STATG019 – Selected Topics in Statistics 2018

Lecture 3

Model Tuning, Ensemble Strategies, and Model Selection

Dr Franz J. Király

Course organization

In-Course-Assessment: group registration

Please register your group for ICA 2 on moodle!

Deadline for electronic registration is Mar 22

Registration possible until submission, but must be written after Mar 22

In-Course-Assessment: deadline moved

is now **11:55am on Apr 26** (half a day later)

Lecture 4&5 topics (almost) selected (poll open until 11:55pm)

Time series

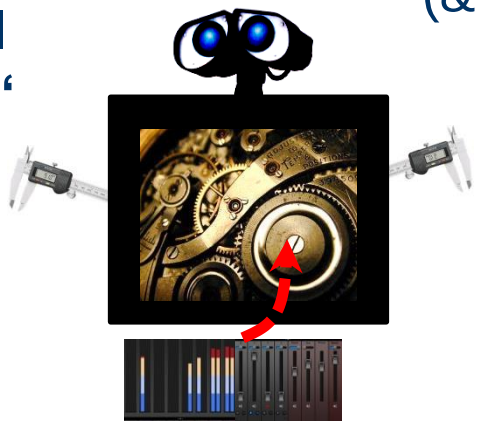
& toolboxes

Response	Average	Total
Validation of probabilistic models including probabilistic classifiers and regressors	9%	2
Validation of unsupervised modelling strategies including clustering and density estimation	14%	3
Model validation in the context of time series related tasks including forecasting	32%	7
Model-specific theoretic guarantees à la Vapnik-Chervonenkis	14%	3
Use and design of model&workflow interfaces for machine learning toolboxes such as mlr/sklearn	32%	7

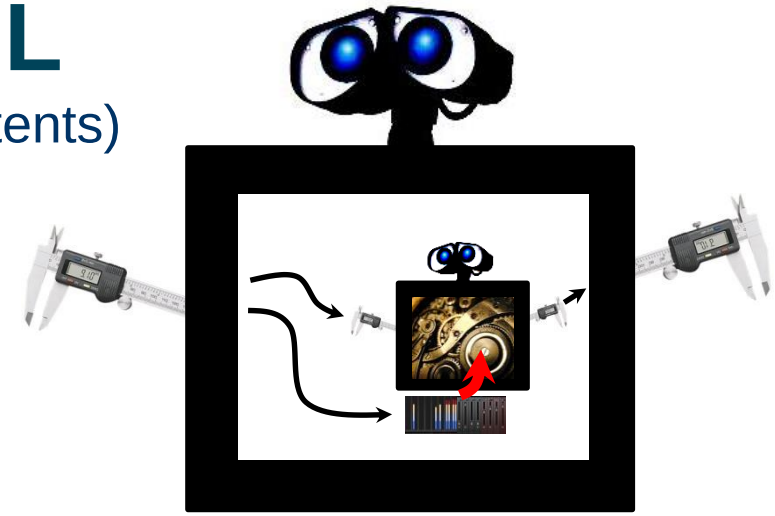
Meta-Strategies in ML

(& table of contents)

„Model tuning“

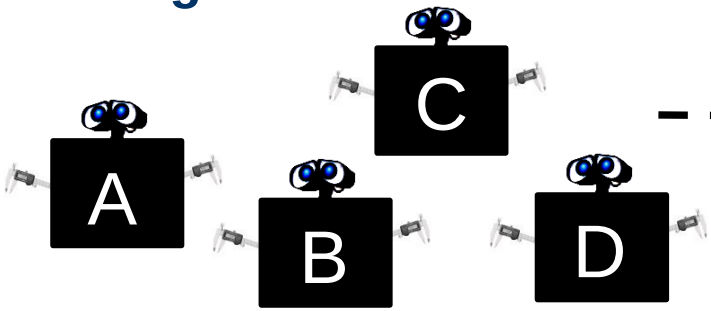


Model with tuning parameters

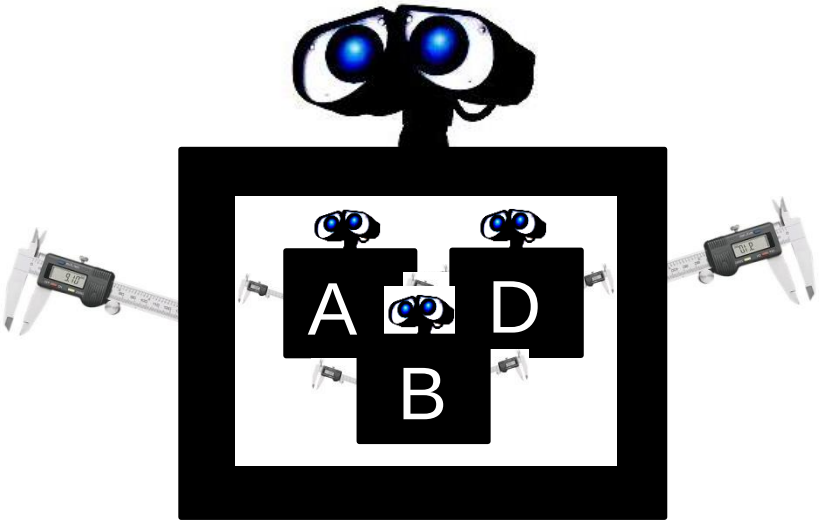


Best tuning parameters are determined using data-driven tuning algorithm

„Ensemble learning“



a number of (possibly „weak“) models



„strong“ ensemble model

Meta-modelling overview

Model selection: among multiple models/strategies, find best

As in validation, *model class* specific vs *model class* agnostic

Model class specific: F-tests for nested models, AIC/BIC/WAIC

(penalized) deviance/likelihood, VC/PAC/MDL learning theory based

Model class agnostic: generalization loss estimate based (today)

(dis)advantages as before: specific may be better if applicable

Model tuning: in given model, algorithmically set free parameters

Special case of model selection: *fixed* model X, param choices Y

May also be seen as generalization: parameter = „which model“?

