— 4 —
Holdout method

The holdout method separates training and testing data, permitting anything goes on the training data, while enforcing the iron rule on the testing data. Not all uses of the holdout method are alike.

Source: The Emerging Science of Machine Learning Benchmarks. M. Hardt, 2025. URL: https://mlbenchmarks.org. Compiled on 2025-11-11.

The previous chapter prepared us for the main protagonist of the book: the *holdout method*. The holdout method is an empirical way to estimate the risk of a predictor. The basic idea is this:

1. Split available data randomly into disjoint training and test sets.
2. Use the training set to train a model f.
3. Use the test set to compute the empirical risk $R_S(f)$.

Let's first understand the logic behind separating training from testing and why we need the split in the first place.

4.1 Testing on the training set

A basic insight about machine learning is that we can't use the same sample for both training and testing. In other words, we can't test on the training set. Once we have trained a model f on a sample S, the sample may no longer provide a good estimate of the risk of f on the data-generating distribution. This will sound like a truism for those familiar with machine learning. But the reasons why this is so are not obvious.

Let's start with an extreme example. Consider a learning algorithm $A(S)$ that given a sample S outputs the predictor f so that $f(x) = y$ if the pair (x, y) occurs in the sample S and $f(x) = 0$ otherwise. This predictor is essentially just a *lookup table* that retrieves the label of seen instances. On previously unseen instances, it does nothing useful.

The empirical risk with respect to the zero-one loss of the lookup table is 0 on the sample S, i.e., $R_S(f) = 0$. The lookup table never makes a mistake on the sample, by construction. Outside the sample, however, the lookup table is never better than the constant 0 predictor. Assuming the sample has measure 0 within the population, the risk of the predictor is therefore no better than that of a constant predictor. If labels are balanced, meaning they are half 0 and half 1 on the population, then this predictor achieves $R(f) = 1/2$.

This simple example shows that $R_S(f)$ can be a terrible estimate of $R(f)$ when the predictor f is trained on the sample S.

You might object that this algorithm is unreasonable, since it involves no "learning" whatsoever. This is true. But consider a slight modification. On each input x, we compute the nearest point x' in, say, Euclidean distance, so that $(x', y) \in S$ and output $f(x) = y$. This is the well-known nearest neighbor classifier, a staple in learning theory. It certainly qualifies as a reasonable

algorithm. And yet, it has $R_S(f) = 0$ just like our extreme example before. The risk $R(f)$ could be anything depending on the population. There are cases where nearest neighbor classifiers are provably optimal and there are cases where they perform poorly. We just can't tell by looking at the empirical risk $R_S(f)$.

To summarize, when we optimize the model on data points in S, its performance on S is likely much better than its performance on the underlying distribution that S was drawn from. Formally, the model depends on the sample S and therefore $R_S(f)$ is no longer an unbiased estimate of $R(f)$. Recall, in contrast, if we draw a fresh sample T, independently of f, then $R_T(f)$ does provide an unbiased estimate of $R(f)$.

4.2 Generalization

In its narrow definition, generalization refers to the difference between empirical risk and risk. Recall that empirical risk has two purposes in machine learning: *training* and *testing*. Generalization is important in both cases. We'll focus on generalization in the context of model testing. But it's good to be clear about the differences between the two.

The first use of empirical risk is as an *objective function* for model training. For training, machine learning practitioners use various optimization methods to find a model that minimizes empirical risk. In the case of training, the loss function has to be suitable for the optimization method we apply. In Chapter 2, we discussed cross entropy as the canonical example of a training loss. When we use a sample S for training, we call the sample the *training set*. We also call the empirical risk $R_S(f)$ the *training error* or *training loss*. In the golden era of learning theory, researchers aimed to develop learning algorithms that provably output predictors of small generalization gap with respect to the training sample. The theoretical program met with serious obstacles in deep learning. It's been difficult to prove any reliable a priori bounds on the generalization performance of the kind of models people build in practice. As a result, practitioners primarily rely on the holdout method to empirically estimate the risk of a trained model.

This brings us to the second use of empirical risk in *testing* or *evaluation*. Here we assume that we have already obtained a predictor and we want to estimate its loss on the population by using empirical risk as a stand in for risk. For the purpose of model testing the loss function typically corresponds to whatever evaluation metric we choose for our application.

In many applications, the loss function for evaluation is the zero-one loss that counts classification errors. Zero-one loss is the same as one minus classification accuracy. The loss function we use for testing is almost always different from the one for training. In general, we might not even know at evaluation time what happened during training. For instance, we might've downloaded the model we evaluate from the internet. We call the sample S a *test set* when we use the sample for testing. In this case, we call the empirical risk $R_S(f)$ the *test error* or *test loss*.

Definition 1. *The generalization gap is the difference between risk and empirical risk:*

$$\Delta_S(f) = R(f) - R_S(f)$$

The generalization gap tells us how much the empirical risk underestimates the loss of the model f on the population. It's a signed quantity that could be positive or negative. Typically we bound its absolute value. We hope that the generalization gap is small in absolute value for the models that we evaluate.

Overfitting is a colloquial term in machine learning that, roughly speaking, describes various situations where we observe a large generalization gap. As such, overfitting can arise both during model training and in the context of evaluation. The empirical phenomena related to overfitting during training and testing, however, are not the same. Model size, for example, plays a major role for training. For testing, model size is often irrelevant. We typically treat the model as a *black box* and evaluate it on the sample we have. Overfitting is a bit of catch-all phrase for problems with machine learning that go beyond the narrow definition of large generalization gap.

4.3 The holdout method

The fact that we cannot test on the training data has motivated a common sense heuristic: Split the available data into two disjoint sets, a training set and a testing set. In practice, we let the community apply the anything goes principle to the training set, not limiting practitioners in how they utilize the available training data. Whatever the result of training is, we evaluate its performance on the test set.

