Kernel methods

Natasha Jaques

UNIVERSITY of WASHINGTON

CSE 446 2025 spring topics overview

https://courses.cs.washington.edu/courses/cse446/25sp/schedule/

- Supervised learning
  - Linear models
    - Linear regression
    - Ridge regression
    - LASSO regression
    - Logistic regression
  - Non-parametric and non-linear methods
    - Nearest neighbors
    - Trees
    - Bootstrap sampling
    - Random forests and boosting
    - **Kernel methods** # We are here!
  - Neural Networks
- Unsupervised learning
  - PCA/SVD
  - Clustering

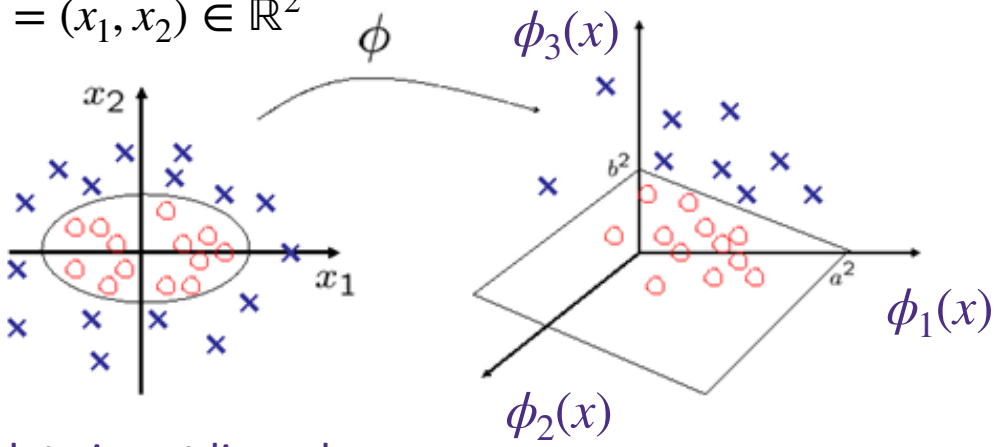
Kernel methods

Lifting to very large dimensional features

What if the data is not linearly separable?

- Use features, for example,

$x = (x_1, x_2) \in \mathbb{R}^2$



$$\frac{\phi_1(x)}{a^2} + \frac{\phi_2(x)}{b^2} \leq 1$$

Our classifier is linear in the transformed feature space

This data is not linearly separable

What function could separate this data?

Circle!

Equation?

$$\frac{x_1^2}{a^2} + \frac{x_2^2}{b^2} \leq 1$$

Can you suggest some features $\phi_1(x_1, x_2), \phi_2(x_1, x_2), \phi_3(x_1, x_2)$ such that this data is linearly separable in this 3-dimensional space?

Feature engineering:

$\phi_1(x_1, x_2) = [x_1^2, x_1, x_2^2]$

So if we want high dimensions to be able to linearly separate our data…. What about ***infinite dimensions!?!***

What if the data is not linearly separable?

- Generally, **high dimensional feature spaces make it easier to linearly separate different classes**
- However, hard to know which feature map will work for given data
- So, **why not just use super high-dimensional features** and hope that the algorithm will automatically pick the right ones?

What could go wrong?

Say we're projecting d features → p features

Unwieldy, computationally expensive

Overfitting

If p > n > d, our data matrix could be rank deficient, not invertible

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:

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

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

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

$$f\left(\begin{bmatrix} u_1^T x \\ u_2^T x \\ \vdots \\ u_p^T x \end{bmatrix} \right)$$

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

A fundamental trick in ML: use kernels

Defn: A function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a ***kernel*** for a map $\phi(\cdot)$ if $K(x, x') \triangleq \phi(x)^T \phi(x') = \phi(x) \cdot \phi(x') = \langle \phi(x), \phi(x') \rangle$

This notation is for dot product (which is the same as inner product)

- Main idea: computing inner products can be much more efficient than explicitly commuting the (very high-dimensional) features
  - Compute kernel K(x,x') not the ϕ
  - Bypass operating on $\phi \in \mathbb{R}^p$

So what might a kernel function look like?

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

$$k(x, x') = \|x - x'\|_2^2$$

What about that smoothed nearest neighbour thing we saw?

$$k(x, x') = e^{-\frac{\|x - x'\|_2^2}{2\sigma^2}}$$

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

A fundamental trick in ML: use kernels

- So, if we can represent our
  - training algorithms and
  - decision rules for prediction
- as functions of dot products of feature maps (i.e. $\{\phi(x) \cdot \phi(x')\}$)
  and if we can find a **kernel** for our feature map such that

$$K(x, x') = \phi(x)^T \phi(x')$$

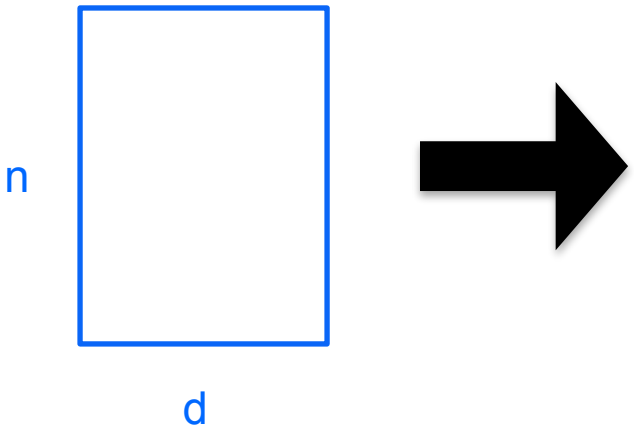
$\phi \in \mathbb{R}^p$

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
$\{K(x_i, x_j)\}_{i,j \in \{1,\ldots,n\}}$

