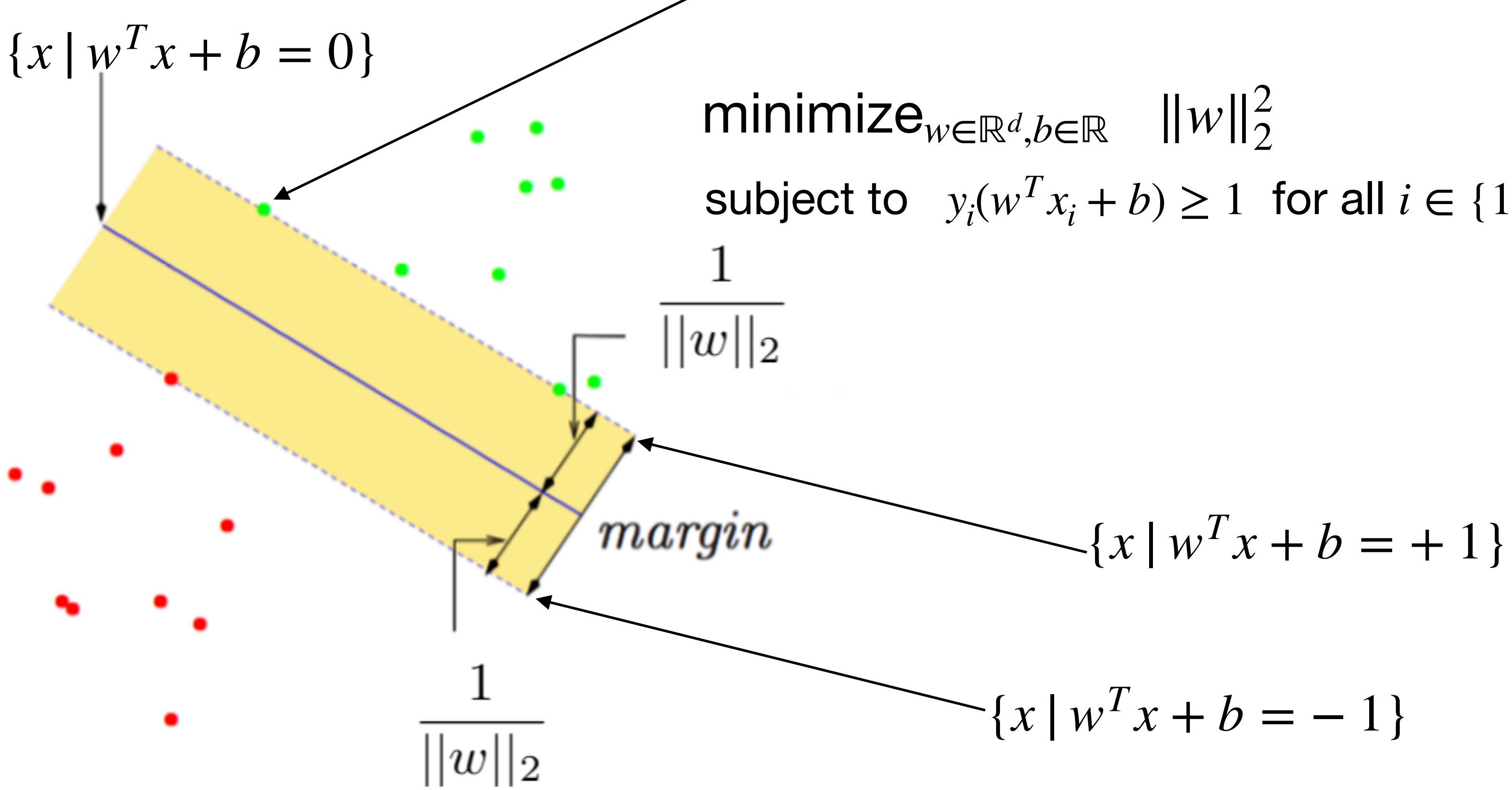
Lecture 16:
Kernels

Support Vector Machine

- we cheated a little in the sense that the reparametrization of $\|w\|_2 = \frac{1}{\gamma}$ is possible only if the the margins are positive,
  i.e. the data is linearly separable with a positive margin
- otherwise, there is no feasible solution
- the examples at the margin are called **support vectors**