The idea to split training data randomly into training and test sets is called *holdout method.* The test set is therefore also called *holdout* set. The intuition for the holdout method is this. Suppose we fit one model f on the training

set and evaluate it on the test set S. Assume the sample S was drawn iid from the population (X, Y). Then, by linearity of expectation and the fact that each sample is distributed identically,

$$\mathbb{E}\, R_S(f) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}\, \ell(f(x_i), y_i) = \mathbb{E}\, \ell(f(X), Y) = R(f).$$

Moreover, computing the variance of $R_S(f)$ and using independence between the samples, we have.

$$\mathbb{V}\, R_S(f) = \frac{1}{n^2}\sum_{i=1}^{n} \mathbb{V}\, \ell(f(x_i), y_i) = \frac{1}{n}\mathbb{V}\, \ell(f(X), Y)$$

For a bounded loss function $\ell\colon \mathcal{Y} \times \mathcal{Y} \to [0, 1]$, we must have $\mathbb{V}\, \ell(f(X), Y) \leq 1/4$. This is because a fair coin tosses maximizes variance among all distributions supported on $[0, 1]$ and its variance is $1/4$. To summarize,

$$\mathbb{E}\, R_S(f) = R(f) \qquad \text{and} \qquad \mathbb{V}\, R_S(f) \leq \frac{1}{4n}.$$

This means that we should expect the empirical risk to approximate the true risk up to a $1/\sqrt{n}$ error:

$$R_S(f) \approx R(f) \pm \frac{1}{\sqrt{n}}$$

Put differently, if we draw *error bars* around the empirical risk of width $1/\sqrt{n}$ we can be reasonably sure that these error bars contain the risk. We'll make the idea of error bars precise further down. But the point is that the number we have could always be this much higher or lower depending on random chance. If we have $10,000$ test cases, we can report loss estimates up to a resolution of two digits after the decimal point.

An equivalent way to think about it is also quite useful. If we want to create a test set so that empirical risk and risk are ϵ-close, we need at least $n \geq 1/\epsilon^2$ samples. This is what's called a sample size calculation. The requirement here is non-trivial. For example, if our goal is to estimate an accuracy number of to 1% error, we need about $10,000$ samples. The argument in Chapter 3 revealed that we can't avoid this cost. As a consequence, it's common for machine learning test sets in practice to have tens of thousands of data points.

We can make the sample requirements more precise using Hoeffding's bound from the previous chapter.

Proposition 1. *Fix a function $f : \mathcal{X} \to \mathcal{Y}$ and fix a bounded loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to [0,1]$. Let $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ denote a random sample of size n where each data point (x_i, y_i) is drawn independently from the same underlying distribution. Then, for every $\delta > 0$,*

$$\mathbb{P}\left\{|\Delta_S(f)| \leq \sqrt{\frac{\log(2/\delta)}{2n}}\right\} \geq 1 - \delta.$$

Proof. Let $a = \sqrt{\log(2/\delta)n/2}$. Using Hoeffding's inequality from Chapter 3 with $c_i = 1, b_i = 0$, we have

$$\mathbb{P}\left\{|\Delta_S(f)| \geq \sqrt{\frac{\log(2/\delta)}{2n}}\right\} = \mathbb{P}\left\{|nR(f) - nR_S(f)| \geq a\right\} \leq 2e^{-2a^2/n} = \delta.$$

□

Guarantees like this are sometimes called *generalization bounds*, since we bound the generalization gap. Plugging in $\delta = 0.05$, we get that $|\Delta_S(f)| \leq 1.36/\sqrt{n}$ with 95% probability. We can make the failure probability very small at a slight increase in sample size.

Model selection

Let's extend our sample size calculation to a slightly more challenging problem called *model selection*. Given a candidate set of k classifiers $f_1, \ldots, f_k$, we want to choose the approximately best among them. To get a bound on the generalization gap of the chosen model, we need to bound the largest possible generalization that could occur among the k functions.

Proposition 2. *Fix k classifiers $f_1, \ldots f_k : \mathcal{X} \to \mathcal{Y}$ and fix a bounded loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to [0,1]$. Let $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ denote a random sample of size n where each data point (x_i, y_i) is drawn independently from the same underlying distribution. Then, for every $\delta > 0$,*

$$\mathbb{P}\left\{\max_{1 \leq t \leq k} |\Delta_S(f_t)| \leq \sqrt{\frac{\log(2k/\delta)}{2n}}\right\} \geq 1 - \delta.$$

Proof. Fix an arbitrary function $f : \mathcal{X} \to \mathcal{Y}$. By Hoeffding's bound, for every $\epsilon > 0$,

$$\mathbb{P}\left\{|R(f) - R_S(f)| \geq \epsilon\right\} \leq 2\exp(-2\epsilon^2 n).$$

Set

$$\epsilon = \sqrt{\frac{\log(2k/\delta)}{2n}}$$

and we get

$$\mathbb{P}\{|R(f) - R_S(f)| \geq \epsilon\} \leq \frac{\delta}{k}.$$

Apply this bound to each of the functions $f_1, \ldots, f_k$. Then, take the union bound over all k functions to conclude

$$\mathbb{P}\left\{\max_{1 \leq t \leq k} |R(f_t) - R_S(f_t)| \geq \epsilon\right\} \leq k \cdot \frac{\delta}{k} = \delta.$$

□

What's remarkable about this bound is that the dependence on the number k of models is only $\sqrt{\log k}$. This means we can, in principle, compare exponentially (in n) many models before the bound becomes trivial. So long as $k = 2^{o(n)}$, we still have a maximum error of $o(1)$.

The iron vault

We just saw powerful theoretical guarantees of the holdout method. Due to concentration of measure, we can evaluate k models with error about $\sqrt{\log(k)/n}$. This is an impressive guarantee, but there is one major caveat.

Unfortunately, these guarantees are for *one-time use*. They only hold if the holdout set is fresh, that is, iid with respect to the population of interest. The moment the model depends on the holdout sample, these guarantees go out of the window. In theory, this makes perfect sense. Remember the example of the lookup table that stores the sample. This is a model that depends on the sample. We can't hope to have any guarantees in this case. That's why statistical authority cautions against using a holdout set twice:

> Ideally, the test set should be kept in a "vault," and be brought out only at the end of the data analysis. Suppose instead that we use the test-set repeatedly, choosing the model with smallest test-set error. Then the test set error of the final chosen model will underestimate the true test error, sometimes substantially.[1]

Think about how restraining it is to keep the holdout data in a vault. Suppose you train a model and after evaluating it on the holdout data, you realize you made a silly mistake. If you stick to the promise of keeping the

data in a vault, you can't update your model and evaluate it again on the same holdout. You'd now have to collect an entirely new holdout data set.

