

Risk, generalisation and model selection

Prof. Dr. Emre Onur Kahya · Department of Physics Engineering, Istanbul Technical University

This note turns the informal goal “predict well on new data” into mathematics and into a working procedure. It defines expected and empirical risk, explains why training error is optimistic, and shows how to estimate generalisation honestly with held-out data and cross-validation. Capacity, under- and overfitting, the bias–variance trade-off and regularisation explain how to choose a model; the double-descent curve and scaling laws show how deep learning has refined, without overturning, this classical picture. It is the methodological core of the course: every later method is selected and evaluated in this way.

1 A learning algorithm has three ingredients

Almost every learning algorithm combines, explicitly or implicitly,

1. a **family of functions** $\mathcal{F} = \{f_\theta\}$ indexed by parameters θ ;
2. a **loss** $L(f(\mathbf{x}), y)$ that measures the cost of predicting $f(\mathbf{x})$ when the truth is y (Note 1, Table 2);
3. an **optimisation procedure** that searches $\mathcal{F}$ for a function with small average loss on the training data.

Families range from the trivial to the enormous: constant predictors $f(\mathbf{x}) = b$; affine functions $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$; polynomials of degree p ; the piecewise-constant functions of histograms (Note 3); weighted sums of kernels centred on the data (Note 5); decision trees; neural networks, where each architecture defines a different family. Choosing the family is the most consequential decision in a project, and it is the subject of this note.

2 Measuring quality

What we ultimately care about is rarely the loss the algorithm minimises. A hierarchy of measures is common:

1. the real objective in the full system — lives saved, money earned, the statistical significance of a physics measurement;
2. a field-specific metric — BLEU for translation, area under the ROC curve for a trigger, word error rate for speech;
3. a generic metric — classification error, precision and recall, mean squared error;
4. the training loss actually optimised.

The lower levels are proxies for the higher ones, and the gap between them deserves attention.

Surrogate losses. The classification error $\mathbf{1}\{f(\mathbf{x}) \neq y\}$ is piecewise constant in the parameters, so its gradient is zero almost everywhere, and even for linear classifiers minimising it exactly is computationally hard. One therefore trains a real-valued score and minimises a smooth *surrogate* that upper-bounds or tracks the error:

Model output	Surrogate loss	Decision rule
$g(\mathbf{x}) \approx P(Y = 1 \mid \mathbf{x})$	binary cross-entropy $-y \log g - (1 - y) \log(1 - g)$	$f(\mathbf{x}) = \mathbf{1}\{g(\mathbf{x}) > \frac{1}{2}\}$
$g(\mathbf{x}) \approx (P(Y = k \mid \mathbf{x}))_k$	cross-entropy $-\log g_y(\mathbf{x})$	$f(\mathbf{x}) = \arg \max_k g_k(\mathbf{x})$
score $g(\mathbf{x})$, $t = 2y - 1 \in \{\pm 1\}$	hinge $\max(0, 1 - t g(\mathbf{x}))$	$f(\mathbf{x}) = \mathbf{1}\{g(\mathbf{x}) > 0\}$
scores $g_1(\mathbf{x}), \dots, g_m(\mathbf{x})$	margin $\max(0, 1 + \max_{k \neq y} g_k - g_y)$	$f(\mathbf{x}) = \arg \max_k g_k(\mathbf{x})$

The cross-entropies are negative log-likelihoods of a categorical model (Note 2, Section 2); the hinge and margin losses are used by support vector machines. Probabilistic outputs have an extra benefit: the threshold can be moved after training to trade one kind of error for another.

In physics — efficiency, rejection and the ROC curve

A trigger or tagger outputs a score $g(\mathbf{x})$, and the analysis keeps events with $g(\mathbf{x}) > c$. Lowering c raises the signal efficiency and lowers the background rejection; the curve traced by all thresholds is the ROC curve. The best threshold is fixed by the physics goal — for instance, the expected discovery significance — not by the training loss, which is why classifiers in physics are trained with cross-entropy and then evaluated on the full ROC curve.

Modern connection — training loss versus what we want

Language models are pre-trained to minimise next-token cross-entropy, yet they are judged on answering questions, writing code and following instructions. Closing that gap is the purpose of fine-tuning on human preferences, in which a second model is trained to predict which of two answers people prefer and the language model is then optimised for that learned reward. The hierarchy above, with level 4 refined step by step towards level 1, is a large part of how current assistants are built.

