Classification Kernel Methods

Matt Golub
Hunter Schafer

UNIVERSITY of WASHINGTON

Logistic Regression

Recall linear regression:

We assumed that for any $\mathbf{x}$, we have: $p(Y = y \,|\, \mathbf{X} = \mathbf{x}) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(y - \mathbf{w}^T \mathbf{x})^2}$

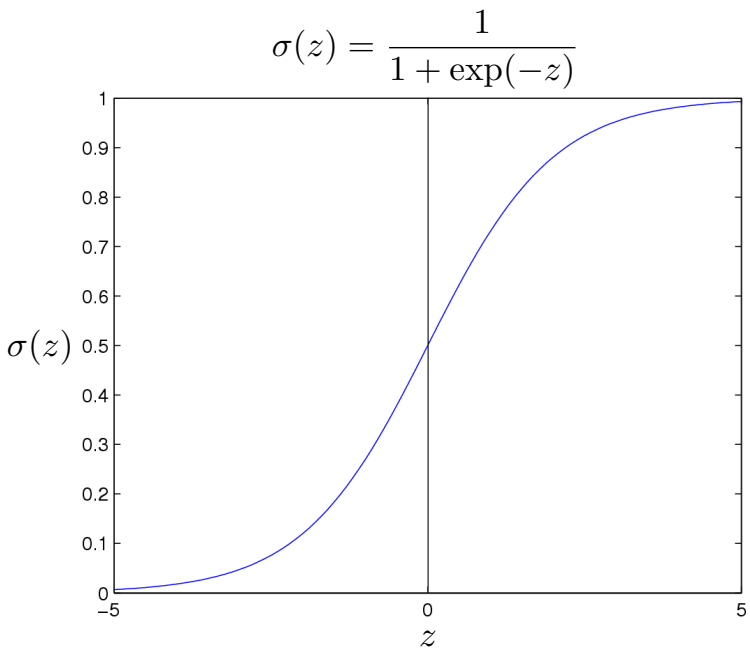
Given data $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ we then computed the MLE for $\mathbf{w}$.

Logistic regression uses a model specialized for classification:

$$\mathbb{P}[Y = 1 | \mathbf{X} = \mathbf{x}, \mathbf{w}] = \sigma(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$

$$\mathbb{P}[Y = 0 | \mathbf{X} = \mathbf{x}, \mathbf{w}] = 1 - \sigma(\mathbf{w}^T \mathbf{x}) = \frac{\exp(-\mathbf{w}^T \mathbf{x})}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$

$$= \frac{1}{1 + \exp(\mathbf{w}^T \mathbf{x})}$$



Logistic Regression

Recall linear regression:

We assumed that for any $\mathbf{x}$, we have: $p(Y = y \,|\, \mathbf{X} = \mathbf{x}) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(y - \mathbf{w}^T \mathbf{x})^2}$

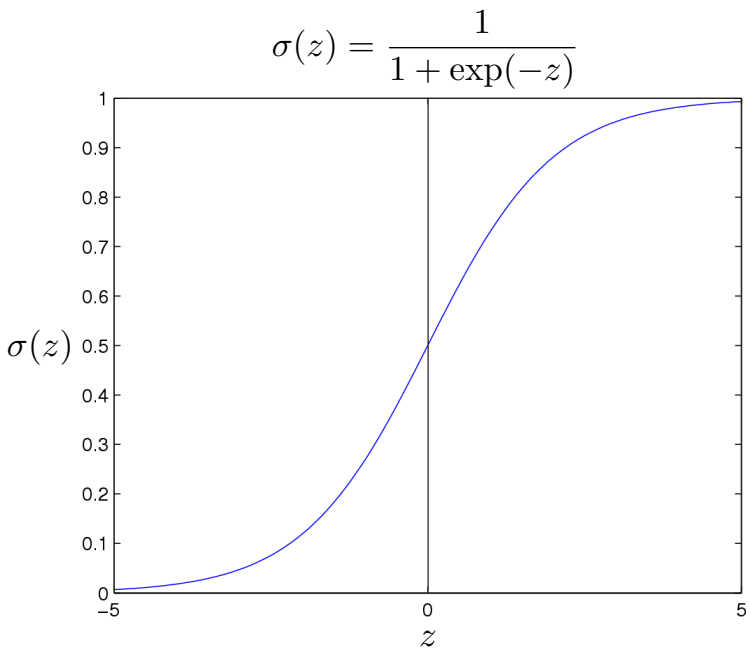
Given data $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ we then computed the MLE for $\mathbf{w}$.

Logistic regression uses a model specialized for classification:

$$\mathbb{P}[Y = 1 | \mathbf{X} = \mathbf{x}, \mathbf{w}] = \sigma(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$

$$\mathbb{P}[Y = 0 | \mathbf{X} = \mathbf{x}, \mathbf{w}] = 1 - \sigma(\mathbf{w}^T \mathbf{x}) = \frac{\exp(-\mathbf{w}^T \mathbf{x})}{1 + \exp(-\mathbf{w}^T \mathbf{x})}$$

$$= \frac{1}{1 + \exp(\mathbf{w}^T \mathbf{x})}$$



Features can be discrete or continuous!