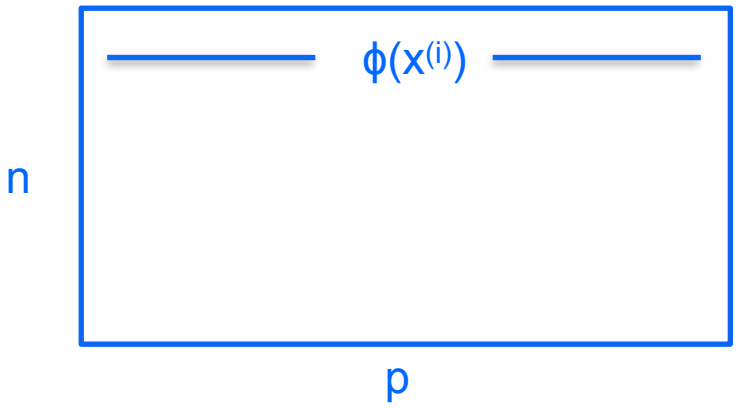
Kernel Matrix vs Feature Space

Notation:
x_1 = feature 1
$x^{(1)}$ = data point 1

Original data matrix X



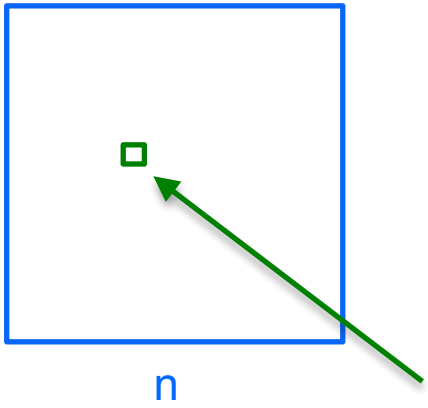
Feature space



Yuck! Not invertible, too computationally $$

K is the kernel matrix!

Kernel space!



$K_{ij} = K(x^{(i)}, x^{(j)})$

Kernels are much more efficient to compute than features

To check this claim, we'll compare $K(x, x')$ to $\phi(x)^T \phi(x')$

Kernel definition: $K(x, x') \triangleq \phi(x)^T \phi(x')$

- 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 $K(x, x') = x_1 x_1' + x_2 x_2'$

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

$$\phi(x)^T \phi(x') = \begin{bmatrix} x_1^2 & x_2^2 & x_1 x_2 & x_2 x_1 \end{bmatrix} \begin{bmatrix} x_1'^2 \\ x_2'^2 \\ x_1 x_2' \\ x_2 x_1' \end{bmatrix} = x_1^2 x_1'^2 + 2 x_1 x_1' x_2 x_2' + x_2^2 x_2'^2$$

ϕ is a 4d vector

$$K(x, x') = (x_1 x_1' + x_2 x_2')^2 = (x^T x')^2$$

K(x,x') is only 2d!

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)

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

ϕ is ∞-d!

- Sigmoid

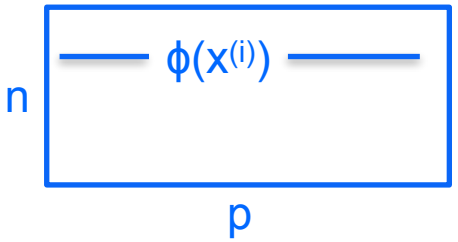
$$K(x, x') = \tanh(\gamma x^T x' + r)$$

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

Example: feature vs. kernel

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

- For now, suppose we are solving this using gradient descent with
  - $w_0 = 0$ and
  - $w_{t+1} \leftarrow w_t + \eta \sum_{i=1}^{n} \phi(x_i)\left(y_i - w_t^T \phi(x_i)\right) - \eta\lambda w_t$ $\in \mathbb{R}^p$



- Claim: For any t, w_t can be represented as $w_t = \sum_{i=1}^{n} \alpha_i \phi(x_i)$
  for some n-dimensional parameter $\alpha = (\alpha_1, \ldots, \alpha_n)$
- Prediction $\hat{y}_{\text{new}} = \hat{w}^T \phi(x_{\text{new}}) = \sum_{i=1}^{n} \alpha_i \phi(x_i)^T \phi(x_{\text{new}})$

Key idea: with n points, can only span an n-dimensional subspace of p (because it's a combo of n basis vectors)

parameter $\hat{w} \in \mathbb{R}^p \rightarrow \hat{\alpha} \in \mathbb{R}^n$ $\hat{\alpha} K(x_i, x_{\text{new}})$