Echoing the earlier quote, learning theorist Francis Bach acknowledges the predicament:

> In theory, the test set can be used only once. In practice, this is unfortunately only sometimes the case. If the test data are seen multiple times, the estimation of the performance on unseen data is overestimated.[2]

The *iron vault assumption* is fundamentally incompatible with the idea of a machine learning benchmark. The whole point of a benchmark is for different scientists to compare their results on the same data. This necessarily involves using the test set repeatedly. The moment a scientist considers a prior result on a benchmark dataset, the new model now has some formal dependency on the test data that invalidates the classical theory.

What if researchers want to incrementally try out tweaks to their models so as to improve test set performance? Pattern recognition pioneers Duda and Hart call this problem *training on the testing data* in a footnote in their 1973 textbook:

> In the early work on pattern recognition, when experiments were often done with very small numbers of samples, the same data were often used for designing and testing the classifier. This mistake is frequently referred to as "testing on the training data." A related but less obvious problem arises when a classifier undergoes a long series of refinements guided by the results of repeated testing on the same data. This form of "training on the testing data" often escapes attention until new test samples are obtained.[3]

The fear runs like a thread through the history of pattern recognition and machine learning. In a seminal paper from 2011, Torralba and Efros caution:

> [...] one major issue for many popular dataset competitions is "creeping overfitting", as algorithms over time become too adapted to the dataset, essentially memorizing all its idiosyncrasies, and losing ability to generalize.[4]

Scientists individually and collectively do what Duda and Hart call *training on the testing data*. This problem will be the focus of subsequent chapters. We'll confront the problem both theoretically and empirically. The bounds from this chapter will actually be quite useful. They'll serve as a yardstick

for what we can hope for under ideal conditions. They don't give you a bound on how bad things could be. Rather they tell you what the optimistic scenario is. That's also quite useful to have.

4.4 What's the holdout method for?

The holdout method has several different purposes in machine learning practice. We'll discuss three for now:

1. Signals for model development
2. Model ranking and model selection
3. Measurement of capabilities

These different uses pose different demands on what the holdout method needs to deliver, sorted from less taxing to more taxing.

Signals for model development

The first use of the holdout method is during model development. Any model builder typically has their favorite test set that they trust. Model building is a highly iterative and incremental process with lots of trial and error. The holdout method is what supports this incremental approach.

Practitioners incrementally tweak the model while looking for small improvements on the holdout set. Once performance has saturated on the holdout set, model builders scale up the candidate model with greater effort to a much larger dataset of interest.

For example, CIFAR-10 has long been the model development cousin of ImageNet. Evaluation on CIFAR-10 is quick and cheap, allowing for rapid model development cycles.

For this use of the holdout method, we don't care about the exact number on the holdout set at all. We care about model comparisons to some degree, but they don't have to be perfect either. Primarily, we want that often enough the holdout set points us in the right direction. Some positive correlation between holdout performance and performance on the actual target is all we want.

Model ranking

A core part of any machine learning benchmark is the *leaderboard*, a ranking of all models by empirical risk on the holdout set. In particular, a ranking gives us a complete set of model comparisons. For any two models, it suggests which of the two is better. A good benchmark should do a reasonable job in sorting out model comparisons.

If you want to use machine learning for a specific application, be it industrial, academic, or recreational, you pick the benchmark closest to your application. You choose the model that performs best in the benchmark as a starting point for your own application. Then you spend some time building on top of it. A useful benchmark should free you from second-guessing yourself: If your application fails, you know it would've failed if you had started from any lower ranked model.

Model ranking doesn't need the empirical risk numbers to mean anything. We just need the model comparisons to be valid. An "easy" benchmark and a "hard" benchmark might deliver the same rankings. In fact, suppose you start from any classification benchmark with accuracy as a metric. Now, you replace a fraction of labels in the holdout set with random labels. This will change the absolute accuracy numbers in the benchmark significantly, moving them all lower. But the transformation of accuracy is monotone. In particular, the ranking is preserved, up to statistical fluctuation affecting very close models.

There's an assumption in the community that better benchmarks capture more realistic data. But for the purpose of model ranking toy data or noisy data can be sufficient. Model ranking is the core function of academic benchmarks. A good ranking should guarantee that the top model on the shelf is the best starting point for additional development and downstream applications.

Model ranking is also the key component of machine learning competitions. Machine learning competitions are closely related to benchmarks, although they differ in some significant ways. Typically, the sponsor of a machine learning competition provides a monetary reward for the highest scoring submission. The prediction task reflects a problem that's of interest to the sponsor. Teams can upload models to the competition host and receive a score from the holdout set. The host of the competition keeps the holdout set private. Kaggle was a startup, founded in 2010, that organized hundreds of machine learning competitions, before Google acquired Kaggle in 2017.

Kaggle used a three way split: A training data set, a holdout set to compute the public leaderboard, and another test set to compute the private leaderboard that determines the winner in the end. Kaggle would actually release the feature vectors for points in the holdout set, and withhold only the labels corresponding to the feature vectors. This has the advantage that participants can compute the predictions of their models themselves locally and submit only the predictions.

Although the private leaderboard determined the winner, the public leaderboard was often more important. After all, for the duration of the competition, the public leaderboard was the focus of everyone's attention in the competition. Once the competition ended, attention quickly faded. The main value proposition of a competition was typically not the winning submission, but rather the community of data scientists that emerged around the problem. Winning submissions in data science competitions frequently just combined standard techniques, feature engineering, and model ensembling in clever, but somewhat hacky ways. The production value of the model was therefore limited. What the sponsor of a competition gained was visibility and an edge in recruiting available data scientists.

Machine learning competitions *gamify* leaderboard climbing. They build momentum out of competition over the public leaderboard. Since the leaderboard is subject to competitive pressure, it's important that it works reliably. The absolute performance numbers in competitions don't matter.

Measuring capabilities

The most ambitious use of the holdout method is in measuring specific capabilities, such as reasoning abilities, professional legal training, software engineering, college-level mathematical problem solving skills, to name a few.