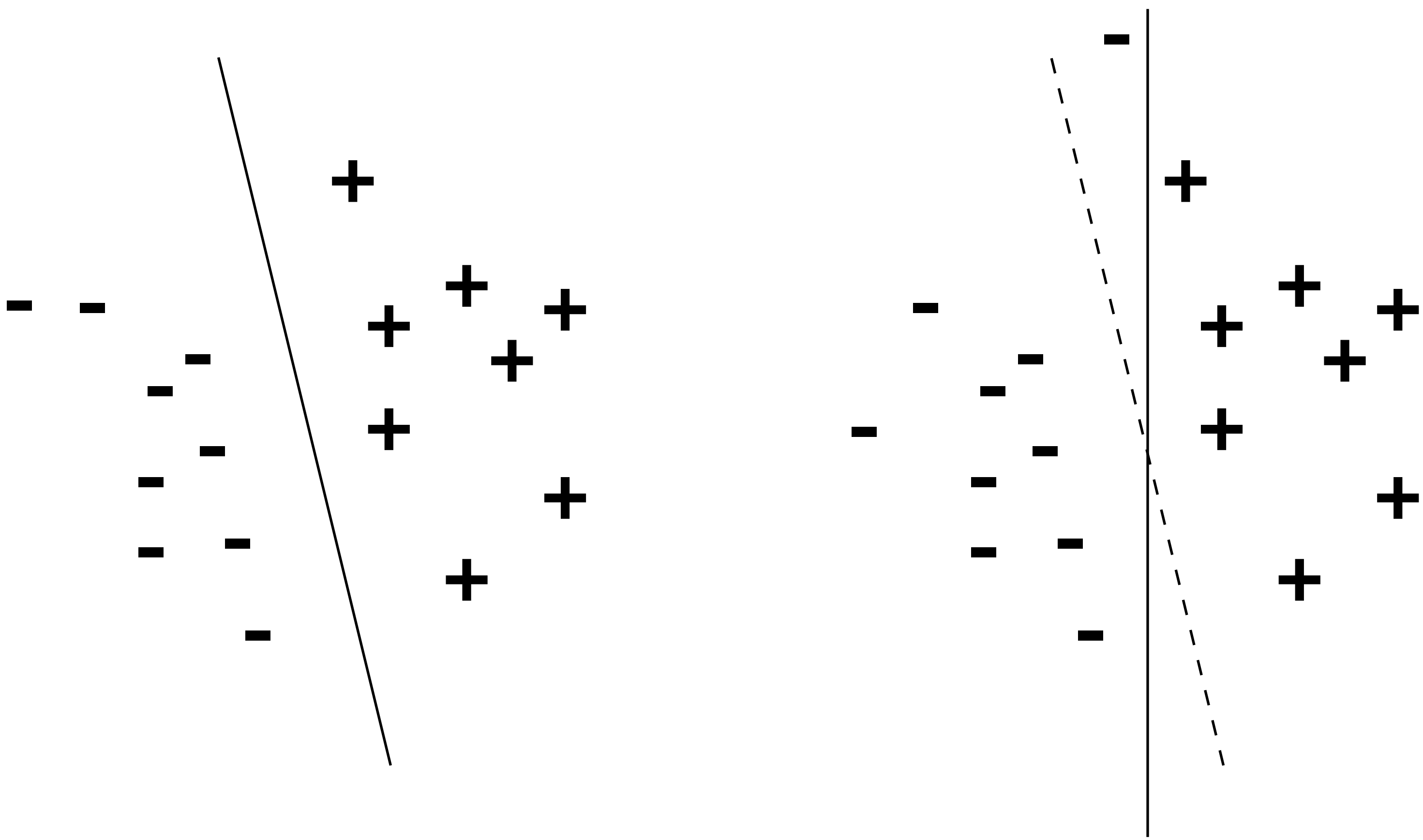
minimize$_{w \in \mathbb{R}^d, b \in \mathbb{R}}$ $\|w\|_2^2$