Sigmoid for binary classes

$$\mathbb{P}[Y=1|\mathbf{X}=\mathbf{x}] = \frac{1}{1+\exp(-\mathbf{w}^T\mathbf{x}-w_0)}$$

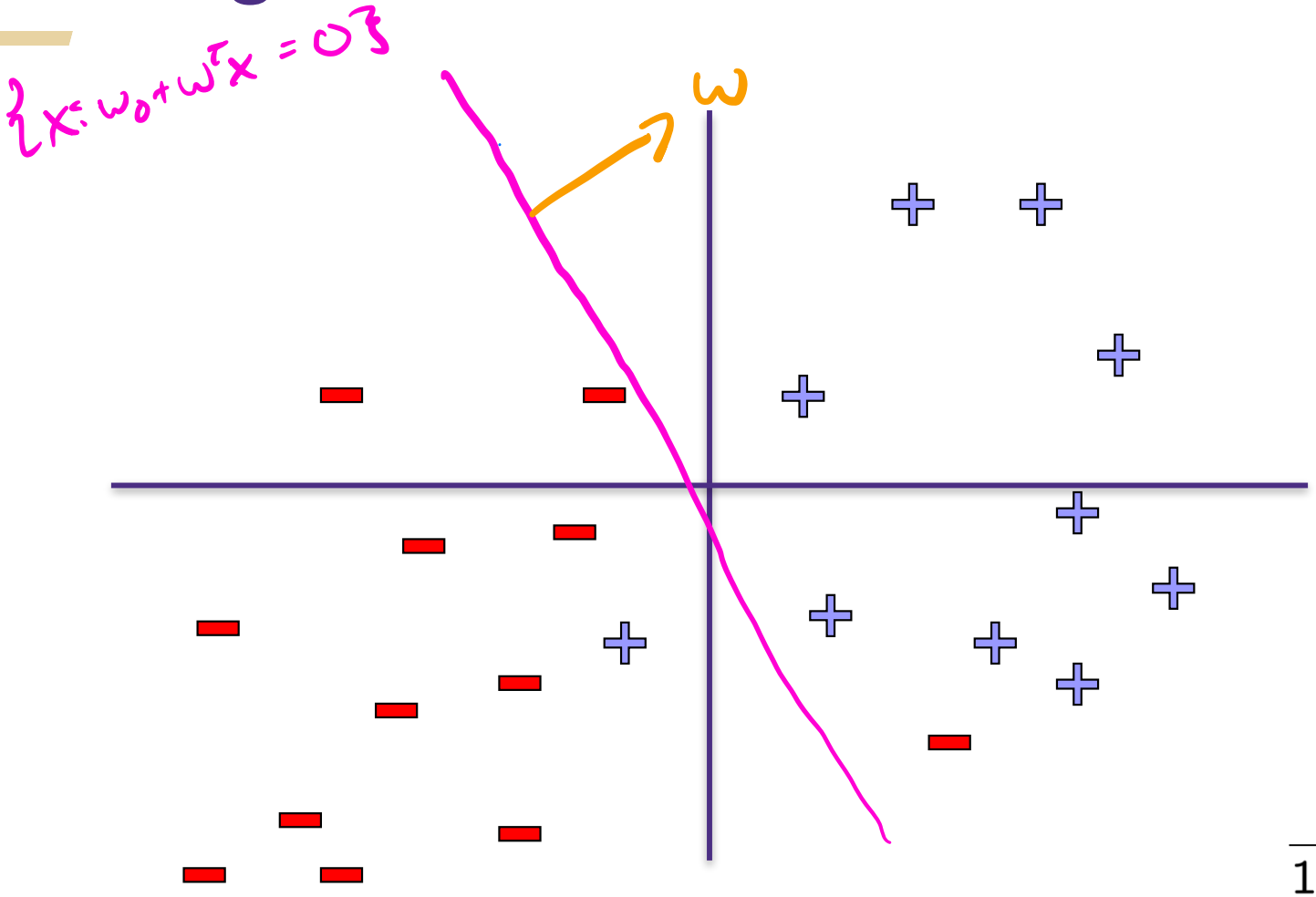
$$\mathbb{P}[Y=0|\mathbf{X}=\mathbf{x}] = \frac{\exp(-\mathbf{w}^T\mathbf{x}-w_0)}{1+\exp(-\mathbf{w}^T\mathbf{x}-w_0)}$$

$$\frac{\mathbb{P}[Y=1|\mathbf{X}=\mathbf{x}]}{\mathbb{P}[Y=0|\mathbf{X}=\mathbf{x}]} = \exp(w_0+\mathbf{w}^T\mathbf{x}) = \exp(w_0+\sum_{k=1}^{d} w_k x_k)$$

Linear Decision Rule!

