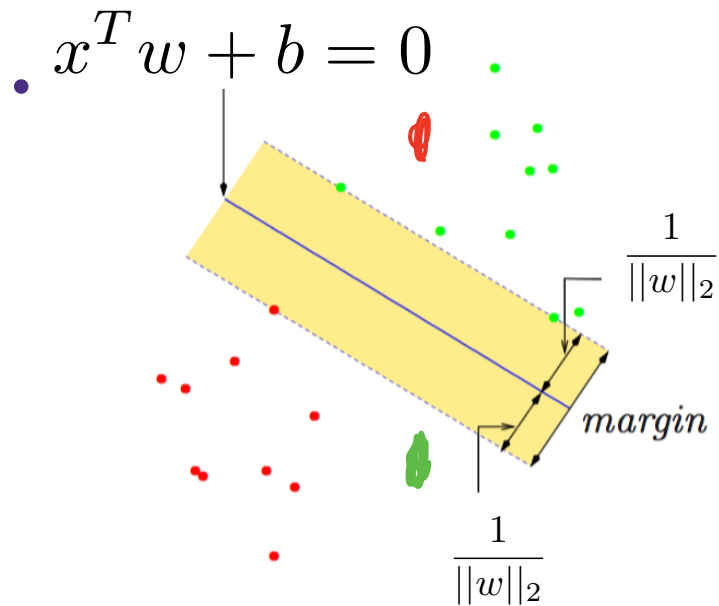


Kernels



What if the data is not linearly separable?



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?

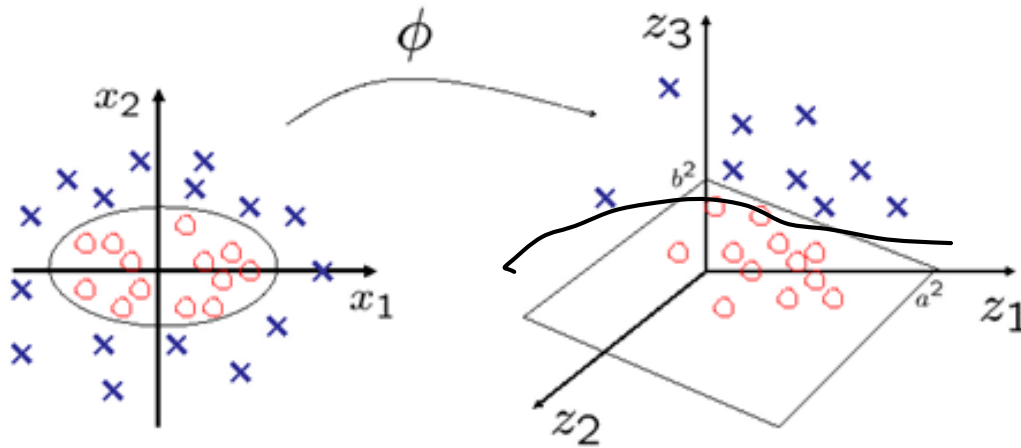
Use features of features of features...

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

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

$p \Rightarrow d$

Transformed data:

$h : \mathbb{R}^d \rightarrow \mathbb{R}^p$ 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:

$h : \mathbb{R}^d \rightarrow \mathbb{R}^p$ 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 $u \in \mathcal{L}^2$, $u = (u_1, u_2)$

degree-1 polynomial

$$\phi(u) =$$

$$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \in \mathcal{L}^2$$

degree-2 polynomial

polynomial

$$\phi(u) =$$

$$\begin{pmatrix} u_1 \cdot u_1 \\ u_1 \cdot u_2 \\ u_2 \cdot u_1 \\ u_2 \cdot u_2 \end{pmatrix}$$

$$\in \mathcal{L}^4$$

if $u \in \mathcal{R}^d$
degree-k poly

$$\phi(u) = \mathcal{O}(d^k)$$

exp in degree

$\gg n$: # of data

degree-3 polynomial

polynomial

$$\phi(u) =$$

$$\begin{pmatrix} u_1 u_1 u_1 \\ u_1 u_1 u_2 \\ u_1 u_2 u_1 \\ u_1 u_2 u_2 \\ u_2 u_1 u_1 \\ u_2 u_1 u_2 \\ u_2 u_2 u_1 \\ u_2 u_2 u_2 \end{pmatrix}$$

$$\in \mathcal{L}^8$$

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

A fundamental trick in ML: use kernels

A function $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ is a *kernel* for a map ϕ 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 $\phi(x)$.

