Hyperparameter Optimieren mit AutoML

Marius Lindauer

Leibniz Universität Hannover
Germany

m.lindauer@ai.uni-hannover.de

Katharina Eggensperger

Eberhard Karls Universität Tübingen
Germany

katharina.eggensperger@uni-tuebingen.de

Leibniz Universität Hannover

LUH|AI

AutoML.org

Eberhard Karls Universität Tübingen

machine learning
new perspectives
for science

Marius Lindauer

Professor an der
Leibniz Universität Hannover

Leiter des Instituts für KI (LUH|AI) &
Leiter der Gruppe für maschinelles Lernen

Sieger in der 1. und 2. internationalen AutoML Challenge; ERC Starting Grant für ixAutoML

Co-Autor von bekannten AutoML tools wie Auto-sklearn, Auto-PyTorch, SMAC3

Co-Leiter von automl.org

Co-Gründer und Advisory Board Mitglied des Forschungsnetzwerkes COSEAL (Configuration and Selection of Algorithms)

Katharina Eggensperger

Doktor in AutoML von der Uni Freiburg

Early Career Forschungsgruppen-Leiterin an der **Universität Tübingen** für **AutoML for Science**

Siegerin in der 1. und 2. internationalen AutoML Challenge; (this was a team effort!)

Core-Entwicklerin von populären AutoML tools: **Auto-sklearn** and **SMAC**

Junior-Leiterin von automl.org

The Big Picture

>> Worum geht es eigentlich?

Maschinelles Lernen ist …

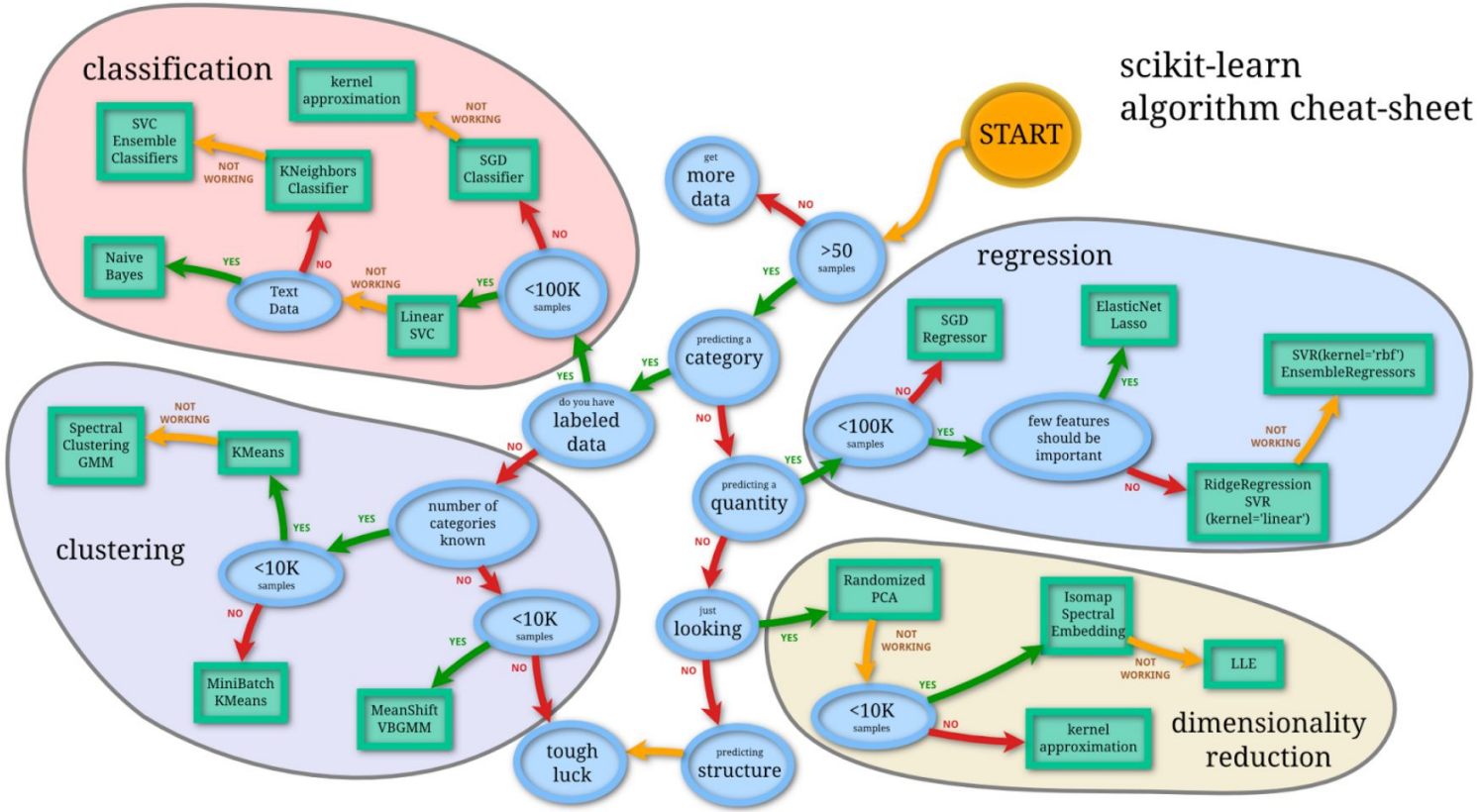
“Machine learning is the science of getting computers to act without being explicitly programmed.”

von Andrew Ng
(wahrscheinlich inspiriert von Arthur Samuels)

… aber auch das hier

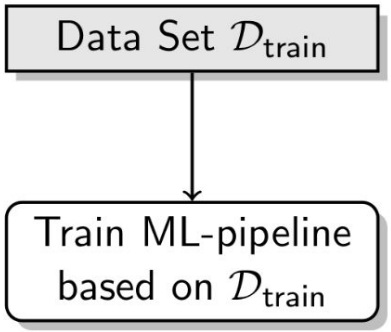
source: XKDC

Design-Entscheidungen

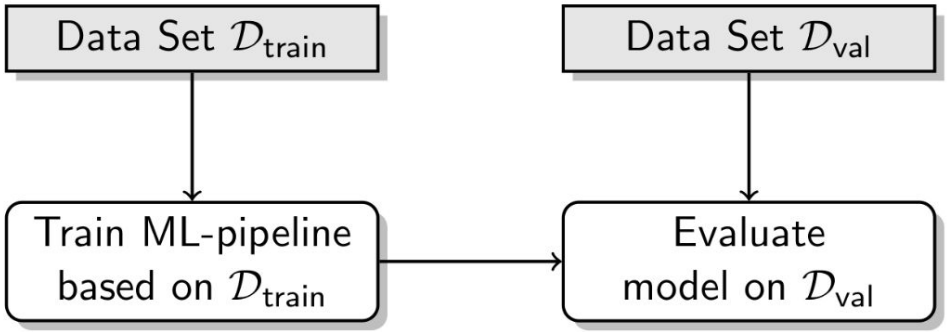


source: https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

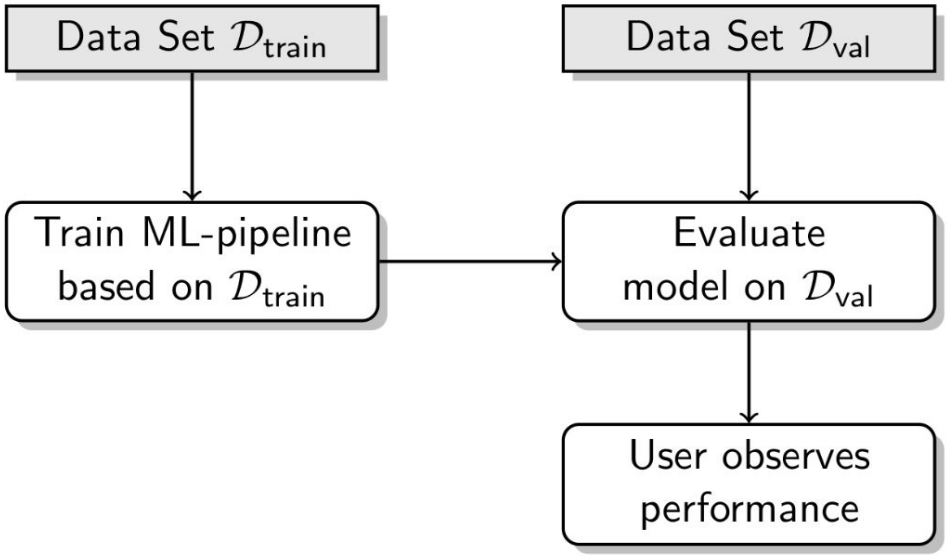
Manueller ML-Workflow



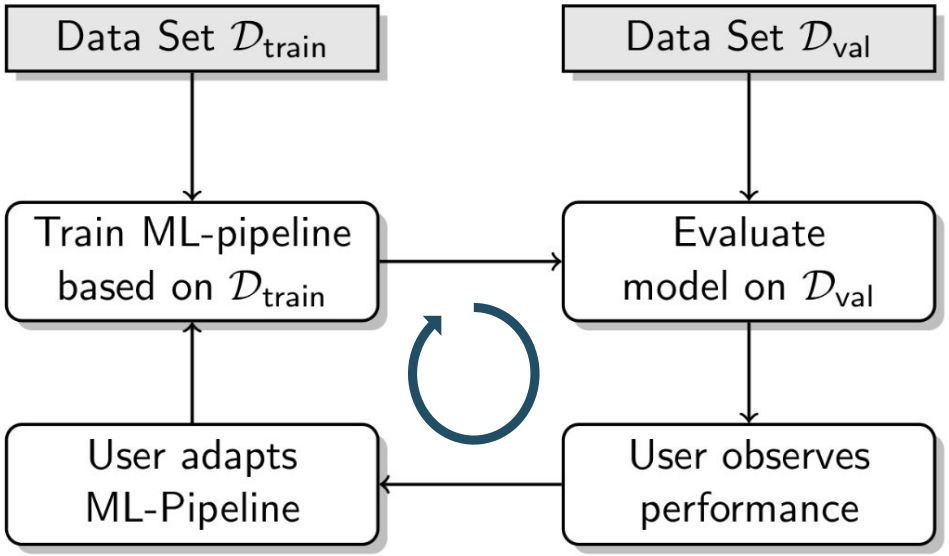
Manual Machine Learning Workflow



Manual Machine Learning Workflow



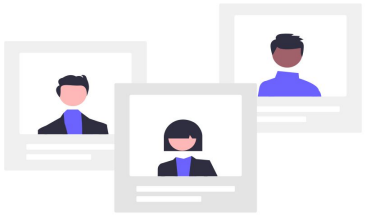
Manual Machine Learning Workflow



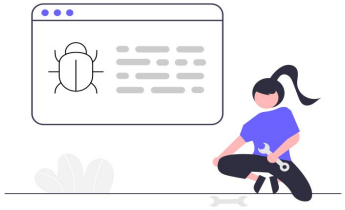
Herausforderungen für neue KI/ML Projekte