Kernel ridge regression

- Ridge regression with feature map $\phi(\cdot) \in \mathbb{R}^p$

$$\widehat{w} = \arg\min_{w \in \mathbb{R}^p} \frac{1}{2}\|y - \phi(X)w\|_2^2 + \lambda \frac{1}{2}\|w\|_2^2$$

- $$w = \sum_{i=1}^{n} \alpha_i \phi(x_i) = \phi(X)^T \alpha$$

  with now an n-dimensional parameter $\alpha = (\alpha_1, \ldots, \alpha_n)$, instead of p

- Let $K \in \mathbb{R}^{n \times n}$ be the kernel matrix, where $K_{i,j} = \phi(x_i)^T \phi(x_j)$
- The new objective is

L2 reg but weights are alphas on phis

$$\frac{1}{2}\left\|y - \phi(x)\phi(x)^T\alpha\right\|_2^2 + \lambda \frac{1}{2}\alpha^T \phi(x)\phi(x)^T \alpha$$

$$= \frac{1}{2}\left\|y - K\alpha\right\|_2^2 + \frac{1}{2}\lambda \alpha^T K \alpha$$

→ take derivative, set to zero →

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

- Prediction $\hat{y}_{\text{new}} = \hat{w}^T \phi(x_{\text{new}}) = \sum_{i=1}^{n} \alpha_i \phi(x_i)^T \phi(x_{\text{new}}) = \sum_{i=1}^{n} K(x_i, x_{\text{new}})\alpha$

Kernel regression

- This holds for more general class of algorithms other than GD:

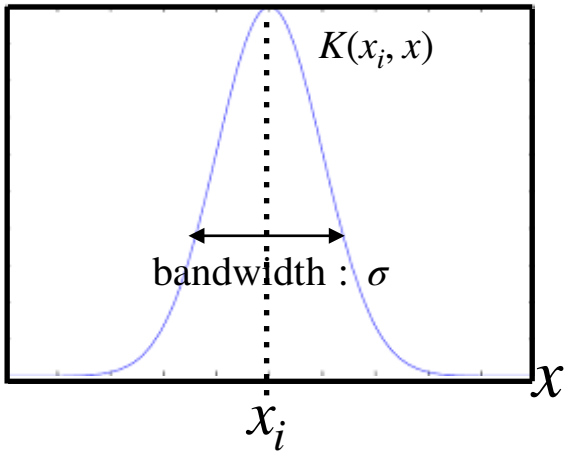
$$w = \sum_{i=1}^{n} \alpha_i \phi(x_i) \;=\; \phi(X)^T \alpha$$

with now an n-dimensional parameter $\alpha = (\alpha_1, \ldots, \alpha_n)$, instead of p

- Kernel method is not limited to linear Ridge regression, but also applies to a broad class of methods including the kernel logistic regression

RBF kernel

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

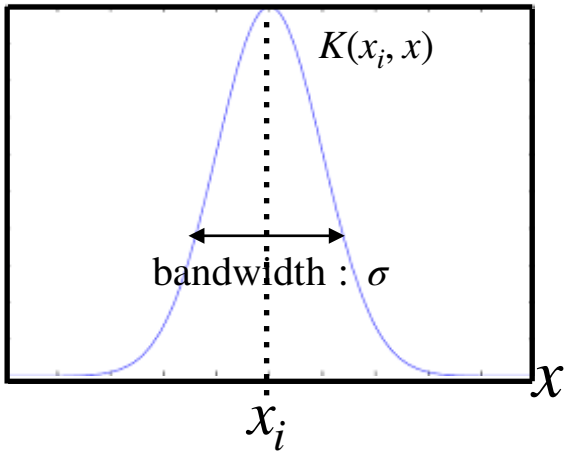


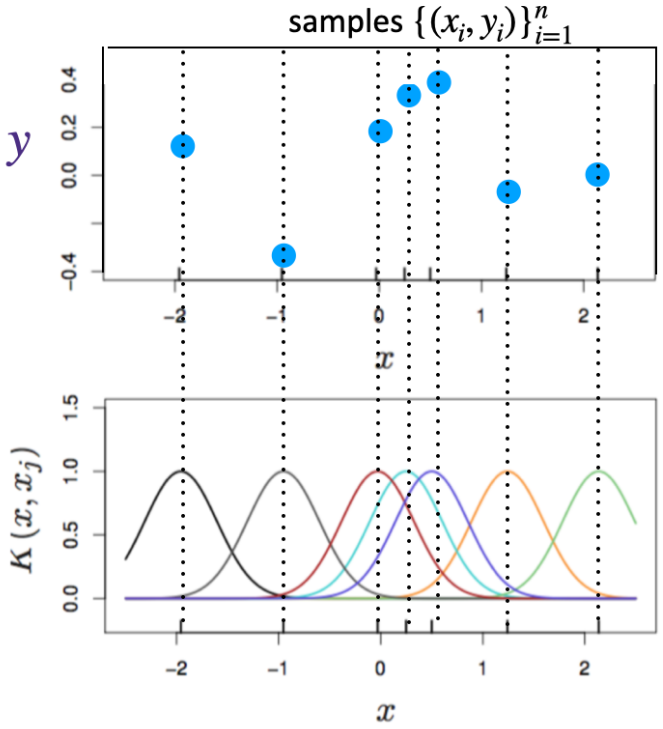
Remember that Gaussian smoothed nearest neighbour?