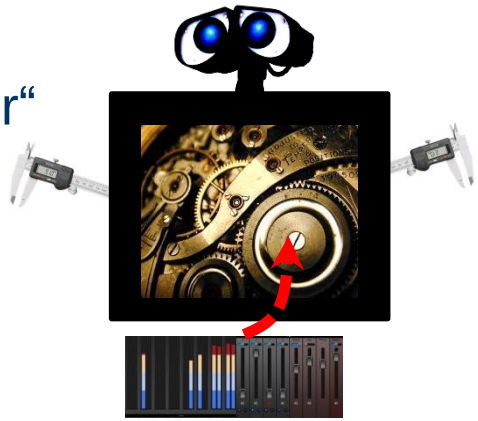
Ensembling & composition: combine multiple models to improve

Model selection is a special case: combination by selecting one

Composition: can involve learners for other tasks (e.g., feature extraction)

Re-sampling based hyper-parameter tuning

What is a hyper-parameter?



aka „tuning parameter“

Common examples:

Fit intercept yes/no? in linear regression

Which variables to use as inputs in modelling

Number of trees in a random forest model

Full architecture of a neural network („deep learning model“)

What is the difference to model parameters such as:

Coefficient values in linear model Weights in a neural network

Hyper-parameter = strategy does not set/fit it given data

Mathematical definition of hyper-parameter?

There is no property-based definition of „hyper-parameter“!

Hyper-parameter is *interface* convention: „settable from outside“

Same parameter may be „hyper-“ or not, depending on interface!

Hyper-parameters that usually need tuning

otherwise the methods not work very well

based on own practical experience and literature recommendations

Regularization parameters

as in shrinkage, support vector machines, gradient methods $\lambda, C, \gamma, \ldots$

Non-linear regression learners

Choice of smoothing basis, interpolation nodes, degree, etc (whichever applies)

Boosting/bagging methods and other ensembles

Number and probability of re-samples, ensemble parameters $p, \kappa, \ldots$

Kernel methods: kernel SVM, ridge regression etc

Choice of kernel, choice of kernel parameters linear vs Gaussian; $\theta, \sigma, \ldots$

Pre-processing for supervised learning

Variable transforms, variable selection, dimension reduction parameters

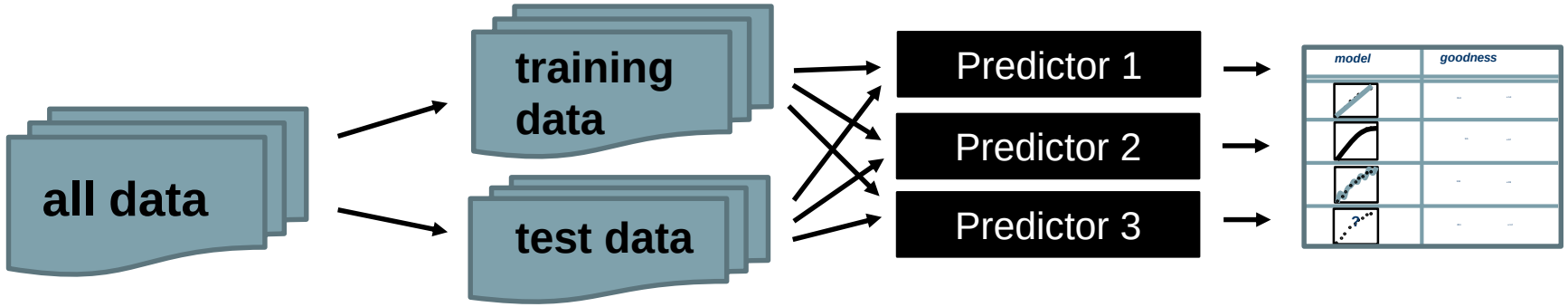
Neural networks for regression and classification (or in any form)

structure of the network; learning parameters; tuning is usually part of the fitting

„manual“ tuning is problematic, especially when learner can „hyperparameter-overfit“