Notwendige Expertise in KI & ML

Lange Entwicklungszeiten von neuen KI Anwendungen

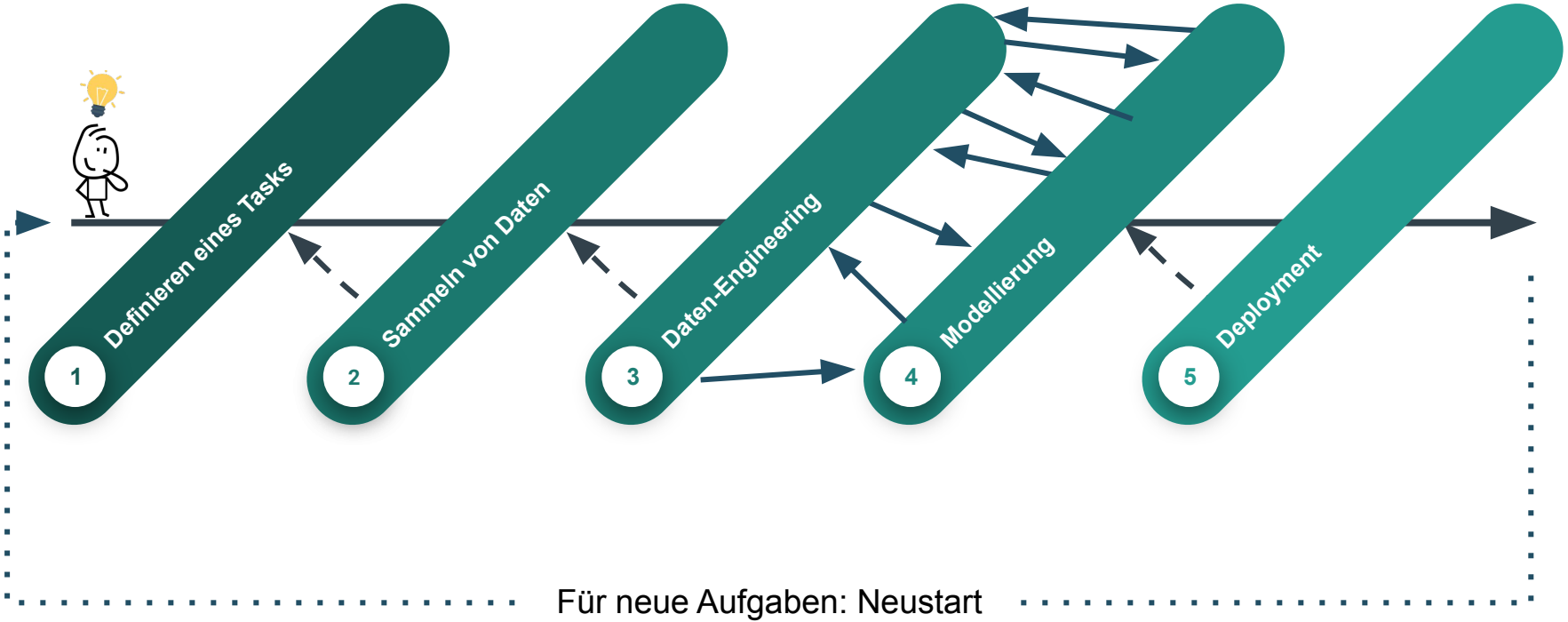


Fachkräftemangel



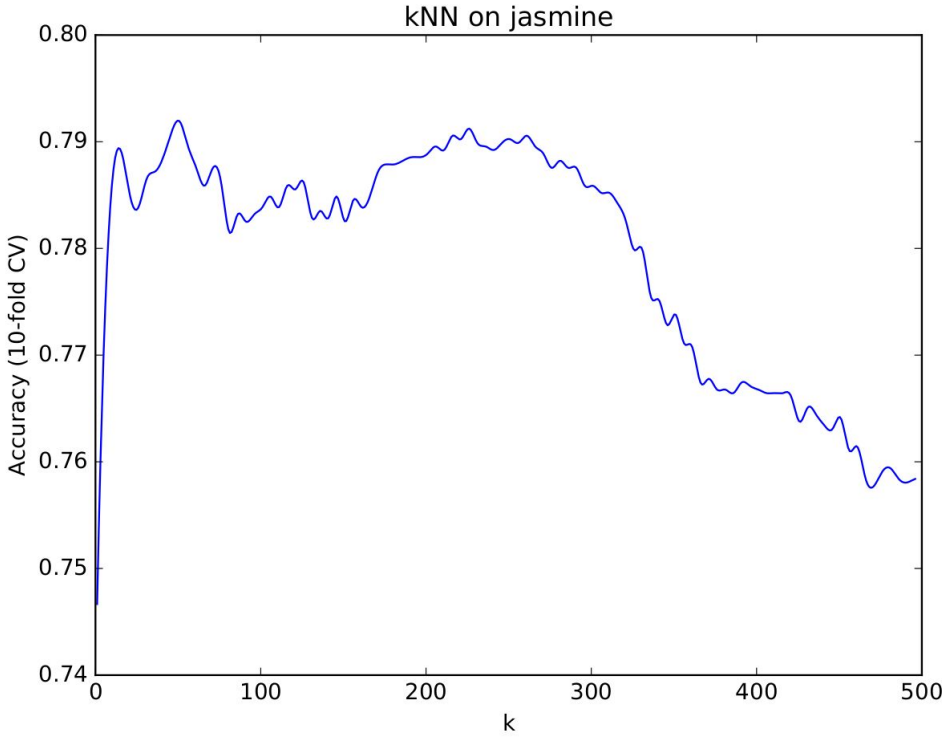
Unstrukturierte & fehleranfällige Entwicklung von KI Anwendungen

Warum braucht ML-Entwicklung so viel Zeit?

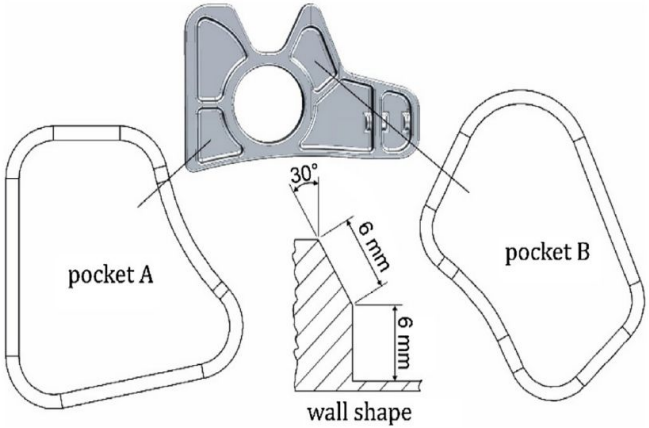


Simples Beispiel: kNN

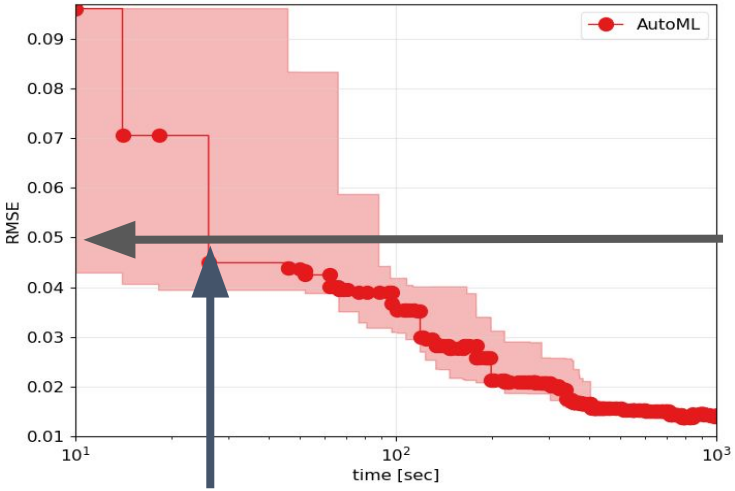
- *k*-nearest neighbors (kNN) ist einer der einfachsten ML-Algorithmen
- Größe der Nachbarschaft (*k*) ist sehr wichtig für die Performanz
- Die Performanz-Funktion in Abhängigkeit von *k* ist ganz schön komplex



Motivierendes Beispiel: ML für Fräsvorgänge



[Denkena et al. 2020]



State of the Art eines Doktoranden

Bessere Performanz nach ~30sec (+ ein bisschen Zeit zum Einlesen der Daten)

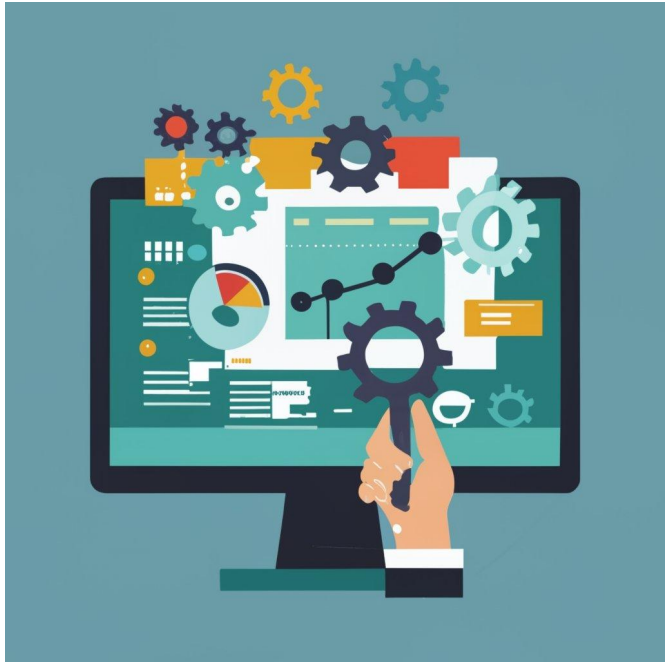
Wobei kann AutoML alles helfen?

Hyperparameter

Auswahl von Methoden zur Dimensionsreduktion

Trainingsschedules

Design ganzer Datenverarbeitungspipelines



Auswahl von Architekturen von neuronalen Netzen

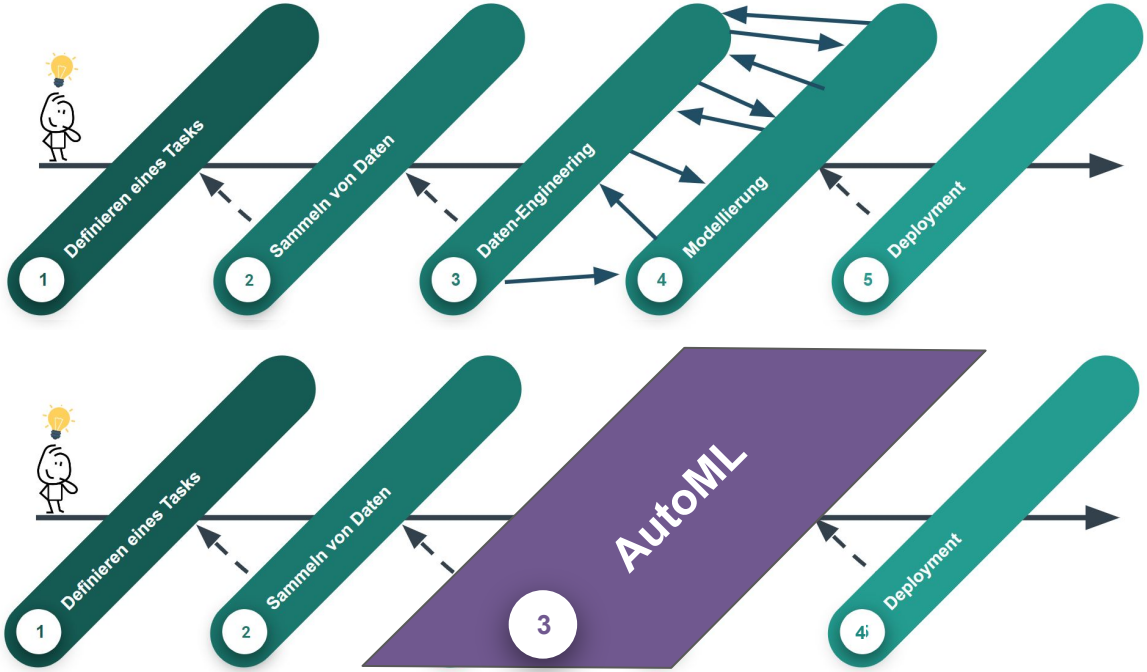
Feature-Engineering

Postprocessing (Ensembling / Stacking)

Auswahl von Modellklassen (XGBoost vs. DNNs)

...

ML vs AutoML


Definieren eines Tasks
Sammeln von Daten
Daten-Engineering
Modellierung
Deployment
1
2
3
4
5
Definieren eines Tasks
Sammeln von Daten
AutoML
Deployment
1
2
3
4

Vorteile

AutoML ermöglicht

- 🚀 **Effizientere** Entwicklung von neuen ML Anwendungen
  - → AutoML hat gezeigt, dass es besser als händische Entwickler:Innen sein kann
- 🧮 **Systematischere** Entwicklung von neuen ML Anwendungen
  - → kein menschlicher Bias or unsystematische Evaluation
- 📋 Bessere **Reproduzierbarkeit**
  - → es ist ja systematisch ;-)
- 🦾 **Breitere Anwendung von ML Methoden**
  - → weniger benötigte ML-Expertise
  - → nicht nur auf Informatiker:Innen begrenzt

Herausforderungen

Aber AutoML ist nicht so einfach, weil

🔁 Jeder Datensatz braucht potentiell ein anderes ML-Design

→ Design-Entscheidungen müssen für jeden Datensatz neu getroffen werden

⌛ Trainieren eines einzelnen ML Models kann schon sehr teuer sein

→ Wir können nicht viele Designs durch probieren

❓Mathematische Zusammenhänge zwischen Design und Performanz sind zumeist unbekannt

→ Gradienten-basierte Optimierung ist nicht einfach möglich

🎛️🎚️ Optimierung in hoch-dimensionalen Suchräumen

→ inkl. kategorische, kontinuierliche und konditionale Dimensionen

Risiken

1. **Blindes Vertrauen**
   → Nutzer:Innen wundern sich, warum AutoML nicht performt, nach dem schlechte Daten verwendet wurden.
2. **Automatisierungs-Bias**.
   → Nutzer:Innen hinterfragen Entscheidungen von Maschinen nicht
3. Nicht-Expert:Innen nutzen ML **ohne sich der Risiken und Konsequenzen bewußt** zu sein.
   → Z.B. Bias in Daten und trainierten Modellen

