CSCI 567: Machine Learning

Vatsal Sharan

Fall 2022

Lecture 4, Sep 15

USC University of Southern California

Administrivia

- HW2 will be released tonight, due in about 2 weeks.
- We will post some practice problems for the quiz by early next week.

Recap

Ensuring generalization

Theorem. *Let $\mathcal{F}$ be a function class with size $|\mathcal{F}|$. Let $y = f^*(\boldsymbol{x})$ for some $f^* \in \mathcal{F}$. Suppose we get a training set $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ of size n with each datapoint drawn i.i.d. from the data distribution D. Let*

$$f_S^{ERM} = \operatorname*{argmin}_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \ell(f(\boldsymbol{x}_i), y_i).$$

For any constants $\epsilon, \delta \in (0,1)$, if $n \geq \frac{\ln(|\mathcal{F}|/\delta)}{\epsilon}$, then with probability $(1-\delta)$ over $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, $R(f_S^{ERM}) < \epsilon$.

A useful rule of thumb: to guarantee generalization, make sure that your training data set size n is at least linear in the number d of free parameters in the function that you're trying to learn.

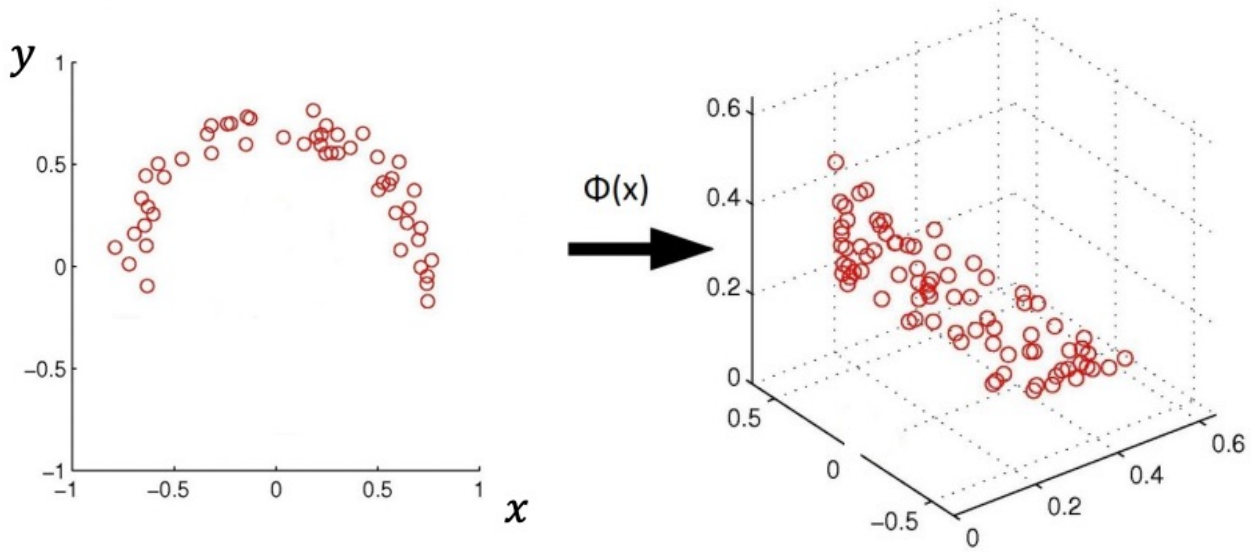
Beyond linear models: nonlinearly transformed features

1. Use a nonlinear mapping

$$\boldsymbol{\phi}(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^d \rightarrow \boldsymbol{z} \in \mathbb{R}^M$$

to transform the data to a more complicated feature space

2. Then apply linear regression (hope: linear model is a better fit for the new feature space).



Polynomial basis functions

Polynomial basis functions for $d = 1$

$$\boldsymbol{\phi}(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^M \end{bmatrix} \quad \Rightarrow \quad f(x) = w_0 + \sum_{m=1}^{M} w_m x^m$$

Learning a linear model in the new space
= learning an *M-degree polynomial model* in the original space

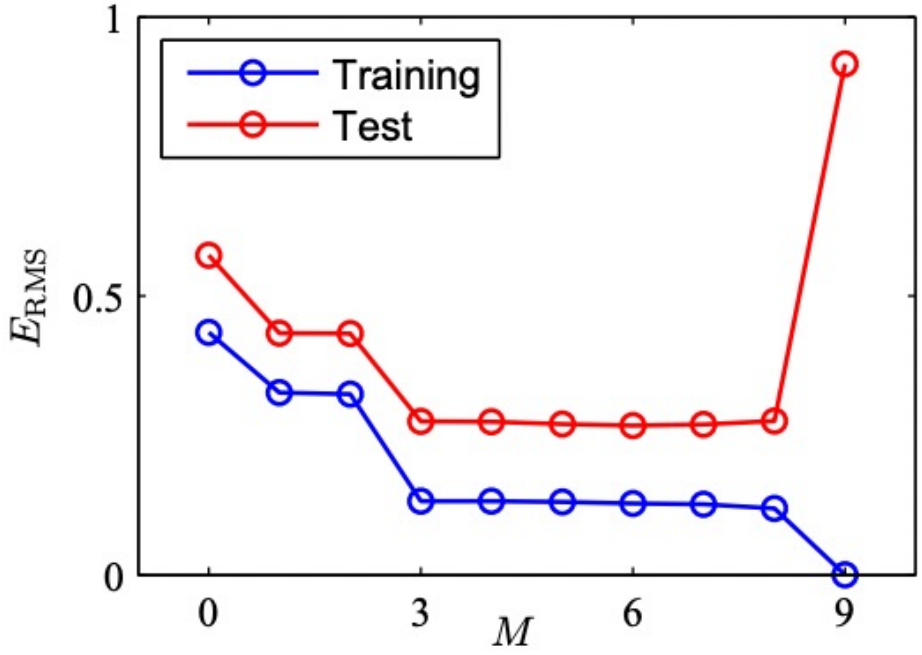
Underfitting and overfitting

$M \leq 2$ is *underfitting* the data

- large training error
- large test error

$M \geq 9$ is *overfitting* the data

- small training error
- **large test error**



More complicated models ⇒ larger gap between training and test error

How to prevent overfitting?

