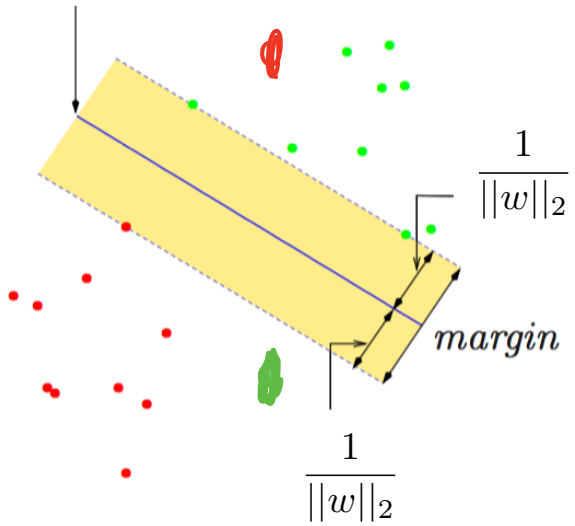
# Kernels



# What if the data is not linearly separable?

- $x^T w + b = 0$




some points don't satisfy margin constraint:

$$\min_{w,b} ||w||_2^2$$

$$y_i(x_i^T w + b) \geq 1 \quad \forall i$$

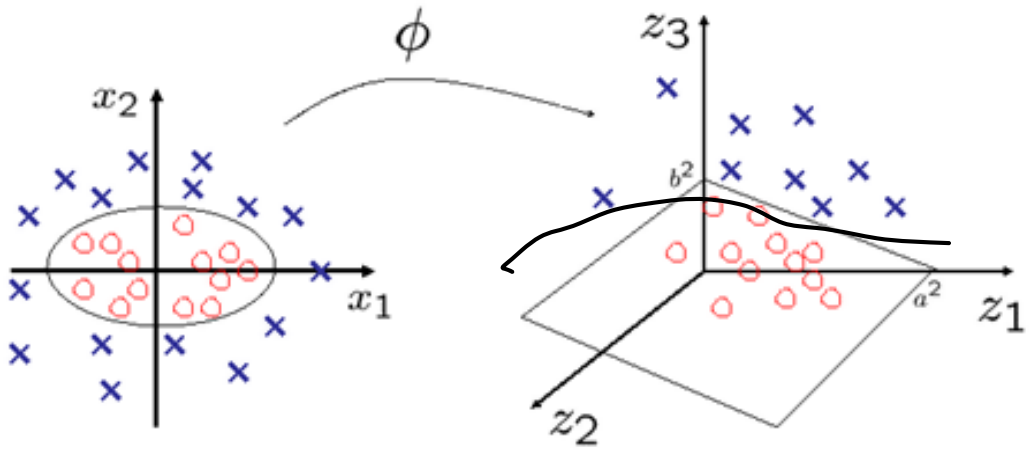
Two options:

1. Introduce slack to this optimization problem
2. **Lift to higher dimensional space**

# What if the data is not linearly separable?

## Use features of features of features…

$x \in \mathbb{R}^2$




we add new feature:
distance to origin

$$z_3 = \sqrt{x_1^2 + x_2^2}$$

$$z = \begin{pmatrix} x_1 \\ x_2 \\ \sqrt{x_1^2 + x_2^2} \end{pmatrix}$$

# Creating Features

$p >> d$

## Transformed data:

$h : \mathbb{R}^d \to \mathbb{R}^p$ maps original features to a rich, possibly high-dimensional space

in d=1:

$$h(x) = \begin{bmatrix} h_1(x) \\ h_2(x) \\ \vdots \\ h_p(x) \end{bmatrix} = \begin{bmatrix} x \\ x^2 \\ \vdots \\ x^p \end{bmatrix}$$

for d>1, generate

$$\{u_j\}_{j=1}^p \subset \mathbb{R}^d$$

$$h_j(x) = (u_j^T x)^2$$

$$h_j(x) = \frac{1}{1 + \exp(u_j^T x)}$$

$$h_j(x) = \cos(u_j^T x)$$

# Creating Features

## Transformed data:

$h : \mathbb{R}^d \to \mathbb{R}^p$ maps original features to a rich, possibly high-dimensional space

in d=1:

$$h(x) = \begin{bmatrix} h_1(x) \\ h_2(x) \\ \vdots \\ h_p(x) \end{bmatrix} = \begin{bmatrix} x \\ x^2 \\ \vdots \\ x^p \end{bmatrix}$$

