Lecture 1a: Polynomial Curve Fitting

A first encounter with model complexity, overfitting, and regularization

Jue Guo

University at Buffalo

May 7, 2026

Outline

Where we are

Lecture 1 – Introduction. Following Bishop, PRML Chapter 1 (§1.1–§1.6).

- **1a (today).** Polynomial curve fitting, overfitting, regularization. (§1.1)
- 1b. Probability theory, maximum likelihood, Bayesian curve fitting. (§1.2)
- 1c. Model selection and the curse of dimensionality. (§1.3–§1.4)
- 1d. Decision theory: loss, Bayes optimal classifier, reject option. (§1.5)
- 1e. Information theory: entropy, KL divergence, mutual information. (§1.6)

Why start here?

Curve fitting is the simplest setting where every concept that drives modern machine learning – capacity, generalization, regularization, prior knowledge – already shows up in a form you can plot on one slide.

Supervised learning, in one slide

Setup. We observe N input-target pairs $\mathcal{D} = \{(x_n, t_n)\}_{n=1}^{N}$ and want to predict t for a new x.

- Regression. $t \in \mathbb{R}$ is continuous (today's setting).
- Classification. t takes one of finitely many labels.

Two sources of difficulty.

- Only finitely many examples $\Rightarrow$ we must *generalize*.
- The targets are noisy $\Rightarrow$ for a given x there is uncertainty in the correct t.

Probability theory (next deck) gives us a language for the second. Today we work *informally*, by curve fitting.

The running example

Generate data from $\sin(2\pi x)$ on $[0, 1]$, corrupt each target with independent Gaussian noise, observe $N = 10$ points.

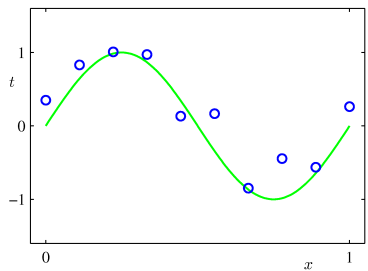

Figure 1.2

Green: the true $\sin(2\pi x)$ (unknown to the learner). Blue: the training set we get to see.

The model: a polynomial in x

Fit the data with a polynomial of order M,

$$y(x, \mathbf{w}) \;=\; w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M \;=\; \sum_{j=0}^{M} w_j \, x^j. \tag{1}$$

Linear in the unknowns. y is nonlinear in x, but *linear in* $\mathbf{w}$.

- This makes it a linear model – closed-form solution, convex error.
- The same trick (fix a basis, learn weights) underlies kernel methods, radial basis functions, and the last layer of a neural network.

Sum-of-squares error

Choose $\mathbf{w}$ to minimize the discrepancy between predictions and targets:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2. \tag{2}$$

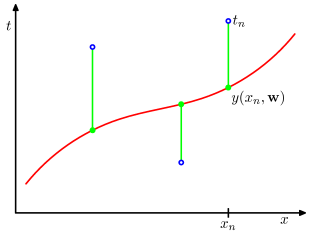

Figure 1.3

Each green bar is a vertical residual; $E(\mathbf{w})$ sums their squares. The $\frac{1}{2}$ is a convenience for differentiation later.

Closed-form minimizer

$E(\mathbf{w})$ is quadratic in $\mathbf{w}$, so $\nabla_{\mathbf{w}} E$ is linear in $\mathbf{w}$. Setting it to zero gives a unique minimizer $\mathbf{w}^\star$ – the normal equations. Stack the polynomial features into a design matrix $\mathbf{\Phi} \in \mathbb{R}^{N \times (M+1)}$ with $\Phi_{nj} = x_n^j$; then

$$\mathbf{w}^\star \;=\; (\mathbf{\Phi}^\top \mathbf{\Phi})^{-1}\, \mathbf{\Phi}^\top \mathbf{t}. \tag{3}$$

The remaining question is how to choose M – the order of the polynomial, which controls how flexible the model is. PRML calls this *model comparison* or *model selection*.