subject to $y_i(w^T x_i + b) \geq 1$ for all $i \in \{1, \ldots, n\}$

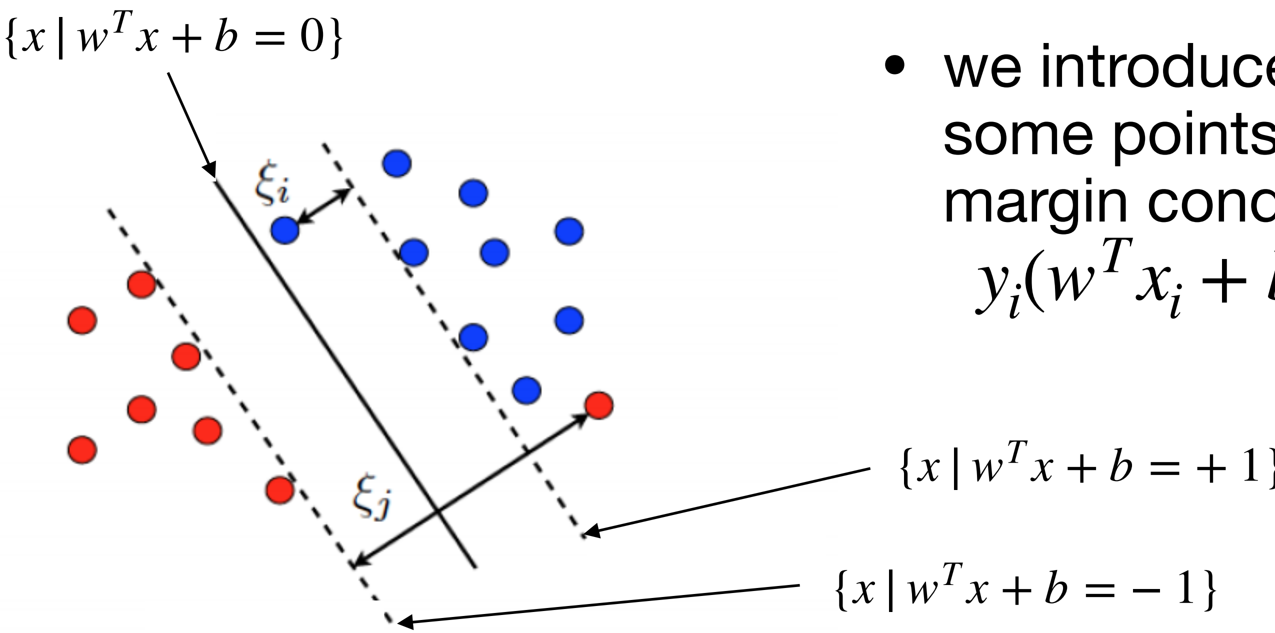


Two issues

- it does not generalize to non-separable datasets
- max-margin formulation we proposed is sensitive to outliers



What if the data is not separable?



- we introduce slack so that some points can violate the margin condition

$$y_i(w^T x_i + b) \;\geq\; 1 - \xi_i$$

- this gives a new optimization problem with some positive constant $c \in \mathbb{R}$

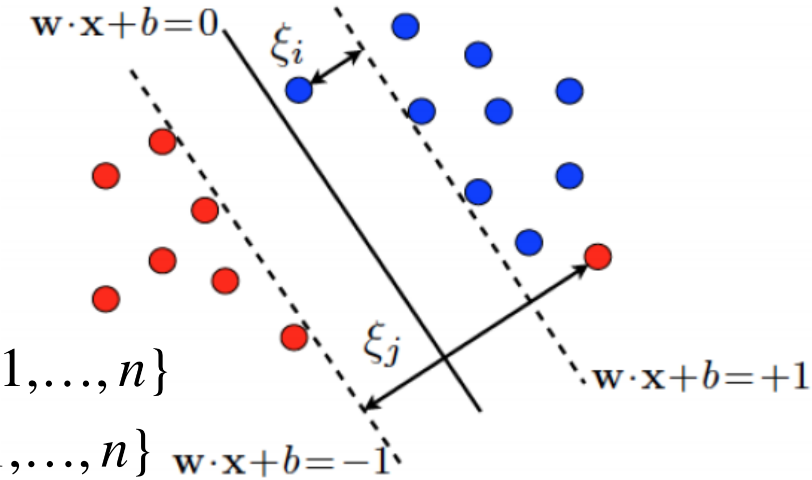
$$\text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}, \xi \in \mathbb{R}^n} \quad \|w\|_2^2 + c \sum_{i=1}^{n} \xi_i$$

$$\text{subject to} \quad y_i(w^T x_i + b) \geq 1 - \xi_i \quad \text{for all } i \in \{1,\ldots,n\}$$

$$\xi_i \;\geq 0 \quad \text{for all } i \in \{1,\ldots,n\}$$

the (re-scaled) margin (for each sample) is allowed to be less than one,
but you pay $c\xi_i$ in the cost, and c balances the two goals:
maximizing the margin for most examples vs. having small number of violations

Support Vector Machine



- for the optimization problem

$$\text{minimize}_{w\in\mathbb{R}^d, b\in\mathbb{R}, \xi\in\mathbb{R}^n} \quad \|w\|_2^2 + c\sum_{i=1}^{n}\xi_i$$

$$\text{subject to} \quad y_i(w^T x_i + b) \geq 1 - \xi_i \quad \text{for all } i \in \{1,\ldots,n\}$$

$$\xi_i \geq 0 \quad \text{for all } i \in \{1,\ldots,n\}$$

notice that at optimal solution, ξ_i's satisfy

- $\xi_i = 0$ if margin is big enough $y_i(w^T x_i + b) \geq 1$, or
- $\xi_i = 1 - y_i(w^T x_i + b)$, if the example is within the margin $y_i(w^T x_i + b) < 1$

