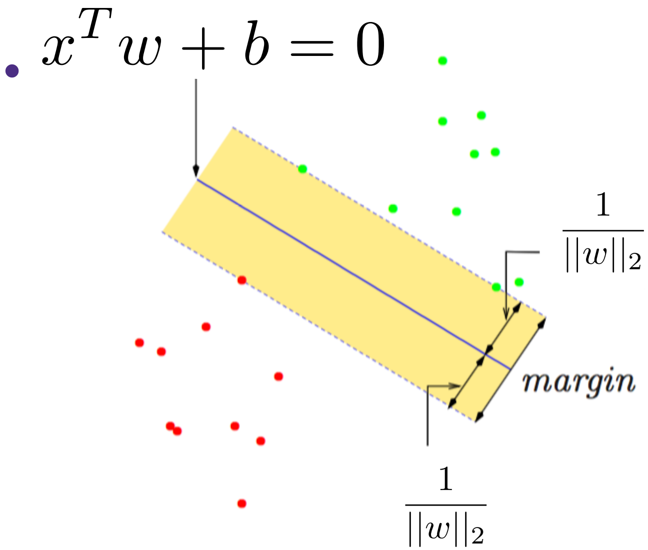
Kernel methods

What if the data is not linearly separable?



some points don't satisfy margin constraint:

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

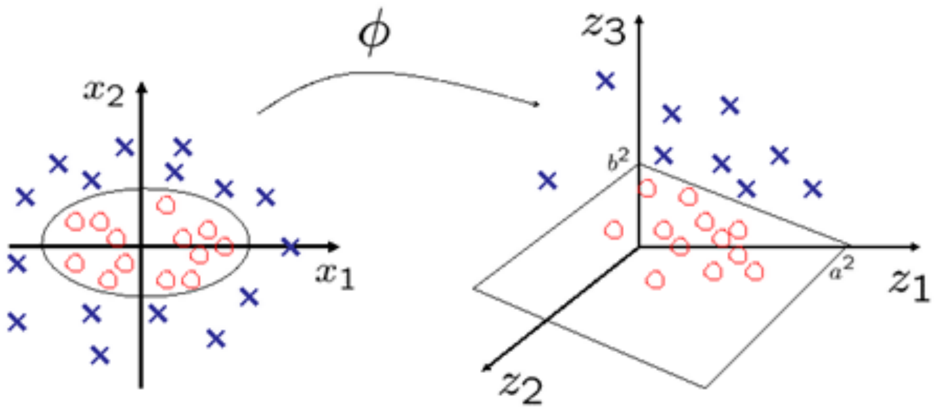
$$y_i(x_i^T w + b) \geq 1 \quad \forall i$$

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…



Creating Features

Transformed data:

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

for d>1, generate

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

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

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

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

in d=1:

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

Creating Features

Transformed data:

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

for d>1, generate

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

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

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

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

in d=1:

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

Feature space can get really large really quickly!

Degree-d Polynomials

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 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

Dot-product of polynomials

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

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

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

Dot-product of polynomials

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

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

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

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 d**

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

- **Polynomials of degree up to d**

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

- **Gaussian (squared exponential) kernel**

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

- **Sigmoid**

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

The Kernel Trick

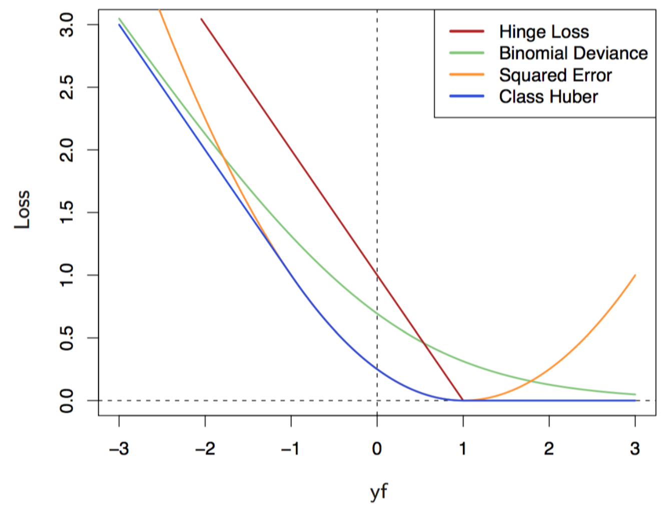
Pick a kernel K

For a linear predictor, show $w = \sum_i \alpha_i x_i$

Change loss function/decision rule to only access data through dot products

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

Loss Functions



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

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

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

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

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

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

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

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

Why regularization?

