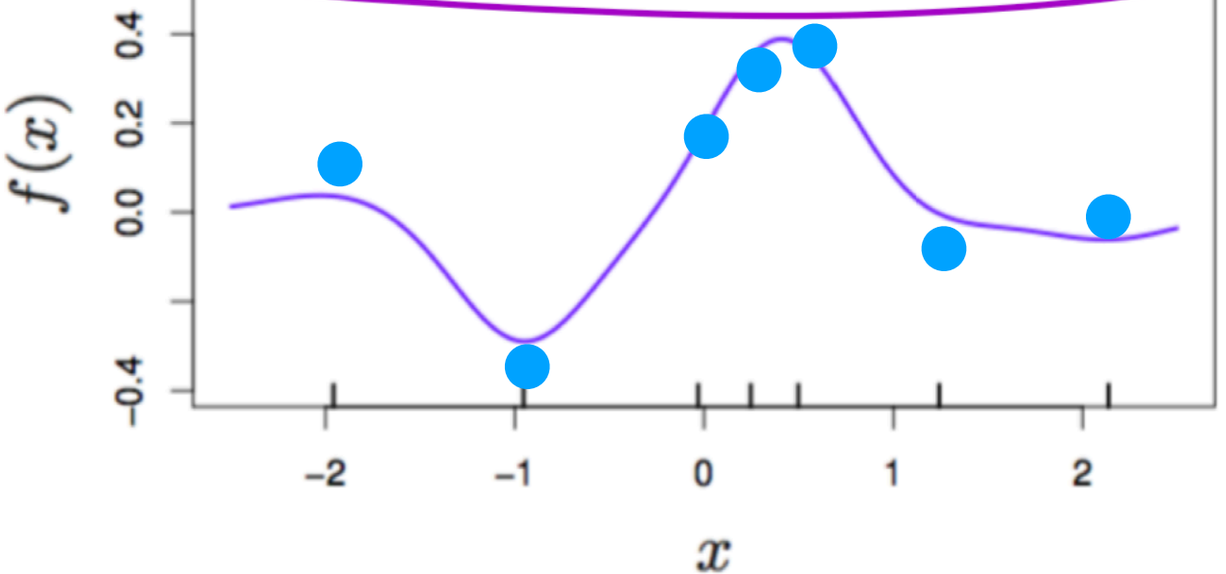




- predictor is taking weighted sum of n kernel functions centered at each sample points

