

From models to predictors

Hicham Janati

Fall 2026

Three of the first four chapters were exercises in modeling. Chapter 1 proposed a distribution for the data themselves, a Gaussian for $\mathbf{X}$; chapter 2 refined it into a mixture when one Gaussian could not hold the shape; chapter 4 turned to the response and modeled the whole conditional distribution of Y given $\mathbf{X}$, complete with assumptions, sampling laws and tests. Now put a practical question next to all that machinery. The store’s manager wants to know which regular customers will stop coming within six months or, for a new customer, how much they will spend this year. An answer is not a distribution; it is a guess per customer, and it will be judged the only way the manager cares about: was it right, often enough? Nothing in the question mentions the shape of a density or the value of a parameter. Is modeling the distribution of the data really necessary, then, when what we want is the answer? The chapter moves from modeling the distributions behind the data to *learning predictive functions* from the data: a function f that takes a customer’s covariates and returns a prediction, built and judged on its predictions alone.

Throughout, the course conventions hold: bold letters denote vectors and matrices, plain italic letters denote scalars.

1 Supervised learning and generalization

1.1 Data, predictors, and two kinds of targets

The raw material is an i.i.d. sample of a random pair: rows $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$ drawn from an unknown joint distribution P of $(\mathbf{X}, Y)$, where $\mathbf{X} \in \mathbb{R}^d$ collects the *inputs* (the covariates) and Y is the *target*. A *predictor* is a function f that maps an input to a guess of the target. The nature of Y splits the field in two.

- *Regression*: Y is continuous. Predict a customer’s monthly spend from their profile; the sale price of an apartment from its surface, floor and district; the monthly demand for a product from its price.
- *Classification*: Y is discrete, a label from a finite set. Predict whether a card transaction is fraudulent ($Y \in \{0, 1\}$) from its amount, hour and history; whether a customer will churn (stop visiting within six months) from their behavior; which digit ($Y \in \{0, \dots, 9\}$) a handwritten image shows. This chapter treats the binary case, coded $Y \in \{0, 1\}$.

Remark

In classification, the learned predictor need not directly output a class. It may output a *score* or an estimated *probability* (“this transaction is fraudulent with probability 0.83”), which is turned into a decision later, by choosing a threshold. The two steps are usefully kept separate: the same fitted score can serve a cautious bank (block above 0.5) and a bolder one (block above 0.95), and the threshold is a question about the costs of the two mistakes, not about fitting.

1.2 Losses, risk, and empirical risk

We want a function that makes as few errors as possible, which requires saying what one error costs: a *loss function* $L(y, \hat{y})$ measures the discrepancy between the target y and the prediction $\hat{y}$. Each kind of problem has its standard choices.

- *Squared loss* (regression): $L(y, \hat{y}) = (y - \hat{y})^2$, the running choice of this chapter, and the natural companion of everything Part 1 did.
- *Absolute loss* (regression): $L(y, \hat{y}) = |y - \hat{y}|$, less sensitive to occasional huge errors than the squared loss, which squares them.
- *0–1 loss* (classification, class output): $L(y, \hat{y}) = \mathbf{1}\{y \neq \hat{y}\}$: every mistake costs one, and its average is the misclassification rate.
- *Log loss* (classification, probability output $p \in (0, 1)$): $L(y, p) = -[y \log p + (1 - y) \log(1 - p)]$, which rewards probabilities that match the outcomes and punishes confident mistakes severely: predicting $p = 0.99$ on a transaction that turns out legitimate costs $-\log 0.01 \approx 4.6$, against 0.7 for a noncommittal $p = 0.5$.

A predictor is then judged by its expected loss on a *new* draw, its *risk*

$$R(f) = \mathbb{E}[L(Y, f(\mathbf{X}))], \quad (\mathbf{X}, Y) \sim P \text{ independent of the sample.} \quad (1)$$

The i.i.d. assumption does real work here: it is what makes future rows resemble past ones, so that a risk estimated from the file says something about the cases to come. The prologue’s caveat about ordered data applies with full force.