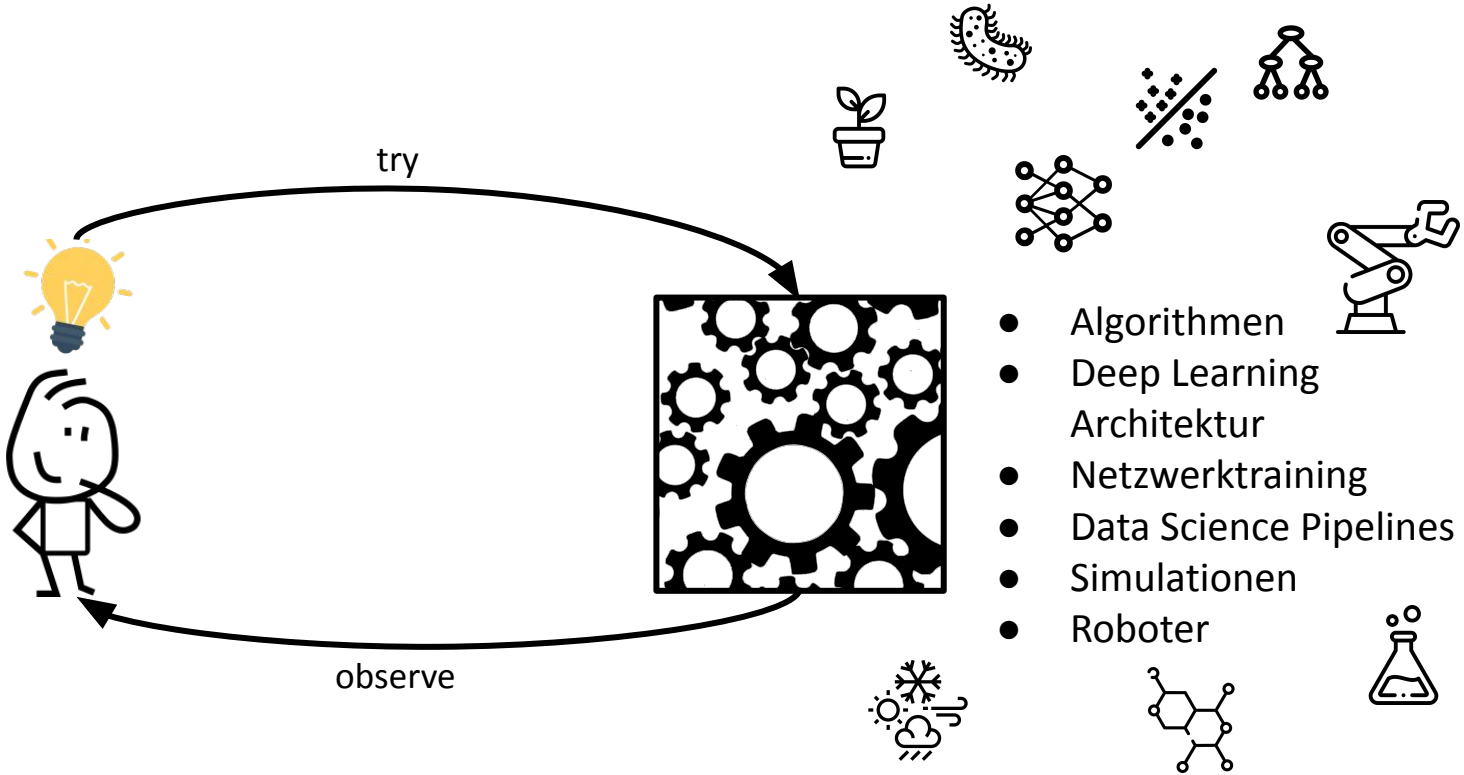
Das führt zu:

- **Ungenauen Vorhersagen wegen fehlenden Verständnis von statistischen Konzepten**
- **Gebiased und unfairen Modeln** wegen fehlenden Verständnis von ethischen Konzepten
  - siehe auch Diskussion ob Fairness nicht automatisiert werden kann [Weerts et al. 2023]

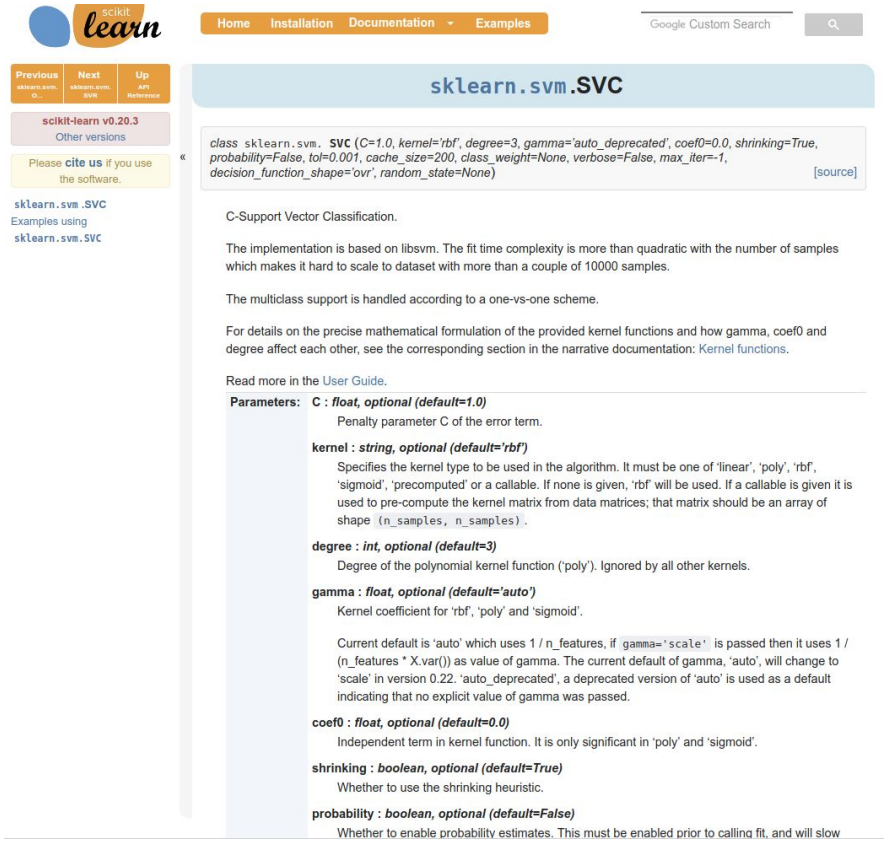
Was optimieren wir?

```
>> Hier sind mein Modell und Daten, was kann ich tun?
```

Sequentielles Experimentieren



Hyperparameter



scikit learn

Home Installation Documentation Examples Google Custom Search

Previous Next Up

scikit-learn v0.20.3
Other versions

Please cite us if you use the software.

sklearn.svm.SVC
Examples using sklearn.svm.SVC

sklearn.svm.SVC

class sklearn.svm. **SVC** (*C=1.0, kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None*) [source]

C-Support Vector Classification.

The implementation is based on libsvm. The fit time complexity is more than quadratic with the number of samples which makes it hard to scale to dataset with more than a couple of 10000 samples.

The multiclass support is handled according to a one-vs-one scheme.

For details on the precise mathematical formulation of the provided kernel functions and how gamma, coef0 and degree affect each other, see the corresponding section in the narrative documentation: Kernel functions.

Read more in the User Guide.

Parameters:

C : *float, optional (default=1.0)*
Penalty parameter C of the error term.

kernel : *string, optional (default='rbf')*
Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`.

degree : *int, optional (default=3)*
Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

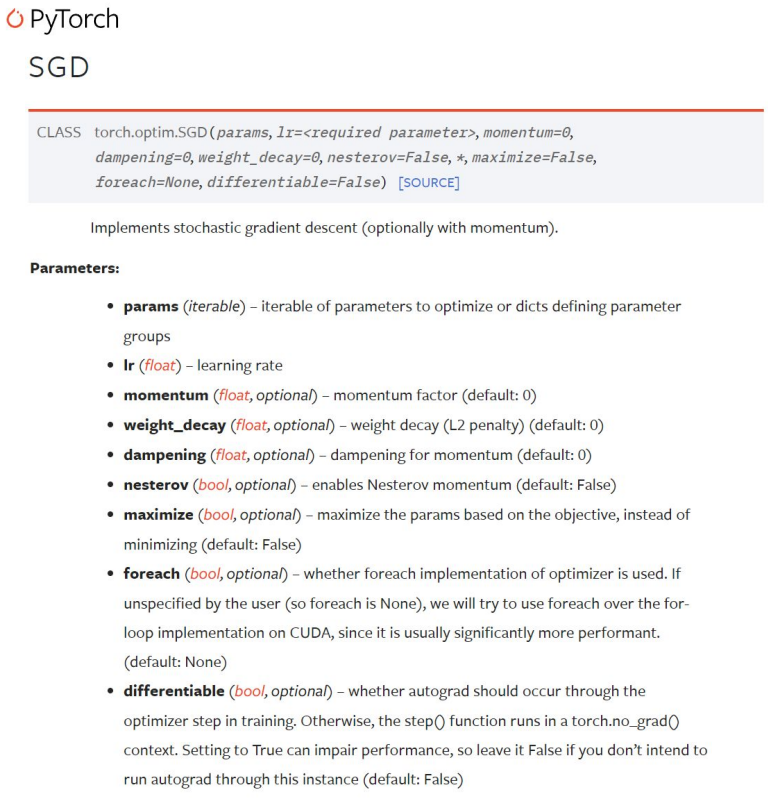
gamma : *float, optional (default='auto')*
Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

Current default is 'auto' which uses 1 / n_features, if `gamma='scale'` is passed then it uses 1 / (n_features * X.var()) as value of gamma. The current default of gamma, 'auto', will change to 'scale' in version 0.22. 'auto_deprecated', a deprecated version of 'auto' is used as a default indicating that no explicit value of gamma was passed.

coef0 : *float, optional (default=0.0)*
Independent term in kernel function. It is only significant in 'poly' and 'sigmoid'.

shrinking : *boolean, optional (default=True)*
Whether to use the shrinking heuristic.

probability : *boolean, optional (default=False)*
Whether to enable probability estimates. This must be enabled prior to calling fit, and will slow



PyTorch

SGD

CLASS torch.optim.SGD(*params*, *lr=<required parameter>*, *momentum=0*, *dampening=0*, *weight_decay=0*, *nesterov=False*, ***, *maximize=False*, *foreach=None*, *differentiable=False*) [SOURCE]

Implements stochastic gradient descent (optionally with momentum).

Parameters:

- **params** (*iterable*) – iterable of parameters to optimize or dicts defining parameter groups
- **lr** (*float*) – learning rate
- **momentum** (*float*, *optional*) – momentum factor (default: 0)
- **weight_decay** (*float*, *optional*) – weight decay (L2 penalty) (default: 0)
- **dampening** (*float*, *optional*) – dampening for momentum (default: 0)
- **nesterov** (*bool*, *optional*) – enables Nesterov momentum (default: False)
- **maximize** (*bool*, *optional*) – maximize the params based on the objective, instead of minimizing (default: False)
- **foreach** (*bool*, *optional*) – whether foreach implementation of optimizer is used. If unspecified by the user (so foreach is None), we will try to use foreach over the for-loop implementation on CUDA, since it is usually significantly more performant. (default: None)
- **differentiable** (*bool*, *optional*) – whether autograd should occur through the optimizer step in training. Otherwise, the step() function runs in a torch.no_grad() context. Setting to True can impair performance, so leave it False if you don't intend to run autograd through this instance (default: False)

Model- vs Hyperparameter

Modelparameter können während dem Training optimiert werden (=Output des Trainings).