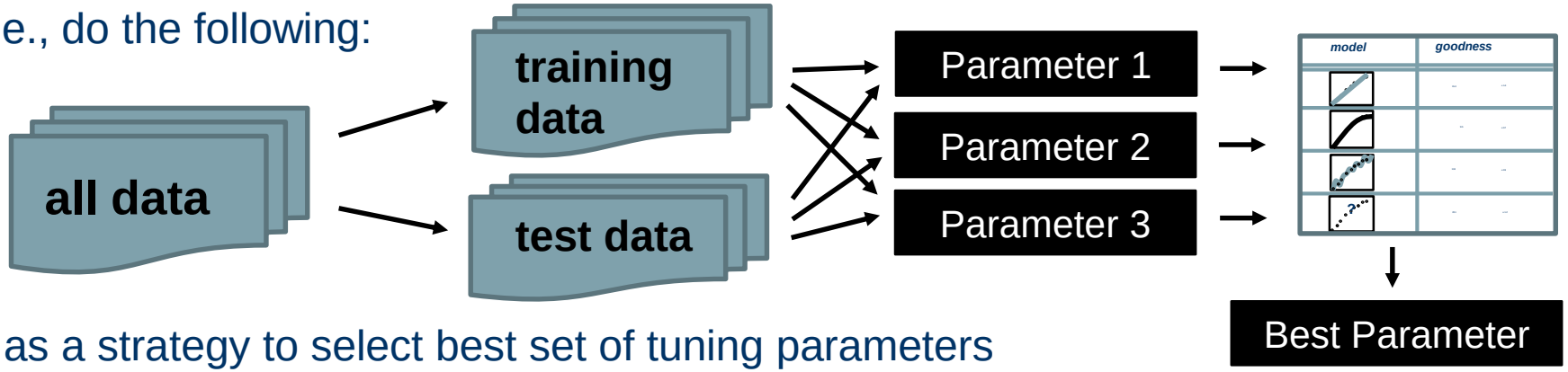
Model tuning by cross-validation

Recall the workflow blueprint for model comparison (lecture 3)



Idea: instead of learning machines, compare different *parameter settings*
then, for auto-tuned parameters choose those with lowest prediction error

i.e., do the following:



as a strategy to select best set of tuning parameters

what is the predictive goodness of [method X] with [best parameter]?

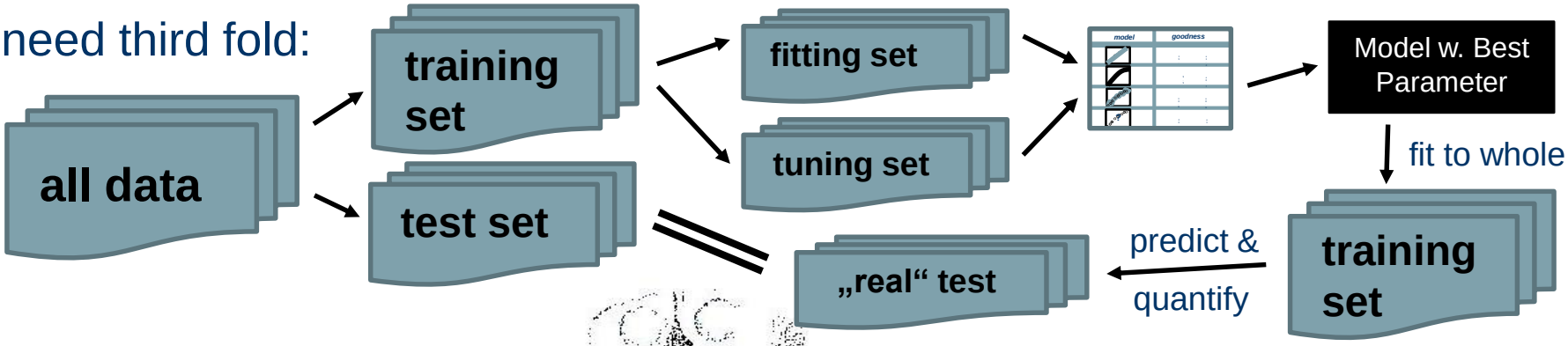
Have we observed [method X] with [best parameter] on ***unseen data?***

Nested re-sampling for tuning & validation

*Have we observed [method X] with [best parameter] on **unseen data?***

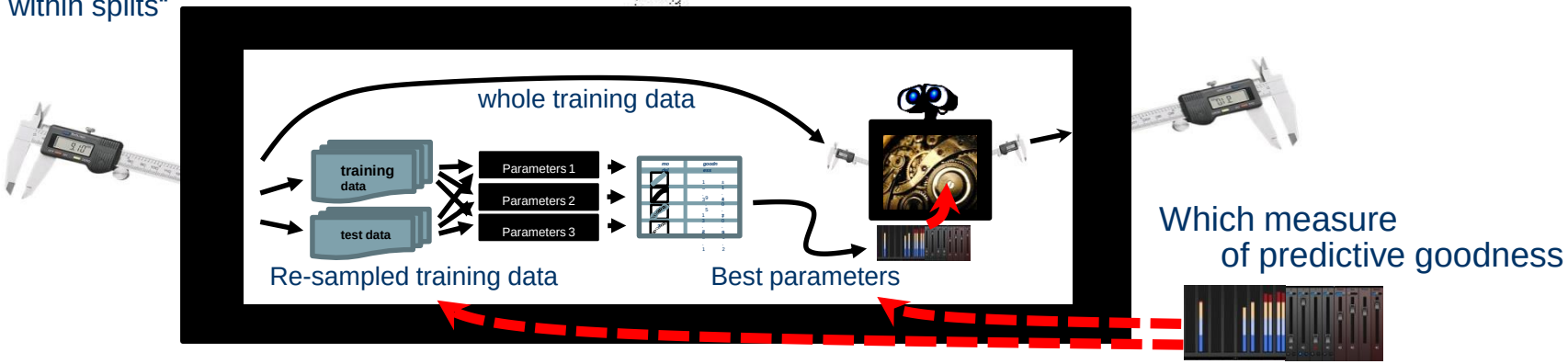
NO! *To determine [best parameter], all data has been used/seen!*

need third fold:



Multi-fold-schemes are nested:

„splits within splits“



Important caveat: the „inner“ fitting/tuning splits need to be part of any „outer“ ***validation*** split ***otherwise validation is not out-of-sample!***

Methods are usually less sensitive to these „new“ tuning parameters

The model tuning paradox explained

simplified: how minimization (e.g., of loss) introduces additional bias

Consider $\widehat{\mu}_1 \sim \mathcal{N}(\mu_1, 1)$ estimating μ_1

$\widehat{\mu}_2 \sim \mathcal{N}(\mu_2, 1)$ estimating $\mu_2 \geq \mu_1$ with $\widehat{\mu}_1, \widehat{\mu}_2$ independent

$\widehat{\mu}_i$ are unbiased estimates of μ_i.

Consider $\widehat{\mu}_{(1)} := \min(\widehat{\mu}_1, \widehat{\mu}_2)$ estimating $\min(\mu_1, \mu_2)$

$$\begin{aligned} F_{\widehat{\mu}_{(1)}}(x) &= 1 - (1 - F_{\widehat{\mu}_1}(x)) \cdot (1 - F_{\widehat{\mu}_2}(x)) \\ &= 1 - (1 - \Phi(x - \mu_1)) \cdot (1 - \Phi(x - \mu_2)) \\ &= \Phi(x - \mu_1) + \Phi(x - \mu_2) \cdot (1 - \Phi(x - \mu_1)) \\ &\gneq \Phi(x - \mu_1) = F_{\widehat{\mu}_1}(x) \geq \Phi(x - \mu_2) = F_{\widehat{\mu}_2}(x) \end{aligned}$$

Thus $\mathbb{E}(\widehat{\mu}_{(1)}) = \int x \, dF_{\widehat{\mu}_{(1)}}(x) \lneq \int x \, dF_{\widehat{\mu}_1}(x) = \mathbb{E}(\widehat{\mu}_1) = \mu_1$

$\widehat{\mu}_{(1)}$ is biased, and negatively biased! Even if $\mu_1 = \mu_1$!

Tuning & independent test set validation

Setting: **i.i.d. *test* data** $(X_1, Y_1), \ldots, (X_M, Y_M) \underset{\text{i.i.d.}}{\sim} (X, Y)$ t.v.in $\mathcal{X} \times \mathcal{Y}$

***tuning* data** $(X'_1, Y'_1), \ldots, (X'_{M'}, Y'_{M'}) \sim (X, Y)$

prediction functional $f_\theta : \mathcal{X} \to \mathcal{Y}$ *with tunable parameter* $\theta \in \Theta$

loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ e.g., $L : (\widehat{y}, y) \mapsto (\widehat{y} - y)^2$ (assume finite Θ)

Simple exhaustive tuning based on single-split re-sampling

1. compute $\widehat{\varepsilon}'(f_\theta) := \frac{1}{M'} \sum_{i=1}^{M'} L(f_\theta(X'_i), Y'_i)$ for all $\theta \in \Theta$

By proposition in lecture 2: $\widehat{\varepsilon}(f_\theta) \approx \varepsilon(f_\theta) + \mathcal{N}(0, \frac{C}{M'})$

2. select $\theta_{opt} := \underset{\theta \in \Theta}{\operatorname{argmin}}\, \widehat{\varepsilon}(f_\theta)$ „the best/tuned parameter“

In analogy to previous slide: $\widehat{\varepsilon}(f_{\theta_{opt}})$ is *not* a good estimate for $\varepsilon(f_{\theta_{opt}})$ **!**

overfitting!

Single-split validation on an independent test set:

compute $\widehat{\varepsilon}(f_{\theta_{opt}}) := \frac{1}{M} \sum_{i=1}^{M} L(f_{\theta_{opt}}(X_i), Y_i)$

By proposition in lecture 2: $\widehat{\varepsilon}(f_{\theta_{opt}}) \approx \varepsilon(f_{\theta_{opt}}) + \mathcal{N}(0, \frac{C}{M})$

reliable estimate of expected loss
(if f_θ are non-random)

Re-sampling based tuning strategies

differ by *which parameters settings* are compared in *which sequence* and by *which principles*

Advanced schemes consider nested re-sampling as (meta)-optimization problem where *function evaluation is costly* and closeby settings yield similar results

tuning scheme	*How parameters are optimized*	*pros/cons*
grid search	all combinations on a pre-specified grid are tried and compared enlarge grid if optimum on boundary	Pro: easy and always applicable con: long run-time due to possible combinatorial explosion
random search	combinations are tried randomly until some „total cost“ is reached	usually better run-time than grid search & not much worse
Friedman-Racing **Maron, Moore (1997)**	parameter combinations are re-tested and re-moved by testing comparison „race winner“ is the one not removed	often better & faster than above but: heuristic and not exhaustive
Further advanced optimization	open research field, approaches include: Evolutionary, simulated annealing, meta-learning, Proxy functions, Bayesian optimization, stochastic search, many heuristics,…	most tend to work well not all are readily available not all work for discrete params

Tuning: frequent issues and mistakes

Frequent mistake: tuning and validation splits overlap

For example: train on set A, tune on set B, then test/validate on set A

Or: split all data into train/tune, then again all data into train/test

(this is the mistake in the Delgado benchmarking study)

Unproblematic alternative: tuning split is contained in training sets

Issue: re-fit method to the full training set with tuned parameter?