Measuring capabilities requires an additional ask from the holdout method—and it's a significant one. Whatever statistical quantity the benchmark reports must be a good *measurement* of the skill that we're interested in. Used this way, the holdout method is analogous to an IQ test. We hope that the number we get is a valid measurement of an underlying theoretical *construct* like intelligence.

Measurement theory is an old field with numerous consequential applications, such as education testing. Measurement theory distinguishes between the so-called *construct* we're trying to measure and the statistical proce-

dure we use to create a numerical representation of it. For example, a well-designed educational test gives a valid representation of a student's ability to navigate specific intellectual tasks. There's a rich scholarly tradition of arguing about the *validity* of measurement. The criteria that make a measurement valid are broad and demanding.

Whether or not the holdout method measures any particular construct validly is not so much about the mechanics of the holdout method. It's a lot more about the specific data and loss function that we choose. You might ask why don't we just use whatever tests we already use for humans and apply these to machine learning models. Unfortunately, this does not necessarily ensure validity. The reason is that measurement about human skills hinges on theories of human psychology and development. The assumptions underlying the measurement therefore may not extend to statistical models.

We sorted the different uses of the holdout method in increasing order of how demanding the use is. Providing signals in the model training loop is typically the least demanding ask. Ranking is often possible without requiring that the data is particularly *realistic*. Measurement of capabilities is the most taxing use of the holdout method. It requires the whole slew of validity criteria that apply to measurement.

4.5 Variants of the holdout method

In the early days of machine learning, sample sizes were small. It seemed wasteful to allocate a sizable fraction of data to testing without ever training on these data points. For this reason, there are a number of cross validation heuristics that make use of the entire sample for both training and testing purposes.

More than two splits

You will often see more than two splits in practice. A common one is a three way split into training, validation, and testing sets. The formal guarantees for the validation and testing sets are the same though. If you haven't touched the data, you get an unbiased estimate that enjoys the strong guarantees above.

The three way split reflects a certain practice that used to be more common. First, we come up with a bunch of models on the training set. Second, we select the best model according to empirical risk on the validation set. Last,

we obtain a final risk estimate for the selected model on the test set. In other words, we use the validation set for model selection and the test set for evaluation of the final model.

Remember that above we derived guarantees for model selection bounding the largest deviation between risk and empirical risk in a family of models. These guarantees apply to the model selection step on the validation set. However, once we select the best model on the validation set, the empirical risk of the best model on the validation set is no longer an unbiased estimate of the risk. The reason is that the model selection step favors outliers. To see, this imagine that all models are just doing random guessing and all have the same risk. Then the model you select is simply the model with the greatest deviation between risk and empirical risk. The additional test set will detect the problem.

Beyond that, there is not much we get from the three way split. Fundamentally, we still need the iron vault assumption for the guarantees to hold. For example, after finding out that the final test performance of the chosen model is lower than its empirical risk on the validation set, we can't go back and start all over, *in theory*.

k-fold cross validation

In k-fold cross validation, we partition the sample randomly into k equally sized parts $S = (S_1, \ldots, S_k)$. For each value $t \in \{1, \ldots, k\}$ we train a model f_t on all but the t-th part, also called *fold*. We then use the t-th fold as a test set to compute $R_{S_t}(f_t)$. The k-fold cross validation estimate is given by averaging out these k estimates:

$$R_k = \frac{1}{k} \sum_{t=1}^{k} R_{S_t}(f_t)$$

Unlike the holdout method, k-fold cross validation uses each and every sample once for testing.

There's a subtlety with k-fold cross validation. Since we train k models, it's not clear what the single model is whose loss the quantity R_k is an estimate of. It makes sense to consider the randomized classifier f that given an input x outputs $f_t(x)$ for a randomly chosen $t \in \{1, \ldots, k\}$. It follows that

$$R(f) = \frac{1}{k} \sum_{t=1}^{k} R(f_t).$$

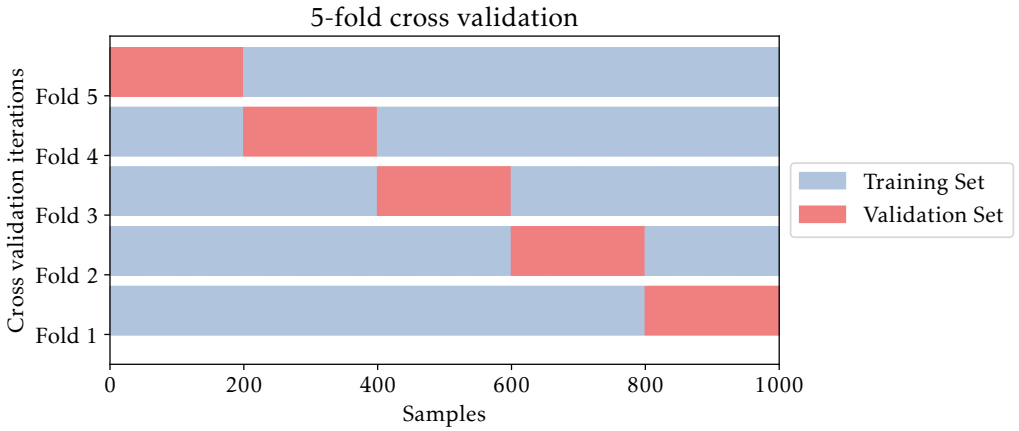

Figure 4.1: Illustration of 5-fold cross validation.

What's less easy to see is why R_k should be a good estimate of the risk $R(f)$. To start, we have that

$$\mathbb{E}\, R_{S_t}(f_t) = R(f_t).$$

This is because f_t was not trained on S_t. Formally, S_t remains iid conditional on f_t. But what else can we say?

Set aside correctness for a moment and imagine all the estimates $R_{S_1}(f_1), \ldots, R_{S_t}(f_t)$ were independent. To be sure, they are *not*. After all, the sample S_t appears in the training data for all models $f_{t'}$ with $t' \neq t$. But if they were independent, we'd get the variance bound

$$\mathbb{V}\, R_k = \frac{1}{k^2} \sum_{t=1}^{k} \mathbb{V}\, R_{S_t}(f_t) \approx \frac{1}{k}\, \mathbb{V}\, R_{S_1}(f_1).$$

The last step is assuming that all variance terms are roughly the same. This looks great, since it's about a factor k smaller than the variance of the holdout method applied to a $1/k$ fraction of the data, i.e., $\mathbb{V}\, R_{S_1}(f_1)$.

