LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

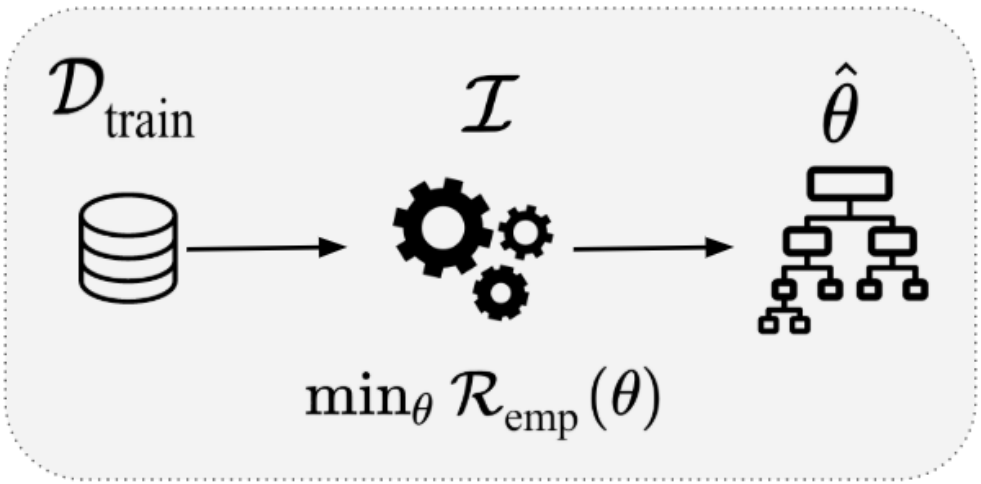
Introduction to Machine Learning

Hyperparameter Tuning

Department of Statistics – LMU Munich

MOTIVATING EXAMPLE

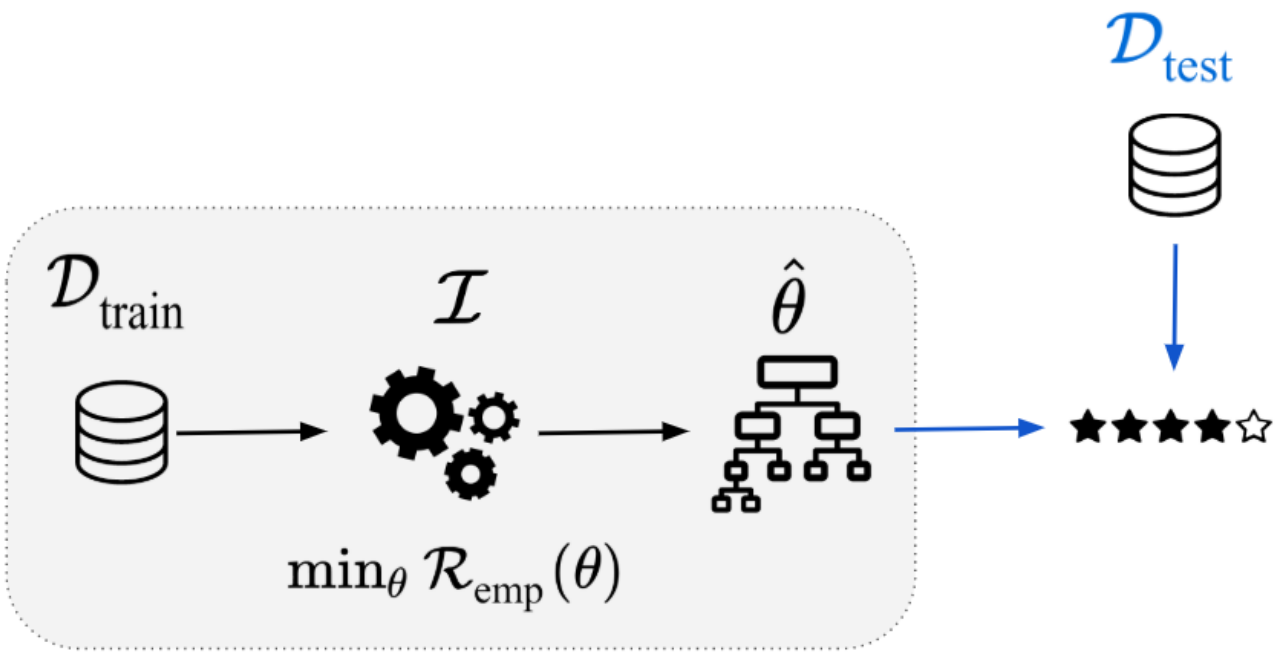
- Given a dataset, we decided to train a classification tree.
- We experienced that a maximum tree depth of 4 usually works well, so we decide to set this hyperparameter to 4.
- The learning algorithm (or inducer) $\mathcal{I}$ takes the input data, internally performs **empirical risk minimization**, and returns that tree (of depth 4) $\hat{f}(\mathbf{x}) = f(\mathbf{x}, \hat{\theta})$ that minimizes the empirical risk.



MOTIVATING EXAMPLE

- Recall, the model's **generalization performance** $GE(\hat{f})$ is what we are **actually** interested in.
- We estimate the generalization performance by evaluating the model $\hat{f}$ on the 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

- For some reason the trained model does not show a good generalization performance.
- One of the most likely reasons is that hyperparameters are suboptimal:
  - 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
- We try out different values for the tree depth. For each value of λ, we have to train the model **to completion**, and evaluate its performance on the holdout set.
- We choose the tree depth λ that is **optimal** w.r.t. the generalization error of the model.