3 Expected risk and empirical risk

Assume, as in Note 1, that examples are i.i.d. draws from an unknown $P(\mathbf{X}, Y)$. The *expected risk* (generalisation error) of a fixed predictor f is its average loss on future data,

$$R(f) = \mathbb{E}_{(\mathbf{X}, Y) \sim P} [L(f(\mathbf{X}), Y)],$$

and its *empirical risk* on a finite set D is

$$\hat{R}(f, D) = \frac{1}{|D|} \sum_{(\mathbf{x}, y) \in D} L(f(\mathbf{x}), y).$$

We cannot compute $R(f)$, because P is unknown, but if f was chosen *without using* D , then $\hat{R}(f, D)$ is an average of i.i.d. terms with mean $R(f)$: it is unbiased and converges to $R(f)$ as $|D| \rightarrow \infty$. Its precision follows from the variance of an average. For the classification error p estimated on n_{test} examples the standard error is $\sqrt{p(1-p)/n_{\text{test}}}$; with $p = 10\%$ and 1000 test examples that is about ± 0.95 percentage points. Differences between models smaller than this are not evidence of anything.

Empirical risk minimisation. Since the expected risk is out of reach, learning minimises its estimate on the training set,

$$\hat{f} = \arg \min_{f \in \mathcal{F}} \hat{R}(f, D_{\text{train}}).$$

This is the training phase. But now $\hat{f}$ was chosen using D_{train} , specifically to make $\hat{R}(\hat{f}, D_{\text{train}})$ small, so the training error is biased downwards as an estimate of $R(\hat{f})$ – often drastically, as the histogram classifier of Note 3 showed (5.2% training error, 34.6% test error).

Never evaluate on data used for choosing $\hat{R}(f, D)$ is an honest estimate of $R(f)$ only if D played no part in choosing f – not in fitting the parameters, not in choosing hyperparameters, not in pre-processing, not in deciding which model to report.

4 Estimating generalisation

Hold-out. Shuffle the data and split it: train on one part, report the empirical risk on the other. This is simple and correct, but with little data it wastes examples and gives a noisy estimate.

k -fold cross-validation. Split the data into k blocks of roughly equal size. For $j = 1, \dots, k$, train on all blocks except j and compute the loss on block j ; average the k results. Every example is used once for evaluation and $k - 1$ times for training. Typical values are $k = 5$ or 10 ; $k = n$ is *leave-one-out*. Cross-validation estimates the performance of the *procedure* trained on a fraction $(k - 1)/k$ of the data, not of one particular fitted function.

Splits that respect the structure of the data. Shuffling assumes i.i.d. examples. When that fails (Note 1, Section 2), the split must follow the dependence:

- *Groups*: all measurements of the same galaxy, patient, sample or detector run go into the same block, or the model is tested on objects it has already seen.
- *Time*: for forecasting, train on the past and validate on the block that follows, then roll the window forward (sequential validation). A random split would let the model learn from the future.
- *Simulation and reality*: a model trained and validated on simulation must still be tested on real data, or on a control region of it, before its performance is believed.

Nested cross-validation. When the same small dataset must both select hyperparameters and estimate the error of the whole procedure, use two loops: an outer cross-validation for the estimate and, inside each outer training set, an inner cross-validation for the selection.

5 Capacity, underfitting and overfitting

The *capacity* of a family $\mathcal{F}$ is, informally, how many different patterns it can fit. It usually grows with the number of free parameters, but what matters is the effective number of degrees of freedom: constraints, shared parameters or regularisation can make a large model behave like a small one. For restricted settings capacity can be measured exactly; for example, the Vapnik–Chervonenkis

dimension of affine classifiers in $\mathbb{R}^d$ is $d + 1$, the largest number of points they can label in every possible way.

Figure 1 shows the three regimes on 15 noisy points from $\sin(\pi x)$. A straight line cannot follow the curve (*underfitting*: high training and test error). A polynomial of degree 12 passes close to every point and oscillates wildly between them (*overfitting*: low training error, high test error). Degree 3 captures the trend and ignores the noise.