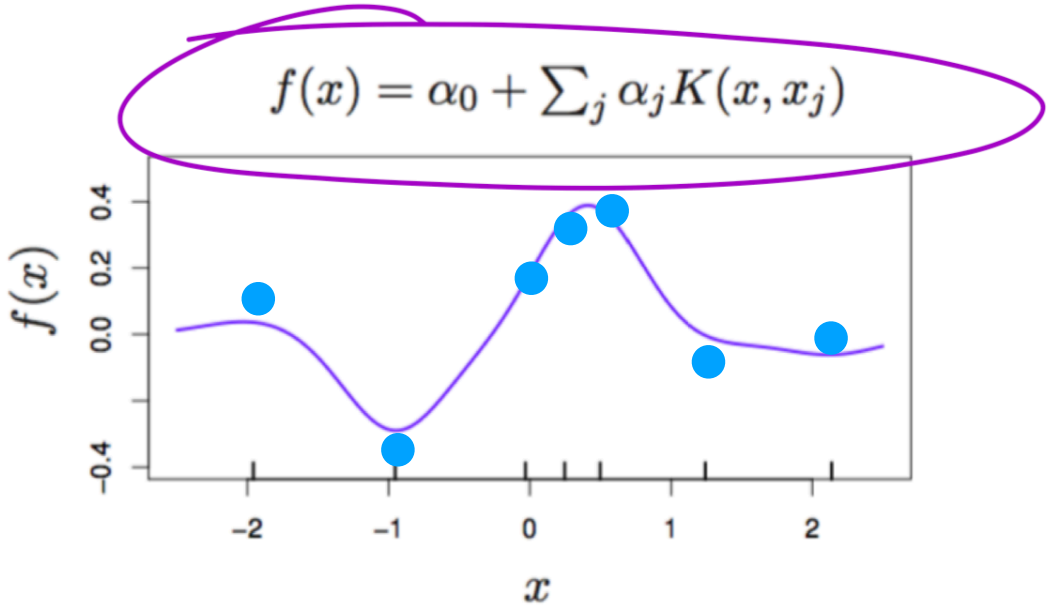
Define a Gaussian at one data point and compute the distance to it

RBF kernel

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

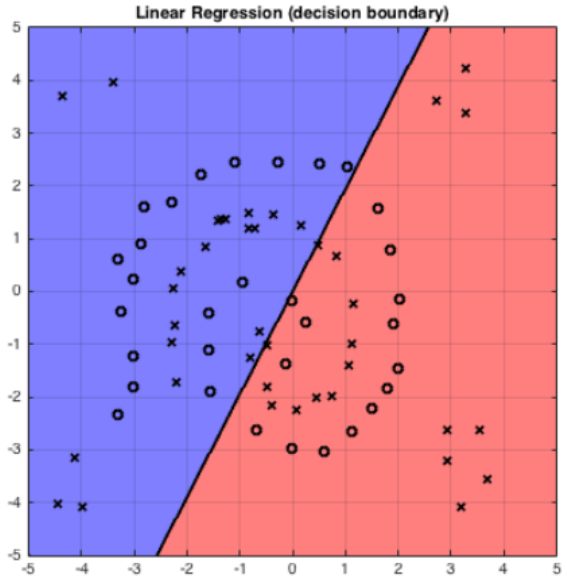




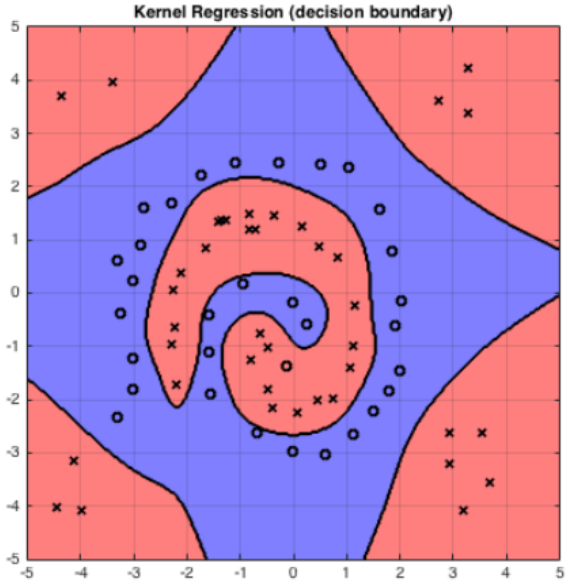


- predictor $f(x) = \sum_{i=1}^{n} \alpha_i K(x_i, x)$ is taking weighted sum of n kernel functions centered at each sample points

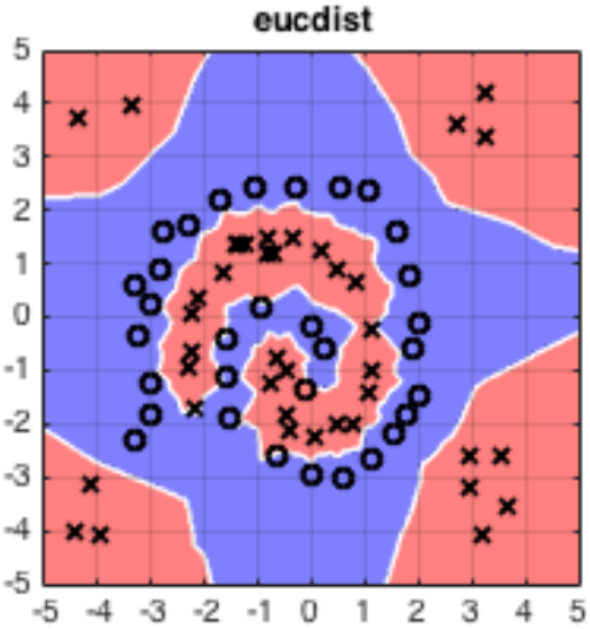
How do the decision boundaries compare?