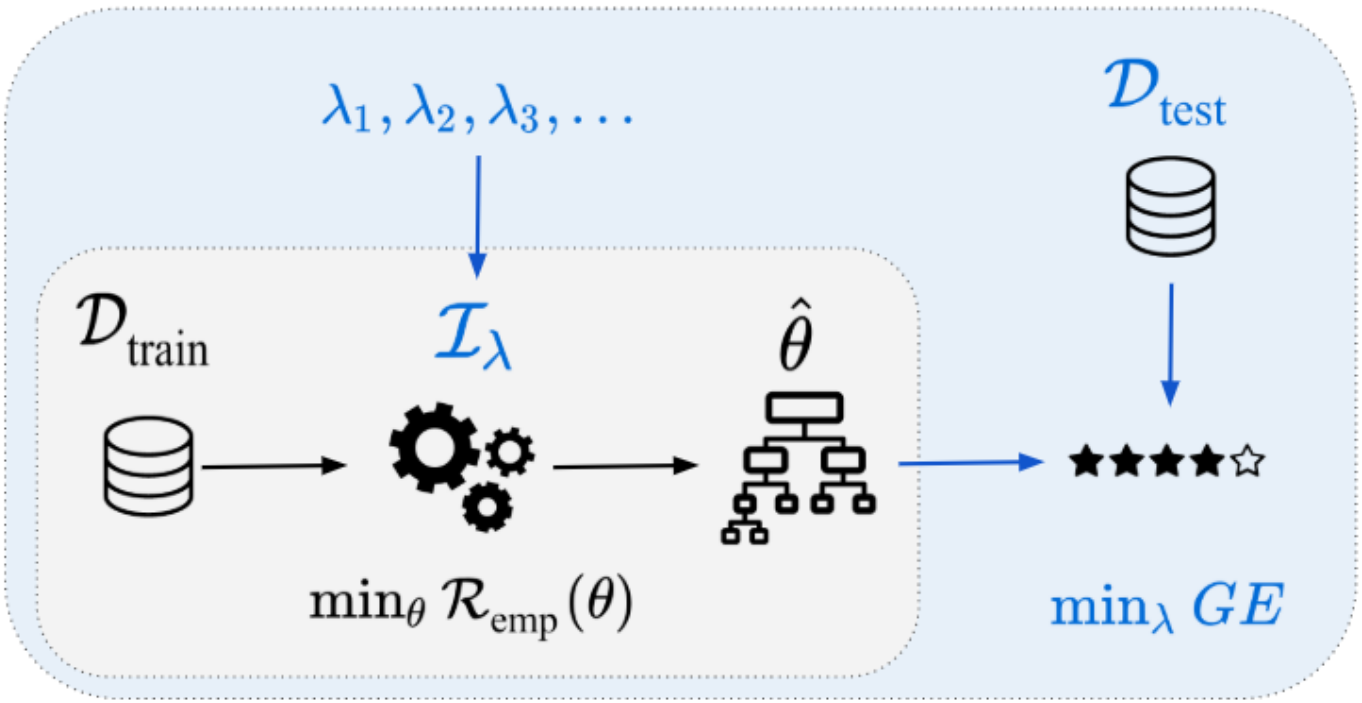
THE ROLE OF HYPERPARAMETERS

- Hyperparameters control the complexity of a model, i.e., how flexible the model is.
- If a model is too flexible so that it simply “memorizes” the training data, we will face the dreaded problem of overfitting.
- Hence, control of capacity, i.e., proper setting of hyperparameters can prevent overfitting the model on the training set.

TUNING: A BI-LEVEL OPTIMIZATION PROBLEM

The process of finding good model hyperparameters λ is called **(hyperparameter) tuning**.

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

We formally state the nested hyperparameter tuning problem as:

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

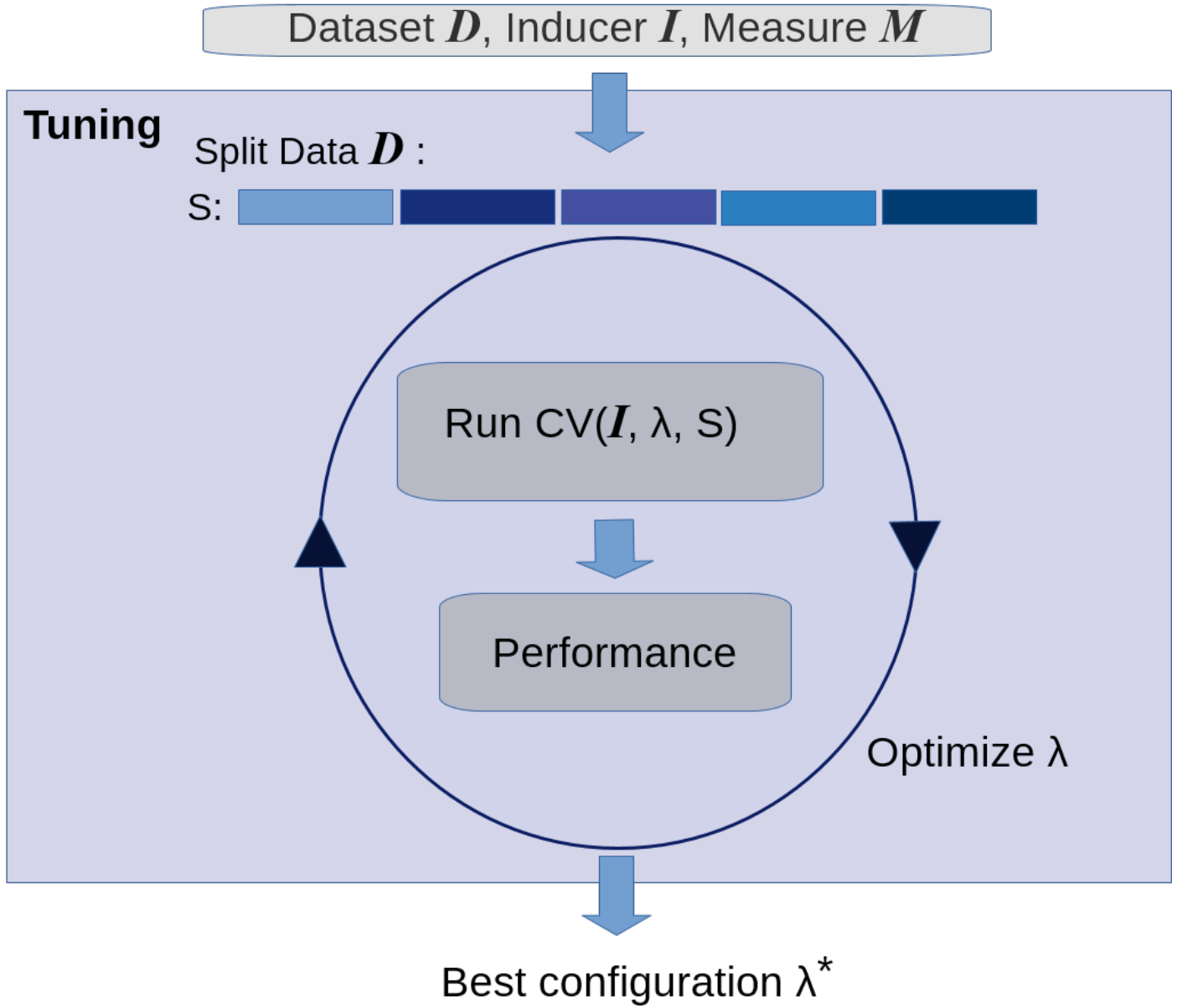
- The learning algorithm $\mathcal{I}(\mathcal{D}_{\text{train}}, \lambda)$ (also: inducer) takes a training dataset as well as hyperparameter settings λ (e.g. the desired depth of a classification tree) as an input.
- $\mathcal{I}(\mathcal{D}_{\text{train}}, \lambda)$ internally performs empirical risk minimization, and returns the optimal model $\hat{f}$.

TUNING: A BI-LEVEL OPTIMIZATION PROBLEM

The components of a tuning problem are:

- 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. Determined by the application. Not necessarily identical to the loss function that the learner tries to minimize. We could even be interested in multiple measures simultaneously, e.g., accuracy and sparseness of our model, TPR and PPV, etc.
- A (resampling) procedure for estimating the predictive performance: The generalization error can be estimated by holdout, but also by more advanced techniques like cross-validation.

TUNING: A BI-LEVEL OPTIMIZATION PROBLEM



MODEL PARAMETERS VS. HYPERPARAMETERS

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

Model parameters are optimized during training, by some form of loss minimization. They are an **output** of the training. Examples:

- Optimal splits of a tree learner
- Coefficients θ of a linear model $f(\mathbf{x}) = \theta^\top \mathbf{x}$

In contrast, **Hyperparameters** (HPs) are not decided during training. They must be specified before the training, they are an **input** of the training. Examples:

- The maximum depth of a tree
- k and which distance measure to use for kNN
- The complexity penalty to be used (e.g. L_1, L_2 penalization) and the complexity control parameter λ for regularization