for d>1, generate

$$\{u_j\}_{j=1}^p \subset \mathbb{R}^d$$

$$h_j(x) = (u_j^T x)^2$$

$$h_j(x) = \frac{1}{1 + \exp(u_j^T x)}$$

$$h_j(x) = \cos(u_j^T x)$$

**Feature space can get really large really quickly!**

# Degree-d Polynomials

input $u \in \mathbb{R}^2$, $u = (u_1, u_2)$

degree-1 polynomial

$$\phi(u) = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \in \mathbb{R}^2$$

degree-2 polynomial

$$\phi(u) = \begin{pmatrix} u_1 \cdot u_1 \\ u_1 u_2 \\ u_2 u_1 \\ u_2 \cdot u_2 \end{pmatrix} \in \mathbb{R}^4$$

degree-3 polynomial

$$\phi(u) = \begin{pmatrix} u_1 u_1 u_1 \\ u_1 u_1 u_2 \\ u_1 u_2 u_1 \\ u_1 u_2 u_2 \\ u_2 u_1 u_1 \\ u_2 u_1 u_2 \\ u_2 u_2 u_1 \\ u_2 u_2 u_2 \end{pmatrix} \in \mathbb{R}^8$$

if $u \in \mathbb{R}^d$

degree-K poly

$\phi(u) = O(d^K)$

exp in degree

=> h: # of data

# How do we deal with high-dimensional lifts/data?

## A fundamental trick in ML: use kernels

A function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a *kernel* for a map $\phi$ if $K(x, x') = \phi(x) \cdot \phi(x')$ for all $x, x'$.

So, if we can represent our algorithms/decision rules as dot products
and we can find a kernel for our feature map
then we can avoid explicitly dealing with ϕ(x).

# Linear Regression as Kernels

Train: $X \in \mathbb{R}^{n \times d}$, $y \in \mathbb{R}^n$, $\lambda$: reg

$$X = \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix}, \quad x_i \in \mathbb{R}^d$$

Least Square:

$$\hat{w} = \underbrace{(X^T X + \lambda \cdot I_d)^{-1}}_{d \times d} \underbrace{X^T y}_{d \times 1}, \quad I_d = \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{pmatrix}_{d \times d}$$

$$= X^T \underbrace{(XX^T + \lambda \cdot I_n)^{-1}}_{n \times n} y \in \mathbb{R}^d$$

$d \times n \quad n \times 1$

(linear algebra 5 ✓ 1)

$\hat{w} = \sum \alpha_i x_i$, $[(XX^T + \lambda I_n)^{-1} y]_i = \alpha_i$

Decision rule:

$x_{new} \in \mathbb{R}^d$

$$\hat{y}_{new} = \hat{w}^T x_{new} = y^T (XX^T + \lambda I_n)^{-1} X x_{new}$$

$n \times 1$

$\phi(x) = x$

$\forall x, x' : K(x, x') = x^T x'$

$XX^T \in \mathbb{R}^{n \times n}$

$i, j : 1, \dots, n \quad [XX^T]_{ij} = x_i^T x_j = K(x_i, x_j)$

$$X X_{new} = \begin{pmatrix} x_1^T x_{new} \\ \vdots \\ x_n^T x_{new} \end{pmatrix} = \begin{pmatrix} K(x_1, x_{new}) \\ \vdots \\ K(x_n, x_{new}) \end{pmatrix}$$

increasing $d \gg n$

# Dot-product of polynomials

$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) =$ polynomials of degree exactly d

$$d = 1 : \phi(u) = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad \langle \phi(u), \phi(v) \rangle = u_1 v_1 + u_2 v_2$$

$$d = 2 : \phi(u) = \begin{bmatrix} u_1^2 \\ u_2^2 \\ u_1 u_2 \\ u_2 u_1 \end{bmatrix} \quad \langle \phi(u), \phi(v) \rangle = u_1^2 v_1^2 + u_2^2 v_2^2 + 2 u_1 u_2 v_1 v_2$$

$K(u, v) = (\langle u, v \rangle)^2$

degree-K polynomial:

(1) explicit $\phi$: $O(d^K)$

(2) $(\langle u, v \rangle)^K$: $O(d + \log K)$

# Dot-product of polynomials

$$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) = \text{polynomials of degree exactly d}$$

$$d = 1 : \phi(u) = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad \langle \phi(u), \phi(v) \rangle = u_1 v_1 + u_2 v_2$$

$$d = 2 : \phi(u) = \begin{bmatrix} u_1^2 \\ u_2^2 \\ u_1 u_2 \\ u_2 u_1 \end{bmatrix} \quad \langle \phi(u), \phi(v) \rangle = u_1^2 v_1^2 + u_2^2 v_2^2 + 2 u_1 u_2 v_1 v_2$$

**Feature space can get really large really quickly!**

General $d$ : Dimension of $\phi(u)$ is roughly $p^d$ if $u \in \mathbb{R}^p$

Feature expansion can be written **implicitly** $K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^p$