Linear model



With RBF kernel



Nearest Neighbours
With L2 (less smooth)

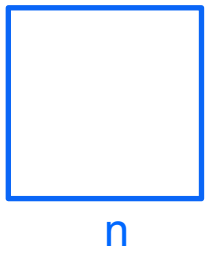
From https://www.cse.wustl.edu/~m.neumann/sp2016/cse517/lecturenotes/lecturenote13.html

Why do we need regularization when using kernels?

PSD

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

$$K(x, x') \triangleq \phi(x)^T \phi(x')$$



Squared, non-negative

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

$$\arg\min_{\alpha} \|\mathbf{y} - \mathbf{K}\alpha\|_2^2$$

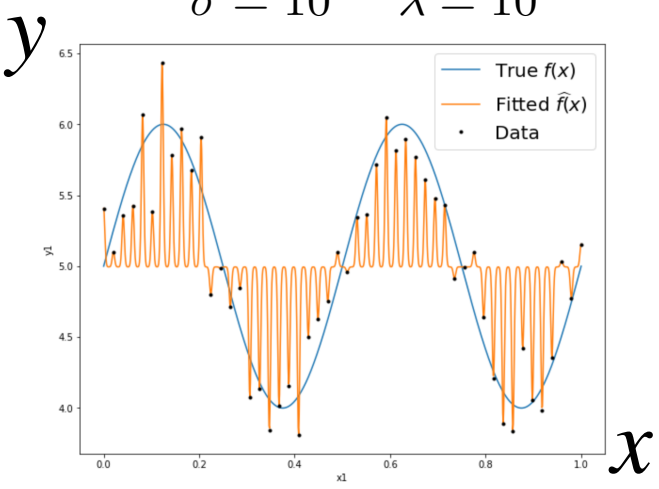
How many params are you fitting? $\alpha \in \mathbb{R}^n$

So fitting n params with n training data points. Can exactly fit each data point, and overfit.

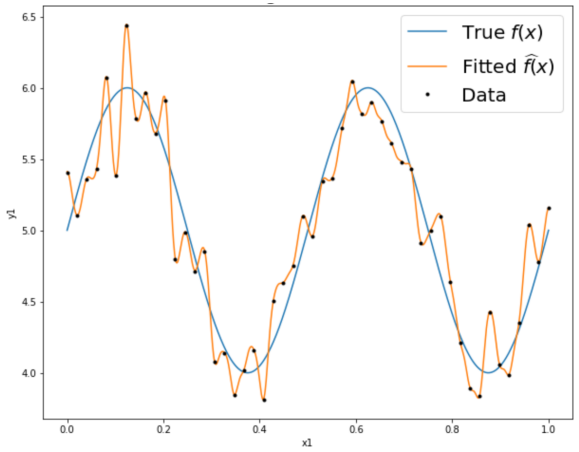
RBF kernel $k(x_i, x) = \exp\left\{ -\frac{\|x_i - x\|_2^2}{2\sigma^2} \right\}$

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

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



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

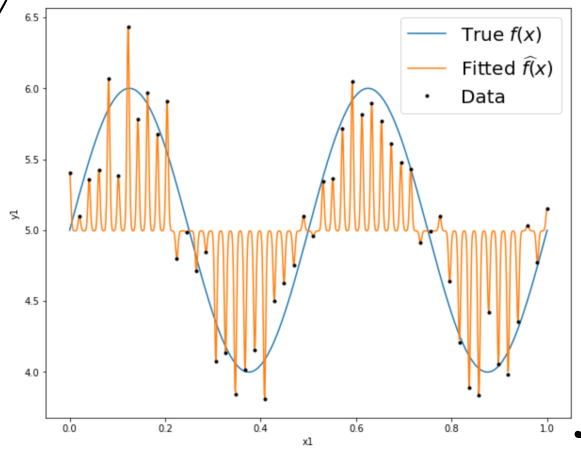


$$\widehat{f}(x) = \sum_{i=1}^{n} \widehat{\alpha}_i K(x_i, x)$$

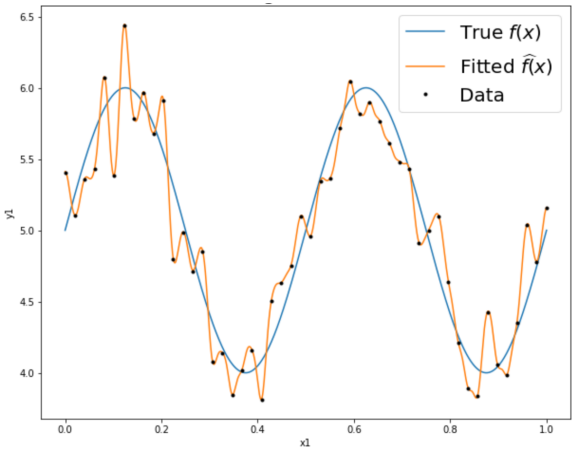
RBF kernel $k(x_i, x) = \exp\left\{ -\frac{\|x_i - x\|_2^2}{2\sigma^2} \right\}$

