Hyperparameter Tuning

Tuning : Finding the best Hyperparameters, also called **HPO** "Hyperpar. Optimization"

Many ML Learning Algorithms are quite sensitive to (good) setting of their hyperparams. These control flexibility/complexity of model, certain structural properties and computational aspects of training process. HPs can influence generalization performance in a subtle way.

Hyperparameters λ are not optimized during training! They must be specified in advance in order to give us a **fixed Learner configuration** which we then optimize w.r.t. model params θ on full training set and evaluate on testset(s). [HPs are input of Learner $\mathcal{I}$.]
Later we will embedd HP-tuning into a more complex training procedure, effectively removing HPs as input of Learner Algorithm ≙ self-tuning learner $\mathcal{I}_{\tilde{\Lambda}, \rho, \mathcal{J}} : \mathbb{D} \to \mathcal{H}$

Search Space $\tilde{\Lambda} \subset \Lambda$ with all optimizable HPs and ranges is a bounded subset of the hyperparameter space
certain λs can only be optimized if we restrict their possible values

$\tilde{\Lambda} = \tilde{\Lambda}_1 \times \tilde{\Lambda}_2 \times \ldots \tilde{\Lambda}_l$ with total l Hyperparams to tune. Search methods to find $\hat{\lambda}$: Grid, Random Search, Bayesian Optimization, Hyperbands
How choose limited amount of candidates?

Tuner τ is a mapping which takes dataset $\mathbb{D}$, inducer $\mathcal{I}$, HP search space $\tilde{\Lambda}$, performance measure ρ_e and some Resampling strategy $\mathcal{J}$ => returns Hyperparameter Configuration $\hat{\lambda} \in \tilde{\Lambda}$

$\tau(\mathbb{D}, \mathcal{I}, \lambda, \rho_e, \mathcal{J}) = \hat{\lambda} \approx \lambda^*$ theoretical "findable" optimum $\lambda^* \in \arg\min_{\lambda \in \tilde{\Lambda}} \underbrace{\widehat{GE}(\mathcal{I}, \mathcal{J}, \rho_e, \lambda)}_{c(\lambda)}$ evaluated cost of HP-config.

Evals are stored in archive $\mathcal{A} = (\underbrace{(\lambda^{(1)}, c(\lambda^{(1)}))}_{\mathcal{A}^{[1]},\, t=1}, \underbrace{(\lambda^{(2)}, c(\lambda^{(2)}))}_{\mathcal{A}^{[2]}}, \ldots)$ where index refers to current iteration of HPO algorithm

- Tuning / HPO algorithm is usually a black box, partly thats because many standard optimization techniques (such as GD) dont work here! No derivatives exist. We can only evaluate performance for given HPC via known resampling procedure
- Categorial HPs (split criterion for Class.Trees, distance measure for kNN) and dependency hierarchy of certain HPs (if we use KDE for Naive Bayes → bandwidth?) means that we may have a very complicated mixed structure in our Hyperparam Space. have to make choice after limited amount of evals
- Evaluation $c(\lambda)$ is not exact but stochastic due to resampling and each Eval requires multiple train & predict steps making it computationally expensive

Tuning implies a two-level optimization process. **First:** $\arg\min \widehat{GE}(\mathcal{I}, \mathcal{J}, \rho_e, \lambda)$ in regards to λ HPO
Second: For each λ do ERM on Inducer Level to obtain optimal model param $\hat{\theta}$ (nested within first Level)
also includes Learner eval. on test data, pessimistic bias due to $\mathcal{J}_{train} \overset{\forall \text{ folds}}{<} n$ which resampling method (such as k-fold CV) mitigates
How to combine both? HPO technically comes first, but if we used test data for 2nd lev. there is no unseen data left to evaluate final selected Learner config!

Specifically] The HP-optimal Learner config was found by choosing minimal $\widehat{GE}$ which is calc. by evaluating different ERM-Models on different train-test-splits and averaging over Error Measure for given λ. Chosen $\hat{\lambda}$ is the HP config that has min. $\widehat{GE}$ among all evaluated configs. comparable & reproducible
But eventhough we use Resampling for every inspected λ-config: Overall by repeatedly doing this many times for different λ, the average or clearer the aggregated Test Information "leaks into evaluation" (we choose λ with min of all these averaged GE-estimates), esp. if we use exact same splits $\forall \lambda^{(i)}$
⟹ **OVERTUNING** This $\widehat{GE}$ of final selected Learner config. is optimistically biased $\mathbb{E}[\min_\lambda \widehat{GE}(\lambda)]$ ↓↓ via Overtuning $< \min_\lambda \mathbb{E}[\widehat{GE}(\lambda)]$ actually want this!
In analogy to multiple testing we could also call this a multiple eval. problem. By searching & testing over more and more λ it becomes increasingly likely that one of these HP configs performs (too) good by random Luck. GE-estims can be viewed as a random variable we sample from; imagine $Bin(n_{train}, \pi = MCE)$ for class. task
↓
for Performance Evaluation
Untouched Test Set / Unseen Test Data Principle needs to be adhered to consistently in the whole ML Pipeline! | HP train | HP test (valid.) | Test
For Model building steps that require resampling themselves this implies nested resampling (HP Tuning, Feature Selection etc.) or extra holdout testset

Nested Resampling for Tuning strictly separates model selection and performance evaluation
Outer Resampling: Split all data into repeated train / test ⟹ Retrain model with $\hat{\lambda}$ on all train data, evaluate Learner (config) on test data
Inner Resampling: Split outer train data again into rep. train / test ⟹ HP tuning as above, yielding $\hat{\lambda} = \arg\min c(\lambda)$
Now doing this k-times for k-fold CV or subsampling we will get k unbiased $\widehat{GE}$ to average over
↳ aggregate $\widehat{GE}_{nested} = \frac{1}{k}\sum_{i=1}^{k} \widehat{GE}^{(k)}_{outer}$ estimates Performance of tuned Learner / Learn Algorithm including HP-finding
can do same for other Learner types on identical splits & ρ_e measure ... ⟹ choose model / Learner type with Min $\widehat{GE}_{nest.}$ & train it on ALL data
incl. Tuning with same strategy as Nested Resamplings inner Loop on n! (CV!)
+ before final training one last resampling for HPs! non-nested, new splits
↓
We have number for $\widehat{GE}$ of Tuned Learn. Algo Process to report and compare (fair)

In reality its actually about performance of the entire Learning Pipeline, not just Learner – HP space – Tuning Search Strategy
That entails: Pre-Processing → Scaling / Standardizing, Transformations, Error Fix, Outliers, Value Imputation for NAs etc.
Feature-Engineering / Selection → Filter, Feat. Encoding, Wrapper (RFE), Embedded, Combining Features, Interactions in Model, Dim. Reduction
ERM Loss & Performance Measure. But dont worry: Most of this is chosen upfront based on exp. and common knowledge [we cant bloat up Nested Res. inner loop too much]
Fix design choices → Thresholding, Early stopping in Trees?, Regularization implemented vs overfitting etc.