# Examples of Kernels

- **Polynomials of degree exactly $p$**

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^p$$

- **Polynomials of degree up to $p$**

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^p$$

- **Gaussian (squared exponential) kernel**

$$K(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{||\mathbf{u} - \mathbf{v}||^2}{2\sigma^2}\right)$$

- **Sigmoid**

$$K(u, v) = \tanh(\gamma \cdot u^T v + r)$$

Pipeline

# The Kernel Trick

Data $\{(x_i, y_i)\}_{i=1}^{n}$

1. **Pick a kernel K** $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ — $\alpha$ depends on $\langle x_i, x_j \rangle, y$
2. **For a linear predictor, show** $w = \sum_i \alpha_i x_i$
3. **Change loss function/decision rule to only access data through dot products**

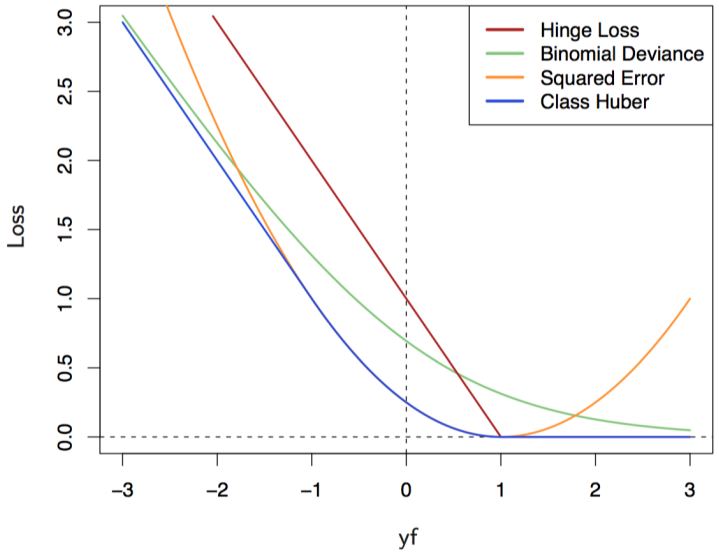
$x_{new} \in \mathbb{R}^d$, $w^T x_{new} = \sum_{i=1}^{n} \alpha_i \langle x_i, x_{new} \rangle$

$\sum_{i=1}^{n} \alpha_i K(x_i, x_{new})$

**Substitute** $K(x_i, x_j)$ for $x_i^T x_j$

Gaussian / RBF ...

# Loss Functions




$$\{(x_i, y_i)\}_{i=1}^n \qquad x_i \in \mathbb{R}^d \qquad y_i \in \mathbb{R}$$

$$w = \sum_{i=1}^{n} \alpha_i \cdot x_i$$

- Loss functions: $\min_w \sum_{i=1}^{n} \ell_i(w)$

$$\min_{\alpha} \sum_{i=1}^{n} \ell_i(\alpha)$$

Squared error Loss: $\ell_i(w) = (y_i - x_i^T w)^2$

Logistic Loss: $\ell_i(w) = \log(1 + \exp(-y_i\, x_i^T w))$

0/1 loss: $\ell_i(w) = \mathbb{I}[\text{sign}(y_i) \neq \text{sign}(x_i^T w)]$

Hinge Loss: $\ell_i(w) = \max\{0, 1 - y_i x_i^T w\}$

# The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_w \sum_{i=1}^{n} (y_i - x_i^T w)^2 + \lambda ||w||_2^2$$

$$w^T w = \left(\sum_{i=1}^{n} \alpha_i x_i\right)\left(\sum_{i=1}^{n} \alpha_i x_i\right)$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i$

$$x_i^T \hat{w} = \sum_{j=1}^{n} \alpha_j \langle x_i, x_j \rangle$$

$$\hat{\alpha} = \arg\min_{\alpha} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{n} \alpha_j \langle x_i, x_j \rangle\right)^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \langle x_i, x_j \rangle$$

$\langle x_i, x_j \rangle \rightarrow K(x_i, x_j)$

$$\Rightarrow \arg\min_{\alpha} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{n} \alpha_j K(x_i, x_j)\right)^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j K(x_i, x_j)$$

(Quadratic form)

$$= \arg\min_{\alpha} \|y - K\alpha\|_2^2 + \lambda \alpha^T K \alpha$$

$K : n \times n$, $K_{ij} = K(x_i, x_j)$

# The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_{w} \sum_{i=1}^{n} (y_i - x_i^T w)^2 + \lambda ||w||_w^2$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i$

$$\widehat{\alpha} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \langle x_i, x_j \rangle$$

$$= \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j K(x_i, x_j))^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j K(x_i, x_j)$$