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

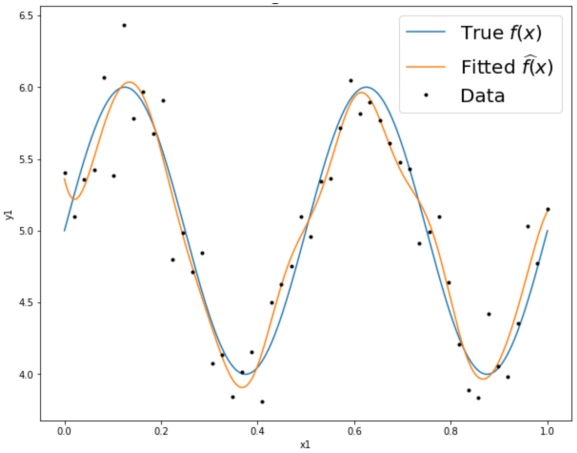
$\sigma = 10^{-3}$ $\lambda = 10^{-4}$



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

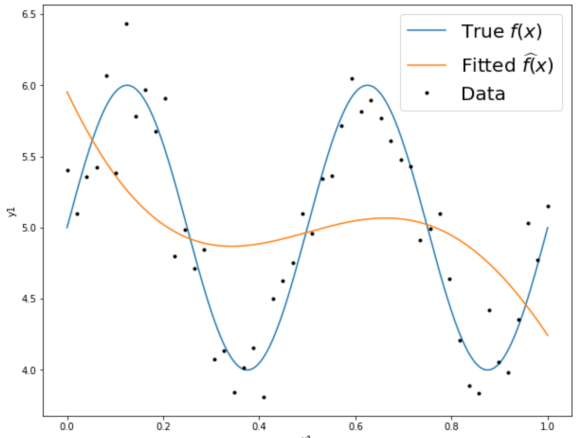


$\sigma = 10^{-1}$ $\lambda = 10^{-4}$



y x

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

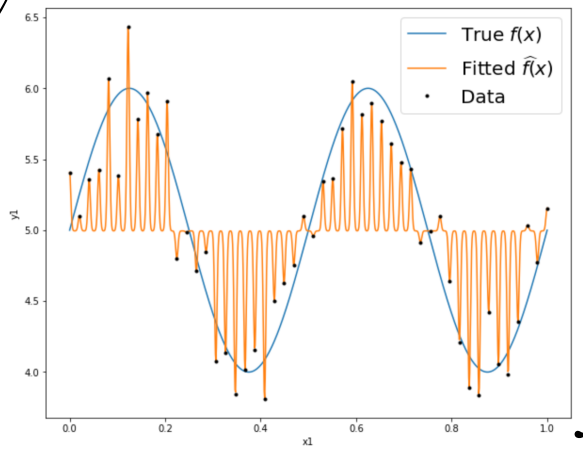


$$\widehat{f}(x) = \sum_{i=1}^{n} \widehat{\alpha}_i K(x_i, x)$$

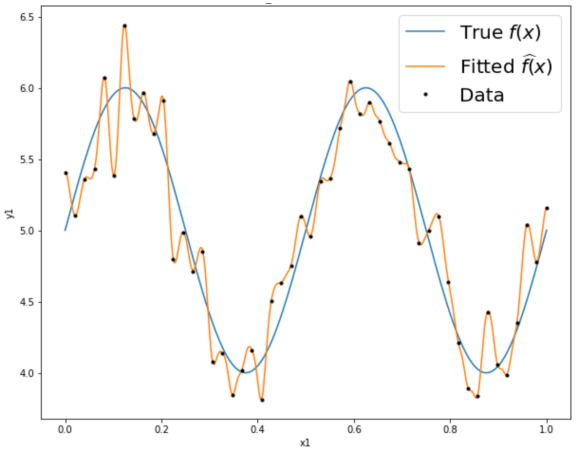
RBF kernel $k(x_i, x) = \exp\left\{ -\frac{\|x_i - x\|_2^2}{2\sigma^2} \right\}$

