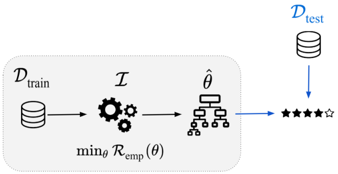
Einführung in das statistische Lernen

Hyperparameter Tuning - Introduction



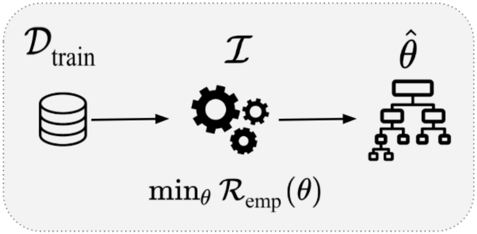
$\mathcal{D}_{\text{test}}$
$\mathcal{D}_{\text{train}}$
$\mathcal{I}$
$\hat{\theta}$
$\min_\theta \mathcal{R}_{\text{emp}}(\theta)$

Learning goals

- Understand the difference between model parameters and hyperparameters
- Know different types of hyperparameters
- Be able to explain the goal of hyperparameter tuning

MOTIVATING EXAMPLE

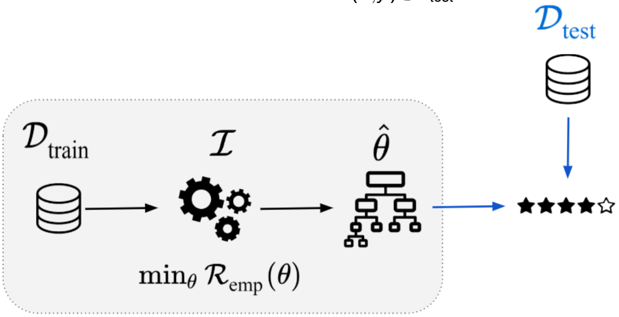
- Given a data set, we want to train a classification tree.
- We feel that a maximum tree depth of 4 has worked out well for us previously, so we decide to set this hyperparameter to 4.
- The learner ("inducer") $\mathcal{I}$ takes the input data, internally performs **empirical risk minimization**,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


$\mathcal{D}_{\text{train}}$
$\mathcal{I}$
$\hat{\theta}$
$\min_\theta \mathcal{R}_{\text{emp}}(\theta)$

MOTIVATING EXAMPLE

- 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 $\mathcal{D}_{\text{test}}$:

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



$\mathcal{D}_{\text{test}}$
$\mathcal{D}_{\text{train}}$
$\mathcal{I}$
$\hat{\theta}$
★★★★☆
$\min_\theta \mathcal{R}_{\text{emp}}(\theta)$

MOTIVATING EXAMPLE

- 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 maximum depth λ, we have to train the model **to completion** and evaluate its performance on the test set.

- We choose the tree depth λ that is **optimal** w.r.t. the generalization error of the model.

MODEL PARAMETERS VS. HYPERPARAMETERS

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 $\boldsymbol{\theta}$ of a linear model $f(\mathbf{x}) = \boldsymbol{\theta}^T\mathbf{x}$

MODEL PARAMETERS VS. HYPERPARAMETERS

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:

- The maximum depth of a tree
- k and which distance measure to use for k-NN
- The number and maximal order of interactions to be included in a linear regression model

TYPES OF HYPERPARAMETERS

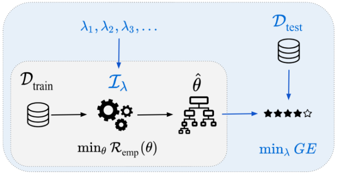
We summarize all hyperparameters we want to tune over in a vector $\boldsymbol{\lambda} \in \Lambda$ of (possibly) mixed type. HPs can have different types:

- 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.:
  - Neighborhood size k for k-NN
  - *mtry* 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?

Einführung in das statistische Lernen

Hyperparameter Tuning - Problem Definition


$\lambda_1, \lambda_2, \lambda_3, \dots$
$\mathcal{D}_{\text{test}}$
$\mathcal{D}_{\text{train}}$
$\mathcal{I}_\lambda$
$\hat{\theta}$
$\min_\theta \mathcal{R}_{\text{emp}}(\theta)$
$\min_\lambda GE$

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

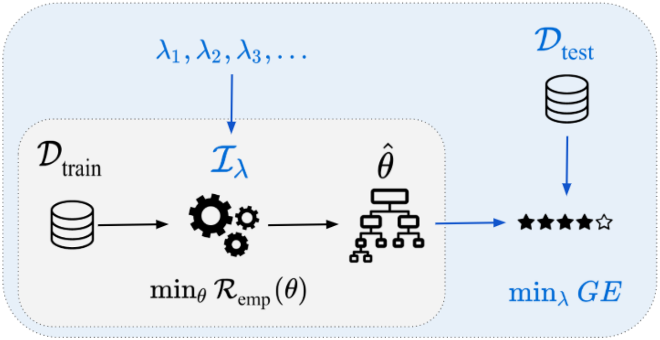
TUNING

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):