- so one can write
  - $\xi_i = \max\{0, 1 - y_i(w^T x_i + b)\}$, which gives

$$\text{minimize}_{w\in\mathbb{R}^d, b\in\mathbb{R}} \quad \frac{1}{c}\|w\|_2^2 + \sum_{i=1}^{n}\max\{0, 1 - y_i(w^T x_i + b)\}$$

Sub-gradient descent for SVM

- SVM is the solution of

$$\text{minimize}_{w\in\mathbb{R}^d, b\in\mathbb{R}} \quad \frac{1}{c}\|w\|_2^2 + \sum_{i=1}^{n} \max\{0, 1 - y_i(w^T x_i + b)\}$$

- as it is non-differentiable, we solve it using sub-gradient descent
- which is exactly the same as gradient descent, except when we are at a non-differentiable point, we take one of the sub-gradients instead of the gradient (recall sub-gradient is a set)
- this means that we can take (a generic form derived from previous page)

$$\partial_w \ell(w^T x_i + b, y_i) \;=\; \mathbf{I}\{y_i(w^T x_i + b) \le 1\}(-y_i x_i)$$

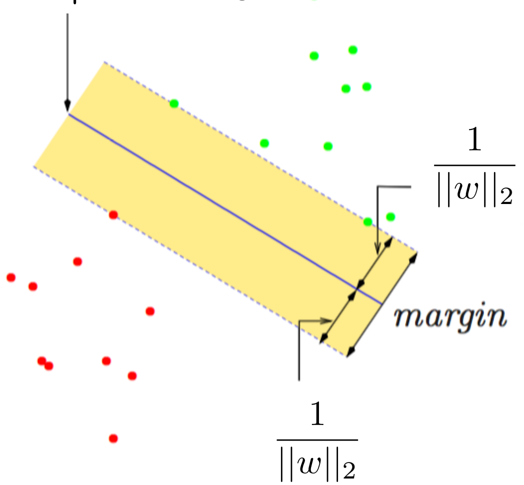
and apply

$$w^{(t+1)} \;\leftarrow\; w^{(t)} - \eta \left(\sum_{i=1}^{n} \mathbf{I}\{y_i((w^{(t)})^T x_i + b^{(t)}) \le 1\}(-y_i x_i) \;+\; \frac{2}{c} w^{(t)} \right)$$

$$b^{(t+1)} \;\leftarrow\; b^{(t)} - \eta \sum_{i=1}^{n} \mathbf{I}\{y_i((w^{(t)})^T x_i + b^{(t)}) \le 1)\}(-y_i)$$

What if the data is not linearly separable?

- $x^T w + b = 0$



some points don’t satisfy margin constraint:

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

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

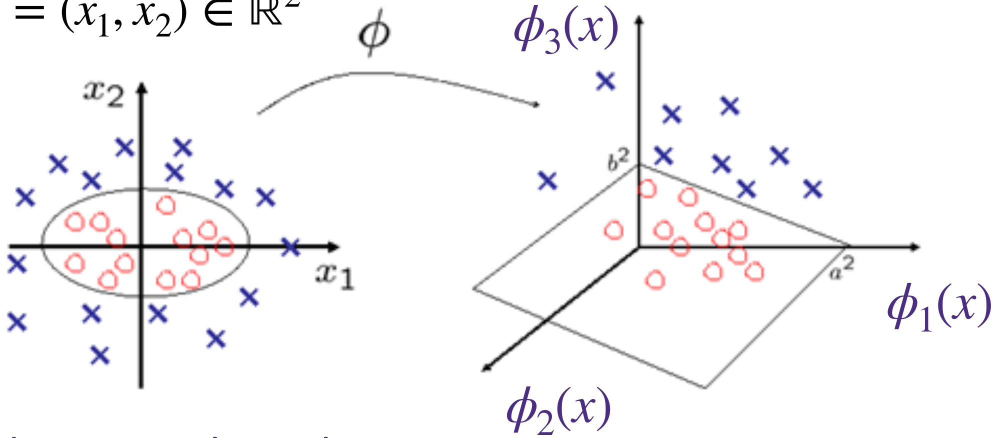
Two options:

1. Introduce slack to this optimization problem (Support Vector Machine)
2. **Lift to higher dimensional space (Kernels)**

What if the data is not linearly separable?

- Use features, for example,

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



This data is not linearly separable

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?

- Generally, in high dimensional feature space,
  it is easier to linearly separate different classes
- However, it is hard to know which feature map will work for given data
- So the rule of thumb is to use high-dimensional features and hope that the algorithm will automatically pick the right set of features
- What is wrong with this approach?

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:

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

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

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

- Feature space can get really large really quickly!
- How many coefficients/parameters are there for degree-k polynomials for $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$?
- At a first glance, it seems inevitable that we need memory (to store the features $\{\phi(x_i) \in \mathbb{R}^p\}_{i=1}^n$) and run-time that increases with p where $d < n \ll p$

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'.