Hyperparameter kontrollieren die Flexibilität, Kapazität, Struktur und Training des Models (=Input zum Training)

Beispiele:

- **Splits** in einem Entscheidungsbaum
- **Gewichte** in neuronalen Netzwerken
- **Koeffizienten** eines linearen Models

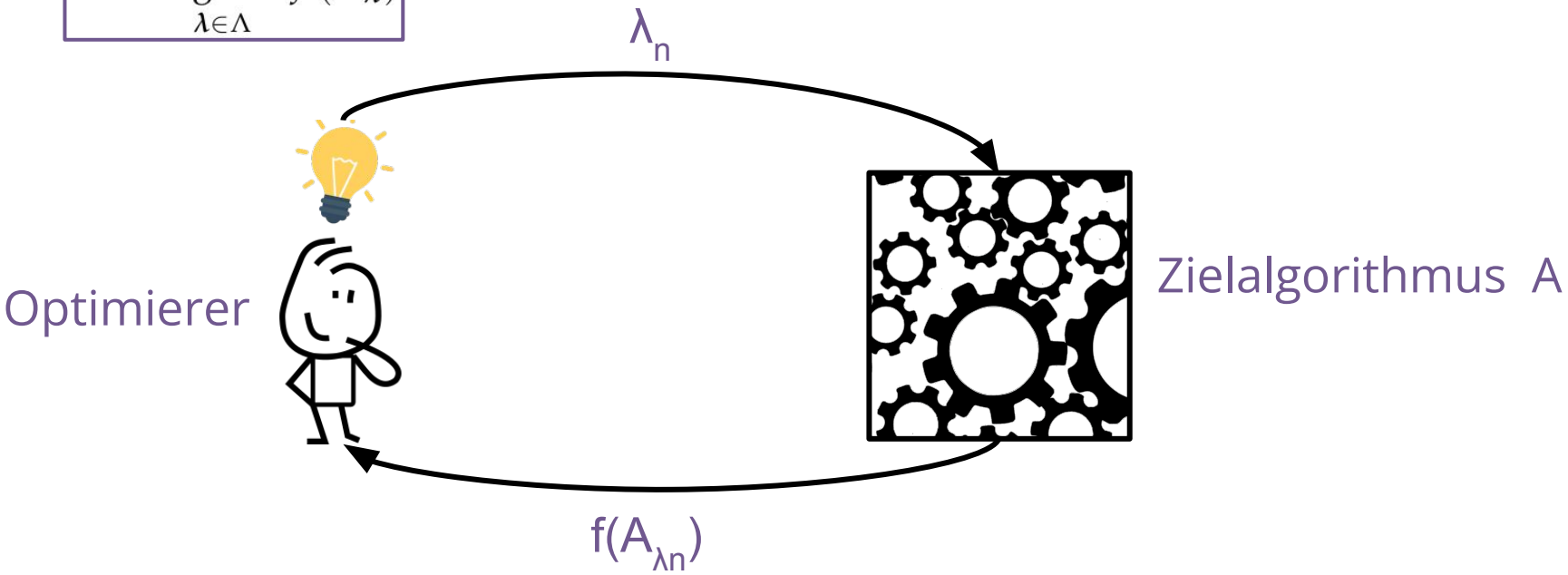
Beispiele:

- **Lernrate** für Gradient Boosting ←kontinuierlich
- **Optimierungsalgorithmus** für NN Training ←kategorisch
- **K** für K-Nearest Neighbours ←diskret

HPO: Hyperparameter Optimierung

Ziel: Finde die beste Konfiguration:

$$\lambda^* \in \arg\min_{\lambda \in \Lambda} f\left(\mathcal{A}_\lambda\right)$$



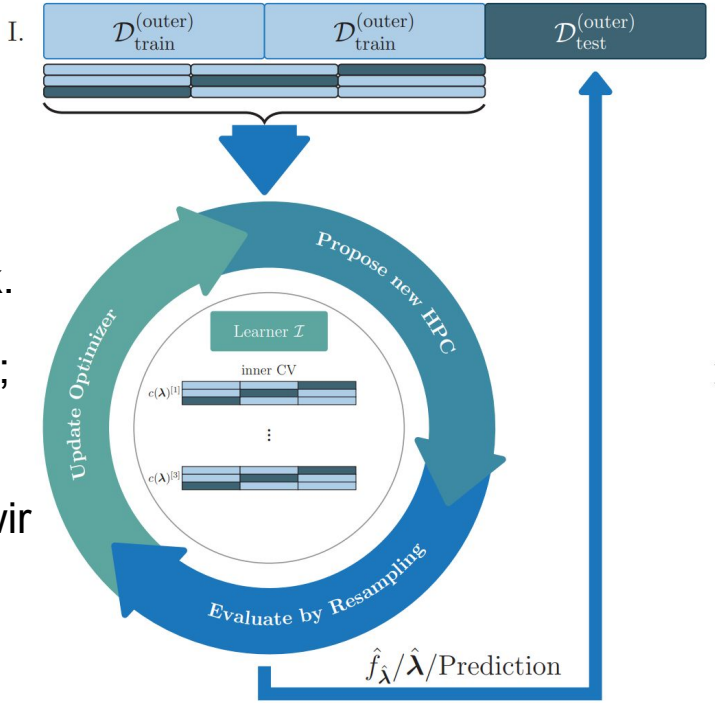
Beispiel

Wir suchen die **optimale** Lernrate und Architektur für ein tiefes NN **mit guter Performanz** für unsere Daten.

Performanz? → Ein “Lerner” gibt ein trainiertes NN zurück.

Gute Performanz? → Ein geringer Generalisierungsfehler; hohe Performanz auf ungesehenen Daten.

Optimal? →Aus den evaluierten Konfigurationen wählen wir die beste bzgl. des Generalisierungsfehlers aus.



Grafik: Bischl et al. 2023

Warum is HPO herausfordernd?

Ziel: Finde die beste Konfiguration:

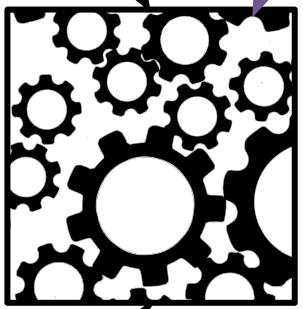
$$\lambda^* \in \arg\min_{\lambda \in \Lambda} f\left(\mathcal{A}_\lambda\right)$$

komplexer Suchraum

Black-Box: Keine Gradienten & kein Vorwissen

λ_n

Optimierer



Zielalgorithmus A

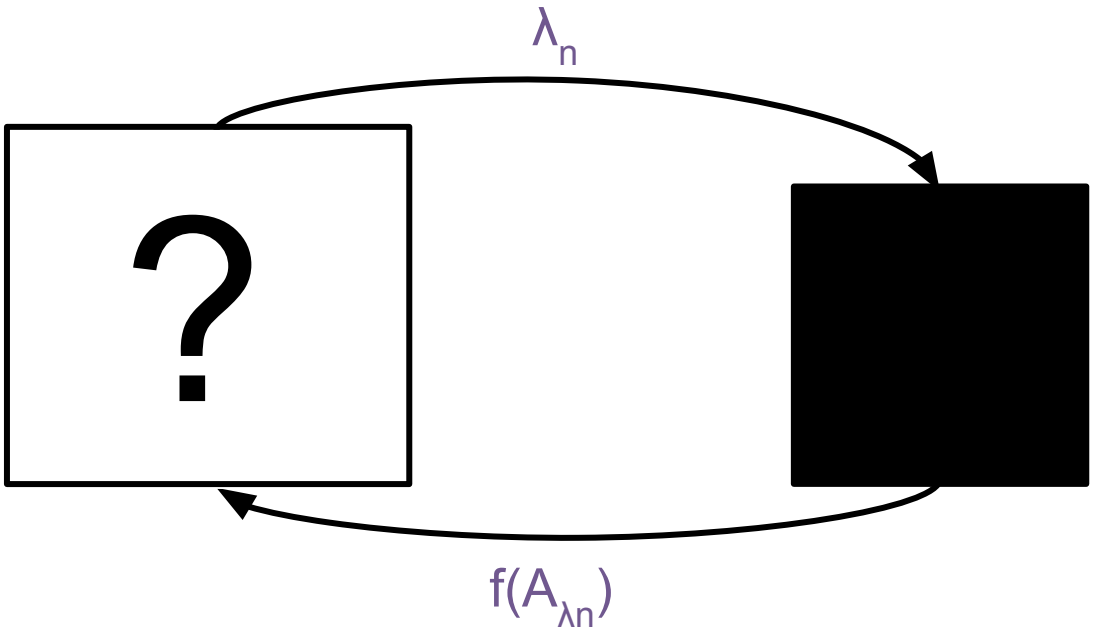
$f(A_{\lambda n})$

verrauscht & teuer zu evaluieren

Wie optimieren wir?

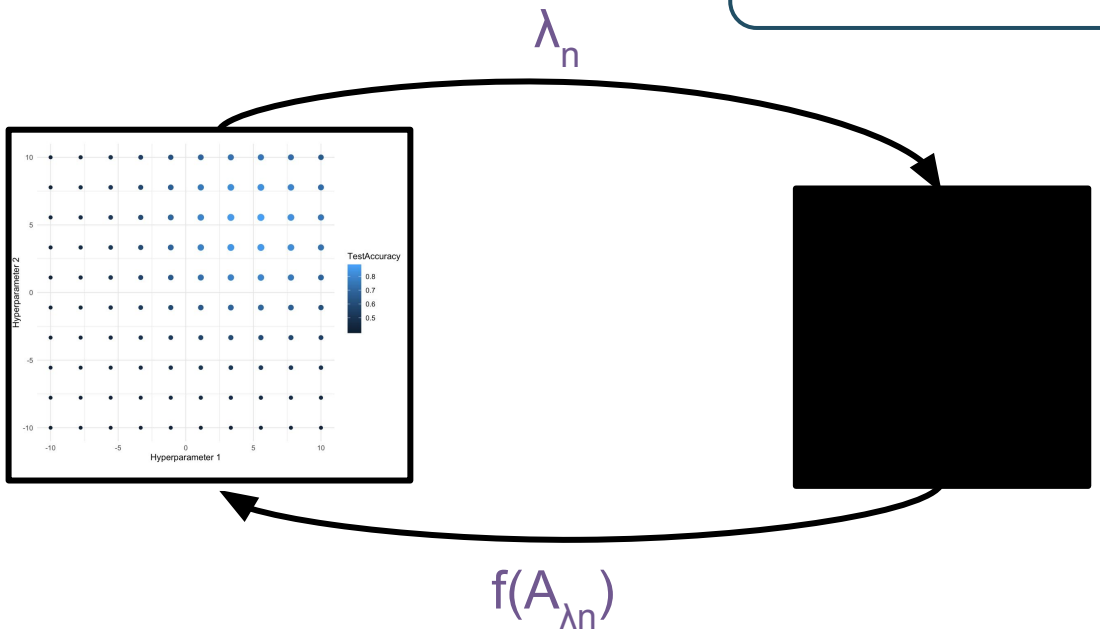
```
>> Okay, hier sind mein Modell, meine Daten, meine Metrik und mein Suchraum. Los geht’s!
```

Black-Box Optimization Problem



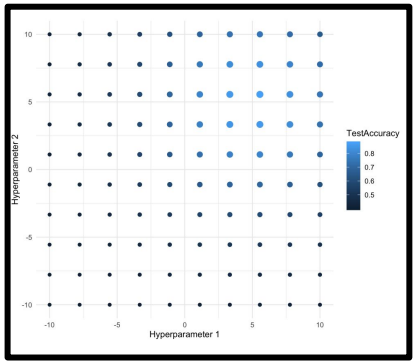
Option 1: Grid Search

Beliebte Technik: Evaluiere alle Kombinationen eines vordefinierten Grids.



Option 1: Grid Search II

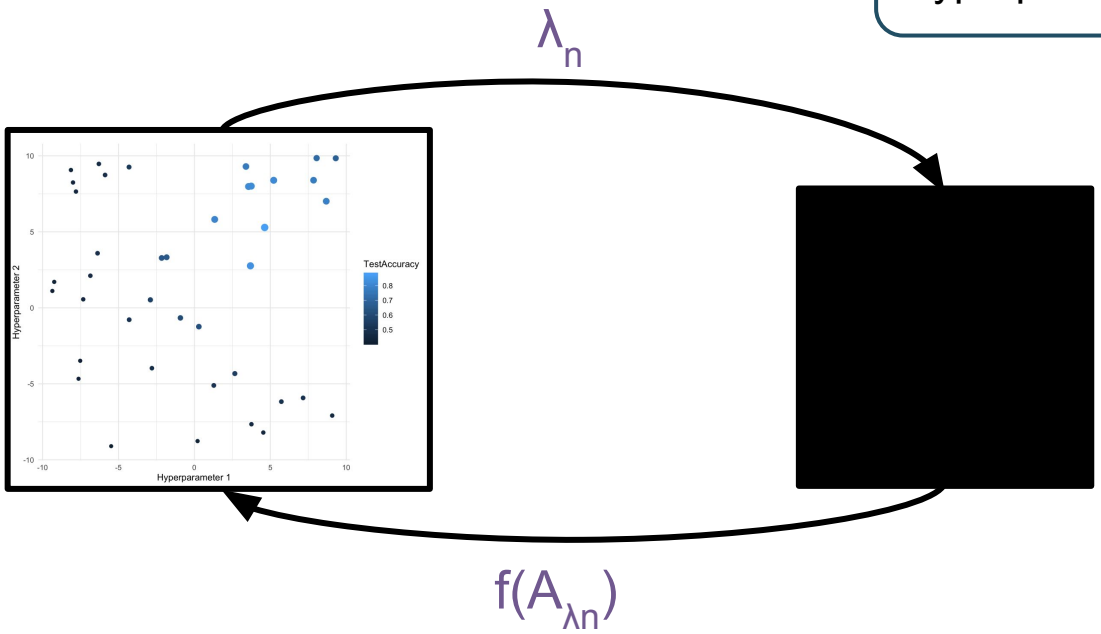
- Einfach zu implementieren
- Einfach zu parallelisieren
- Kann alle Typen von Hyperparametern handhaben