This sort of heuristic calculation gives hope that k-fold cross validation might lead to a factor k reduction in variance compared with the basic holdout method. Unfortunately, there are counterexamples showing that, in general, k-fold cross validation is no better than a single fold. At least, it's to show that it's never worse than that. Indeed, by Jensen's inequality, the variance of an average is never larger than the average variance. Therefore,

$$\mathbb{V}\, R_k \leq \frac{1}{k} \sum_{t=1}^{k} \mathbb{V}\, R_{S_t}(f_t) \approx \mathbb{V}\, R_{S_1}(f_1).$$

Notwithstanding this more pessimistic bound, k-fold cross validation empirically does seem to reduce variance as k grows. The empirical phenomenon has motivated a fair bit of research showing reasonable assumptions under which the variance of the estimator indeed shrinks as k grows.

If we take k-fold cross validation to the extreme and set $k = n$, we get *leave-one-out* cross validation. The name refers to the fact that each fold contains only a single data point. We train on $n - 1$ points and test on 1. Then we average out the n estimates. The same estimator is called *jackknife* in statistics circles.

If our earlier heuristic calculation is a good guide, then $k = n$ should achieve the greatest variance reduction. It also has the greatest computational cost. We need to train n different models.

There is a notable case where we can get all n estimates at once. For linear models—and kernel methods, by extension—some clever matrix algebra gives us all n estimates essentially for the price of one. In general, however, that's too much to ask for.

To summarize, k-fold cross validation makes most sense when data are few. When we have sufficient data for training and testing, the plain holdout method is easiest and sufficient.

4.6 Error bars and confidence intervals

When you look at any sample quantity like the empirical risk $R_S(f)$ a predictor, you shouldn't take the quantity literally down to its lower digits of precision. The reason is that the number would likely come out a bit different if we took another random sample of the same size. Remember, we always expect fluctuations of around $\sqrt{1/n}$ where $n = |S|$ is the sample size.

Ultimately, we're interested in the true risk $R(f)$, but the exact value is unknown to us. Since different samples give different estimates, how should we indicate our uncertainty about the true value? It makes sense to report not a single value but an interval of possible values. The interval depends on the sample. It should contain the unknown risk $R(f)$ with high probability over the random sample. This is what statisticians call a *confidence interval*. More informally, *error bars* refer to the end points of the interval, since they are typically plotted as bars above and below the estimate.

Let's make this more precise. Consider the case of estimating the classification error (equivalently, accuracy) of a model. The sample provides n loss

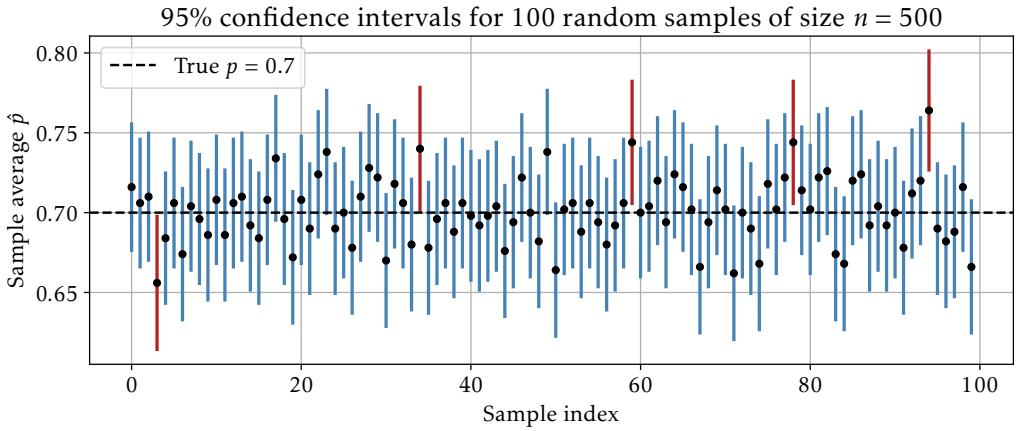

Figure 4.2: Illustration of 95% confidence intervals computed from 100 random samples. We expect that five intervals do not contain the true mean.

values, each of them in $\{0,1\}$. Recall from Chapter 3 that the average of these loss values is a binomial variable scaled by $1/n$ so that its mean $p \in [0,1]$ is the zero-one loss. Moreover, its variance is $p(1-p)/n$. The empirical risk $\hat{p} = R_S(f)$ is an unbiased estimate of p.

We want to come up with an interval $I_n(\hat{p}) \subseteq [0,1]$ based on our sample average $\hat{p}$ so that $\mathbb{P}\{p \in I_n(\hat{p})\} = 0.95$. That is, 95% of the time when we compute the sample average $\hat{p}$ from a random sample of size n, the interval contains the true mean p. Since the sample average $\hat{p}$ is a random variable, so is the confidence interval $I_n(\hat{p})$.

A bit of math shows that we get the approximate guarantee $\mathbb{P}\{p \in I_n(\hat{p})\} \approx 0.95$ by taking the confidence interval

$$I_n(\hat{p}) = \left[\hat{p} - 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}, \hat{p} + 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}\right].$$

The reason is that for typical values satisfying

$$n\hat{p} \geq 50 \quad \text{and} \quad n(1-\hat{p}) \geq 50$$

the scaled Binomial is very similar to a Gaussian variable $N(\mu, \sigma^2)$ with $\mu = p$ and $\sigma^2 = p(1-p)/n$. The expression above follows from what the interval would be under this normal approximation. For a normal distribution, 95% of its mass is within 1.96 standard deviations from its mean.

The same idea also works for general bounded loss functions in $[0,1]$. Here,

the interval we pick is given by

$$R_S(f) \pm z \cdot \frac{s}{\sqrt{n}} \qquad \text{with} \quad z = 1.96,$$

and s is the sample standard deviation

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (\ell(f(x_i), y_i) - R_S(f))^2}.$$

The $n-1$ in the denominator rather than n is for technical reasons that won't matter in machine learning applications. The scalar z is a knob that we can dial to get a certain level of confidence. Since we're working with a normal approximation, we get a 95% confidence interval if we set $z = 1.96$.