Linear Regression as Kernels

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

$$X = \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix}$$

$x_i \in \mathbb{R}^d$

Least Square:

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

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

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

(linear algebra)
 (SVD)

Decision rule:

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

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

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

imagine $d \gg n$

$\forall x, x': K(x, x') = x^T x'$
 $X X^T \in \mathbb{R}^{n \times n}$
 $i, j: 1, \dots, n$
 $[X X^T]_{ij} = x_i^T x_j = K(x_i, x_j)$

Dot-product of polynomials

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

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

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

degree- k polynomial:

(1) explicit $\phi: \mathcal{O}(d^k)$

(2) $(\langle u, v \rangle)^k: \mathcal{O}(d + \log k)$

Dot-product of polynomials

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

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

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

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**

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

$\phi(u)$: all poly of deg p

- **Polynomials of degree up to p**

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

- **Gaussian (squared exponential) kernel**

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

$\phi(u)$: inf-dim

- **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 $K: \mathcal{L}^d \times \mathcal{L}^d \rightarrow \mathcal{R}$ α 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

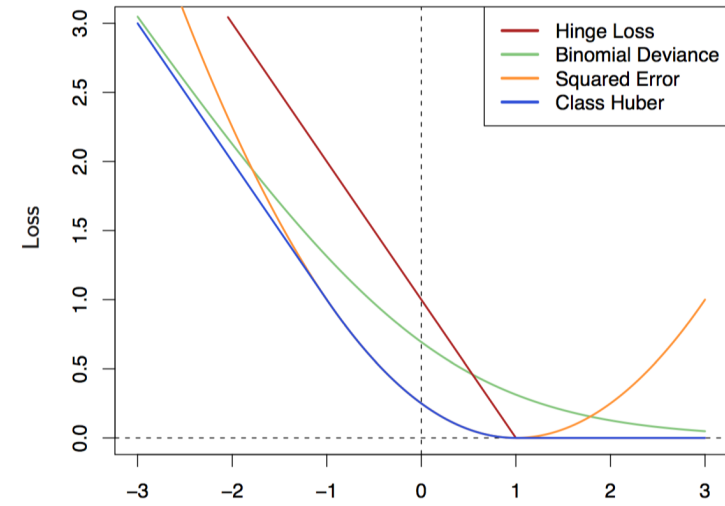
$$x_{\text{new}} \in \mathcal{L}^d, w^T x_{\text{new}} = \sum_{i=1}^n \alpha_i \langle x_i, x_{\text{new}} \rangle$$

Substitute

$K(x_i, x_j)$ for $x_i^T x_j$
Gaussian / poly ...

Loss Functions

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



- Loss functions:

$$\min_w \sum_{i=1}^n l_i(w)$$

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

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

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

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

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

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

The Kernel Trick for regularized least squares

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

$$= \begin{pmatrix} \sum_{i=1}^n \alpha_i x_i \end{pmatrix} \begin{pmatrix} \sum_{i=1}^n \alpha_i x_i \end{pmatrix} = w^T w$$

There exists an $\alpha \in \mathbb{R}^n$: $\hat{w} = \sum_{i=1}^n \alpha_i x_i$ $x_i^T \hat{w} = \sum_{j=1}^n \alpha_j \langle x_i, x_j \rangle$

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

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

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

(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

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

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

$$\begin{aligned} \hat{\alpha} &= \arg \min_{\alpha} \sum_{i=1}^n (y_i - \sum_{j=1}^n \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j \langle x_i, x_j \rangle \\ &= \arg \min_{\alpha} \sum_{i=1}^n (y_i - \sum_{j=1}^n \alpha_j K(x_i, x_j))^2 + \lambda \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j K(x_i, x_j) \\ &= \arg \min_{\alpha} \|\mathbf{y} - \mathbf{K}\alpha\|_2^2 + \lambda \alpha^T \mathbf{K}\alpha \end{aligned}$$

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

Why regularization?

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

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

Why regularization?

