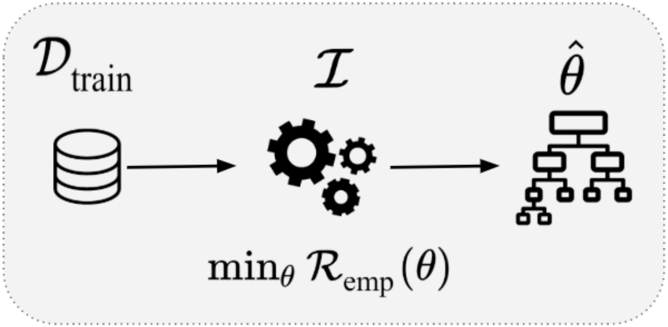
Introduction to Machine Learning

Hyperparameter Tuning

Department of Statistics – LMU Munich

MOTIVATING EXAMPLE

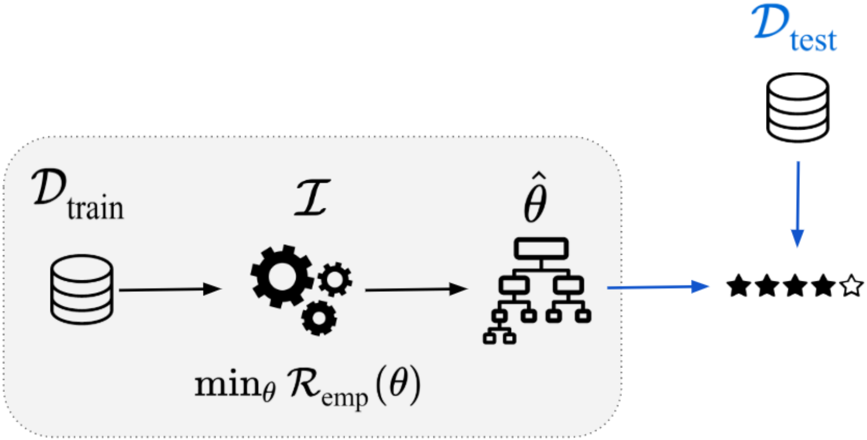
- Given a dataset, we decided to train a classification tree.
- We experienced that a maximum tree depth of 4 usually works well, so we decide to set this hyperparameter to 4.
- The learning algorithm (or inducer) $\mathcal{I}$ takes the input data, internally performs **empirical risk minimization**,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


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

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

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


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

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.

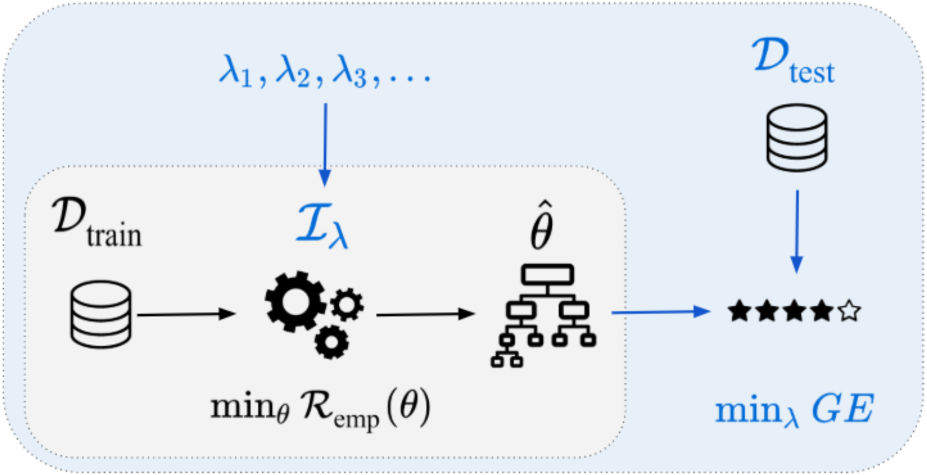
THE ROLE OF HYPERPARAMETERS

- 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).


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

TUNING: A BI-LEVEL OPTIMIZATION PROBLEM

We formally state the nested hyperparameter tuning problem as:

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

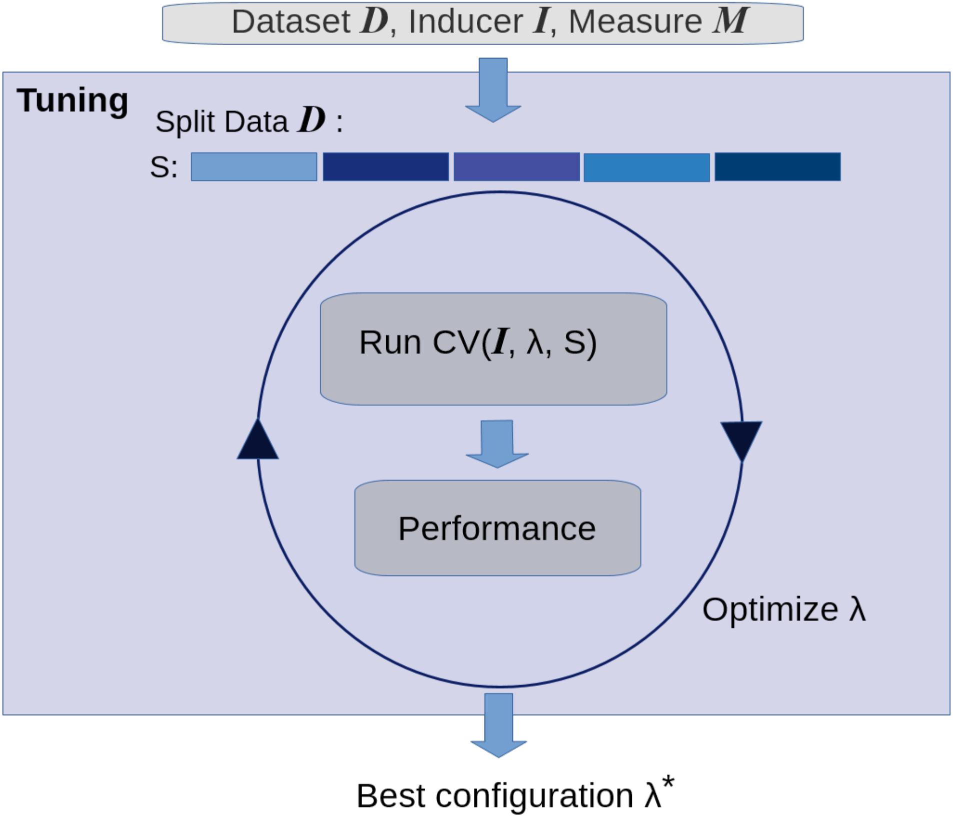
- The learning algorithm $\mathcal{I}(\mathcal{D}_{\text{train}}, \lambda)$ (also: inducer) takes a training dataset as well as hyperparameter settings λ (e.g. the desired depth of a classification tree) as an input.
- $\mathcal{I}(\mathcal{D}_{\text{train}}, \lambda)$ internally performs empirical risk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

TUNING: A BI-LEVEL OPTIMIZATION PROBLEM

The components of a tuning problem are:

- The 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. 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


Dataset $\boldsymbol{D}$, Inducer $\boldsymbol{I}$, Measure $\boldsymbol{M}$
Tuning
Split Data $\boldsymbol{D}$:
S:
Run CV($\boldsymbol{I}$, λ, S)
Performance
Optimize λ
Best configuration λ^*

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

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

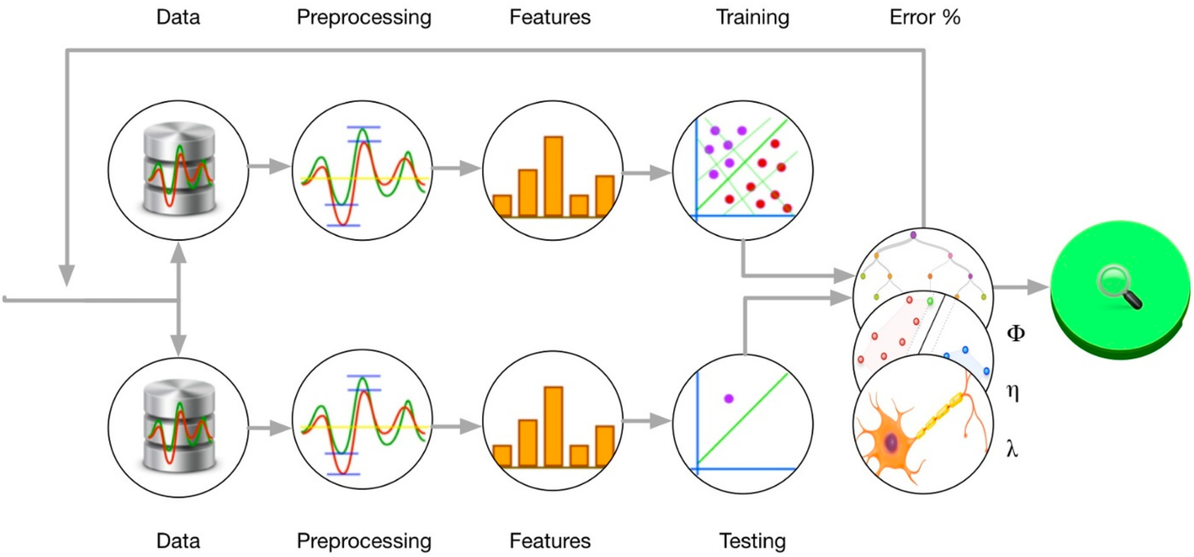
We summarize all hyperparameters we want to tune over in a vector $\lambda \in \Lambda$ of (possibly) mixed type. HPs can have different types:

- 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



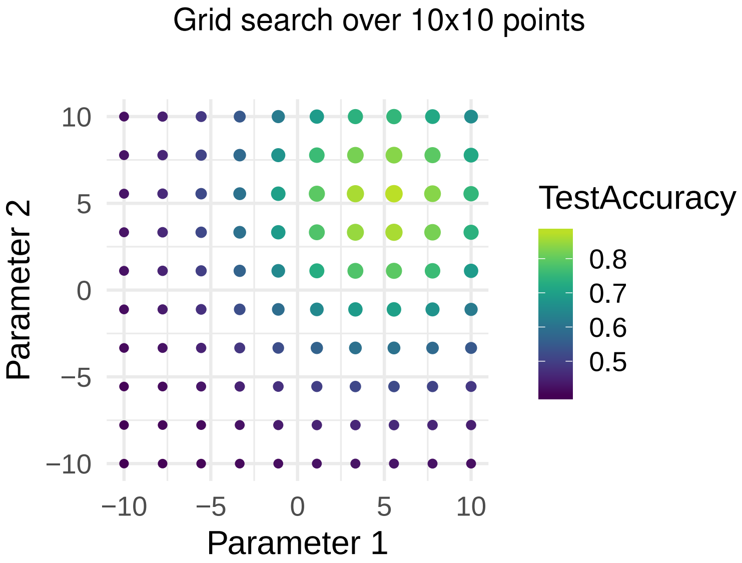
WHY IS TUNING SO HARD?

- 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

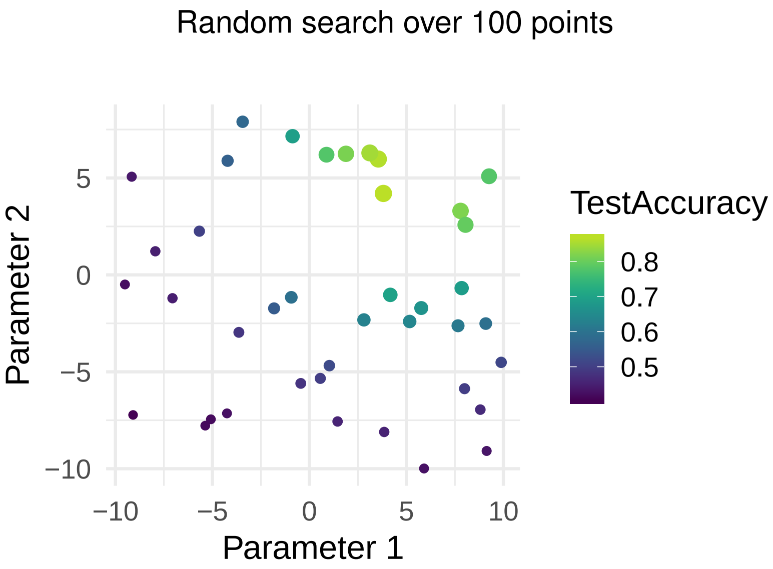
- Very easy to implement, therefore very popular
- All parameter types possible
- Parallelization is trivial

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

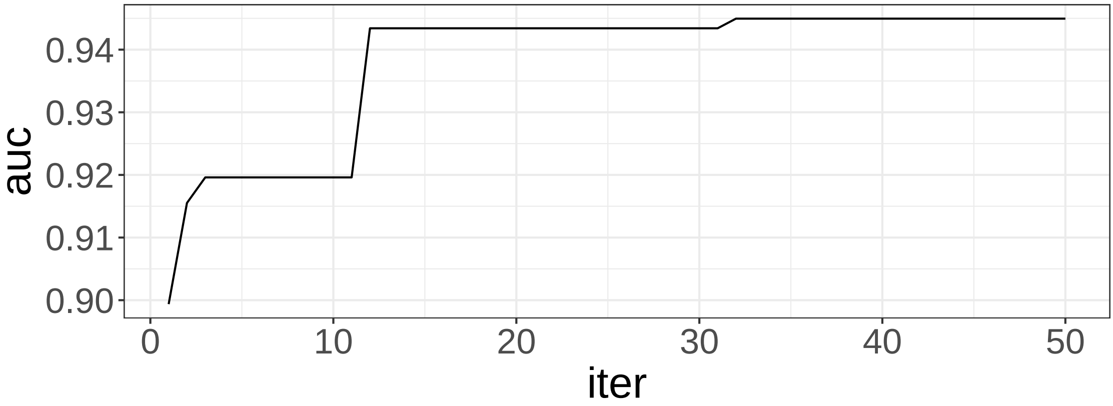
Disadvantages

- 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



Introduction to Machine Learning

Nested Resampling

Department of Statistics – LMU Munich

MOTIVATION

In model selection, we are interested in selecting the best model from a set of potential candidate models (e.g., different model classes, different hyperparameter settings, different feature sets).

