CSCI 567: Machine Learning

Vatsal Sharan

Spring 2024

Lecture 3, January 26

USC University of Southern California

Administrivia

- HW1 due in less than 2 weeks (2/7 at midnight).
- Each student gets 2 late days in total (late days get subtracted for each member for team submitting late).
- Max 1 late day per HW.
- Peer to peer (P2P) mentoring sessions for HWs led by course producers (in addition to OHs) --- see course calendar

Recap

Supervised learning in one slide

Loss function: What is the right loss function for the task?

Representation: What class of functions should we use?

Optimization: How can we efficiently solve the empirical risk minimization problem?

Generalization: Will the predictions of our model transfer gracefully to unseen examples?

All related! And the fuel which powers everything is data.

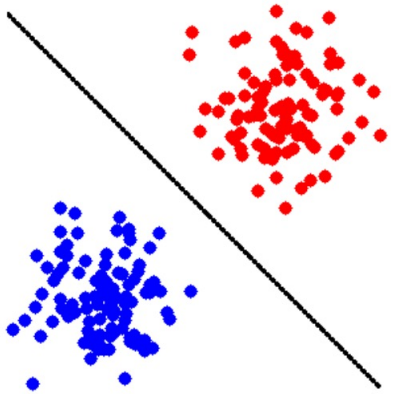
Summary: **Optimization** methods

- **GD/SGD** is a first-order optimization method.
- GD/SGM coverages to a stationary point. For convex objectives, this is all we need. For nonconvex objectives, it is possible to get stuck at local minimizers or “bad” saddle points (random initialization escapes “good” saddle points).
- **Newton’s method** is a second-order optimization method.
- Newton’s method has a much faster convergence rate, but each iteration also takes much longer. Usually for large scale problems, GD/SGD and their variants are the methods of choice.

Linear classifiers

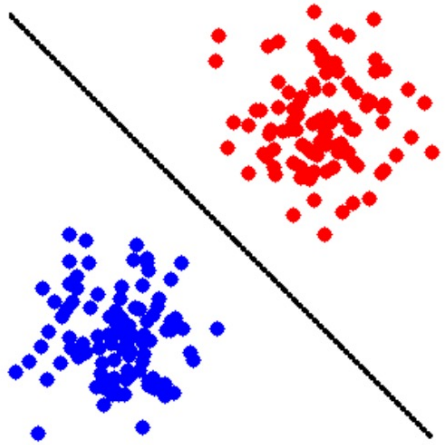
Binary classification:

- input (feature vector): $\boldsymbol{x} \in \mathbb{R}^{\mathsf{d}}$
- output (label): $y \in \{-1, +1\}$.
- goal: learn a mapping $f : \mathbb{R}^{\mathsf{d}} \to \{-1, +1\}$



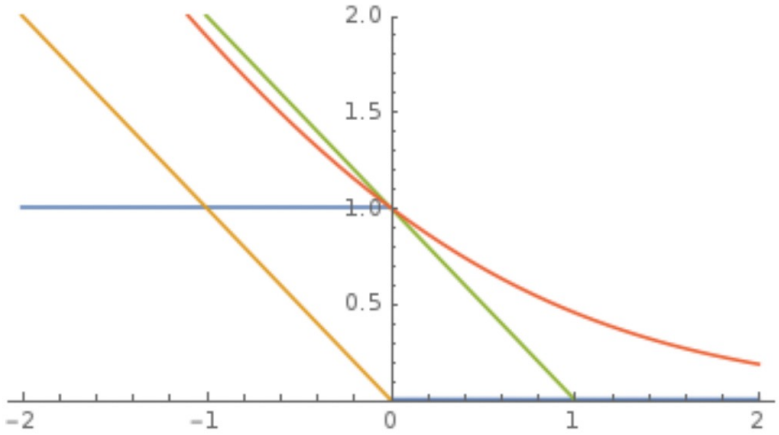
Representation

Definition: The **function class of separating hyperplanes** is defined as $\mathcal{F} = \{f(\boldsymbol{x}) = sign(\boldsymbol{w}^T\boldsymbol{x}) : \boldsymbol{w} \in \mathbb{R}^d\}$.



Loss function

Use a **convex surrogate loss**



- perceptron loss $\ell_{\text{perceptron}}(z) = \max\{0, -z\}$ (used in Perceptron)
- hinge loss $\ell_{\text{hinge}}(z) = \max\{0, 1 - z\}$(used in SVM and many others)
- logistic loss $\ell_{\text{logistic}}(z) = \log(1 + \exp(-z))$ (used in logistic regression; the base of $\log$ doesn't matter)

Optimization

Empirical risk minimization (ERM) problem:

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}\in\mathbb{R}^{\mathsf{d}}}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} \ell(y_i \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_i)$$

Solve using a suitable optimization algorithm:

- **GD:** $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla F(\boldsymbol{w})$
- **SGD:** $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \tilde{\nabla} F(\boldsymbol{w})$ $\qquad (\mathbb{E}[\tilde{\nabla} F(\boldsymbol{w})] = \nabla F(\boldsymbol{w}))$
- **Newton:** $\boldsymbol{w} \leftarrow \boldsymbol{w} - \left(\nabla^2 F(\boldsymbol{w})\right)^{-1} \nabla F(\boldsymbol{w})$

Maximum likelihood estimation

What we observe are labels, not probabilities.

Take a **probabilistic view**

- assume data is independently generated in this way by some $\boldsymbol{w}$
- perform Maximum Likelihood Estimation (MLE)

Specifically, what is the probability of seeing labels $y_1, \cdots, y_n$ given $\boldsymbol{x_1}, \cdots, \boldsymbol{x_n}$, as a function of some $\boldsymbol{w}$?

$$P(\boldsymbol{w}) = \prod_{i=1}^{N} \mathbb{P}(y_i \mid \boldsymbol{x_i}; \boldsymbol{w})$$

MLE: find $\boldsymbol{w}^*$ that **maximizes the probability** $P(\boldsymbol{w})$

Minimizing logistic loss is exactly doing MLE for the sigmoid model!