Derivation – where the normal equations come from

Step 1. Stack predictions and targets in vector form: $\mathbf{y} = \mathbf{\Phi w}$ where $(\mathbf{\Phi})_{nj} = x_n^j$. Then

$$E(\mathbf{w}) \;=\; \tfrac{1}{2}\sum_{n=1}^{N}\big\{y(x_n,\mathbf{w}) - t_n\big\}^2 \;=\; \tfrac{1}{2}(\mathbf{\Phi w} - \mathbf{t})^\top(\mathbf{\Phi w} - \mathbf{t}).$$

Step 2. Expand the quadratic:

$$E(\mathbf{w}) \;=\; \tfrac{1}{2}\,\mathbf{w}^\top\mathbf{\Phi}^\top\mathbf{\Phi}\,\mathbf{w} \;-\; \mathbf{w}^\top\mathbf{\Phi}^\top\mathbf{t} \;+\; \tfrac{1}{2}\,\mathbf{t}^\top\mathbf{t}.$$

Step 3. Take the gradient (using $\nabla_{\mathbf{w}}\,\mathbf{w}^\top\mathbf{A}\mathbf{w} = (\mathbf{A} + \mathbf{A}^\top)\mathbf{w}$ and that $\mathbf{\Phi}^\top\mathbf{\Phi}$ is symmetric):

$$\nabla_{\mathbf{w}} E(\mathbf{w}) \;=\; \mathbf{\Phi}^\top\mathbf{\Phi}\,\mathbf{w} \;-\; \mathbf{\Phi}^\top\mathbf{t}.$$

Step 4. Set to zero and solve. The condition $\mathbf{\Phi}^\top\mathbf{\Phi}\,\mathbf{w}^\star = \mathbf{\Phi}^\top\mathbf{t}$ is the normal equations; if $\mathbf{\Phi}$ has full column rank (distinct x_n, $N \geq M+1$),

$$\mathbf{w}^\star \;=\; (\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\,\mathbf{\Phi}^\top\mathbf{t}.$$

Geometry. $\mathbf{\Phi w}^\star$ is the orthogonal projection of $\mathbf{t}$ onto the column span of $\mathbf{\Phi}$; the residual $\mathbf{t} - \mathbf{\Phi w}^\star$ is perpendicular to every feature column.

Increasing M: from underfit to overfit

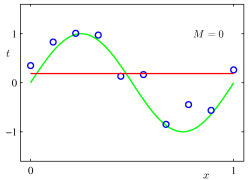

Figure 1.4a

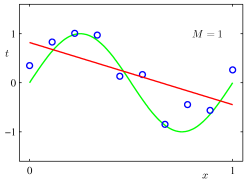

Figure 1.4b

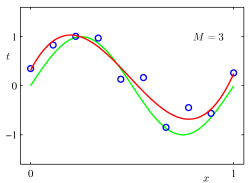

Figure 1.4c

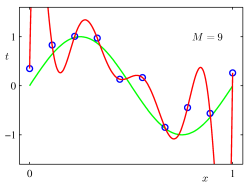

Figure 1.4d

$M = 0, 1$: underfit. $M = 3$: captures the sine. $M = 9$: passes through every training point, oscillates wildly between them – overfit.

Quantifying generalization: RMS error

Compare across dataset sizes by reporting error in the units of t:

$$E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^\star)/N}. \tag{4}$$

Evaluate on

- the training set (the 10 points the model fitted),
- an independent test set of 100 points drawn the same way.

Training error tells us how well we fit; test error tells us how well we *generalize*. Their gap is the diagnostic for overfitting.

Train vs. test error as M grows

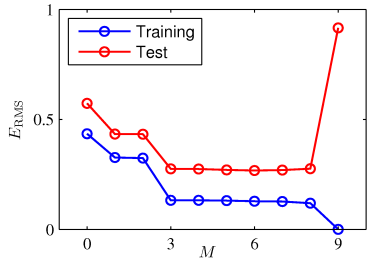

Figure 1.5

Training error decreases monotonically; test error has a *sweet spot* around $3 \leq M \leq 8$ and explodes at $M = 9$. The U-shape on test error is the central object of model selection.

What blows up? The coefficients.

	$M = 0$	$M = 1$	$M = 6$	$M = 9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

The $M = 9$ fit pays for its zero training error with coefficients of order 10^5–10^6, tuned to cancel each other. This is overfitting in numbers.