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

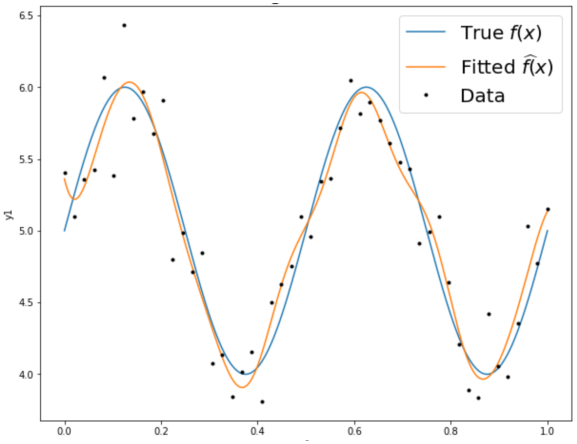
$\sigma = 10^{-3}$ $\lambda = 10^{-4}$



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



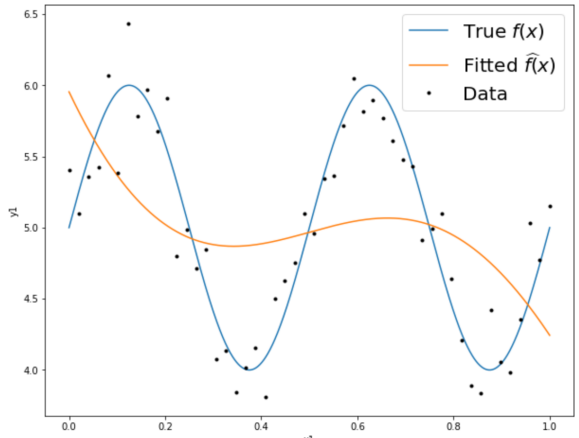
$\sigma = 10^{-1}$ $\lambda = 10^{-4}$



y

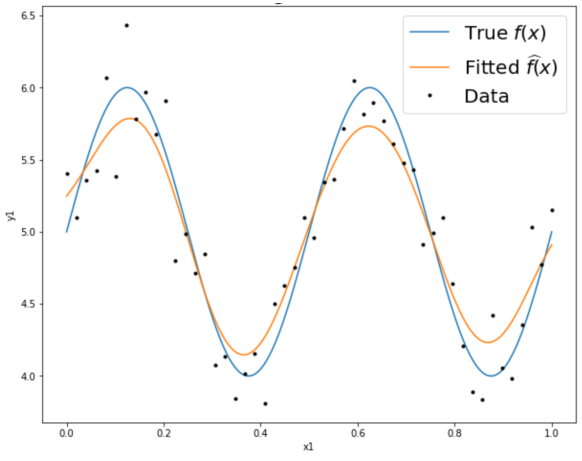
x

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



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

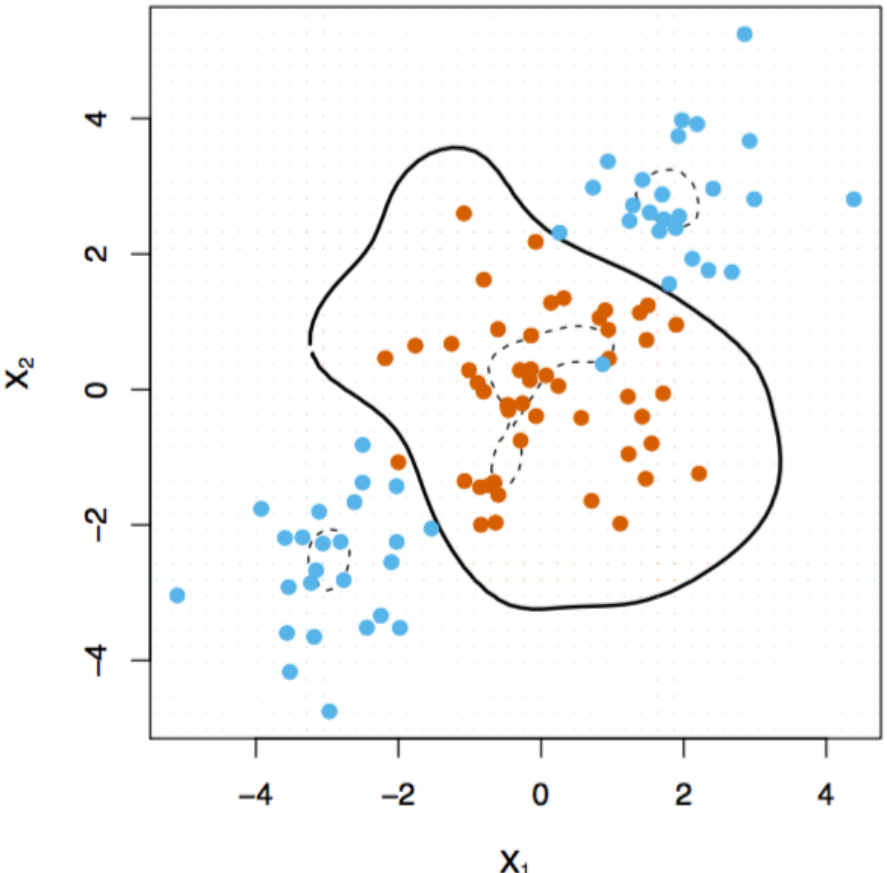
$\sigma = 10^{-1}$ $\lambda = 10^{-0}$



RBF kernel and random features

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