TYPES OF HYPERPARAMETERS

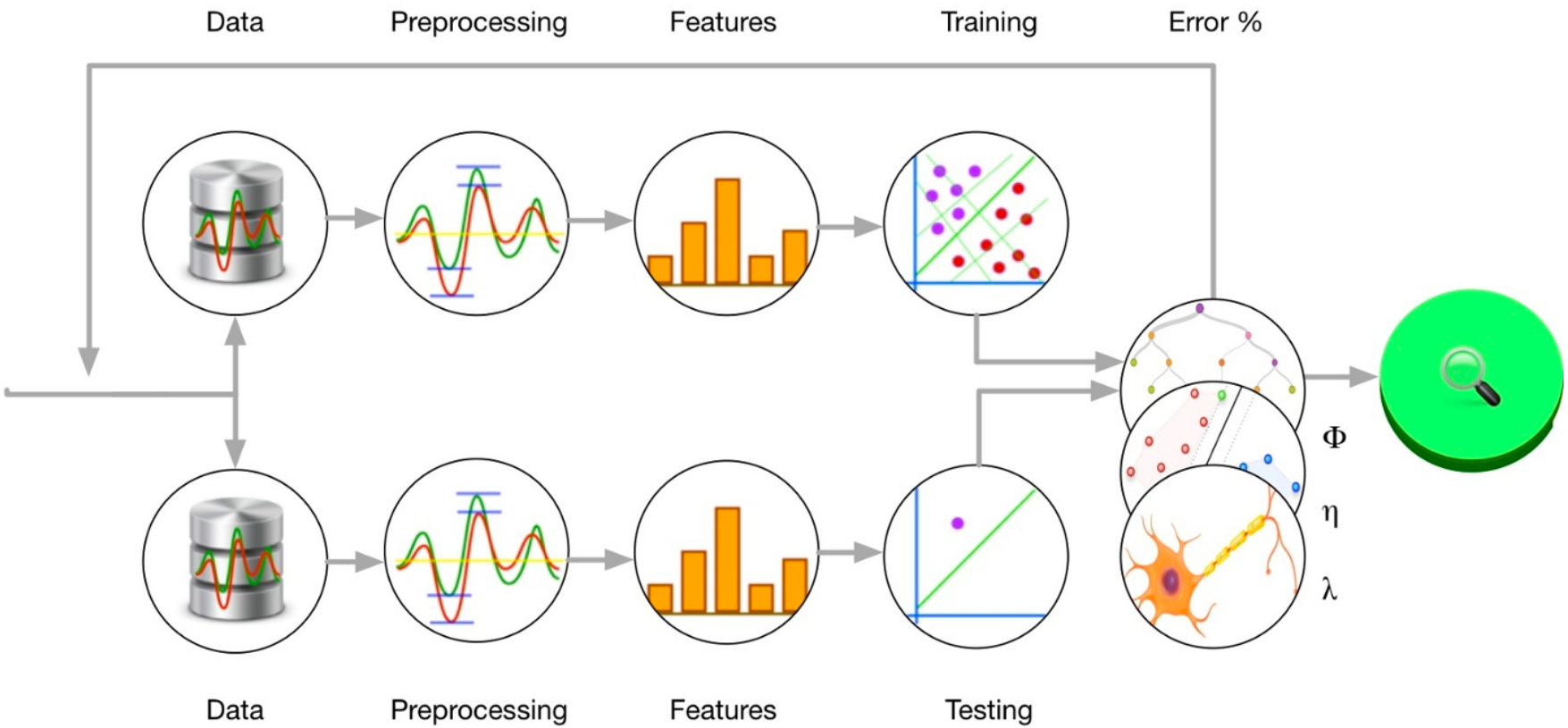
We summarize all hyperparameters we want to tune over in a vector $\lambda \in \Lambda$ of (possibly) mixed type. HPs can have different types:

- Numerical parameters (real valued / integers)
  - *mtry* in a random forest
  - Neighborhood size *k* for kNN
  - The complexity control parameter λ for regularization
- Categorical parameters:
  - Which split criterion for classification trees?
  - Which distance measure for kNN?
  - Which type of complexity penalization to use for reguarlized empirical risk minimization?
- Ordinal parameters:
  - $\{\texttt{low}, \texttt{medium}, \texttt{high}\}$
- Dependent parameters:
  - If we use the Gaussian kernel for the SVM, what is its width?

TUNING PIPELINES

- Many other factors like optimization control settings, distance functions, scaling, algorithmic variants in the fitting procedure can heavily influence model performance in non-trivial ways. It is extremely hard to guess the correct choices here.
- The choice of the learner itself (e.g. logistic regression, decision tree, random forest) can also be seen as a hyperparameter.
- In general, we might be interested in optimizing an entire ML "pipeline" (including preprocessing, feature construction, and other model-relevant operations).

TUNING PIPELINES



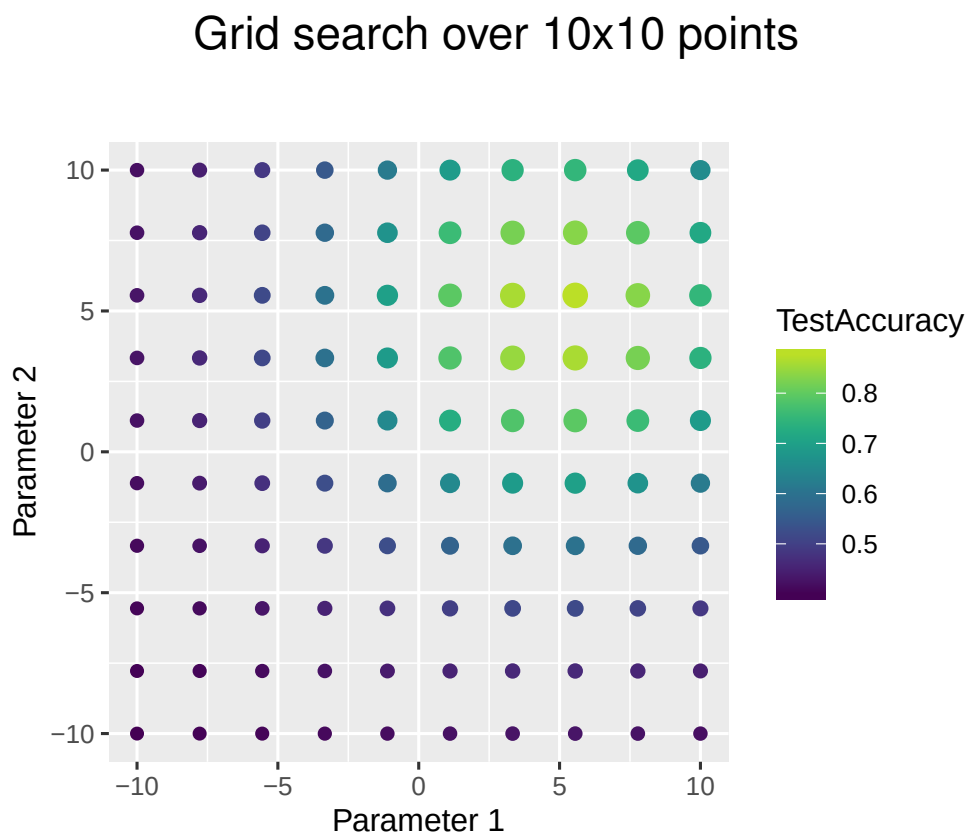
WHY IS TUNING SO HARD?