$\lambda_1, \lambda_2, \lambda_3, \ldots$
$\mathcal{D}_{\text{test}}$
$\mathcal{D}_{\text{train}}$
$\mathcal{I}_\lambda$
$\hat{\theta}$
$\min_\theta \mathcal{R}_{\text{emp}}(\theta)$
$\min_\lambda GE$

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)

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

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:

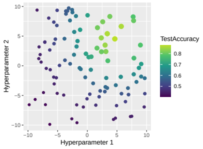
- The data set
- The learner (possibly: several competing learners?) that is tuned
- The learner's hyperparameters and their respective regions-of-interest over which we optimize
- The performance measure, as determined by the application. Not necessarily identical to the loss function that defines the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

Einführung in das statistische Lernen

Hyperparameter Tuning - Basic Techniques

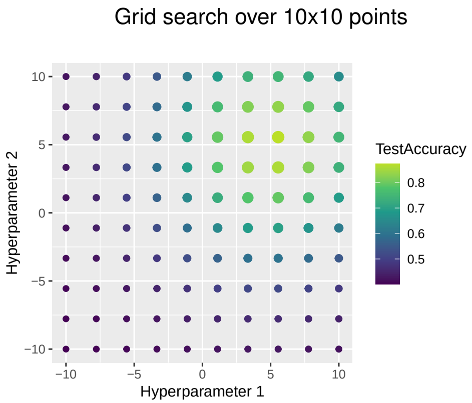


Learning goals

- Understand the idea of grid search
- Understand the idea of random search
- Be able to discuss advantages and disadvantages of the two methods

GRID SEARCH

- Simple technique which is still quite popular, tries all HP combinations on a multi-dimensional discretized grid
- For each hyperparameter a finite set of candidates is predefined
- Then, we simply search all possible combinations in arbitrary order



GRID SEARCH

Advantages

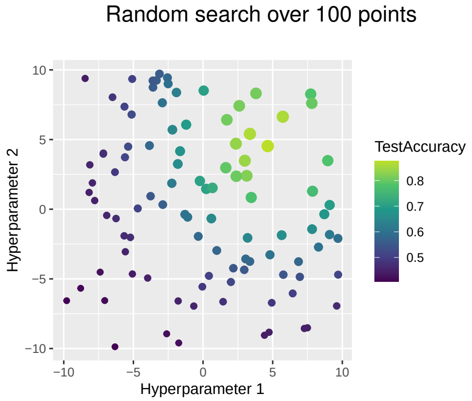
- Very easy to implement
- All parameter types possible
- Parallelizing computation is trivial

Disadvantages

- Scales badly: combinatorial explosion
- Inefficient: searches large irrelevant areas
- Arbitrary: which values / discretization?

RANDOM SEARCH

- Small variation of grid search
- Uniformly sample from the region-of-interest



RANDOM SEARCH

Advantages

- Like grid search: very easy to implement, all parameter types possible, trivial parallelization
- Anytime algorithm: can stop the search whenever our budget for computation is exhausted, or continue until we reach our performance goal.
- No discretization: each individual parameter is tried with a different value every time

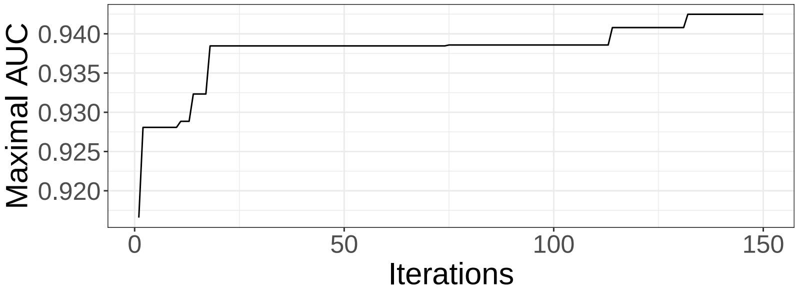
Disadvantages

- Inefficient: many evaluations in areas with low likelihood for improvement
- Scales badly: high-dimensional hyperparameter spaces need *lots* of samples to cover.

TUNING EXAMPLE

Tuning random forest with random search and 5CV on the `sonar` data set for AUC:

Hyperparameter	Type	Min	Max
`num.trees`	integer	3	500
`mtry`	integer	5	50
`min.node.size`	integer	10	100


Maximal AUC
Iterations