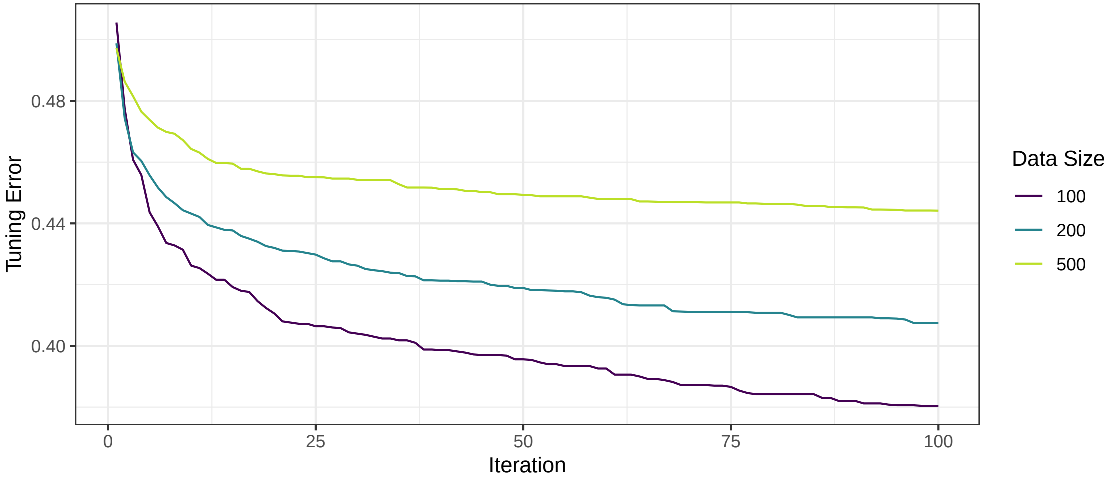
Problem

- We cannot evaluate our finally selected learner on the same resampling splits that we have used to perform model selection for it, e.g., to tune its hyperparameters.
- By repeatedly evaluating the learner on the same test set, or the same CV splits, information about the test set “leaks” into our evaluation.
- Danger of overfitting to the resampling splits / overtuning!
- The final performance estimate will be optimistically biased.
- One could also see this as a problem similar to multiple testing.

INSTRUCTIVE AND PROBLEMATIC EXAMPLE

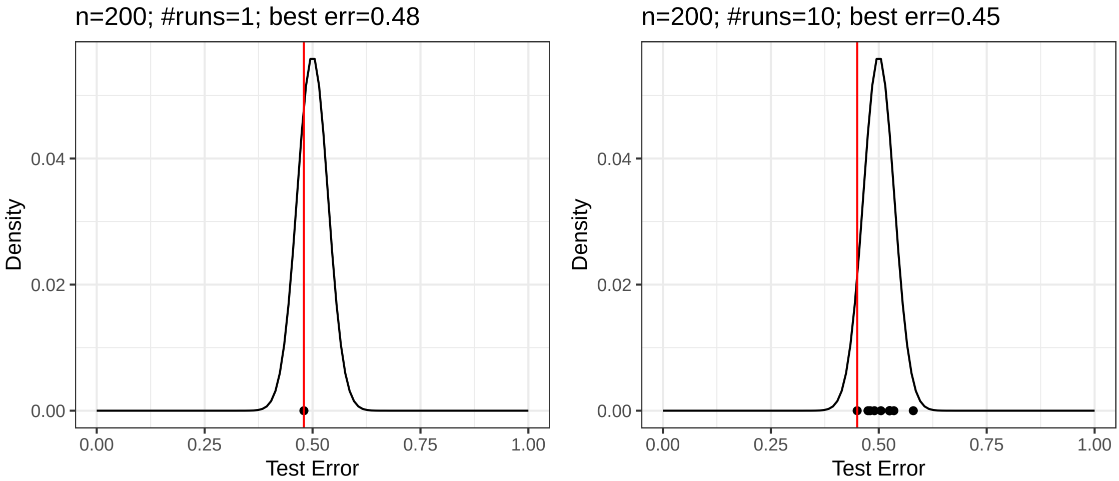
- Assume a binary classification problem with equal class sizes.
- Assume a learner with hyperparameter λ.
- Here, the learner is a (nonsense) feature-independent classifier, where λ has no effect. The learner simply predicts random labels with equal probability.
- Of course, it's true generalization error is 50%.
- A cross-validation of the learner (with any fixed λ) will easily show this (given that the partitioned data set for CV is not too small).
- Now lets "tune" it, by trying out 100 different λ values.
- We repeat this experiment 50 times and average results.