Generalization

Reviewing definitions

- Input space: $\mathcal{X}$
- Output space: $\mathcal{Y}$
- Predictor: $f(\boldsymbol{x}) : \mathcal{X} \to \mathcal{Y}$
- Distribution D over $(\boldsymbol{x}, y)$.
- Let D^n denote the distribution of n samples $\{(\boldsymbol{x}_i, y_i), i \in [n]\}$ drawn i.i.d. from D. → independent & identically distributed
- Risk of a predictor $f(\boldsymbol{x})$ is $R(f) = \mathbb{E}_{(\boldsymbol{x},y)\sim D}\Big[\ell(f(\boldsymbol{x}), y)\Big]$
- Consider the 0-1 loss, $\ell(f(\boldsymbol{x}, y)) = \mathbb{1}(f(\boldsymbol{x}) \neq y)$.

The analysis we'll do could also help you solve Problem 2 on HW1.

Assumptions for today's theory

Finite sized function class.

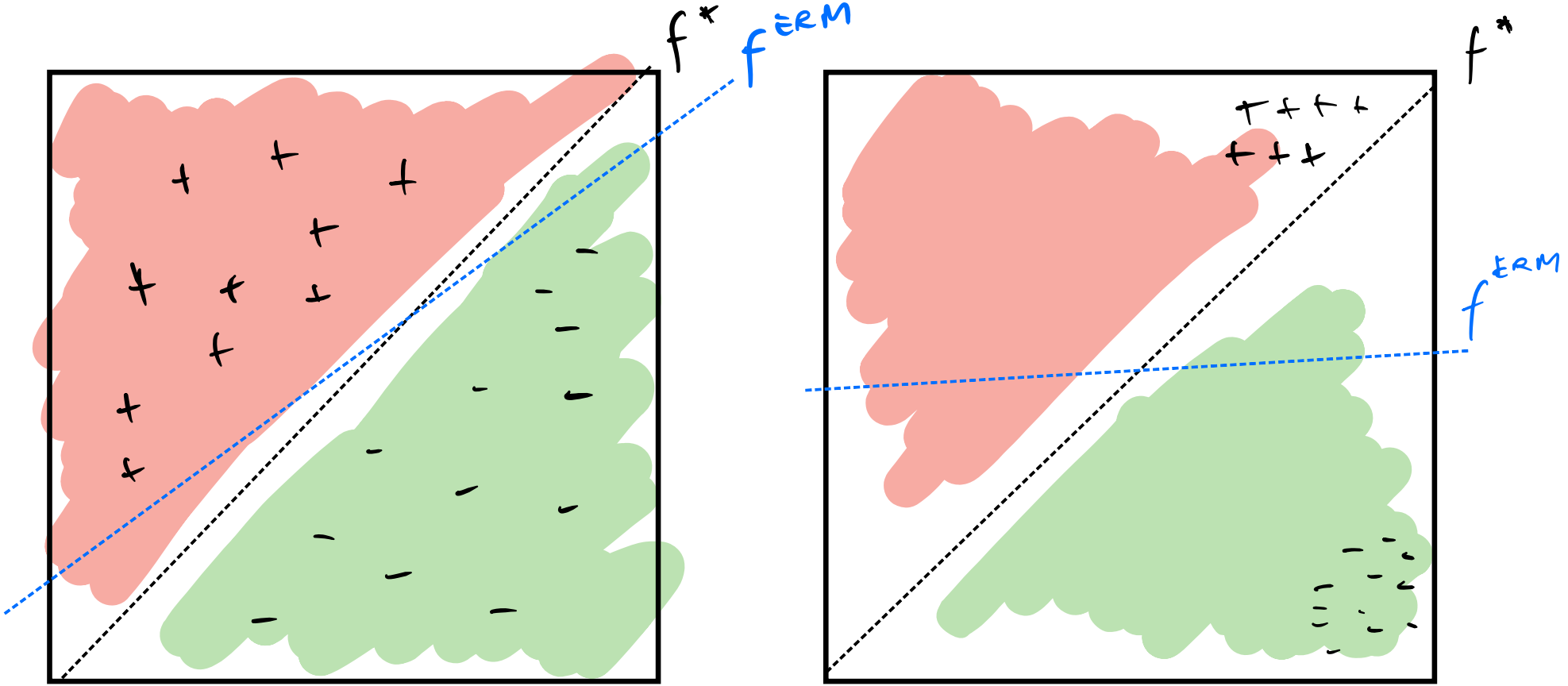
Def: A function class $\mathcal{F}$ is finite-sized if $|\mathcal{F}|$ is finite.

e.g. $\mathcal{F} = \{ f(x) = \text{sign}(w^T x) : w \in \{-1, 0, 1\}^d \}$

$|\mathcal{F}| = 3^d$

Realizability: There exists $f^* \in \mathcal{F}$ s.t. $y = f^*(x) \; \forall \, x \in \mathcal{X}$.

Intuition: When does ERM generalize?



Distribution over $X = (t_1, t_2)$ is uniform in box → same here

Theorem: Let $\mathcal{F}$ be a function class with size $|\mathcal{F}|$. Let $y = f^*(x)$ for some $f^* \in \mathcal{F}$. Suppose we get a training set $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ of size n drawn iid from the data distribution D. Let

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

If $n \geq \frac{\ln(|\mathcal{F}|/\delta)}{\varepsilon}$, then with probability $(1-\delta)$ over $\{(x_i, y_i), i \in [n]\}$, $R(f_S^{ERM}) \leq \varepsilon$ (for constants ε, δ).

e.g. if $\varepsilon = 0.1$, $\delta = 0.1$, then with $n \geq 10 \ln(|\mathcal{F}| \cdot 10)$ samples, then with probability 0.9, $R(f_S^{ERM}) \leq 0.1$.

Proof. Note that there exists $f^* \in \mathcal{F}$ s.t. $R(f^*) = 0$.

Let $\mathcal{F}_{bad} = \{ f \in \mathcal{F} : R(f) > \varepsilon \}$

Goal ①: What is the probability of "getting tricked" by one fixed $f \in \mathcal{F}_{bad}$?