There are numerous ways to compute confidence intervals. Even for the simple case of Binomial confidence intervals you'll hear Wald (normal approximation), Clopper-Pearson (strict, conservative guarantees), Wilson Score (often recommended as *default*), Agresti-Coull (adjusted Wald), and Jeffreys Interval (Bayesian). There is no need to memorize these. They all agree for reasonably large values of n when the mean p is not too close to 0 or 1. But even if the sample is relatively small, say, $n = 100$, and the mean $p = 0.9$ is close to 1, all of these intervals have a roughly the same width.

Despite the massive datasets in some domains, small sample sizes have always been a reality in some application areas where data are scarce and costly. In the context of benchmarking, small datasets are actually becoming more common again. One reason is that recent models may perform so much computation at test time that evaluation on a single data point can take hours. As a result, we can only evaluate few data points in a reasonable amount of time. Another reason for the return of small test sets is that models can solve more challenging problems and creating adequate test cases may require significant expertise. We'll dig deeper into these topics from Chapter 9 onward.

Validity of confidence intervals. Confidence intervals demand a bit more from the iid-assumption than the holdout method. For model evaluation, the holdout method only needs that empirical risk is close enough to risk. This could happen for any number of reasons. Even if the strict iid-assumption fails, there may be enough randomness left in the sampling process for things to work out well enough.

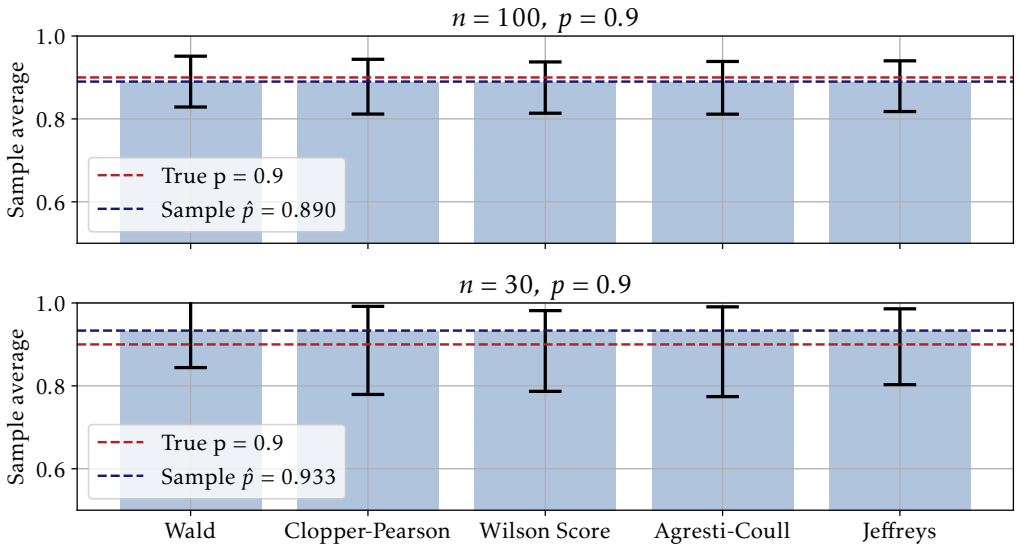

Figure 4.3: Comparison of different Binomial confidence intervals. They are all about the same except in extreme cases.

For model ranking, the conditions are even weaker. We only need that the correct ordering is preserved. This could happen even if empirical risk systematically underestimates risk so long as there is a monotone relationship between empirical risk and risk.

With confidence intervals, however, we're really asking for the shape of the distribution to be of specific kind. For example, above we needed it to be Binomial with certain parameters. As a result, confidence intervals fail more easily than the holdout method. To appreciate this point, consider a simple example: Take your test set and copy each data point 10 times. This doesn't change your benchmark at all. All empirical risk estimates are the same. Model rankings are unchanged. But it will make the confidence intervals overly optimistic!

Bootstrap sampling

There's a useful heuristic, called *Bootstrap*, to get empirical error bars without doing any math. The idea is to check empirically how much our estimate moves around when we repeatedly randomly draw a new sample from the one sample that we have. More precisely, the Bootstrap works like this:

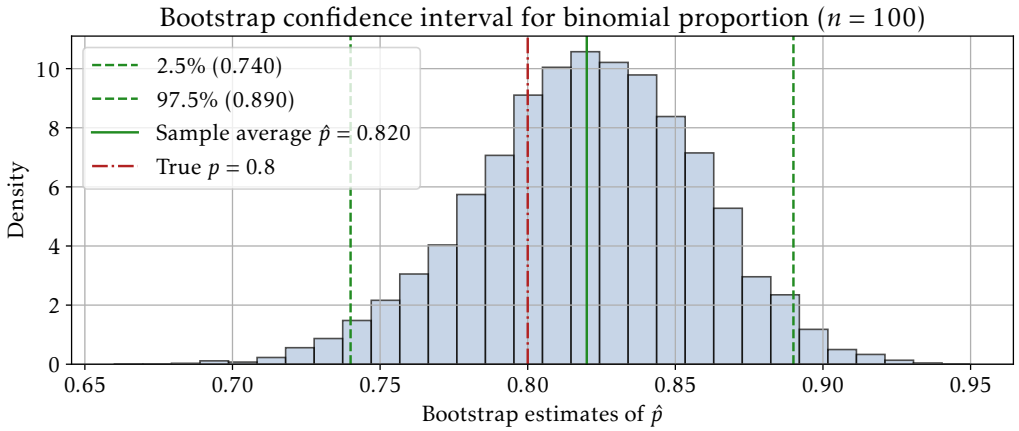

Figure 4.4: Illustration of Bootstrap confidence intervals. The confidence interval is defined by the quantiles of the empirical distribution of the estimated quantity on many repeated samples.

Bootstrap:

1. Compute the n loss values $L = \{l_1, \ldots, l_n\}$, where $l_i = \ell(f(x_i), y_i)$ is the loss of your predictor on the i-th sample.
2. Repeat 1000 times:
   a. Sample n values independently with replacement from L.
   b. Compute the mean of the resampled values.
3. Take the 2.5th and 97.5th percentiles of the 1000 resampled means for a 95% confidence interval.

