Lecture 16,17: Kernels

Recap: Kernels are much more efficient to compute than features

- As illustrating examples, consider polynomial features of degree exactly k
  - $\phi(x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ for $k = 1$ and $d = 2$, then $K(x, x') = x_1 x_1' + x_2 x_2'$
  - $\phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1 x_2 \\ x_2 x_1 \end{bmatrix}$ for $k = 2$ and $d = 2$, then $K(x, x') = (x^T x')^2$
- Note that for a data point x_i, **explicitly** computing the feature $\phi(x_i)$ takes memory/time $p = d^k$
- For a data point x_i, if we can make predictions by only computing the kernel, then computing $\{K(x_i, x_j)\}_{j=1}^n$ takes memory/time dn
  - The features are **implicit** and accessed only via kernels, making it efficient

Examples of popular Kernels

- Polynomials of degree exactly k

$$K(x, x') = (x^T x')^k$$

- Polynomials of degree up to k

$$K(x, x') = (1 + x^T x')^k$$

- Gaussian (squared exponential) kernel
(a.k.a RBF kernel for Radial Basis Function)

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

- Sigmoid

$$K(x, x') = \tanh(\gamma x^T x' + r)$$

- All these kernels are efficient to compute, but the corresponding features are in high-dimensions

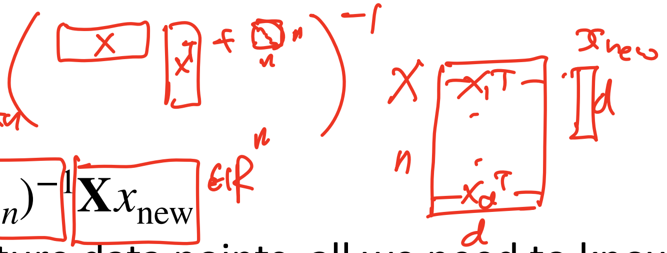
Ridge Linear Regression as Kernels

- Consider Ridge regression: $\hat{w} = \arg\min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_2^2$
- We will represent prediction with $\widehat{w}$ using linear kernel defined as

  $\boxed{K(x, x') = x^T x'}$ (corresponding feature is x itself and hence $d = p$)
- Training: $\widehat{w} = \begin{cases} (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}_{d\times d})^{-1}\mathbf{X}^T\mathbf{y} & \text{(when } n > d\text{)} \\ \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{y} & \text{(when } n \leq d \text{ via linear algebra)} \end{cases}$
- Prediction: $x_{\text{new}} \in \mathbb{R}^d$

$$\hat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}}$$

$$= \mathbf{y}^T \overset{\in\mathbb{R}^{n\times n}}{\boxed{(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}}} \boxed{\mathbf{X}x_{\text{new}}} \in \mathbb{R}^n$$



- Hence, to make prediction on any future data points, all we need to know is

- $$\mathbf{X}x_{\text{new}} = \begin{bmatrix} \overset{\text{vec}}{x_1^T} \overset{\text{vec}}{x_{\text{new}}} \\ \vdots \\ x_n^T x_{\text{new}} \end{bmatrix} = \begin{bmatrix} K(x_1, x_{\text{new}}) \\ \vdots \\ K(x_n, x_{\text{new}}) \end{bmatrix} \in \mathbb{R}^n, \text{ and } \quad \mathbf{X}\mathbf{X}^T = \begin{bmatrix} K(x_1, x_1) & K(x_1, x_2) & \cdots \\ \vdots & \vdots & \\ K(x_n, x_1) & K(x_n, x_2) & \cdots \end{bmatrix} \in \mathbb{R}^{n\times n}$$

- For **ridge regression**, even if we run on feature map $\phi(x) \in \mathbb{R}^p$, we only need to access the features via kernel $K(x_i, x_j)$ and $K(x_i, x_{\text{new}})$ and not the features $\phi(x_i)$

Example: feature vs. kernel

- Ridge regression with feature map $\phi(\cdot) \in \mathbb{R}^p$
  - Solve for $\widehat{w} = \arg\min_{w \in \mathbb{R}^p} \sum_{i=1}^{n} \left(y_i - w^T \phi(x_i)\right)^2 + \lambda \|w\|_2^2$
  - Slow when $p \gg d$