Potential problem: randomness of a prediction *strategy* (vs functional)

Recommendation: yes, guarantee is then given by validation split

Issue: use of re-sampling scheme with replacement (e.g., .632 BS)

Potential problem: duplicated data points end up in both train&tune

Publications ignore assumptions for bootstrap, violated in practice

Recommendation: do *not* use! (unless you publish the paper that fixes this)

Advanced tuning and black-box-optimization

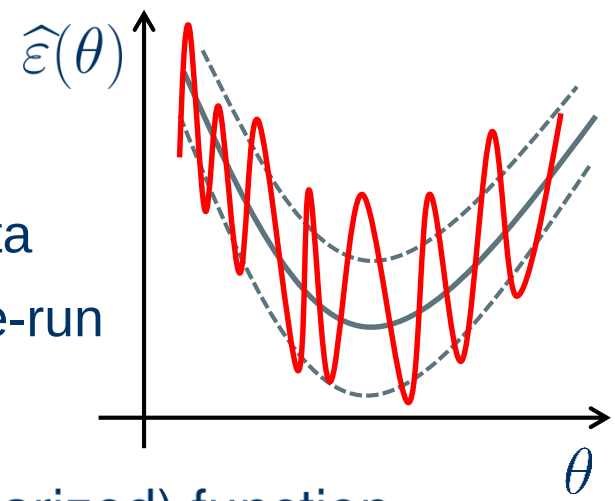
inherent issue with grid-tuning

and loss estimate based parameter selection:

$\widehat{\varepsilon}(\theta)$ carries variance caused by training data

and independent randomness for each re-run

For each evaluation: $\widehat{\varepsilon}(\theta) = \varepsilon(\theta) + \text{noise}$



Advanced idea: estimate $\widehat{\varepsilon}(\theta)$ as a (regularized) function

General problem class: *black-box function optimization*

Given: costly algorithm to obtain $g(\theta) + \epsilon_\theta$ for unknown $g : \Theta \to \mathbb{R}$, noise ε_θ

Estimate: $\operatorname*{argmin}_{\theta \in \Theta} g(\theta)$

Goodness: $L(g(\widehat{\theta}), g(\theta_{opt}))$ is small

& few evaluations necessary

Solutions of this type: supervised modelling on parameter space

Bayesian optimization active learning stochastic & genetic search

Major issue in ML: dependence of the evalutes (through data)

Ensemble learning and model composition

Overview of Ensemble Learning Strategies

(some)

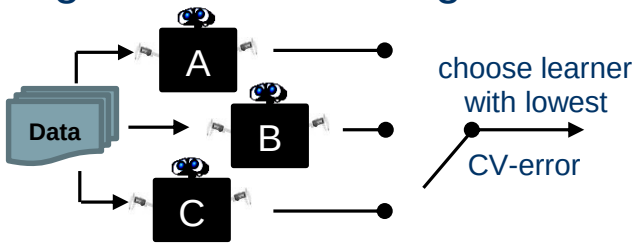
Setting: base learners are given, which can be a „committee“ of distinct learners; or obtained by re-sampling data and/or sub-setting features of a single learner

Cross-validation multiplexer strategy

this is CV model selection as a *meta-strategy*.

an important baseline for ensemble strategies

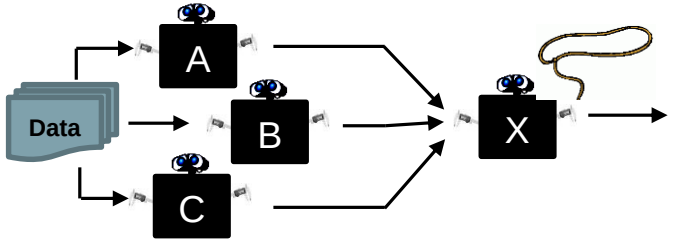
usually not helpful in „classical setting“



Stacking

feed predictions or residuals in a „committee“ learner, usually OLS or LASSO

predictions input should be out-of-sample (e.g.CV)

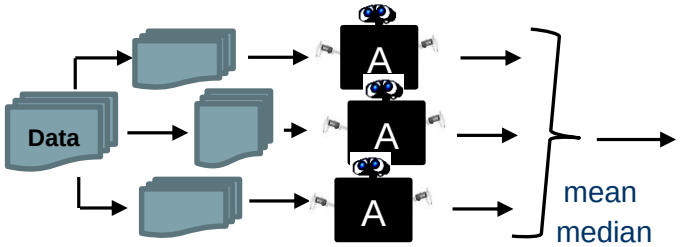


Bagging/Bragging

(= bootstrap aggregation)