The risk cannot be computed, since P is unknown. The operational principle of learning replaces the expectation by the sample average and minimizes over a chosen *hypothesis class* $\mathcal{F}$, the set of functions the learner is allowed to pick from:

$$\hat{f} = \arg \min_{f \in \mathcal{F}} \hat{R}_n(f), \quad \hat{R}_n(f) = \frac{1}{n} \sum_{i=1}^n L(y_i, f(\mathbf{x}_i)), \quad (2)$$

the criterion being the *empirical risk* and the principle *empirical risk minimization*. Nothing in (2) mentions a true parameter, an error distribution, or a probabilistic assumption on $\mathbf{X}$ and Y : there is no model, only a family of functions and a score.

1.3 Generalization

Solving the minimization problem (2) is called *training* the model: the observations used are the *training data*, and the solution $\hat{f}$ is the learned prediction function. Training produces two things (the function, and the number $\hat{R}_n(\hat{f})$, its average loss on the training rows), and the number must be handled with care. It is not a measure of performance. The function $\hat{f}$ was chosen precisely to make this quantity as small as possible, so the training loss is inherently better than what $\hat{f}$ achieves anywhere else: part of what the minimization absorbed is the noise of the particular rows at hand, and noise does not repeat. Evaluating a predictor therefore requires data the fit has never seen, which we call *test data*.

Formally, set aside a *test set* of n_{te} rows that take no part in training, and report the average loss of the learned function on them:

$$\hat{R}_{\text{te}}(\hat{f}) = \frac{1}{n_{\text{te}}} \sum_{i \in \text{test}} L(y_i, \hat{f}(\mathbf{x}_i)), \quad (3)$$

the *test error*. It is an unbiased estimate of the risk $R(\hat{f})$: conditionally on the training data, the test terms are i.i.d. draws of the very quantity the risk averages. The discipline that makes

(3) trustworthy is procedural: the test set is consulted *once*, after every choice has been made; any row that influenced a choice is part of training, whatever it is called. For the Gaussian linear model the optimism of the training error can even be computed exactly: the gap between fresh-data error and training error averages $2\sigma^2(d+1)/n$, growing with the number of fitted parameters.

The other question (2) leaves open is the choice of $\mathcal{F}$. Minimizing over *all* functions is not an option: the space of all functions is infinite-dimensional, and a function free to take any value at any input can simply memorize the training file. We must restrict the search to a smaller family. How large, then, should the family be? A concrete problem shows what is at stake at both ends. Suppose we are to detect fraudulent card transactions from features such as the amount, the time, the location, the account identifier, the card holder’s name and the time elapsed since the last transaction.

- If $\mathcal{F}$ is too small, only very simple rules are available, for instance “flag a transaction if the amount exceeds 300 Dh and the time is between midnight and 3 a.m.” Some frauds do look like that, but many honest purchases do too, and most frauds do not. The rule scores poorly on the training data already, and no amount of fitting can help, because nothing better exists inside the family.
- If $\mathcal{F}$ is too large, arbitrarily intricate rules become available, and among them are rules that describe the training file rather than fraud, for instance “flag a transaction if it is between 2:10 and 2:25 a.m., for an amount above 430 Dh, when the phone number ends with 81 and the card name is Ilyas.” If the training file happens to contain a few frauds matching exactly this combination, the rule is rewarded during training and may look perfect there. What it has learned is an accident of the sample; the next fraud will match none of it.

The first failure is *underfitting*: the family is too rigid to capture the structure of the problem, and the fit is poor on the training data itself. The second is *overfitting*: the family is flexible enough to fit the noise of the sample, the training loss becomes excellent, and the model still predicts poorly. Behind the two failures sits a trade-off the course will meet again and again, the *bias–variance* trade-off. A small family is wrong on average, however much data it is given: that systematic error is *bias*. A large family molds itself to whatever sample it is shown, noise included, so retraining on another sample from the same population would give a very different function: that instability is *variance*.

What counts as “too large” is not absolute: it is relative to the quantity and quality of the data. The Ilyas rule survived training because the file contained only a handful of frauds under those circumstances; with more observed frauds in similar conditions, the file would inevitably contain other card names than Ilyas and other final phone numbers than 81, and the rule would stop paying off on the training data itself. A poor fit on fresh data can therefore come from scarce or poor data as much as from a badly sized family: complexity must be matched to the data that has to support it.

We want a learned function $\hat{f}$ that *generalizes*, that is, predicts well on unseen data, and the test error (3) measures how well it does. The next section rediscovers the linear models of the statistical chapters from this new perspective.

2 Linear models revisited

The simplest hypothesis classes are linear in the inputs, and each kind of problem has a linear workhorse. We present both without any of chapter 4’s statistical apparatus: no assumptions, no standard errors, no tests; only a family, a loss, and (2).