Consider some $f' \in \mathcal{F}_{bad}$.

$$\Pr_{S \sim D^n}\left[f' \text{ is an ERM} \right] \quad (f' \text{ gets } \hat{R}_S(f') = 0)$$

$$= \Pr_{S \sim D^n}\left[\{ \forall i \in \{1, \dots, n\}, f'(x_i) = f^*(x_i) \} \right]$$

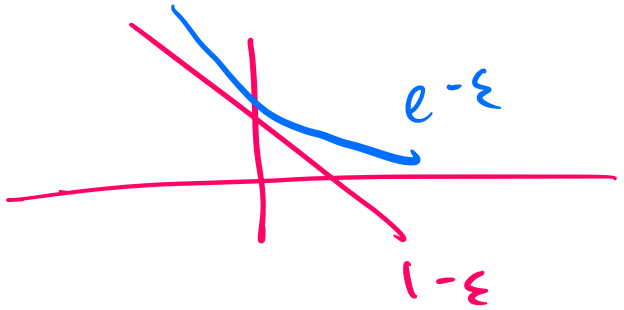
$$= \prod_{i=1}^{n} \Pr_{x_i \sim D}\left[f'(x_i) = f^*(x_i) \right]$$

(i.i.d assumption, $\Pr[E_1 \wedge E_2] = \Pr[E_1]\Pr[E_2]$ if E_1, E_2 independent)

$$\leq \prod_{i=1}^{n} (1 - \varepsilon) \qquad (f' \in \mathcal{F}_{bad})$$

$$\leq \prod_{i=1}^{n} e^{-\varepsilon} \qquad (\text{since } 1-\varepsilon \leq e^{-\varepsilon}$$

$$= e^{-\varepsilon n}$$



Goal (2): What is the probability of being tricked by any $f \in F_{bad}$.

Union bound

$$Pr_{S\sim D^n}\left[\bigcup_{f\in F_{bad}} \{f \text{ is an ERM}\}\right]$$

$$\leq \sum_{f\in F_{bad}} Pr[f \text{ is an ERM}) \quad — \quad (*)$$

Union bound: $P_S[E_1 \cup E_2 \cup E_3] \leq \sum_{i=1}^{3} P_S(E_i)$

Think of each point here as a training set S

E_2 is set of all training sets for which some bad hypothesis $f_2 \in F_{bad}$ is picked

From Goal (1) $(*) \leq \sum_{f \in F_{bad}} e^{-\varepsilon n} = |F_{bad}| e^{-\varepsilon n}$

$\leq |F| e^{-\varepsilon n}$

$\therefore$ if $n \geq \frac{1}{\varepsilon}\left(\ln(|F|) + \ln(1/\delta)\right)$

then $Pr_{S\sim D^n}\left[\bigcup_{f \in F_{bad}} \{f \text{ is an ERM}\}\right] \leq \delta$

$\therefore$ if $n \geq \frac{\ln(|F|/\delta)}{\varepsilon}$, w.p. $(1-\delta)$ $f_S^{ERM} \notin F_{bad}$

$\Rightarrow R(f_S^{ERM}) \leq \varepsilon$.

Here $\hat{R}_S(f_S^{ERM}) = 0$ $\therefore$ generalization gap $= R(f_S^{ERM})$.

Relaxing our assumptions

- We assumed that the function class is finite-sized. Results can be extended to **infinite function classes** (such as separating hyperplanes).
- We considered 0-1 loss. Can extend to **real-valued loss** (such as for regression).
- We assumed realizability. Can prove similar theorem which guarantees small generalization gap **without realizability** (but with an ϵ^2 instead of ϵ in the denominator). This is called agnostic learning.

Rule of thumb for generalization

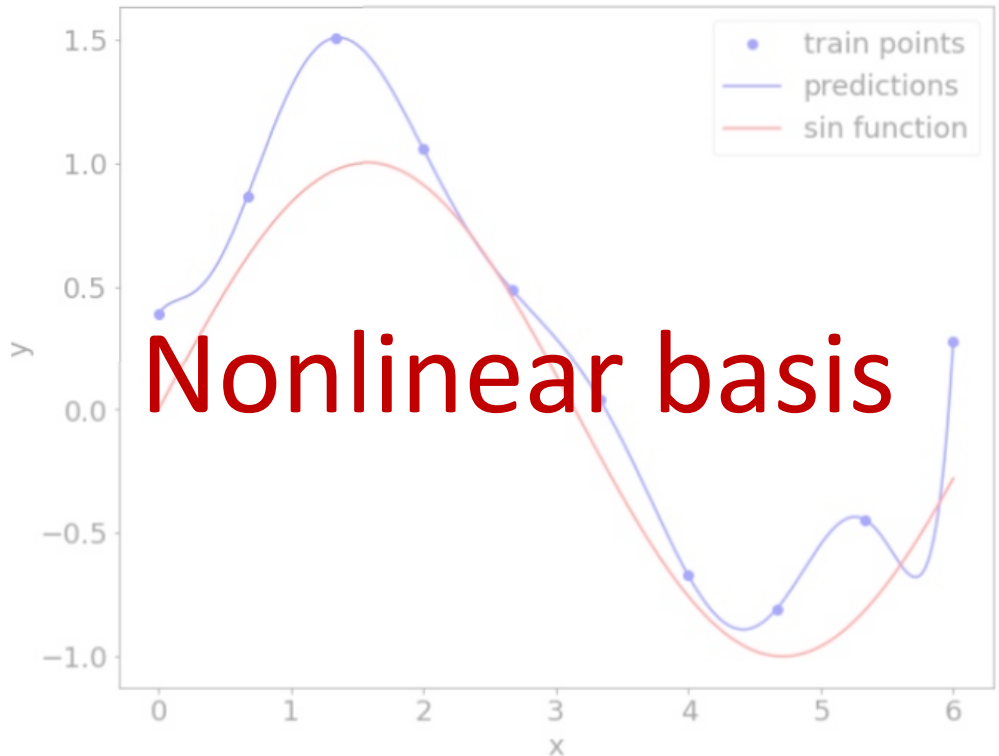
Suppose the functions f in our function class $\mathcal{F}$ have d parameters which can be set. Assume we discretize these parameters so they can each take 3 possible values $\{-1,0,+1\}$. How much data do we need to have small generalization gap?