on bootstrap sub-samples, learner is fit, results are mean/median-aggregated

with the aim of reducing the variance

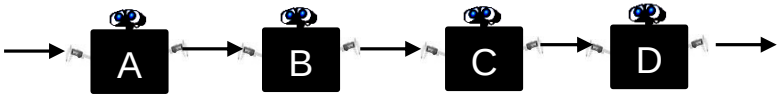


Boosting

Residuals are repeatedly passed to a further learner, which is fit to residuals/loss

final learner is „chain“ of learners

Learners can be fixed or also optimized in the process



A frequently bagged/boosted model

Regression Trees and Decision Trees

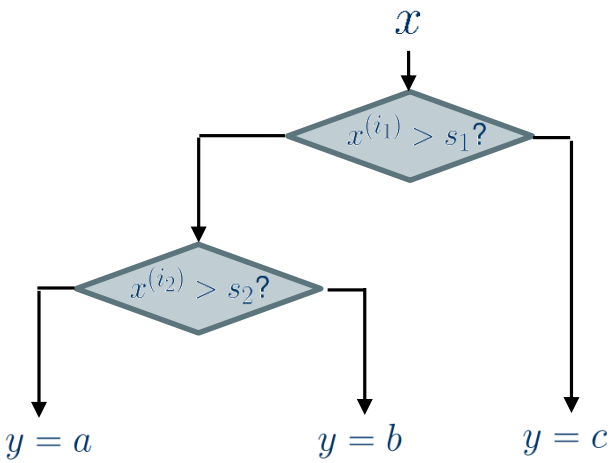
Learning task: ***Regression & Classification***

Predict target $y \in \mathbb{R}$ **from** features $x \in \mathbb{R}^n$

Main ideas: prediction rule is a tree rule

each node is comparison of single variable

fitted by minimizing prediction error on data



Fitting to **data** $x_1, \ldots, x_N \in \mathbb{R}^n$ **labels** $y_1, \ldots, y_N \in \mathbb{R}$

At each step,

Find: feature $i \in \{1, \ldots, n\}$ threshold $s \in \mathbb{R}$

e.g. by minimizing squared residuals of split,

$R(i,s) = \text{Var}(y_j | x_j^{(i)} < s) + \text{Var}(y_j | x_j^{(i)} > s)$ or MMCE, Gini (classif.)

Combine with greedy/pruning strategies

Predicting from **features** $x_* \in \mathbb{R}^n$ **a label** $y_* \in \mathbb{R}$

Go through the tree using features

Tuning parameters

Depth of tree

Number of nodes

How to select variables

How to select thresholds

Advantages

Parsimonious

Well-interpretable

Relatively fast to fit/select

Disadvantages

Usually does not work really well

Prone to overfitting

Not robust without added tuning

Variants Bayesian trees

MARS Random feature trees

Ref's EoSL section 9.2

Bagging and Random Forests (Breiman 1990s)

Main mathematical idea: use variance reduction by averaging

Suppose one can obtain a number of slightly correlated random variables

$$Z_1, \ldots, Z_M \quad \text{Var}(Z_i) = \sigma^2 \quad \text{Corr}(Z_i, Z_j) = \rho \ \text{ for } i \neq j$$

Then $$\text{Var}\left(\frac{1}{M}\sum_{i=1}^{M} Z_i\right) = \rho \cdot \sigma^2 + \frac{1-\rho}{M} \cdot \sigma^2 \qquad \leq \sigma^2 \ \text{(unless } Z_i \text{ are equal)}$$

So if Z_i are correlated re-samples of a prediction

averaging reduces the variance while not changing the bias

Hence by bias-variance trade-off, expected out-of-sample error is reduced

Main technical observation:

Regression/decision trees are positively correlated

when randomly sub-sampling data (rows) and features (columns)

Note: correlation here is conditional on test data, not on the model or training data

Remark: this holds for a number of other „low-tech“ predictors as well,

such as logistic regression, linear models, shallow neural networks

Out-of-bag and variable importance

Out-of-bag predictions

„out-of-sample“ predictions on the data batch can be achieved by
using only trees from boosting sub-samples without that data point

tree sampling is i.i.d., so *out-of-bag is asymptotically equivalent to out-of-sample*

out-of-bag features can be used in tuning, error estimation, or both

But: do *not* use in performance estimation for model comparison
since for proper comparison all models need to be validated the same way

Variable importance

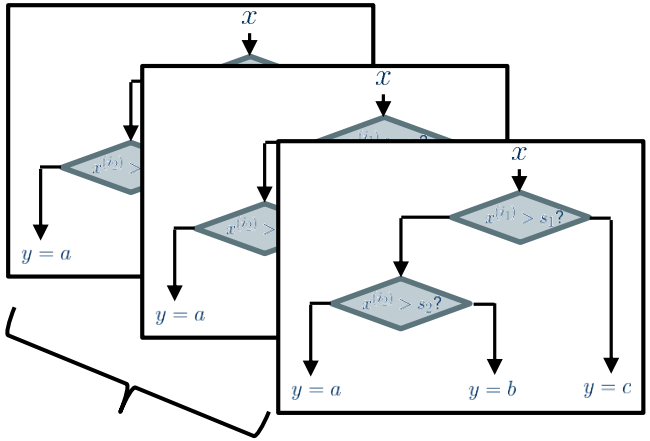
decrease in out-of-bag estimated accuracy when permuting variable
out-of-bag proxy to out-of-sample permutation importance

Caveat: will underestimate „importance“ in presence of correlations
(or duplicated variables)