- Ridge regression with kernel $K(\cdot, \cdot)$ corresponding to the feature map $\phi(\cdot)$
  - Finds the optimal solution of the above problem, but
  - only accesses the data via kernel $\{K(x_i, x_j)\}$, which is independent of p and only depends on n, if kernel is efficient to compute
    (which is true for all kernels we looked at and all kernels people use in practice)

The Kernel Trick

- Given data $\{(x_i, y_i)\}_{i=1}^n$, pick a kernel $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

1. For a choice of a loss, use a linear predictor of the form

$$\boxed{\underset{\uparrow \atop \mathbb{R}^d}{\widehat{w}} = \sum_{i=1}^n \underset{=}{\boxed{\overset{\mathbb{R} \atop \downarrow}{\alpha_i}}} \underset{\nwarrow \mathbb{R}^d}{x_i}} \quad \text{for some } \alpha = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} \in \mathbb{R}^n \text{ to be learned}$$

Prediction is $\underline{\widehat{y}_{\text{new}}} = \underline{\widehat{w}^T x_{\text{new}}} = \underline{\sum_{i=1}^n \alpha_i \boxed{x_i^T x_{\text{new}}}}$

2. Design an algorithm that finds α while accessing the data only via $\{x_i^T x_j\}$
3. $\boxed{\text{Substitute } x_i^T x_j \text{ with } K(x_i, x_j)\text{, and find } \alpha}$ using the above algorithm from step 2.
4. Make prediction with $\widehat{y}_{\text{new}} = \sum_{i=1}^n \alpha_i K(x_i, x_{\text{new}})$

   (replacing $x_i^T x_{\text{new}}$ with $K(x_i, x_{\text{new}})$)

The Kernel Trick for regularized least squares

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

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i$ (Step 1. We will prove it later)

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

(Step 2. Write an algorithm in terms of $\widehat{\alpha}$)

$$\underline{\widehat{\alpha}_{\text{kernel}}} = \arg\min_{\alpha} \sum_{i=1}^{n} \Big(y_i - \sum_{j=1}^{n} \alpha_j \boxed{K(x_i, x_j)}\Big)^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \boxed{K(x_i, x_j)}$$

(Step 3. Switch inner product with kernel)

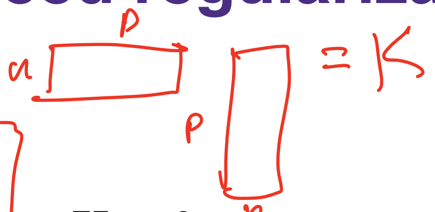
$$= \arg\min_{\alpha} ||\mathbf{y} - \mathbf{K}\alpha||_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

Where $\mathbf{K}_{ij} = K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$

(Solve for $\widehat{\alpha}_{\text{kernel}}$)

Thus, $\widehat{\alpha}_{\text{kernel}} = (\mathbf{K} + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{y}$

Why do we need regularization when using kernels?



n

- Typically, $p \gg$ ~~d~~ and $\mathbf{K} \succ 0$. Why?

why x $\quad x^T K x > 0$

$\mathbb{R}^p$

$K = \Phi \Phi^T$

$$\Phi = \begin{array}{c|c} & p \\ n & \begin{bmatrix} \phi(x_1) \\ \vdots \\ \phi(x_n) \end{bmatrix} \end{array}$$

$K \succ 0$

$K = \begin{bmatrix} K_{11} & K_{12} & \cdots \\ & & \end{bmatrix}$

$K_{ij} = \phi(x_i)^T \phi(x_j)$

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

$\phi(x_i) \in \mathbb{R}^p$

$d=2$

- So $\boxed{\mathbf{K} \text{ is invertible}}$ and $\widehat{\alpha} = (\mathbf{K} + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{y}$ is well defined.
- What if $\lambda = 0$? What goes wrong?