INSTRUCTIVE AND PROBLEMATIC EXAMPLE



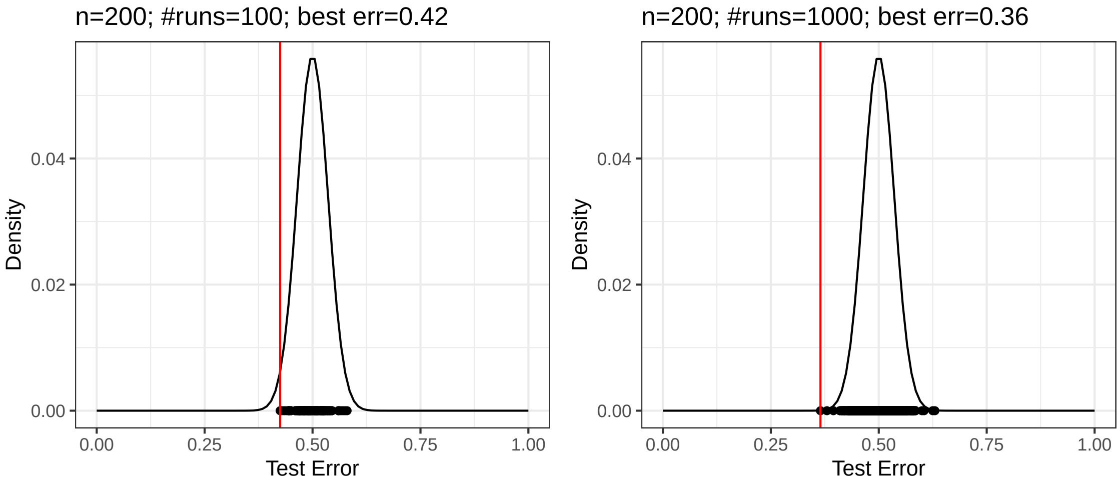
- Plotted is the best "tuning error" (i.e. the performance of the model with fixed λ as evaluated by the cross-validation) after k tuning iterations.
- We have performed the experiment for different sizes of learning data that where cross-validated.

INSTRUCTIVE AND PROBLEMATIC EXAMPLE



- For 1 experiment, the CV score will be nearly 0.5, as expected
- We basically sample from a (rescaled) binomial distribution when we calculate error rates
- And multiple experiment scores are also nicely arranged around the expected mean 0.5

INSTRUCTIVE AND PROBLEMATIC EXAMPLE



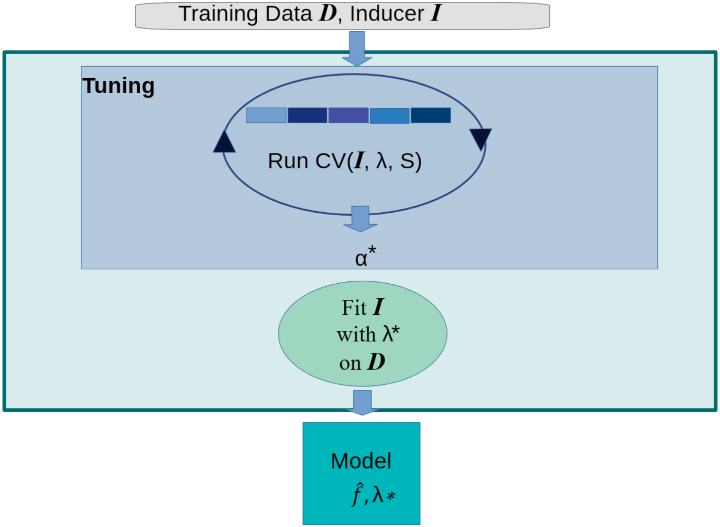
- But in tuning we take the minimum of those!
- The more we sample, the more "biased" this value becomes.

UNTOUCHED TEST SET PRINCIPLE

- Again, simply simulate what happens in model application.
- All parts of the model building (including model selection, preprocessing) should be embedded in the model-finding process **on the training data**.
- The test set we should only touch once, so we have no way of “cheating”. The test dataset is only used once a model is completely trained, after deciding for example on specific hyper-parameters. Performances obtained from the test set are **unbiased estimates** of the true performance.
- For steps that themselves require resampling (e.g., hyperparameter tuning) this results in two **nested resampling** loops, i.e., a resampling strategy for both tuning and outer evaluation.