Breiman‘s Random Forests

Learning task: ***Regression & Classification***

Predict target $y \in \mathbb{R}$ **from** features $x \in \mathbb{R}^n$



Main ideas:

Bagging classification/regression trees

in a variant where set of features is sub-sampled

Fitting to **data** $x_1, \ldots, x_N \in \mathbb{R}^n$
labels $y_1, \ldots, y_N \in \mathbb{R}$

Fit M classification/regression trees T_i

each on a random subset of $m < n$ variables

$$T(x) := \frac{1}{M} \sum_{i=1}^{M} T_i(x)$$

Predicting **from** **features** $x_* \in \mathbb{R}^n$
a label $y_* \in \mathbb{R}$

Substitute into fitted prediction function

$$\widehat{y}_* = T(x_*) = \frac{1}{M} \sum_{i=1}^{M} T_i(x_*)$$

Tuning parameters

Number of trees M

Number of variables m

Individual tree parameters

Optional: sub-sampling data

Advantages

Often the best/one best predictor

Out-of-bag estimates of error

Very robust w.r.t.tuning parameters

Disadvantages

Somewhat lacking interpretability

Moderate computational demand

Model aligned with coordinates

Variants Robust bagging

Additional randomisation to de-correlate

Ref’s EoSL chapter 15

Breiman 2001 Ho 1995

Gradient Boosting (Schapire 1990)

Setting: base learners $f_1, \ldots, f_M : \mathbb{R}^n \to \mathbb{R}$ training data $x_1, \ldots, x_N \in \mathbb{R}^n$, $y_1, \ldots, y_N \in \mathbb{R}$

Main idea: Update model residuals with base learners iteratively as follows

$F_0, F_1, \ldots, F_m$

0. $F_0 : \mathbb{R}^n \to \mathbb{R}$ = predicting the training mean *initialization*

For $m = 1, \ldots, M$

1. Fit f_m to the residual sample $(x_1, \rho_1), \ldots, (x_N, \rho_N)$

where $\rho_i = y_i - F_m(x_i)$ are current residuals

2. $\gamma_m := \arg\min_{\gamma} \sum_{i=1}^{N} (y_i - F_m(x_i) - \gamma f_m(x_i))^2$ *finding update coefficient*

3. $F_m := F_{m-1} + \gamma_m \cdot f_m$ *gradient update*

End For & 4. Return F_M ***„gradient boosting“***

Usually one uses loss functions other than squared loss, especially in classification

Further good ideas for gradient boosting trees: **(Friedman 1990s)**

(empirical)

Also do *boosting*
(further variance reduction)

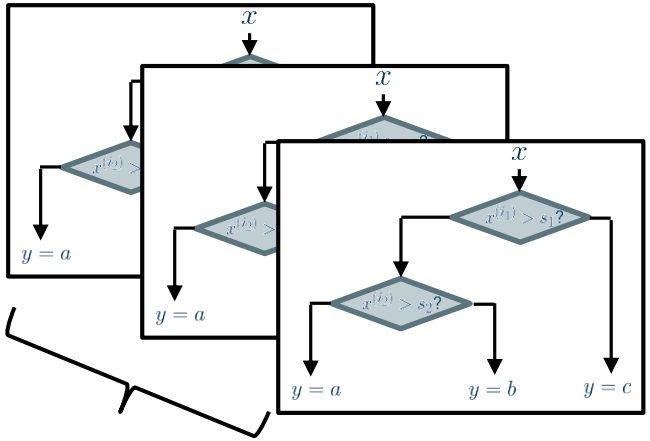
Minimize loss per „tree branch“
(improves fit and bias)

Data weights
(key performange gain)

Stochastic Gradient Boosted Trees

Learning task: ***Regression & Classification***

Predict target $y \in \mathbb{R}$ **from** features $x \in \mathbb{R}^n$



Main ideas:

Bagged and boosted classification/regression trees

Boosting-adaptive determination of tree weights

Fitting to **data** $x_1, \ldots, x_N \in \mathbb{R}^n$ **labels** $y_1, \ldots, y_N \in \mathbb{R}$

SGB algorithm leads to M trees T_i

each on a random sub-sample of size s

$$T(x) := \mu + \sum_{i=1}^{M} \gamma_i \cdot T_i(x)$$

Predicting **from features** $x_* \in \mathbb{R}^n$ **a label** $y_* \in \mathbb{R}$

Substitute into fitted prediction function

$$\widehat{y}_* = T(x_*) = \mu + \sum_{i=1}^{M} \gamma_i \cdot T_i(x_*)$$

Tuning parameters

Number of trees M

Sub-sample size s

Individual tree parameters

Loss function in the algorithm

Advantages

Often the best/one best predictor

Out-of-bag estimates of error

Very robust w.r.t.tuning parameters

Disadvantages

Somewhat lacking interpretability

Moderate computational demand

Model aligned with coordinates

Variants Gradient boosted trees

Regularized additive trees

Ref's EoSL section 10.10 and chapter 10

Schapire 1990 Friedman 1999

On selection vs validation

Key difference between model selection and validation:

Purpose of model selection is finding a good model candidate

Purpose of model validation is answering whether model is sensible

Thus, model selection should always be followed by validation

(while of course model validation can exist without prior model selection)

This particularly holds for model selection via the multiplexer.