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

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

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

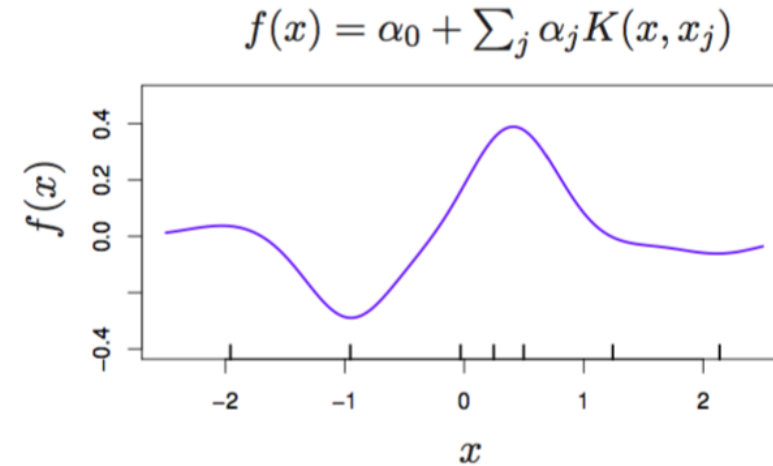
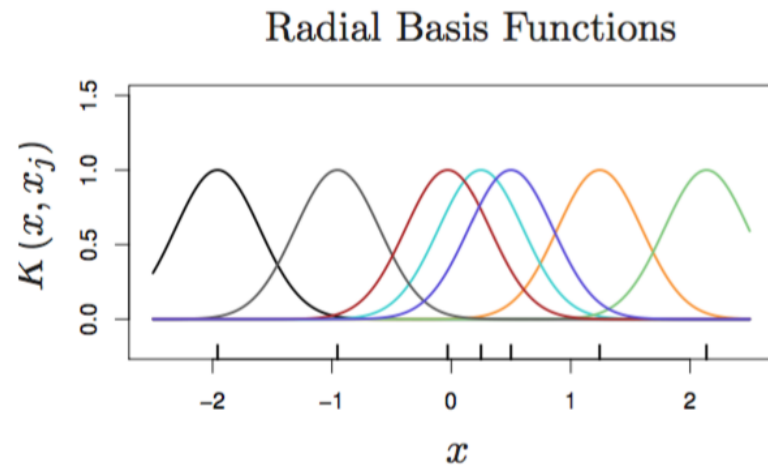
$$\hat{\alpha} = \mathbf{K}^{-1} \mathbf{y}$$

The Kernel Trick for SVMs

RBF Kernel

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

This is like weighting “bumps” on each point

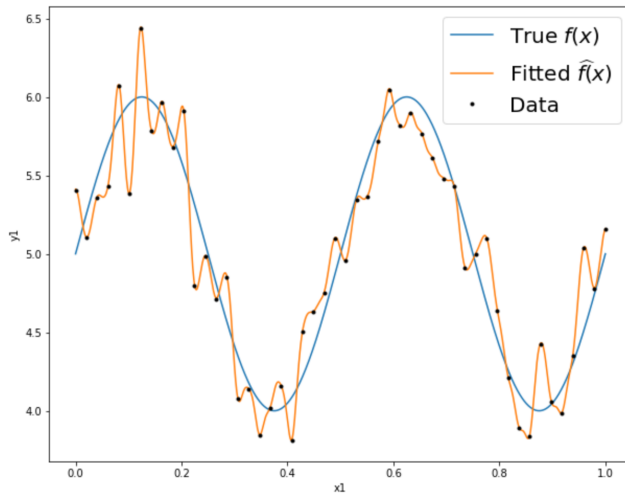


RBF Kernel

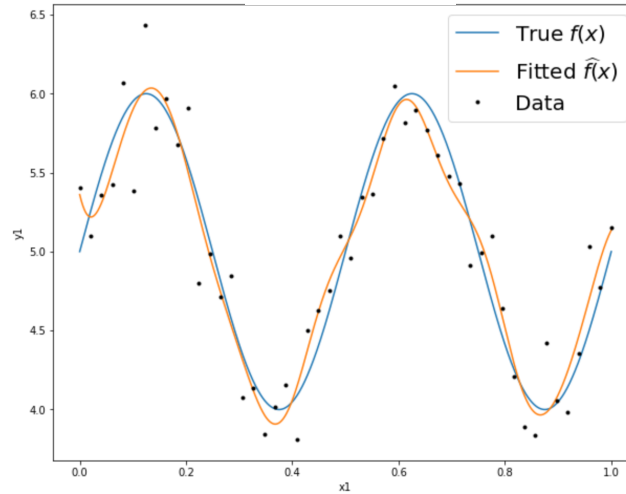
$$K(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{\|\mathbf{u} - \mathbf{v}\|_2^2}{2\sigma^2}\right)$$

