AutoML: Evaluation

Overview and Motivation

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

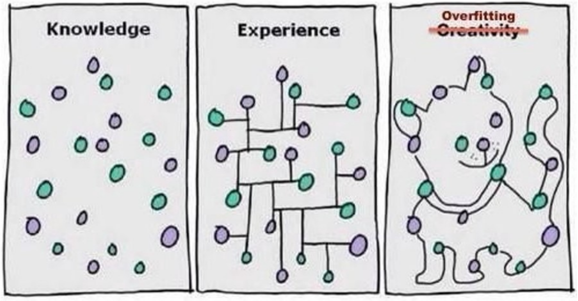
Training Machine Learning Models

- fundamentally an optimization problem
- determine model parameters such that loss on data is minimized
- quality of fit depends on model class (i.e. degrees of freedom)

Which model is best?

Generalization

- we want models that *generalize* – make "reasonable" predictions on new data
  - ignore outliers
  - smooth
  - captures general trend



Usually model performance gets better with more data/higher model complexity and then worse, but see [Nakkiran et al. 2019]

Overview

- evaluating machine learning models and quantifying generalization performance
- learning curves
- comparing multiple models/learners on multiple data sets
- statistical tests
- higher levels of optimization, higher levels of evaluation
  - automated machine learning (meta-optimization) can lead to meta-overfitting
  - simple training/testing split(s) no longer sufficient → nested evaluation

Evaluate Early, Evaluate Often

https://www.autodeskresearch.com/publications/samestats

AutoML: Evaluation

Evaluation of ML Models (Review)

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Introduction

Performance estimation of a model Estimate generalization error of a model on new (unseen) data, drawn from the same data generating process.

Performance estimation of an algorithm Estimate generalization error of a learning algorithm, trained on a data set of a certain size, on new (unseen) data, all drawn from the same data generating process.

Model selection Select the best model from a set of potential candidate models (e.g., different model classes, different hyperparameter settings, different feature sets).

Performance Evaluation

ML performance evaluation provides clear and simple protocols for reliable model validation.

- often simpler than classical statistical model diagnosis
- relies only on few assumptions
- still hard enough and offers **lots** of options to cheat and make mistakes

Performance Measures

We measure performance using a statistical estimator for the **generalization error** (GE).
GE $=$ expected loss of a fixed model
$\hat{\text{GE}}$ $=$ estimated loss averaged across finite sample
Example: Mean squared error (L2 loss)

$$\hat{\text{GE}} = MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$$

Measures: Inner vs. Outer Loss

Inner loss = loss used in learning (training)
Outer loss = loss used in evaluation (testing)
= evaluation measure

Optimally: inner loss = outer loss

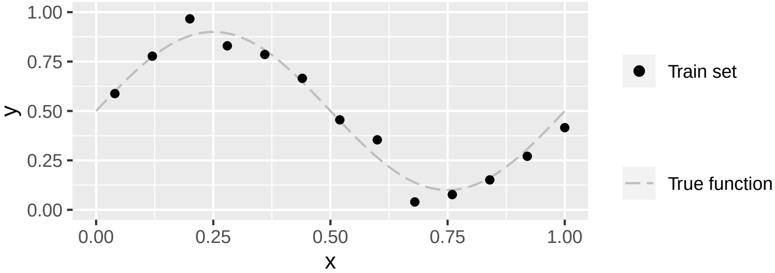
Not always possible:
some losses are hard to optimize or no loss is specified directly

Example:

Logistic Regression	→ minimize binomial loss
kNN	→ no explicit loss minimization

Inner Loss Example: Polynomial Regression I

Sample data from sinusoidal function $0.5 + 0.4 \cdot \sin(2\pi x) + \epsilon$ with measurement error ϵ.

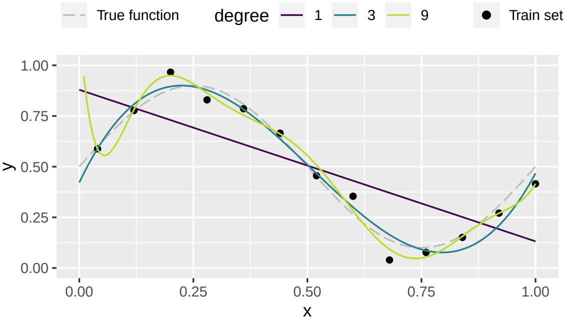


Assume data generating process unknown. Approximate with dth-degree polynomial:

$$f(\mathbf{x}|\theta) = \theta_0 + \theta_1 x + \cdots + \theta_d x^d = \sum_{j=0}^{d} \theta_j x^j$$