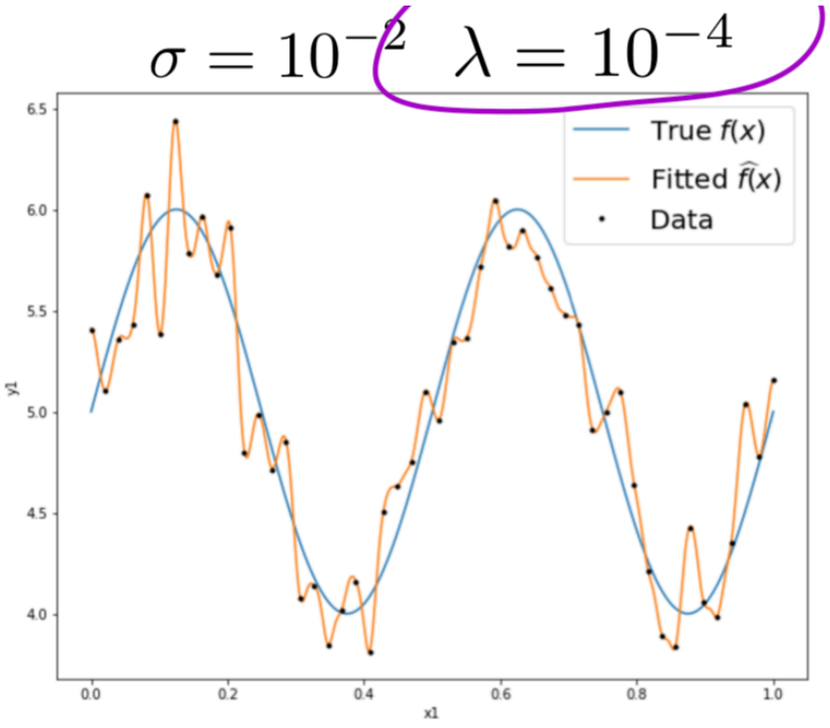


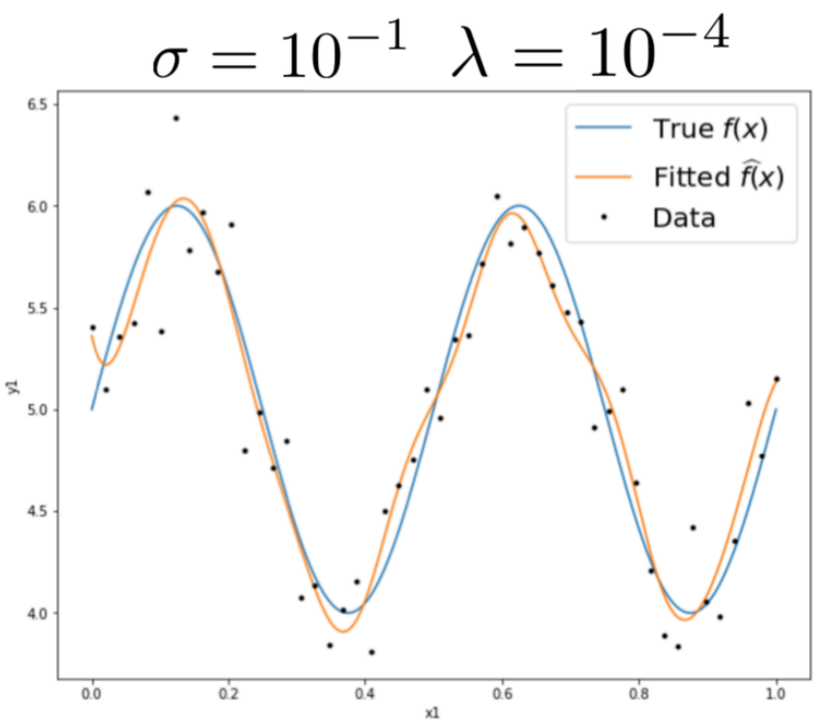
$f(x) = \alpha_0 + \sum_j \alpha_j K(x, x_j)$

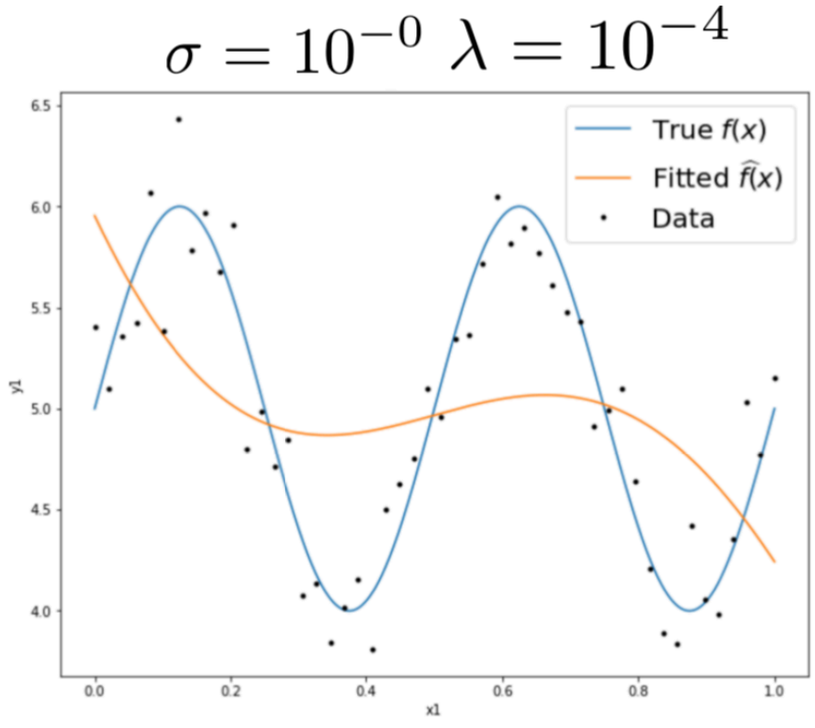


RBF kernel $k(x_i, x) = \exp\left\{ -\frac{\|x_i - x\|_2^2}{2\sigma^2} \right\}$

- $\mathscr{L}(w) = \|\mathbf{H}w - \mathbf{y}\|_2^2 + \lambda\|w\|_2^2$
- The bandwidth σ^2 of the kernel regularizes the predictor







From kernels to feature maps

- recall that selecting the right feature map $h(\cdot)$ is important for the model to be accurate,
- now that (potentially challenging) task of feature engineering can be replaced by selecting the kernel

$$K(x_i, x_j) = h(x_i)^T h(x_j)$$

- in particular, we do not even need to write down the feature map $h(\cdot)$, we only need to ensure existence, i.e. make sure that the kernel $K(\cdot, \cdot)$ we use is derived from **some** feature map

- but first, let's look at some concrete examples
  - linear kernel $K(x_i, x_j) = x_i^T x_j$ corresponds to $h(x_i) = x_i$
  - affine kernel $K(x_i, x_j) = x_i^T x_j + 1$ corresponds to $h(x_i) = \begin{bmatrix} 1 \\ x_i \end{bmatrix}$