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

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

Why regularization?

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

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

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

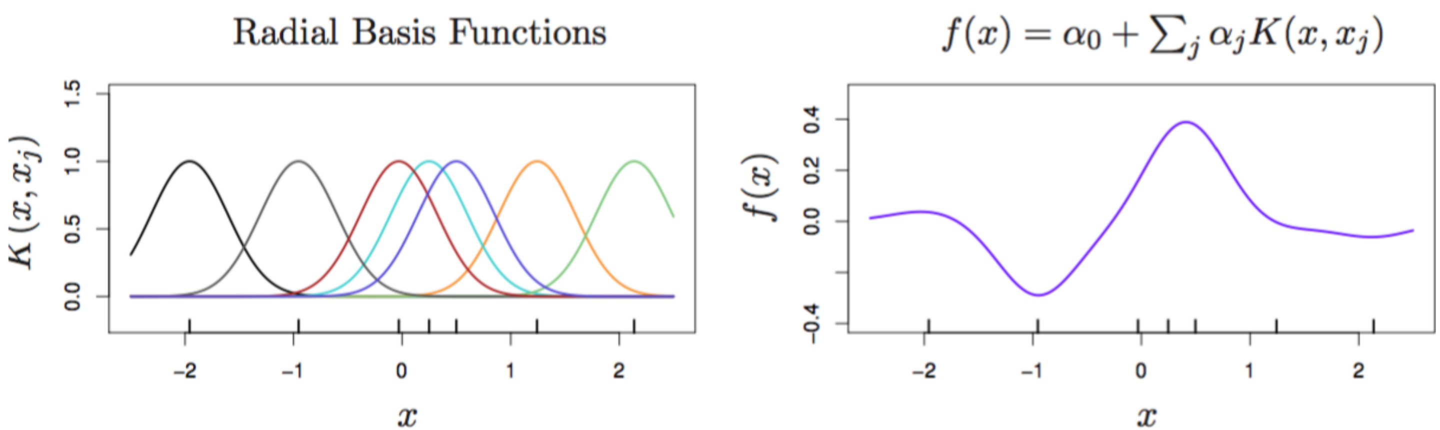
$$\widehat{\alpha} = \mathbf{K}^{-1}\mathbf{y}$$

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

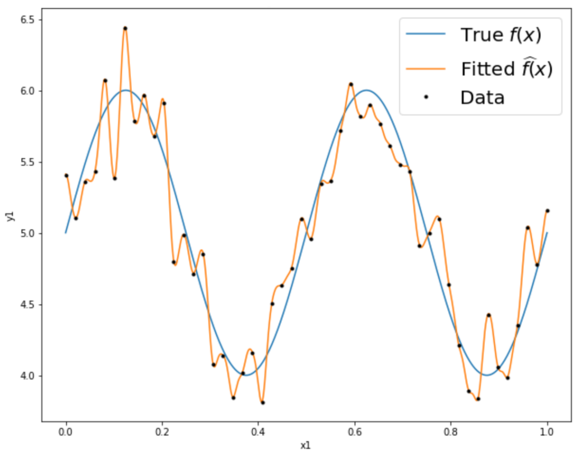


RBF Kernel

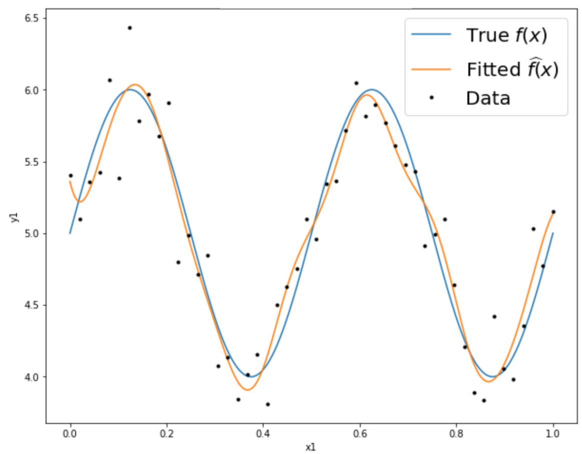
$$K(\mathbf{u},\mathbf{v}) = \exp\left(-\frac{||\mathbf{u}-\mathbf{v}||_2^2}{2\sigma^2}\right)$$

The bandwidth sigma has an enormous effect on fit:

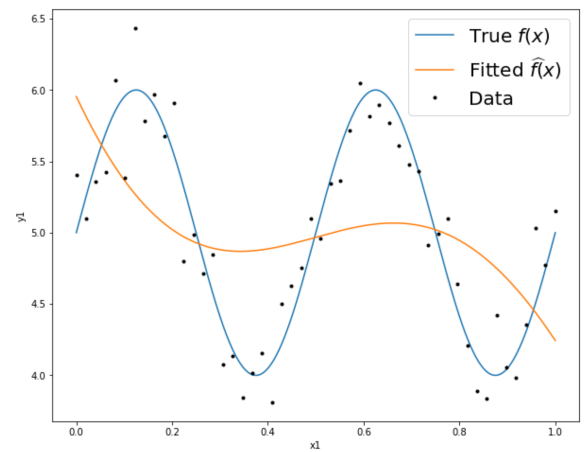
$\sigma = 10^{-2}$ $\lambda = 10^{-4}$



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



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



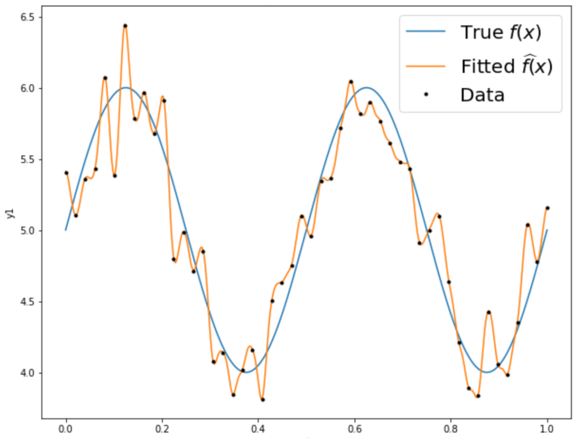
$$\widehat{f}(x) = \sum_{i=1}^{n} \widehat{\alpha}_i K(x_i, x)$$

RBF Kernel

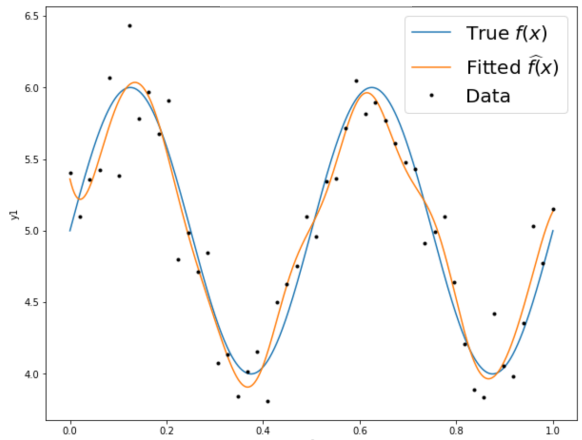
$$K(\mathbf{u},\mathbf{v}) = \exp\left(-\frac{||\mathbf{u}-\mathbf{v}||_2^2}{2\sigma^2}\right)$$

The bandwidth sigma has an enormous effect on fit:

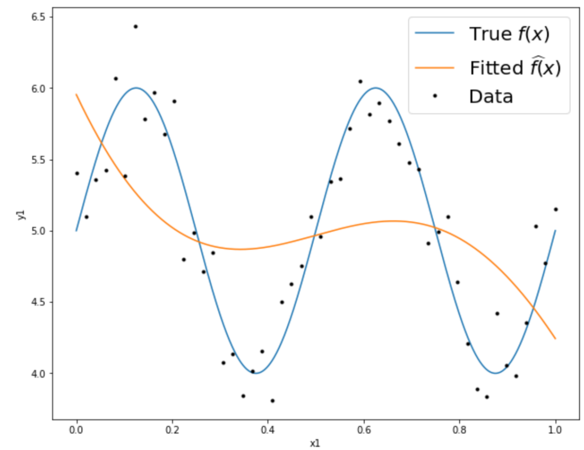
$\sigma = 10^{-2} \quad \lambda = 10^{-4}$



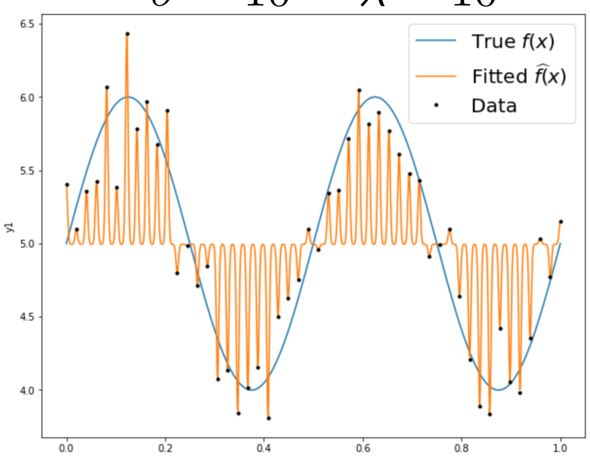
$\sigma = 10^{-1} \quad \lambda = 10^{-4}$