More data tames overfitting

Same $M = 9$ polynomial, larger training sets:

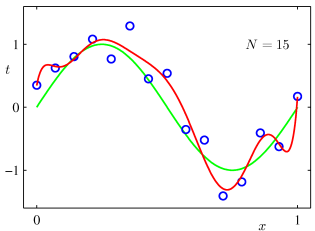

Figure 1.6a

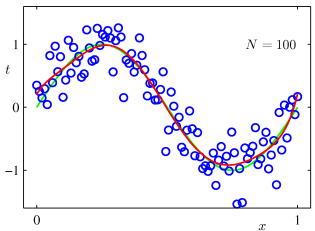

Figure 1.6b

Same model class. With more data the same flexible polynomial recovers $\sin(2\pi x)$ cleanly. PRML's heuristic: *N should be several times the number of parameters*. Useful as a rule of thumb, but not a real theory.

Two ways to fight overfitting

- More data. Always works, but data is expensive and sometimes fixed by the problem.
- Regularization. When data is fixed, *constrain the model* so it cannot use its full flexibility to chase noise.

Both make the same observation in different language: the trouble at $M = 9$ is not the polynomial degree per se, but the unbounded magnitudes the coefficients are allowed to take.

Idea

Penalize large coefficients directly inside the error function.

Regularized least squares

Add a quadratic penalty on the weights:

$$\widetilde{E}(\mathbf{w}) \;=\; \tfrac{1}{2}\sum_{n=1}^{N}\big\{y(x_n,\mathbf{w})-t_n\big\}^2 \;+\; \tfrac{\lambda}{2}\,\|\mathbf{w}\|^2, \qquad \|\mathbf{w}\|^2=\mathbf{w}^\top\mathbf{w}. \tag{5}$$

- $\lambda \to 0$: recover ordinary least squares.
- $\lambda \to \infty$: coefficients are forced to zero – maximally smooth, useless.
- λ somewhere in between: the sweet spot.

Names you will hear.

- Statistics: *ridge regression* (Hoerl & Kennard, 1970).
- Neural networks: *weight decay*.
- Bayesian view (next deck): a Gaussian prior on $\mathbf{w}$.

Effect of the regularizer at $M = 9$

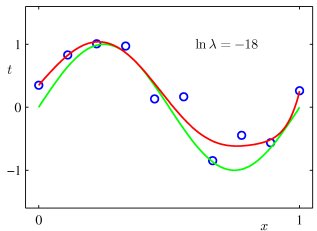

Figure 1.7a

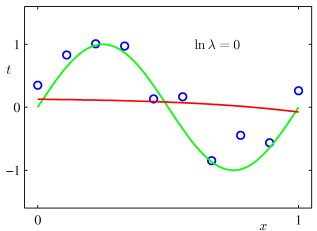

Figure 1.7b

Recall $\ln \lambda = -\infty$ is the wild $M = 9$ curve from earlier. A moderate $\ln \lambda = -18$ recovers the underlying sine; too large ($\ln \lambda = 0$) over-smooths and underfits.

Coefficients shrink as λ grows

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

λ controls the *effective complexity* of the model, even though the polynomial degree is unchanged. Same hypothesis class, smaller usable region in weight space.

From one knob to another – but still a knob

Fixing $M = 9$ and sweeping $\ln \lambda$ gives a familiar U-shape on the test error: too little regularization overfits, too much underfits.

- We replaced the question "what is the right M?" with the question "what is the right λ?".
- Both are hyperparameters: not learned by minimizing training error.
- How do we pick them honestly, without peeking at the test set?

Next time

Hold-out validation, S-fold cross-validation, and a probabilistic re-derivation of the same regularizer as a Gaussian prior.

Takeaways

- Curve fitting with polynomials is a clean window onto the entire bias-variance / capacity story.
- Overfitting is visible three ways at once: in the *plot*, in the *train/test gap*, and in the *coefficient magnitudes*.
- Two reliable cures: more data, or a regularizer that keeps weights small.
- The regularization parameter λ controls effective complexity – and is itself a thing we have to choose.

Reading

Bishop, *Pattern Recognition and Machine Learning* (2006), §1.1.