- 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.
- For large and difficult problems parallelizing the computation seems relevant, to evaluate multiple HP configurations in parallel or to speed up the resampling-based performance evaluation

Tuning Techniques

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

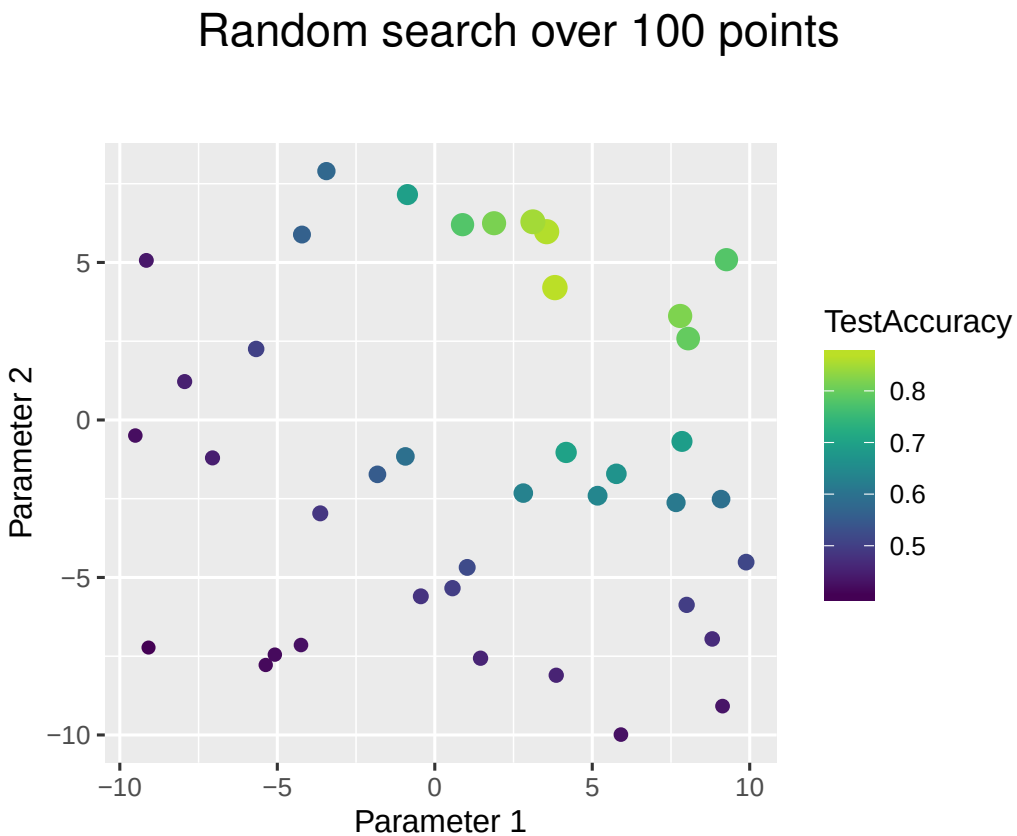
- Very easy to implement, therefore very popular
- All parameter types possible
- Parallelization is trivial

Disadvantages

- Combinatorial explosion, inefficient
- Searches large irrelevant areas
- Which values / discretization?

RANDOM SEARCH

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



RANDOM SEARCH

Advantages

- Very easy to implement, therefore very popular
- All parameter types possible
- Parallelization is trivial
- Anytime algorithm - we can always increase the budget when we are not satisfied
- Often better than grid search, as each individual parameter has been tried with m different values, when the search budget was m. Mitigates the problem of discretization