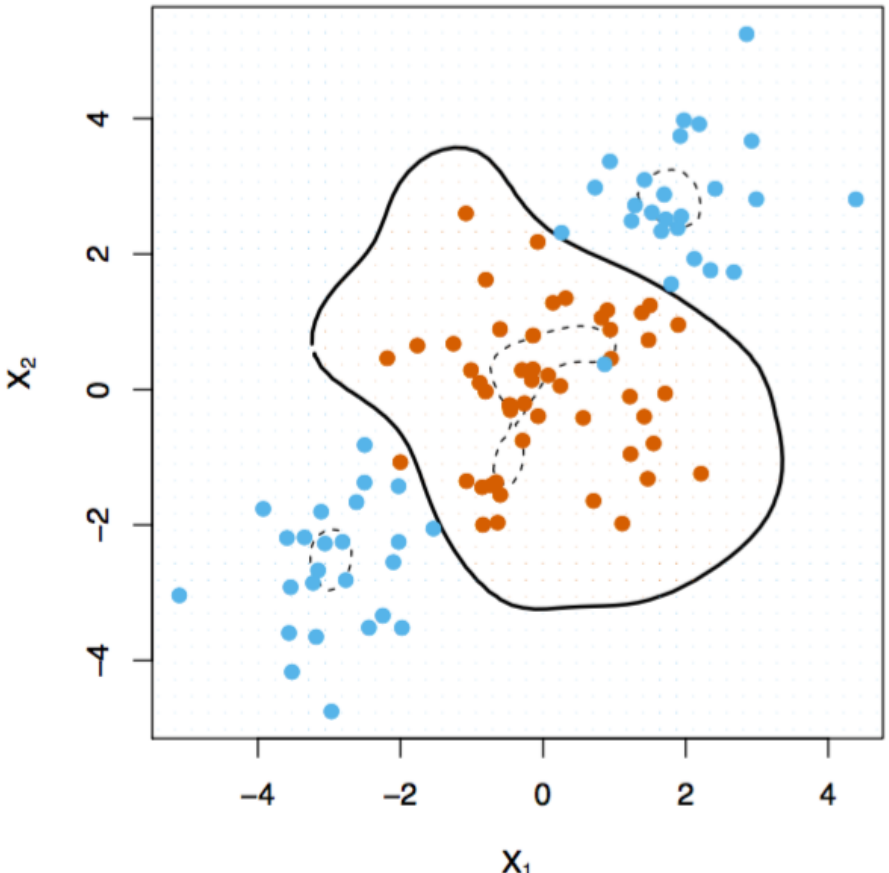
Bandwidth σ is large enough



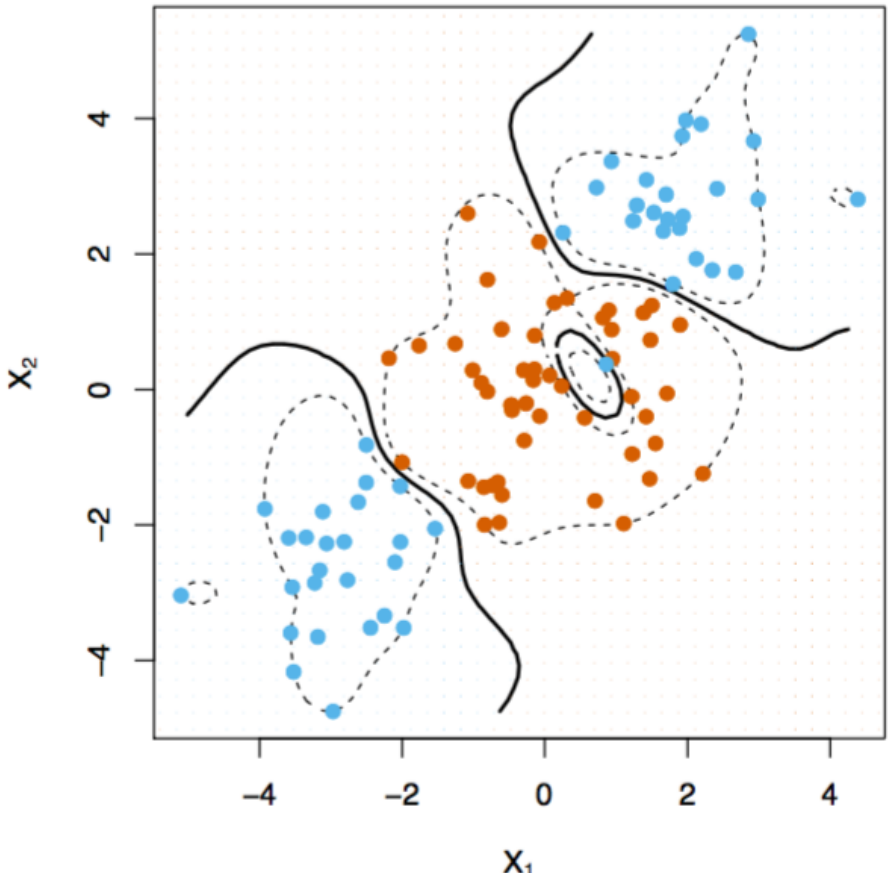
RBF kernel and random features

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

Bandwidth σ is large enough



Bandwidth σ is small



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

- Kernel method works well if we choose a good kernel such that the data is linearly separable in the corresponding (possibly infinite dimensional) feature space

- In practice, it is hard to choose a good kernel for a given problem

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

Neural networks!
(Our next topic :D)

Questions?