2.1 Linear regression

The data are i.i.d. observations $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$ with a continuous target, the family is the affine functions $f(\mathbf{x}) = \beta_0 + \beta^\top \mathbf{x}$, and the loss is squared. One change of variables simplifies everything, as it did in chapter 4: absorb the intercept into the coefficient vector by appending a constant coordinate to every input,

$$\mathbf{x}_i \mapsto (1, x_{i1}, \dots, x_{id})^\top \in \mathbb{R}^{d+1}, \quad \beta = (\beta_0, \beta_1, \dots, \beta_d)^\top \in \mathbb{R}^{d+1},$$

so that $\beta_0 + \beta^\top \mathbf{x}_i$ becomes the single product $\beta^\top \mathbf{x}_i$ with the new convention. The observations then vectorize: stack the inputs as the rows of a matrix $\mathbb{X} \in \mathbb{R}^{n \times (d+1)}$ and the targets in $\mathbf{y} = (y_1, \dots, y_n)^\top$. The empirical risk (2) of the squared loss is

$$\hat{R}_n(\beta) = \frac{1}{n} \sum_{i=1}^n (y_i - \beta^\top \mathbf{x}_i)^2 = \frac{1}{n} \|\mathbf{y} - \mathbb{X}\beta\|^2, \quad (4)$$

the second form because the vector $\mathbf{y} - \mathbb{X}\beta$ has i -th coordinate $y_i - \beta^\top \mathbf{x}_i$, and the squared norm of a vector is the sum of its squared coordinates. To minimize, expand and take the gradient in β :

$$\|\mathbf{y} - \mathbb{X}\beta\|^2 = \mathbf{y}^\top \mathbf{y} - 2\beta^\top \mathbb{X}^\top \mathbf{y} + \beta^\top \mathbb{X}^\top \mathbb{X} \beta, \quad \nabla_\beta \hat{R}_n(\beta) = \frac{2}{n} (\mathbb{X}^\top \mathbb{X} \beta - \mathbb{X}^\top \mathbf{y}).$$

The objective is convex (its Hessian $\frac{2}{n} \mathbb{X}^\top \mathbb{X}$ is positive semidefinite), so setting the gradient to zero characterizes the global minimum: $\mathbb{X}^\top \mathbb{X} \beta = \mathbb{X}^\top \mathbf{y}$, the normal equations of chapter 4, and when $\mathbb{X}^\top \mathbb{X}$ is invertible,

$$\hat{\beta} = (\mathbb{X}^\top \mathbb{X})^{-1} \mathbb{X}^\top \mathbf{y}. \quad (5)$$

This is exactly OLS. The computation is familiar; the objective has changed. Nothing was assumed about how the data arose, and in exchange nothing entitles us to the laws, intervals and tests of chapter 4.

2.2 Logistic regression

For a binary target, an affine function is a poor direct output: it ranges over all of $\mathbb{R}$, while a probability must live in $(0, 1)$. Logistic regression keeps the affine core (written $\beta^\top \mathbf{x}$ with the intercept absorbed into $\beta \in \mathbb{R}^{d+1}$, the convention of section 2.1) and squashes it through the *sigmoid* $\sigma(t) = 1/(1 + e^{-t})$, an increasing S-shaped map from $\mathbb{R}$ onto $(0, 1)$, predicting

$$p_\beta(\mathbf{x}) = \sigma(\beta^\top \mathbf{x}) = \frac{1}{1 + e^{-\beta^\top \mathbf{x}}}, \quad (6)$$

an estimated probability that $Y = 1$. The coefficients are fitted by empirical risk minimization with the log loss:

$$\hat{\beta} = \arg \min_{\beta \in \mathbb{R}^{d+1}} \frac{1}{n} \sum_{i=1}^n - \left[y_i \log p_\beta(\mathbf{x}_i) + (1 - y_i) \log(1 - p_\beta(\mathbf{x}_i)) \right]. \quad (7)$$

Unlike least squares, problem (7) has no closed-form solution; the objective is smooth and convex, and a numerical optimizer finds the minimum. Statistical software does this routinely and reliably. The output is a score in the sense of the remark of section 1.1: thresholding at $\frac{1}{2}$ turns it into a class, and since $\sigma(t) = \frac{1}{2}$ exactly at $t = 0$, the *decision boundary*, the set of inputs where the two classes are predicted equally probable, is the hyperplane $\beta^\top \mathbf{x} = 0$. The frontier between the predicted classes is straight: logistic regression is the linear model of classification.