$x_i = [x_1 \;\; x_2]$

$\phi(x_i) = \begin{bmatrix} x_1^2 \\ x_1 \\ x_2^2 \\ x_2 \\ x_1 x_2 \end{bmatrix}$

$p=5$

$$\underset{\alpha}{\arg\min}\, \|\mathbf{y} - \mathbf{K}\alpha\|_2^2$$

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

$\|$

0

$y - K \cdot K^{-1} y = 0.$

The Kernel Trick for SVMs

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

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

(Step 1. We will prove it later)

$$\widehat{\alpha}, \widehat{b} = \arg\min_{\alpha \in \mathbb{R}^n, b} \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i(b + \sum_{j=1}^{n} \alpha_j x_j^T x_i)\} + \lambda \sum_{i=1, j=1}^{n} \alpha_i \alpha_j x_i^T x_j$$

(Step 2. Write an algorithm in terms of $\widehat{\alpha}$)

$$\widehat{\alpha}_{\text{kernel}}, \widehat{b}_{\text{kernel}} = \arg\min_{\alpha \in \mathbb{R}^n, b} \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i(b + \sum_{j=1}^{n} \alpha_j K(x_j, x_i))\} + \lambda \sum_{i=1, j=1}^{n} \alpha_i \alpha_j K(x_i, x_j)$$

(Step 3. Switch inner product with kernel)

$$= \arg\min_{\alpha \in \mathbb{R}^n, b} \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i(b + \mathbf{K}\alpha)\} + \lambda \alpha^T \mathbf{K} \alpha$$

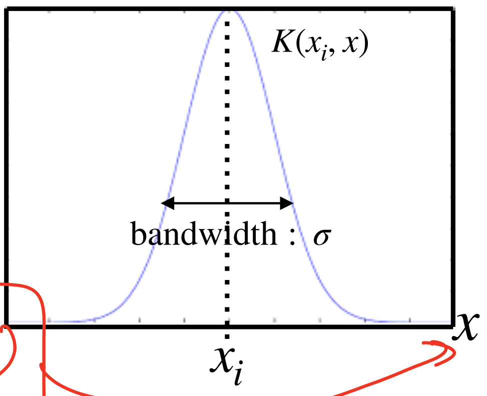
Where $\mathbf{K}_{ij} = K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$

Prediction for x_{new}:

(Solve for $\widehat{\alpha}_{\text{kernel}}, \widehat{b}_{\text{kernel}}$ using optimization)

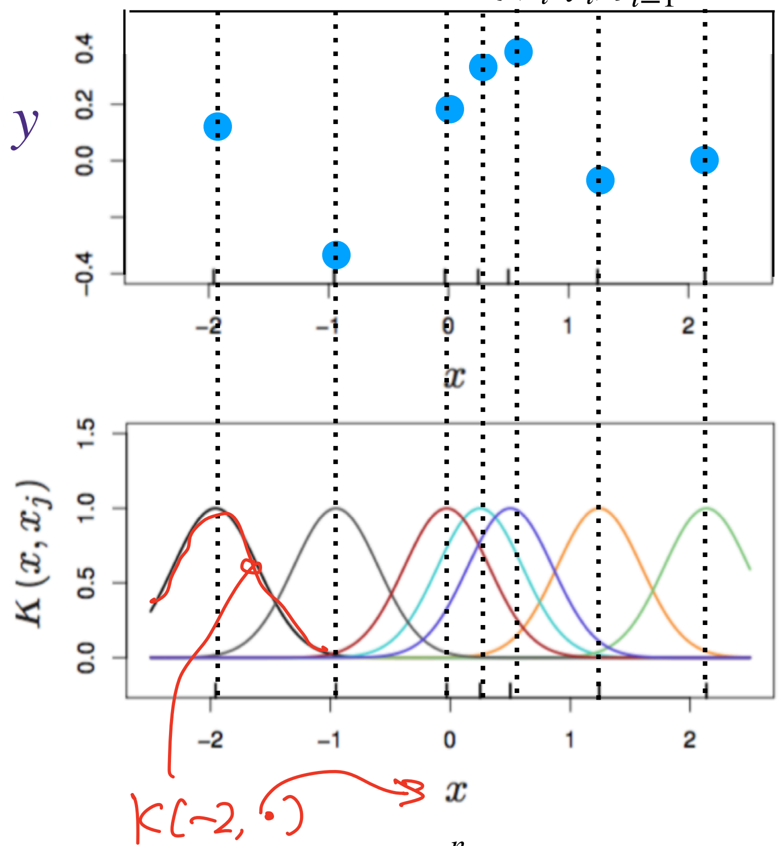
$$\widehat{y} = \text{sign}\left(\sum_{i=1}^{n} \widehat{\alpha}_{\text{kernel},i} K(x_i, x_{\text{new}}) + \widehat{b}_{\text{kernel}} \right)$$

