Machine Learning — Statistical Methods for Machine Learning

Cross-validation

Instructor: *Nicolò Cesa-Bianchi* version of March 23, 2019

In practice, learning algorithms are often specified up to one or more parameters, and setting the value of these parameters in the wrong way can lead to underfitting or overfitting. In order to choose the parameters correctly, we cannot use the set set. If we used the test set to help us choose the correct value of the parameter of some learning algorithm, then the final predictor obtained in this way would depend on both the training and the test set. As using the test error as estimate for the risk of a predictor relies on the independence of the test set from the data used to select the predictor, we would not be able to estimate the risk of this predictor.

A learning algorithm with a free parameter —like k in k-NN or the number of nodes of a tree predictor— is not really an algorithm, but rather a family of algorithms, one for each possible value of the parameter. Let $\{A_k : k \in \mathcal{K}\}$ be such a family of learning algorithms, where $\mathcal{K}$ is the set of all possible values for the parameter. Fix a learning problem $(\mathcal{D}, \ell)$ and let $A_k(S)$ be the predictor output when A_k is run on the training set S. Let $\ell_{\mathcal{D}}\big(A_k(S)\big)$ be the risk of the predictor $A_k(S)$, and let $\mathbb{E}\big[\ell_{\mathcal{D}}(A_k)\big]$ be the expected risk of $A_k(S)$ where the expectation is with respect to the random draw of the training set S of given fixed size.

Tuning parameters on given a training/dev/test split. Sometimes the dataset is distributed in three chunks: training set, dev set, and test set. The developmment (or validation) set is made available to help tuning the parameters of a learning algorithm. The learning algorithm is repeatedly run on the training set with different values of the parameters. The resulting predictors are tested on the dev set to determine the one with smallest validation error. In order to produce the final predictor, the learning algorithm is run once more on the union of training and dev sets using the best value of the parameter (corresponding to the predictor with smallest validation error). The resulting predictor can then be evaluated on the test set to estimate its risk.

Evaluating a learning algorithm using external cross-validation. External cross-validation is a technique for estimating $\mathbb{E}\big[\ell_{\mathcal{D}}(A)\big]$ given a learning algorithm A and a dataset

$$S \equiv \big\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\big\}$$

(we use set-theoretic notation for convenience even though S may contain repeated examples). In order to perform K-fold cross-validation, we partition S in K subsets (also known as *folds*) $D_1, \ldots, D_K$ of size m/K each (assume for simplicity that K divides m). The extreme case $K = m$ provides a risk 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 $S = \big\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_{20}, y_{20})\big\}$ in $K = 4$ subsets

$$\begin{array}{ll} D_1 = \big\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_5, y_5)\big\} & D_2 = \big\{(\boldsymbol{x}_6, y_6), \ldots, (\boldsymbol{x}_{10}, y_{10})\big\} \\ D_3 = \big\{(\boldsymbol{x}_{11}, y_{11}), \ldots, (\boldsymbol{x}_{15}, y_{15})\big\} & D_4 = \big\{(\boldsymbol{x}_{16}, y_{16}), \ldots, (\boldsymbol{x}_{20}, y_{20})\big\} \end{array}$$

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

In order to estimate $\mathbb{E}\big[\ell_{\mathcal{D}}(A)\big]$, we run A on each $S^{(k)}$, for $k \in \{1, \ldots, K\}$, and compute the *K-fold cross-validation estimate*

$$\frac{1}{K}\sum_{k=1}^{K} \widehat{\ell}_{D_k}\big(h_k\big) \qquad \text{where} \quad h_k = A\big(S^{(k)}\big) \quad \text{and} \quad \widehat{\ell}_{D_k}(h_k) = \frac{K}{m}\sum_{(\boldsymbol{x},y)\in D_k} \ell\big(y, h_k(\boldsymbol{x})\big)$$

denotes the average loss of h_k on D_k.

Tuning parameters via nested cross-validation. We now turn to the problem of estimating

$$\mathbb{E}\Big[\min_{k\in\mathcal{K}} \ell_{\mathcal{D}}\big(A_k\big)\Big] \tag{1}$$

This is the risk, averaged over all training sets S of given fixed size, of the predictor generated by $A_k(S)$ when k is optimized on the training data S. The quantity (1) is typically estimated through *nested cross-validation*. This works as follows: First, an "external" cross-validation is run on the given dataset. On each fold, we run an "internal" cross-validation for each algorithm in $\{A_k : k \in \mathcal{K}\}$. The A_k with best internal cross-validation estimate is retrained on the entire training part of the fold; the resulting predictor is then evaluated on the testing part of the fold. The value of the external cross-validation estimate is finally used as nested cross validation estimate.

Note that in each run of internal cross-validation we optimize the parameter k locally on the training part of the external cross-validation fold. Hence, the nested cross-validation score is computed by averaging the performance of predictors obtained using the values of the parameter found in each fold, which are potentially different.

A simpler (and less statistically sound) way of tuning parameters is by computing the standard (single-level) cross-validation estimate of each algorithm in $\{A_k : k \in \mathcal{K}\}$, and then using the lowest of such estimates for estimating (1). This procedure is computationally simpler than nested cross-validation, but the resulting estimate tends to underestimate (1). In practice, however, the difference is typically small.