The bandwidth sigma has an enormous effect on fit:

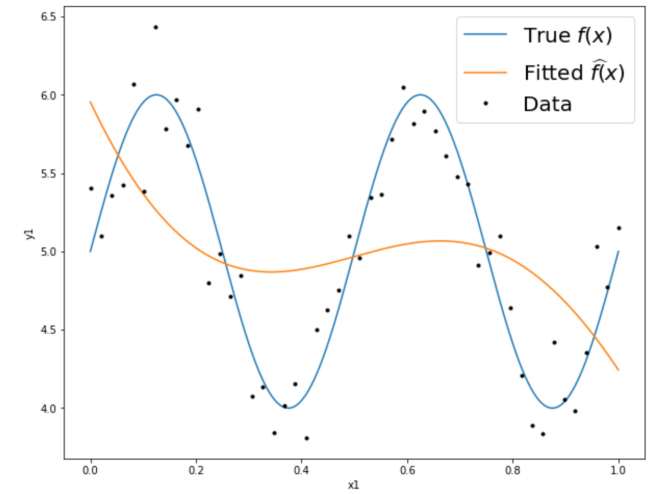
$$\sigma = 10^{-2} \quad \lambda = 10^{-4}$$



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



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



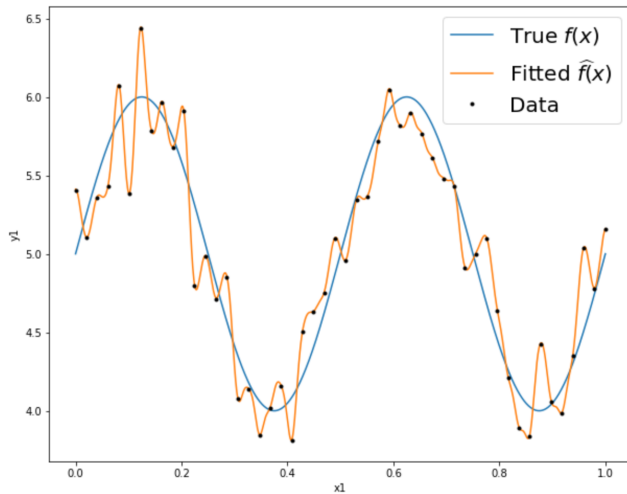
$$\hat{f}(x) = \sum_{i=1}^n \hat{\alpha}_i K(x_i, x)$$

RBF Kernel

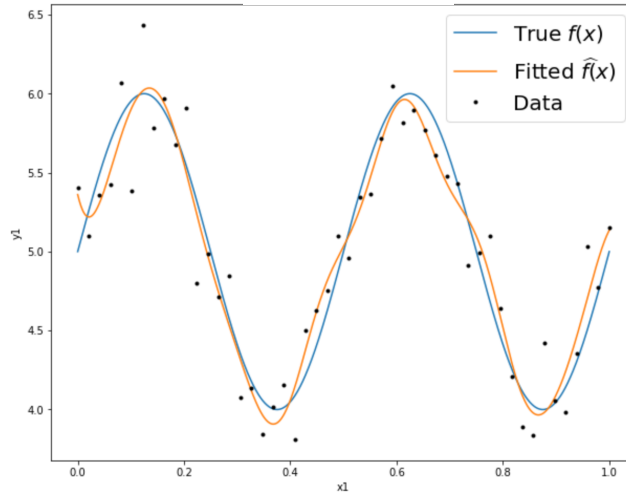
$$K(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{\|\mathbf{u} - \mathbf{v}\|_2^2}{2\sigma^2}\right)$$

The bandwidth sigma has an enormous effect on fit:

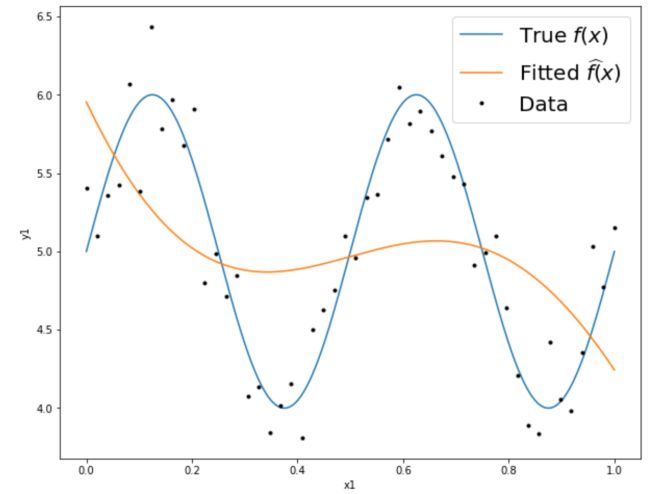
$$\sigma = 10^{-2} \quad \lambda = 10^{-4}$$



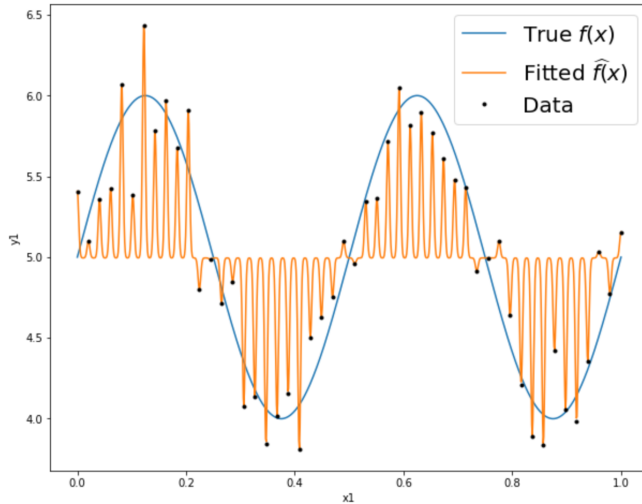
$$\sigma = 10^{-1} \quad \lambda = 10^{-4}$$



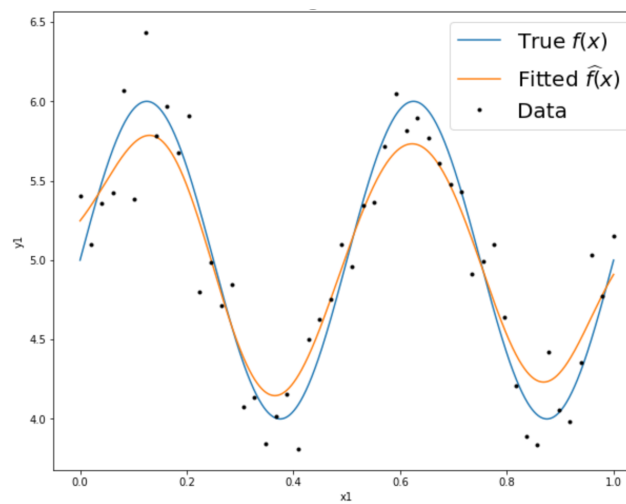
$$\sigma = 10^0 \quad \lambda = 10^{-4}$$



