

CSCI 567: Machine Learning

Vatsal Sharan
Fall 2022

Lecture 4, Sep 15

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

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

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

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.

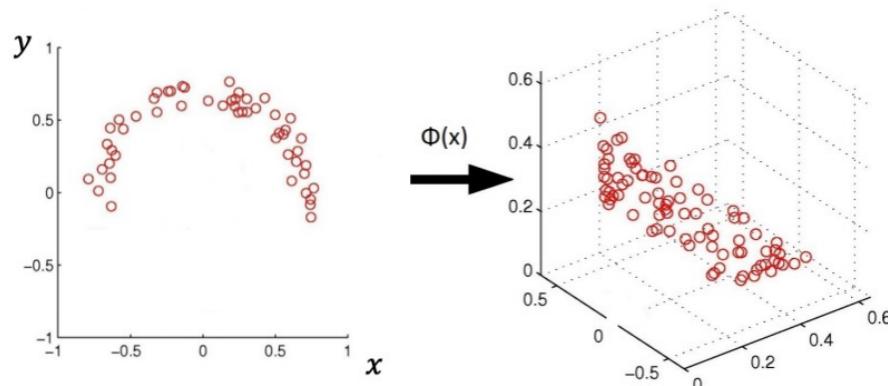
Beyond linear models: nonlinearly transformed features

1. Use a nonlinear mapping

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

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 $d = 1$

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

Learning a linear model in the new space

= learning an *M-degree polynomial model* in the original space

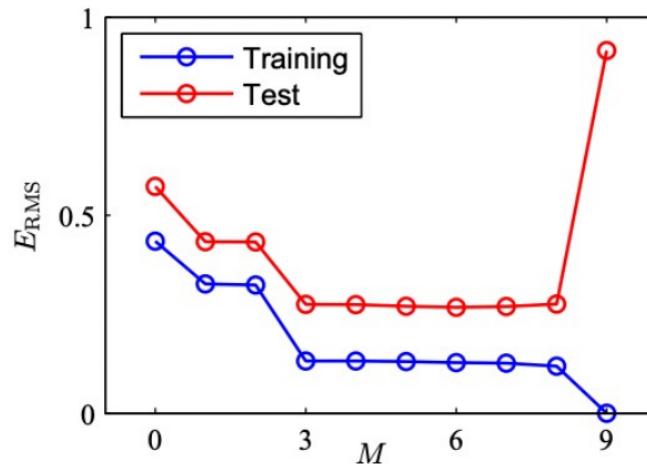
Underfitting and overfitting

$M \leq 2$ 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 \Rightarrow larger gap between training and test error

How to prevent overfitting?

Preventing overfitting: **Regularization**

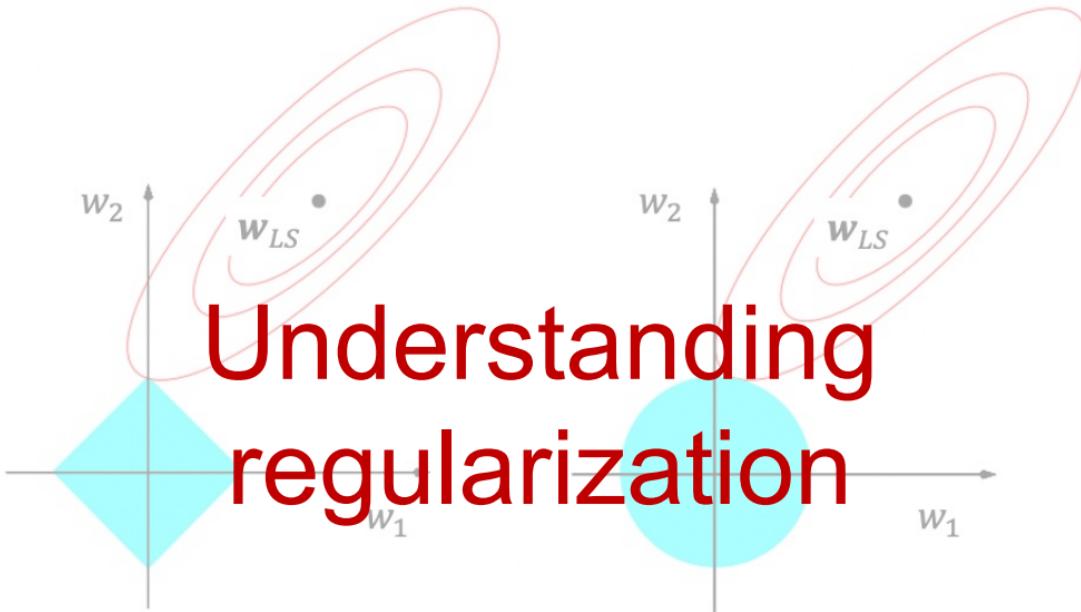
Regularized linear regression: new objective