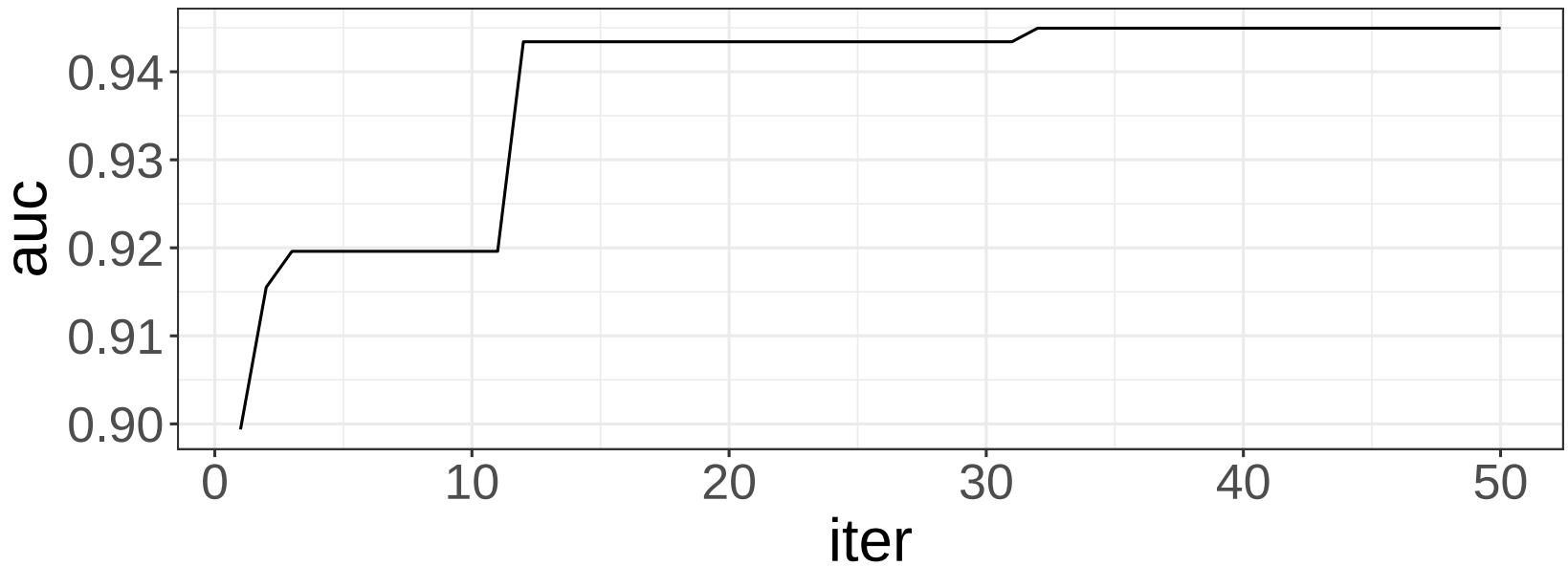
Disadvantages

- As for grid search, many evaluations in areas with low likelihood for improvement

TUNING EXAMPLE

Tuning gradient boosting with random search and 5CV on the spam data set for AUC.

Parameter	Type	Min	Max
n.trees	integer	3	500
shrinkage	numeric	0	1
interaction	integer	1	5
bag.fraction	numeric	0.2	0.9