Inner Loss Example: Polynomial Regression II

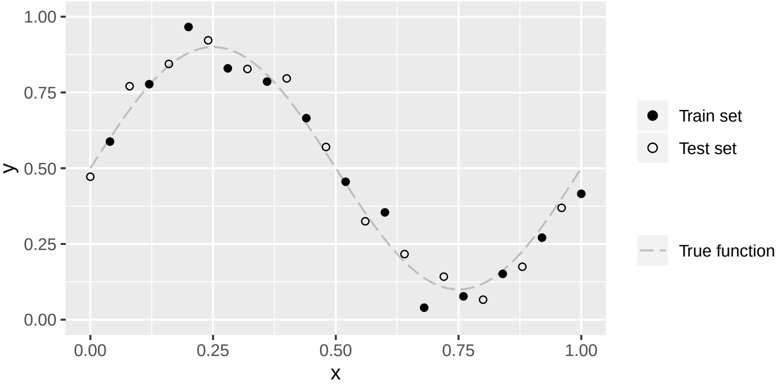
How should we choose d?



d=1: MSE = 0.036 – clear underfitting, d=3: MSE = 0.003 – ok?, d=9: MSE = 0.001 – clear overfitting

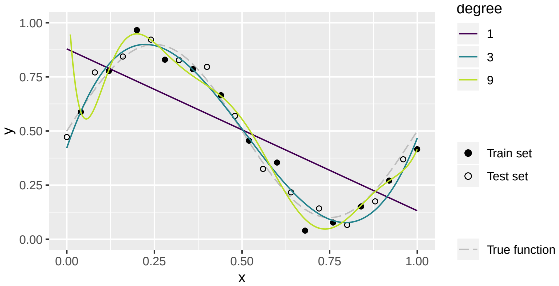
Simply using the training error seems to be a bad idea.

Outer Loss Example: Polynomial Regression I



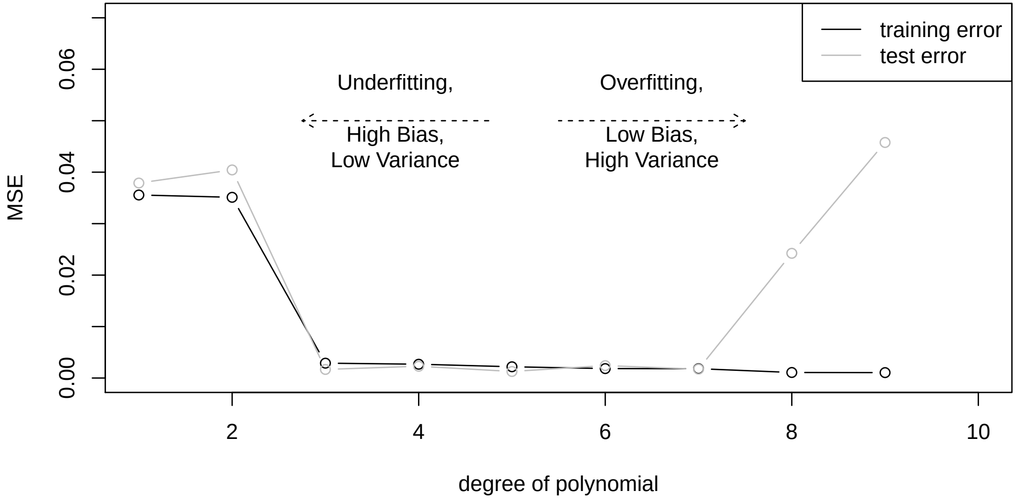
Outer Loss Example: Polynomial Regression II

How should we choose d?

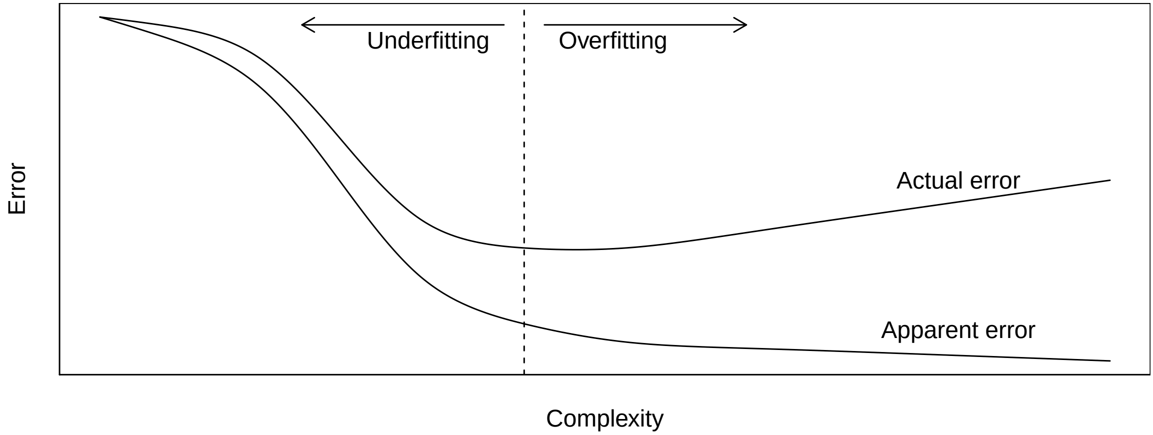


d=1: MSE = 0.038 – clear underfitting, d=3: MSE = 0.002 – ok?, d=9: MSE = 0.046 – clear overfitting