$$G(\mathbf{w}) = \text{RSS}(\mathbf{w}) + \lambda\psi(\mathbf{w})$$

Goal: find $\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} G(\mathbf{w})$

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

Understanding regularization



ℓ_2 regularization: penalizing large weights

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

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

$$\begin{aligned}\nabla G(\mathbf{w}) &= 2(\mathbf{X}^\top \mathbf{X}\mathbf{w} - \mathbf{X}^\top \mathbf{y}) + 2\lambda\mathbf{w} = 0 \\ \Rightarrow (\mathbf{X}^\top \mathbf{X} + \lambda\mathbf{I})\mathbf{w} &= \mathbf{X}^\top \mathbf{y} \\ \Rightarrow \mathbf{w}^* &= (\mathbf{X}^\top \mathbf{X} + \lambda\mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y}\end{aligned}$$

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

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

Encouraging sparsity: ℓ_0 regularization

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

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

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

Good: "Information-theoretically" great! (Need less data to learn).

Suppose weights in \mathbf{w} are in $\{-W, -W+1, \dots, 0, \dots, W\}$

How many such s -sparse vectors are there in dimensions?

Answer: $\binom{d}{s} \cdot (2W)^s$ possibilities.

ℓ_0 regularization: The good, the bad and the ugly

How much data to learn?

$$\left(\frac{d}{s}\right) \asymp \left(\frac{d}{s}\right)^s$$

About $\log(1/F)$ samples to learn. (using the Theorem from last time,

$$\begin{aligned} \rightarrow \log \left(\binom{d}{s} (2w)^s \right) &= \log \left(\left(\frac{d}{s} \right)^s \right) + \log \left((2w)^s \right) \\ &= \textcircled{s} \log \left(\frac{d}{s} \right) + \textcircled{s} \log (2w) \end{aligned}$$

note that we're ignoring ϵ, s here)

How many free parameters?

every bit is like a parameter
you need to learn

→ choose s co-ordinates : need $\log d$ bits per co-ordinate
 $\Rightarrow s \log d$ in total.

→ choose the value for non-zero coordinates : fix s values $\asymp s \log w$ in total.

ℓ_0 regularization: The good, the bad and the ugly

In contrast, without s -sparsity need about $\approx d$ samples in d -dimensions.

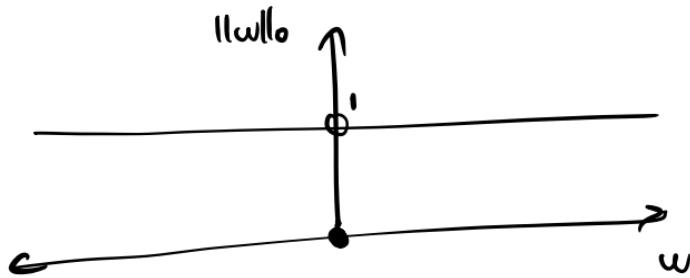
\therefore If $s \ll d$, need much less data to learn!!

Bad: $\|w\|_{l_0}$ is non-convex : ($\|w\|_p, p \leq 1$ is non-convex.

minimizing $b(w) = \sum_{i=1}^n (w^T x_i - y_i)^2 + \lambda \|w\|_0$ is NP-hard : (

ℓ_0 regularization: The good, the bad and the ugly

Ugly: $\|w\|_{\ell_0}$ is highly-discontinuous.



