Machine Learning — Statistical Methods for Machine Learning

Hyperparameter tuning and risk estimates

Instructor: *Nicolò Cesa-Bianchi* version of March 28, 2025

Evaluating a learning algorithm using external cross-validation. Given a learning algorithm A, we focus on the problem of estimating $\mathbb{E}\big[\ell_{\mathcal{D}}(A(S))\big]$ where the expectation is over the random draw of the training set S of size m. In other words, we want to estimate the risk of a typical predictor $A(S)$ generated by A on a training set S of size m.

A trivial estimate is obtained using a dataset of size $m+1$: we use m examples as training set S to obtain $A(S)$, which we then test on the remaining example. This estimate has of course a large variance. To have a better control on the variance, we can use a technique called K-fold (external) cross-validation.

Let S be our entire dataset. We partition S in K subsets (also known as *folds*) $S_1, \dots, S_K$ of size m/K each (assume for simplicity that K divides m). The extreme case $K = m$ provides an estimate known as *leave-one-out*. Now let $S_{-i} \equiv S \setminus S_i$. We call S_i the **testing part** of the i-th fold while S_{-i} is the **training part**.

For example, if we partition $S = \big\{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_{20}, y_{20})\big\}$ in $K = 4$ subsets

$$\begin{array}{ll} S_1 = \big\{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_5, y_5)\big\} & S_2 = \big\{(\boldsymbol{x}_6, y_6), \dots, (\boldsymbol{x}_{10}, y_{10})\big\} \\ S_3 = \big\{(\boldsymbol{x}_{11}, y_{11}), \dots, (\boldsymbol{x}_{15}, y_{15})\big\} & S_4 = \big\{(\boldsymbol{x}_{16}, y_{16}), \dots, (\boldsymbol{x}_{20}, y_{20})\big\} \end{array}$$

then $S_{-2} = S_1 \cup S_3 \cup S_4$.

The *K-fold CV estimate* of A on S, denoted by $\ell_S^{\mathrm{cv}}(A)$, is then computed as follows: we run A on each training part S_{-i} of the folds $i = 1, \dots, K$ and obtain the predictors $h_1 = A\big(S_{-i}\big), \dots, h_K = A\big(S_{-K}\big)$. We then compute the average error on the testing part of each fold,

$$\ell_{S_i}(h_i) = \frac{K}{m} \sum_{(\boldsymbol{x},y) \in S_i} \ell\big(y, h_i(\boldsymbol{x})\big)$$

Finally, we compute the CV estimate by averaging these errors

$$\ell_S^{\mathrm{cv}}(A) = \frac{1}{K} \sum_{i=1}^{K} \ell_{S_i}\big(h_i\big)$$

For $K = m = |S|$, the CV estimate is aptly called leave-one-out estimate,

$$\ell_S^{\mathrm{loo}}(A) = \frac{1}{m} \sum_{t=1}^{m} \ell\big(y_t, h_t(\boldsymbol{x}_t)\big) \qquad \text{where} \quad h_t = A\big(S_{-t}\big)$$

It is relatively easy to prove that, for $K \geq 2$, the K-fold CV estimate for A on a random sample of size $m + m/(K-1)$ is an estimate of the risk of $A(S)$ on a random sample S of size m. We prove this for $K = m + 1$.

Theorem 1. *For any $m \geq 1$ and any learning algorithm A,*

$$\mathbb{E}\Big[\ell_{\mathcal{D}}\big(A(S)\big)\Big] = \mathbb{E}\Big[\ell_{S'}^{\mathrm{loo}}(A)\Big]$$

where the expectation on the left-hand side is over the random draw of a training set S of size m and the expectation on the right-hand side is over the random draw of a training set S' of size $m+1$