$$|\mathcal{F}| = 3^d$$

$\therefore$ generalization gap is at most ε (with probability $1-\delta$)

with $n \geq \frac{\ln(|\mathcal{F}|/\delta)}{\varepsilon}$ samples $= \frac{d \ln(3/\delta)}{\varepsilon}$ samples

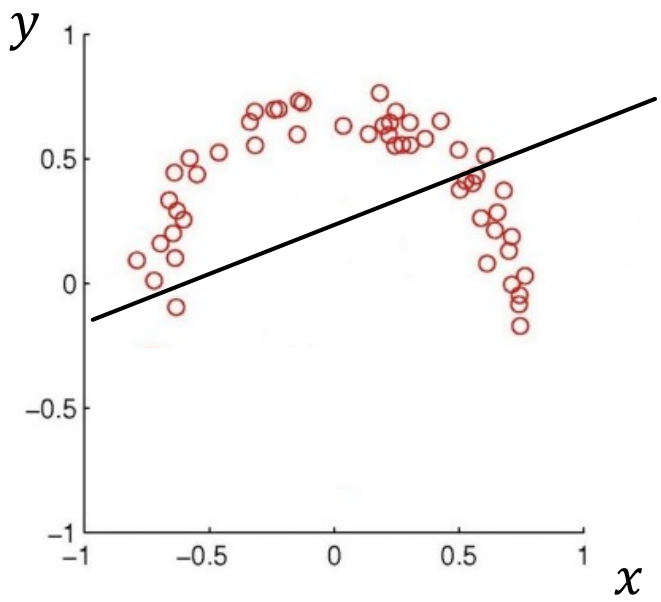
A useful rule of thumb: to guarantee generalization, make sure that your training data set size n is at least linear in the number d of free parameters in the function that you're trying to learn.



Nonlinear basis

What if a linear model is not a good fit?

Let's go back to the regression setup (output $\mathcal{Y} \in R$).
A linear model could be a bad fit for the following data:



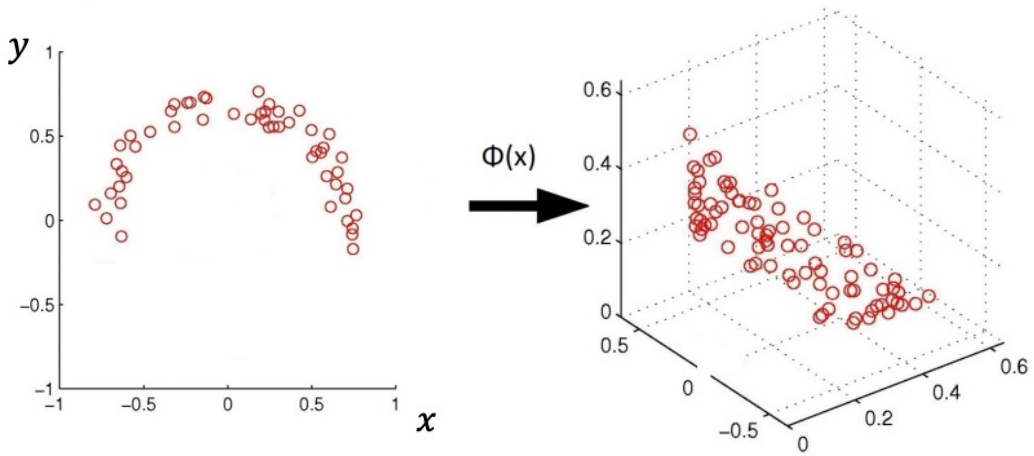
A solution: nonlinearly transformed features

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).



A solution: nonlinearly transformed features

1. Use a nonlinear mapping

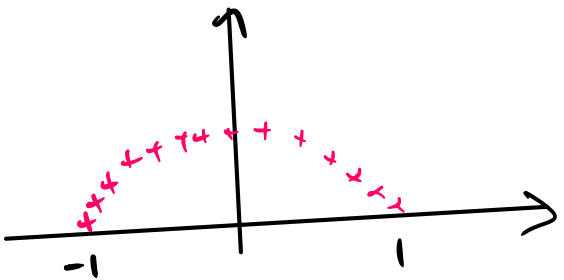
$$\boldsymbol{\phi}(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^d \rightarrow \boldsymbol{z} \in \mathbb{R}^M$$

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).

E.g. consider $y = 1 - x^2$, $x \in [-1, 1]$

$$\phi(x) = \begin{pmatrix} 1 \\ x \\ x^2 \end{pmatrix} \in \mathbb{R}^3$$



$$y = w^T \phi(x)$$

if $w = (1, 0, -1)$

then $y = 1 - x^2$

Regression with nonlinear basis

Model: $f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$ where $\boldsymbol{w} \in \mathbb{R}^M$

Objective:

$$\mathrm{RSS}(\boldsymbol{w}) = \sum_{i=1}^{n} \left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i) - y_i\right)^2$$

Similar least square solution:

$$\boldsymbol{w}^* = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y} \quad \text{where} \quad \boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1)^{\mathrm{T}} \\ \boldsymbol{\phi}(\boldsymbol{x}_2)^{\mathrm{T}} \\ \vdots \\ \boldsymbol{\phi}(\boldsymbol{x}_n)^{\mathrm{T}} \end{pmatrix} \in \mathbb{R}^{n \times M}$$