GD has no hope!!

ℓ_1 regularization as a proxy for ℓ_0 regularization

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

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

$\|\mathbf{w}\|_1$ is convex :) can use GD / SGD to solve!

Minimizing $\|\mathbf{w}\|_1$ often suffices to minimize $\|\mathbf{w}\|_0$!

ℓ_1 regularization as a proxy for ℓ_0 regularization

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

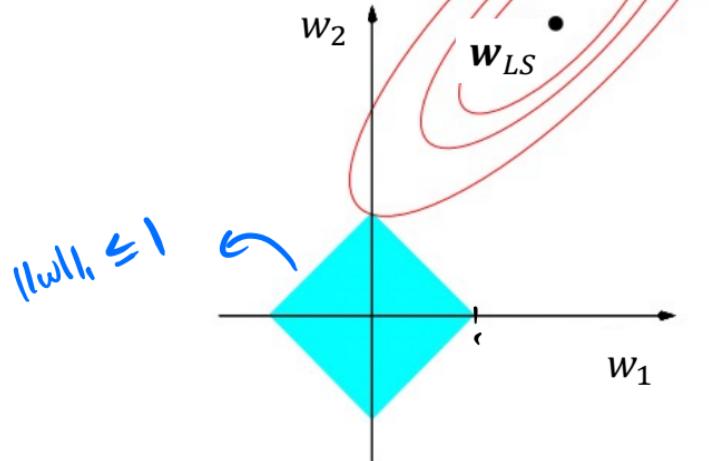
[similar result can also be proven under more general conditions].

(the details of this Theorem are
not important, just focus on the takeaway)

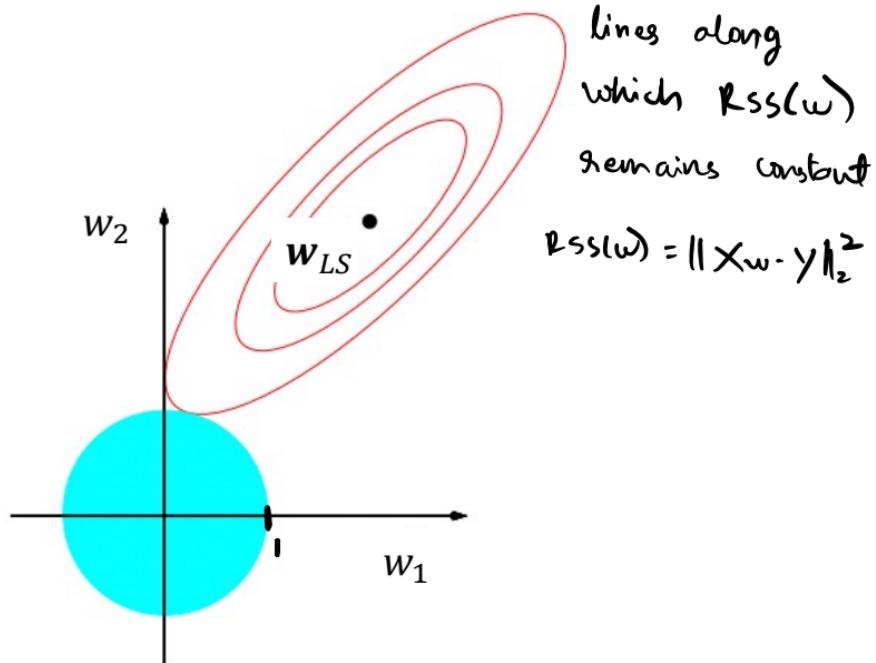
Why does ℓ_1 regularization encourage sparse solutions?