In other words, the confidence interval is given by quantiles of the observed values under the resampling method. Bootstrap can be especially useful in cases where we have a complicated loss function for which we don't know a mathematically derived confidence interval.

Internal versus external validity of reported numbers

Error bars only tell you about fluctuations due to random sampling. They do not tell you that your estimate replicates under different conditions. In this sense, confidence intervals and error bars are a measure of *internal validity*. Internal validity is about whether your result would replicate if you were to repeat the experiment within the same setup. In the context of statistical work, this means if you were to repeat sampling from the sample population, how much would your estimate wiggle around? This does not—and cannot—

say anything about what would happen on a different population. Your model might fail entirely on a dataset drawn from a different data-generating distribution. How well results transfer from one population to another is a yardstick of *external validity*.

Formal definitions of internal and external validity are somewhat futile, as the concepts vary greatly from one discipline and scientific culture to the next. We therefore commit to a pragmatic rule of thumb throughout: Internal validity is about generalization to *more of the same*. External validity is about generalization to *different populations*.

Error bars and confidence intervals are a measure of internal validity. For this reason, they tend to understate uncertainty about how well a model works. Even if error bars are small, there are many things that can go wrong if you try to apply your model in a different setting than the one you tested. Nevertheless, error bars are better than nothing. If you see large error bars, it's a sign that there is a problem.

Notes

History of the holdout method. Stone's 1974 article is an early study about the holdout method and cross validation for model selection.[5] Stone sketches out a history of the holdout method dating back to the 1930s[6]. Larson observed the shrinkage of the R^2 coefficient of multiple correlation between training and testing a regression model using a random split of the data. Following up on this, Wherry proposed a formula for adjusting the correlation coefficient that avoids sample splitting.[7] Stone also mentions a 1951 *Symposium: The need and means of cross-validation*. Unfortunately, there is little else I could find about the symposium. McCornacks 1959 attributes a method of double cross validation to the symposium.[8]

In the context of pattern recognition, Benjamin Recht and I found that benchmarks go back to a 1959 digit recognition dataset put together by pattern recognition pioneer Bill Highleyman.[9] Stored on a set of punched cards, the dataset featured a modern train-test split.[10] In 1961, Highleyman considered the two problems, accuracy estimation and model selection, we studied here.[11] Specifically, he derived binomial confidence intervals for evaluating the accuracy of classifiers on a finite sample. Keep in mind that Highleyman's study predated Hoeffding's inequality. Even Chernoff's bound was still young at the time. Highleyman then considered the model selection problem and asked what proportion of a sample should be allocated to

testing. Here, his answer wasn't quite right, suggesting that almost all samples should be dedicated to testing.

Benchmarks follow what Liberman calls the *common task framework*.[12,13] Liberman's 2015 talk provides a valuable historical perspective on the rise of benchmarking in the 1980s due to program managers at DARPA who embraced the idea. A 2018 article by Church gives more context.[14] Along these lines, sociologists Koch and Peterson analyze the history of artificial intelligence with an emphasis on the role of benchmarking since the 1980s.[15] Drummond and Japkowicz give an early (2009) critique of modern machine learning benchmarks, anticipating many points that have been raised repeatedly since.[16]

See also Donoho's *50 Years of Data Science* for additional background.[17]

Cross validation and algorithmic stability. Kearns and Ron give guarantees for leave-one-out cross validation under an algorithmic stability assumption.[18] Algorithmic stability says that the algorithm is insensitive to changes in a single data point. Blum, Kalai, and Langford give assumptions under which k-fold cross validation is strictly better than the holdout method.[19] Under a different algorithmic stability assumption, k-fold cross validation provably reduces variance as k grows.[20]

We encountered a stability condition in the context of McDiarmid's inequality in Chapter 3. Algorithmic stability is an important concept in learning theory, since it implies generalization bounds: If a learning algorithm satisfies a suitable stability guarantee, then it must output a classifier for which risk and empirical risk are close.[10,21,22] On the flip side, you might argue that don't really need the holdout method if we run an algorithm with a provable stability guarantee to begin with.

Lei provides a comprehensive study of cross-validation from the perspective of stability.[23] Cross-validation has seen a bit of a revival in statistics in recent years.[23,24]

Problems with the holdout method and cross validation. Rao, Fung, and Rosales show how excessive use of leave-one-out cross validation can lead to biased results.[25] In a similar vein, Reunanen discusses *overfitting in making comparisons between variable selection methods* using leave-one-out cross validation.[26]

Dodge et al. argue that reporting test set performance alone can be misleading and advocate for also reporting the computational budget required

to reach a certain test set performance.[27]

Chapter 9 in *Fairness and Machine Learning* critically examines the test sets from the perspective of measurement and construct validity.[28] The chapter contains many pointers to additional resources on this aspect.

Machine learning competitions and open science platforms. Kaggle is a commercial platform for hosting competitions. The platform hosted numerous competition, before Google acquired the company and pivoted away from machine learning competitions.

CodaLab is academic an open source platform for organizing machine learning competitions.[29]. CodaLab competitions build on CodaLab worksheets that provide tools for doing reproducible experimental and computational workflows (https://codalab.org/). Isabelle Guyon, Percy Liang, Evelyne Viegas are key figures in the creation and development of CodaLab. CodaLab goes back to 2013, predating Jupyter Notebook and Google Colab.