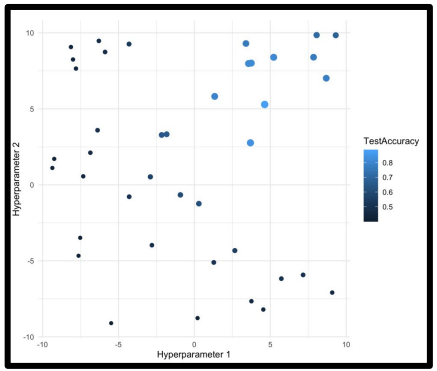
- Skaliert schlecht mit vielen Hyperparametern
- Ineffizient: Durchsucht irrelevante Bereiche
- Manuelles Diskretisieren des Suchraums
- Alle Konfigurationen müssen evaluiert werden

Option 2: Random Search

Evaluiere zufällige Hyperparameter-Konfigurationen



Option 2: Random Search II



- Einfach zu implementieren
- Einfach zu parallelisieren
- Kann alle Typen von Hyperparametern handhaben

- Keine Diskretisierung notwendig
- “Anytime”-Algorithm: Kann jederzeit gestoppt und fortgesetzt werden

- Skaliert schlecht mit vielen Hyperparametern
- Ineffizient: Durchsucht irrelevante Bereiche

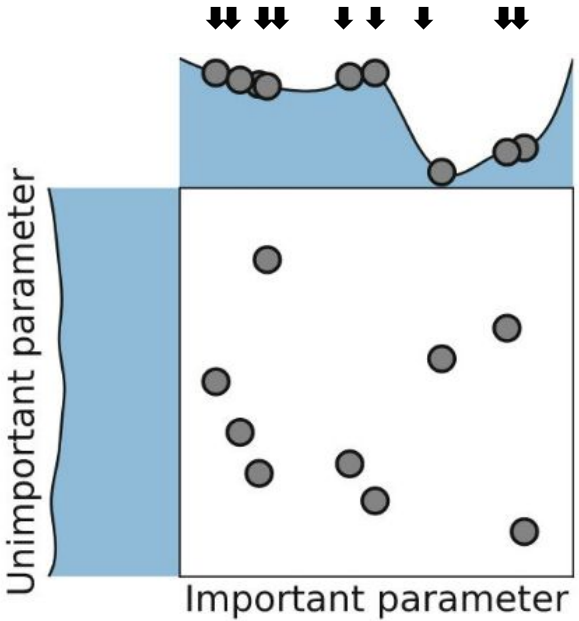
Grid Search vs. Random Search

Budget

T Konfigurationen

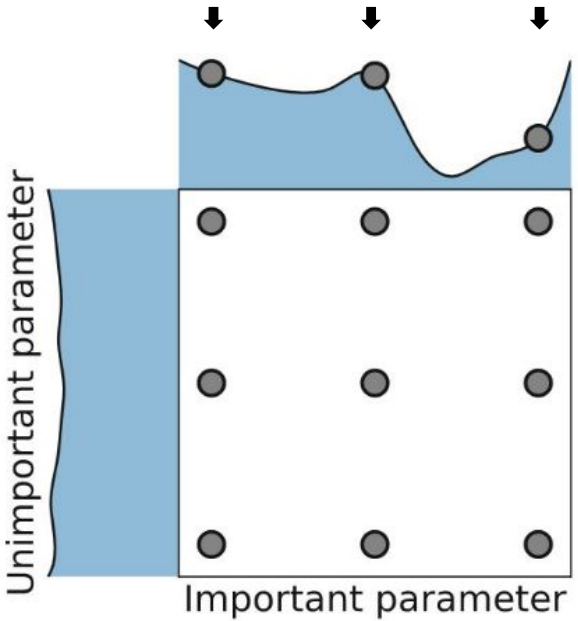
Random Search

T Werte pro Hyperparameter



Grid Search

$T^{1/d}$ Werte pro Hyperparameter



Random Search is effizienter als Grid Search, **wenn nur wenige Hyperparameter wichtig sind**

[Bergstra et al. 2012]

Grafik: [Hutter et al. 2019]

Wie optimieren wir effizient?

```
>> Hier sind mein Algorithmus, meine Daten, meine Metrik, mein Suchraum und ich habe nur wenig Zeit, was soll ich tun?
```

Modell-basierte Optimierung

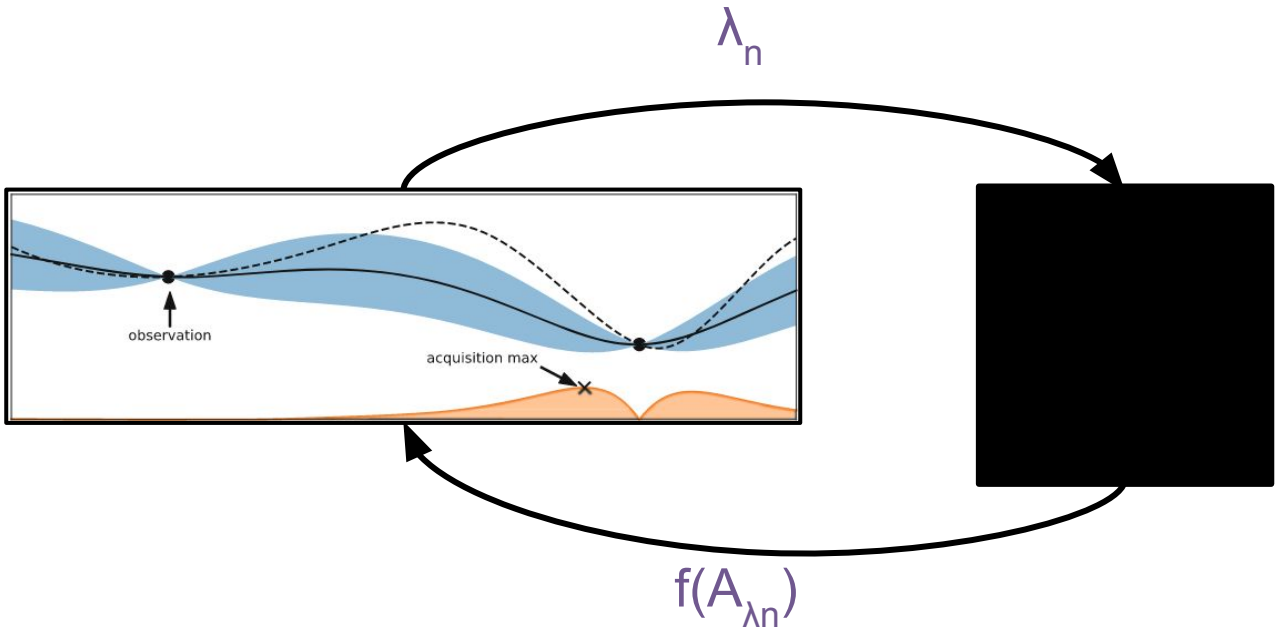
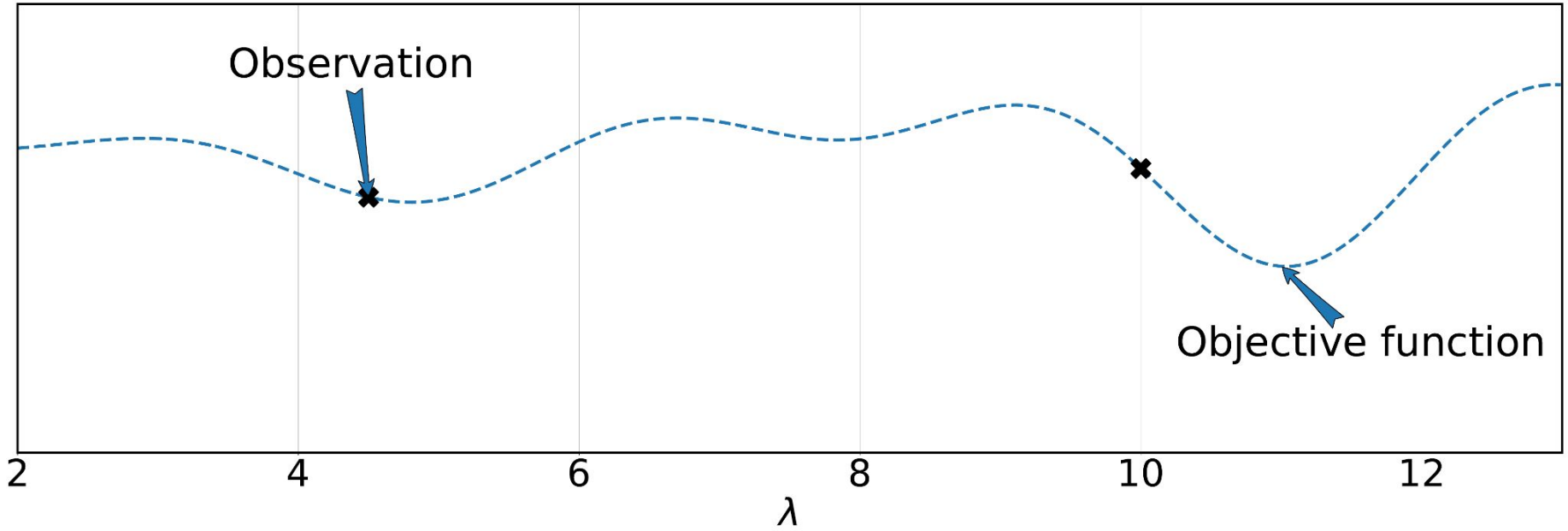
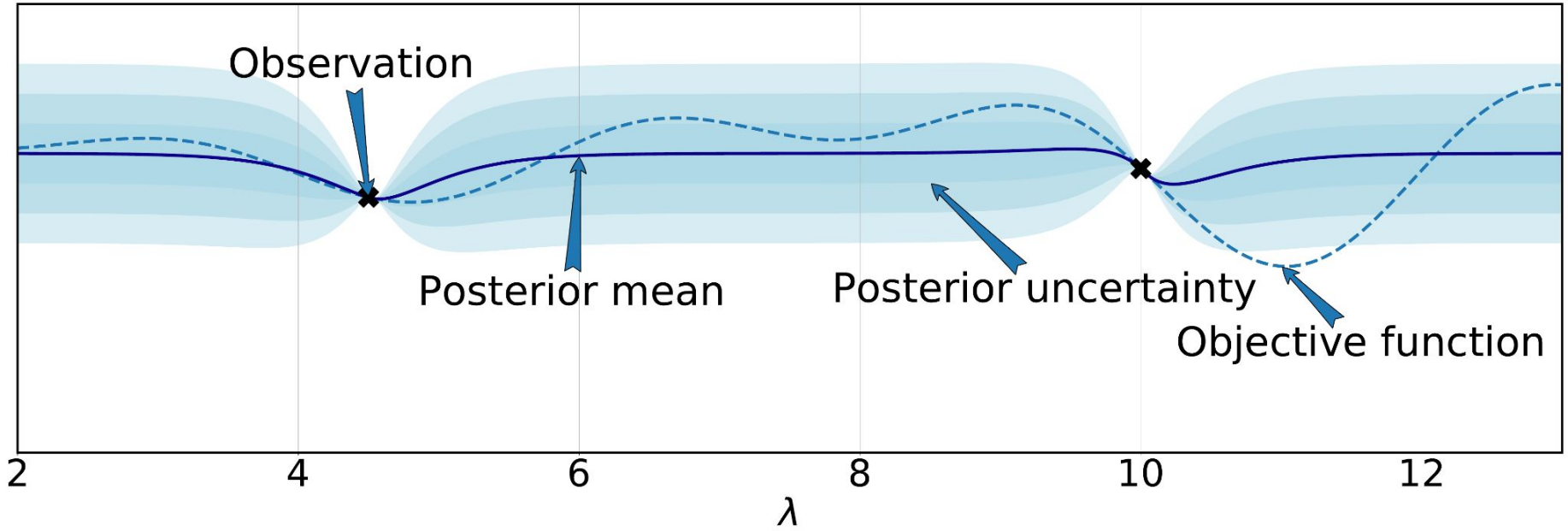


Photo by Wilhelm Gunkel on Unsplash
Image by Feurer, Hutter: Hyperparameter Optimization. In: Automated Machine Learning