Example

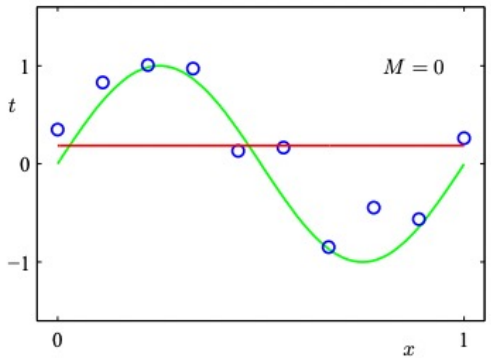
Polynomial basis functions for $d = 1$

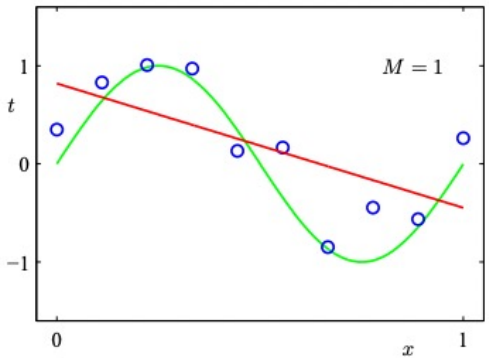
$$\boldsymbol{\phi}(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^M \end{bmatrix} \quad \Rightarrow \quad f(x) = w_0 + \sum_{m=1}^{M} w_m x^m$$

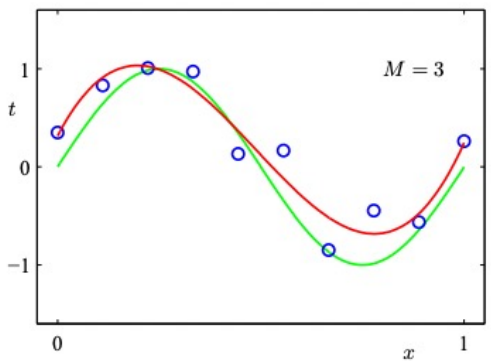
Learning a linear model in the new space
= learning an *M-degree polynomial model* in the original space

Example

Fitting a noisy sine function with a polynomial ($M = 0, 1,$ or 3):







See Colab notebook

Why nonlinear?

Can I use a fancy **linear feature map**?

$$\boldsymbol{\phi}(\boldsymbol{x}) = \begin{bmatrix} x_1 - x_2 \\ 3x_4 - x_3 \\ 2x_1 + x_4 + x_5 \\ \vdots \end{bmatrix} = \boldsymbol{A}\boldsymbol{x} \quad \text{for some } \boldsymbol{A} \in \mathbb{R}^{\mathsf{M} \times d}$$

Why nonlinear?

Can I use a fancy **linear feature map**?

$$\boldsymbol{\phi}(\boldsymbol{x}) = \begin{bmatrix} x_1 - x_2 \\ 3x_4 - x_3 \\ 2x_1 + x_4 + x_5 \\ \vdots \end{bmatrix} = \boldsymbol{A}\boldsymbol{x} \quad \text{for some } \boldsymbol{A} \in \mathbb{R}^{\mathsf{M} \times d}$$

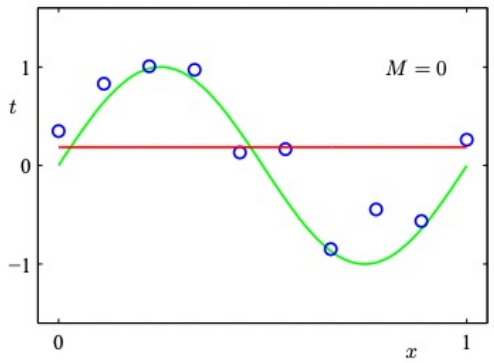
No, it basically *does nothing* since

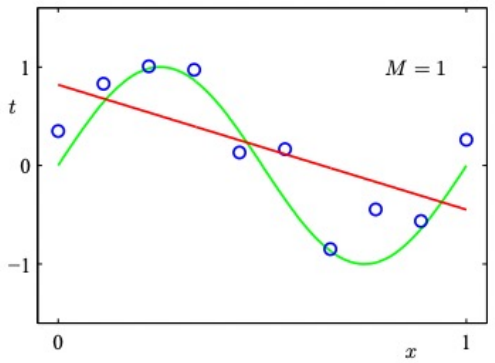
$$\min_{\boldsymbol{w} \in \mathbb{R}^{\mathsf{M}}} \sum_i \left(\boldsymbol{w}^{\mathsf{T}} \boldsymbol{A} \boldsymbol{x}_i - y_i\right)^2 = \min_{\boldsymbol{w}' \in \mathrm{Im}(\boldsymbol{A}^{\mathsf{T}}) \subset \mathbb{R}^d} \sum_i \left(\boldsymbol{w}'^{\mathsf{T}} \boldsymbol{x}_i - y_i\right)^2$$

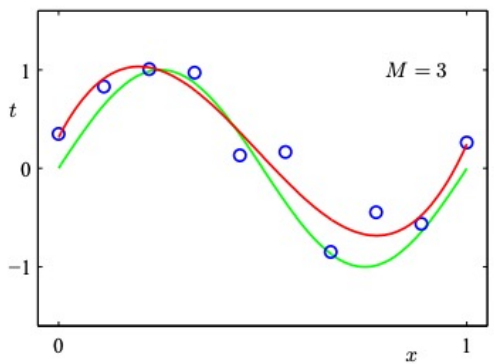
Overfitting and Regularization

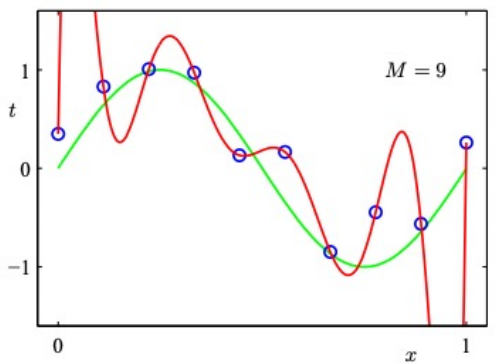
Should we use a very complicated mapping?