Outer Loss Example: Polynomial Regression III



General Trade-Off Between Error and Complexity



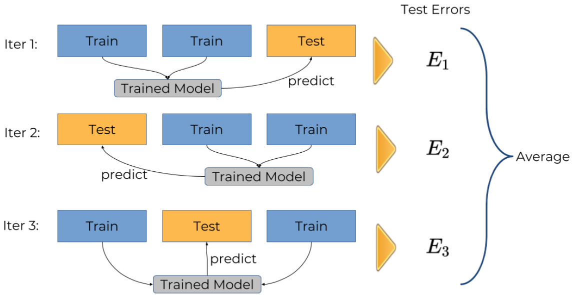
Resampling

- uses data efficiently
- repeatedly split in train and test, average results
- make training sets large (to keep the pessimistic bias small), reduce variance introduced by smaller test sets through many repetitions and averaging of results

Cross-Validation

- split data into k roughly equally-sized partitions
- use each part as test set and join the $k-1$ others for training, repeat for all k combinations
- obtain k test errors and average

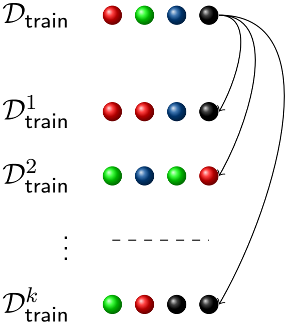
Example 3-fold cross-validation:



10-fold cross-validation is common.

Bootstrap

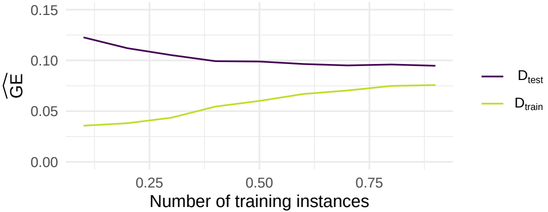
- randomly draw k training sets of size n with replacement from the data
- evaluate on observations from the original data that are not in the training set
- obtain k test errors and average


$\mathcal{D}_{\text{train}}$
$\mathcal{D}^1_{\text{train}}$
$\mathcal{D}^2_{\text{train}}$
$\vdots$
$\mathcal{D}^k_{\text{train}}$

Training sets will contain about 63.2% of observations on average.

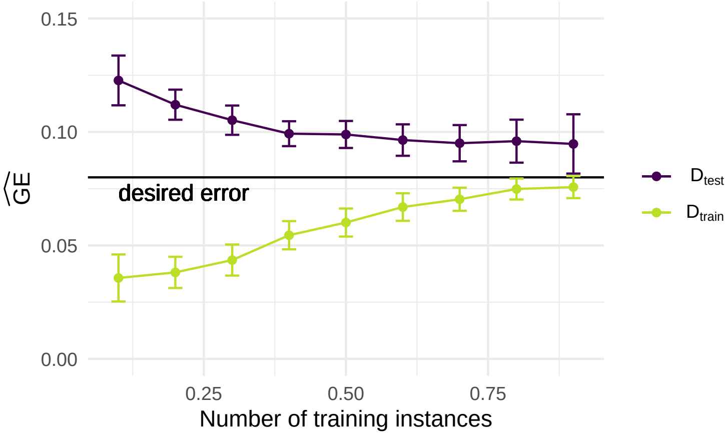
Learning Curves I

- compares performance of a model on training and test data over a varying number of training instances → how fast can a learner learn the given relationship in the data?
- can also be over number of iterations of a learner (e.g. epochs in deep learning), or AutoML system over time
- learning usually fast in the beginning
- visualizes when a learner has learned as much as it can:
  - when performance on training and test set reach a plateau
  - when gap between training and test error remains the same



Learning Curves II

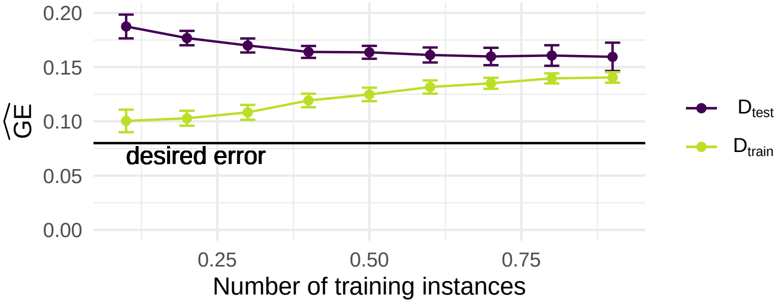
Ideal learning curve:



Learning Curves III

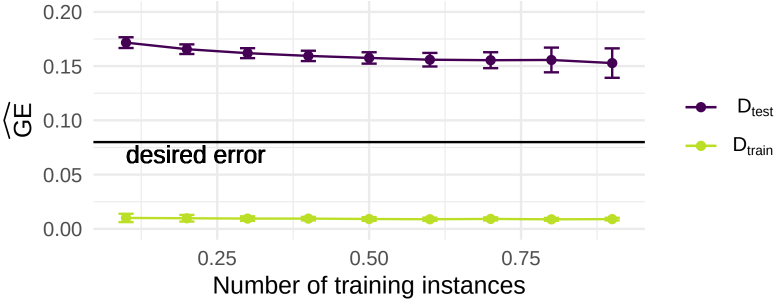
In general, there are two reasons for a bad learning curve:

1. high bias in model/underfitting
    - training and test errors converge at a high value
    - model can't learn underlying relationship and has high systematic errors, no matter how big the training set
    - poor fit, which also translates to high test error



Learning Curves IV

2. high variance in model/overfitting
    - large gap between training and test errors
    - model requires more training data to improve
    - model has a poor fit and does not generalize well



AutoML: Evaluation

Benchmarking and Comparing Learners

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Benchmark Experiments

- different learning algorithms applied to one or more data sets to compare and rank their performances
- synchronized train and test sets, i.e. the same resampling method with the same train-test splits should be used to determine performance

  Example: Benchmark results (per CV-fold) of CART and random forest using 2-fold CV with MSE as performance measure:

data set	k-th fold	MSE (rpart)	MSE (randomForest)
BostonHousing	1	29.4	17.13
BostonHousing	2	20.5	8.90
mtcars	1	35.0	7.53
mtcars	2	38.9	6.73

Hypothesis Testing in Benchmarking I

We want to know if the difference in performance between models (or algorithms) is significant or only by chance.

Null Hypothesis Statistical Testing (NHST) in Benchmarking:

- formulate null hypothesis H_0 (e.g. the expected generalization error of two algorithms is equivalent) and alternative hypothesis H_1
- use hypothesis test to reject H_0 (not rejecting H_0 does not mean that we accept it)
- rejecting H_0 gives some confidence that the observed results may not be random

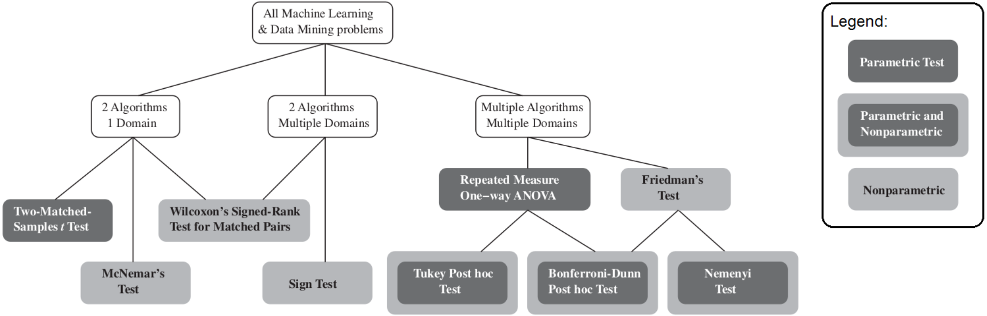
Typical example in machine learning:

- H_0: on average, model 1 does not perform better than model 2
- H_1: on average, model 1 outperforms model 2
- Aim: Reject H_0 with confidence level of $1 - \alpha$ (common values for α include 0.05 and 0.01)

Hypothesis Testing in Benchmarking II

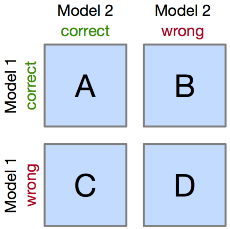
Selection of an appropriate hypothesis test is at least based on the type of problem, i.e. if the aim is to compare

- 2 models / algorithms on a single domain (i.e. on a single data set)
- 2 algorithms across different domains (i.e. on multiple data sets)
- multiple algorithms across different domains / data sets


All Machine Learning & Data Mining problems
2 Algorithms 1 Domain
2 Algorithms Multiple Domains
Multiple Algorithms Multiple Domains
Two-Matched-Samples *t* Test
McNemar's Test
Wilcoxon's Signed-Rank Test for Matched Pairs
Sign Test
Repeated Measure One–way ANOVA
Friedman's Test
Tukey Post hoc Test
Bonferroni-Dunn Post hoc Test
Nemenyi Test
Legend:
Parametric Test
Parametric and Nonparametric
Nonparametric

McNemar Test I

- non-parametric test used on paired dichotomous nominal data; does not make any distributional assumptions beyond statistical independence of samples
- pairs are e.g. labels predicted by different models on the same data
- compares the classification accuracy of two **models**
- both models trained and evaluated on the exact same training and test set; **contingency table** based on two paired vectors that indicate whether each model predicted an observation correctly



	Model 2 correct	Model 2 wrong
Model 1 correct	A	B
Model 1 wrong	C	D

This work by Sebastian Raschka is licensed under a Creative Commons Attribution 4.0 International License.