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

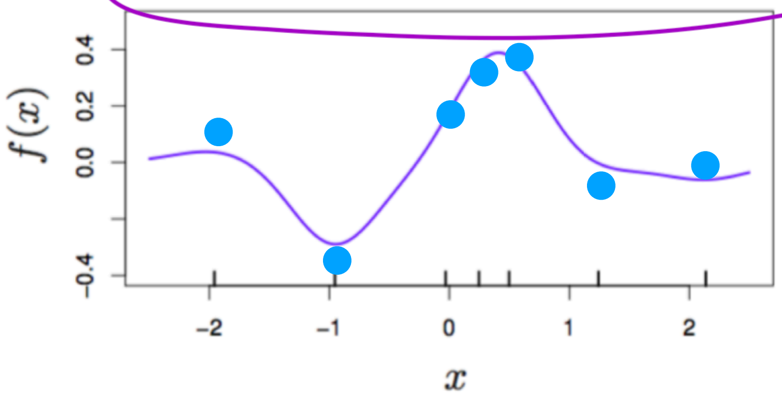


$K(x_i, \bullet)$

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



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

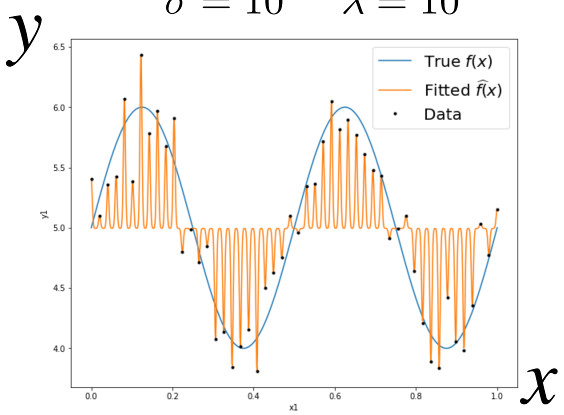


- predictor $f(x) = \sum_{i=1}^{n} \alpha_i K(x_i, x)$ is taking weighted sum of n kernel functions centered at each sample points

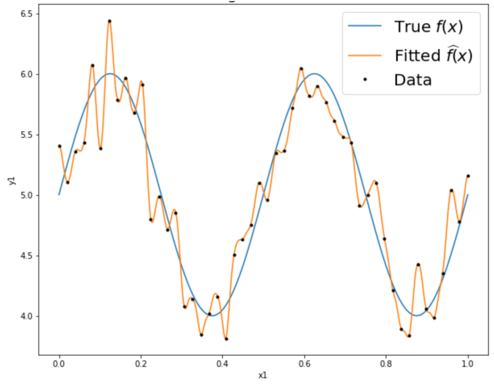
RBF kernel $k(x_i, x) = \exp\left\{ -\frac{\|x_i - x\|_2^2}{2\sigma^2} \right\}$

- $\mathcal{L}(\alpha) = \|\mathbf{K}\alpha - \mathbf{y}\|_2^2 + \lambda\|w\|_2^2$
- The bandwidth σ^2 of the kernel regularizes the predictor, and the regularization coefficient λ also regularizes the predictor