Ex: fitting a noisy sine function with a polynomial:









See Colab notebook

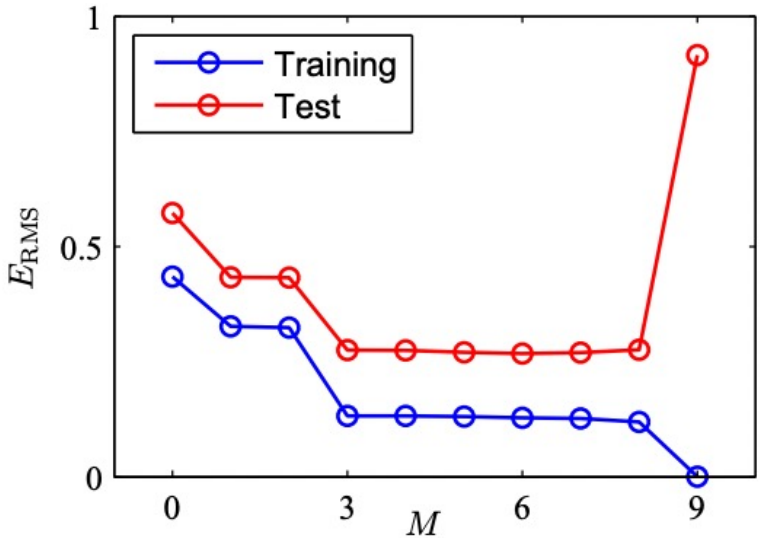
Underfitting and overfitting

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

- large training error
- large test error

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

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

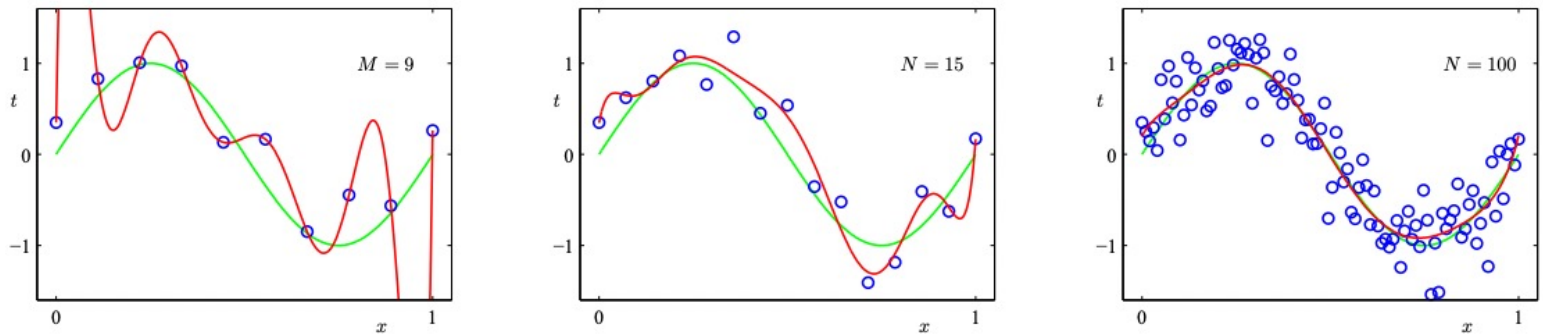


More complicated models ⇒ larger gap between training and test error

How to prevent overfitting?

See Colab notebook

Method 1: **More data!!**



More data ⇒ smaller gap between training and test error

See Colab notebook

Method 2: **Control model complexity**

For polynomial basis, the **degree** M clearly controls the complexity

- use **cross-validation** to pick hyper-parameter M

Cross-validation: Explored in HW1. Idea is to do a three-way split in addition to training set/test set, and tune hyperparameters on a *validation set*.

When M or in general Φ is fixed, are there still other ways to control complexity?

Magnitude of the weights

Least square solution for the polynomial example:

	$M=0$	$M=1$	$M=3$	$M=9$
w_0	0.19	0.82	0.31	0.35
w_1		-1.27	7.99	232.37
w_2			-25.43	-5321.83
w_3			17.37	48568.31
w_4				-231639.30
w_5				640042.26
w_6				-1061800.52
w_7				1042400.18
w_8				-557682.99
w_9				125201.43

Intuitively, **large weights $\Rightarrow$ more complex model**

See Colab notebook

How to make the weights small?

Regularized linear regression: new objective

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

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

- $\psi : \mathbb{R}^d \to \mathbb{R}^+$ is the *regularizer*
  - measure how complex the model $\boldsymbol{w}$ is, penalize complex models
  - common choices: $\|\boldsymbol{w}\|_2^2$, $\|\boldsymbol{w}\|_1$, etc.

How to make the weights small?

Regularized linear regression: new objective

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

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

- $\psi : \mathbb{R}^d \to \mathbb{R}^+$ 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
  - $\lambda \to +\infty$, $\boldsymbol{w} \to \text{argmin}_w \, \psi(\boldsymbol{w})$
  - i.e. control **trade-off** between training error and complexity

ℓ_2 regularization with non-linear basis: The effect of λ

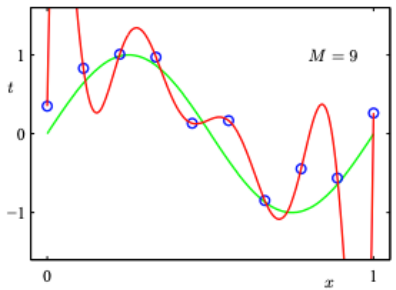
When we increase regularization coefficient λ:

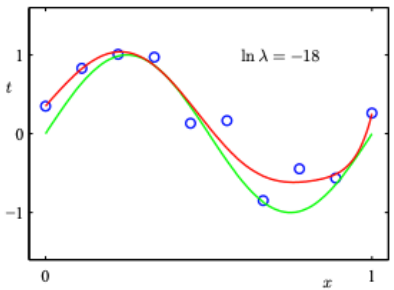
	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
w_0	0.35	0.35	0.13
w_1	232.37	4.74	-0.05
w_2	-5321.83	-0.77	-0.06
w_3	48568.31	-31.97	-0.06
w_4	-231639.30	-3.89	-0.03
w_5	640042.26	55.28	-0.02
w_6	-1061800.52	41.32	-0.01
w_7	1042400.18	-45.95	-0.00
w_8	-557682.99	-91.53	0.00
w_9	125201.43	72.68	0.01