From kernels to feature maps

- kernel $K(x_i, x_j) = (x_i^T x_j)^2$

$$= \left(\sum_{i'=1}^{d} x_i[i'] x_j[i'] \right)^2$$

$$= \sum_{i',i''=1}^{d} \left(x_i[i'] x_i[i''] \right) \left(x_j[i'] x_j[i''] \right)$$

feature map is $h(x_i) = \begin{bmatrix} x_i[1]x_i[1] \\ x_i[1]x_i[2] \\ \vdots \\ x_i[d]x_i[d] \end{bmatrix}$, which is the second order polynomial features

- similarly, kernel $K(x_i, x_j) = (x_i^T x_j + 1)^2$

$$= \sum_{i',i''} (x_i[i] x_i[i'']) (x_j[i'] x_j[i'']) + \sum_{i'=1}^{d} \sqrt{2} x_i[i'] x_j[i'] + 1$$

feature map is all monomials up to degree two

From kernels to feature maps

- in general $K(x_i, x_j) = (x_i^T x_j + 1)^p$ corresponds to polynomial feature map of degree p

- Gaussian kernel is

$$K(x_i, x_j) = \exp\left\{\frac{-\|x_i - x_j\|_2^2}{2\sigma^2}\right\}$$

  which is a common measure of similarity between two points
- finding the corresponding feature map is a homework problem