2.3 More than two classes: the softmax

When the target takes K classes (the handwritten digit of section 1.1, with $K = 10$), two devices extend the binary construction. The first acts on the labels: *one-hot encoding* replaces the class k by the vector $\mathbf{y} \in \{0, 1\}^K$ whose k -th coordinate is 1 and all others 0, so a label becomes a probability vector concentrated on one class. Concretely, with the three customer segments of chapter 2 (occasional, regular, bulk, in that order, so $K = 3$),

$$\text{occasional} \mapsto (1, 0, 0), \quad \text{regular} \mapsto (0, 1, 0), \quad \text{bulk} \mapsto (0, 0, 1),$$

and for the digits ($K = 10$, counted from zero) the label 3 becomes the ten-dimensional vector $(0, 0, 0, 1, 0, \dots, 0)$, with the 1 in the fourth position. The second acts on the scores: give each class its own coefficient vector $\beta_1, \dots, \beta_K \in \mathbb{R}^{d+1}$ (intercepts absorbed as before) and turn the K affine scores into probabilities with the *softmax*,

$$p_k(\mathbf{x}) = \frac{\exp(\beta_k^\top \mathbf{x})}{\sum_{j=1}^K \exp(\beta_j^\top \mathbf{x})}, \quad k = 1, \dots, K, \quad (8)$$

which are positive and sum to one: a probability distribution over the classes at every input. The log loss generalizes to the *cross-entropy*: for a one-hot label $\mathbf{y}$ encoding the class k ,

$$L(\mathbf{y}, p(\mathbf{x})) = - \sum_{j=1}^K y_j \log p_j(\mathbf{x}) = - \log p_k(\mathbf{x}), \quad (9)$$

since only the term of the true class survives: the loss is minus the log of the probability the model assigns to the correct class, exactly as in the binary case. Training minimizes the average cross-entropy over $\beta_1, \dots, \beta_K$; the problem is again smooth, convex, and solved numerically. For $K = 2$ the construction collapses to logistic regression: dividing the numerator and denominator of (8) by $\exp(\beta_2^\top \mathbf{x})$ gives $p_1(\mathbf{x}) = \sigma((\beta_1 - \beta_2)^\top \mathbf{x})$, the sigmoid with coefficient vector $\beta_1 - \beta_2$, and the cross-entropy is then the log loss.

2.4 Linear models can overfit

A linear model looks immune to the failures of section 1.3: with $d + 1$ coefficients and nothing else, it seems too rigid to produce an intricate rule like the Ilyas one. The flexibility comes from the features. Nothing forces us to feed the model the raw inputs $x_1, \dots, x_d$: we can also add their powers $x_1^2, x_2^2, \dots$, and even their products $x_1 x_2, x_1 x_3, \dots$ (so-called *polynomial features*) and regress on the enlarged list. With two inputs, for instance,

$$f(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_2^2 + \beta_5 x_1 x_2 :$$

every engineered feature carries its own coefficient, the predictor is $\beta^\top \phi(\mathbf{x})$ for the feature vector $\phi(\mathbf{x}) = (1, x_1, x_2, x_1^2, x_2^2, x_1 x_2)^\top$, and the OLS and logistic machinery of the previous subsections runs on it unchanged. Pushing further (the *degree* q keeps every monomial $x_1^a x_2^b$ with $a + b \leq q$), the family grows with q , and with it the capacity to overfit.

Remark

With polynomial features the fitted function is nonlinear in $\mathbf{x}$ (a conic, a degree-10 curve), but the model remains *linear*: the predictor $f(\mathbf{x}) = \beta^\top \phi(\mathbf{x})$ depends linearly on β , and that is what “linear model” refers to (the coefficients, not the shape of f in $\mathbf{x}$). Everything in this section (empirical risk, gradients, closed forms) applies verbatim with $\phi(\mathbf{x})$ in place of $\mathbf{x}$.