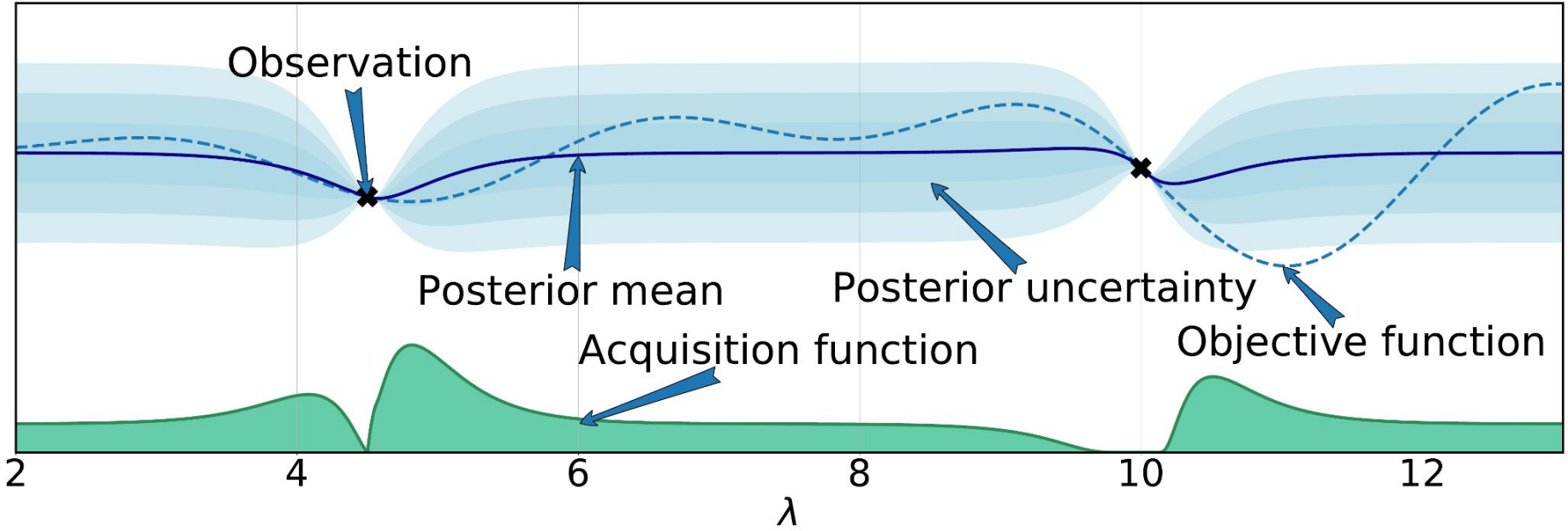
Bayes’sche Optimierung



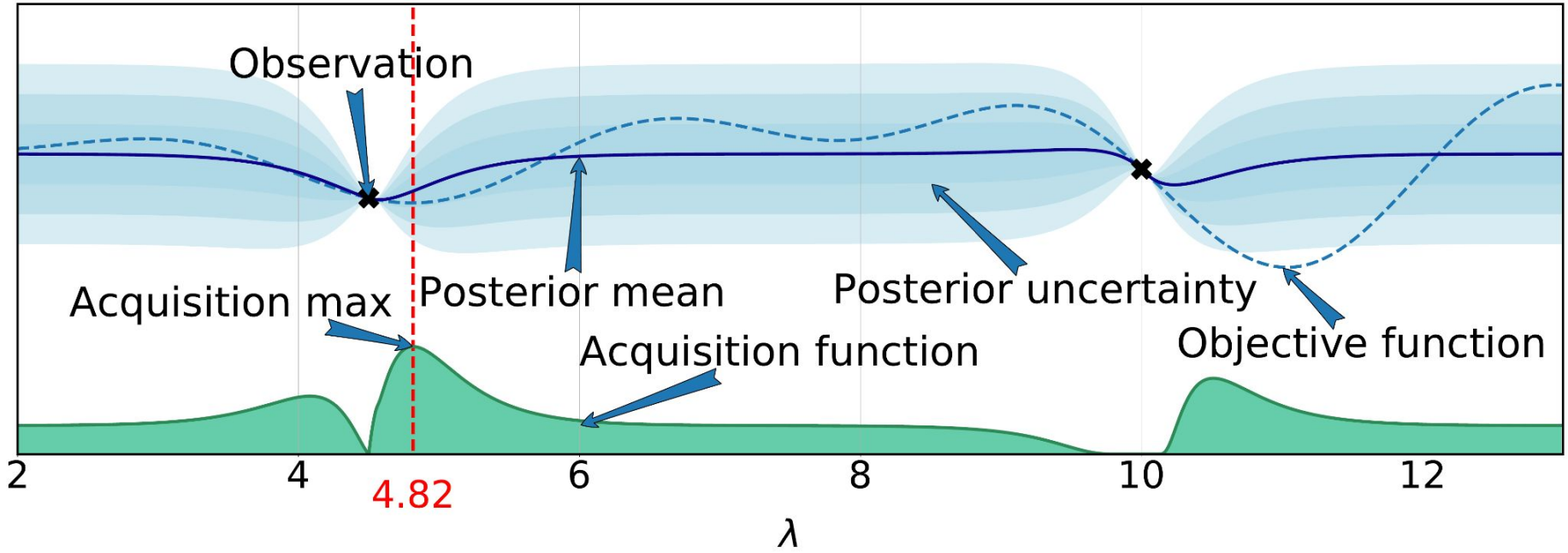
Bayes’sche Optimierung



Bayes’sche Optimierung



Bayes’sche Optimierung

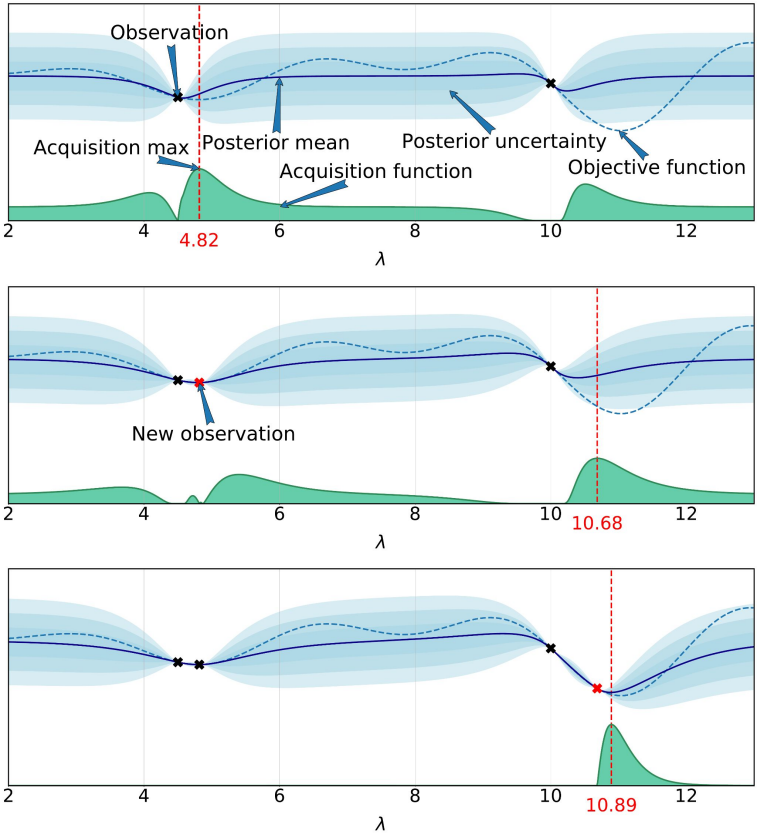


Bayes’sche Optimierung

Solange Budget übrig ist:

1. 🏋️ **Trainiere ein probabilistisches Modell** auf allen evaluierten Konfigurationen
2. 🎯 Nutze das Modell um eine **vielversprechende Konfiguration zu wählen**
3. 🔍 **Evaluiere** die Konfiguration

🪄 Die Magie liegt in der Wahl des **probabilistischen Modells** und der **Balance von Exploration vs Exploitation**



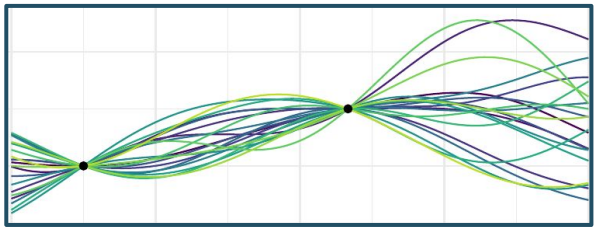
Bayesian Optimization: Pros & Cons

- Effizient bzgl. #Konfigurationen
- Kann alle Typen von Hyperparametern handhaben
- Keine Diskretisierung notwendig
- Kann verrauschte Evaluationen handhaben
- Vorwissen kann integriert werden
- Hat theoretische Garantien

- Overhead durch Training eines Models & Auswahl der nächsten Konfiguration
- Qualität des probabilistischen Modells ausschlaggebend
- Hat selbst Hyperparameter

Typen von BO-Modellen

- Gaussian Processes



- Random Forests

- (Bayesian) Neural Networks

Photo by Filip Zrnzević on Unsplash
Photo by Alina Grubnyak on Unsplash

Wie optimieren wir noch effizienter?

```
>> Hier sind mein Algorithmus, meine Daten, meine Metrik, mein Suchraum und ich habe nur ganz wenig Zeit, was soll ich tun?
```

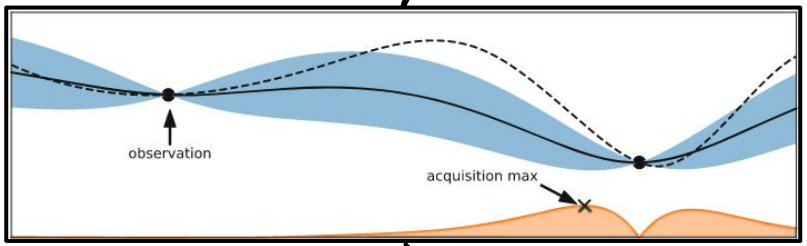
Bayesian Optimization: Erweiterungen

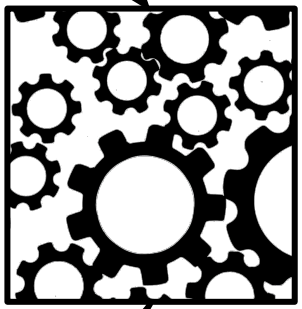
Ersetze alles durch LLMs

Verschiedene Modelle für unterschiedliche HPO Probleme

Höhere Effizienz durch Fidelities

λ_n





Multi-Kriteriell

Kann Beschränkungen berücksichtigen

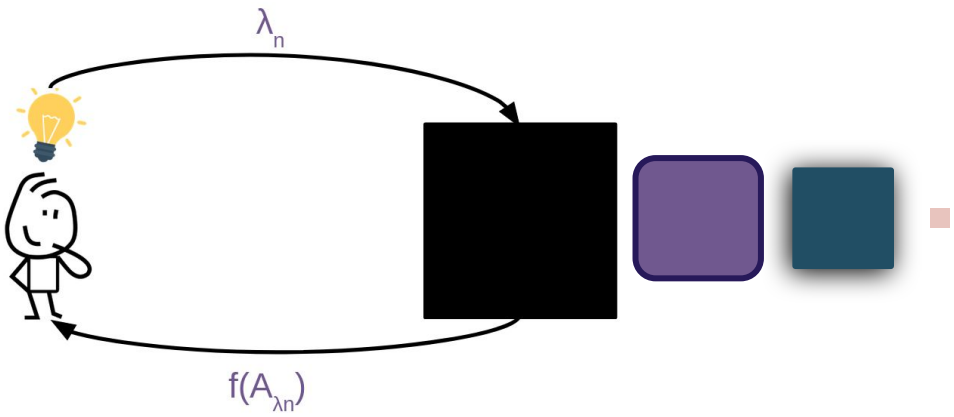
Kann Nutzer- Vorwissen einbeziehen

Kann von vorherigen HPO-Läufen lernen

$f(A_{\lambda n})$

Photo by Wilhelm Gunkel on Unsplash
Image by Feurer, Hutter: Hyperparameter Optimization. In: Automated Machine Learning

Multi-Fidelity Bayes'sche Optimierung



Maschinelles Lernen …

- ist oft ein **iterativer** Prozess,
- hat günstige **Approximationen**,
- oder kann **partiell** evaluiert werden.

Idee: Nutze solche Informationen, die etwas über die eigentliche Performanz aussagen, aber günstiger zu berechnen sind.

