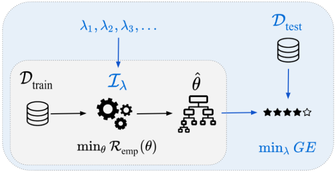
Introduction to Machine Learning

Hyperparameter Tuning - Problem Definition



Learning goals

- Understand tuning as a bi-level optimization problem
- Know the components of a tuning problem
- Be able to explain what makes tuning a complex problem

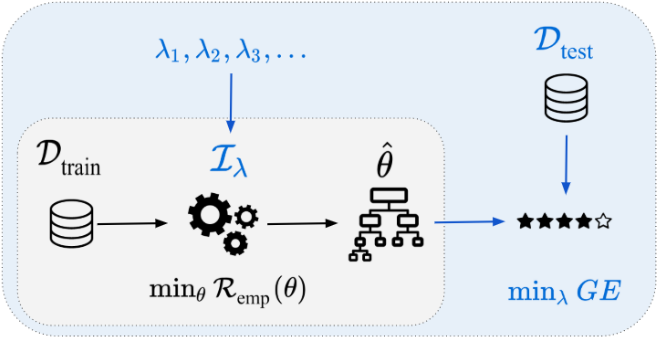
TUNING

Recall: **Hyperparameters $\boldsymbol{\lambda}$** are parameters that are *inputs* to the training problem in which a learner $\mathcal{I}$ minimizes the empirical risk on a training data set in order to find optimal **model parameters $\boldsymbol{\theta}$** which define the fitted model $\hat{f}$.

(Hyperparameter) Tuning is the process of finding good model hyperparameters $\boldsymbol{\lambda}$.

TUNING: A BI-LEVEL OPTIMIZATION PROBLEM

We face a **bi-level** optimization problem: The well-known risk minimization problem to find $\hat{f}$ is **nested** within the outer hyperparameter optimization (also called second-level problem):



TUNING: A BI-LEVEL OPTIMIZATION PROBLEM

- For a learning algorithm $\mathcal{I}$ (also inducer) with d hyperparameters, the hyperparameter **configuration space** is:

$$\boldsymbol{\Lambda} = \Lambda_1 \times \Lambda_2 \times \ldots \times \Lambda_d,$$

  where Λ_i is the domain of the i-th hyperparameter.

- The domains can be continuous, discrete or categorical.
- For practical reasons, the domain of a continuous or integer-valued hyperparameter is typically bounded.
- A vector in this configuration space is denoted as $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$.
- A learning algorithm $\mathcal{I}$ takes a (training) dataset $\mathcal{D} \in \mathbb{D}$ and a hyperparameter configuration $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ and returns a trained model (through risk minimization)

$$\begin{aligned} \mathcal{I} : \left(\bigcup_{n \in \mathbb{N}} (\mathcal{X} \times \mathcal{Y})^n \right) \times \boldsymbol{\Lambda} \quad & \rightarrow \quad \mathcal{H} \\ (\mathcal{D}, \boldsymbol{\lambda}) \quad & \mapsto \quad \mathcal{I}(\mathcal{D}, \boldsymbol{\lambda}) = \hat{f}_{\mathcal{D}, \boldsymbol{\lambda}} \end{aligned}$$

TUNING: A BI-LEVEL OPTIMIZATION PROBLEM

We formally state the nested hyperparameter tuning problem as:

$$\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} \widehat{GE}_{\mathcal{D}_{\text{test}}} \left(\mathcal{I}(\mathcal{D}_{\text{train}}, \boldsymbol{\lambda})\right)$$

- The learner $\mathcal{I}(\mathcal{D}_{\text{train}}, \boldsymbol{\lambda})$ takes a training data set as well as hyperparameter settings $\boldsymbol{\lambda}$ (e.g., the maximal depth of a classification tree) as an input.
- $\mathcal{I}(\mathcal{D}_{\text{train}}, \boldsymbol{\lambda})$ performs empirical risk minimization on the training data and returns the optimal model $\hat{f}$ for the given hyperparameters.
- Note that for the estimation of the generalization error, more sophisticated resampling strategies like cross-validation can be used.

TUNING: A BI-LEVEL OPTIMIZATION PROBLEM

The components of a tuning problem are:

- The data set
- The learner (possibly: several competing learners?) that is tuned
- The learner's hyperparameters and their respective regions-of-interest over which we optimize
- The performance measure, as determined by the application. Not necessarily identical to the loss function that defines the risk minimization problem for the learner!
- A (resampling) procedure for estimating the predictive performance

WHY IS TUNING SO HARD?

- Tuning is derivative-free (“black box problem”): It is usually impossible to compute derivatives of the objective (i.e., the resampled performance measure) that we optimize with regard to the HPs. All we can do is evaluate the performance for a given hyperparameter configuration.
- Every evaluation requires one or multiple train and predict steps of the learner. I.e., every evaluation is very **expensive**.
- Even worse: the answer we get from that evaluation is **not exact, but stochastic** in most settings, as we use resampling.
- Categorical and dependent hyperparameters aggravate our difficulties: the space of hyperparameters we optimize over has a non-metric, complicated structure.