Figure 1 stages the danger on 120 training points with two input features and a binary label whose true structure is curved: the red class concentrates inside an elliptical region, the blue class surrounds it. Plain logistic regression can only draw a straight boundary, and no

straight line separates an inside from an outside: the fit underfits, at 36% test error, barely better than guessing. Degree-2 features ($x_1, x_2, x_1^2, x_1x_2, x_2^2$) make the boundary, linear in the enriched features, a conic in the original ones. Degree 2 captures the ellipse and cuts the test error to 11%. Raising the degree further does not help: at degree 10 the boundary grows lobes and islands that chase individual training points, the training error collapses to 2%, and the test error *rises* to 14%. The extra flexibility is spent fitting noise, the boundary’s islands playing the role of section 1.3’s Ilyas rule. The model is overfitting, and the right panel condenses the story: training error falls with the degree, mechanically, while the test error traces a U.

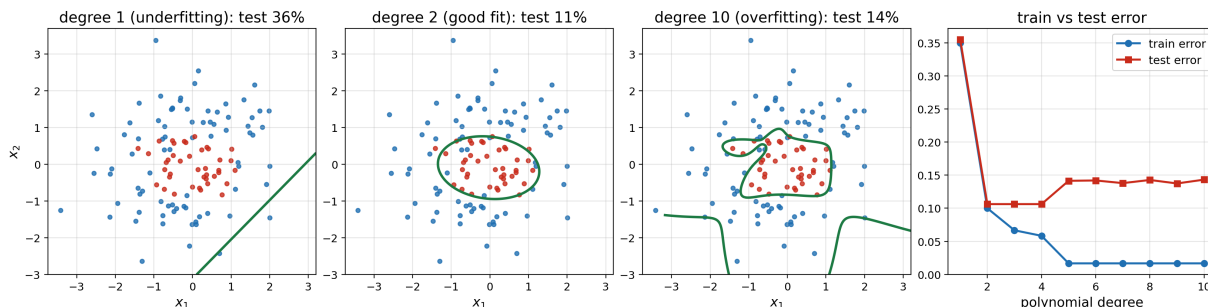


Figure 1: Logistic regression with polynomial features of growing degree, on data whose true class boundary is an ellipse. Degree 1 underfits (36% test error); degree 2 matches the structure (11%); degree 10 chases individual training points (14% test error against 2% training error). Right: training error falls monotonically with the degree while test error is U-shaped.

The degree measures the flexibility of the family, and it is a *hyperparameter*: a setting of the procedure, not a coefficient the fit determines (the successor of chapter 2’s number of components K). The U-shape is the bias–variance trade-off made quantitative: raising the degree lowers the bias and inflates the variance, the error on fresh data includes both, and the minimum lies where the two balance. What the picture rules out is using the training error to choose the degree: it would always answer “more”. How to control complexity, and how to choose these hyperparameters honestly, is the next section’s subject.

3 Regularization and model selection

Overfitting is a routine problem in practice: model families are flexible, data are finite, and a fit that does well on the training file while failing on fresh rows is common. Managing it means tuning model complexity, and replacing the whole family $\mathcal{F}$ by a different one (fewer covariates, a lower degree) is a blunt instrument. This section develops a finer control: keep the family, make its functions simpler by constraining their coefficients, and choose the strength of the constraint from the data.

3.1 From smaller families to penalties

Overfitting came from too-large families, so the direct fix is to make the family smaller. Within a parametrized family f_β , the size of the coefficient vector is a natural handle: instead of minimizing the empirical risk over every β , constrain its norm,

$$\min_{\beta} \hat{R}_n(f_\beta) \quad \text{subject to} \quad \|\beta\|^2 \leq \tau.$$

The constraint sets nest: a small τ leaves a tightly restricted family, a large τ restores the full one, and the family size has become a continuous parameter rather than a discrete choice of columns or degrees. Constrained problems are awkward to solve directly, but this one is equivalent (we

use the fact without proof) to the *penalized* problem

$$\min_{\beta} \widehat{R}_n(f_{\beta}) + \lambda \|\beta\|^2, \quad (10)$$

in the sense that for every τ there is a $\lambda \geq 0$ giving the same solution, and conversely. This is *Tikhonov regularization*, and $\lambda \geq 0$, the *regularization strength*, plays the role of $1/\tau$: at $\lambda = 0$ nothing is constrained, as λ grows the effective family shrinks, and at $\lambda = \infty$ the fit is frozen at $\beta = \mathbf{0}$, ignoring the data entirely. Like the polynomial degree, λ is a hyperparameter.