See Colab notebook

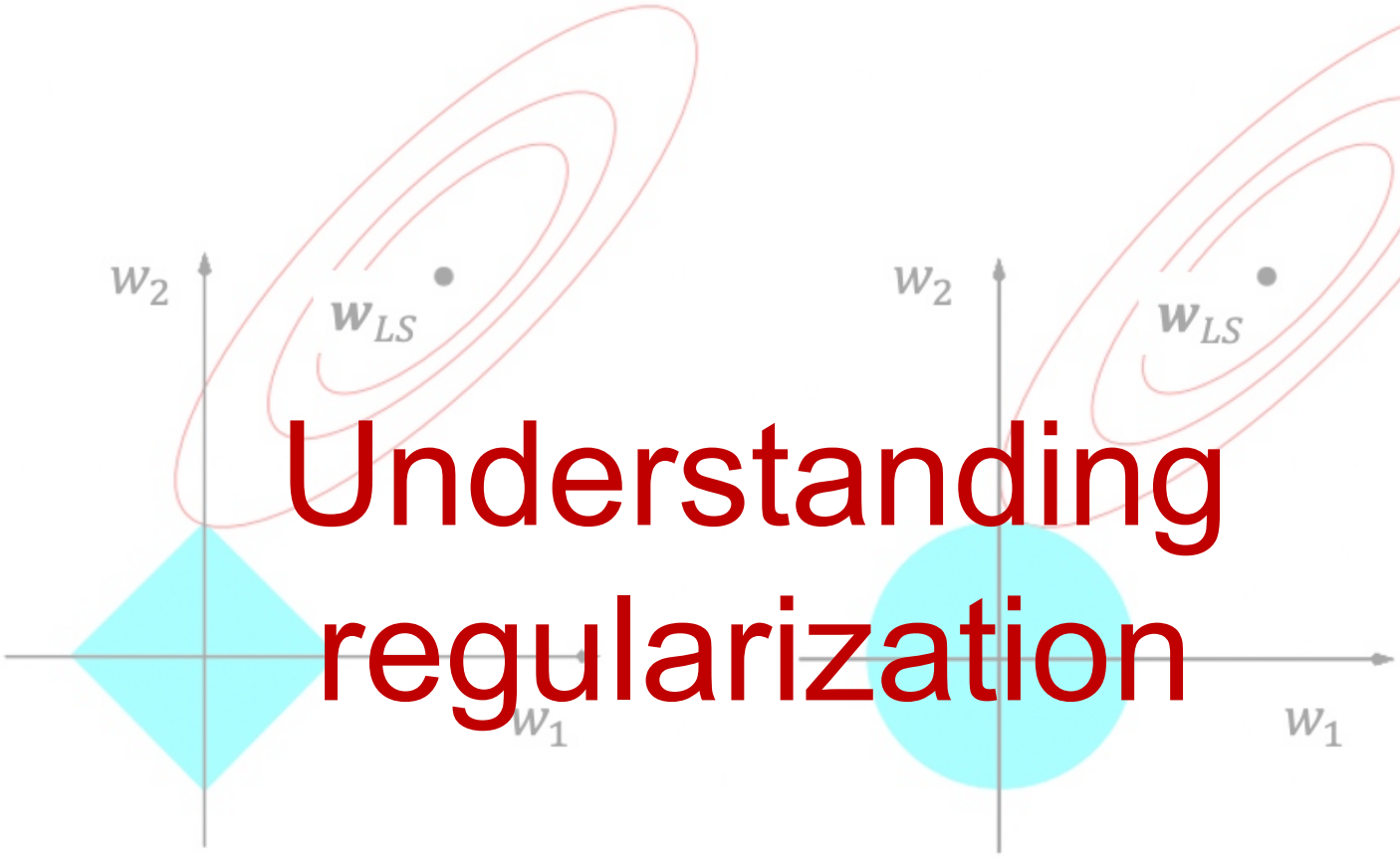
Preventing overfitting: **Regularization**

Regularized linear regression: new objective

$$G(\boldsymbol{w}) = \mathrm{RSS}(\boldsymbol{w}) + \lambda\psi(\boldsymbol{w})$$

Goal: find $\boldsymbol{w}^* = \mathrm{argmin}_w\, G(\boldsymbol{w})$

- $\psi : \mathbb{R}^d \to \mathbb{R}^+$ is the *regularizer*
  - measure how complex the model $\boldsymbol{w}$ is, penalize complex models
  - common choices: $\|\boldsymbol{w}\|_2^2$, $\|\boldsymbol{w}\|_1$, etc.
- $\lambda > 0$ is the *regularization coefficient*
  - $\lambda = 0$, no regularization
  - $\lambda \to +\infty$, $\boldsymbol{w} \to \mathrm{argmin}_w\, \psi(\boldsymbol{w})$
  - i.e. control **trade-off** between training error and complexity



Understanding regularization

ℓ_2 regularization: penalizing large weights

ℓ_2 regularization, $\psi(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2$:

$$G(\boldsymbol{w}) = \text{RSS}(\boldsymbol{w}) + \lambda\|\boldsymbol{w}\|_2^2 = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2$$

$$\begin{aligned}
&\nabla G(\boldsymbol{w}) = 2(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}) + 2\lambda\boldsymbol{w} = 0 \\
&\Rightarrow \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda\boldsymbol{I}\right)\boldsymbol{w} = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} \\
&\Rightarrow \boldsymbol{w}^* = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}
\end{aligned}$$

Linear regression with ℓ_2 regularization is also known as **ridge regression**.

With a Bayesian viewpoint, corresponds to a Gaussian prior for $\boldsymbol{w}$.

Encouraging sparsity: ℓ_0 regularization

Sparsity of w: Number of non-zero coefficients in $\boldsymbol{w}$. Same as $||\mathbf{w}||_0$

Advantage:

- Sparse models are a natural inductive bias in many settings. In many applications we have numerous possible features, only some of which may have any relationship with the label.
- Sparse models may also be more **interpretable**. They could narrow down a small number of features which carry a lot of signal.
- Data required to learn sparse model maybe significantly less than to learn dense model.

We'll see more on the third point next.

ℓ_0 regularization: The good, the bad and the ugly

Choose $\psi(\boldsymbol{w}) = \|\boldsymbol{w}\|_0$.

$$G(\boldsymbol{w}) = \sum_{i=1}^{n} (\boldsymbol{w}^T \boldsymbol{x}_i - y_i)^2 + \lambda \|\boldsymbol{w}\|_0.$$

ℓ_0 regularization: The good, the bad and the ugly

ℓ_0 regularization: The good, the bad and the ugly

ℓ_0 regularization: The good, the bad and the ugly

ℓ_1 regularization as a proxy for ℓ_0 regularization

Choose $\psi(\boldsymbol{w}) = \|\boldsymbol{w}\|_1$.

$$G(\boldsymbol{w}) = \sum_{i=1}^{n} (\boldsymbol{w}^T \boldsymbol{x}_i - y_i)^2 + \lambda \|\boldsymbol{w}\|_1.$$

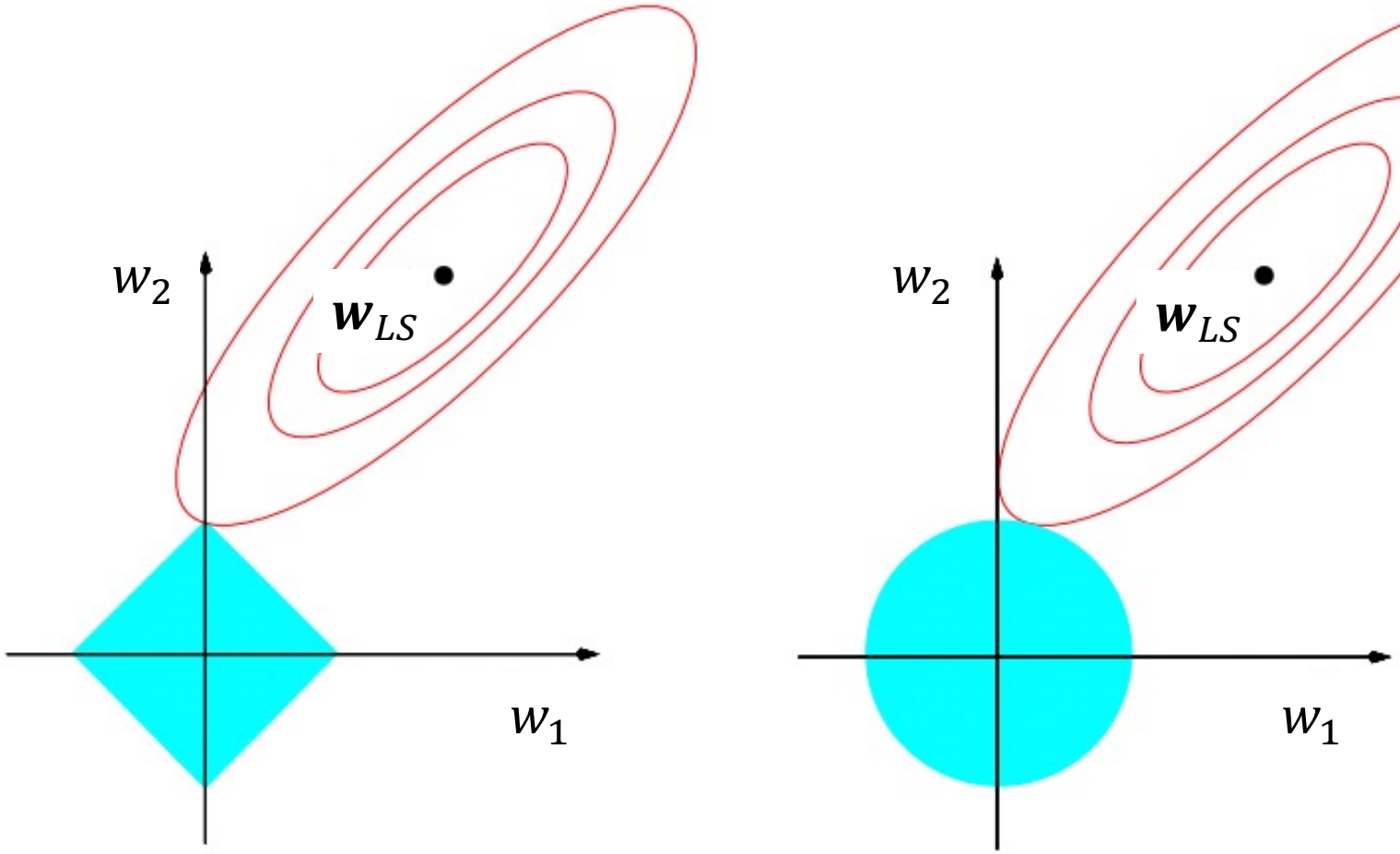
ℓ_1 regularization as a proxy for ℓ_0 regularization