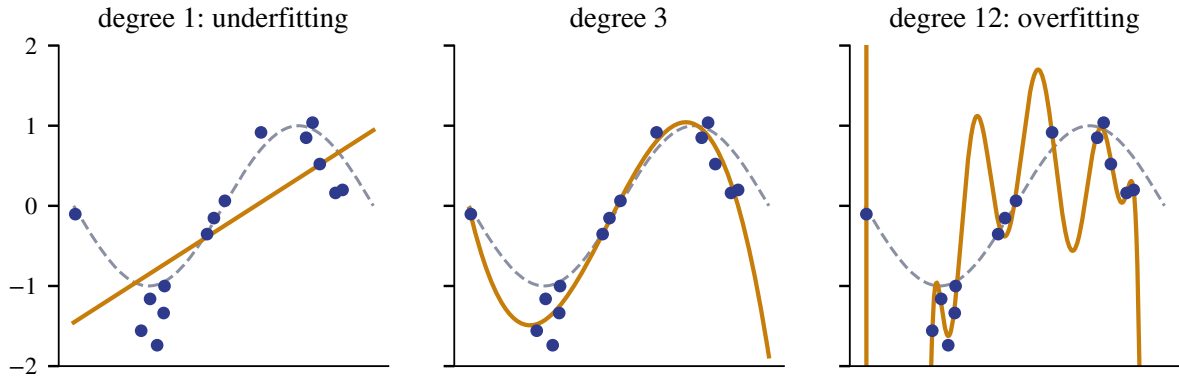


Figure 1. Least-squares polynomial fits to the same 15 points (dashed: the true function $\sin(\pi x)$).

Averaging over many training sets gives Figure 2: training error decreases monotonically with the degree, while validation error first falls, reaches its minimum at degree 3 and then rises steeply. The optimal capacity depends on the amount of data — with more points, higher degrees become affordable. Occam’s razor, “prefer the simplest model that explains the data”, is the qualitative version of this curve.

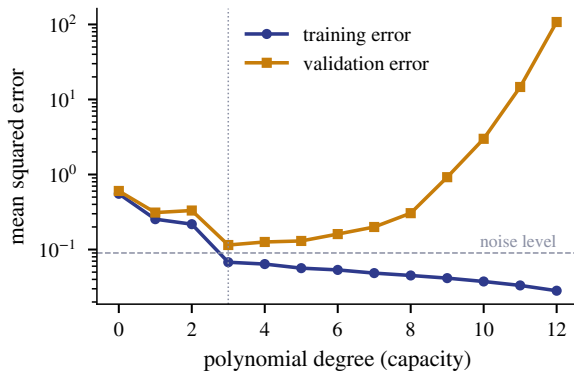


Figure 2. Median training and validation error over 200 training sets of $n = 20$ points. The dashed line is the noise variance 0.09, the smallest error any predictor can achieve.

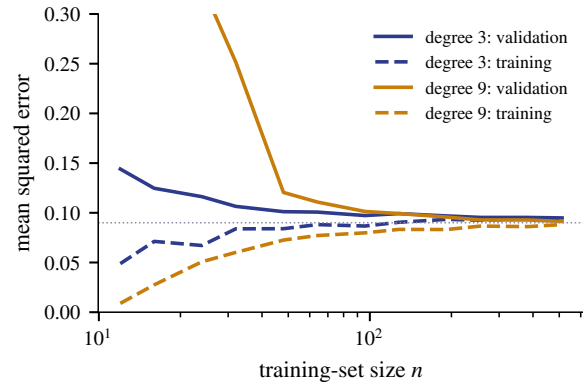


Figure 3. Learning curves: median error against training-set size for degrees 3 and 9. Training and validation error converge to the same limit; the more flexible model needs more data to get there.

6 Where the error comes from

Let $f^* = \arg \min_f R(f)$ be the best possible predictor (its risk is the Bayes risk of Note 1, Section 4), $f_{\mathcal{F}} = \arg \min_{f \in \mathcal{F}} R(f)$ the best member of the family, and $\hat{f}$ the function returned by the algorithm. Then

$$R(\hat{f}) - R(f^*) = \underbrace{R(f_{\mathcal{F}}) - R(f^*)}_{\text{approximation error}} + \underbrace{R(\hat{f}) - R(f_{\mathcal{F}})}_{\text{estimation error}}.$$

The approximation error shrinks as $\mathcal{F}$ grows; the estimation error — the price of having to find $f_{\mathcal{F}}$ from finitely many noisy examples — grows with the size of $\mathcal{F}$ and shrinks with n .

Bias and variance for squared loss. Let $y = f^*(\mathbf{x}) + \varepsilon$ with $\mathbb{E}[\varepsilon] = 0$ and $\text{Var}(\varepsilon) = \sigma^2$, and let $\hat{f}_D$ be trained on a random training set D . At a fixed $\mathbf{x}$, averaging over both D and the noise,

$$\mathbb{E}[(y - \hat{f}_D(\mathbf{x}))^2] = \underbrace{\sigma^2}_{\text{noise}} + \underbrace{(f^*(\mathbf{x}) - \mathbb{E}_D[\hat{f}_D(\mathbf{x})])^2}_{\text{bias}^2} + \underbrace{\text{Var}_D(\hat{f}_D(\mathbf{x}))}_{\text{variance}}. \quad (1)$$

Bias measures how far the *average* fitted function is from the truth; variance measures how much the fit changes from one training set to another. A richer family lowers bias and raises variance; more data lowers variance (Figure 3), which is why “the best regulariser is more data”. The proof extends the decomposition of Note 1, Section 4, by one line (Exercise E2).