3.2 Ridge regression

Specializing (10) to linear regression gives the *ridge* estimator [1],

$$\hat{\beta}_{\lambda} = \arg \min_{\beta} \frac{1}{n} \|\mathbf{y} - \mathbb{X}\beta\|^2 + \lambda \|\beta\|^2. \quad (11)$$

The gradient computation of section 2.1 carries over with one extra term:

$$\nabla_{\beta} = \frac{2}{n} (\mathbb{X}^{\top} \mathbb{X} \beta - \mathbb{X}^{\top} \mathbf{y}) + 2\lambda \beta = \mathbf{0} \iff \left(\frac{1}{n} \mathbb{X}^{\top} \mathbb{X} + \lambda \mathbf{I} \right) \beta = \frac{1}{n} \mathbb{X}^{\top} \mathbf{y},$$

so that

$$\hat{\beta}_{\lambda} = \left(\widehat{\Sigma} + \lambda \mathbf{I} \right)^{-1} \frac{1}{n} \mathbb{X}^{\top} \mathbf{y}, \quad \widehat{\Sigma} = \frac{1}{n} \mathbb{X}^{\top} \mathbb{X}. \quad (12)$$

The closed form exposes the two benefits of the penalty.

- *The solution is unique, whatever the columns.* Every eigenvalue of $\widehat{\Sigma} + \lambda \mathbf{I}$ is at least $\lambda > 0$, so the matrix is invertible even when $\mathbb{X}^{\top} \mathbb{X}$ is singular, as with more columns than rows or perfectly collinear columns. No such guarantee protected OLS.
- *It regularizes the empirical covariance.* With standardized covariates, $\widehat{\Sigma}$ is the empirical covariance matrix of the columns, and the penalty adds λ to its diagonal: exactly chapter 1's regularization $\widehat{\Sigma} + \varepsilon \mathbf{I}$ of a near-singular covariance. A collinear pair makes $\widehat{\Sigma}$ nearly singular and the OLS coefficients unstable along the ill-determined direction; the diagonal term floors those eigenvalues, which addresses the multicollinearity problem left open in chapter 4.

As λ grows the coefficients shrink smoothly toward zero; in the simplest setting every coefficient is divided by the same factor $1 + \lambda$. Two practical conventions: the intercept is not penalized, and the covariates are standardized first, since the penalty compares coefficients across covariates and is meaningless if they live on different scales.

Figure 2 shows the estimator across the whole range of λ on a data-scarce version of chapter 4's problem: the first 100 regulars, log spend as the response, and all twenty-nine candidate columns (visits, household, distance, the items twin, and the twenty-five junk attributes of the testing screen). At the OLS end the collinear pair (visits, items) carries large coefficients of opposite signs; as λ grows, two large opposed coefficients become expensive, the pair is pulled onto a shared stable path, and the junk coefficients collapse toward zero.

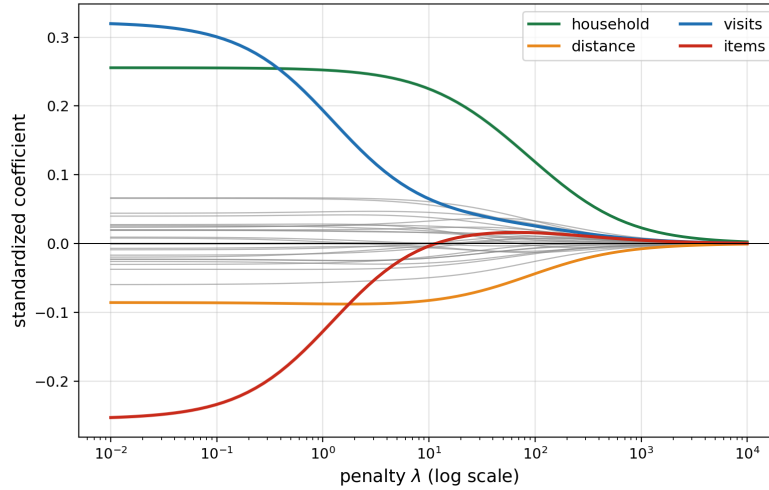


Figure 2: Ridge coefficient paths on the 100-customer file with all 29 covariates (standardized). At $\lambda \rightarrow 0$ (OLS), the collinear pair is out of control: visits at $+0.32$, items at -0.25 , opposite signs for two near-copies of each other. The penalty pulls the pair together and collapses the junk paths (gray) toward zero.