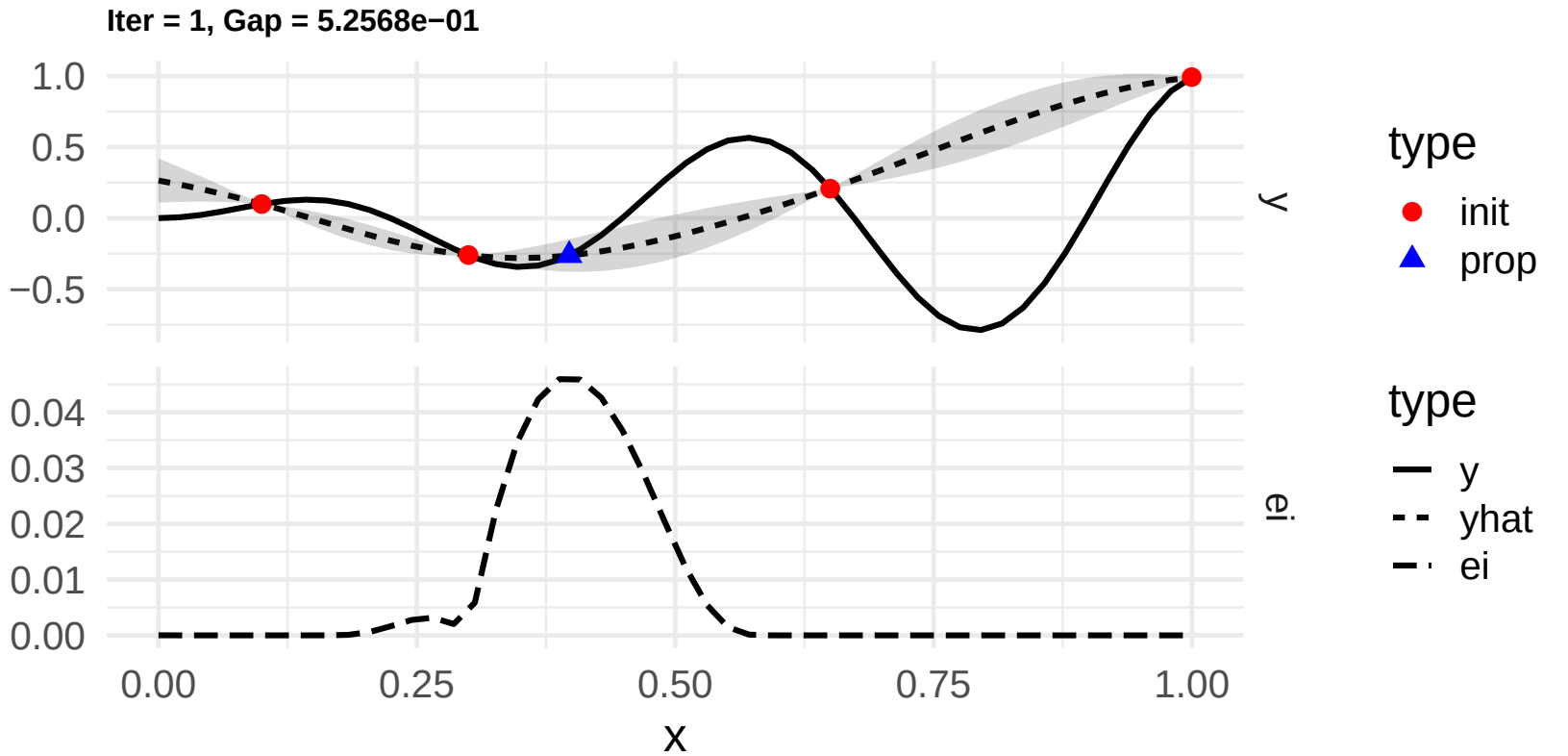
Advanced Tuning Techniques

MODEL-BASED OPTIMIZATION

Model-based optimization (MBO) is a sequential optimization procedure. We start with an initial design, i.e. a set of configurations λ_i where we have evaluated the corresponding resampling performance.

- Given the initial design, we build a **surrogate model** that models the relationship between model-hyperparameters and estimated generalization error. It serves as a cheap approximation of the expensive objective.
- Based on information provided by the surrogate model, a new configuration $\lambda^{(\text{new})}$ is proposed.
- The resampling performance of the learner with hyperparameter setting $\lambda^{(\text{new})}$ is evaluated and added to the set of design points.

MODEL-BASED OPTIMIZATION

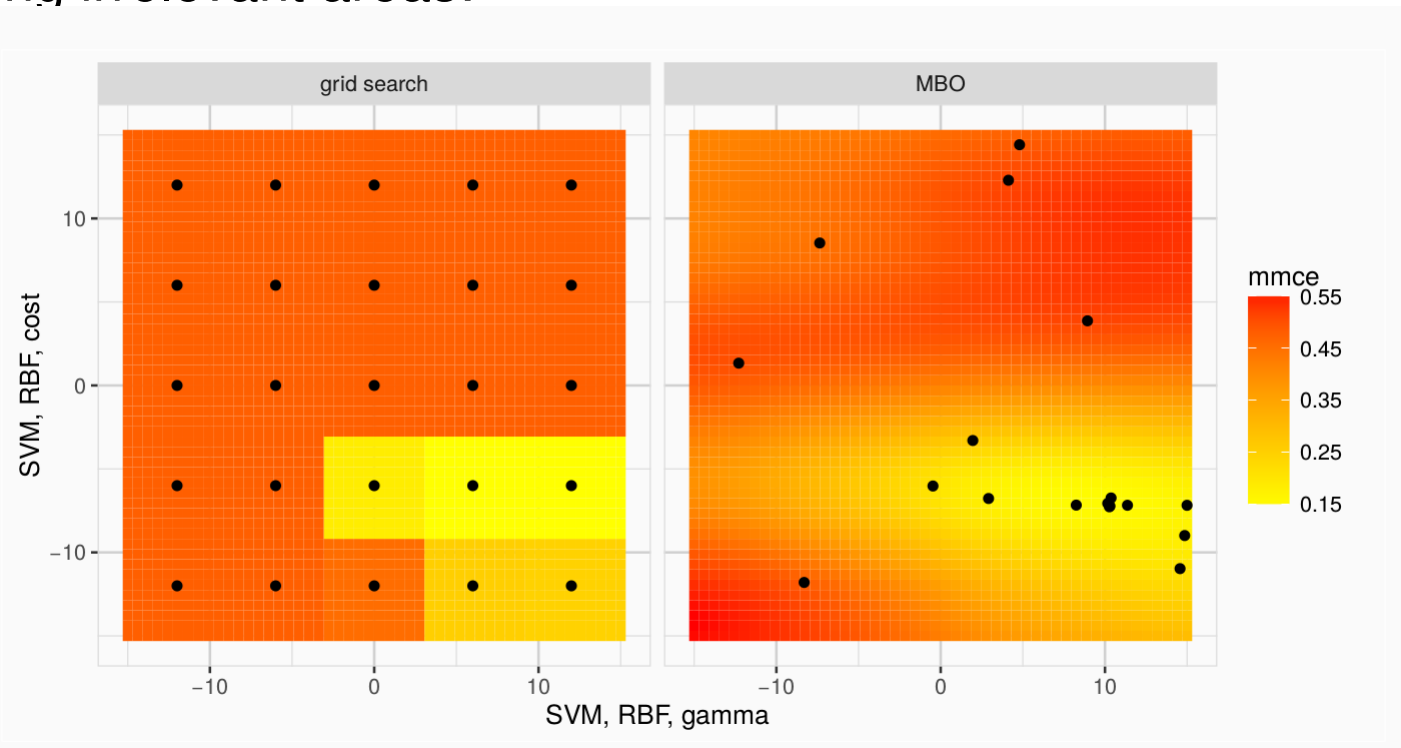


Upper plot: The surrogate model (black, dashed) models a the *unknown* relationship between input and output (black, solid) based on the initial design (red points). Lower plot: Mean and variance of the surrogate model is used to derive the expected improvement (EI) criterion. The point that maximizes the EI is proposed (blue point).