Optimization problem: $\operatorname{argmin}_{\mathbf{w}} \text{RSS}(\mathbf{w})$, subject to $\psi(\mathbf{w}) \leq \beta$

$$\beta = 1$$



$$\Psi(\mathbf{w}) = \|\mathbf{w}\|_1$$



$$\Psi(\mathbf{w}) = \|\mathbf{w}\|_2^2$$

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

Isotropic assumption : $X^T X = I$

① $\Psi(w) = \|w\|_2^2$

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

$$w^* = (X^T X + \lambda I)^{-1} X^T y$$

Now, $X^T X = I \Rightarrow w^* = \begin{pmatrix} 1 \\ 1+\lambda \end{pmatrix} X^T y$

$$w_j^* = \left(\frac{1}{1+\lambda} \right)$$

jth coordinate of w^*

$$\underbrace{x_{(j)}^T y}_{\text{jth row of } X}$$

correlation of jth feature with label

Isotropic informally means

- ① all features have mean 0
- ② all features have variance 1
- ③ features are uncorrelated

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

ℓ_2 regularization “shrinks” the estimated parameters.

Note: when features have unequal variance, ℓ_2 regularization applies similar shrinkage to all of them
 \therefore scaling features can be important.

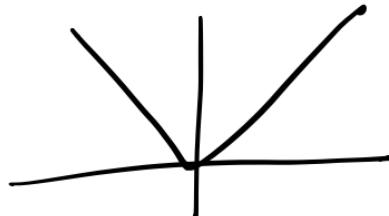
② $\Psi(w) = \|w\|_1,$

$$L(w) = \sum_{i=1}^n (x_i^T w - y_i)^2 + d\|w\|_1,$$

Let's examine the gradient.

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

What is gradient of $|w|$?



$$\frac{\partial |w|}{\partial w} = \begin{cases} 1 & w > 0 \\ -1 & w < 0 \end{cases}$$

At $w=0$, we have a subgradient, ignore for now.

For $w_j \neq 0$
jth coordinate of w

$$\frac{\partial L(w)}{\partial w_j} = 2 \sum_{i=1}^n (x_i^T w - y_i) \underbrace{x_{i,j}}_{j\text{-th coordinate of } x_i} + 1 \operatorname{sign}(w_j)$$

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

$$\frac{\partial L(w)}{\partial w_j} = 2 \sum_{i=1}^n (\tau_{i,j} x_i^\top w) - 2 \sum_{i=1}^n \tau_{i,j} y_i + \lambda \text{sign}(w_j)$$

$$= 2 \sum_{i=1}^n \tau_{i,j} \begin{array}{c} x_i \\ \hline \end{array} - 2 x_{(i)}^\top y + \lambda \text{sign}(w_j)$$

This follows
as $x^\top x = I$

$$= 2 w_j - 2 x_{(i)}^\top y + \lambda \text{sign}(w_j)$$

$$\therefore \text{GD steps : } w_j \leftarrow w_j - \eta (2(w_j - x_{(i)}^\top y) + \lambda \text{sign}(w_j))$$

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

Let's understand the gradient.

First, without ℓ_1 regularization,

$$w_j \leftarrow w_j - 2\eta (w_j - x_{(i)}^\top y)$$

With ℓ_1 regularization: GD always has a shift of $-n\lambda \text{sign}(w_j)$
which pushes towards 0.