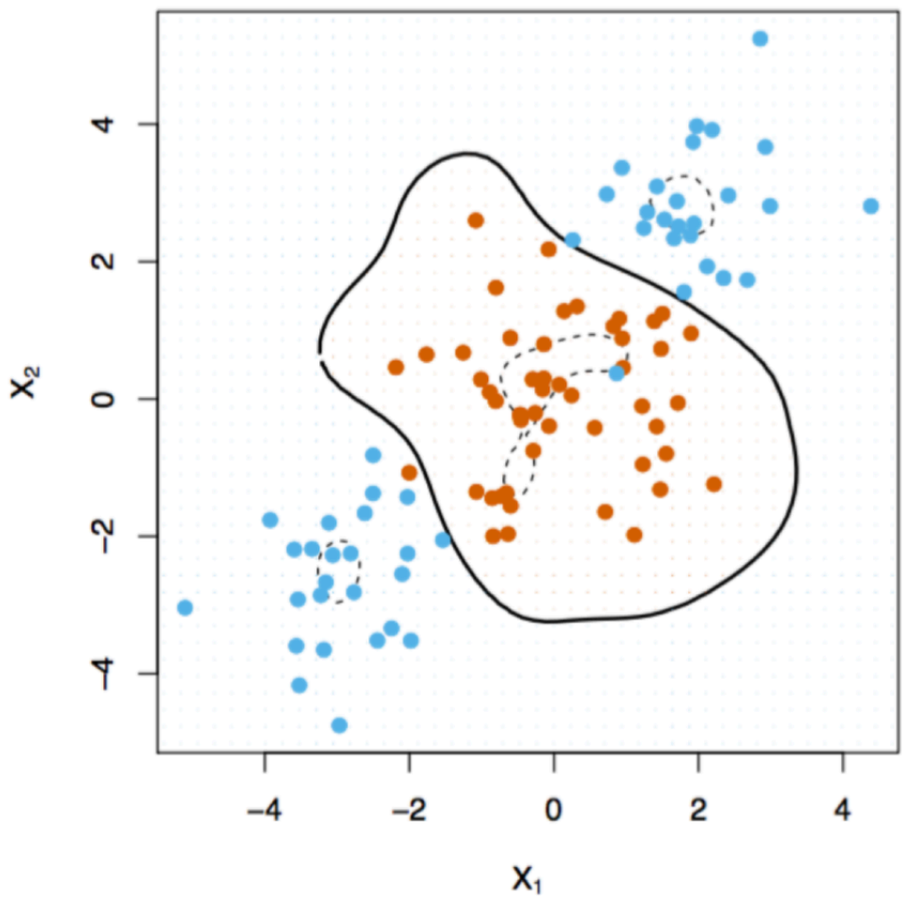
classification with kernel

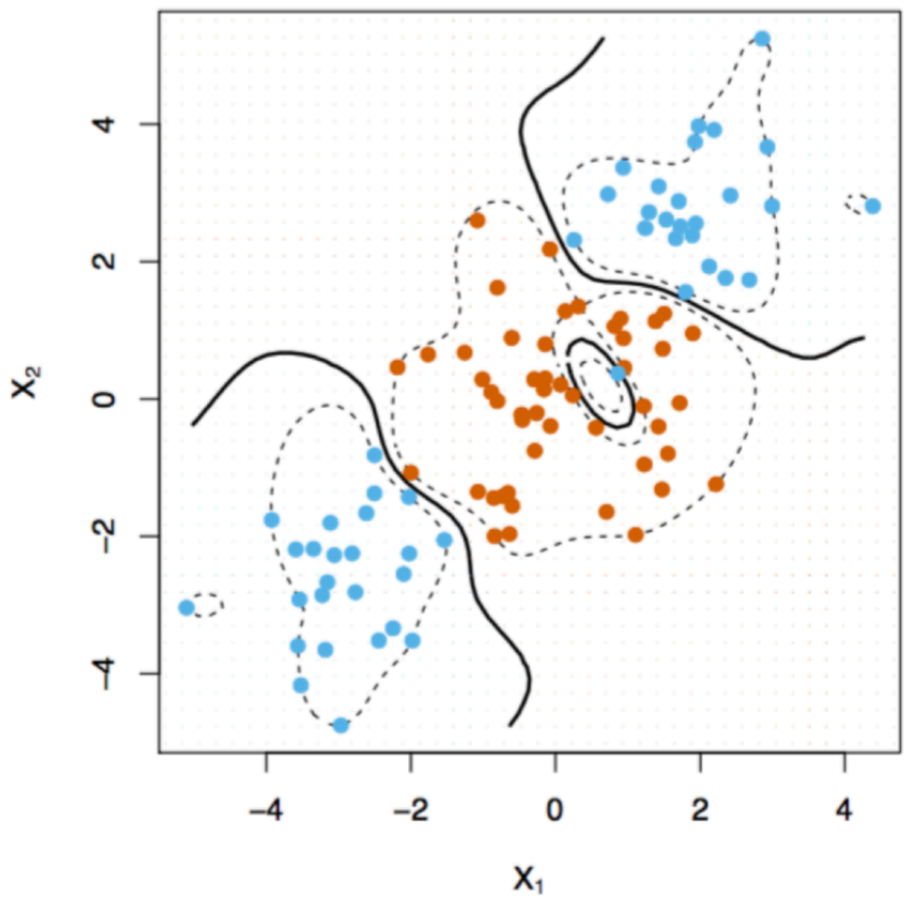
- $$\hat{w} = \arg\min_{w} \sum_{i=1}^{n} \max\{0, 1 - y_i(b + w^T h(x_i))\} + \lambda \|w\|_2^2$$

  using kernels, it can be simplified as

  $$\hat{\alpha} = \arg\min_{\alpha} \sum_{i=1}^{n} \max\{0, 1 - y_i(b + \alpha^T \mathbf{K}[:, i])\} + \lambda \alpha^T \mathbf{K} \alpha$$

overfitting when σ is small with RBF kernel





Bootstrap

finding confidence interval

confidence interval

- suppose you have training data $\{(x_i, y_i)\}_{i=1}^n$ drawn i.i.d. from some true distribution $P_{x,y}$
- we train a kernel ridge regressor, with some choice of a kernel $K : \mathbb{R}^{d \times d} \to \mathbb{R}$

$$\text{minmize}_\alpha \; \|\mathbf{K}\alpha - \mathbf{y}\|_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