3.3 The lasso

Replacing the squared penalty by an absolute one gives the *lasso* [2],

$$\hat{\beta}_{\lambda}^{\text{lasso}} = \arg \min_{\beta} \frac{1}{n} \|\mathbf{y} - \mathbb{X}\beta\|^2 + \lambda \sum_j |\beta_j|. \quad (13)$$

The change looks minor and is not: the absolute value’s corner at zero makes the minimizer set coefficients *exactly* to zero once λ is large enough. The lasso solution is *sparse*, and sparsity is *feature selection* performed by the fit itself: a junk column’s coefficient does not merely become small, it disappears from the model, and the covariates that survive are the ones the fitted predictor actually uses. There is no closed form; we keep the lasso as an optional extension of the same idea, used without its algorithms and theory.

Beyond least squares. Nothing in the recipe is specific to regression. Adding $\lambda\|\beta\|^2$ or $\lambda\sum_j |\beta_j|$ to the log-loss objective (7) gives penalized logistic regression, with the same benefits; and the template (10) applies to *any* family parametrized by a coefficient vector, whatever the loss.

3.4 Cross-validation

We want the value of λ under which the learned predictor generalizes best, that is, the one whose model has the smallest loss on unseen data. Choosing λ is therefore an optimization problem in its own right, over λ this time, and a direct strategy suggests itself: pick a grid of candidate values $\lambda_1 < \dots < \lambda_m$, train one model per value, and keep the value whose model errs least on rows held out from its training. The training error itself cannot be used for this, since it would choose $\lambda = 0$ mechanically. So: carve a validation split off the training data, and compare the candidates there? Two objections refine the idea. First, the chosen value would depend on the particular random split: with a different draw of validation rows a different λ can win, and on a small file the swings are large. Second, the test set must not do this job: it is reserved for the final assessment of the chosen model, and spending it during selection would destroy that role (section 3.5 makes this precise). *K-fold cross-validation* answers the first objection by averaging over splits; figure 3 draws the procedure for one candidate.

1. Partition the training rows into K folds of (nearly) equal size, with $K = 5$ or 10 in practice.
2. For each fold $k = 1, \dots, K$: fit the candidate model on the other $K - 1$ folds, and record the average loss e_k of its predictions on fold k .
3. Report the average CV = $\frac{1}{K} \sum_k e_k$, the CV score of the candidate.

Every row is predicted exactly once, by a fit that never saw it, and every row also serves fitting. The score estimates the risk of the candidate trained on about $n(1-1/K)$ rows (slightly pessimistic for the final refit on all n), and it is a random estimate: the K fold errors provide a spread, and reporting their standard error alongside the average is the honest practice, with the caveat that folds share training rows, so the naive standard error is an approximation.

Cross-validation is the second resampling method of this course, after chapter 4's bootstrap. The bootstrap redraws rows *with replacement* and refits, to estimate how an *estimator* fluctuates; cross-validation *partitions* the rows and refits, to estimate how well a *procedure* predicts rows it did not see. Both replace unavailable fresh draws from the population by reuses of the one sample at hand; they differ in what they estimate.

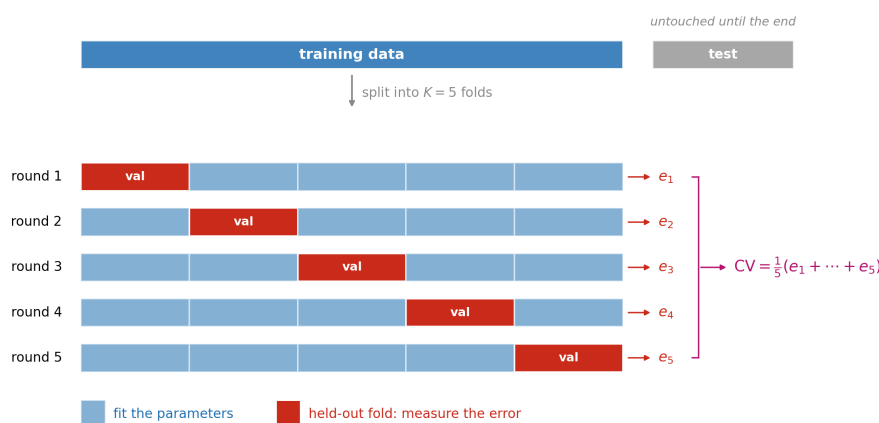


Figure 3: Five-fold cross-validation. The test set is put aside untouched; the training rows are split into five folds, each fold serving once as validation for a fit on the other four; the five fold errors are averaged into the CV score.