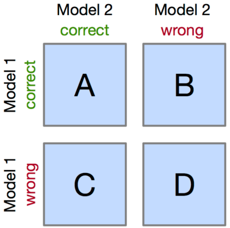
- A: #obs. correctly classified by both
- B: #obs. only correctly classified by model 1
- C: #obs. only correctly classified by model 2
- D: #obs. misclassified by both

McNemar Test II

Error of each model can be computed as follows:

- Model 1: (C+D)/(A+B+C+D)
- Model 2: (B+D)/(A+B+C+D)

Even if the models have the **same** errors (indicating equal performance), cells B and C may be different because the models may misclassify different instances.



	Model 2 correct	Model 2 wrong
Model 1 correct	A	B
Model 1 wrong	C	D

This work by Sebastian Raschka is licensed under a Creative Commons Attribution 4.0 International License.

McNemar tests the following hypothesis:

- H_0 : both models have the same performance (we expect B $=$ C)
- H_1 : performances of the two models are not equal

The test statistic is computed as

$$\chi^2_{Mc} = \frac{(|B-C|-1)^2}{B+C} \sim \chi^2_1.$$

Note: The McNemar test should only be used if $B + C > 20$.

McNemar Test III

Example:

		Random Forest	
		0	1
Tree	0	30	5
	1	17	42

Calculating the test statistic:

$$\chi^2_{Mc} = \frac{(|5 - 17| - 1)^2}{5 + 17} = 5.5 > 3.841 = \chi^2_{1,0.95}$$

We can reject H_0 at a significance level of 0.05, i.e. we reject the hypothesis that the tree and the random forest have the same performance.
Significance level must be chosen before applying the test (avoid p-value hacking).

Two-Matched-Samples t-Test I

- two-matched-samples t-test (i.e. a paired t-test) is the simplest hypothesis test to compare two **models** on a single test set based on arbitrary performance measures
- parametric test and distributional assumptions must be made (which are often problematic):

  (pseudo-)normality usually met when sample size > 30

  i.i.d. samples usually met if loss of individual observations from single test set considered

  equal variances of populations can be investigated through plots

Two-Matched-Samples t-Test II

Compare two different models $\hat{f}_1$ and $\hat{f}_2$ w.r.t. performance measure calculated on test set of size n_{test}:

- H_0: $GE(\hat{f}_1) = GE(\hat{f}_2)$ vs. H_1: $GE(\hat{f}_1) \neq GE(\hat{f}_2)$
- test statistic $T = \sqrt{n_{\text{test}}}\frac{\bar{d}}{\sigma_d}$ where
  - mean performance difference of both models is $\bar{d} = \hat{GE}_{\mathcal{D}_{\text{test}}}(\hat{f}_1) - \hat{GE}_{\mathcal{D}_{\text{test}}}(\hat{f}_2)$
  - standard deviation of this mean difference is

$$\sigma_d = \sqrt{\frac{1}{n_{\text{test}} - 1}\sum_{i=1}^{n_{\text{test}}}\left(d_i - \bar{d}\right)^2},$$

where $d_i = L(y^{(i)}, \hat{f}_1(\mathbf{x}^{(i)})) - L(y^{(i)}, \hat{f}_2(\mathbf{x}^{(i)}))$ and $\bar{d} = \frac{1}{n_{\text{test}}}\sum_{i=1}^{n_{\text{test}}} d_i$

Note: d_i is the difference of the outer loss of individual observations from the test set between the two models to be compared.

Two-Matched-Samples t-Test III

- could also use a k-**fold CV paired t-test** to compare two **algorithms** (instead of two models) on single data set
- instead of comparing outer loss of individual observations, compare generalization errors per CV fold (i.e. k generalization errors for k CV folds)
- performance differences are not independent across CV folds due to overlapping training sets (which violates the assumption of i.i.d. samples)
- to partly overcome issue of overlapping training sets across folds, Dietterich suggests using 5 times 2-fold CV so that at least within each repetition neither training nor test sets overlap [Dietterich, 1998]

Friedman Test I

Compare multiple classifiers on multiple data sets:

- H_0 : all algorithms are equivalent in their performance and hence their average ranks should be equal
- H_1 : the average ranks for at least one algorithm is different

To evaluate n data sets and k algorithms:

- rank each algorithm on each data set from best-performing algorithm (rank 1) to worst-performing algorithm using any performance measure
- R_{ij} is the rank of algorithm j on data set i
- if there is a d-way tie after rank r, assign rank of $\left[(r+1)+(r+2)+...+(r+d)\right]/d$ to each tied classifier

Friedman Test II

Can now compute:

- overall mean rank $\bar{R} = \frac{1}{nk} \sum_{i=1}^{n} \sum_{j=1}^{k} R_{ij}$
- sum of squares total $SS_{Total} = n \sum_{j=1}^{k} (\bar{R}_{.j} - \bar{R})^2$ where $\bar{R}_{.j} = \frac{1}{n} \sum_{i=1}^{n} R_{ij}$
- sum of squares error $SS_{Error} = \frac{1}{n(k-1)} \sum_{i=1}^{n} \sum_{j=1}^{k} (R_{ij} - \bar{R})^2$

Test statistic calculated as:

$$ {\chi_F}^2 = \frac{SS_{Total}}{SS_{Error}} \sim \chi^2_{k-1} \text{ for large n (>15) and k (>5)} $$

For smaller n and k, the χ^2 approximation is imprecise and a look up of χ^2_F values that were approximated specifically for the Friedman test is suggested.