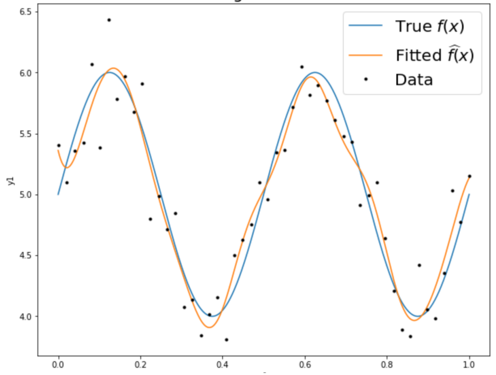
$\sigma = 10^{-3}\ \lambda = 10^{-4}$



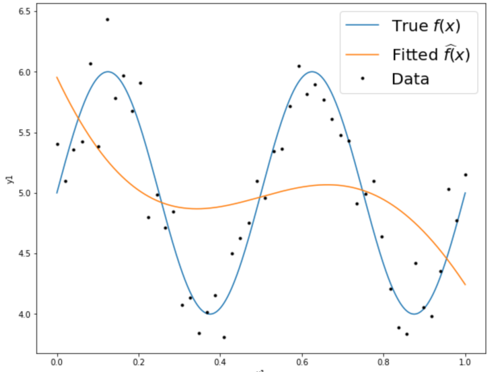
$\sigma = 10^{-2}\ \lambda = 10^{-4}$



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

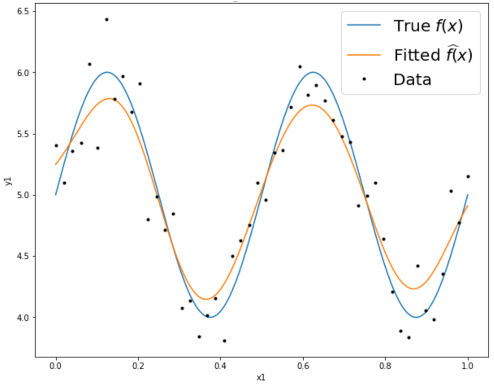


$\sigma = 10^{-0}\ \lambda = 10^{-4}$



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

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

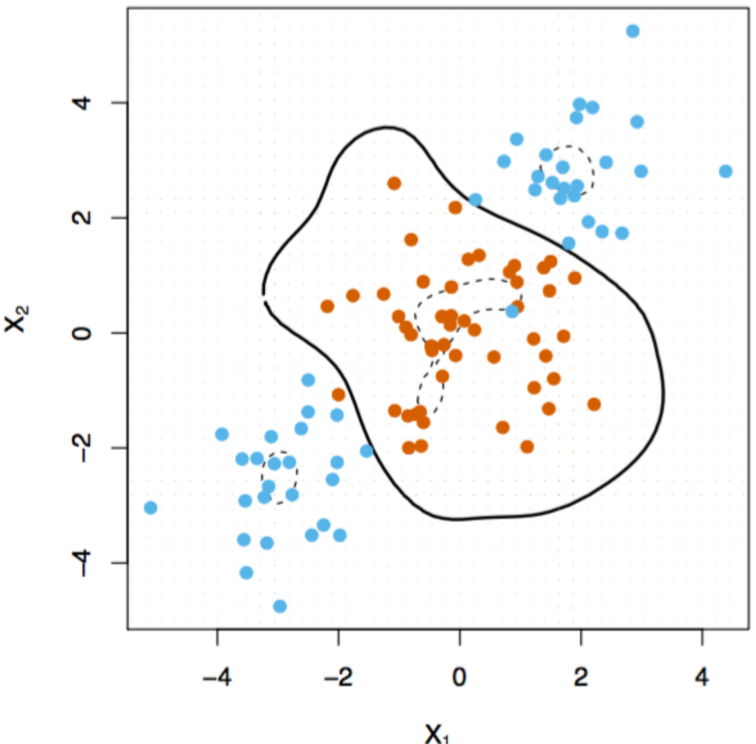


RBF kernel and random features

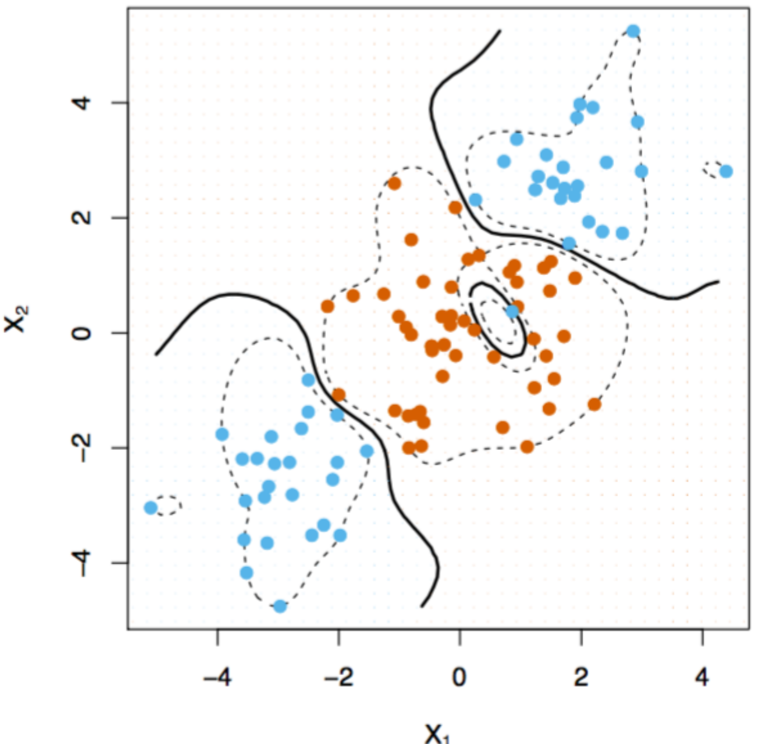
$$\widehat{w} = \arg\min_{w,b} \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i(b + w^T x_i)\} + \lambda \|w\|_2^2$$

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

Bandwidth σ is large enough



Bandwidth σ is small



Features vs. RBF kernel vs. random features

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

$p = \infty >> n >> ? \; \tilde{p}$

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

Instead, consider generating random feature maps of the form:

$$\phi(x) = \begin{bmatrix} \sqrt{2}\cos(w_1^T x + b_1) \\ \vdots \\ \sqrt{2}\cos(w_p^T x + b_p) \end{bmatrix} \qquad \begin{array}{l} w_k \sim \mathcal{N}(0, 2\gamma\, I) \\ b_k \sim \text{uniform}(0,\pi) \end{array}$$

with $\tilde{p} \ll n$

One can show that

$$\mathbb{E}_{w,b}\left[\frac{1}{p}\phi(x)^T\phi(x')\right] = \exp(-\gamma\|x-x'\|_2^2)$$