Key idea

The quality of a learning algorithm cannot be judged on the same observations that were used to fit it. The rest of the course applies this rule to every algorithm it meets, and cross-validation is its standard tool.

3.5 Choosing hyperparameters

Cross-validation can now be put to work: compute the five-fold CV score of ridge at each of twenty-five grid values of λ , and keep the minimizer. Figure 4 shows the result for the store's problem (five folds on the 100 training customers, all twenty-nine covariates). The colored curves are the five single-fold scores, and they disagree: one fold's minimum sits near $\lambda = 8$, another's near 150. Each is what one lone validation split would have reported, the split-dependence that motivated cross-validation in the first place. Their average, the CV score, is the stable bold curve, with its minimum at $\lambda_{\min} = 21.5$.

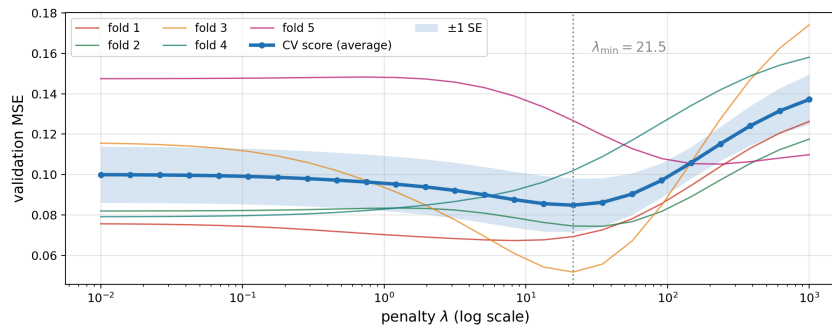


Figure 4: Five-fold cross-validation of ridge on the 100-customer file (29 covariates). Colored curves: each fold’s validation MSE across the λ grid; bold curve: their average, the CV score, with a ± 1 standard-error band, minimized at $\lambda_{\min} = 21.5$. Refit at $\lambda_{\min}$ on all 100 rows, the tuned model reaches 0.081 on the untouched test customers, against 0.091 for plain OLS.

Selecting a model and *evaluating* it are different acts, and the same score cannot do both. The minimum of many noisy estimates is biased downward: among twenty-five candidate values of λ , the winner won partly on merit and partly on luck, and its displayed score inherits the luck. This is the same phenomenon as chapter 4’s multiple testing. (A common conservative variant, the one-standard-error rule, picks the largest λ whose CV score sits within one standard error of the minimum, the standard error being read off the spread of the fold curves.)

The remedy is a nested division of labor: *training* data fit each candidate; *validation* data (here, the cross-validation folds) choose among candidates; and an untouched *test* set, consulted once at the very end, evaluates the chosen model. Anything that influenced a choice belongs to the first two tiers. On the store’s file, tuned ridge reaches a test error of 0.081 against 0.091 for plain OLS on the same twenty-nine covariates, an eleven percent improvement, obtained by a deliberately constrained fit whose λ was set by validation, never by the test set.

4 Conclusion

The table below summarizes the chapter’s machinery; every learning algorithm met from here on can be read through its three rows:

component	in this chapter	the general principle
model family	linear, logistic, softmax; polynomial features	the hypothesis class $\mathcal{F}$
complexity control	degree; penalty λ	restrict $\mathcal{F}$, or prefer simple f
validation	test data; K -fold CV	judge on data that did not fit

What was given up relative to chapter 4 is everything it derived so carefully: no exact law attaches to a penalized coefficient, no t -test bears on it. In exchange, the chapter gained predictors for continuous and discrete targets alike, that hold up on unseen data, that no collinear pair or junk column can destabilize, and whose complexity is chosen from estimated risk. As throughout the course, dropping assumptions gives up inference and relies on the data instead.

The predictors of this chapter all consumed a ready-made numerical table (every input already a number, on a sensible scale, with nothing missing). Real files rarely arrive in that form. Where that table comes from, and why its construction is part of the learning problem itself, is the next chapter’s subject.

References

- [1] A. E. Hoerl and R. W. Kennard. Ridge regression: biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.
- [2] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58(1):267–288, 1996.