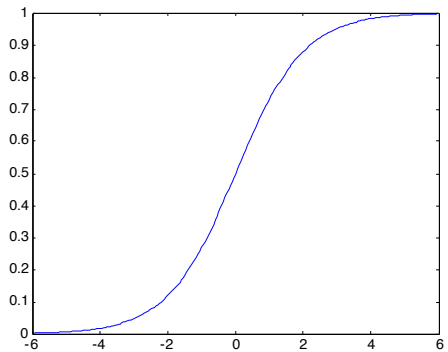
$$\log\frac{\mathbb{P}[Y=1|\mathbf{X}=\mathbf{x}]}{\mathbb{P}[Y=0|\mathbf{X}=\mathbf{x}]} = w_0+\sum_{k=1}^{d} w_k x_k$$

Logistic Regression – A Linear Classifier



$$\log \frac{\mathbb{P}\left[Y=1|\mathbf{X}=\mathbf{x}\right]}{\mathbb{P}\left[Y=0|\mathbf{X}=\mathbf{x}\right]} = w_0 + \sum_{k=1}^{d} w_k x_k$$

$$\frac{1}{1+exp(-z)}$$

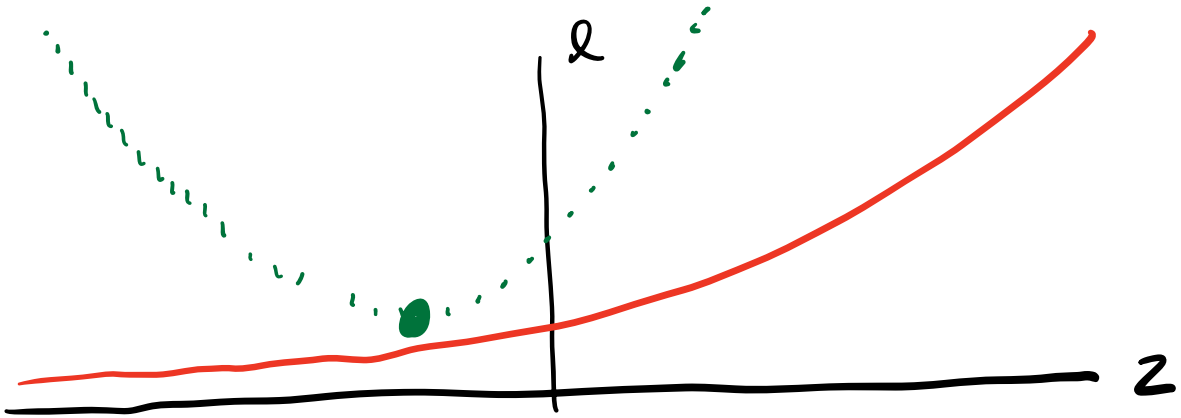


Regularized Conditional Log Likelihood

Add a penalty to avoid high weights/overfitting?:

$$\arg\min_{w,b} \sum_{i=1}^{n} \log\left(1 + \exp(-y_i\left(x_i^T w + b\right))\right) + \lambda||w||_2^2$$

Be sure to not regularize the offset b!



- Logistic regression

2 classes

$$\mathbb{P}(y_i = -1 \,|\, x_i) = \frac{1}{1 + e^{w^T x_i}}$$

$$\mathbb{P}(y_i = +1 \,|\, x_i) = \frac{1}{1 + e^{-w^T x_i}} = \frac{e^{w^T x_i}}{1 + e^{w^T x_i}}$$

k classes

$$\mathbb{P}(y_i = c_1 \,|\, x_i) = \frac{e^{w[:,1]^T x_i}}{e^{w[:,1]^T x_i} + \cdots + e^{w[:,k]^T x_i}}$$

$$\vdots$$

$$\mathbb{P}(y_i = c_k \,|\, x_i) = \frac{e^{w[:,k]^T x_i}}{e^{w[:,1]^T x_i} + \cdots + e^{w[:,k]^T x_i}}$$

Without loss of generality setting w[:,1]=0 when $k = 2$ recovers the original binary class case

Maximum Likelihood Estimator

$$\text{maximize}_w \quad \frac{1}{n} \sum_{i=1}^{n} \log(\mathbb{P}(y_i \,|\, x_i))$$

$$\text{maximize}_{w \in \mathbb{R}^d} \quad \frac{1}{n} \sum_{i=1}^{n} \log\left(\frac{1}{1 + e^{-y_i w^T x_i}}\right)$$

$$\text{maximize}_{W \in \mathbb{R}^{d \times k}} \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} \mathbf{I}\{y_i = c_j\} \log\left(\frac{e^{w[:,j]^T x_i}}{\sum_{j'=1}^{k} e^{w[:,j']^T x_i}}\right)$$

$\mathbf{I}\{y_i = j\}$ is an indicator that is one only if $y_i = j$

Kernels

Creating Features

- Feature mapping $\phi : \mathbb{R}^d \to \mathbb{R}^p$ maps original data into a rich and high-dimensional feature space (usually $d \ll p$)

For example, in d=1, one can use

$$\phi(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_k(x) \end{bmatrix} = \begin{bmatrix} x \\ x^2 \\ \vdots \\ x^k \end{bmatrix}$$