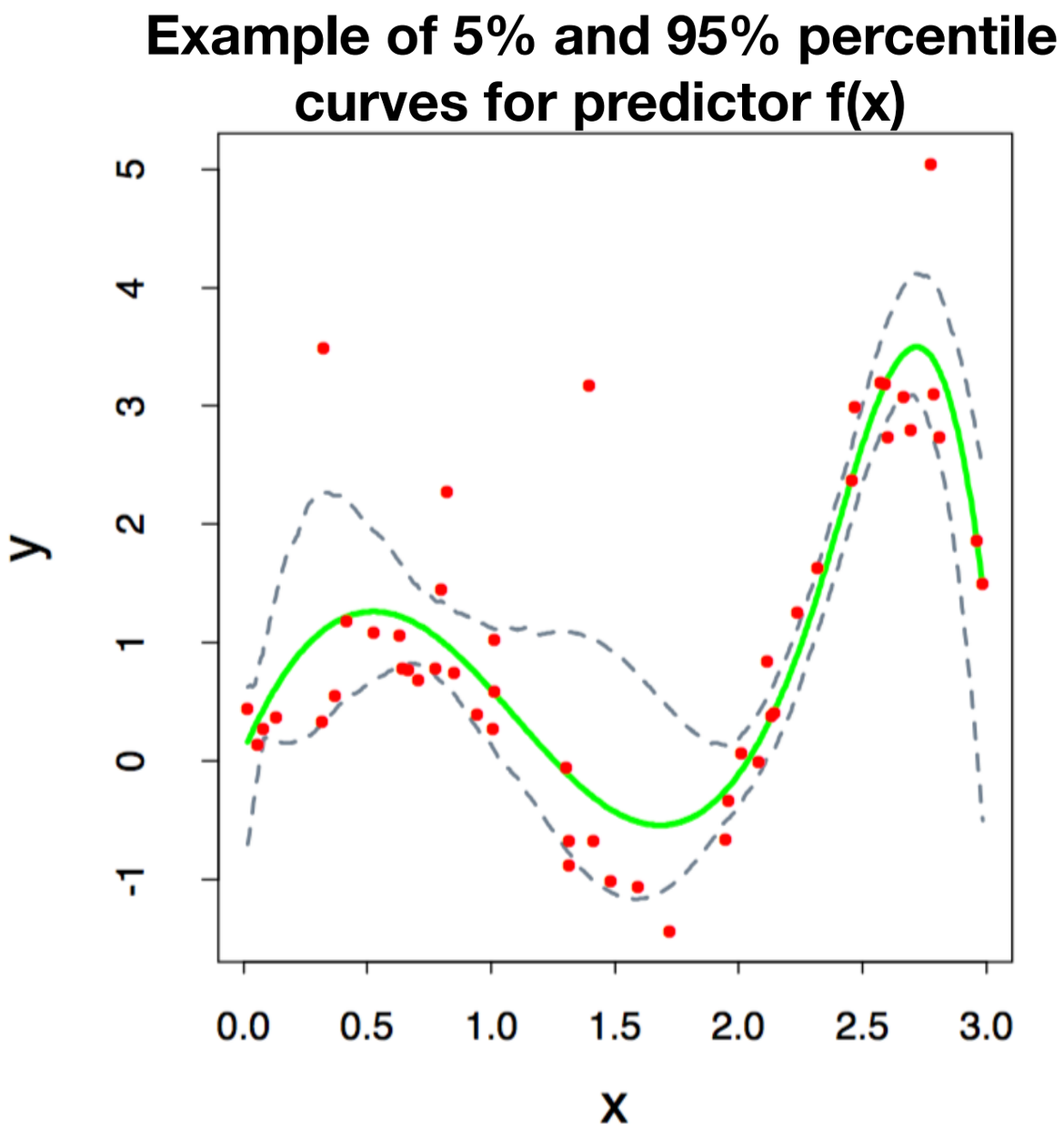
- the resulting predictor is

$$f(x) = \sum_{i=1}^{n} K(x_i, x)\hat{\alpha}_i,$$

where

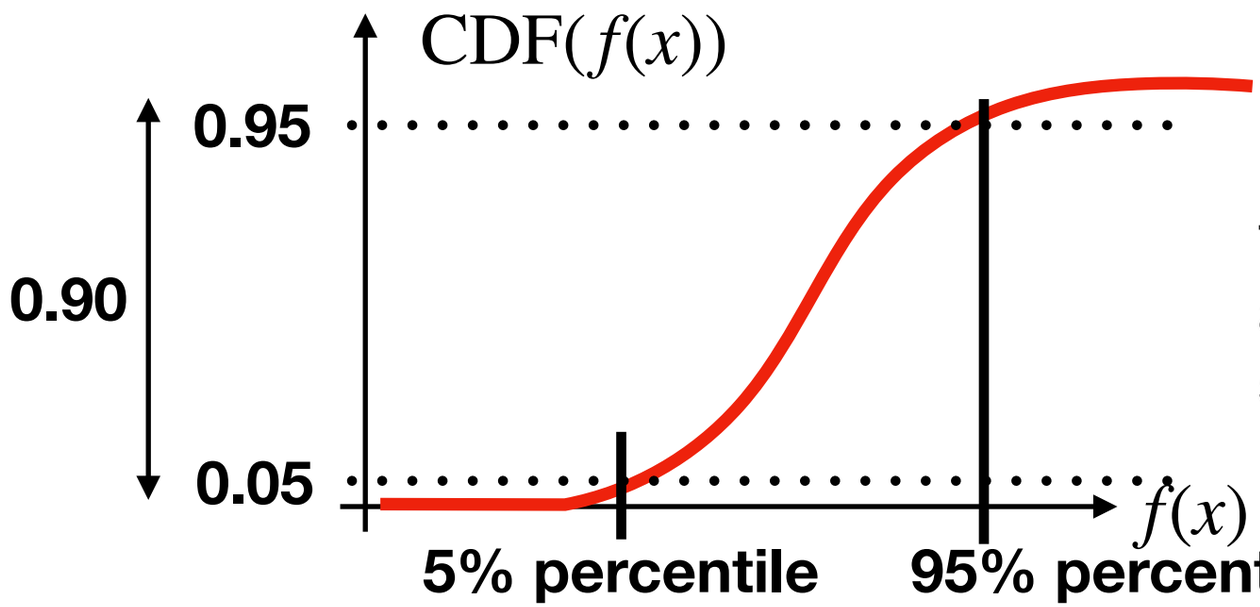
$$\hat{\alpha} = (\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y} \quad \in \mathbb{R}^n$$