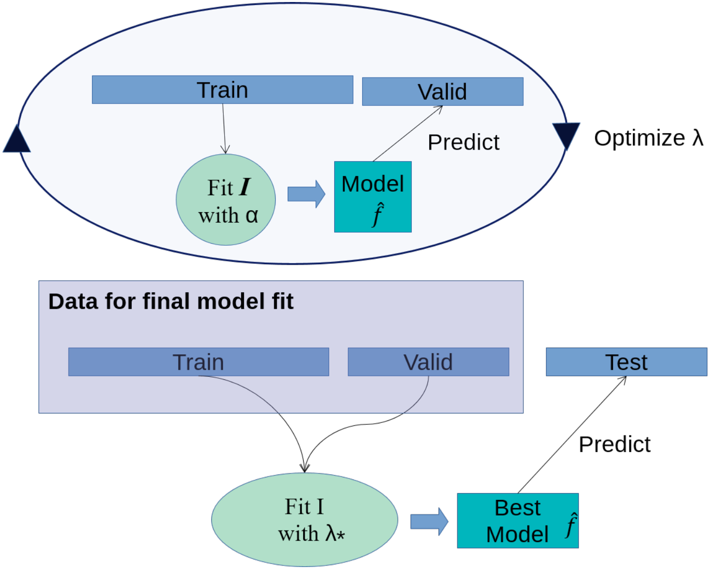
TUNING AS PART OF MODEL BUILDING

- It conceptually helps to see the tuning step as now effectively part of a more complex training procedure.
- We could see this as removing the hyperparameters from the inputs of the algorithm and making it “self-tuning”.


Training Data $\boldsymbol{D}$, Inducer $\boldsymbol{I}$
Tuning
Run CV($\boldsymbol{I}$, λ, S)
α*
Fit $\boldsymbol{I}$ with λ* on $\boldsymbol{D}$
Model
$\hat{f}$, λ*

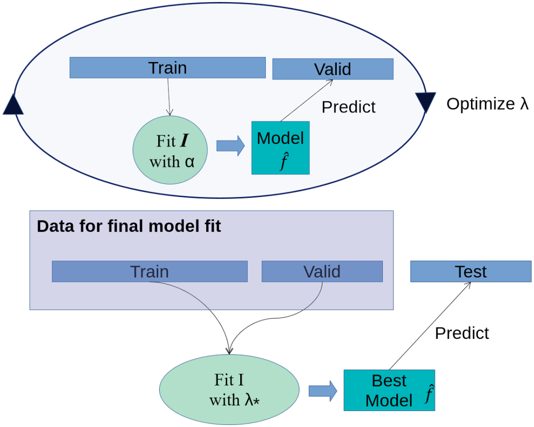
TRAIN VALIDATION TEST

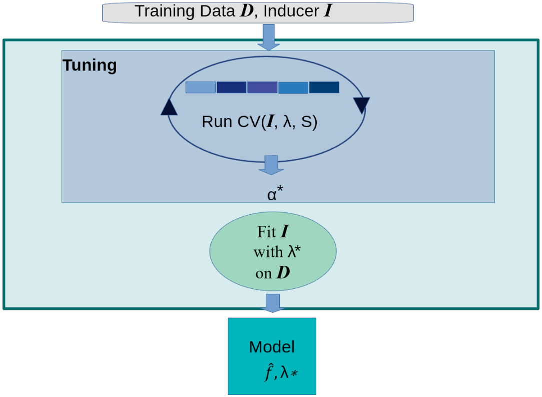
- Simple 3-way split; during tuning, a learner is trained on the training set, evaluated on the validation set
- After the final model is selected, we fit on joint (training+validation) set and evaluate a final time on the test set



TRAIN VALIDATION TEST

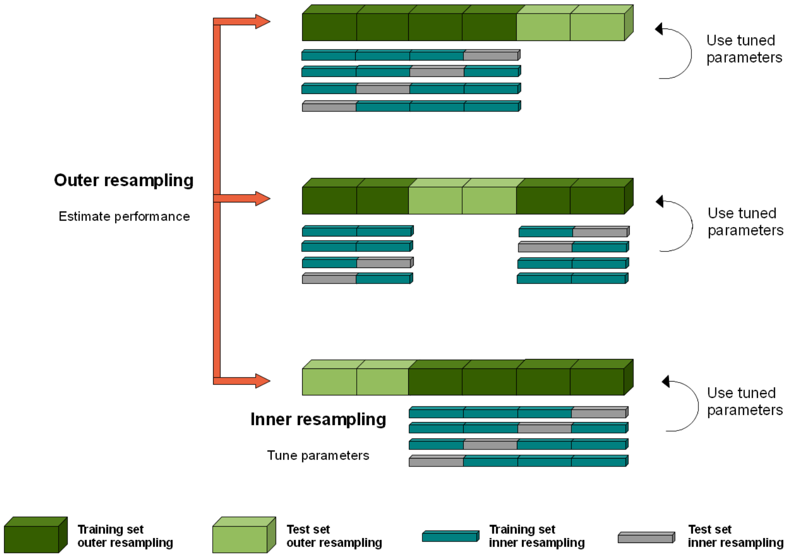
More precisely: the joint train + valid set is actually the training test for the "self-tuning" endowed algorithm.