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

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

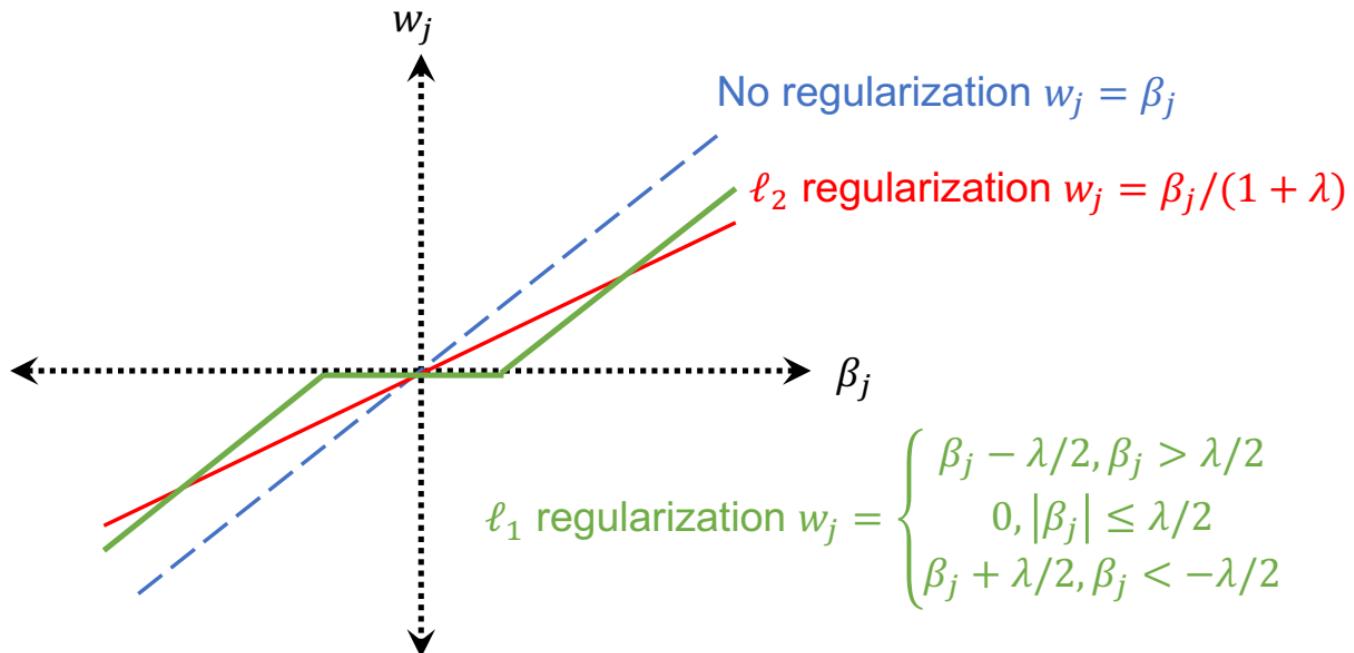
Using subgradients, we can show that for the ℓ_1 regularized case:

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

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

Summary: Isotropic case ($X^T X = I$).

Let $\beta_j = X_{(j)}^T \mathbf{y}$



Implicit regularization

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

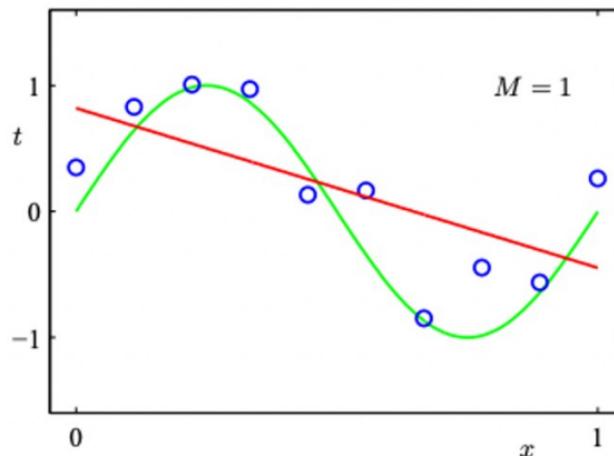
In many cases, the optimization algorithm we use can themselves act as regularizers, favoring some solutions over others.

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.