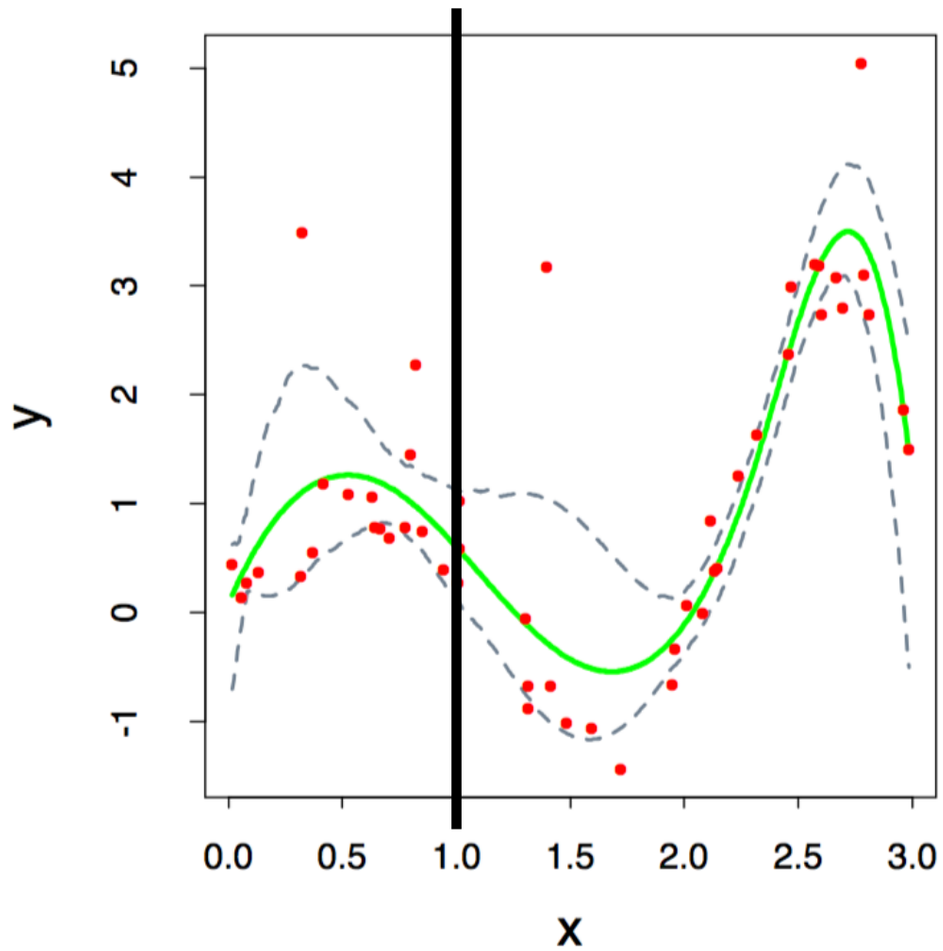
- we wish to build a confidence interval for our predictor $f(x)$, using 5% and 95% percentiles



confidence interval

- let's focus on a single $x \in \mathbb{R}^d$
- note that our predictor $f(x)$ is a random variable, whose randomness comes from the training data $S_{\text{train}} = \{(x_i, y_i)\}_{i=1}^n$
- if we know the statistics (in particular the CDF of the random variable $f(x)$) of the predictor, then the **confidence interval** with **confidence level 90%** is defined as





if we know the distribution of our predictor $f(x)$
the green line is the expectation $\mathbb{E}[f(x)]$
and the black dashed lines are the
5% and 95% percentiles in the figure above

- as we do not have the cumulative distribution function (CDF), we need to approximate them