Theorem. *Given n vectors $\{\boldsymbol{x}_i \in \mathbb{R}^d, i \in [n]\}$ drawn i.i.d. from $N(0, \boldsymbol{I})$, let $y_i = \boldsymbol{w}^{*T}\boldsymbol{x}_i$ for some $\boldsymbol{w}^*$ with $\|\boldsymbol{w}^*\|_0 = s$. Then for some fixed constant $C > 0$, the minimizer of $G(\boldsymbol{w})$ with $\psi(\boldsymbol{w}) = \|\boldsymbol{w}\|_1$ will be $\boldsymbol{w}^*$ as long as $n > C \cdot s \log d$ (with high probability over the randomness in the training datapoints $\boldsymbol{x}_i$).*

[similar result can also be proven under more general conditions].

Why does ℓ_1 regularization encourage sparse solutions?

Optimization problem: $\text{argmin}_w \text{RSS}(\boldsymbol{w})$, subject to $\psi(\boldsymbol{w}) \leq \beta$



Adapted from ESL

Diving deeper: ℓ_1 and ℓ_2 regularization for the “isotropic” case

Diving deeper: ℓ_1 and ℓ_2 regularization for the “isotropic” case

Diving deeper: ℓ_1 and ℓ_2 regularization for the “isotropic” case

Diving deeper: ℓ_1 and ℓ_2 regularization for the “isotropic” case

Diving deeper: ℓ_1 and ℓ_2 regularization for the “isotropic” case

Diving deeper: ℓ_1 and ℓ_2 regularization for the “isotropic” case

Diving deeper: ℓ_1 and ℓ_2 regularization for the "isotropic" case

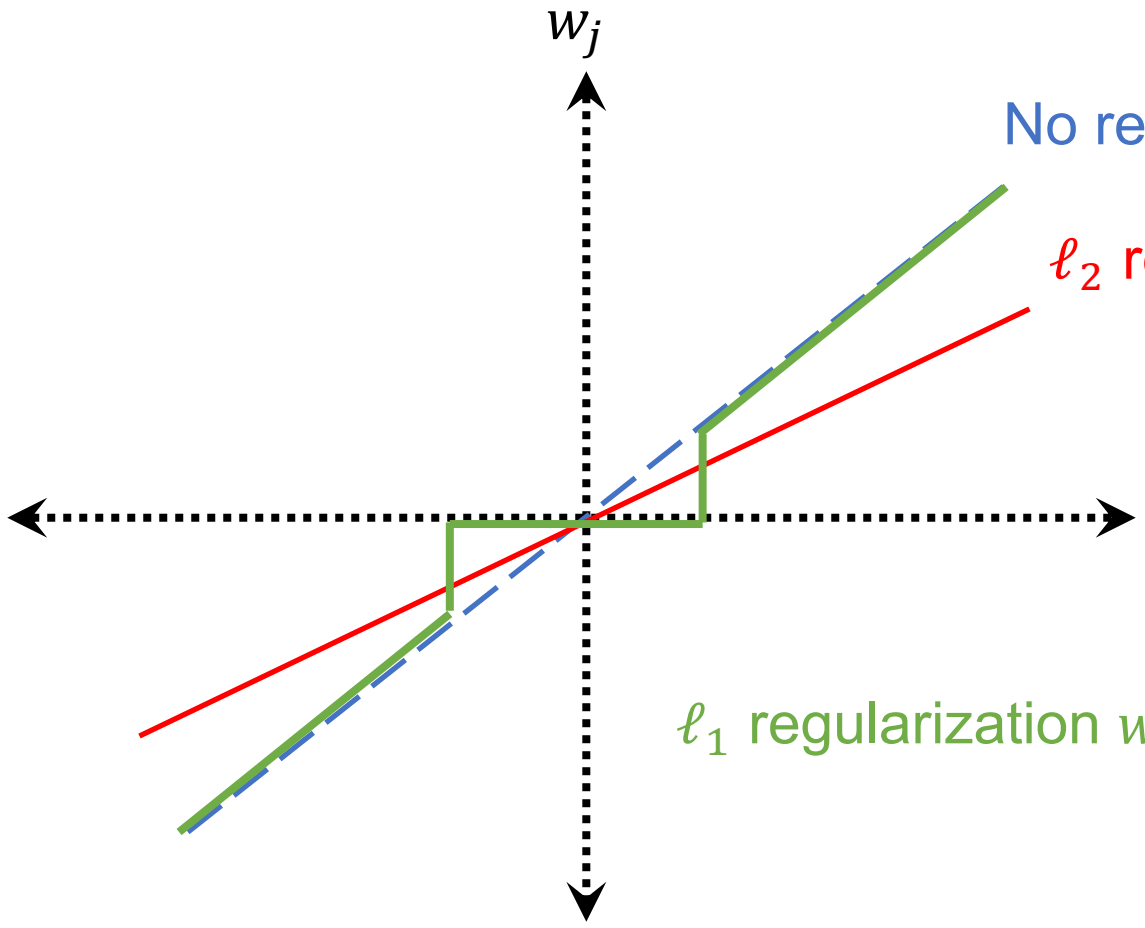
Let $\beta_j = \boldsymbol{X}_{(i)}^T \boldsymbol{y}$

Using subgradients, we can show that for the ℓ_1 regularized case:

$$w_j = \begin{cases} \beta_j - \lambda/2, \beta_j > \lambda/2 \\ 0, \left|\beta_j\right| \leq \lambda/2 \\ \beta_j + \lambda/2, \beta_j < -\lambda/2 \end{cases}$$

Diving deeper: ℓ_1 and ℓ_2 regularization for the "isotropic" case

Summary: Isotropic case ($\boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{I}$).
Let $\beta_j = \boldsymbol{X}_{(i)}^T\boldsymbol{y}$



No regularization $w_j = \beta_j$

ℓ_2 regularization $w_j = \beta_j/(1+\lambda)$

ℓ_1 regularization $w_j = \begin{cases} \beta_j - \lambda/2, \beta_j > \lambda/2 \\ 0, |\beta_j| \leq \lambda/2 \\ \beta_j + \lambda/2, \beta_j < -\lambda/2 \end{cases}$

Implicit regularization

So far, we explicitly added a $\psi(\boldsymbol{w})$ term to our objective function to regularize.