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

$$K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$$

n: # of data

## Why regularization?

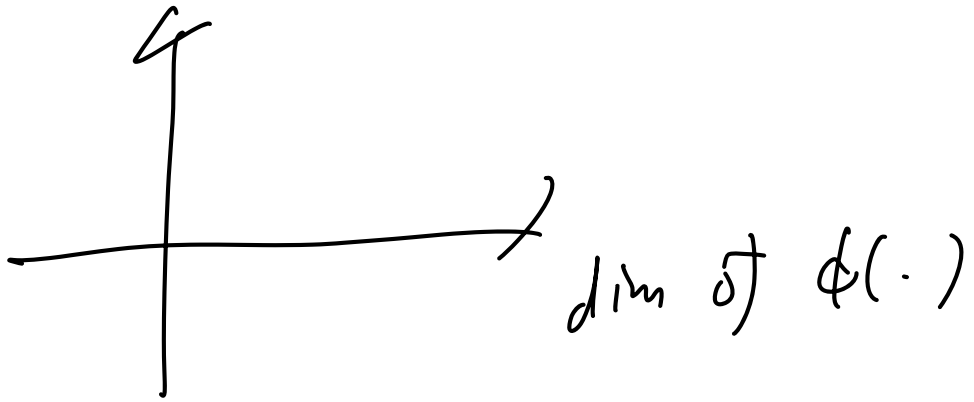
positive definite, $\forall v \neq 0$, $v^T K v \geq 0$

Typically, $\mathbf{K} \succ 0$. What if $\lambda = 0$?

$$\widehat{\alpha} = \arg\min_{\alpha} ||\mathbf{y} - \mathbf{K}\alpha||_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

if $\phi(x) \in \mathcal{R}^p$, if $p \gg n$

generally speaking, $K \succ 0$



# Why regularization?

$$\hat{y} = \begin{pmatrix} K(x_1,x_1) & \cdots & K(x_1,x_n) \\ & \ddots & \\ K(x_n,x_1) & & \end{pmatrix} \alpha$$

Typically, $\mathbf{K} \succ 0$. What if $\lambda = 0$?

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

$$= K\alpha$$

$$= K \cdot K^{-1} y$$

$$= y$$

Unregularized kernel least squares can (over) fit **any data**!

$$\hat{\alpha} = \mathbf{K}^{-1}\mathbf{y}$$

Let $\hat{y} \in \mathbb{R}^n$ predictions on training data.

$i = 1, \ldots, n$

$$\hat{y}_i = \begin{pmatrix} K(x_1,x_i) \\ \vdots \\ K(x_n,x_i) \end{pmatrix}^T \alpha$$

$$w = \sum_{i=1}^{n} \alpha_i x_i$$

# The Kernel Trick for SVMs

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

$$\Rightarrow \min_{\alpha,b} \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i(b + \sum_{j=1}^{n} \langle x_i, x_j \rangle)\} + \lambda \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j \langle x_i, x_j \rangle$$

replace $\langle x_i, x_j \rangle \rightarrow K(x_i, x_j)$

$$\Rightarrow \min_{\alpha,b} \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i(b + \sum_{j=1}^{n} K(x_i, x_j)\} + \lambda \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j K(x_i, x_j)$$

convex in $\alpha, b$

$\hat{\alpha}, \hat{b}$: solution

Decision rule: $x_{new}$

$$\sum_{j=1}^{n} \hat{\alpha}_j K(x_j, x_{new}) + \hat{b} \Rightarrow \begin{cases} > 0 \Rightarrow \text{predict } 1 \\ < 0 \Rightarrow \text{predict } -1 \end{cases}$$