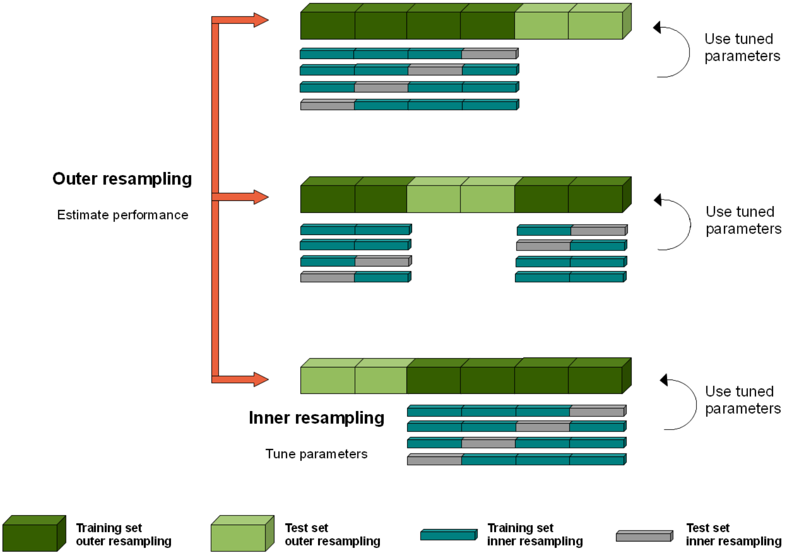
NESTED RESAMPLING

As we can generalize holdout splitting to resampling, we can generalize the train+valid+test approach to nested resampling. This results in two nested resampling loops, i.e., a resampling strategy for both tuning and outer evaluation.



NESTED RESAMPLING

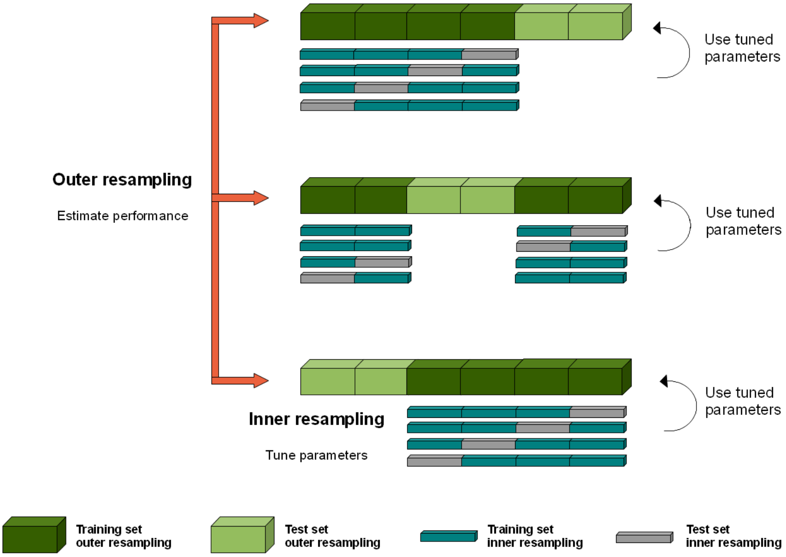
Assume we want to tune over a set of candidate HP configurations $\lambda_i; i = 1, \dots$ with 4-fold CV in the inner resampling and 3-fold CV in the outer loop. The outer loop is visualized as the light green and dark green parts.