For example, for d>1, one can generate vectors $\{u_j\}_{j=1}^{p} \subset \mathbb{R}^d$

and define features:

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

$$\phi_j(x) = (u_j^T x)^2$$

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

Creating Features

- Feature mapping $\phi : \mathbb{R}^d \to \mathbb{R}^p$ maps original data into a rich and high-dimensional feature space (usually $d \ll p$)

For example, in d=1, one can use

$$\phi(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_k(x) \end{bmatrix} = \begin{bmatrix} x \\ x^2 \\ \vdots \\ x^k \end{bmatrix}$$

For example, for d>1, one can generate vectors $\{u_j\}_{j=1}^p \subset \mathbb{R}^d$

and define features:

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

$$\phi_j(x) = (u_j^T x)^2$$

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

- How many coefficients/parameters are there for degree-k polynomials for $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$?

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

The kernel trick:

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

Big idea: if we can represent our

- training algorithms and
- decision rules for prediction

as functions of dot products of feature maps (i.e. $\{\langle \phi(x), \phi(x') \rangle\}$) and we can find a kernel for our feature map such that

$$K(x \,.\, x') = \langle \phi(x), \phi(x') \rangle$$

then we can avoid explicitly computing and storing (high-dimensional) $\{\phi(x_i)\}_{i=1}^n$ and instead only work with the kernel matrix of the training data $\{K(x_i, x_j)\}_{i,j \in \{1,\ldots,n\}}$

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 $K(x, x') = x_1 x_1' + x_2 x_2' = x^T x'$
  - $\phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1 x_2 \\ x_2 x_1 \end{bmatrix}$ for $k = 2$ and $d = 2$, then $K(x, x') = (x^T x')^2$

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

$\phi(x) =$ p-degree polynomials

$x_i^p + x_i^{p-1} x_j + x_i^{p-2} x_j^2 + x_i^{p-2} x_j x_k$

$d \quad \binom{d}{2} \quad \binom{d}{2} \quad \binom{d}{3} \approx d^p$

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

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

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)

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

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)

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

- 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

- Recall Ridge regression: $\hat{w} = \arg\min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_2^2$
- Consider the trivial kernel $K(x, x') = x^T x'$
- Training: $\widehat{w} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}_{d\times d})^{-1}\mathbf{X}^T\mathbf{y} = \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{y}$

Push through matrix identity

$(AB + cI)^{-1}A = A(BA + cI)^{-1}$

$c \in \mathbb{R}$, $A \in \mathbb{R}^{l\times w}$ $B \in \mathbb{R}^{w\times l}$

Proof

$A(cI + BA) = cAI + A(BA)$

$= cIA + (AB)A$

$= (cI + AB)A$

$c = \lambda$

$A = X^T$

$B = X$

$X \in \mathbb{R}^{n\times d}$

$y \in \mathbb{R}^{n\times 1}$

$(X^TX + \lambda I_d)^{-1}X^Ty$

$= X^T(XX^T + \lambda I_n)^{-1}y$

Ridge Linear Regression as Kernels

- Recall Ridge regression: $\hat{w} = \arg\min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_2^2$
- Consider the trivial kernel $K(x, x') = x^T x'$
- Training: $\widehat{w} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}_{d\times d})^{-1}\mathbf{X}^T\mathbf{y} = \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{y}$
- Prediction: $x_{\text{new}} \in \mathbb{R}^d$ $\quad \hat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}}$

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

Ridge Linear Regression as Kernels

- Recall Ridge regression: $\hat{w} = \arg\min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_2^2$
- Consider the trivial kernel $K(x, x') = x^T x'$
- Training: $\widehat{w} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}_{d\times d})^{-1}\mathbf{X}^T\mathbf{y} = \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{y}$
- Prediction: $x_{\text{new}} \in \mathbb{R}^d$ $\quad \hat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}}$

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

$(XX^T)_{ij} = x_i^T x_j$ $\quad [Xx_{new}]_i = x_i^T x_{new}$

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

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

- **Key idea**: Now consider $\widehat{w} = \arg\min_{w \in \mathbb{R}^p} \sum_{i=1}^{n} \left(y_i - w^T \phi(x_i)\right)^2 + \lambda \|w\|_2^2$ and use an *any* kernel $K(x, x') = \phi(x)^T \phi(x')$!

