Machine Learning — Statistical Methods for Machine Learning

Hyperparameter tuning and risk estimates

Instructor: *Nicolò Cesa-Bianchi* version of March 26, 2021

In practice, learning algorithms are often specified up to one or more hyperparameters. These are special parameters (like k in k-NN or the learning rate, the number of epochs, and the batch size 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. Fix a learning problem $(\mathcal{D}, \ell)$ and let $A_\theta(S)$ be the predictor output when A_θ is run on the training set S. Let $\ell_{\mathcal{D}}\big(A_\theta(S)\big)$ be the risk of the predictor $A_\theta(S)$, and let $\mathbb{E}\big[\ell_{\mathcal{D}}(A_\theta)\big]$ be the expected risk of $A_\theta(S)$ where the expectation is with respect to the random draw of the training set S of a given fixed size. Intuitively, $\mathbb{E}\big[\ell_{\mathcal{D}}(A_\theta)\big]$ measures the performance of A_θ on a typical training set of that size.

Evaluating a learning algorithm using external cross-validation. Assume for now the hyperparameter θ is fixed and focus on the problem of estimating $\mathbb{E}\big[\ell_{\mathcal{D}}(A)\big]$. To do so 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*) $D_1, \dots, D_K$ 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 $S^{(k)} \equiv S \setminus D_k$. We call D_k the **testing part** of the k-th fold while $S^{(k)}$ 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

$$D_1 = \big\{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_5, y_5)\big\} \qquad D_2 = \big\{(\boldsymbol{x}_6, y_6), \dots, (\boldsymbol{x}_{10}, y_{10})\big\}$$
$$D_3 = \big\{(\boldsymbol{x}_{11}, y_{11}), \dots, (\boldsymbol{x}_{15}, y_{15})\big\} \quad D_4 = \big\{(\boldsymbol{x}_{16}, y_{16}), \dots, (\boldsymbol{x}_{20}, y_{20})\big\}$$

then $S^{(2)} = \big\{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_5, y_5), (\boldsymbol{x}_{11}, y_{11}), \dots, (\boldsymbol{x}_{20}, y_{20})\big\}$.

The *K-fold CV estimate* of $\mathbb{E}\big[\ell_{\mathcal{D}}(A)\big]$ on S, denoted by $\widehat{\ell}^{\,\mathrm{cv}}_S(A)$, is then computed as follows: we run A on each training part $S^{(k)}$ of the folds $k = 1, \dots, K$ and obtain the predictors $h_1 = A\big(S^{(1)}\big), \dots, h_K = A\big(S^{(K)}\big)$. We then compute the (rescaled) errors on the testing part of each fold,

$$\widehat{\ell}_{D_k}(h_k) = \frac{K}{m} \sum_{(\boldsymbol{x}, y) \in D_k} \ell\big(y, h_k(\boldsymbol{x})\big)$$

Finally, we compute the CV estimate by averaging these errors

$$\widehat{\ell}^{\,\mathrm{cv}}_S(A) = \frac{1}{K} \sum_{k=1}^{K} \widehat{\ell}_{D_k}\big(h_k\big)$$

Tuning hyperparameters on a given training set. 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 $[0,1]$). If S is our training set, then we want to find $\theta^* \in \Theta_0$ such that

$$\ell_{\mathcal{D}}\big(A_{\theta^*}(S)\big) = \min_{\theta \in \Theta_0} \ell_{\mathcal{D}}\big(A_\theta(S)\big) \tag{1}$$

The estimate is computed by splitting 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. 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. In order to obtain the final predictor, the learning algorithm is run once more on the original training set S using the value of the hyperparameter corresponding to the predictor with smallest error on the validation set.

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 S (of fixed size)?

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

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

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

$$\min_{\theta \in \Theta_0} \widehat{\ell}^{\,\text{cv}}_S(A_\theta)$$

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

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

Data: Training set S
Split S into folds $D_1, \dots, D_K$
for $k = 1, \dots, K$ **do**
  Compute training part of k-th fold: $S^{(k)} \equiv S \setminus D_k$
  Run CV on $S^{(k)}$ for each $\theta \in \Theta_0$ and find best one: $\theta_k = \operatorname*{argmin}_{\theta \in \Theta_0} \widehat{\ell}^{\,\text{cv}}_{S^{(k)}}(A_\theta)$
  Re-train A_{θ_k} on $S^{(k)}$: $h_k = A_{\theta_k}\big(S^{(k)}\big)$
  Compute error of k-th fold: $\varepsilon_k = \widehat{\ell}_{D_k}(h_k)$
end
Output: $(\varepsilon_1 + \dots + \varepsilon_K)/K$

Algorithm 1: K-fold nested cross-validation

Note that in each run of internal cross-validation we optimize θ locally, on the training part $S^{(k)}$ of the external cross-validation fold. Hence, the nested cross-validation estimate is computed by averaging the performance of predictors obtained with potentially different values of their hyperparameters.