confidence interval

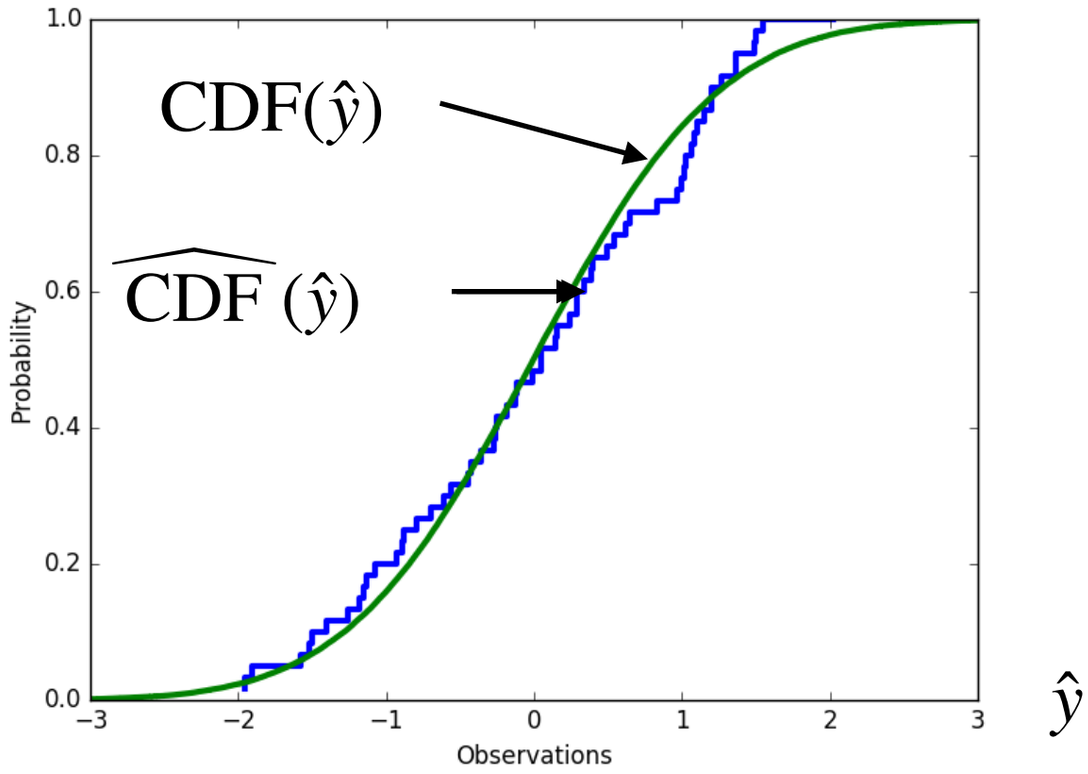
$\hat{y} = f(x) \in \mathbb{R}$. Random Variable

$\hat{y}^{(b)}$

- hypothetically, if we can sample as many times as we want, then we can train $B \in \mathbb{Z}^+$ i.i.d. predictors, each trained on n fresh samples to get **empirical estimate of the CDF of** $\hat{y} = f(x)$

- for b=1,…,B
  - draw n fresh samples
  - train a regularized kernel regression $\alpha^{*(b)}$
  - Predict $\hat{y}^{(b)} = (\alpha^{*(b)})^T h(x)$

$\{\hat{y}^{(1)}, \hat{y}^{(2)}, \ldots, \hat{y}^{(B)}\}$



- let the empirical CDF of those B predictors $\{\hat{y}^{(b)}\}_{b=1}^{B}$ be $\widehat{\text{CDF}}(\hat{y})$, defined as

$$\widehat{\text{CDF}}(\hat{y}) = \frac{1}{B}\sum_{b=1}^{B} \mathbf{I}\{\hat{y}^{(b)} \le \hat{y}\} = \frac{1}{B}\sum_{b=1}^{B} \mathbf{I}\{(\alpha^{*(b)})^T h(x) \le \hat{y}\}$$

- compute the confidence interval using $\widehat{\text{CDF}}(\hat{y})$

Bootstrap

- as we cannot sample repeatedly (in typical cases), we use **bootstrap samples** instead
- bootstrap is a general tool for assessing statistical accuracy
- we learn it in the context of confidence interval for trained models

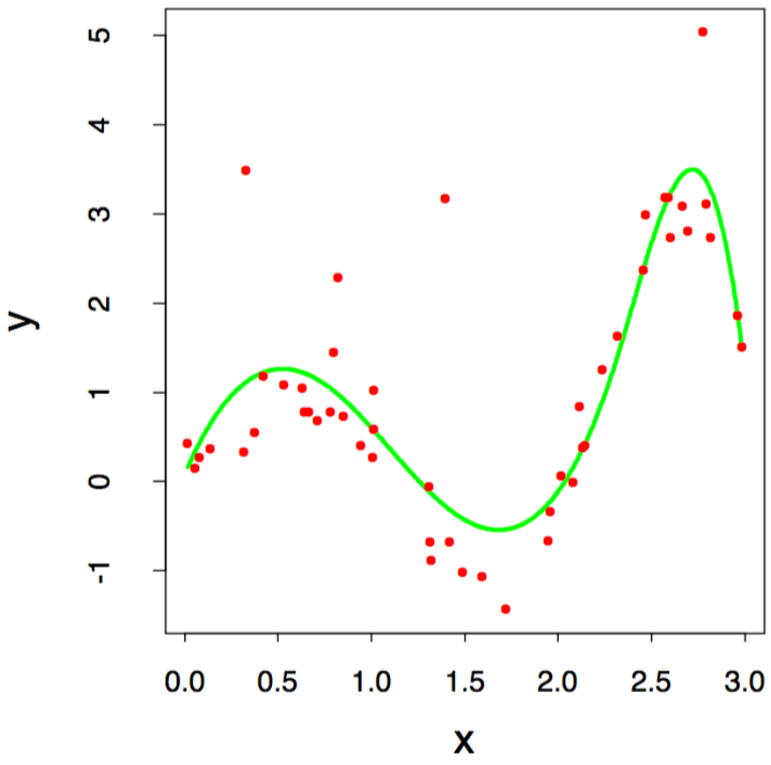
- a **bootstrap dataset** is created from the training dataset by taking n (the same size as the training data) examples uniformly at random **with replacement** from the training data $\{(x_i, y_i)\}_{i=1}^n$