So this choice of random features approximate the desired RBF kernel with $\gamma = \dfrac{1}{2\sigma^2}$

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

Kernel trick finds the optimal solution for linear models under a feature map $\phi(\,\cdot\,)$

- Once we have chosen to use a feature map $\phi(\,\cdot\,) \in \mathbb{R}^p$, what we want to solve is
  $$\widehat{w} = \arg\min_{w \in \mathbb{R}^p} \sum_{i=1}^{n} \ell\left(y_i, w^T\phi(x_i)\right)$$ for some convex loss $\ell(,)$
- Gradient descent update (from initialization $w^{(0)} = 0$) that find the optimal solution is
  $$w^{(t+1)} = w^{(t)} - \eta \sum_{i=1}^{n} \ell'(y_i, w^T\phi(x_i))\phi(x_i)$$
- One crucial observation is that all $w^{(t)}$'s (including the optimal solution $w^{(\infty)}$) lie on the subspace spanned by $\{\phi(x_1), \ldots, \phi(x_n)\}$, which is an n-dimensional subspace in $\mathbb{R}^p$
- Hence, it is sufficient to look for a solution that is represented as
  $$\widehat{w} = \sum_{i=1}^{n} \alpha_i \phi(x_i)$$ to find the optimal solution
- Kernel trick finds the optimal solution efficiently, by searching over the model that can be represented as $\widehat{w} = \sum_{i=1}^{n} \alpha_i \phi(x_i)$

Fixed Feature V.S. Learned Feature

Can we learn the feature mapping $\phi : \mathbb{R}^d \to \mathbb{R}^p$ from data also?