Post-Hoc Tests I

- Friedman test checks if all algorithms are ranked equally
- does not allow to identify best-performing algorithm

→ post-hoc tests

Post-hoc Nemenyi test:

- compares all pairs of algorithms to find best-performing algorithm after H_0 of the Friedman-test was rejected
- for n data sets and k algorithms, $\frac{k(k-1)}{2}$ comparisons
- calculate average rank of algorithm j on all n data sets: $\bar{R}_{.j} = \frac{1}{n} \sum_{i=1}^{n} R_{ij}$

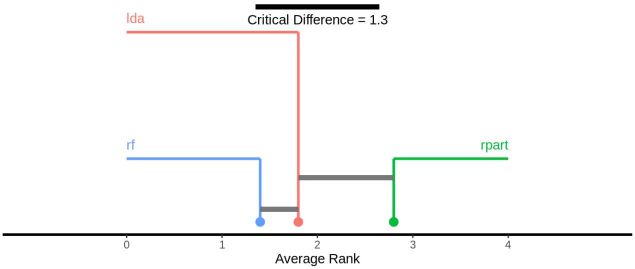
For any two algorithms j_1 and j_2, test statistic computed as:

$$q = \frac{\bar{R}_{.j_1} - \bar{R}_{.j_2}}{\sqrt{\frac{k(k+1)}{6n}}}$$

Post-Hoc Tests II

Critical Difference Plot:

- quick way to see what differences are significant across all compared learners
- all learners that do not differ by at least the critical difference are connected by line
- a learner not connected to another learner and of lower rank can be considered better according to the chosen significance level



Post-Hoc Tests III

Post-hoc Bonferonni-Dunn test:

- compares all algorithms with baseline (i.e. $k-1$ comparisons)
- used after Friedman test to find which algorithms differ from the baseline significantly
- uses Bonferonni correction to prevent randomly accepting one of the algorithms as significant due to multiple testing

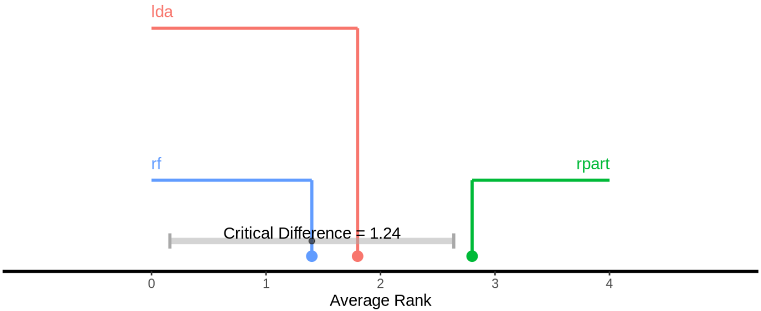
The test statistic is the same as before:

$$q = \frac{\bar{R}_{\cdot j_1} - \bar{R}_{\cdot j_2}}{\sqrt{\frac{k(k+1)}{6n}}}.$$

The performance of j_1 and j_2 are significantly different when $|q| > q_\alpha$, where the critical value q_α is obtained from a table of the studentized range statistic divided by $\sqrt{2}$.

Post-Hoc Tests IV

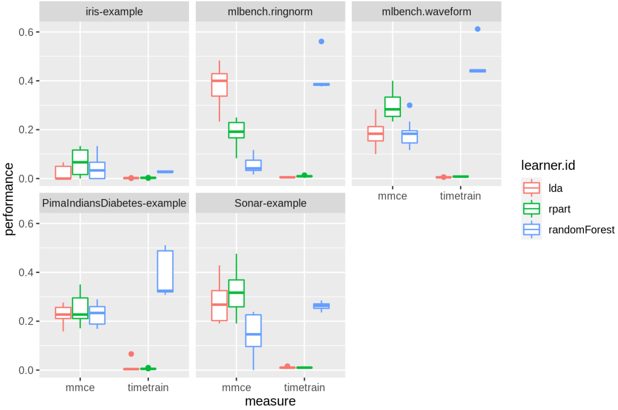
- learners within the baseline interval (gray line) perform not significantly different from the baseline



Comparing Visually I

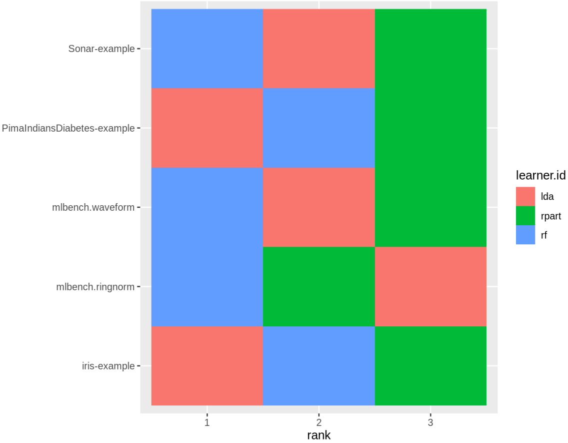
It can be helpful to inspect distributions visually for additional insights, e.g.