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

- 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

  $$K(x . x') = \phi(x) \cdot \phi(x')$$

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

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

Ridge Linear Regression as Kernels

- Consider Ridge regression: $\hat{w} = \arg\min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_2^2$
- As an exercise, we will represent prediction with $\widehat{w}$ using linear kernel defined as $K(x, x') = x^T x'$
- Training: $\widehat{w} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}_{d\times d})^{-1}\mathbf{X}^T\mathbf{y}$

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

(when $n \ll d$ via linear algebra)

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

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

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

- Hence, to make prediction on any future data points, all we need to know is
- $\mathbf{X}x_{\text{new}} = \begin{bmatrix} K(x_1, x_{\text{new}}) \\ \vdots \\ K(x_n, x_{\text{new}}) \end{bmatrix} \in \mathbb{R}^n$, and $\mathbf{X}\mathbf{X}^T = \begin{bmatrix} K(x_1, x_1) & K(x_1, x_2) & \cdots \\ \vdots & \vdots & \\ K(x_n, x_1) & K(x_n, x_2) & \cdots \end{bmatrix} \in \mathbb{R}^{n\times n}$
- Even if we run ridge linear regression 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 $\phi(x_i)$

Kernel (i.e., dot-product) of polynomial features

- Recall kernel is defined as $K(x, x') = \phi(x) \cdot \phi(x') = \langle \phi(x), \phi(x') \rangle = \phi(x)^T \phi(x')$
- 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 $K(x, x') = x_1 x_1' + x_2 x_2'$
  - $\phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1 x_2 \\ x_2 x_1 \end{bmatrix}$ for $k = 2$ and $d = 2$,

    then $K(x, x') = x_1^2 (x_1')^2 + x_2^2 (x_2')^2 + 2 x_1 x_2 x_1' x_2' = (x_1 x_1' + x_2 x_2')^2$
- 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 (as we saw in the previous slide) 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)

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

- **Sigmoid**

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

The Kernel Trick

- **Given data** $\{(x_i, y_i)\}_{i=1}^n$
- **For a choice of a loss, use a linear predictor of the form** $\widehat{w} = \sum_{i=1}^n \alpha_i x_i$ **for some** $\alpha = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} \in \mathbb{R}^n$ **to be learned**
- **Prediction is** $\widehat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}} = \sum_{i=1}^n \alpha_i \, x_i^T x_{\text{new}}$
- **Design an algorithm that finds** α **while accessing the data only via** $\{x_i^T x_j\}$
- **Pick a kernel** $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$
- **Substitute** $x_i^T x_j$ **with** $K(x_i, x_j)$**, and find** α **using he above algorithm**
- **Make prediction with** $\widehat{y}_{\text{new}} = \sum_{i=1}^n \widehat{\alpha}_i K(x_i, x_{\text{new}})$

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

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

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

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

(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?

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

- So $\mathbf{K}$ is invertible and $\widehat{\alpha} = (\mathbf{K} + \lambda \mathbf{I}_{n \times n})^{-1} \mathbf{y}$ is well defined, but we still want to choose positive λ.
- What if $\lambda = 0$? What goes wrong?

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

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

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

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

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

(Switch inner product with kernel)

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

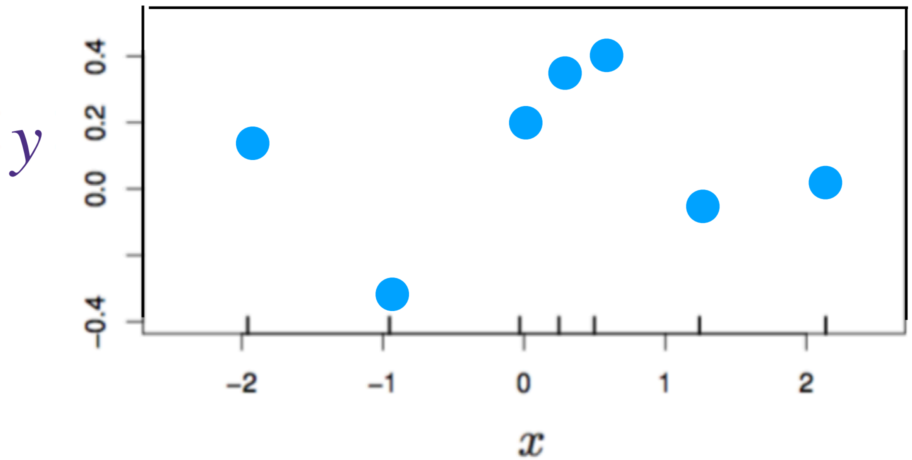
Where $\mathbf{K}_{ij} = K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$