7 Model selection

Parameters and hyperparameters. *Parameters* are fitted by the training algorithm; *hyperparameters* are fixed before training and control the family, the preference among its members or the optimisation. They usually control capacity, which is why they cannot be chosen on the training set: the training error would always favour the largest capacity.

Method	Hyperparameters	Parameters
Histogram (Note 3)	number and placement of bins	counts in each cell
k -NN (Note 5)	k , the distance	none beyond the stored training set
Parzen windows (Note 5)	kernel and its width	none beyond the stored training set
Polynomial regression	degree, regularisation strength λ	coefficients
Neural network	architecture, learning rate, batch size, training time, λ	weights and biases

Algorithm 1 Model selection with a training, a validation and a test set

- 1: Shuffle (or split by group or time, Section 4) and divide D into D_{train} , D_{valid} , D_{test}
 - 2: **for** each candidate algorithm A and each hyperparameter setting λ **do**
 - 3: $\hat{f}_{A,\lambda} \leftarrow A_{\lambda}(D_{\text{train}})$ ▷ fit parameters
 - 4: $e_{A,\lambda} \leftarrow \hat{R}(\hat{f}_{A,\lambda}, D_{\text{valid}})$ ▷ out-of-sample score
 - 5: **end for**
 - 6: $(A^*, \lambda^*) \leftarrow \arg \min e_{A,\lambda}$
 - 7: Optionally refit: $f^* \leftarrow A_{\lambda^*}^*(D_{\text{train}} \cup D_{\text{valid}})$
 - 8: **return** f^* and the estimate $\hat{R}(f^*, D_{\text{test}})$ ▷ D_{test} used once, at the very end
-

The validation error of the winner is itself optimistic — it was the minimum over many candidates — which is why a separate test set is needed. Cross-validation can replace the single validation set in line 4. For more than two or three hyperparameters a grid becomes expensive; random search covers the important directions far more efficiently, because usually only a few hyperparameters matter [2].

Regularisation. Instead of choosing among nested families, one can keep a rich family and add a preference for simple members,

$$\hat{f}_\lambda = \arg \min_{f \in \mathcal{F}} \hat{R}(f, D_{\text{train}}) + \lambda \Omega(f),$$

where Ω penalises complexity and $\lambda \geq 0$ is a hyperparameter. Weight decay ($\Omega = \|\mathbf{w}\|^2$) is the most common choice; stopping the optimisation early, injecting noise (dropout) and augmenting the data with transformed copies act as implicit regularisers. Shrinking the covariance estimate in Note 2 is the same idea.

In physics — blind analysis

Particle and nuclear physicists guard against tuning an analysis towards the answer they expect by *blinding*: the signal region, or the final numerical result, stays hidden while selection criteria and corrections are fixed on simulation and control regions, and is revealed only once [6]. This is Algorithm 1 with the test set locked in a drawer — the discipline that keeps $\hat{R}(f^*, D_{\text{test}})$ honest.

Modern connection — test sets wear out

Benchmarks are reused by thousands of papers, so the test set slowly becomes a validation set. When ImageNet’s test set was re-created by following the original collection process, accuracies dropped by 11–14 percentage points, although the ranking of models was largely preserved [9]. For language models trained on much of the public web, a second danger is *contamination*: benchmark questions and their answers may be in the training data. Fresh, private or time-stamped test sets are the remedy.

8 Beyond the U-shaped curve

Modern neural networks often have far more parameters than training examples and can fit even randomly labelled data [10], yet they generalise well. Figure 4 shows a simple model with the same behaviour: least squares on p random nonlinear features of $n = 100$ training points. As p grows the test error first follows the classical U, then rises sharply as p approaches n — where the model can just interpolate the data and the fit is extremely sensitive to noise — and then *decreases again*. Beyond $p = n$ there are infinitely many interpolating solutions, and the algorithm returns the one of smallest norm; as p grows, that implicit preference makes the fit smoother. In this example the test error at $p = 6000$ (0.134) is below the best value of the classical regime (0.141).

This *double descent* [1, 8] does not overturn the ideas of this note; it refines what “capacity” means. The number of parameters is a poor measure of it; the norm of the solution, set implicitly by the optimiser, matters more. Validation still decides. The other modern lesson concerns scale: for language models, test loss falls as a smooth power law in model size, data and compute over many orders of magnitude [5], and for a fixed compute budget model size and number of training tokens should grow together, at roughly twenty tokens per parameter [4]. Learning curves like Figure 3, measured on small models, are now used to plan the training of large ones.