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



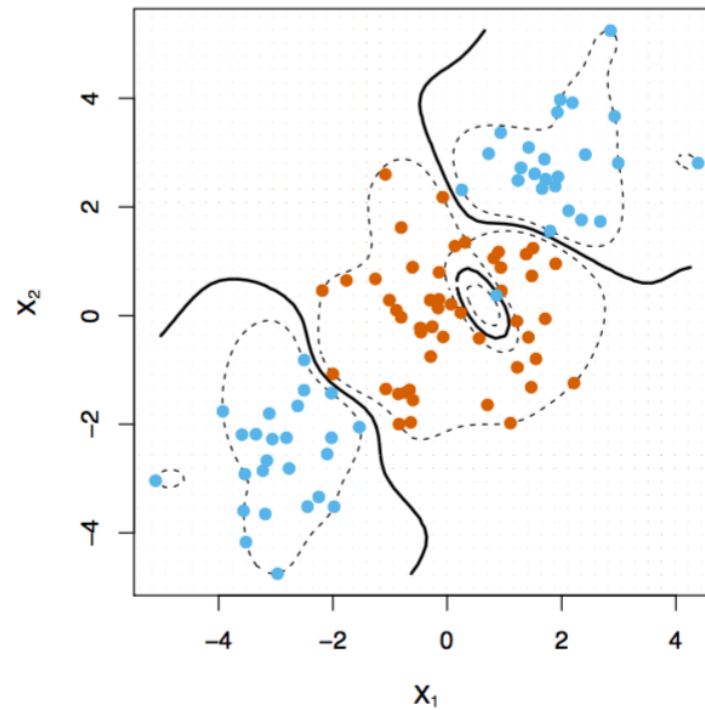
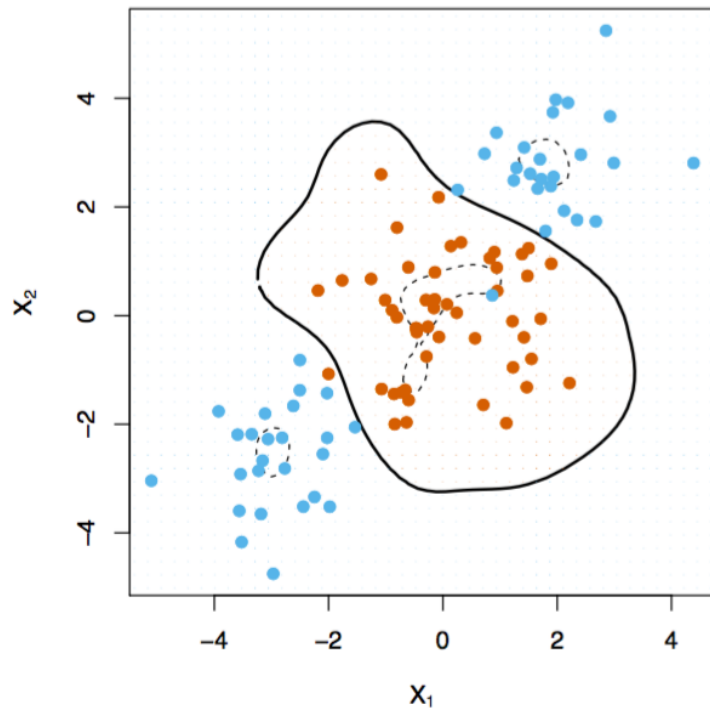
$$\sigma = 10^{-1} \quad \lambda = 10^{-0}$$



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

RBF kernel

$$\hat{w} = \sum_{i=1}^n \max\{0, 1 - y_i(b + x_i^T w)\} + \lambda \|w\|_2^2$$
$$\min_{\alpha, b} \sum_{i=1}^n \max\{0, 1 - y_i(b + \sum_{j=1}^n \alpha_j \langle x_i, x_j \rangle)\} + \lambda \sum_{i,j=1}^n \alpha_i \alpha_j \langle x_i, x_j \rangle$$



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.

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.

$$2 \cos(\alpha) \cos(\beta) = \cos(\alpha + \beta) + \cos(\alpha - \beta)$$
$$e^{jz} = \cos(z) + j \sin(z)$$

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

$$w_k \sim \mathcal{N}(0, 2\gamma I)$$

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

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

String Kernels

Example from Efron and Hastie, 2016

Amino acid sequences of different lengths:

x1

IPTSALVKETLALLSTHRTLIIANETLRIPVPVHKNHQLCTEEIFQGIGTLESQTVQGGTV
ERLFDKNSLIKKYIDGQKKKCGEERRRVNQFLDY**LQE**FLGVMNTEWI

x2

PHRRDLCSRSIWLARKIRSDLTALTESYVKHQGLWSELTEAER**LQEN**LQAYRTFHVLLA
RLLEDQQVHFTPTGDFHQAIHTLLLQVAAFAYQIEELMILLEYKIPRNEADGMLFEKK
LWGLKVL**LQE**LSQWTVRSIHDLRFISSHQTGIP

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

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

Fixed Feature V.S. Learned Feature