Boxplots



Comparing Visually II

Rank plots



AutoML: Evaluation

Nested Resampling

Bernd Bischl[1] Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

[1] Some slides taken from Bernd Bischl's "Introduction to Machine Learning" lecture at LMU. [Bischl]

Motivation

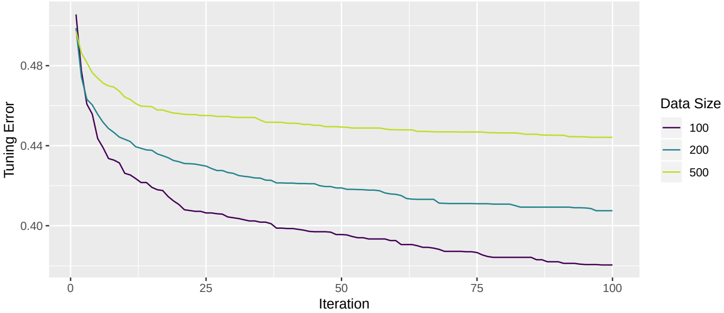
Selecting the best model from a set of potential candidates (e.g. different classes of learners, different hyperparameter settings, different feature sets, different preprocessing. . .) is an important part of most machine learning problems. However,

- cannot evaluate selected learner on the same resampling splits used to select it
- repeatedly evaluating learner on same test set or same CV splits “leaks” information about test set into evaluation
- danger of overfitting to the resampling splits or overtuning
- final performance estimate would be optimistically biased
- similar to multiple hypothesis testing

Motivating Example I

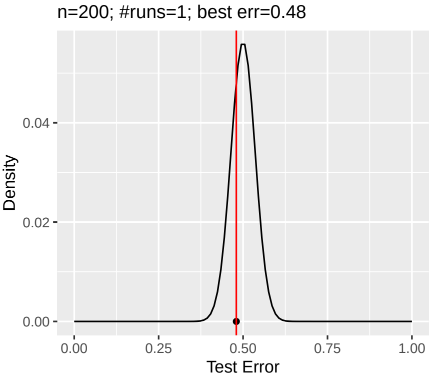
- binary classification problem with equal class sizes
- learner with hyperparameter $\boldsymbol{\lambda}$
- learner is (nonsense) feature-independent classifier that picks random labels with equal probability, $\boldsymbol{\lambda}$ has no effect
- true generalization error is 50%
- cross-validation of learner (with any fixed $\boldsymbol{\lambda}$) will easily show this (if the partitioned data set for CV is not too small)
- let's "tune" it by trying out 100 different $\boldsymbol{\lambda}$ values
- repeat this experiment 50 times and average results

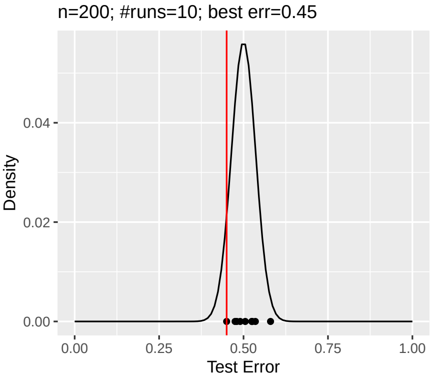
Motivating Example II



- shown is best “tuning error” (i.e. performance of model with fixed $\boldsymbol{\lambda}$ in cross-validation) after k tuning iterations
- evaluated for different data set sizes

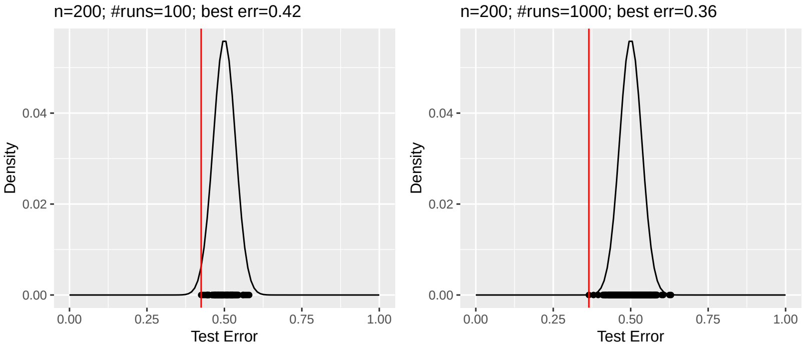
Motivating Example III





- for one experiment, CV score is close to 0.5, as expected
- errors essentially sampled from (rescaled) binomial distribution
- scores from multiple experiments also arranged around expected mean of 0.5

Motivating Example IV



- tuning means we take the minimum of the scores
- not estimate of average performance, but best-case performance
- the more we sample, the more “biased” this value becomes → unrealistic generalization performance estimate