Gaussian

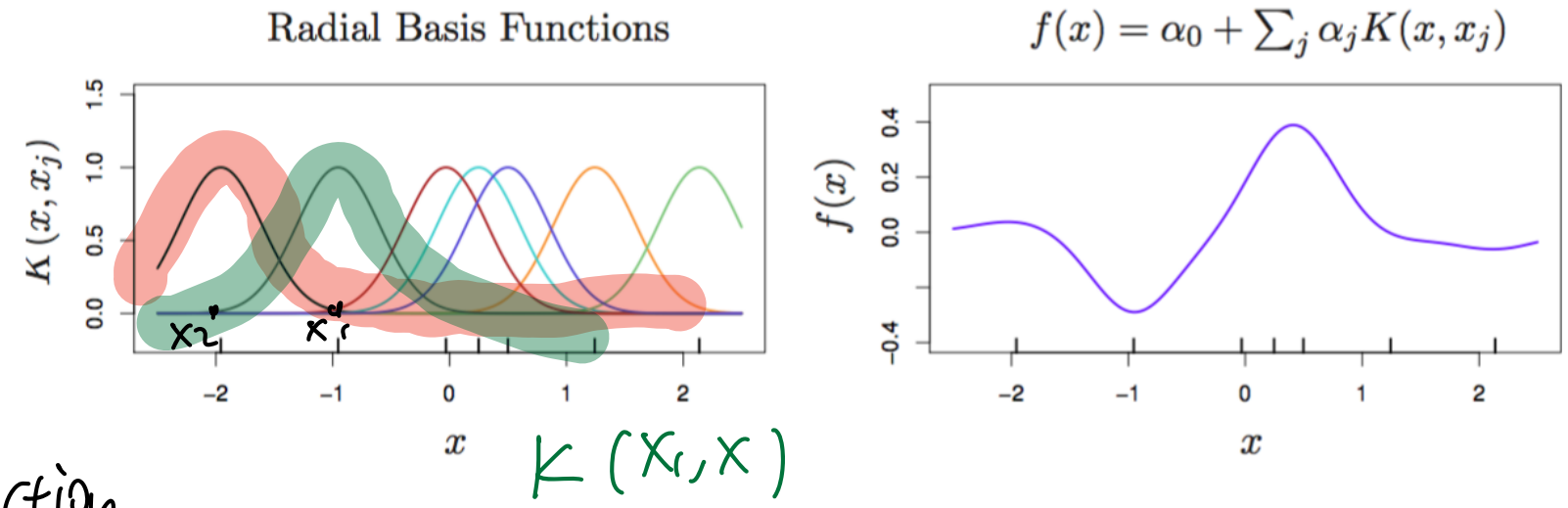
# RBF Kernel

$$K(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{||\mathbf{u}-\mathbf{v}||_2^2}{2\sigma^2}\right)$$

$$= \langle \phi(u), \phi(v) \rangle$$

**This is like weighting "bumps" on each point**

$x \in R$




$K(x_1, x)$

function

$K(x_2, x)$

max: $x = x_2$

$K(x_2, x) = \exp\left(-\frac{||x_2 - x||_2^2}{2\sigma^2}\right)$

# RBF Kernel

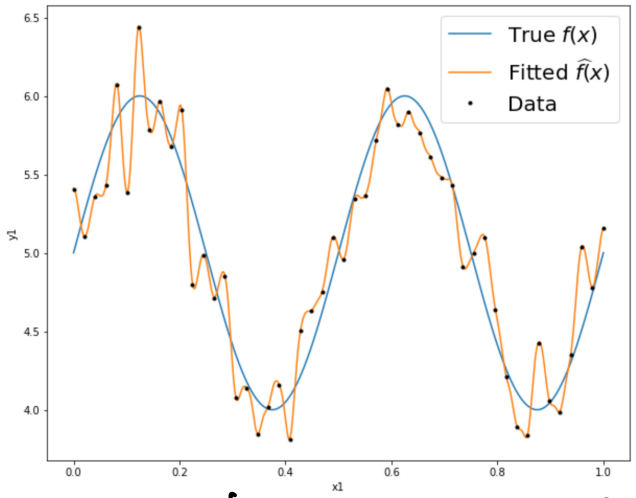
$$K(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{||\mathbf{u}-\mathbf{v}||_2^2}{2\sigma^2}\right)$$

$\lambda$: regularization

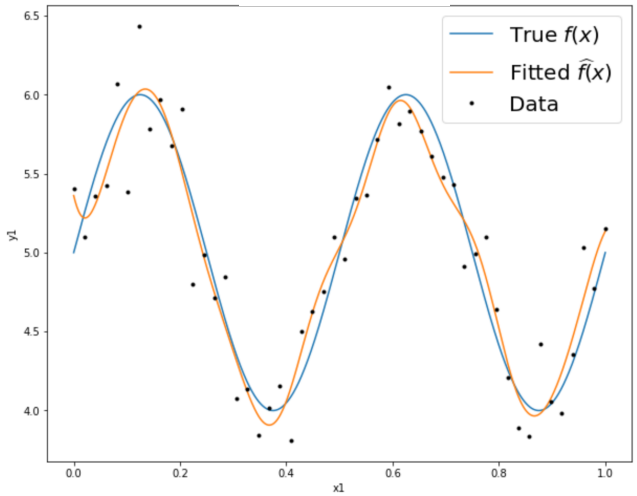
bandwidth

The bandwidth sigma has an enormous effect on fit:

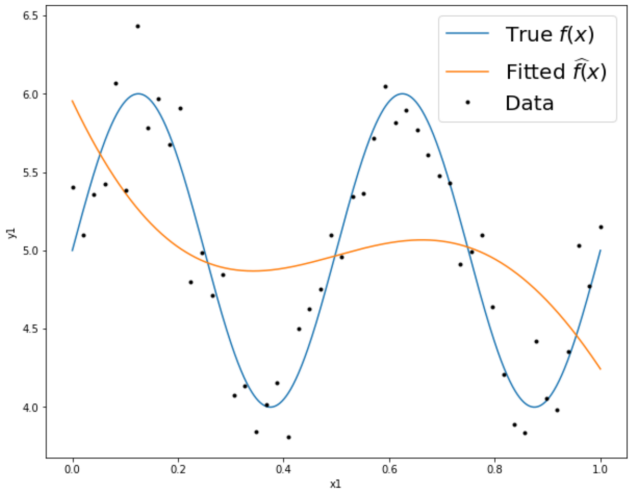
$\sigma = 10^{-2}$ $\lambda = 10^{-4}$




$\sigma = 10^{-1}$ $\lambda = 10^{-4}$




$\sigma = 10^{-0}$ $\lambda = 10^{-4}$




$\sigma$: bias – variance

as $\sigma \to 0$, only if $x = x_i$, $K(x, x_i) \neq 0$

$\sigma \to \infty$, $K(x_i, x) = 1$, average of $y$

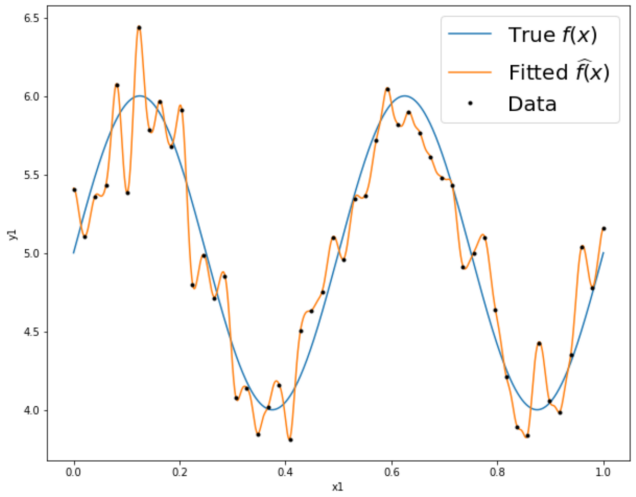
$$\hat{f}(x) = \sum_{i=1}^{n} \hat{\alpha}_i K(x_i, x)$$

# RBF Kernel

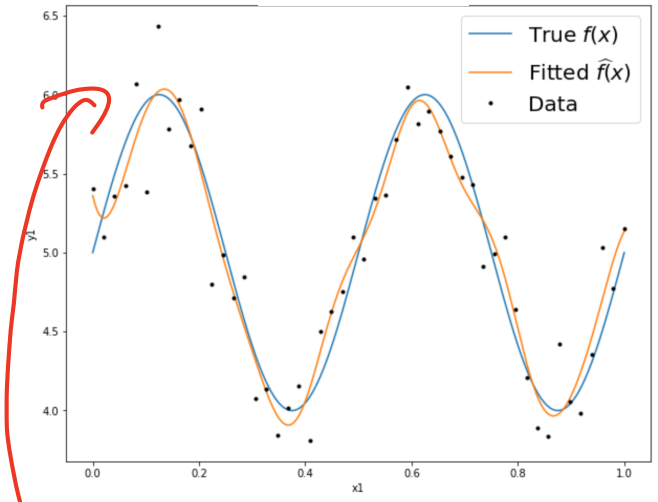
$$K(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{||\mathbf{u} - \mathbf{v}||_2^2}{2\sigma^2}\right)$$

The bandwidth sigma has an enormous effect on fit:

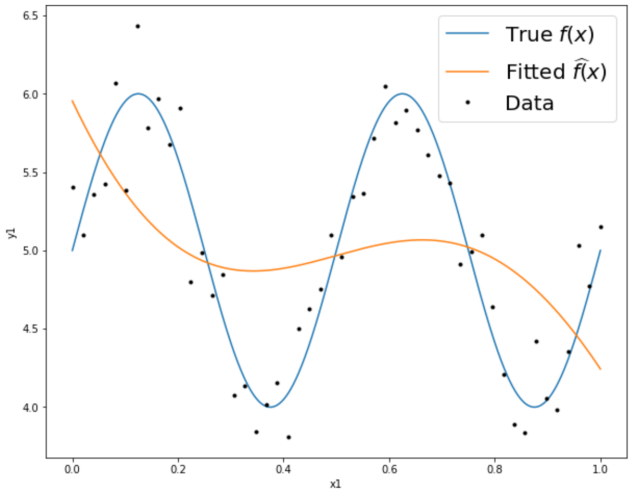
$\sigma = 10^{-2}$ $\lambda = 10^{-4}$



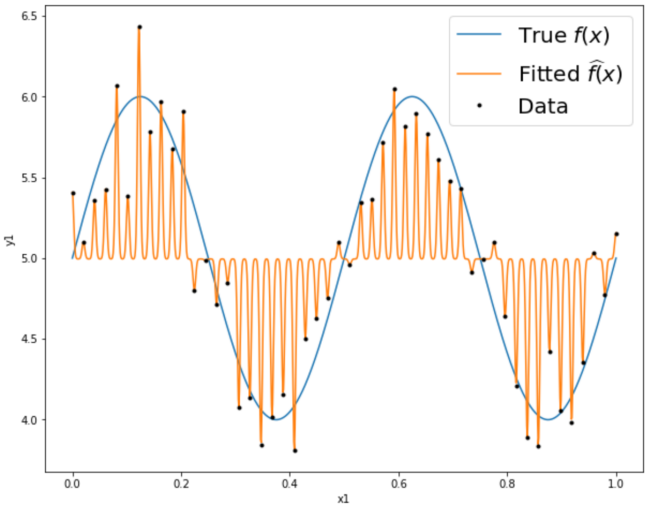

$\sigma = 10^{-1}$ $\lambda = 10^{-4}$



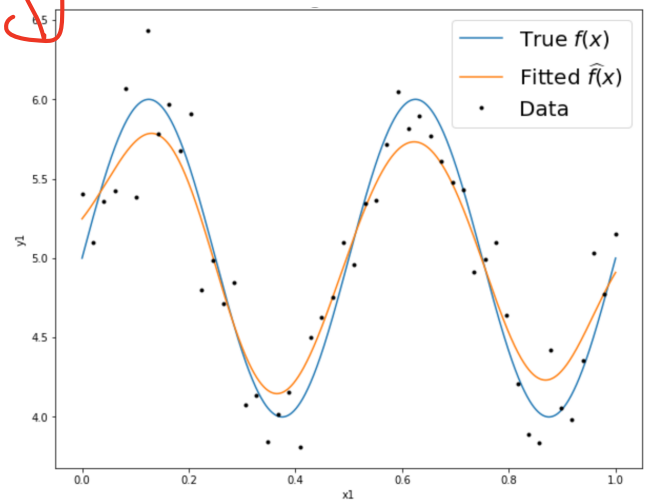

$\sigma = 10^{-0}$ $\lambda = 10^{-4}$




$\sigma = 10^{-3}$ $\lambda = 10^{-4}$




$\sigma = 10^{-1}$ $\lambda = 10^{-0}$

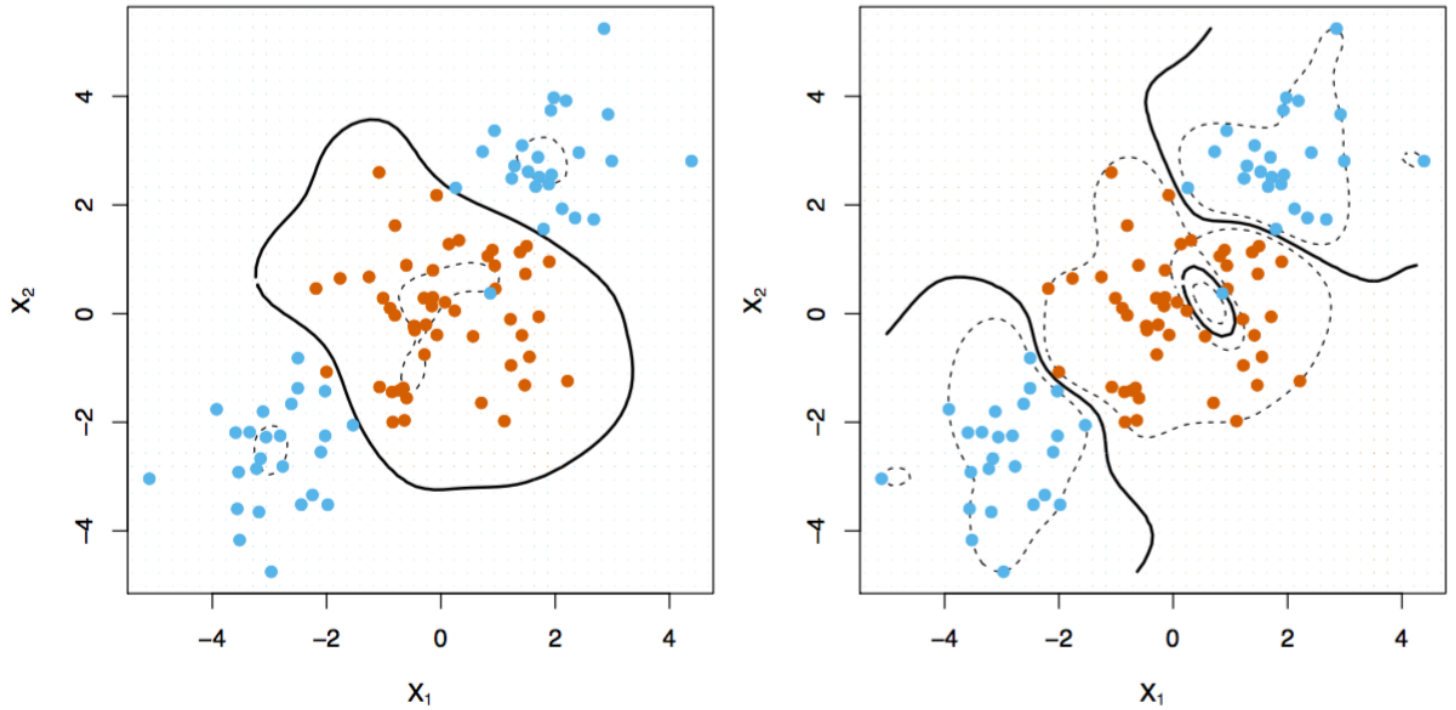



$$\widehat{f}(x) = \sum_{i=1}^{n} \widehat{\alpha}_i K(x_i, x)$$

# RBF kernel

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

$$\min_{\alpha,b} \sum_{i=1}^{n} \max\{0, 1 - y_i(b + \sum_{j=1}^{n} \alpha_j \langle x_i, x_j \rangle)\} + \lambda \sum_{i,j=1}^{n} \alpha_i \alpha_j \langle x_i, x_j \rangle$$



# RBF kernel and random features

$$K(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{\|\mathbf{u}-\mathbf{v}\|_2^2}{2\sigma^2}\right)$$

If n is very large, allocating an n-by-n matrix is tough.

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

→ worst case $O(n^3)$

$O(n^2)$

n is large

n: 1m, 100b

RBF : $\phi_\infty$ inf-dim feature space / $K(u,v) = \langle \phi_\infty(u), \phi_\infty(v) \rangle$

# RBF kernel and random features

find $\phi$, $\langle \phi(u), \phi(v) \rangle \approx K(u,v)$

$$K(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{||\mathbf{u} - \mathbf{v}||_2^2}{2\sigma^2}\right)$$