MODEL-BASED OPTIMIZATION

We iteratively perform those steps: (1) fit a surrogate model, (2) propose a new configuration, (3) evaluate the learners performance and update the design.

This guides us more to the “interesting” areas, and prevents us from searching irrelevant areas:

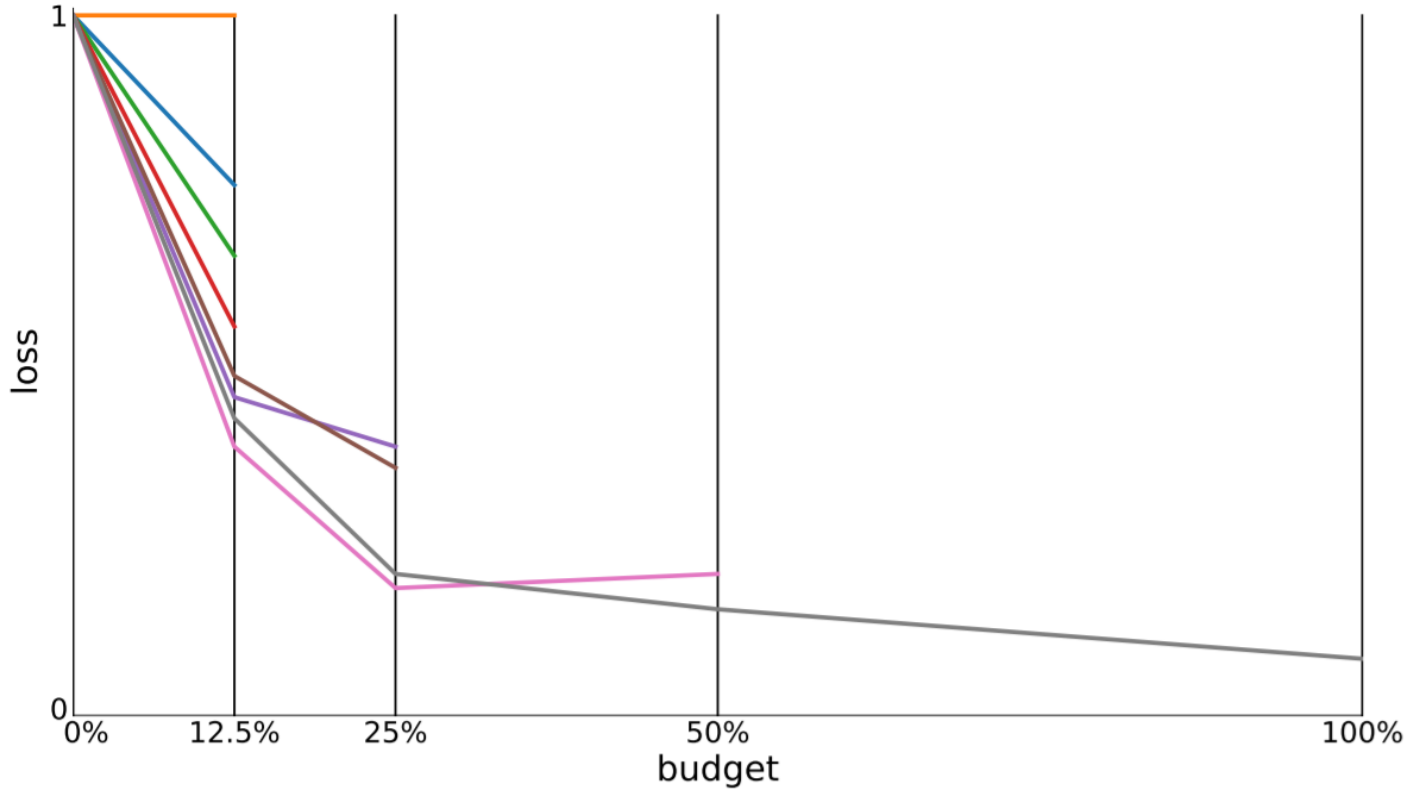


HYPERBAND

- It is extremely expensive to train complex models on large datasets
- For many configurations, it might be clear early on that further training is not likely to significantly improve the performance
- More importantly, the relative ordering of configurations (for a given dataset) can also become evident early on.
- **Idea:** “weed out” poor configurations early during training
- One approach is **successive halving**: Given an initial set of configurations, all trained for a small initial budget, repeat:
  - Remove the half that performed worst, double the budget
  - Continue until the new budget is exhausted
- Successful halving is performed several times with different trade-offs between the number of configurations considered and the budget that is spent on them.

HYPERBAND

Only the most promising configuration(s) are trained to completion:



HYPERBAND

Other advanced techniques besides model-based optimization and the hyperband algorithm are:

- Stochastic local search, e.g. simulated annealing
- Genetic algorithms / CMAES
- Iterated F-Racing
- . . .