Questions?

for k-degree poly.

$$K(x, x') = (1+x^T x')^k \; ; \quad \boxed{\mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}}$$

$$\updownarrow$$

$$\phi(x)^T \phi(x') = \begin{bmatrix} x_0 \\ x_0^2 \\ x^3 \\ \vdots \\ x^k \end{bmatrix} \underline{\begin{bmatrix} x' & x'^2 & \cdots & x'^k \end{bmatrix}}$$

Bootstrap

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

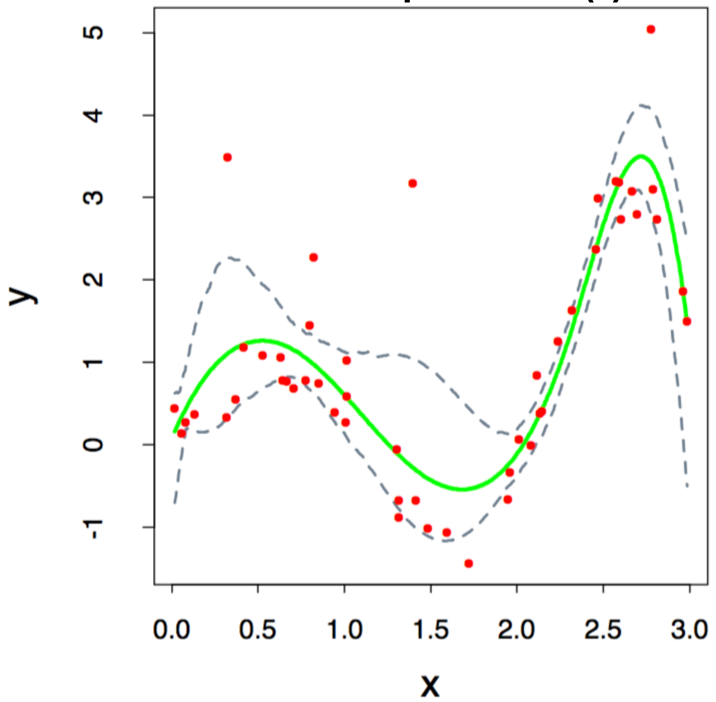
$$f(x) \ = \ \sum_{i=1}^n K(x_i, x) \hat{\alpha}_i,$$

where

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

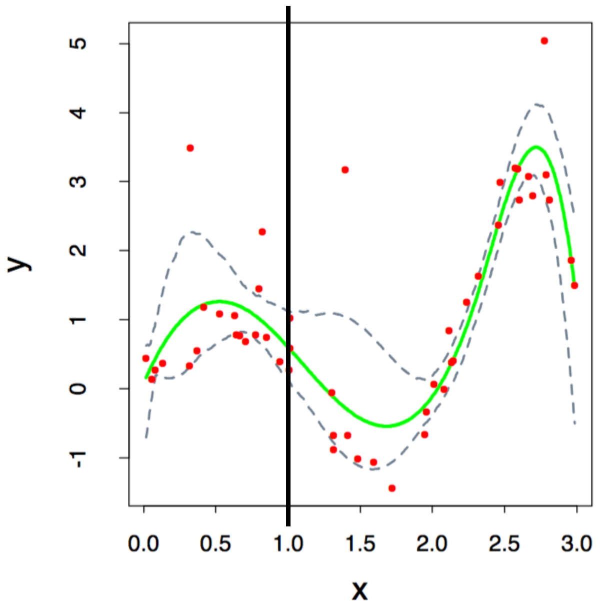
- we wish to build a confidence interval for our predictor $f(x)$, using 5% and 95% percentiles

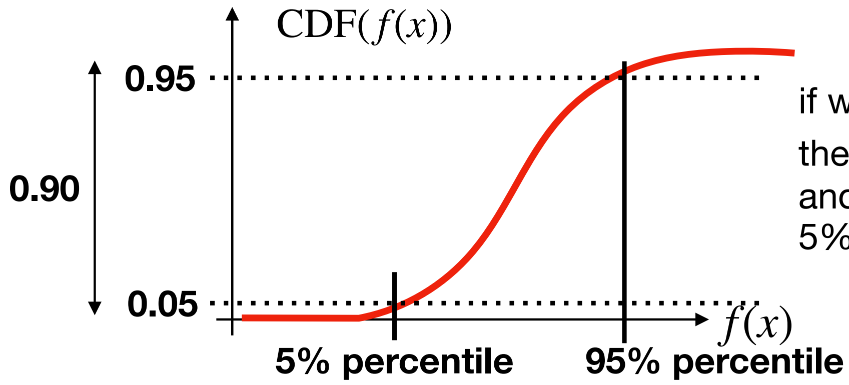
Example of 5% and 95% percentile curves for predictor f(x)