Cave: *inference about the „best“ model is substantially different from inference about a fixed model which happens to be best.*

The model selection „paradox“ always impacts the first!

Another important caveat:

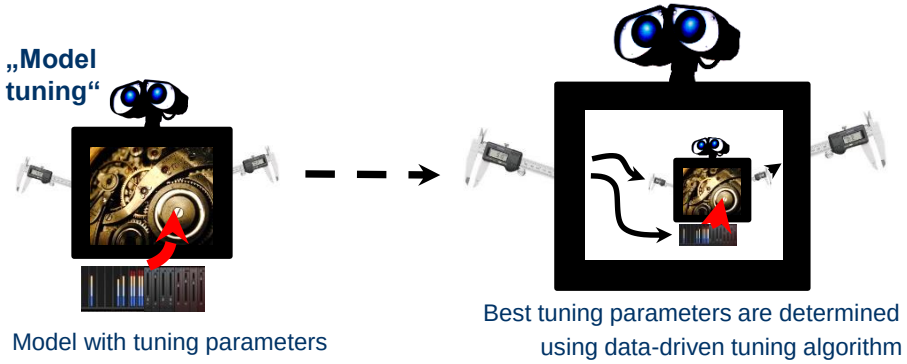
Be careful which guarantees are for strategies vs trained models

„random functions/estimators“ „fixed prediction functionals“

The first-order modelling formalism

Algorithmically,

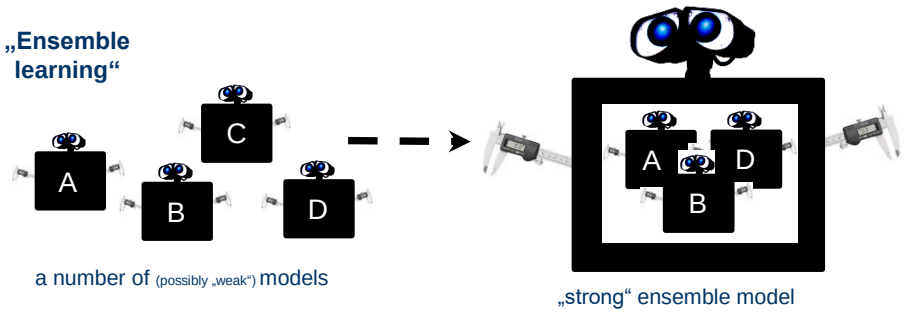
Tuning has the form
of a *„model wrapper“*



$$\text{tune} : \text{model} \times \text{how} \rightarrow \text{model}$$
$$\text{tunedmodelX} := \text{tune}(\text{modelX}, \text{ctrl})$$

„tune function wraps around modelX“

Ensembling has the form
of a *„model aggregator“*



$$\text{ensemble} : \text{listof}(\text{model}) \times \text{how} \rightarrow \text{model}$$
$$\text{ensemblemodel} := \text{ensemble}\left([\text{modelA}, \text{modelB}, \text{modelC}, \text{modelD}], \text{ctrl}\right)$$

„ensemble function wraps around modelA, modelB, modelC, model D“

Strategies that „eat“ models and „produce“ models = *first-order strategies*

Mirrored in toolbox-implementations as explicit/implicit first-order types

Meta-modelling with mlr

Extended tutorial:

http://mlr-org.github.io/mlr-tutorial/release/html/index.html

First-order meta-modelling with mlr

Tuning and ensembling are function wrappers around **Learner** objects

0. Prepare the **learner** which should be wrapped

treelearner <- makeLearner("classif.ctree", id = "ctree")

(or do 0. and 1. in a single go)

1. Apply the respective wrapper, e.g.,

baggedtreelearner <- makeBaggingWrapper(treelearner, etc)

for bagging of **treelearner**

tunededtreelearner <- makeTuneWrapper(treelearner, etc)

for hyper-parameter tuning of **treelearner**

all wrappers need parameters specified and change hyper-parameters

for bagging: specify bagging probabilities and re-sampling

for tuning: specify tuning strategy and hyper-parameter range

mlr abstracts/encapsulates parameter range via **makeParamSet**

2. Use the wrapped learner as you would use any other learner

Current, persistent bug: you cannot wrap a wrapper

Meta-modelling with scikit-learn

Extended tutorial:

http://scikit-learn.org/stable/tutorial/basic/tutorial.html

First-order meta-modelling with sklearn

Tuning and ensembling are descendants of **BaseEstimator** class

Wrapping is done through the constructor of first-order strategy classes

0. Prepare the **Estimator** which should be wrapped

est_DT = ensemble.DecisionTreeRegressor() (or do 0. and 1. in a single go)

1. Initialize the first-order estimator with an estimator as argument

est_DT_tuned = GridSearchCV(etc, estimator = est_DT, etc)

in **model_selection** module, for hyper-parameter tuning of **est_DT**

est_DT_tuned = BaggingRegressor(base_estimator = est_DT, etc)

in **ensemble** module, for bagging of **est_DT**

all wrappers need parameters specified and change hyper-parameters

for bagging: specify bagging probabilities and re-sampling

for tuning: specify tuning strategy and hyper-parameter range

2. Use the wrapped estimator as you would use any other estimator

in sklearn, you can wrap wrappers just fine, because object orientation!