Applied Machine Learning

Hyperparameter Tuning

Hyperparameter Tuning Problem & Methods

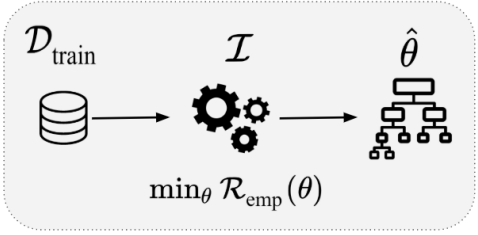
Learning goals

- The Hyperparameter Optimization Problem
- Grid Search, Random Search and Bayesian optimization

Hyperparameter Tuning

MOTIVATING EXAMPLE

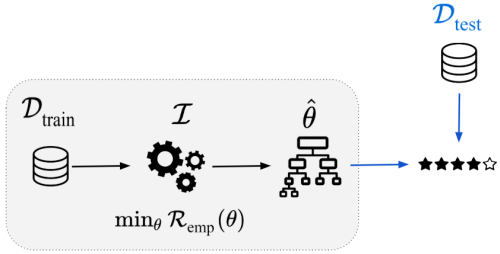
- Given a data set, we want to train a classification tree.
- We feel that a maximum tree depth of 4 has worked out well for us previously, so we decide to set this hyperparameter to 4.
- The learner ("inducer") $\mathcal{I}$ takes the input data, internally performs **empirical risk minimization**,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



MOTIVATING EXAMPLE / 2

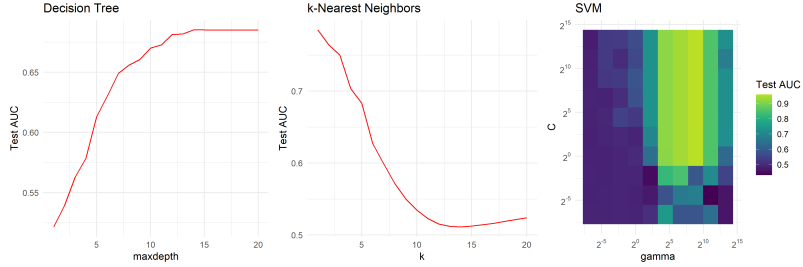
- We are **actually** interested in the **generalization performance** $\mathrm{GE}\left(\hat{f}\right)$ of the estimated model on new, previously unseen data.
- We estimate the generalization performance by evaluating the model $\hat{f} = \mathcal{I}(\mathcal{D}_{\text{train}}, \boldsymbol{\lambda})$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

$$\widehat{\mathrm{GE}}_{\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}}(\mathcal{I}, \boldsymbol{\lambda}, n_{\text{train}}, \rho) = \rho\left(\mathbf{y}_{\mathcal{D}_{\text{test}}}, \boldsymbol{F}_{\mathcal{D}_{\text{test}}, \hat{f}}\right)$$



MOTIVATING EXAMPLE / 3

- 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.
- Consider a simulation example of 3 ML algorithms below, where we use the dataset *mlbench.spiral* and 10,000 testing points. As can be seen, variating hyperparameters can lead to big difference in model's generalization performance.



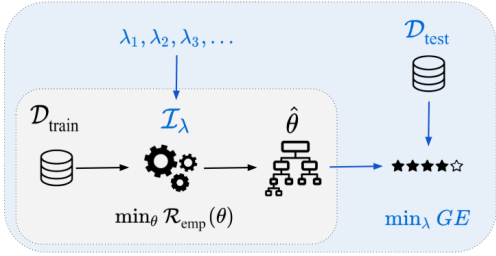
MOTIVATING EXAMPLE / 4

For our example this could mean:

- Data too complex to be modeled by a tree of depth 4
- Data much simpler than we thought, 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 θ are optimized during training. They are an **output** of the training.

Examples:

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

MODEL PARAMETERS VS. HYPERPARAMETERS

/ 2

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

The process of finding the best hyperparameters is called **tuning**.