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

$$\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i \quad \text{for some } \alpha = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} \in \mathbb{R}^n \text{ to be learned}$$

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

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

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

The Kernel Trick for regularized least squares

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

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

(Step 1. We will prove it later)

The Kernel Trick for regularized least squares

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

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

(Step 1. We will prove it later)

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

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

The Kernel Trick for regularized least squares

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

$\phi(x_i)$

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

(Step 1. We will prove it later)

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

$\phi(x_j)^T \phi(x_i)$

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

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

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

$\hat{y} = X\hat{w} = XX^T\alpha = K(K + \lambda I_n)^{-1} y$

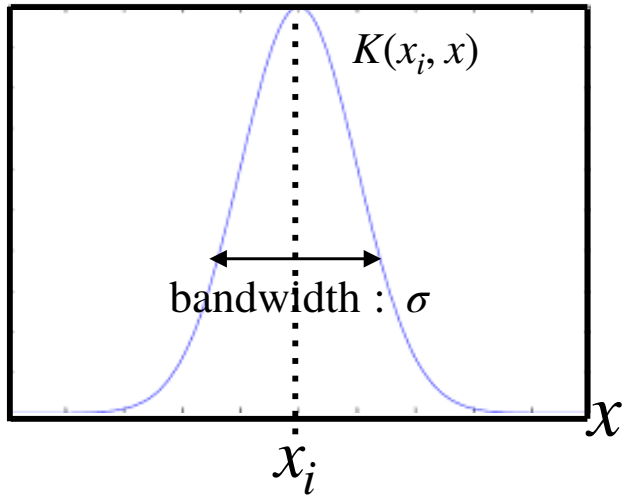
Set $\lambda = 0$ $= K K^{-1} y = y$

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

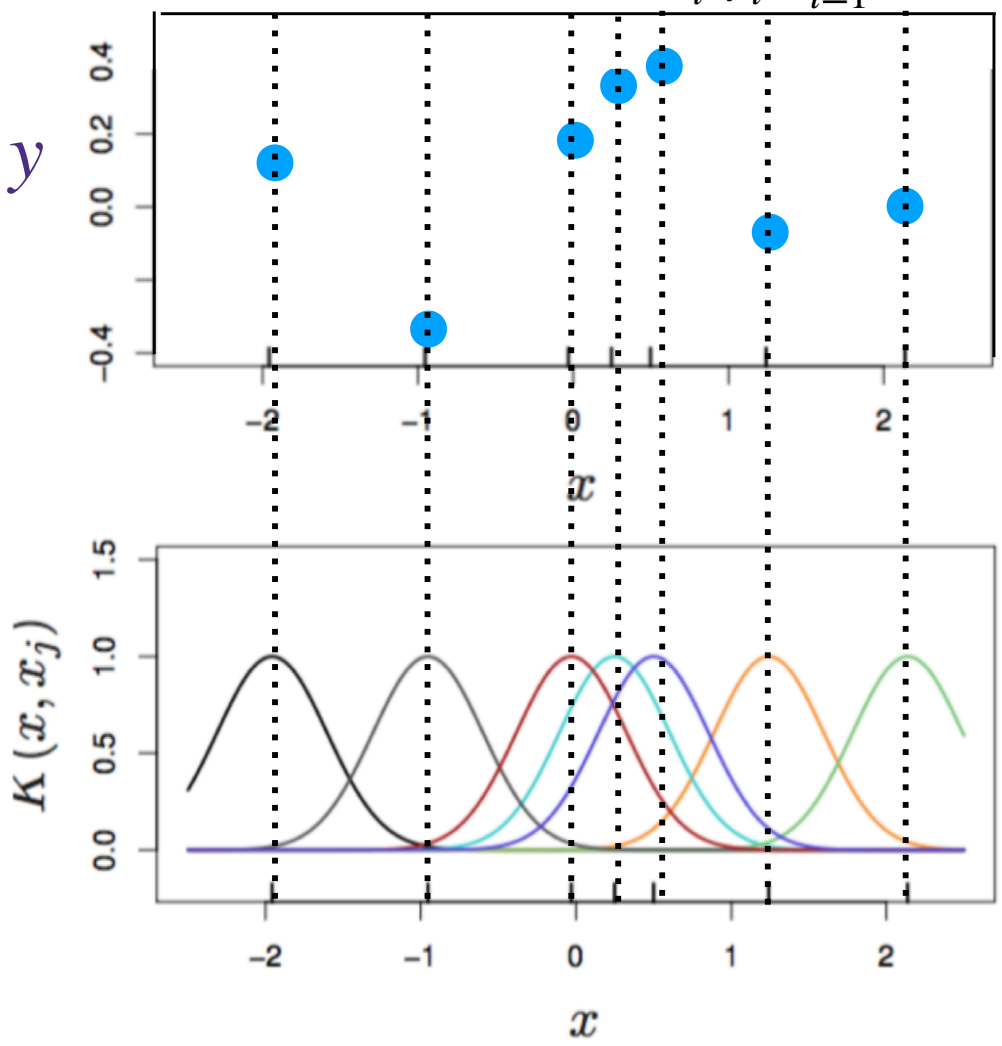
Why do we need regularization when using kernels?

- $\widehat{\alpha} = \arg\min_{\alpha} ||\mathbf{y} - \mathbf{K}\alpha||_2^2 + \lambda \alpha^T \mathbf{K} \alpha$
- Typically, $\mathbf{K}$ is invertible so that $\widehat{\alpha} = (\mathbf{K} + \lambda \mathbf{I}_{n \times n})^{-1} \mathbf{y}$ is well defined.
- What if $\lambda = 0$? What goes wrong?