- for b=1,…,B
  - create a bootstrap dataset $S_{\text{bootstrap}}^{(b)}$
  - train a regularized kernel regression $\alpha^{*(b)}$
  - predict $(\alpha^{*(b)})^T h(x)$

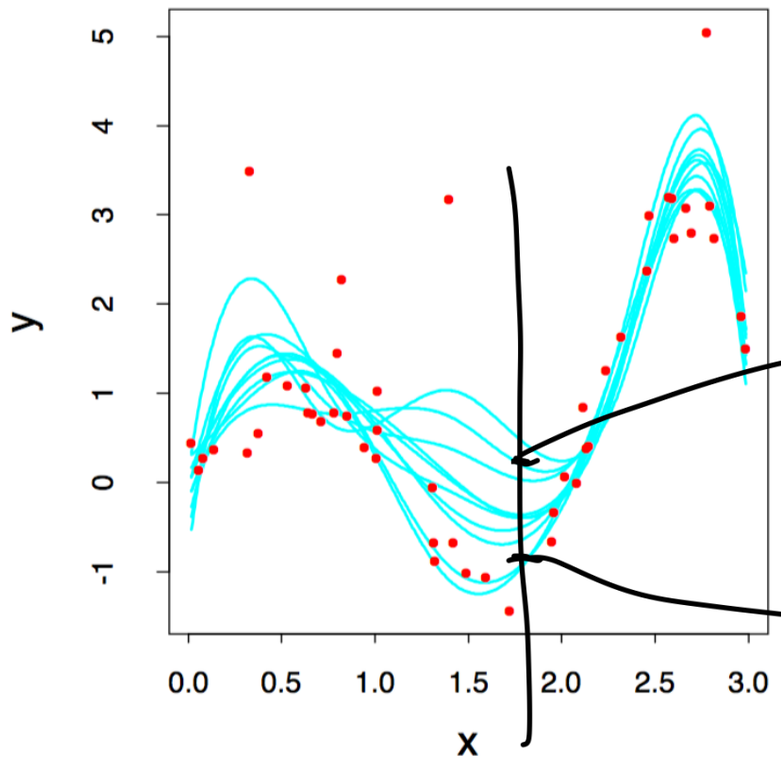
- compute the empirical CDF from the bootstrap datasets, and compute the confidence interval

bootstrap

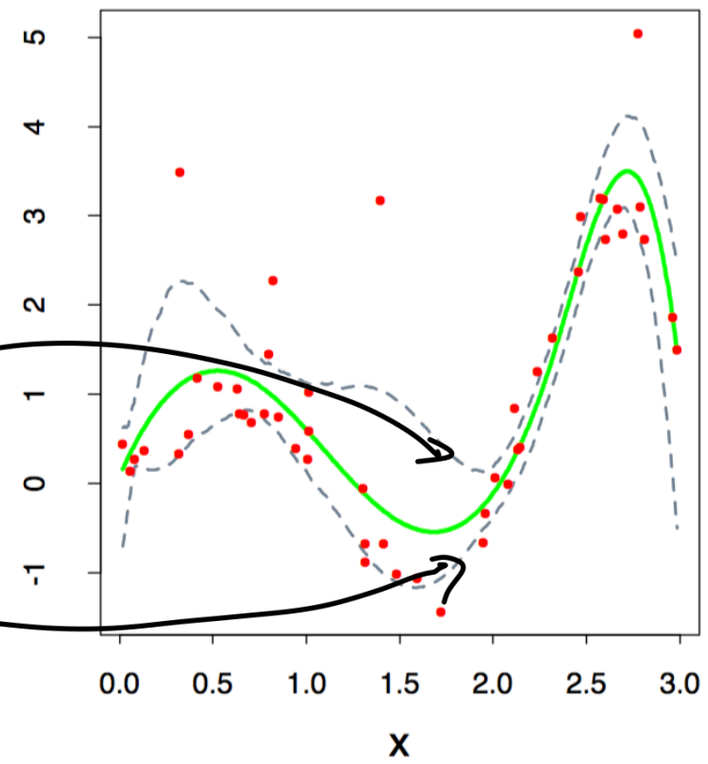
training a single predictor



multiple bootstrapped predictors



90% confidence interval



Figures from Hastie et al