Examples:

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

MODEL PARAMETERS VS. HYPERPARAMETERS

/ 3

- Number of optimization steps if the empirical risk minimization is done via gradient descent

TYPES OF HYPERPARAMETERS

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

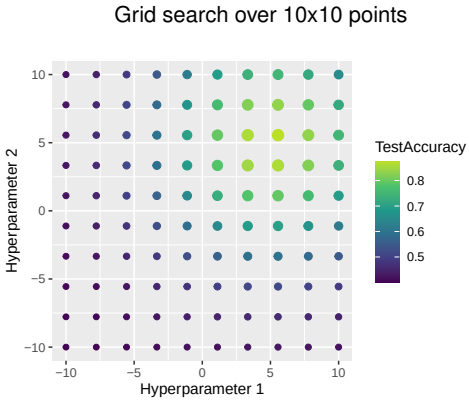
- Real-valued parameters, e.g.:
  - Minimal error improvement in a tree to accept a split
  - Bandwidths of the kernel density estimates for Naive Bayes
- Integer parameters, e.g.:
  - Neighborhood size k for k-NN
  - *mtry* in a random forest
- Categorical parameters, e.g.:
  - Which split criterion 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?

Hyperparameter Tuning Algorithms

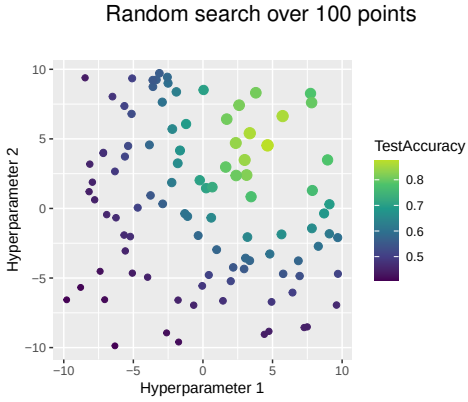
GRID SEARCH

- 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



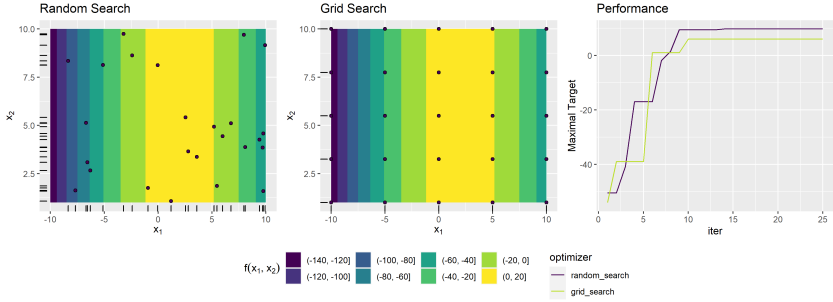
RANDOM SEARCH

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



RANDOM SEARCH VS. GRID SEARCH

We consider a maximization problem on the function $f(x_1, x_2) = g(x_1) + h(x_2) \approx g(x_1)$, i.e. in order to maximize the target, x_1 should be the parameter to focus on.

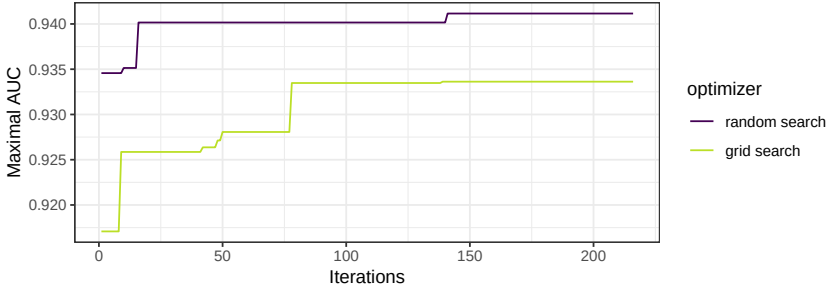


$\Rightarrow$ In this setting, random search is more superior as we get a better coverage for the parameter x_1 in comparison with grid search, where we only discover 5 distinct values for x_1.

TUNING EXAMPLE

Tuning random forest with grid search/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

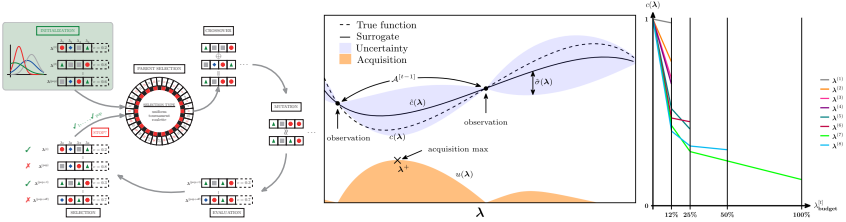


SUMMARY

Property	Grid Search	Random Search
Easy to implement	✓	✓
All parameter types possible	✓	✓
Parallelization is trivial		✓
Does not suffer from the curse of dimensionality		✓
Does not require discretization of hyperparameters		✓
Anytime algorithm		✓