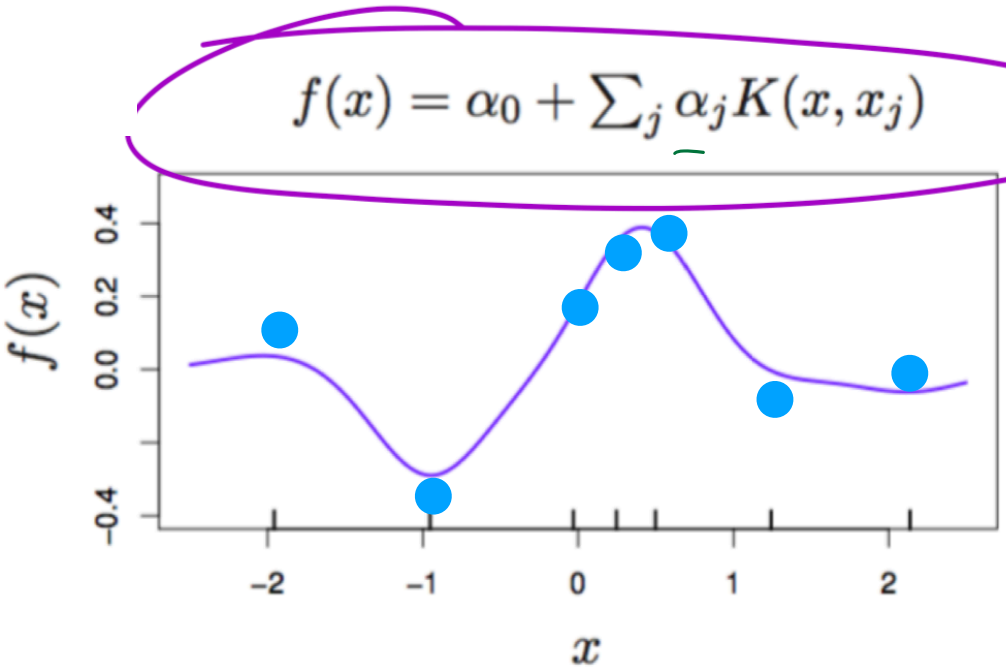
RBF kernel $k(x_i, x) = \exp\left\{ -\frac{\|x_i - x\|_2^2}{2\sigma^2} \right\}$



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



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

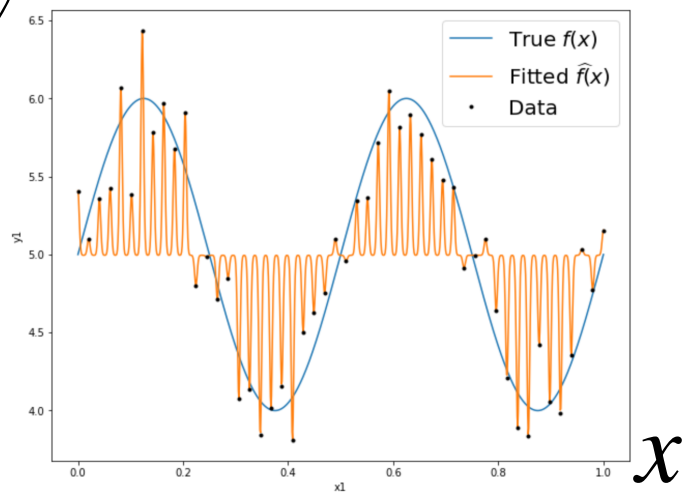


- predictor $f(x) = \sum_{i=1}^{n} \alpha_i K(x_i, x)$ is taking weighted sum of n kernel functions centered at each sample points

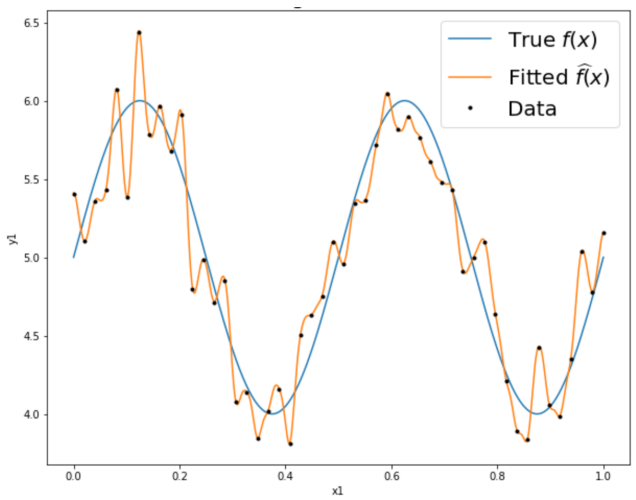
RBF kernel $k(x_i, x) = \exp\left\{ -\frac{\|x_i - x\|_2^2}{2\sigma^2} \right\}$