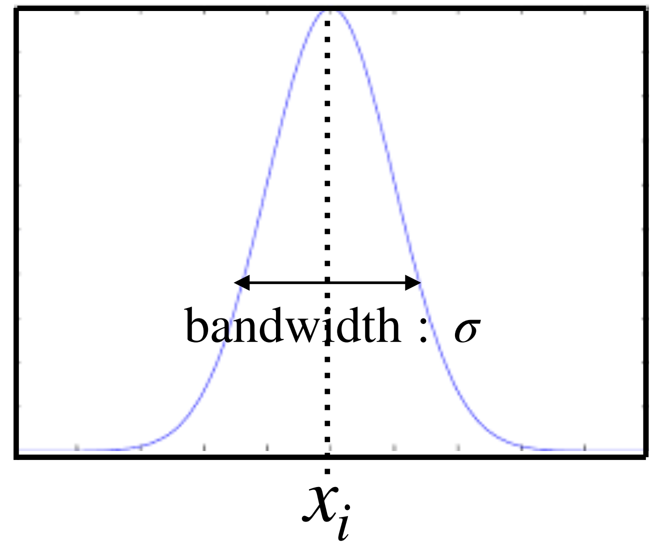
Prediction for x_{new}:

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

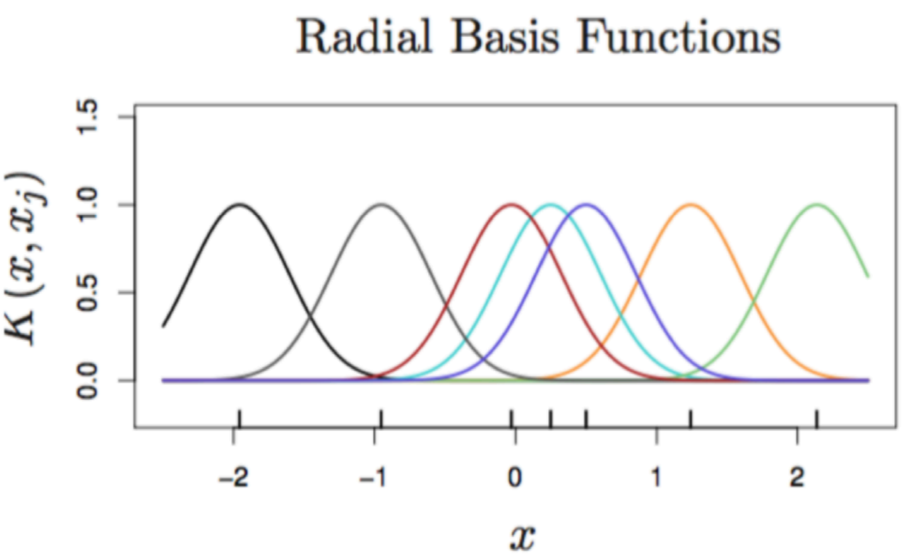
$$\widehat{y} = \text{sign}\left(\sum_{i=1}^{n} \widehat{\alpha}_i K(x_i, x_{\text{new}}) + \widehat{b} \right)$$

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

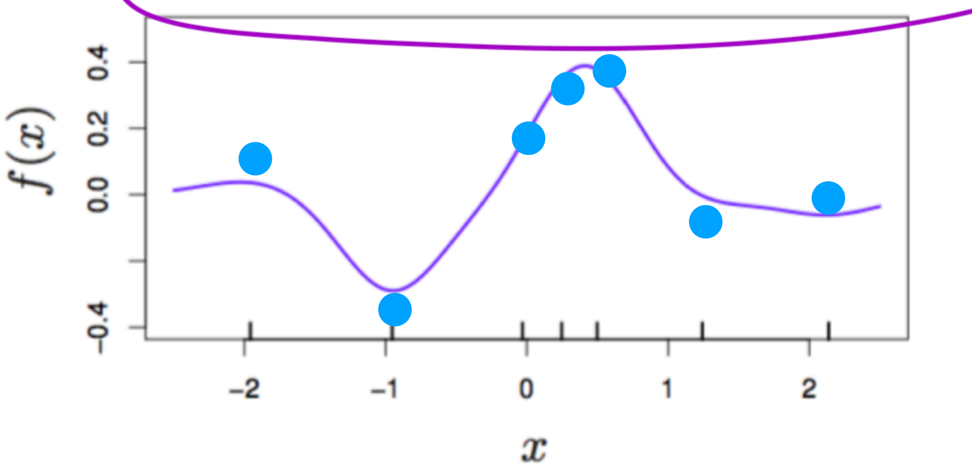




- predictor is taking weighted sum of n kernel functions centered at each sample points