HPO – MANY APPROACHES

- Evolutionary algorithms
- Bayesian / model-based optimization
- Multi-fidelity optimization, e.g. Hyperband



HPO methods can be characterized by:

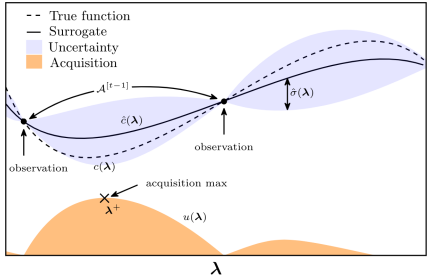
- how the exploration vs. exploitation trade-off is handled
- how the inference vs. search trade-off is handled

Further aspects: Parallelizability, local vs. global behavior, handling of noisy observations, multifidelity and search space complexity.

BAYESIAN OPTIMIZATION

BO sequentially iterates:

1. **Approximate** $\boldsymbol{\lambda} \mapsto c(\boldsymbol{\lambda})$ by (nonlin) regression model $\hat{c}(\boldsymbol{\lambda})$, from evaluated configurations (archive)
2. **Propose candidates** via optimizing an acquisition function that is based on the surrogate $\hat{c}(\boldsymbol{\lambda})$
3. **Evaluate** candidate(s) proposed in 2, then go to 1

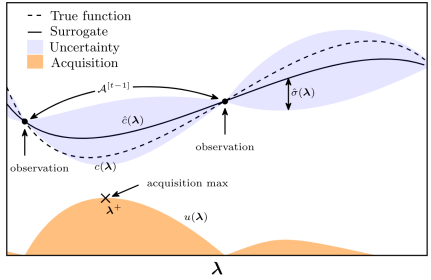


Important trade-off: **Exploration** (evaluate candidates in under-explored areas) vs. **exploitation** (search near promising areas)

BAYESIAN OPTIMIZATION

Surrogate Model:

- Probabilistic modeling of $C(\boldsymbol{\lambda}) \sim (\hat{c}(\boldsymbol{\lambda}), \hat{\sigma}(\boldsymbol{\lambda}))$ with posterior mean $\hat{c}(\boldsymbol{\lambda})$ and uncertainty $\hat{\sigma}(\boldsymbol{\lambda})$.
- Typical choices for numeric spaces are Gaussian Processes; random forests for mixed spaces



Acquisition Function:

- Balance exploration (high $\hat{\sigma}$) vs. exploitation (low $\hat{c}$).
- Lower confidence bound (LCB): $a(\boldsymbol{\lambda}) = \hat{c}(\boldsymbol{\lambda}) - \kappa \cdot \hat{\sigma}(\boldsymbol{\lambda})$
- Expected improvement (EI): $a(\boldsymbol{\lambda}) = \mathbb{E}\left[\max\left\{c_{\text{min}} - C(\boldsymbol{\lambda}), 0\right\}\right]$ where (c_{min} is best cost value from archive)
- Optimizing $a(\boldsymbol{\lambda})$ is still difficult, but cheap(er)

BAYESIAN OPTIMIZATION / 2

Since we use the sequentially updated surrogate model predictions of performance to propose new configurations, we are guided to "interesting" regions of $\boldsymbol{\Lambda}$ and avoid irrelevant evaluations:

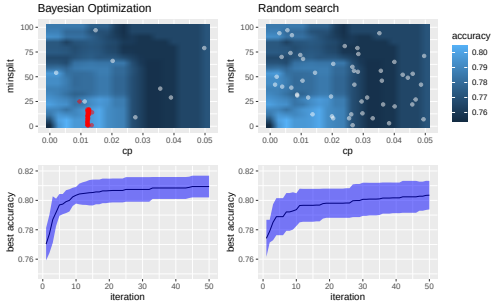


Figure: Tuning complexity and minimal node size for splits for CART on the `titanic` data (10-fold CV maximizing accuracy).
Left panel: BO, 50 configurations; right panel: random search, 50 iterations.
Top panel: one run (initial design of BO is white); bottom panel: mean $\pm$ std of 10 runs.

PRACTICAL ASPECTS OF HPO

- Choosing HPO algorithm
  - For few HPS (1-3), grid search can be used
  - BO with GPs for upto 10 numeric HPs
  - BO with RFs handle mixed HP spaces
  - Random search and Hyperband work well as long as the “effective” dimension is low
  - EAs are somewhat in-between BO and RS, can handle very complex spaces, but less sample efficient than BO
  - **Also: use something that’s stable and robust! More an aspect of the implementation than the algo!**