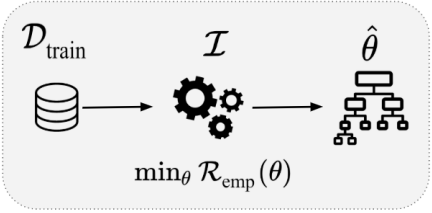
# AutoML: Hyperparameter Optimization

## Overview and Introduction

Bernd Bischl  Frank Hutter  Lars Kotthoff
Marius Lindauer  Joaquin Vanschoren

# Motivating Example I

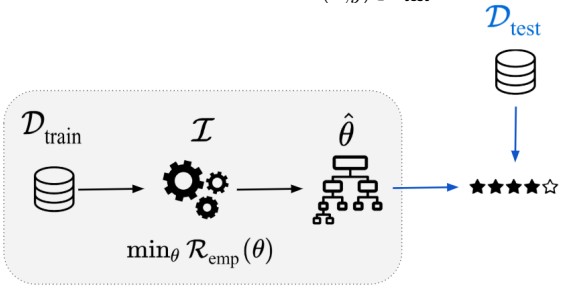
- Given a dataset, we want to train a classification tree.
- We feel that a maximum tree depth of $4$ has worked out well for us previously, so we decide to set this hyperparameter to $4$.
- The learner ("inducer") $\mathcal{I}$ takes the input data, internally performs **empirical risk minimization**, and returns a fitted tree model $\hat{f}(\mathbf{x}) = f(\mathbf{x}, \hat{\theta})$ of at most depth $\boldsymbol{\lambda} = 4$ that minimizes the empirical risk.



# Motivating Example II

- We are **actually** interested in the **generalization performance** $GE\left(\hat{f}\right)$ of the estimated model on new, previously unseen data.
- We estimate the generalization performance by evaluating the model $\hat{f}$ on a test set $\mathcal{D}_{\text{test}}$:

$$\widehat{GE}_{\mathcal{D}_{\text{test}}}\left(\hat{f}\right) = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(\mathbf{x},y)\in\mathcal{D}_{\text{test}}} L\left(y, \hat{f}(\mathbf{x})\right)$$



# Motivating Example III

- But many ML algorithms are sensitive w.r.t. a good setting of their hyperparameters, and generalization performance might be bad, if we have chosen a suboptimal configuration:
  - The data may be too complex to be modeled by a tree of depth $4$
  - The data may be much simpler than we thought, and a tree of depth $4$ overfits

$\Longrightarrow$ algorithmically try out different values for the tree depth. For each maximal depth $\boldsymbol{\lambda}$, we have to train the model **to completion** and evaluate its performance on the test set.

- We choose the tree depth $\boldsymbol{\lambda}$ that is **optimal** w.r.t. the generalization error of the model.

# Model Parameters vs. Hyperparameters I

It is critical to understand the difference between model parameters and hyperparameters.

**Model parameters** are optimized during training, typically via loss minimization. They are an **output** of the training. Examples:

- The splits and terminal node constants of a tree learner
- Coefficients $\theta$ of a linear model $f(\mathbf{x}) = \theta^\top \mathbf{x}$

# Model Parameters vs. Hyperparameters II

In contrast, **hyperparameters** (HPs) are not decided during training. They must be specified before the training, they are an **input** of the training. Hyperparameters often control the complexity of a model, i.e., how flexible the model is. But they can in principle influence any structural property of a model or computational part of the training process.

Examples:

- Tree: The maximum depth of a tree
- $k$ Nearest Neighbours: Number of neighbours $k$ and distance measure
- Linear regression: Number and maximal order of interactions

# Types of hyperparameters I

We summarize all hyperparameters we want to tune over in a vector $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ of (possibly) mixed type. HPs can have different types:

- Real-valued parameters, e.g.:
  - Minimal error improvement in a tree to accept a split
  - Bandwidths of the kernel density estimates for Naive Bayes
- Integer parameters, e.g.:
  - Neighbourhood size $k$ for $k$-NN
  - Minimum number of samples for a split in a random forest
- Categorical parameters, e.g.:
  - Which split criterion for classification trees?
  - Which distance measure for $k$-NN?

Hyperparameters are often **hierarchically dependent** on each other, e.g., *if* we use a kernel-density estimate for Naive Bayes, what is its width?

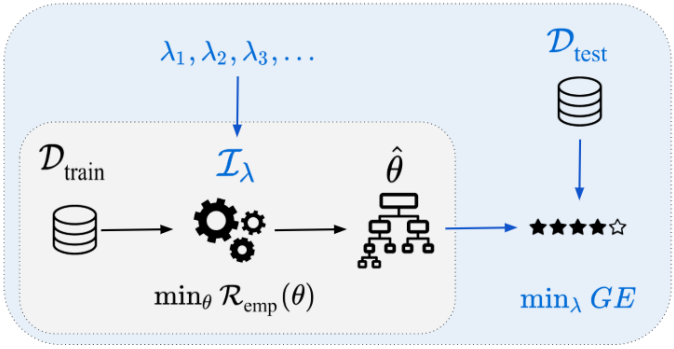
# Tuning

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** $\theta$ 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 I

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 II

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

$$\mathbf{\Lambda} = \mathbf{\Lambda}_1 \times \mathbf{\Lambda}_2 \times \ldots \times \mathbf{\Lambda}_d$$

  where $\mathbf{\Lambda}_i$ 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 $\boldsymbol{\lambda} \in \mathbf{\Lambda}$.
- A learning algorithm $\mathcal{I}$ takes a (training) dataset $\mathcal{D}$ and a hyperparameter configuration $\boldsymbol{\lambda} \in \mathbf{\Lambda}$ and returns a trained model (through risk minimization).

$$\begin{aligned} \mathcal{I} : (\mathcal{X} \times \mathcal{Y})^n \times \mathbf{\Lambda} &\rightarrow \mathcal{H} \\ (\mathcal{D}, \mathbf{\Lambda}) &\mapsto \mathcal{I}(\mathcal{D}, \mathbf{\Lambda}) = \hat{f}_{\mathcal{D},\mathbf{\Lambda}} \end{aligned}$$

# Tuning: A bi-level optimization problem III

We formally state the nested hyperparameter tuning problem as:

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

- The learner $\mathcal{I}(\mathcal{D}_{\text{train}}, \boldsymbol{\lambda})$ takes a training dataset 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 IV

The components of a tuning problem are:

- The dataset
- 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 according to the performance measure.

# 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 (and often stochastic learners).
- Categorical and dependent hyperparameters aggravate our difficulties: the space of hyperparameters we optimize over has a non-metric, complicated structure.