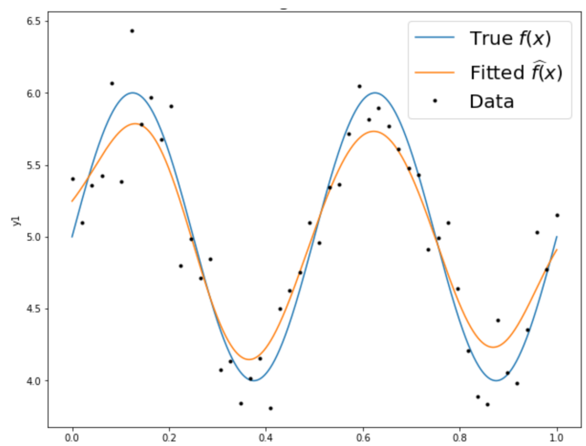
$\sigma = 10^{-0} \quad \lambda = 10^{-4}$



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



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

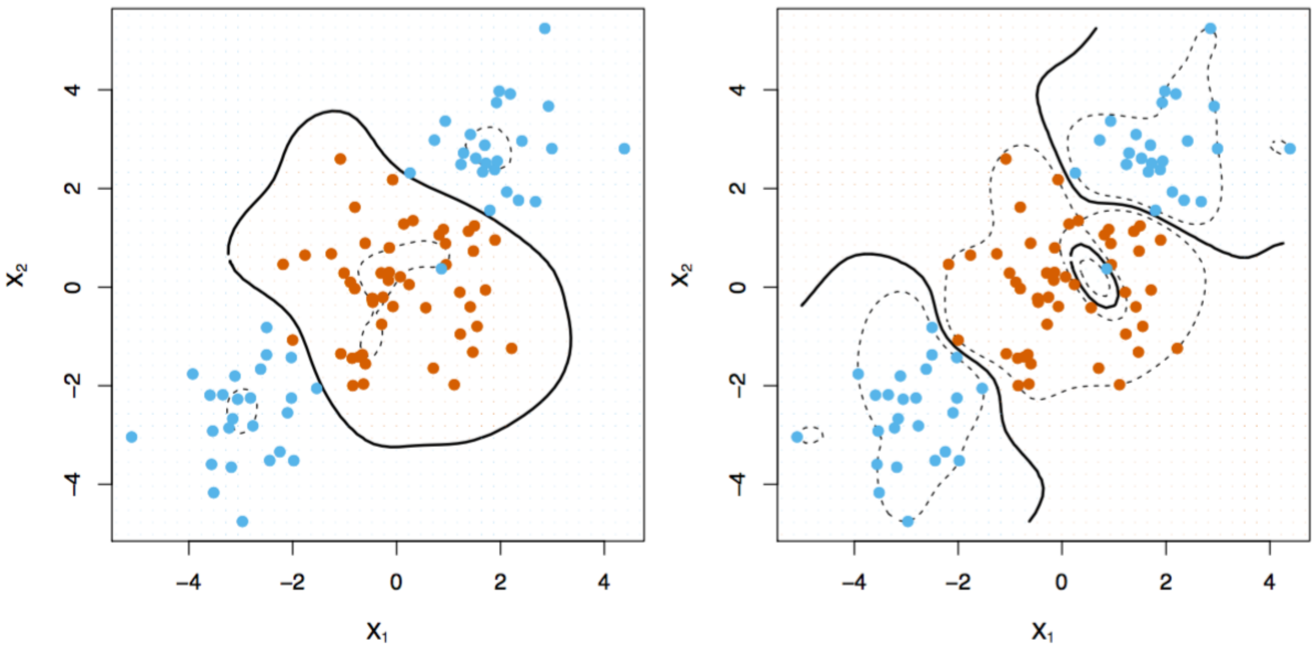


$$\widehat{f}(x) = \sum_{i=1}^{n} \widehat{\alpha}_i K(x_i, x)$$

RBF kernel and random features

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

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



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.

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

IPTSALVKETLALLSTHRTLLIANETLRIPVPVHKNHQLCTEEIFQGIGTLESQTVQGGTV
ERLFKNLSLIKKYIDGQKKKCGEERRRVNQFLDY**LQE**FLGVMNTEWI

x2

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

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

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

Fixed Feature V.S. Learned Feature