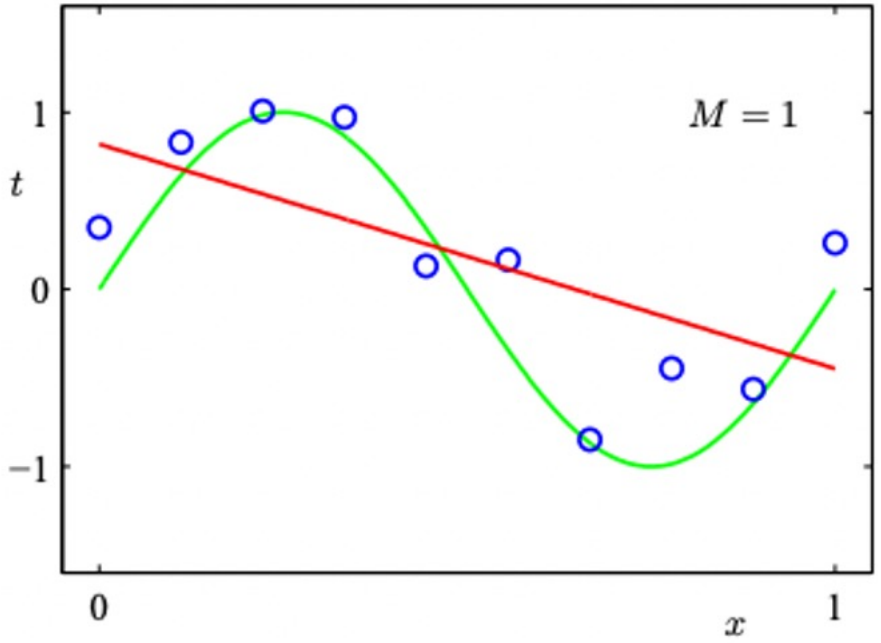
In many cases, the optimization algorithm we use can themselves act as regularizers, favoring some solutions over others.

Currently a very active area of research, you'll see more in the homework.

Bias-variance tradeoff

The phenomenon of underfitting and overfitting is often referred to as the ***bias-variance tradeoff*** in the literature.

A model whose complexity is too *small* for the task will *underfit*. This is a model with a large bias because the model's accuracy will not improve even if we add a lot of training data.

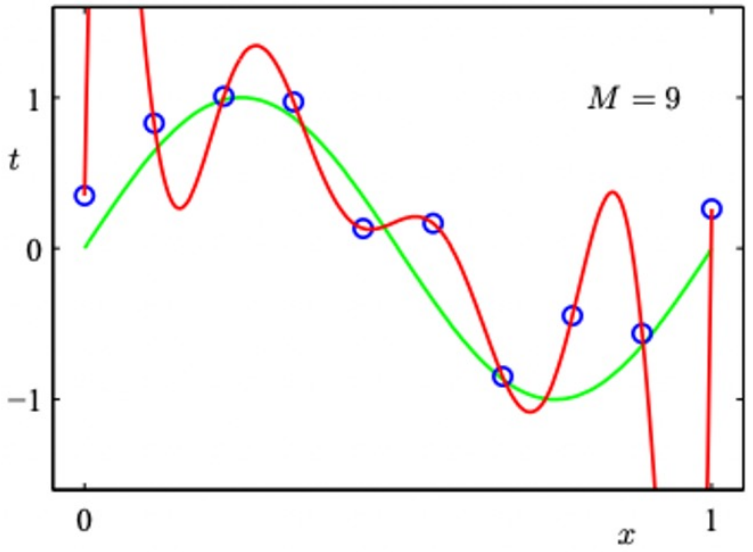


sin(x) fitting example we saw in Lec 3

Bias-variance tradeoff

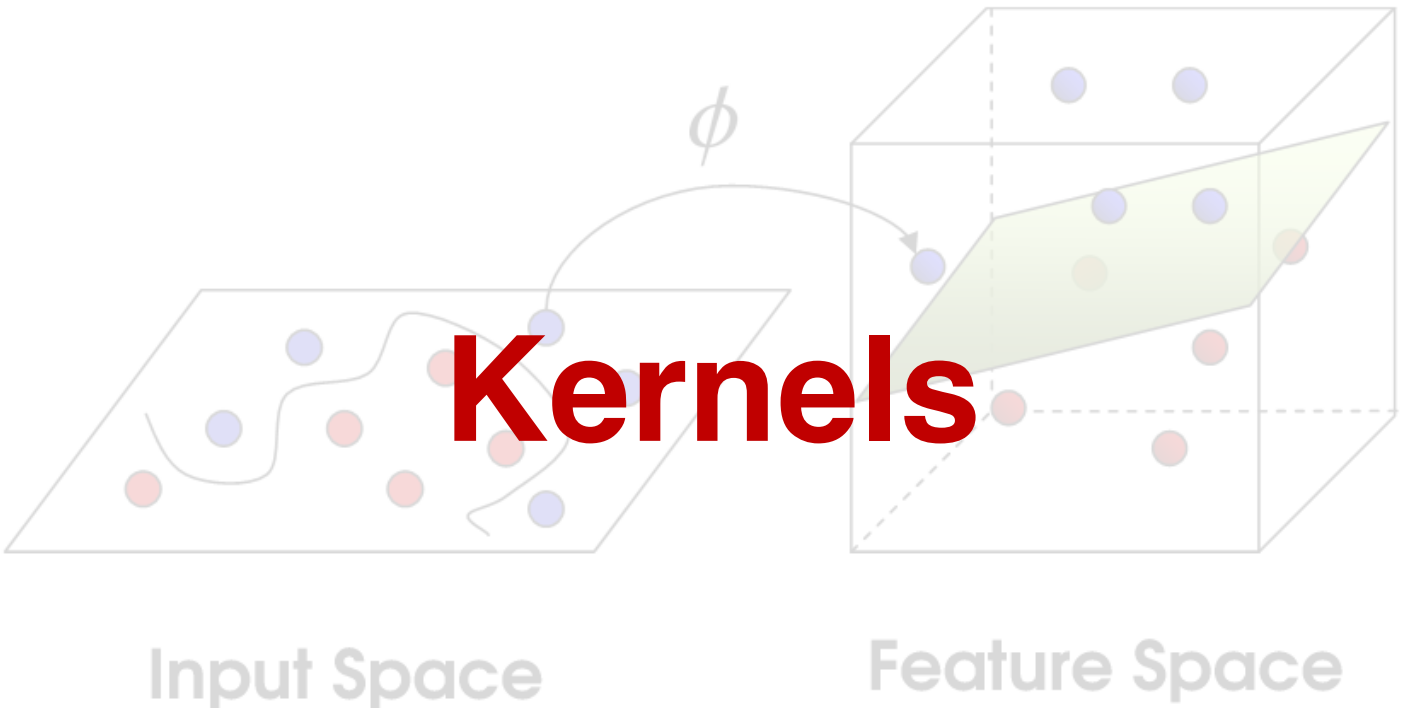
The phenomenon of underfitting and overfitting is often referred to as the ***bias-variance tradeoff*** in the literature.

A model whose complexity is too *large* for the amount of available training data will *overfit*. This is a model with high variance, because the model's predictions will vary a lot with the randomness in the training data (it can even fit any noise in the training data).



sin(x) fitting example we saw in Lec 3

Kernels



Motivation

Recall the nonlinear function map for linear regression:

1. Use a nonlinear mapping

$$\boldsymbol{\phi}(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^d \rightarrow \boldsymbol{z} \in \mathbb{R}^M$$

to transform the data to a more complicated feature space

2. Then apply linear regression (hope: linear model is a better fit for the new feature space).

Kernel methods give a way to choose and efficiently work with the nonlinear map $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^M$ (for linear regression, and much more broadly).

Regularized least squares

Let's continue with regularized least squares with non-linear basis:

$$
\begin{aligned}
\boldsymbol{w}^* &= \operatorname*{argmin}_{\boldsymbol{w}} F(\boldsymbol{w}) \\
&= \operatorname*{argmin}_{\boldsymbol{w}} \left(\|\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda \|\boldsymbol{w}\|_2^2 \right) \\
&= \left(\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} + \lambda \boldsymbol{I} \right)^{-1} \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{y}
\end{aligned}
\qquad
\boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1)^{\mathrm{T}} \\ \boldsymbol{\phi}(\boldsymbol{x}_2)^{\mathrm{T}} \\ \vdots \\ \boldsymbol{\phi}(\boldsymbol{x}_n)^{\mathrm{T}} \end{pmatrix}, \quad \boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}
$$

This operates in space $\mathbb{R}^M$ and M could be huge (and even infinite).