Proof. It is convenient to view a training set as a sequence $S = \big((\boldsymbol{X}_1, Y_1), \dots, (\boldsymbol{X}_m, Y_m)\big)$ of m independent random draws from $\mathcal{D}$. Let $A(S)(\boldsymbol{x}) = h(\boldsymbol{x})$ for $h = A(S)$. Let also $Z_t = (\boldsymbol{X}_t, Y_t)$ for all $t \geq 1$ so that $S = (Z_1, \dots, Z_m)$. We still use the notation S_{-i} for $i \in \{1, \dots, |S|\}$, which now denotes the sequence $(Z_1, \dots, Z_{i-1}, Z_{i+1}, \dots, Z_m)$. The first observation is that

$$\ell_{\mathcal{D}}\big(A(S)\big) = \ell\big(Y_{m+1}, A(Z_1, \dots, Z_m)(\boldsymbol{X}_{m+1})\big) = \ell\big(Y_{m+1}, A(S'_{-(m+1)})(\boldsymbol{X}_{m+1})\big)$$

where $S' = (Z_1, \dots, Z_{m+1})$ and $S'_{-(m+1)} = S$. The second observation is that the training sequences

$$(Z_1, \dots, Z_{t-1}, Z_{m+1}, Z_{t+1}, \dots, Z_m, Z_t) \qquad t = 1, \dots, m+1$$

have all the same probability because each Z_t is drawn independently from $\mathcal{D}$. Therefore

$$\mathbb{E}\Big[\ell\big(Y_{m+1}, A(S'_{-(m+1)})(\boldsymbol{X}_{m+1})\big)\Big] = \mathbb{E}\Big[\ell\big(Y_t, A(S'_{-t})(\boldsymbol{X}_t)\big)\Big] \qquad t = 1, \dots, m$$

This implies

$$\begin{aligned}
\mathbb{E}\Big[\ell_{\mathcal{D}}\big(A(S)\big)\Big] &= \mathbb{E}\Big[\ell\big(Y_{m+1}, A(S'_{-(m+1)})(\boldsymbol{X}_{m+1})\big)\Big] \\
&= \mathbb{E}\left[\frac{1}{m+1}\sum_{t=1}^{m+1} \ell\big(Y_t, A(S'_{-t})(\boldsymbol{X}_t)\big)\right] \\
&= \mathbb{E}\Big[\ell_{S'}^{\mathrm{loo}}(A)\Big]
\end{aligned}$$

concluding the proof. □

The choice of the optimal K in K-fold cross-validation depends on the problem. In general it is safe to choose K neither too small nor too large. In practice, $K = 10$ is a typical choice.

Learning algorithms often have hyperparameters. These are special parameters (like k in k-NN or the learning rate in neural networks) whose value must be determined before the training phase can start. Crucially, setting the hyperparameters in the wrong way can lead to underfitting or overfitting. A learning algorithm with one or more hyperparameters is not really an algorithm, but rather a family of algorithms, one for each possible assignment of values to the hyperparameters. Let $\{A_\theta : \theta \in \Theta\}$ be such a family of learning algorithms, where Θ is the set of all possible hyperparameter values.

In practice, we face the problem of choosing the hyperparameters so to obtain a predictor with small risk. This is typically done by minimizing a risk estimate computed using the training data. As Θ may be very large, possibly infinite, the minimization is generally not over Θ, but over a suitably chosen subset $\Theta_0 \subset \Theta$ (for example, if $\Theta = [0, 1]$, then Θ_0 could by a finite grid of equally spaced values in $[0, 1]$).

Tuning hyperparameters on a given training set. Given a training set S, consider the task of finding the predictor $\{A_\theta(S) : \theta \in \Theta_0\}$ with smallest risk,

$$\theta_{\text{opt}} = \operatorname*{argmin}_{\theta \in \Theta_0} \ell_{\mathcal{D}}\big(A_\theta(S)\big)$$

To do so, we split the training data in two subsets S_{train} and S_{dev}. The development set S_{dev} (also called validation set) is used as a surrogate test set. Now our goal becomes that of finding $\theta^* \in \Theta_0$ such that

$$\theta^* = \operatorname*{argmin}_{\theta \in \Theta_0} \ell_{\mathcal{D}}\big(A_\theta(S_{\text{train}})\big) \tag{1}$$