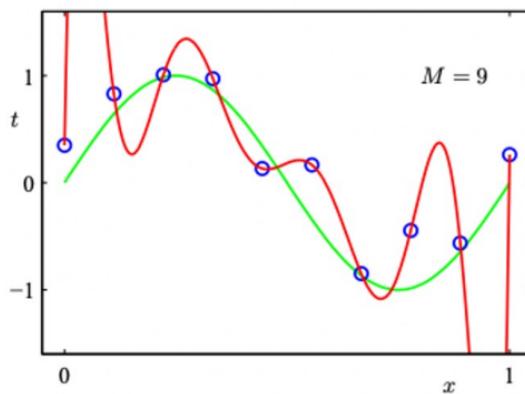


$\sin(x)$ fitting example we saw in Lec 3

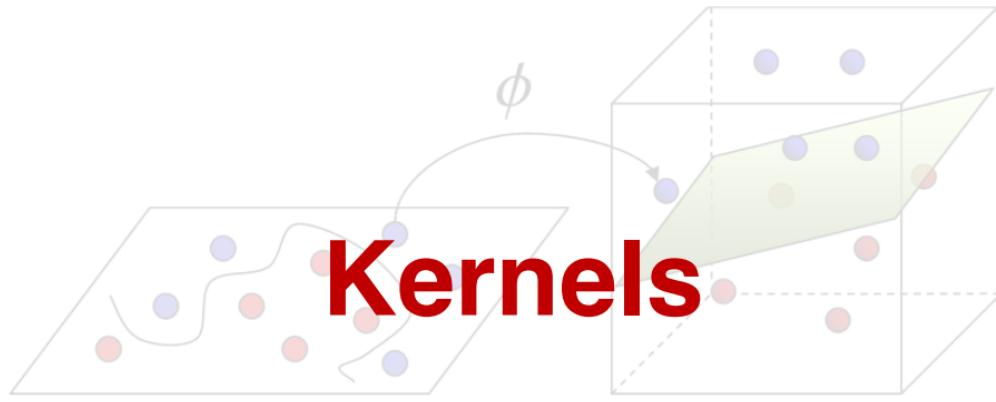
Bias-variance tradeoff

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



Input Space

Feature Space

Kernels

Motivation

Recall the nonlinear function map for linear regression:

1. Use a nonlinear mapping

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

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:

$$\begin{aligned}\mathbf{w}^* &= \underset{\mathbf{w}}{\operatorname{argmin}} F(\mathbf{w}) \\ &= \underset{\mathbf{w}}{\operatorname{argmin}} (\|\Phi\mathbf{w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_2^2) \\ &= (\Phi^T\Phi + \lambda\mathbf{I})^{-1} \Phi^T\mathbf{y}\end{aligned}$$

$$\Phi = \begin{pmatrix} \phi(\mathbf{x}_1)^T \\ \phi(\mathbf{x}_2)^T \\ \vdots \\ \phi(\mathbf{x}_n)^T \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

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 $F(\mathbf{w}) = \|\Phi\mathbf{w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_2^2$ to be 0:

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

scale by $1/\lambda$

w* = M

Φ^T M

we know

$$\mathbf{w}^* = \frac{1}{\lambda} \Phi^T(\mathbf{y} - \Phi\mathbf{w}^*) = \Phi^T\alpha = \sum_{i=1}^n \alpha_i \phi(x_i)$$

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

This calculation does not show what α should be, but ignore that for now.

Why is this helpful?

Assuming we know α , the prediction of w^* on a new example x is

$$w^{*T} \phi(x) = \sum_{i=1}^n \alpha_i \phi(x_i)^T \phi(x) \leftarrow \Sigma(\alpha_i \phi(x_i))^T \phi(x)$$

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 α is first!

Solving for α , Step 1: Kernel matrix

Plugging in $w = \Phi^T \alpha$ into $F(w)$ gives

$$H(\alpha) = F(\Phi^T \alpha)$$

$$= \|\Phi\Phi^T \alpha - y\|_2^2 + \lambda \|\Phi^T \alpha\|_2^2$$

$$= \|K\alpha - y\|_2^2 + \lambda \alpha^T K \alpha \quad (K = \Phi\Phi^T \in \mathbb{R}^{n \times n})$$

$$= (\Phi^T \alpha)^T (\Phi^T \alpha) = \alpha^T \Phi \Phi^T \alpha$$

K is called **Gram matrix** or **kernel matrix** where the (i, j) -th entry is

$$K_{(i,j)} = \phi(x_i)^T \phi(x_j)$$

Kernel matrix: Example

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

Gram/Kernel matrix

$$\begin{aligned} K &= \begin{pmatrix} \phi(x_1)^T \phi(x_1) & \phi(x_1)^T \phi(x_2) & \phi(x_1)^T \phi(x_3) \\ \phi(x_2)^T \phi(x_1) & \phi(x_2)^T \phi(x_2) & \phi(x_2)^T \phi(x_3) \\ \phi(x_3)^T \phi(x_1) & \phi(x_3)^T \phi(x_2) & \phi(x_3)^T \phi(x_3) \end{pmatrix} \\ &= \begin{pmatrix} 4 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 4 \end{pmatrix} \end{aligned}$$