If n is very large, allocating an n-by-n matrix is tough.

use sampling instead of using $K(\cdot,\cdot)$

use $\phi$, $p \ll n$,

$$\phi(x) = \begin{bmatrix} \sqrt{2}\cos(w_1^T x + b_1) \\ \vdots \\ \sqrt{2}\cos(w_p^T x + b_p) \end{bmatrix}$$

$$2\cos(\alpha)\cos(\beta) = \cos(\alpha+\beta) + \cos(\alpha-\beta)$$

$$e^{jz} = \cos(z) + j\sin(z)$$

$w_k \sim \mathcal{N}(0, 2\gamma I)$ / $\gamma = \frac{1}{2\sigma^2}$

$b_k \sim \text{uniform}(0, \pi)$

$\mathbb{E}\left[\frac{1}{p}\phi(u)^T\phi(v)\right] = \frac{1}{p}\mathbb{E}\left(\sum_{k=1}^{p} 2\cos(w_k^T u + b_k)\cos(w_k^T v + b_k)\right)$

$= e^{-\gamma \|u-v\|_2^2}$

[Rahimi, Recht NIPS 2007]
"NIPS Test of Time Award, 2018"

# String Kernels

Graph Kernel

Example from Efron and Hastie, 2016

Amino acid sequences of different lengths:

x1

IPTSALVKETLALLSTHRTLLIANETLRIPVPVHKNHQLCTEEIFQGIGTLESQTVQGGTV
ERLFKNLSLIKKYIDGQKKKCGEERRRVNQFLDY**LQE**FLGVMNTEWI

x2

PHRRDLCSRSIWLARKIRSDLTALTESYVKHQGLWSELTEAER**LQE**NLQAYRTFHVLLA
RLLEDQQVHFTPTEGDFHQAIHTLLLQVAAFAYQIEELMILLEYKIPRNEADGMLFEKK
LWGLKV**LQE**LSQWTVRSIHDLRFISSHQTGIP

All subsequences of length 3 (of possible 20 amino acids) $20^3 = 8{,}000$

$$h^3_{\text{LQE}}(x_1) = 1 \text{ and } h^3_{\text{LQE}}(x_2) = 2.$$

# Fixed Feature V.S. Learned Feature