Regularized least squares solution: Another look

By setting the gradient of $F(\boldsymbol{w}) = \|\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2$ to be $\mathbf{0}$:

$$\boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{\Phi}\boldsymbol{w}^* - \boldsymbol{y}) + \lambda\boldsymbol{w}^* = \mathbf{0}$$

we know

$$\boldsymbol{w}^* = \frac{1}{\lambda}\boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w}^*) = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha} = \sum_{i=1}^{n} \alpha_i \boldsymbol{\phi}(\boldsymbol{x}_i)$$

Thus the least square solution is **a linear combination of features of the datapoints**!

This calculation does not show what $\boldsymbol{\alpha}$ should be, but ignore that for now.

Why is this helpful?

Assuming we know $\boldsymbol{\alpha}$, the prediction of $\boldsymbol{w}^*$ on a new example $\boldsymbol{x}$ is

$$\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i \boldsymbol{\phi}(\boldsymbol{x}_i)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$$

Therefore, *only inner products in the new feature space matter!*

Kernel methods are exactly about computing inner products *without explicitly computing* ϕ.

But we need to figure out what $\boldsymbol{\alpha}$ is first!

Solving for α, Step 1: Kernel matrix

Plugging in $\boldsymbol{w} = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}$ into $F(\boldsymbol{w})$ gives

$$\begin{aligned} H(\boldsymbol{\alpha}) &= F(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}) \\ &= \|\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}\|_2^2 \\ &= \|\boldsymbol{K}\boldsymbol{\alpha} - \boldsymbol{y}\|_2^2 + \lambda\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{K}\boldsymbol{\alpha}\boldsymbol{K} \quad (\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} \in \mathbb{R}^{n\times n}) \end{aligned}$$

$\boldsymbol{K}$ is called **Gram matrix** or **kernel matrix** where the (i, j)-th entry is

$$\boldsymbol{K}_{(i,j)} = \boldsymbol{\phi}(\boldsymbol{x}_i)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_j)$$

Kernel matrix: Example

$$\boldsymbol{\phi}(x_1) = \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix} \quad \boldsymbol{\phi}(x_2) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \boldsymbol{\phi}(x_3) = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Gram/Kernel matrix

$$\boldsymbol{K} = \begin{pmatrix} \boldsymbol{\phi}(x_1)^{\mathrm{T}}\boldsymbol{\phi}(x_1) & \boldsymbol{\phi}(x_1)^{\mathrm{T}}\boldsymbol{\phi}(x_2) & \boldsymbol{\phi}(x_1)^{\mathrm{T}}\boldsymbol{\phi}(x_3) \\ \boldsymbol{\phi}(x_2)^{\mathrm{T}}\boldsymbol{\phi}(x_1) & \boldsymbol{\phi}(x_2)^{\mathrm{T}}\boldsymbol{\phi}(x_2) & \boldsymbol{\phi}(x_2)^{\mathrm{T}}\boldsymbol{\phi}(x_3) \\ \boldsymbol{\phi}(x_3)^{\mathrm{T}}\boldsymbol{\phi}(x_1) & \boldsymbol{\phi}(x_3)^{\mathrm{T}}\boldsymbol{\phi}(x_2) & \boldsymbol{\phi}(x_3)^{\mathrm{T}}\boldsymbol{\phi}(x_3) \end{pmatrix}$$

$$= \begin{pmatrix} 4 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 4 \end{pmatrix}$$

Kernel matrix vs Covariance matrix

	dimensions	entry (i, j)	property
$\mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}$			
$\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}$			

Solving for α, Step 2: Minimize the dual

Minimize (the so-called *dual formulation*)

$$H(\boldsymbol{\alpha}) = \|\boldsymbol{K}\boldsymbol{\alpha} - \boldsymbol{y}\|_2^2 + \lambda \boldsymbol{\alpha}^{\mathrm{T}} \boldsymbol{K} \boldsymbol{\alpha}$$

Setting the derivative to $\mathbf{0}$ we have

$$\mathbf{0} = (\boldsymbol{K}^2 + \lambda \boldsymbol{K})\boldsymbol{\alpha} - \boldsymbol{K}\boldsymbol{y} = \boldsymbol{K}\left((\boldsymbol{K} + \lambda \boldsymbol{I})\boldsymbol{\alpha} - \boldsymbol{y}\right)$$

Thus $\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{y}$ **is a minimizer** and we obtain

$$\boldsymbol{w}^* = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha} = \boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{y}$$

Exercise: *are there other minimizers? and are there other $\boldsymbol{w}^*$'s?*

Comparing two solutions

Minimizing $F(w)$ gives $w^* = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T y$

Minimizing $H(\alpha)$ gives $w^* = \Phi^T(\Phi\Phi^T + \lambda I)^{-1} y$

Note I has different dimensions in these two formulas.

Natural question: *are the two solutions the same or different?*

They have to be the same because $F(w)$ has a unique minimizer!

And they are:

$$
\begin{aligned}
&(\Phi^T\Phi + \lambda I)^{-1}\Phi^T y \\
&= (\Phi^T\Phi + \lambda I)^{-1}\Phi^T(\Phi\Phi^T + \lambda I)(\Phi\Phi^T + \lambda I)^{-1} y \\
&= (\Phi^T\Phi + \lambda I)^{-1}(\Phi^T\Phi\Phi^T + \lambda\Phi^T)(\Phi\Phi^T + \lambda I)^{-1} y \\
&= (\Phi^T\Phi + \lambda I)^{-1}(\Phi^T\Phi + \lambda I)\Phi^T(\Phi\Phi^T + \lambda I)^{-1} y \\
&= \Phi^T(\Phi\Phi^T + \lambda I)^{-1} y
\end{aligned}
$$

The kernel trick

If the solutions are the same, then what is the difference?

First, computing $(\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} + \lambda \boldsymbol{I})^{-1}$ can be more efficient than computing $(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1}$ when $n \leq M$.

More importantly, computing $\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{y}$ also *only requires computing inner products in the new feature space!*

Now we can conclude that the exact form of $\boldsymbol{\phi}(\cdot)$ is not essential; *all we need to do is know the inner products* $\boldsymbol{\phi}(\boldsymbol{x})^T\boldsymbol{\phi}(\boldsymbol{x}')$.

For some $\boldsymbol{\phi}$ it is indeed possible to compute $\boldsymbol{\phi}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}')$ without computing/knowing $\boldsymbol{\phi}$. This is the *kernel trick*.

The kernel trick: Example 1

Consider the following polynomial basis $\boldsymbol{\phi} : \mathbb{R}^2 \rightarrow \mathbb{R}^3$:

$$\boldsymbol{\phi}(\boldsymbol{x}) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1 x_2 \\ x_2^2 \end{pmatrix}$$

What is the inner product between $\boldsymbol{\phi}(\boldsymbol{x})$ and $\boldsymbol{\phi}(\boldsymbol{x}')$?

$$\begin{aligned} \boldsymbol{\phi}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}') &= {x_1}^2{x'_1}^2 + 2x_1x_2x'_1x'_2 + {x_2}^2{x'_2}^2 \\ &= (x_1x'_1 + x_2x'_2)^2 = (\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}')^2 \end{aligned}$$

Therefore, *the inner product in the new space is simply a function of the inner product in the original space*.