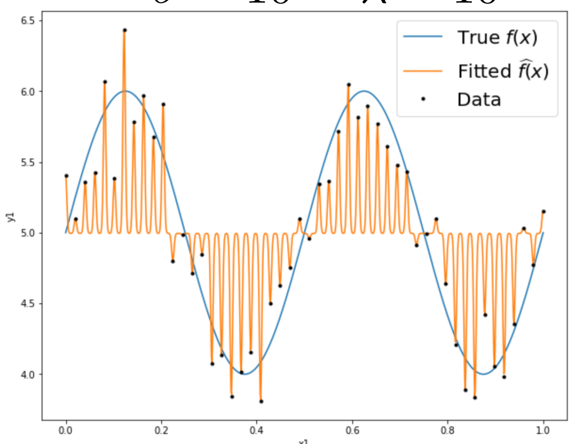
$f(x) = \alpha_0 + \sum_j \alpha_j K(x, x_j)$



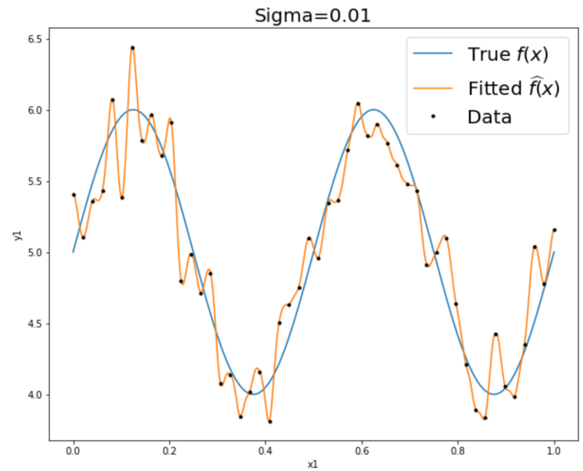
RBF kernel $k(x_i, x) = \exp\left\{ -\dfrac{\|x_i - x\|_2^2}{2\sigma^2} \right\}$

- $\mathscr{L}(w) \;=\; \|\mathbf{K}w - \mathbf{y}\|_2^2 \;+\; \lambda\|w\|_2^2$
- The bandwidth σ^2 of the kernel regularizes the predictor

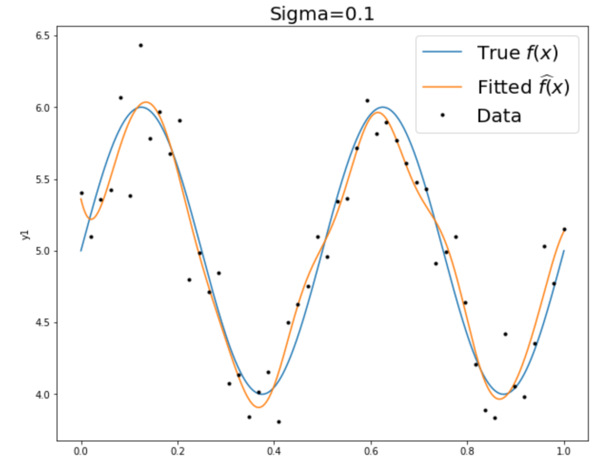
$\sigma = 10^{-3} \;\; \lambda = 10^{-4}$



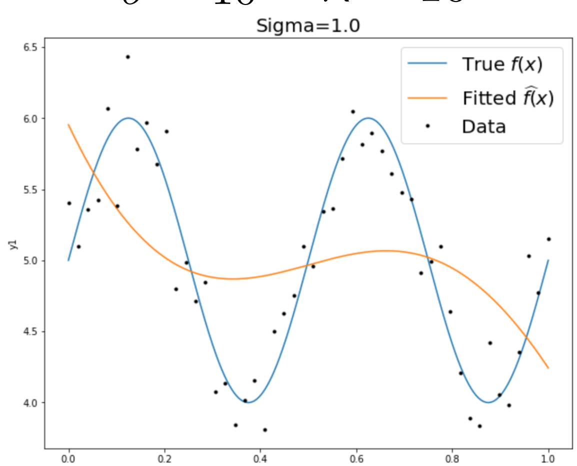
$\sigma = 10^{-2} \;\; \lambda = 10^{-4}$