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

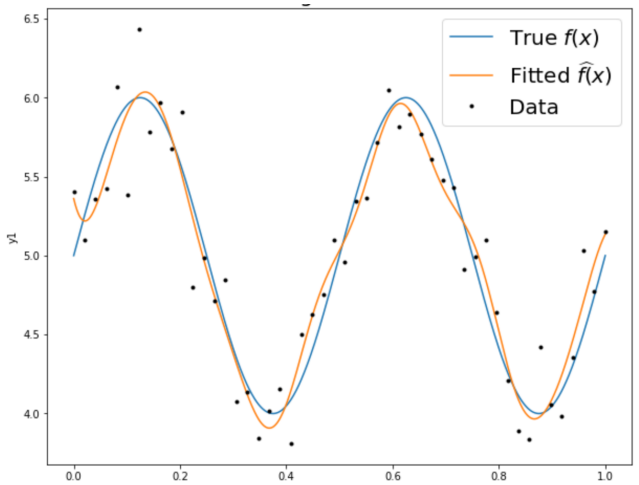
$\sigma = 10^{-3}$ $\lambda = 10^{-4}$



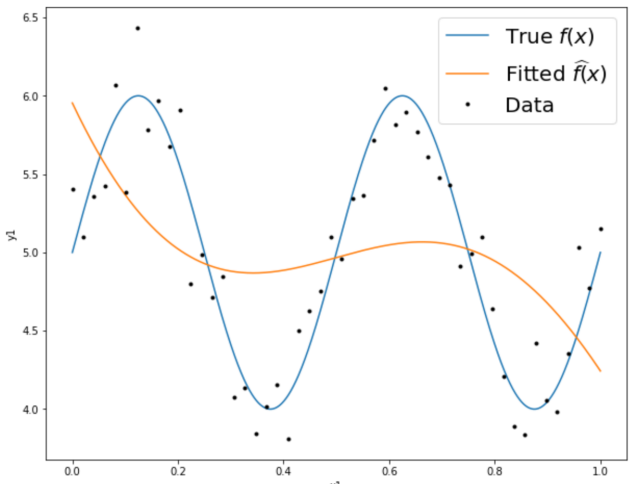
$\sigma = 10^{-2}$ $\lambda = 10^{-4}$



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

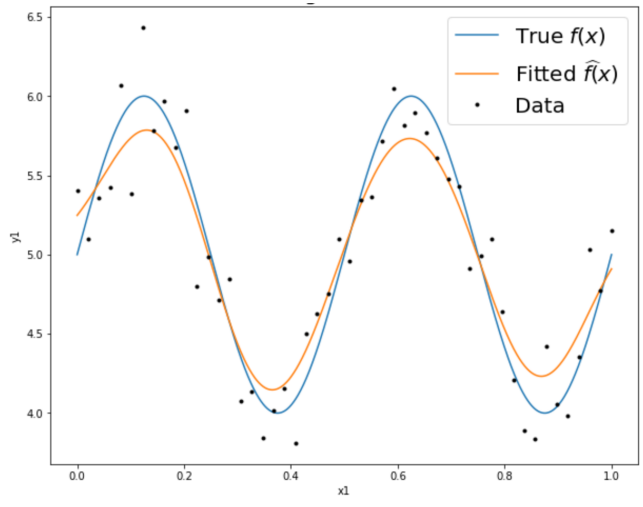


$\sigma = 10^{-0}$ $\lambda = 10^{-4}$



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

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

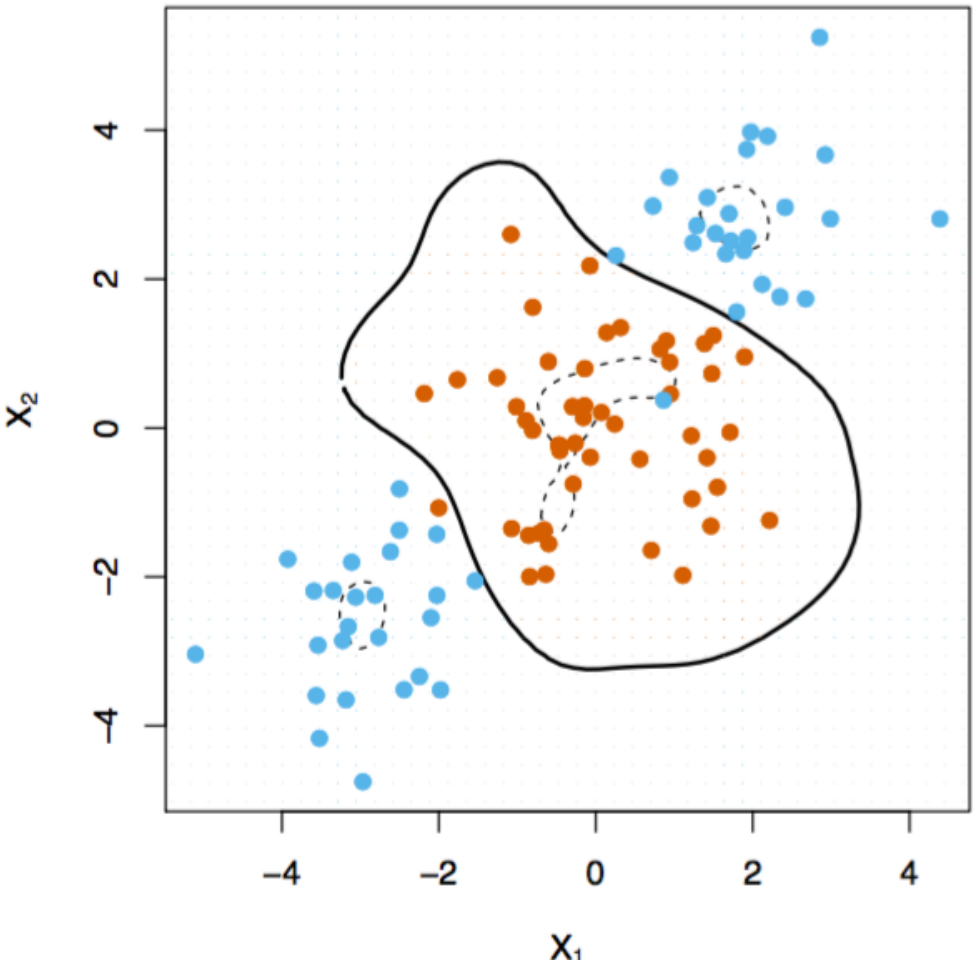


RBF kernel and random features

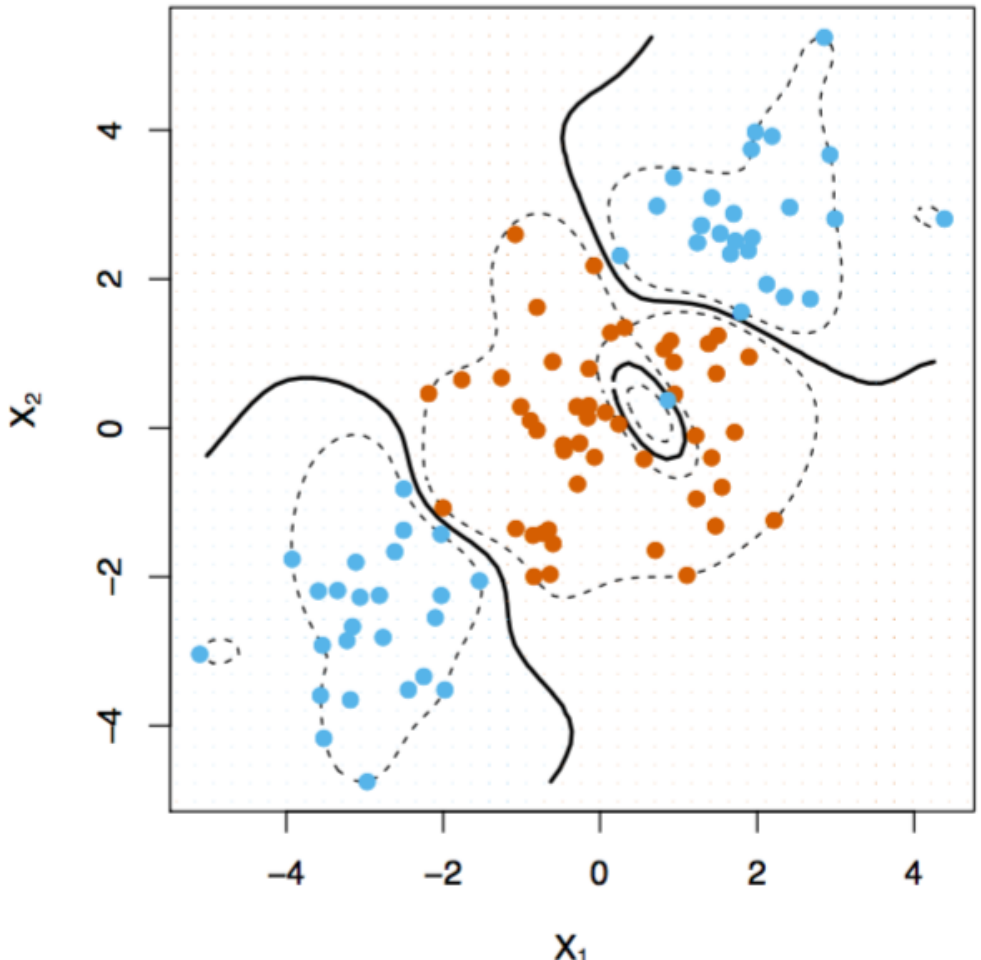
$$\widehat{w} = \arg\min_{w,b} \frac{1}{n}\sum_{i=1}^{n} \log(1+\exp(-y_i(b+w^T x_i))) + \lambda\|w\|_2^2$$

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

Bandwidth σ is large enough



Bandwidth σ is small



Features vs. RBF kernel vs. random features

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

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

Instead, consider generating random feature maps of the form:

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

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

with $p \ll n$

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 $\gamma = \frac{1}{2\sigma^2}$

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

Fixed Feature V.S. Learned Feature

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

Questions?