The kernel trick: Example 2

$\boldsymbol{\phi} : \mathbb{R}^d \to \mathbb{R}^{2d}$ is parameterized by θ:

$$\boldsymbol{\phi}_\theta(\boldsymbol{x}) = \begin{pmatrix} \cos(\theta x_1) \\ \sin(\theta x_1) \\ \vdots \\ \cos(\theta x_m) \\ \sin(\theta x_m) \end{pmatrix}$$

What is the inner product between $\boldsymbol{\phi}_\theta(\boldsymbol{x})$ and $\boldsymbol{\phi}_\theta(\boldsymbol{x}')$?

$$\begin{aligned} \boldsymbol{\phi}_\theta(\boldsymbol{x})^{\mathrm{T}} \boldsymbol{\phi}_\theta(\boldsymbol{x}') &= \sum_{m=1}^{d} \cos(\theta x_m)\cos(\theta x'_m) + \sin(\theta x_m)\sin(\theta x'_m) \\ &= \sum_{m=1}^{d} \cos(\theta(x_m - x'_m)) \qquad \text{(trigonometric identity)} \end{aligned}$$

Once again, *the inner product in the new space is a simple function of the features in the original space.*

The kernel trick: Example 3

Based on ϕ_θ, define $\phi_L : \mathbb{R}^d \to \mathbb{R}^{2d(L+1)}$ for some integer L:

$$\phi_L(\boldsymbol{x}) = \begin{pmatrix} \phi_0(\boldsymbol{x}) \\ \phi_{\frac{2\pi}{L}}(\boldsymbol{x}) \\ \phi_{2\frac{2\pi}{L}}(\boldsymbol{x}) \\ \vdots \\ \phi_{L\frac{2\pi}{L}}(\boldsymbol{x}) \end{pmatrix}$$

What is the inner product between $\phi_L(\boldsymbol{x})$ and $\phi_L(\boldsymbol{x}')$?

$$\begin{aligned} \phi_L(\boldsymbol{x})^{\mathrm{T}} \phi_L(\boldsymbol{x}') &= \sum_{\ell=0}^{L} \phi_{\frac{2\pi\ell}{L}}(\boldsymbol{x})^{\mathrm{T}} \phi_{\frac{2\pi\ell}{L}}(\boldsymbol{x}') \\ &= \sum_{\ell=0}^{L} \sum_{m=1}^{d} \cos\left(\frac{2\pi\ell}{L}(x_m - x'_m)\right) \end{aligned}$$

The kernel trick: Example 4

When $L \to \infty$, even if we cannot compute $\phi(x)$ (since it's a vector of *infinite dimension*), we can still compute inner product:

$$\begin{aligned}\boldsymbol{\phi}_\infty(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{\phi}_\infty(\boldsymbol{x}') &= \int_0^{2\pi} \sum_{m=1}^{d} \cos(\theta(x_m - x'_m))\, d\theta \\ &= \sum_{m=1}^{d} \frac{\sin(2\pi(x_m - x'_m))}{x_m - x'_m}\end{aligned}$$

Again, a simple function of the original features.

Note that when using this mapping in linear regression, we are *learning a weight $\boldsymbol{w}^*$ with infinite dimension!*

Kernel functions

Definition: a function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is called a *kernel function* if there exists a function $\boldsymbol{\phi} : \mathbb{R}^d \to \mathbb{R}^M$ so that for any $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^d$,

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}')$$

Examples we have seen

$$k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^{\mathrm{T}} \boldsymbol{x}')^2$$

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sum_{m=1}^{d} \frac{\sin(2\pi(x_m - x'_m))}{x_m - x'_m}$$

Using kernel functions

Choosing a nonlinear basis ϕ becomes equivalent to choosing a kernel function.

As long as computing the kernel function is more efficient, we should apply the kernel trick.

Gram/kernel matrix becomes:

$$\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} = \begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & k(\boldsymbol{x}_1, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ k(\boldsymbol{x}_2, \boldsymbol{x}_1) & k(\boldsymbol{x}_2, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_2, \boldsymbol{x}_n) \\ \vdots & \vdots & \vdots & \vdots \\ k(\boldsymbol{x}_n, \boldsymbol{x}_1) & k(\boldsymbol{x}_n, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{pmatrix}$$

In fact, k is a kernel if and only if $\boldsymbol{K}$ is positive semidefinite for *any n and any $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$* (**Mercer theorem**).

- useful for proving that a function is not a kernel

Examples which are not kernels

Function

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

is *not a kernel*, why?

If it is a kernel, the kernel matrix for two data points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$:

$$\boldsymbol{K} = \begin{pmatrix} 0 & \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|_2^2 \\ \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|_2^2 & 0 \end{pmatrix}$$

must be positive semidefinite, *but is it?*

Properties of kernels

For any function $f : \mathbb{R}^d \to \mathbb{R}$, $k(\boldsymbol{x}, \boldsymbol{x}') = f(\boldsymbol{x})f(\boldsymbol{x}')$ is a kernel.

If $k_1(\cdot, \cdot)$ and $k_2(\cdot, \cdot)$ are kernels, then the following are also kernels:

- **conical combination**: $\alpha k_1(\cdot, \cdot) + \beta k_2(\cdot, \cdot)$ if $\alpha, \beta \geq 0$
- **product**: $k_1(\cdot, \cdot)k_2(\cdot, \cdot)$
- **exponential**: $e^{k(\cdot,\cdot)}$
- $\cdots$

Verify using the definition of kernel!

Popular kernels

Polynomial kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}' + c)^M$$

for $c \geq 0$ and M is a positive integer.

What is the corresponding ϕ?

Popular kernels

Gaussian kernel or Radial basis function (RBF) kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2}{2\sigma^2}\right) \quad \text{for some } \sigma > 0.$$

What is the corresponding ϕ?

Popular kernels

Gaussian kernel or Radial basis function (RBF) kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2}{2\sigma^2}\right) \quad \text{for some } \sigma > 0.$$

What is the corresponding ϕ?

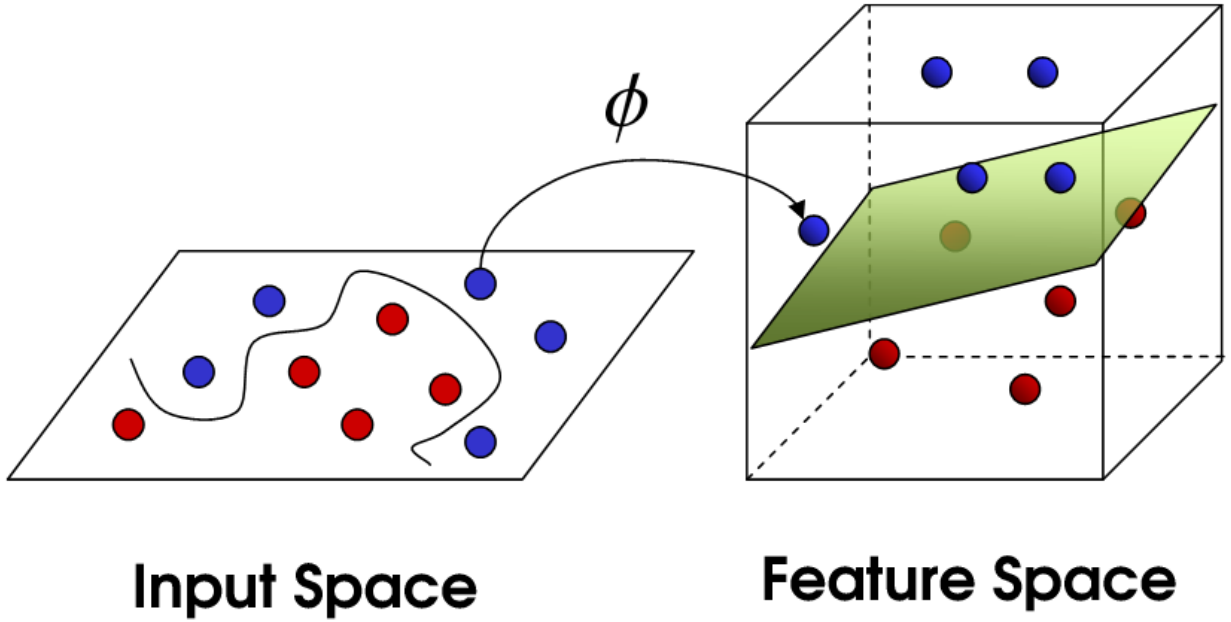
Prediction with kernels

As long as $\boldsymbol{w}^* = \sum_{i=1}^n \alpha_i \boldsymbol{\phi}(\boldsymbol{x}_i)$, prediction on a new example $\boldsymbol{x}$ becomes

$$\boldsymbol{w}^{*\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}) = \sum_{i=1}^n \alpha_i \boldsymbol{\phi}(\boldsymbol{x}_i)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}) = \sum_{i=1}^n \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}).$$

This is known as a **non-parametric method**. Informally speaking, this means that there is no fixed set of parameters that the model is trying to learn (remember $\boldsymbol{w}^*$ could be infinite). Nearest-neighbors is another non-parametric method we have seen.