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

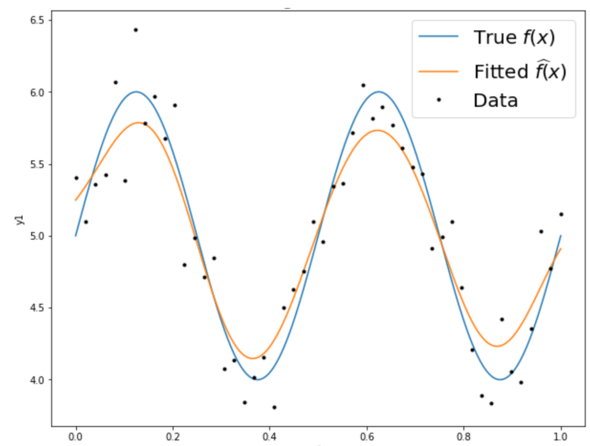


$\sigma = 10^{-0} \;\; \lambda = 10^{-4}$



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

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



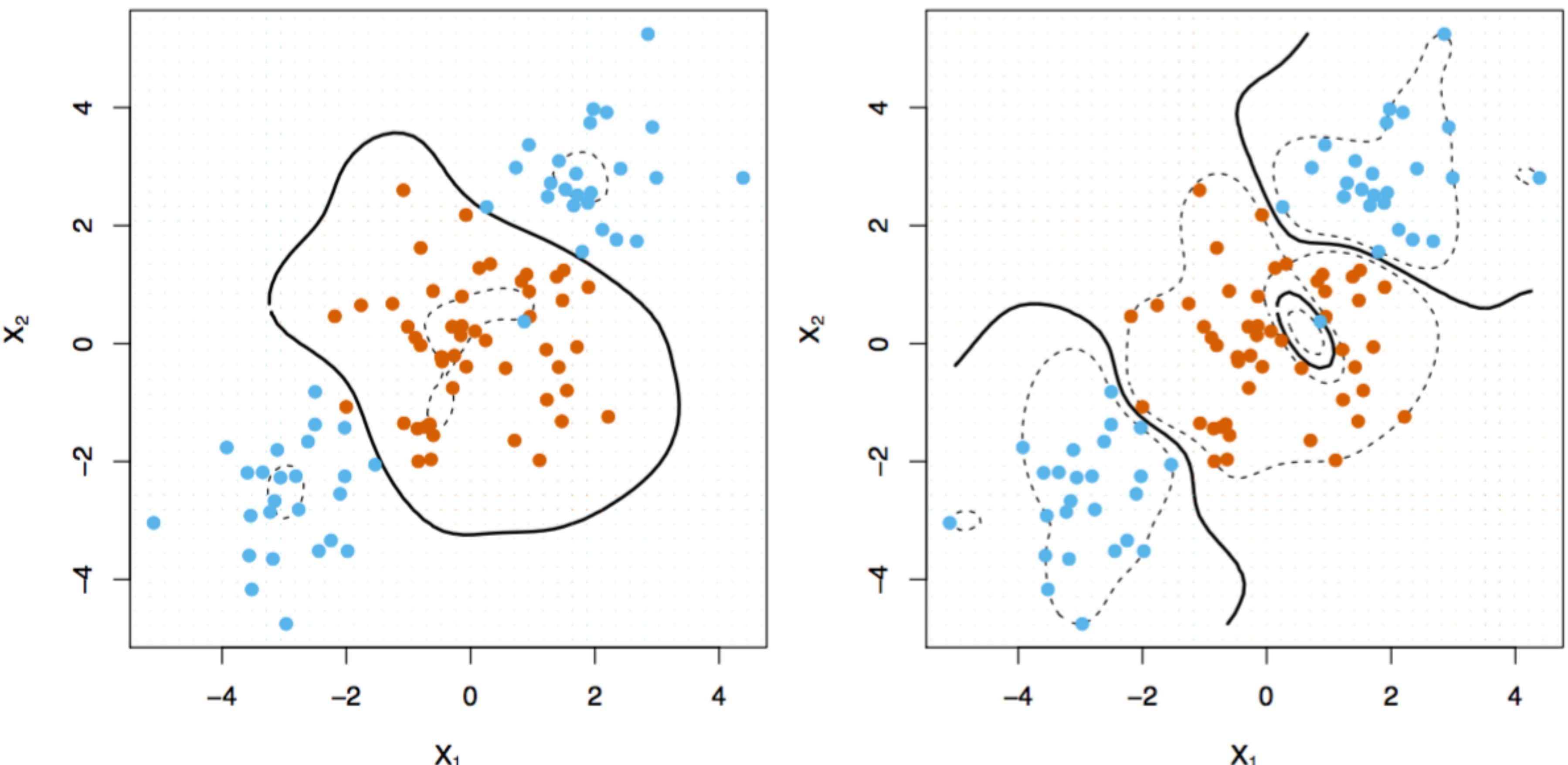
RBF kernel and random features

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

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

Bandwidth σ is large enough

Bandwidth σ is small



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.

$$2\cos(\alpha)\cos(\beta) = \cos(\alpha+\beta) + \cos(\alpha-\beta)$$

$$e^{jz} = \cos(z) + j\sin(z)$$

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

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

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

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

Random features approximate RBF kernel with $\gamma = \dfrac{1}{2\sigma^2}$

[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) $p = 20^3 = 8{,}000$

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

Fixed Feature V.S. Learned Feature

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

Questions?