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





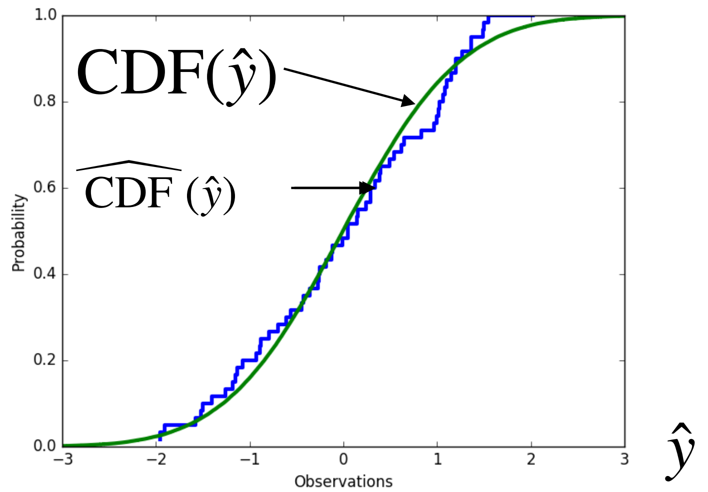
if we know the distribution of our predictor $f(x)$, the green line is the expectation $\mathbb{E}[f(x)]$ and the black dashed lines are the 5% and 95% percentiles in the figure above

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

confidence interval

- 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 $\hat{y}^{(b)} = (\alpha^{*(b)})^T h(x)$



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

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

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

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

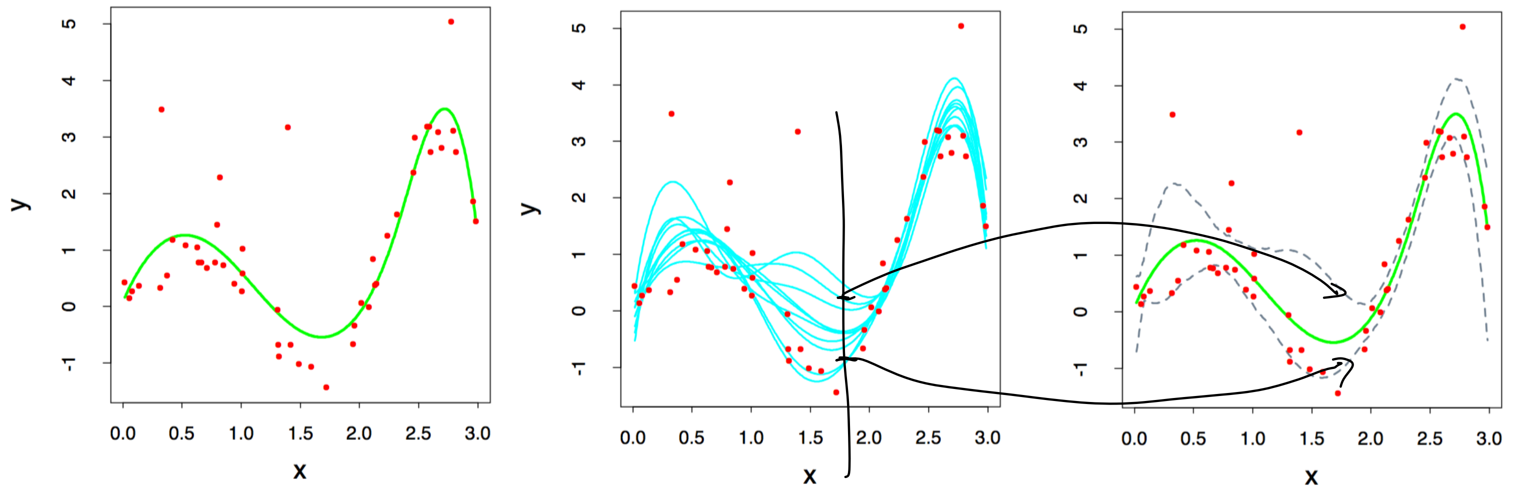
- a **bootstrap dataset** is created from the training dataset by taking n (the same size as the training data) examples uniformly at random **with replacement** from the training data $\{(x_i, y_i)\}_{i=1}^n$

- 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

Questions?