Zwei Motivierende Beispiele

Performanz einer SVM
auf Subsets von MNIST

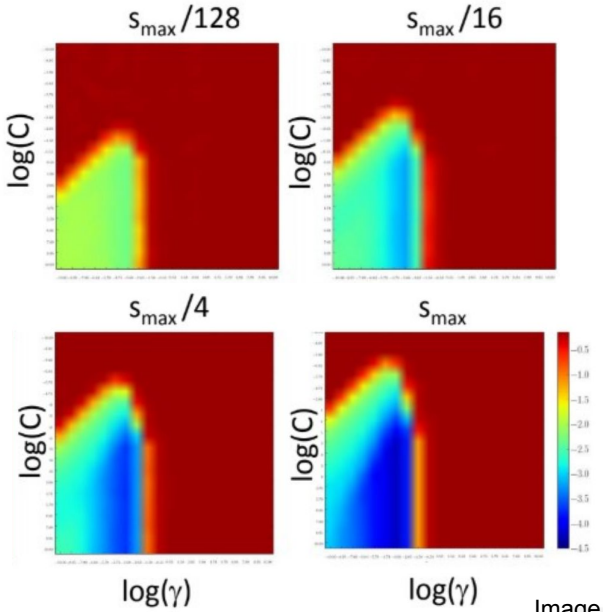


Image Source: [Klein et al. 2016]

Lernkurven von DNNs
auf CIFAR-10

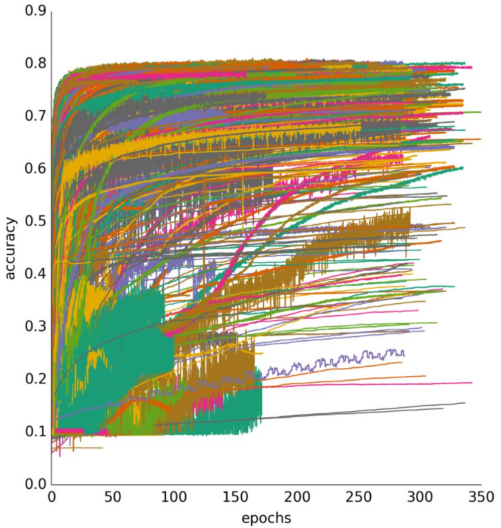
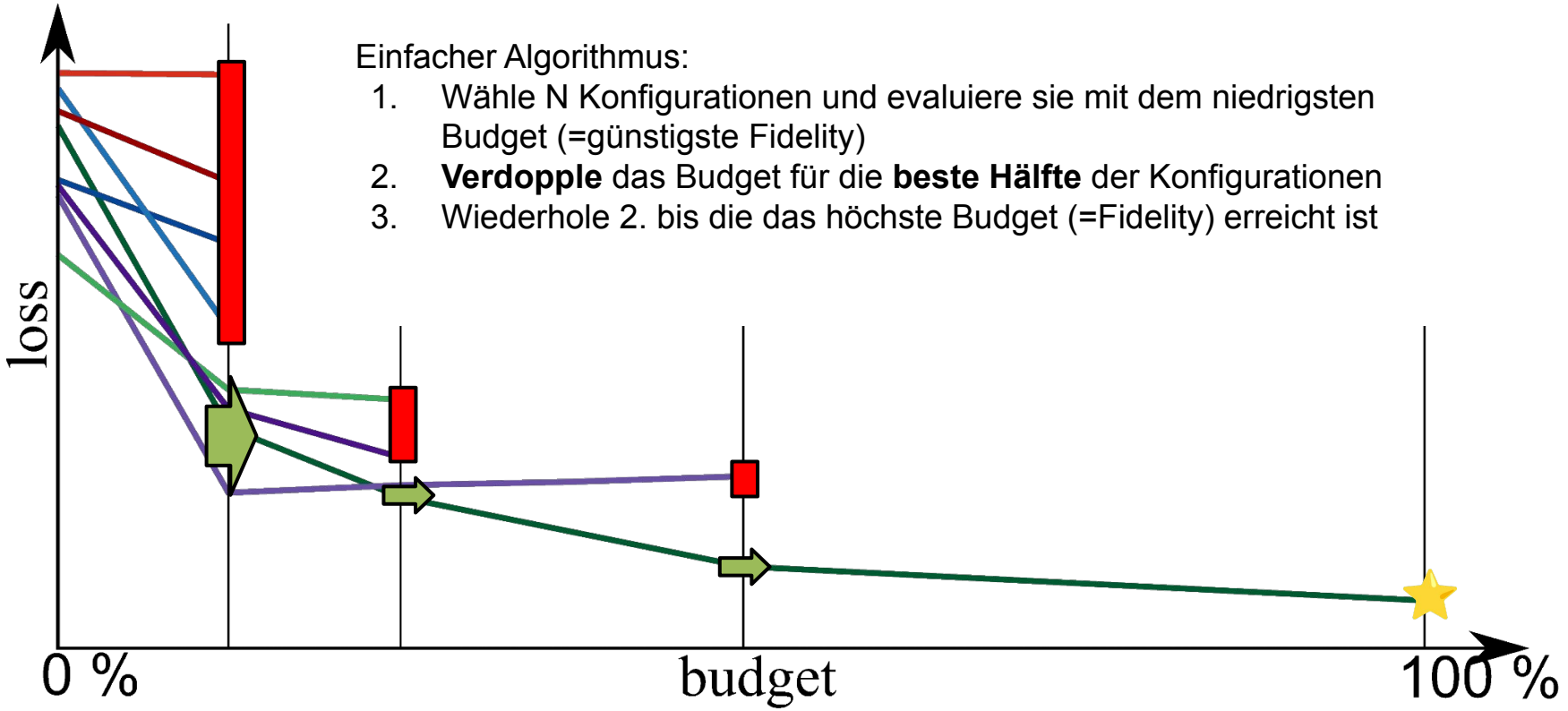


Image Source: [Domhan et al., 2015]

Successive Halving

Einfacher Algorithmus:

1. Wähle N Konfigurationen und evaluiere sie mit dem niedrigsten Budget (=günstigste Fidelity)
2. **Verdopple** das Budget für die **beste Hälfte** der Konfigurationen
3. Wiederhole 2. bis die das höchste Budget (=Fidelity) erreicht ist



Hyperband

Die niedrigste Fidelity abzuschätzen ist oft schwer.

→ Lass mehrere Iterationen mit verschiedenen niedrigsten Fidelities von SH laufen

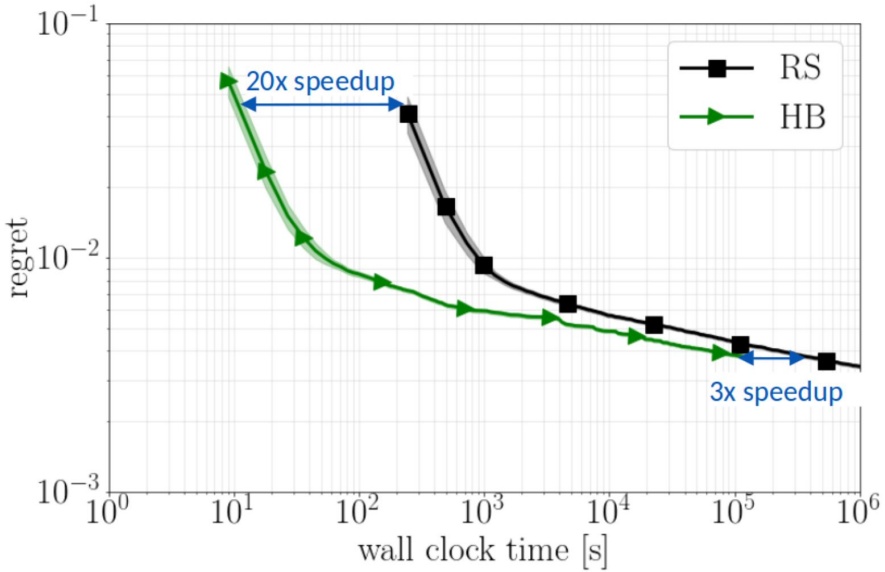


image source: [Falkner et al. 2018]

BOHB: Hyperband x Bayes'sche Optimierung

Idee: Nutze Bayes'sche Optimierung um Konfigurationen auszuwählen [Falkner et al. 2018]

- HB um schnell am Anfang gute Konfigurationen zu identifizieren
- BO um die besten Konfigurationen zu finden

→ Vereint das beste von beiden Ansätzen

→ einfach zu parallelisieren

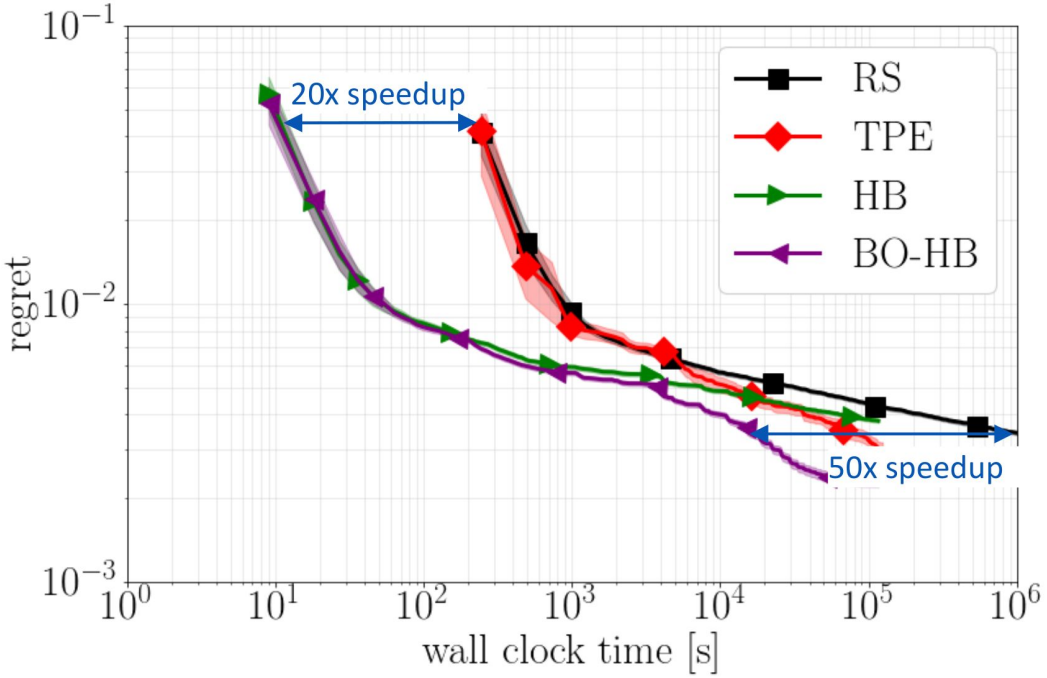


image source: [Falkner et al. 2018]

Welches Tools soll ich nutzen?

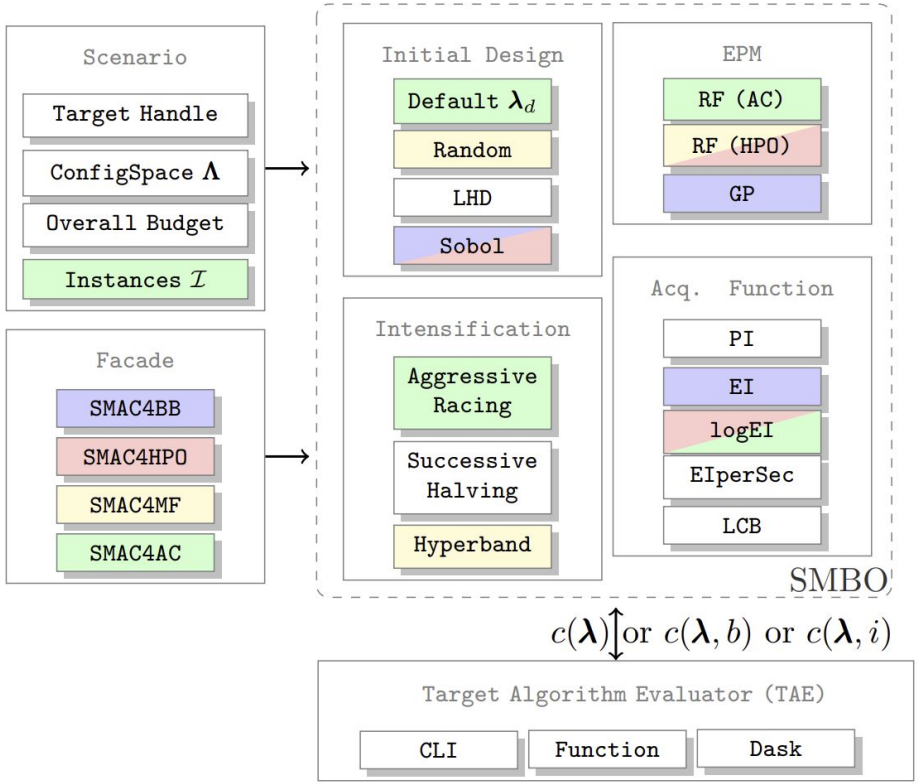
```
>> Ja, und was mache ich nun, wenn ich diese großartigen Techniken nutzen will?
```

automl / SMAC3 Public

Notifications | Fork 206 | Star 920

Code | Issues 74 | Pull requests 10 | Actions | Projects 1 | Security | ...

- Ermöglicht verschiedenste Hyperparameter-Optimierungstechniken
- State-of-the-art Techniken und Performanz
  - Bayes'sche Optimierung
  - Multi-fidelity Optimierung
  - Multi-kriterielle Optimierung
  - Algorithmenkonfiguration
- Konfigurierbar und modular
- Parallelisierbar



AutoML ist doch nicht nur HPO, oder?