The algorithm is run on S_{train} once for each value of the hyperparameter in Θ_0. The resulting predictors are tested on the dev set to find the value

$$\widehat{\theta} = \operatorname*{argmin}_{\theta \in \Theta_0} \ell_{\text{dev}}\big(A_\theta(S_{\text{train}})\big)$$

of the hyperparameter corresponding to the predictor $A_{\widehat{\theta}}(S_{\text{train}})$ with smallest error $\ell_{\text{dev}}\big(A_\theta(S_{\text{train}})\big)$ on the validation set. Using Chernoff-Hoeffding bounds and the union bound, we can observe that

$$\ell_{\mathcal{D}}\big(A_{\widehat{\theta}}(S_{\text{train}})\big) \le \ell_{\mathcal{D}}\big(A_{\theta^*}(S_{\text{train}})\big) + \sqrt{\frac{2}{|S_{\text{dev}}|} \ln \frac{2|\Theta_0|}{\delta}}$$

with probability at least $1-\delta$ over the random draw of S_{dev}. This is telling us how big S_{dev} must be compared to Θ_0. Although our theoretical arguments apply to θ^*, $\widehat{\theta}$ is often used in practice as an estimate of θ_{opt}, which amounts to using the predictor $A_{\widehat{\theta}}(S)$ instead of $A_{\widehat{\theta}}(S_{\text{train}})$ (note that S is larger than S_{train}).

Tuning parameters via nested cross-validation. What if we want to estimate the expected value of (1) with respect to the random draw of the training set of fixed size?

$$\mathbb{E}\Big[\min_{\theta \in \Theta_0} \ell_{\mathcal{D}}\big(A_\theta(S)\big)\Big] \tag{2}$$

In other words, we want to estimate the performance of A_θ on a typical training set of a given size when θ is chosen using the training set.

Given a dataset S, a cheap way of estimating (2) is to use the best CV-estimate over $\{A_\theta : \theta \in \Theta_0\}$,

$$\min_{\theta \in \Theta_0} \ell_S^{\text{cv}}(A_\theta)$$

This estimate tends to underestimate (2), although in practice the difference is typically small.

A better, though more computationally intensive estimate of (2) is computed through nested CV (Algorithm 1).

Note that in each run of internal cross-validation we optimize θ locally, on the training part $S^{(i)}$ of the external cross-validation fold. Hence, the nested cross-validation estimate is computed by

```
Data: Dataset $S$
Split $S$ into folds $S_1, \dots, S_K$
for $i = 1, \dots, K$ do
    Compute training part of $i$-th fold: $S_{-i} \equiv S \setminus S_i$
    Run CV on $S_{-i}$ for each $\theta \in \Theta_0$ and find $\theta_i = \operatorname*{argmin}_{\theta \in \Theta_0} \ell^{\mathrm{cv}}_{S_{-i}}(A_\theta)$
    Re-train $A_{\theta_i}$ on $S_{-i}$: $h_i = A_{\theta_i}\big(S_{-i}\big)$
    Compute error of $i$-th fold: $\varepsilon_i = \ell_{S_i}(h_i)$
end
Output: $(\varepsilon_1 + \cdots + \varepsilon_K)/K$
```

Algorithm 1: K-fold nested cross-validation

averaging the performance of predictors obtained with potentially different values of their hyper-parameters.

There are cheaper variants of nested cross-validation in which the inner cross-validation is replaced by a dev set estimate. In other words, instead of running CV on S_{-i}, we split S_{-i} (of size $(1 - 1/K)m$) in a validation set of size m/K and in a training set of size $(1 - 2/K)m$. Then, we find θ_i by running the same procedure used to estimate $\min_{\theta \in \Theta_0} \ell_{\mathcal{D}}\big(A_\theta(S_{-i})\big)$ and return $h_i = A_{\theta_i}(S_{-i})$ to the outer cross-validation loop.

Quantity to estimate	Estimator
$\ell_{\mathcal{D}}\big(A(S)\big)$	test error
$\min_\theta \ell_{\mathcal{D}}\big(A_\theta(S)\big)$	development set
$\mathbb{E}\big[\ell_{\mathcal{D}}(A(S))\big]$	cross validation
$\mathbb{E}\Big[\min_\theta \ell_{\mathcal{D}}\big(A_\theta(S)\big)\Big]$	nested cross-validation

The above table collects the different risk estimators. In the first two cases, the training set S is given. In the remaining two cases, the expectation is computed over the random draw of S of fixed size.