Kernel matrix vs Covariance matrix

	dimensions	entry (i, j)	property
$\Phi \Phi^T$	$n \times n$	$\phi(x_i)^T \phi(x_j)$	both are symmetric
$\Phi^T \Phi$	$M \times M$	$\sum_{k=1}^M \phi(x_k)_i \phi(x_k)_j$ <i>ith coordinate of feature</i>	& positive semi-definite (psd)

Why are they psd?

Any matrix $A = UU^T$ is psd.

$$(x^T A x = x^T U U^T x = \|U^T x\|_2^2 \geq 0)$$

Solving for α , Step 2: Minimize the dual

Minimize (the so-called *dual formulation*)

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

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

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

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

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

Exercise: *are there other minimizers? and are there other \mathbf{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 $(\Phi\Phi^T + \lambda I)^{-1}$ can be more efficient than computing $(\Phi^T\Phi + \lambda I)^{-1}$ when $n \leq M$.

Can solve in $O(d^3)$ time

$n+n$ dimensional

$M+M$ dimensional

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

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

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

The kernel trick: Example 1

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

$$\textcolor{red}{x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}} \quad \phi(x) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}$$

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

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

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

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

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_d \end{pmatrix} \quad \phi_{\theta}(\mathbf{x}) = \begin{pmatrix} \cos(\theta x_1) \\ \sin(\theta x_1) \\ \vdots \\ \cos(\theta x_m) \\ \sin(\theta x_m) \end{pmatrix}$$

What is the inner product between $\phi_{\theta}(\mathbf{x})$ and $\phi_{\theta}(\mathbf{x}')$?

$$\begin{aligned} \phi_{\theta}(\mathbf{x})^T \phi_{\theta}(\mathbf{x}') &= \sum_{m=1}^d \cos(\theta x_m) \cos(\theta x'_m) + \sin(\theta x_m) \sin(\theta x'_m) \\ &= \sum_{m=1}^d \cos(\theta(x_m - x'_m)) \quad (\text{trigonometric identity}) \end{aligned}$$

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 $\phi_L : \mathbb{R}^d \rightarrow \mathbb{R}^{2d(L+1)}$ for some integer L :

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

Θ varies from
 $(0, \frac{2\pi}{L}, \dots, 2\pi)$

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

$$\begin{aligned} \phi_L(\mathbf{x})^\top \phi_L(\mathbf{x}') &= \sum_{\ell=0}^L \phi_{\frac{2\pi\ell}{L}}(\mathbf{x})^\top \phi_{\frac{2\pi\ell}{L}}(\mathbf{x}') \\ &= \sum_{\ell=0}^L \sum_{m=1}^d \cos\left(\frac{2\pi\ell}{L}(x_m - x'_m)\right) \end{aligned}$$

The kernel trick: Example 4

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

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

Change order of summation & integral

Again, a simple function of the original features.

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

Kernel functions

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

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}')$$

Examples we have seen

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}')^2$$

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

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:

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

$\overset{\phi(\mathbf{x}_1)^T \phi(\mathbf{x}_1)}{\nearrow}$ $\overset{\phi(\mathbf{x}_n)^T \phi(\mathbf{x}_n)}{\nearrow}$

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

- useful for proving that a function is not a kernel

Examples which are not kernels

Function

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

is *not a kernel*, why?

If it is a kernel, the kernel matrix for two data points \mathbf{x}_1 and \mathbf{x}_2 : *this entry is $\|\mathbf{x}-\mathbf{x}'\|_2^2 = 0$*

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

must be positive semidefinite, *but is it?*

$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ is not PSD. Why?

$$\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} -1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = -2$$

Properties of kernels

For any function $f : \mathbb{R}^d \rightarrow \mathbb{R}$, $k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})f(\mathbf{x}')$ is a kernel. $\xrightarrow{\text{What is } \phi?}$
 $\phi : \mathbb{R}^d \rightarrow \mathbb{R}$, $\phi(\mathbf{x}) = f(\mathbf{x})$

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 $\alpha, \beta \geq 0$ $\xrightarrow{\text{What is } \phi?}$

- **product:** $k_1(\cdot, \cdot)k_2(\cdot, \cdot)$ $\xrightarrow{\text{HW2}}$

- **exponential:** $e^{k(\cdot, \cdot)}$

- ...