Classification with kernels



Similar ideas extend to the classification case, and we can predict using $\text{sign}(\boldsymbol{w}^T \boldsymbol{\phi})$.
Data may become linearly separable in the feature space!

Support vector machines (SVMs)

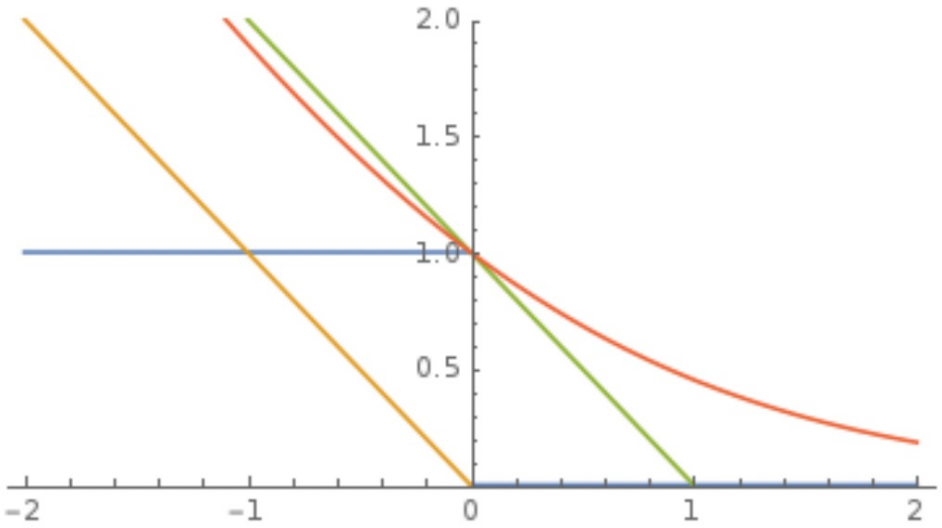
Why study SVM?

- One of the most commonly used classification algorithms
- Allows us to explore the concept of *margins* in classification
- Works well with the kernel trick
- Strong theoretical guarantees

We focus on **binary classification** here.

"Primal" formulation

In one sentence: **linear model with ℓ_2 regularized hinge loss**. Recall:



- perceptron loss $\ell_{\text{perceptron}}(z) = \max\{0, -z\} \rightarrow$ Perceptron
- logistic loss $\ell_{\text{logistic}}(z) = \log(1 + \exp(-z)) \rightarrow$ logistic regression
- hinge loss $\ell_{\text{hinge}}(z) = \max\{0, 1 - z\} \rightarrow$ **SVM**

"Primal" formulation

For a linear model $(\boldsymbol{w}, b)$, this means

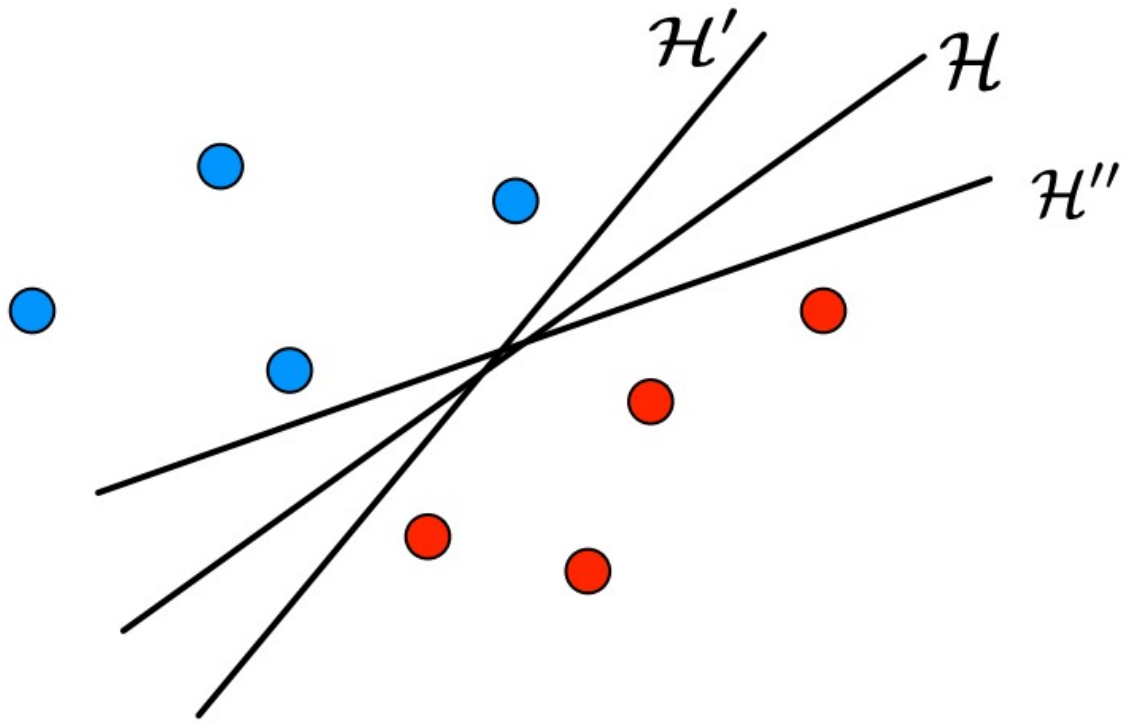
$$\min_{\boldsymbol{w},b} \sum_i \max\left\{0, 1 - y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b)\right\} + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

- recall $y_i \in \{-1, +1\}$
- a nonlinear mapping $\boldsymbol{\phi}$ is applied
- the bias/intercept term b is used explicitly (why?)

So why L2 regularized hinge loss?

Geometric motivation: separable case

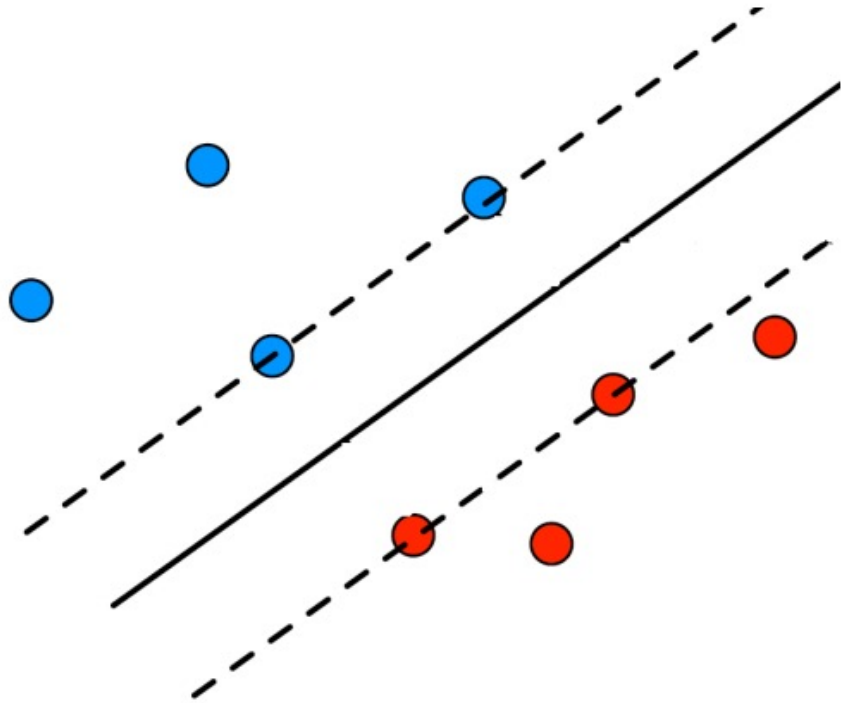
When data is **linearly separable**, there are infinitely many hyperplanes with zero training error:



Which one should we choose?

Geometric motivation: separable case

The further away the separating hyperplane is from the datapoints, the better.



Margins: formalizing this intuition

Distance to hyperplane

What is the **distance** from a point $\boldsymbol{x}$ to a hyperplane $\{\boldsymbol{x} : \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = 0\}$?