Created in 2014, OpenML is another major effort to create an open platform for sharing datasets, benchmarks, workflows, and model evaluations.[30,31]

Guyon, Pavao, and Viegas are currently editing a book on machine learning competitions, available online at (https://book.chalearn.org/), drawing on many years of experience with competitions.[32] The book contains a wealth of practice insights and background about machine learning competitions. In particular, one chapter of the book surveys competition platforms.[33]

Statistics background. Wasserman's *All of Statistics* is also good background reading for the statistics in this chapter.[34] Chapter 7 of *The Elements of Statistical Learning* covers cross validation and the Bootstrap.[1] Chapter 8 of *A Probabilistic Theory of Pattern Recognition* covers the classical guarantees of the holdout method.[35] For an accessible entry point aimed at machine learning practitioners, Miller discusses recommendations for error bars and confidence intervals in the model evaluation context.[36]

Bibliography

1. Hastie, T., Tibshirani, R. & Friedman, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction (Corrected 12th printing)* (Springer, 2017) (↑ 7, 22).
2. Bach, F. *Learning theory from first principles* (MIT press, 2024) (↑ 8).
3. Duda, R. O. & Hart, P. E. *Pattern Classification and Scene Analysis* (Wiley New York, 1973) (↑ 8).
4. Torralba, A. & Efros, A. A. *Unbiased look at dataset bias* in *Conference on Computer Vision and Pattern Recognition (CVPR)* (2011), 1521–1528 (↑ 8).
5. Stone, M. Cross-validatory choice and assessment of statistical predictions. *Journal of the royal statistical society: Series B (Methodological)* **36,** 111–133 (1974) (↑ 20).
6. Larson, S. C. The shrinkage of the coefficient of multiple correlation. *Journal of Educational Psychology* **22,** 45 (1931) (↑ 20).
7. Wherry, R. J. A new formula for predicting the shrinkage of the coefficient of multiple correlation. *The annals of mathematical statistics* **2,** 440–457 (1931) (↑ 20).
8. Cornack, R. L. M. An Evaluation of Two Methods of Cross-Validation. *Psychological Reports* **5,** 127–130 (1959) (↑ 20).
9. Highleyman, W. H. & Kamentsky, L. A. *A Generalized Scanner for Pattern- and Character-Recognition Studies* in *Western Joint Computer Conference* (1959), 291–294 (↑ 20).
10. Hardt, M. & Recht, B. *Patterns, predictions, and actions: Foundations of machine learning* (Princeton University Press, 2022) (↑ 20, 21).
11. Highleyman, W. H. The design and analysis of pattern recognition experiments. *Bell System Technical Journal* **41,** 723–744 (1962) (↑ 20).
12. Liberman, M. *Reproducible research and the common task method* 2015. https://www.simonsfoundation.org/lecture/reproducible-research-and-thecommon-task-method (↑ 21).
13. Liberman, M. Obituary: Fred Jelinek. *Computational Linguistics* **36,** 595–599 (2010) (↑ 21).
14. Church, K. W. Emerging trends: A tribute to Charles Wayne. *Natural Language Engineering* **24,** 155–160 (2018) (↑ 21).
15. Koch, B. J. & Peterson, D. From protoscience to epistemic monoculture: How benchmarking set the stage for the deep learning revolution. *arXiv:2404.06647* (2024) (↑ 21).
16. Drummond, C. & Japkowicz, N. Warning: statistical benchmarking is addictive. Kicking the habit in machine learning. *Journal of Experimental & Theoretical Artificial Intelligence* **22,** 67–80 (2010) (↑ 21).
17. Donoho, D. 50 years of data science. *Journal of Computational and Graphical Statistics* **26,** 745–766 (2017) (↑ 21).

18. Kearns, M. & Ron, D. *Algorithmic stability and sanity-check bounds for leave-one-out cross-validation* in *Conference on Computational Learning Theory (COLT)* (1997), 152–162 (↑ 21).
19. Blum, A., Kalai, A. & Langford, J. *Beating the hold-out: Bounds for k-fold and progressive cross-validation* in *Conference on Computational Learning Theory (COLT)* (1999), 203–208 (↑ 21).
20. Kale, S., Kumar, R. & Vassilvitskii, S. *Cross-Validation and Mean-Square Stability.* in *ICS* (2011), 487–495 (↑ 21).
21. Bousquet, O. & Elisseeff, A. Stability and generalization. *Journal of machine learning research* **2,** 499–526 (2002) (↑ 21).
22. Shalev-Shwartz, S., Shamir, O., Srebro, N. & Sridharan, K. Learnability, stability and uniform convergence. *The Journal of Machine Learning Research* **11,** 2635–2670 (2010) (↑ 21).
23. Lei, J. A Modern Theory of Cross-Validation through the Lens of Stability. *arXiv:2505.23592* (2025) (↑ 21).
24. Bates, S., Hastie, T. & Tibshirani, R. Cross-validation: what does it estimate and how well does it do it? *Journal of the American Statistical Association* **119,** 1434–1445 (2024) (↑ 21).
25. Rao, R. B., Fung, G. & Rosales, R. *On the dangers of cross-validation. An experimental evaluation* in *International Conference on Data Mining (ICDM)* (2008), 588–596 (↑ 21).
26. Reunanen, J. Overfitting in making comparisons between variable selection methods. *Journal of Machine Learning Research* **3,** 1371–1382 (2003) (↑ 21).
27. Dodge, J., Gururangan, S., Card, D., Schwartz, R. & Smith, N. A. Show your work: Improved reporting of experimental results. *arXiv:1909.03004* (2019) (↑ 22).
28. Barocas, S., Hardt, M. & Narayanan, A. *Fairness and Machine Learning: Limitations and Opportunities* (MIT Press, 2023) (↑ 22).
29. Pavao, A. *et al.* Codalab competitions: An open source platform to organize scientific challenges. *Journal of Machine Learning Research* **24,** 1–6 (2023) (↑ 22).
30. Vanschoren, J., Van Rijn, J. N., Bischl, B. & Torgo, L. OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter* **15,** 49–60 (2014) (↑ 22).
31. Bischl, B. *et al.* OpenML: Insights from 10 years and more than a thousand papers. *Patterns* (2025) (↑ 22).
32. *AI Competitions and Benchmarks: The Science Behind the Contests* (eds Guyon, I., Pavao, A. & Viegas, E.) https://book.chalearn.org/ (ChaLearn, 2024) (↑ 22).
33. Ustyuzhanin, A. & Carlens, H. AI Competitions and Benchmarks: Competition platforms. *arXiv:2312.05185* (2023) (↑ 22).
34. Wasserman, L. *All of statistics: a concise course in statistical inference* (Springer Science & Business Media, 2013) (↑ 22).
35. Devroye, L., Györfi, L. & Lugosi, G. *A probabilistic theory of pattern recognition* (Springer Science & Business Media, 2013) (↑ 22).
36. Miller, E. Adding error bars to evals: A statistical approach to language model evaluations. *arXiv:2411.00640* (2024) (↑ 22).