NESTED RESAMPLING

In each outer loop we do:

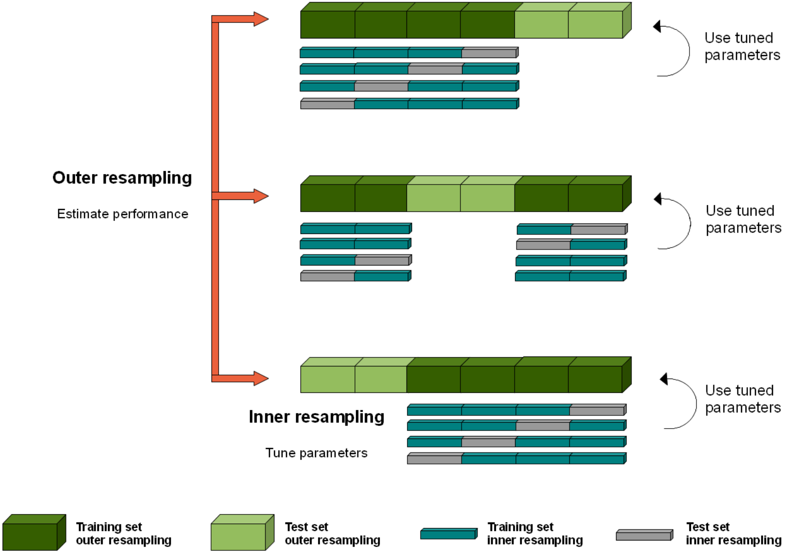
- Split off light green testing data
- Run the tuner on the dark green part, e.g., evaluate each λ_i through 4CV on the dark green part



NESTED RESAMPLING

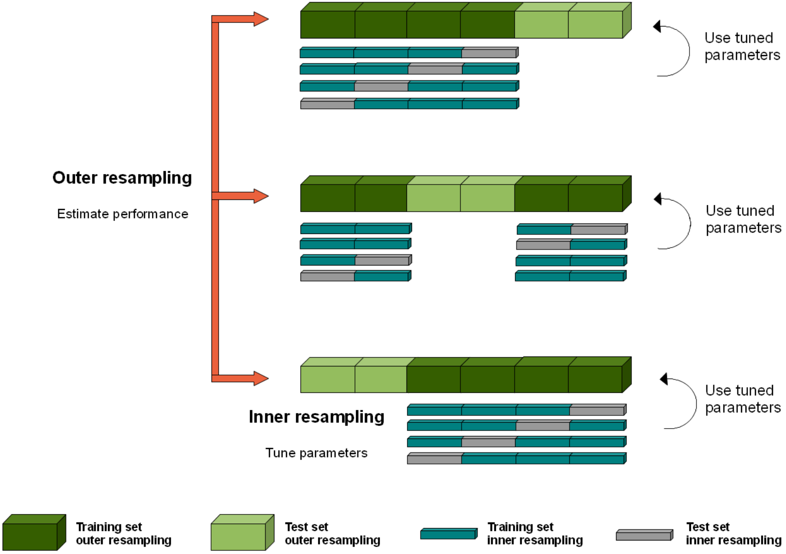
In each outer loop we do:

- Return the winning λ^*
- Re-train the model on the full outer dark green train set
- Predict on the outer light green test set



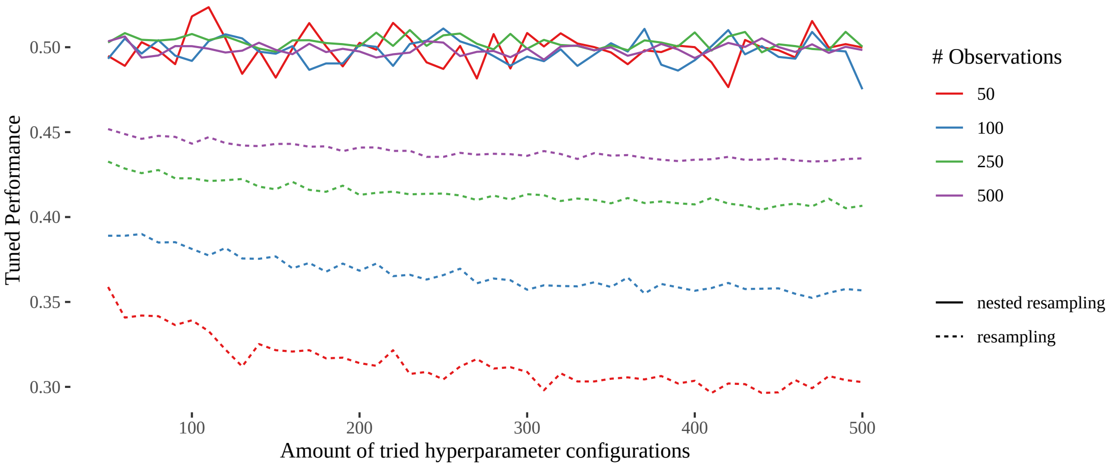
NESTED RESAMPLING

The error estimates on the outer samples (light green) are unbiased because this data was strictly excluded from the model-building process of the model that was tested on.



NESTED RESAMPLING - INSTRUCTIVE EXAMPLE

Taking again a look at the motivating example and adding a nested resampling outer loop, we get the expected behavior:


Tuned Performance
Amount of tried hyperparameter configurations
# Observations
nested resampling
resampling