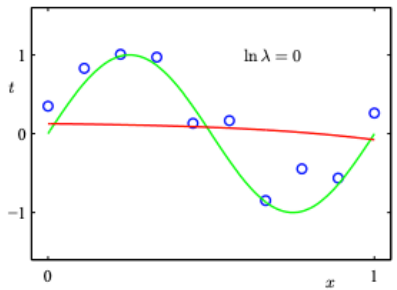
See Colab notebook

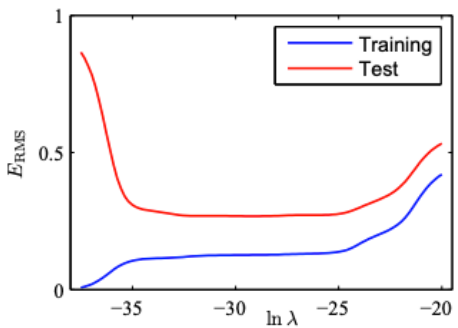
ℓ_2 regularization with non-linear basis : A **tradeoff**

when we increase regularization coefficient λ









See Colab notebook

Why is regularization useful?

If you don’t have sufficient data to fit your more expressive model, then ERM will overfit. **Regularization helps with generalization**.

So should it not be useful in many practical settings, where we have enough data?

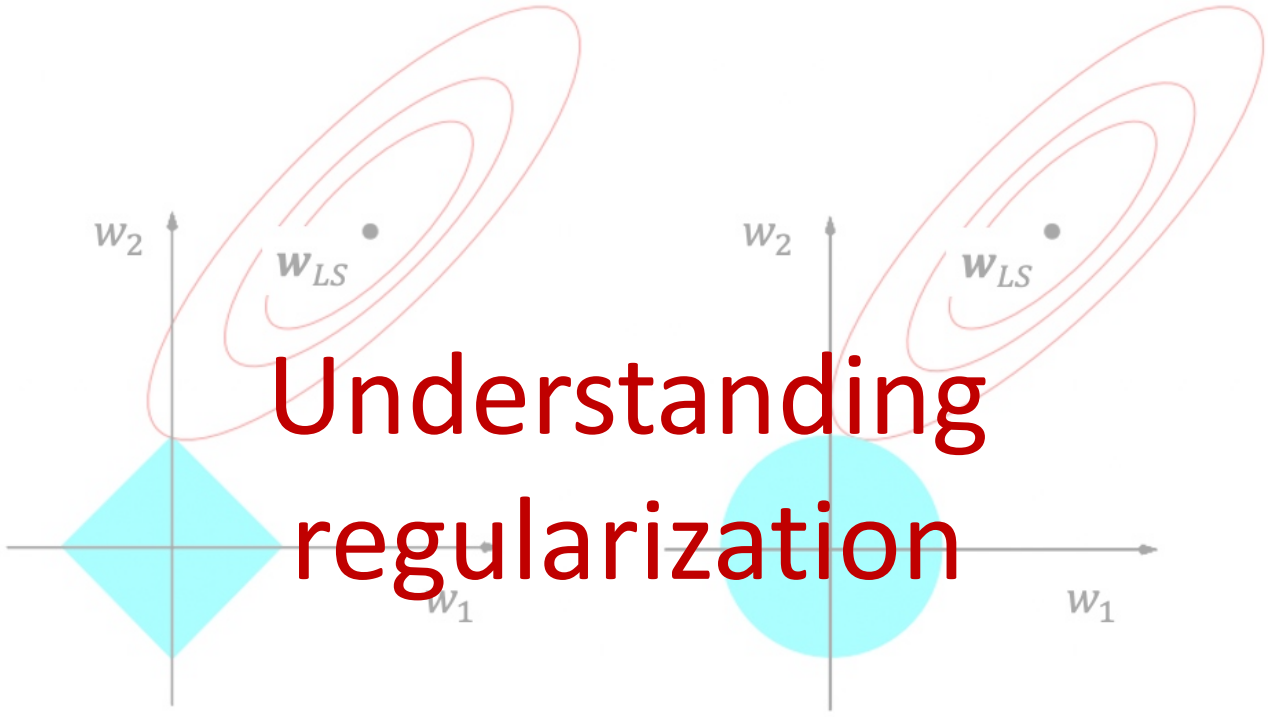
Why is regularization useful?

If you don't have sufficient data to fit your more expressive model, then ERM will overfit. **Regularization helps with generalization**.

So should it not be useful in many practical settings, where we have enough data?

In general, a viewpoint is that *we should always be trying to fit a more expressive model if possible*. We want our function class to be rich enough that we could almost overfit if we are not careful.

Since we're often in this regime where the models we want to fit are more and more complex, regularization is very useful to help generalization (it's also a relatively simple knob to control).



Understanding regularization

How to solve the regularized objective $G(\boldsymbol{w})$?

Let's go back to the original linear model.

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

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

How to solve the regularized objective $G(\boldsymbol{w})$?

Let's go back to the original linear model.

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

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

$$\nabla G(\boldsymbol{w}) = 2(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}) + 2\lambda\boldsymbol{w} = 0$$

$$\Rightarrow 2\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda\boldsymbol{I}\right)\boldsymbol{w} = 2\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$$

$$\Rightarrow \boldsymbol{w}^* = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$$

How to solve the regularized objective $G(\boldsymbol{w})$?

Let's go back to the original linear model.

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

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

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

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

For other regularizers, as long as it's **convex**, standard optimization algorithms can be applied.

Aside: Least-squares when $\boldsymbol{X^T X}$ is not invertible

When $\boldsymbol{X}^\top \boldsymbol{X}$ is not invertible $\boldsymbol{w}_{LS} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$ is not defined.

Aside: Least-squares when $\boldsymbol{X^T X}$ is not invertible

When $\boldsymbol{X}^\top \boldsymbol{X}$ is not invertible $\boldsymbol{w}_{LS} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$ is not defined.

This could happen when:

1. ∞ many $\boldsymbol{w}$ s.t. $\boldsymbol{X}\boldsymbol{w} = \boldsymbol{y}$
2. No such $\boldsymbol{w}$ s.t. $\boldsymbol{X}\boldsymbol{w} = \boldsymbol{y}$

The first condition can happen when $n < d$ (do not have enough data to learn)

Aside: Least-squares when $\boldsymbol{X^T X}$ is not invertible

When $\boldsymbol{X}^\top \boldsymbol{X}$ is not invertible $\boldsymbol{w}_{LS} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}$ is not defined.