Assume the **projection** is $\boldsymbol{x} - \beta \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}$, then

$$0 = \boldsymbol{w}^{\mathrm{T}} \left(\boldsymbol{x} - \beta \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2} \right) + b = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} - \beta \|\boldsymbol{w}\| + b$$

and thus $\beta = \frac{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b}{\|\boldsymbol{w}\|_2}$.

Therefore the distance is

$$\frac{|\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b|}{\|\boldsymbol{w}\|_2}$$

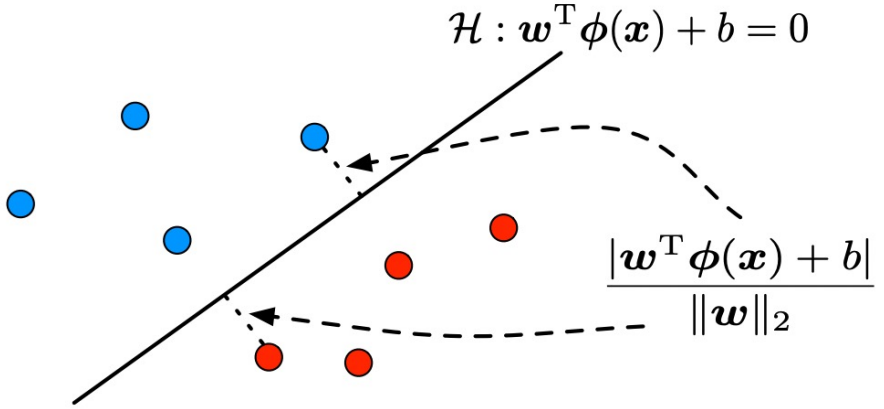
For a hyperplane that correctly classifies $(\boldsymbol{x}, y)$, the distance becomes

$$\frac{y(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b)}{\|\boldsymbol{w}\|_2}$$

Maximizing margin

Margin: the *smallest* distance from all training points to the hyperplane

$$\text{MARGIN OF } (\boldsymbol{w}, b) = \min_i \frac{y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b)}{\|\boldsymbol{w}\|_2}$$



The intuition "**the further away the better**" translates to solving

$$\max_{\boldsymbol{w},b} \min_i \frac{y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b)}{\|\boldsymbol{w}\|_2} = \max_{\boldsymbol{w},b} \frac{1}{\|\boldsymbol{w}\|_2} \min_i y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b)$$

Rescaling

Note: rescaling $(\boldsymbol{w}, b)$ does not change the hyperplane.

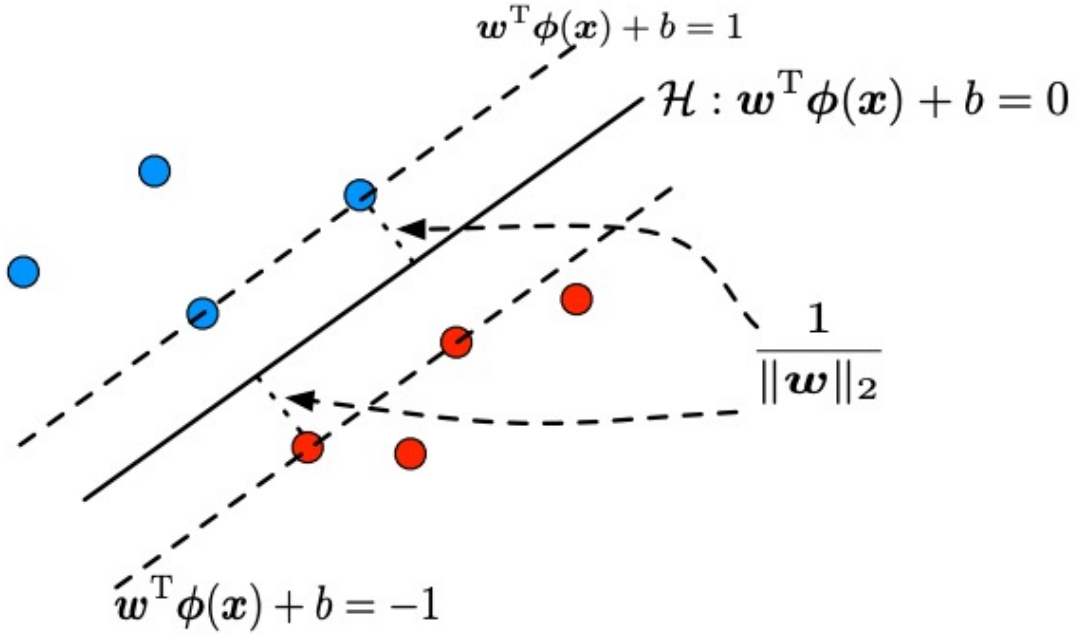
We can thus always scale $(\boldsymbol{w}, b)$ s.t. $\min_i y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b) = 1$

The margin then becomes

MARGIN OF $(\boldsymbol{w}, b)$

$$= \frac{1}{\|\boldsymbol{w}\|_2} \min_i y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b)$$

$$= \frac{1}{\|\boldsymbol{w}\|_2}$$



Summary for separable data

For a separable training set, we aim to solve

$$\max_{\boldsymbol{w},b} \frac{1}{\|\boldsymbol{w}\|_2} \quad \text{s.t.} \quad \min_i y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b) = 1$$

This is equivalent to

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b) \geq 1, \quad \forall\, i \in [n] \end{aligned}$$

SVM is thus also called *max-margin* classifier. The constraints above are called *hard-margin* constraints.

General non-separable case

If data is not linearly separable, the previous constraint

$$y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b) \geq 1, \quad \forall\, i \in [n]$$

is obviously *not feasible*.

To deal with this issue, we relax them to **soft-margin** constraints:

$$y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b) \geq 1 - \xi_i, \quad \forall\, i \in [n]$$

where we introduce **slack variables** $\xi_i \geq 0$.

SVM Primal Formulation

We want ξ_i to be as small as possible too. The objective becomes

$$\begin{aligned} \min_{\boldsymbol{w}, b, \{\xi_i\}} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_i \xi_i \\ \text{s.t.} \quad & y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b) \geq 1 - \xi_i, \quad \forall\, i \in [n] \\ & \xi_i \geq 0, \quad \forall\, i \in [n] \end{aligned}$$

where C is a hyperparameter to balance the two goals.

Equivalent form

The formulation

$$\begin{aligned} \min_{\boldsymbol{w},b,\{\xi_i\}} \quad & C\sum_i \xi_i + \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & 1 - y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b) \le \xi_i, \quad \forall\, i \in [n] \\ & \xi_i \ge 0, \quad \forall\, i \in [n] \end{aligned}$$

is equivalent to

$$\begin{aligned} \min_{\boldsymbol{w},b,\{\xi_i\}} \quad & C\sum_i \xi_i + \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & \max\left\{0, 1 - y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b)\right\} = \xi_i, \quad \forall\, i \in [n] \end{aligned}$$

Equivalent form

$$\min_{\boldsymbol{w},b,\{\xi_i\}} \quad C\sum_i \xi_i + \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

$$\text{s.t.} \quad \max\left\{0, 1 - y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b)\right\} = \xi_i, \quad \forall\, i \in [n]$$

is equivalent to

$$\min_{\boldsymbol{w},b} \; C\sum_i \max\left\{0, 1 - y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b)\right\} + \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

and

$$\min_{\boldsymbol{w},b} \; \sum_i \max\left\{0, 1 - y_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) + b)\right\} + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$$

with $\lambda = 1/C$. *This is exactly minimizing ℓ_2 regularized hinge loss!*

Optimization

$$\begin{aligned} \min_{\boldsymbol{w}, b, \{\xi_i\}} \quad & C \sum_i \xi_i + \frac{1}{2} \|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & 1 - y_i(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_i) + b) \leq \xi_i, \quad \forall\, i \in [n] \\ & \xi_i \geq 0, \quad \forall\, i \in [n] \end{aligned}$$

- it is a convex (in fact, a **quadratic**) problem
- thus can apply any convex optimization algorithms, e.g. SGD
- there are **more specialized and efficient** algorithms
- but usually we apply kernel trick, which requires solving the *dual problem*