Summary A learning algorithm is a family, a loss and an optimiser. Training minimises empirical risk, which underestimates the expected risk of the result; honest estimates need data never used for any choice, via hold-out or cross-validation, with splits that respect groups and time. Capacity trades approximation against estimation error (bias against variance); hyperparam-

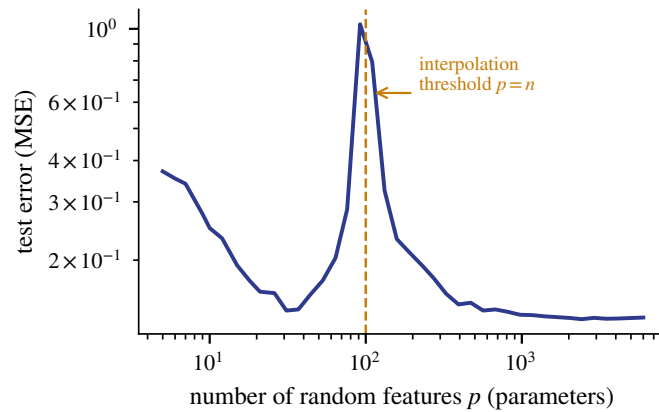


Figure 4. Double descent: median test error of minimum-norm least squares on p random ReLU features of 10-dimensional inputs, $n = 100$ training points, 20 repetitions.

ters and regularisation control it and are chosen on validation data, with the test set used once. Over-parameterised models add a second descent of the test error, but not a new principle.

Exercises

- E1. A classifier has 8.0% error on a test set of 500 examples and another has 7.0%. Estimate the standard error of each. Are the two classifiers distinguishable? How large a test set would make a one-point difference significant at about two standard errors?
- E2. Prove (1). Start from Note 1's decomposition at fixed $\mathbf{x}$ and add and subtract $\mathbb{E}_D[\hat{f}_D(\mathbf{x})]$.
- E3. Explain why choosing k for k -NN by minimising the training error always gives $k = 1$, and what the training error of 1-NN is.
- E4. Ten research groups each try 20 hyperparameter settings on the same public validation set and report the best. Why is the reported number optimistic? Simulate this with a classifier that guesses at random on a balanced binary problem with 200 validation examples.
- E5. Reproduce Figure 2 for $n = 20, 100$ and 1000 and report the degree that minimises the validation error in each case. Then fix degree 12 and $n = 20$ and add weight decay; plot validation error against λ .
- E6. Galaxy images come from 300 telescope pointings, several galaxies per pointing. Design the cross-validation splits, and explain what a naive random split would overestimate.

Further reading. Hastie, Tibshirani and Friedman [3], chapter 7; Murphy [7], chapters 4–5; for double descent, Belkin *et al.* [1].

References

- [1] M. Belkin, D. Hsu, S. Ma and S. Mandal, Reconciling modern machine-learning practice and the classical bias–variance trade-off, *Proc. Natl. Acad. Sci. USA* 116, 15849–15854 (2019).

- [2] J. Bergstra and Y. Bengio, Random search for hyper-parameter optimization, *J. Mach. Learn. Res.* 13, 281–305 (2012).
- [3] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning*, 2nd ed., Springer (2009).
- [4] J. Hoffmann *et al.*, Training compute-optimal large language models, *NeurIPS* (2022).
- [5] J. Kaplan *et al.*, Scaling laws for neural language models, arXiv:2001.08361 (2020).
- [6] J. R. Klein and A. Roodman, Blind analysis in nuclear and particle physics, *Annu. Rev. Nucl. Part. Sci.* 55, 141–163 (2005).
- [7] K. P. Murphy, *Probabilistic Machine Learning: An Introduction*, MIT Press (2022).
- [8] P. Nakkiran *et al.*, Deep double descent: where bigger models and more data hurt, *ICLR* (2020).
- [9] B. Recht, R. Roelofs, L. Schmidt and V. Shankar, Do ImageNet classifiers generalize to ImageNet?, *ICML* (2019).
- [10] C. Zhang, S. Bengio, M. Hardt, B. Recht and O. Vinyals, Understanding deep learning requires rethinking generalization, *ICLR* (2017).

The selection and order of topics in weeks 1–3 follow the course IFT 6390 *Fondements de l'apprentissage machine* at Université de Montréal (taught by Ioannis Mitliagkas, slides by Pascal Vincent). The text, examples, figures and exercises of this note are original. Figures are computed from synthetic data by `figures.py`, published with the sources of these notes.