This could happen when:

1. ∞ many $\boldsymbol{w}$ s.t. $\boldsymbol{X}\boldsymbol{w} = \boldsymbol{y}$
2. No such $\boldsymbol{w}$ s.t. $\boldsymbol{X}\boldsymbol{w} = \boldsymbol{y}$

The first condition can happen when $n < d$ (do not have enough data to learn)

What does L_2 regularization do here?

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

0 for all w s.t. $Xw=y$

$\therefore$ l_2 regularization chooses w with smallest $\|w\|_2$ s.t. $Xw=y$.

Aside: Least-squares when $\boldsymbol{X^T X}$ is not invertible

Intuition: what does inverting $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ do?

$$\textbf{eigendecomposition}: \quad \boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} = \boldsymbol{U}^{\mathrm{T}} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \lambda_{\mathsf{D}} & 0 \\ 0 & \cdots & 0 & \lambda_{\mathsf{D}+1} \end{bmatrix} \boldsymbol{U}$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \lambda_{\mathsf{D}+1} \geq 0$ are **eigenvalues**.

$$\textbf{inverse}: \quad (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1} = \boldsymbol{U}^{\mathrm{T}} \begin{bmatrix} \frac{1}{\lambda_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{\lambda_2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \frac{1}{\lambda_{\mathsf{D}}} & 0 \\ 0 & \cdots & 0 & \frac{1}{\lambda_{\mathsf{D}+1}} \end{bmatrix} \boldsymbol{U}$$

i.e. just invert the eigenvalues

Aside: Least-squares when $\boldsymbol{X^T X}$ is not invertible

Non-invertible $\Rightarrow$ some eigenvalues are 0.

One natural fix: add something positive

$$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda \boldsymbol{I} = \boldsymbol{U}^{\mathrm{T}} \begin{bmatrix} \lambda_1 + \lambda & 0 & \cdots & 0 \\ 0 & \lambda_2 + \lambda & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \lambda_{\mathsf{D}} + \lambda & 0 \\ 0 & \cdots & 0 & \lambda_{\mathsf{D}+1} + \lambda \end{bmatrix} \boldsymbol{U}$$

where $\lambda > 0$ and $\boldsymbol{I}$ is the identity matrix. Now it is invertible:

$$(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda \boldsymbol{I})^{-1} = \boldsymbol{U}^{\mathrm{T}} \begin{bmatrix} \frac{1}{\lambda_1 + \lambda} & 0 & \cdots & 0 \\ 0 & \frac{1}{\lambda_2 + \lambda} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \frac{1}{\lambda_{\mathsf{D}} + \lambda} & 0 \\ 0 & \cdots & 0 & \frac{1}{\lambda_{\mathsf{D}+1} + \lambda} \end{bmatrix} \boldsymbol{U}$$

A "Bayesian view" of ℓ_2 regularization

Maximum a posteriori probability (MAP) estimation: A Bayesian generalization of maximum likelihood estimation (MLE).

Let's continue with the linear model, and Q4 from the practice problems for today.

Have training set $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^d \times \mathbb{R}$

$y_i = w_*^T x_i + \epsilon_i$, $\epsilon_i \sim N(0, \sigma^2)$

$\therefore$ for any w, $Pr[y_i \mid x_i; w, \sigma] = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y_i - w^T x_i)^2}{2\sigma^2}\right)$

MLE: find w which maximizes likelihood

A "Bayesian view" of ℓ_2 regularization

Maximum a posteriori probability (MAP) estimation: A Bayesian generalization of maximum likelihood estimation (MLE).

Let's continue with the linear model, and Q4 from the practice problems for today.

$$\log\left[\Pr[y_i \mid x_i; w]\right) = -\frac{(y_i - w^T x_i)^2}{2\sigma^2} + \log\left(\frac{1}{\sigma\sqrt{2\pi}}\right)$$

The solution is $w^* = (X^T X)^{-1} X^T y$

A "Bayesian view" of ℓ_2 regularization

Maximum a posteriori probability (MAP) estimation: A Bayesian generalization of maximum likelihood estimation (MLE).

Bayesian view: A **prior** over $\boldsymbol{w}$

Suppose our prior for w is $N(0, \gamma^2 I)$

Now we find the model which maximizes aposteriori probability (MAP)

$$\text{Posterior} = \frac{\text{Prior} \cdot \text{Likelihood}}{\text{Normalization (same for all vectors } w)}$$

A "Bayesian view" of ℓ_2 regularization

Maximum a posteriori probability (MAP) estimation: A Bayesian generalization of maximum likelihood estimation (MLE).

Bayesian view: A **prior** over $\boldsymbol{w}$

$$\text{Posterior}(w) \propto \prod_{j=1}^{d} \exp\left(\frac{-w_j^2}{2\gamma^2}\right) \prod_{i=1}^{n} \exp\left(-\frac{(y_i - w^T x_i)^2}{2\sigma^2}\right)$$

$$\log(\text{Posterior}(w)) = -\frac{\|w\|_2^2}{2\gamma^2} - \sum_{i=1}^{n} (y_i - w^T x_i)^2$$

$\therefore$ $\max(\log(\text{Posterior}))$ is same as $\min G(w)$ for $\psi(w) = \|w\|_2^2$

An equivalent form, and a "Frequentist view"

"Frequentist" approach to justifying regularization is to argue that if the true model has a specific property, then regularization will allow you to recover a good approximation to the true model. We this view, we can equivalently formulate regularization as:

$$\underset{\boldsymbol{w}}{\operatorname{argmin}} \operatorname{RSS}(w) \quad \textbf{subject to } \psi(\boldsymbol{w}) \leq \beta$$

$\psi(w) = \|w\|_2^2$

where β is some hyper-parameter.

Finding the solution becomes a *constrained optimization problem*.

Choosing either λ or β can be done by cross-validation.