ϕ_1 : map for k_1

ϕ_2 : map for k_2

ϕ' : map for $\alpha k_1(\cdot, \cdot) + \beta k_2(\cdot, \cdot)$

$\leftarrow \alpha \phi_1(\cdot, \cdot) + \beta \phi_2(\cdot, \cdot)$

Exercise: find ϕ'

Verify using the definition of kernel!

Popular kernels

Polynomial kernel

$$k(x, x') = (x^T x' + c)^M$$

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

polynomial in original i.p.

What is the corresponding ϕ ?

$c=0, M=2$, we saw earlier

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

The case of larger M can be obtained by applying this repeatedly.

Popular kernels

Gaussian kernel or Radial basis function (RBF) kernel

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

What is the corresponding ϕ ?

$$k(\mathbf{x}, \mathbf{x}') = \underbrace{\exp\left(-\frac{\|\mathbf{x}\|_2^2}{2\sigma^2}\right)}_{\text{focus on this}} \cdot \exp\left(-\frac{\|\mathbf{x}'\|_2^2}{2\sigma^2}\right) \cdot \exp\left(\frac{\mathbf{x}^T \mathbf{x}'}{\sigma^2}\right)$$

$$k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x}) f(\mathbf{x}')$$

$$\text{where } f(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x}\|_2^2}{2\sigma^2}\right)$$

transformation for the product.

Popular kernels

Gaussian kernel or Radial basis function (RBF) kernel

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

What is the corresponding ϕ ?

$$e^{\mathbf{x}} = 1 + \mathbf{x} + \frac{\mathbf{x}^2}{2!} + \frac{\mathbf{x}^3}{3!} + \dots$$

$$\exp\left(\frac{\mathbf{x}^\top \mathbf{x}'}{\sigma^2}\right) = 1 + \frac{\mathbf{x}^\top \mathbf{x}'}{\sigma^2} + \underbrace{\frac{1}{2!} \frac{(\mathbf{x}^\top \mathbf{x}')^2}{(\sigma^2)^2} + \frac{1}{3!} \left(\frac{\mathbf{x}^\top \mathbf{x}'}{\sigma^2}\right)^3}_{\dots} + \dots$$

each of these is a polynomial kernel

∞ dimensional feature space!

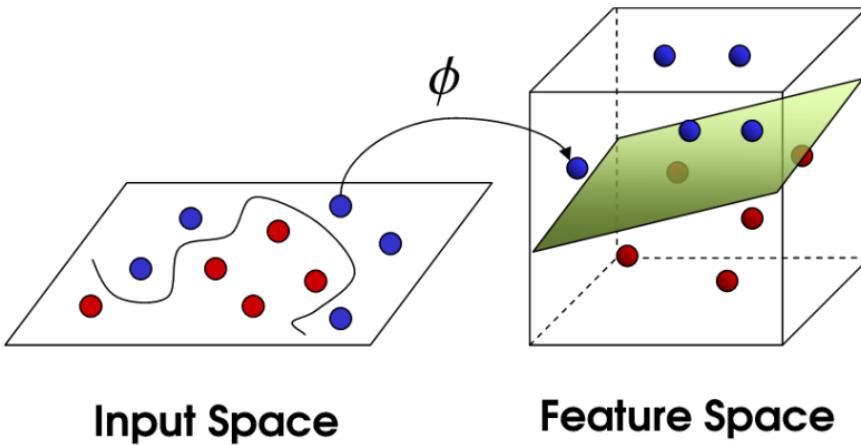
Prediction with kernels

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

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

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 \mathbf{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 $\text{sign}(\mathbf{w}^T \phi)$. Data may become linearly separable in the feature space!