Untouched Test Set Principle I

Instead: simulate what actually happens when applying machine learning models

- all parts of model construction (including model selection, preprocessing) evaluated **on training data**
- test set only touched once, so no way of “cheating”
- test dataset is only used once *after* model is completely trained (including e.g. deciding hyperparameter values)
- performance estimates from test set now **unbiased estimates** of the true performance

Untouched Test Set Principle II

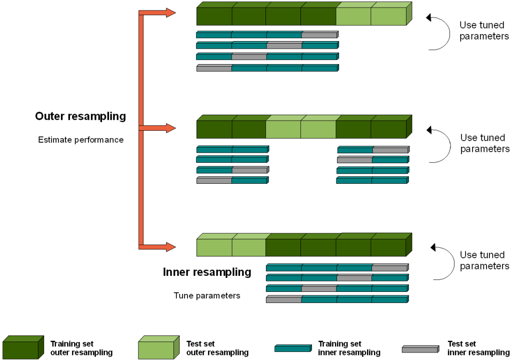
- for steps that themselves require resampling (e.g. hyperparameter tuning) this results in **nested resampling**, i.e. resampling strategies for both
  - inner evaluation to find what works best based on training data
  - outer evaluation on data not used in inner evaluation to get unbiased estimates of expected performance on new data

Nested Resampling I

- holdout can be generalized to resampling for more reliable generalization performance estimates
- resampling can be generalized to nested resampling
- nested resampling loops for inner and outer evaluation for hyperparameter tuning

Nested Resampling II

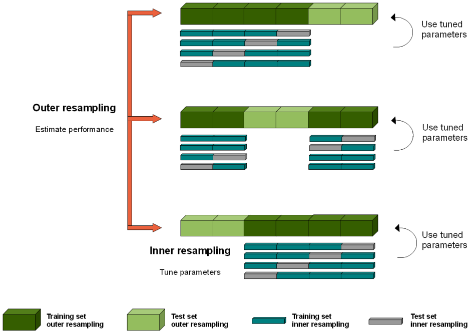
Example: four-fold CV for inner resampling, three-fold CV in outer resampling



Nested Resampling III

In each iteration of the outer loop:

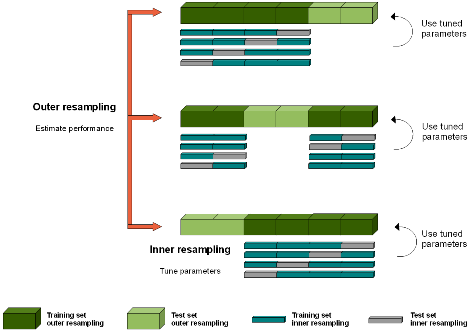
- split light green testing data
- run hyperparameter tuner on dark green part of data, i.e. evaluate each $\boldsymbol{\lambda}_i$ through four-fold CV on dark green part



Nested Resampling IV

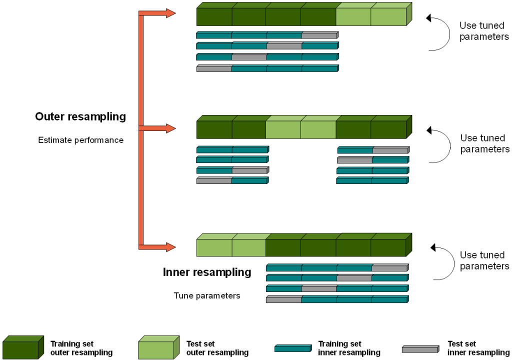
In each iteration of the outer loop:

- return winning $\hat{\boldsymbol{\lambda}}$ that performed best on the grey inner test sets
- re-train model on full outer dark green training set
- evaluate model on outer light green test set



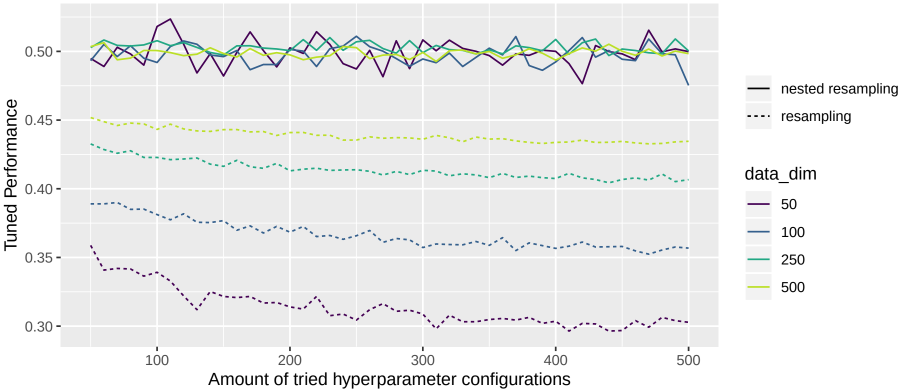
Nested Resampling V

→ error estimates on outer samples (light green) are unbiased because this data was not used process constructing the tested model



Nested Resampling Example

Revisited motivating example: expected performance estimate with nested resampling:


Tuned Performance
Amount of tried hyperparameter configurations
nested resampling
resampling
data_dim