"Machine Learning for everyone
in 4 lines of code"

Automatisches Design der kompletten Vorhersagepipeline

```
import automl.classification as automl
> cls = automl.classification.AutoMLClassifier()
> cls.fit(X_train, y_train)
> predictions = cls.predict(X_test)
```

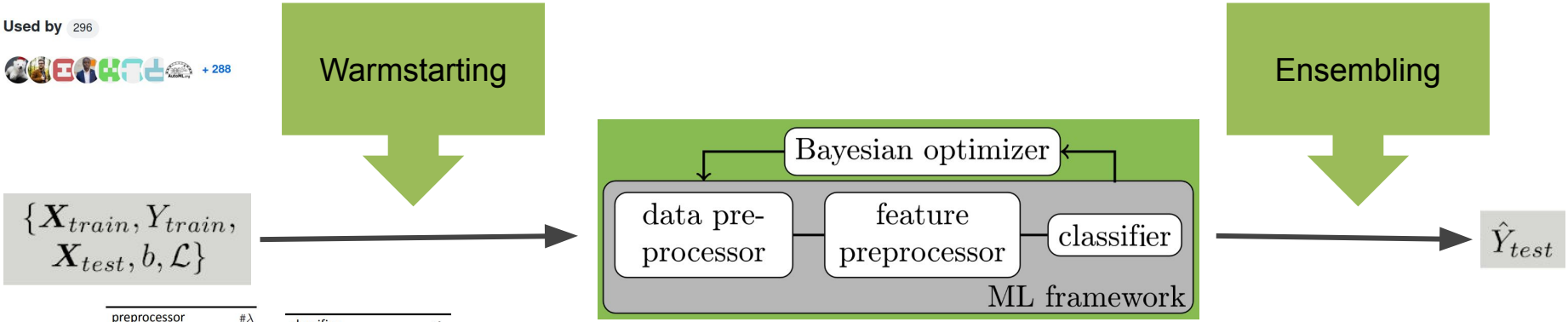
Auto-Sklearn

automl / **auto-sklearn** Public

Edit Pins | Unwatch 214 | Fork 1.2k | Starred 6.5k

Code | Issues 110 | Pull requests 9 | Discussions | Actions | Projects 1 | Wiki | ...

Used by 296 + 288



preprocessor	#λ
extreml. rand. trees prepr.	5
fast ICA	4
feature agglomeration	4
kernel PCA	5
rand. kitchen sinks	2
linear SVM prepr.	3
no preprocessing	-
nystroem sampler	5
PCA	2
polynomial	3
random trees embed.	4
select percentile	2
select rates	3
one-hot encoding	2
imputation	1
balancing	1
rescaling	1

classifier	#λ
AdaBoost (AB)	4
Bernoulli naïve Bayes	2
decision tree (DT)	4
extreml. rand. trees	5
Gaussian naïve Bayes	-
gradient boosting (GB)	6
kNN	3
LDA	4
linear SVM	4
kernel SVM	7
multinomial naïve Bayes	2
passive aggressive	3
QDA	2
random forest (RF)	5
Linear Class. (SGD)	10

/automl/auto-sklearn

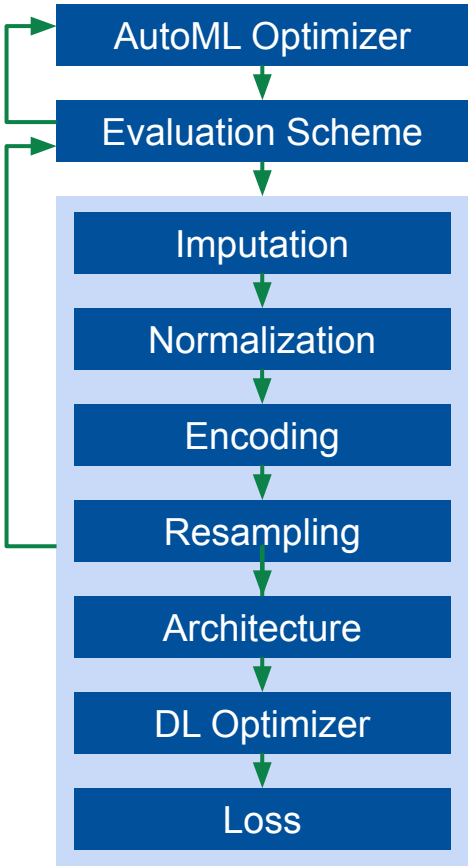
Contributors 78

Auto-PyTorch

automl / Auto-PyTorch Public

Notifications | Fork 265 | Star 2.2k

Code | Issues 50 | Pull requests 20 | Actions | Projects 3 | Wiki | Security | Insights



1. Auch Deep Learning basiert auf komplexen Pipelines
2. Architektursuche & HPO müssen Hand in Hand gehen

➜ Auto-PyTorch [Zimmer et al. 2021]

➜ Auto-PyTorch for Time Series Forecasting [Deng et al. 2022]

```python
# initialise Auto-PyTorch api
api = TabularClassificationTask()

# Search for an ensemble of machine learning algorithms
api.search(
    X_train=X_train,
    y_train=y_train,
    X_test=X_test,
    y_test=y_test,
    optimize_metric='accuracy',
    total_walltime_limit=300,
    func_eval_time_limit_secs=50
)

# Calculate test accuracy
y_pred = api.predict(X_test)
```

Weitere Open-Source AutoML Packages

AutoPrognosis

… und viele mehr, siehe: https://openml.github.io/automlbenchmark/frameworks.html [Gijsbers et. al, 2022]

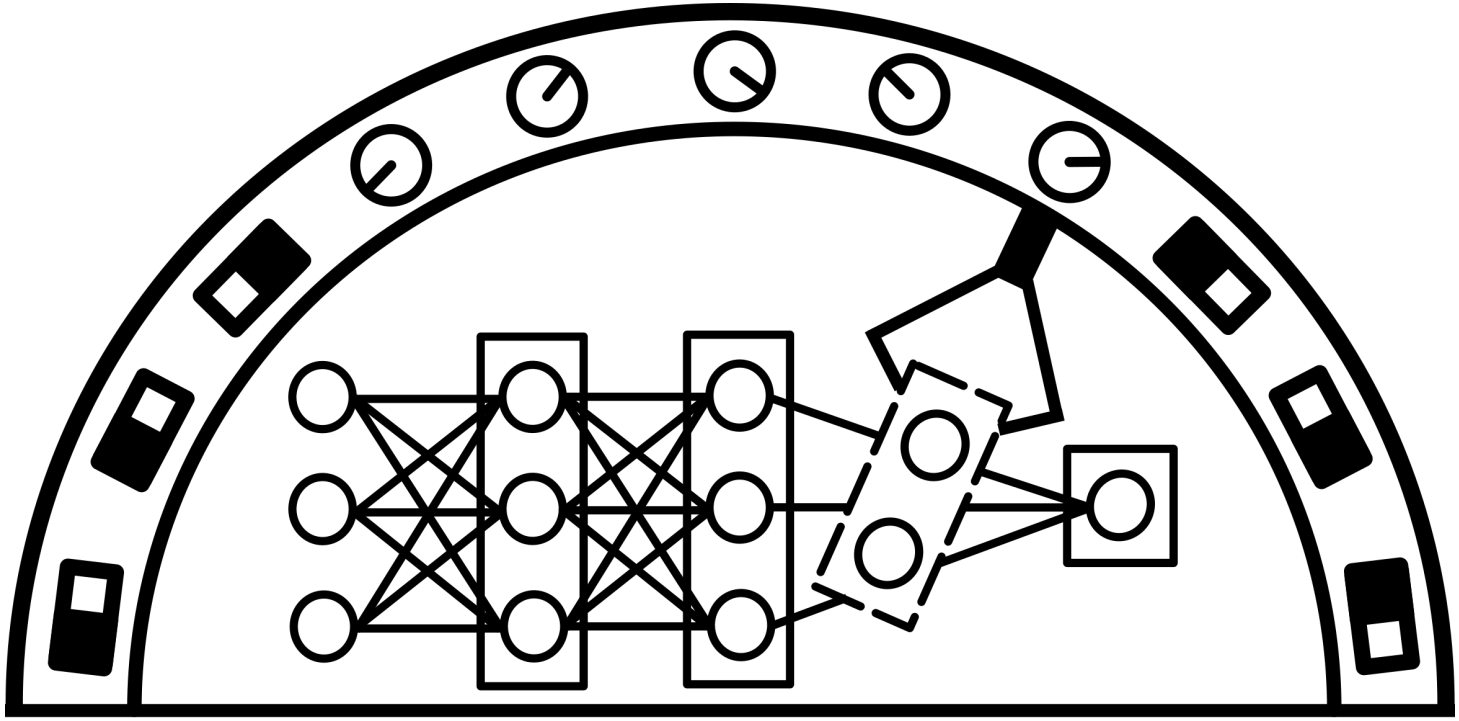
Take-Home Message

>> An was sollte ich mich erinnern, wenn ich mir nur 4 Sachen merken will?

Zusammenfassung

AutoML ermöglicht

- 🚀 **Effizientere** Entwicklung von ML Anwendungen
- 🧮 **Systematischere Anwendung** von ML Methoden
- 📋 Bessere **Reproduzierbarkeit**
- 🦾 **Breitere Anwendung von ML Methoden**

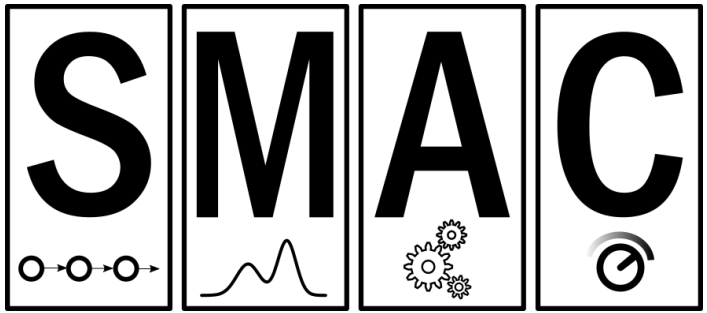


AutoML.org

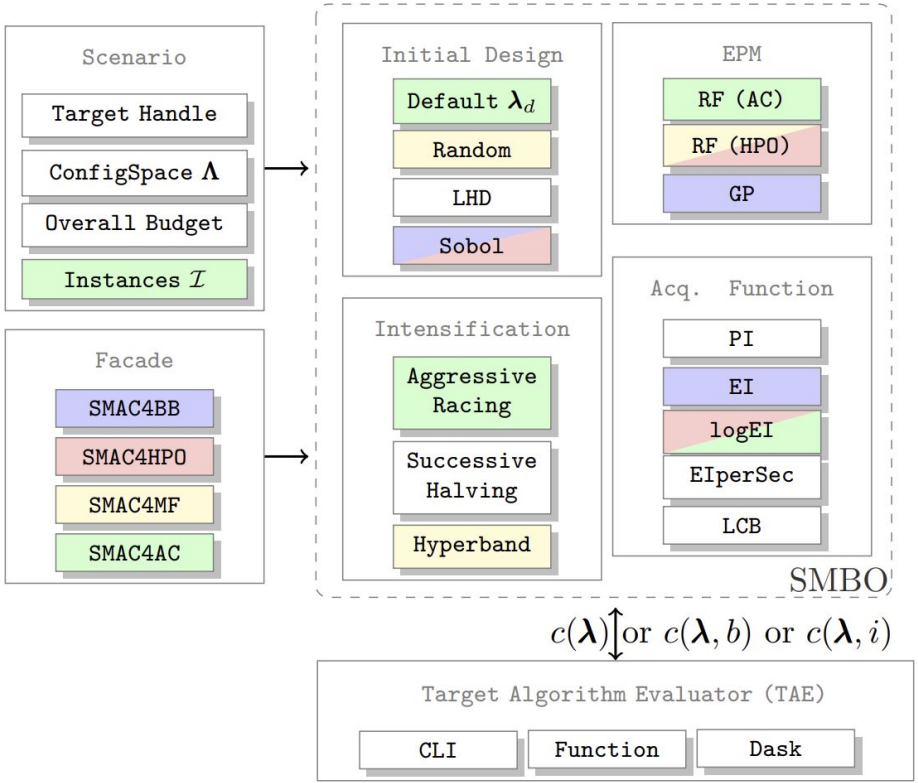
Backup Slides

Welches Tool soll ich nutzen?

```
>> Ihr habt mich überzeugt; was kann ich nun nutzen?
```

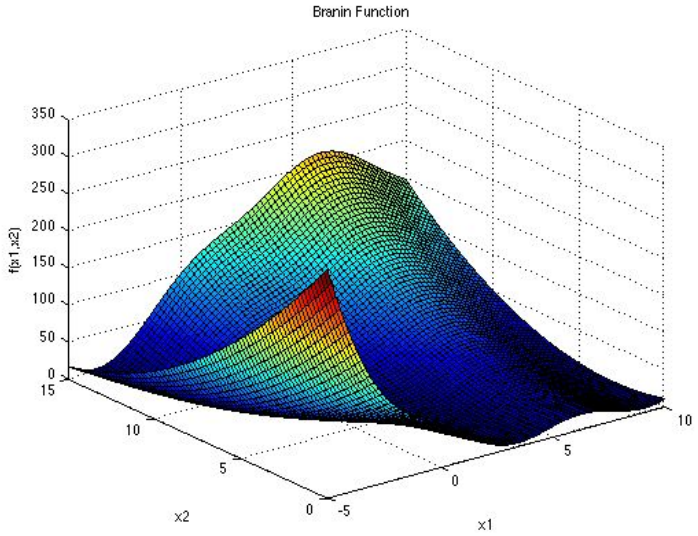


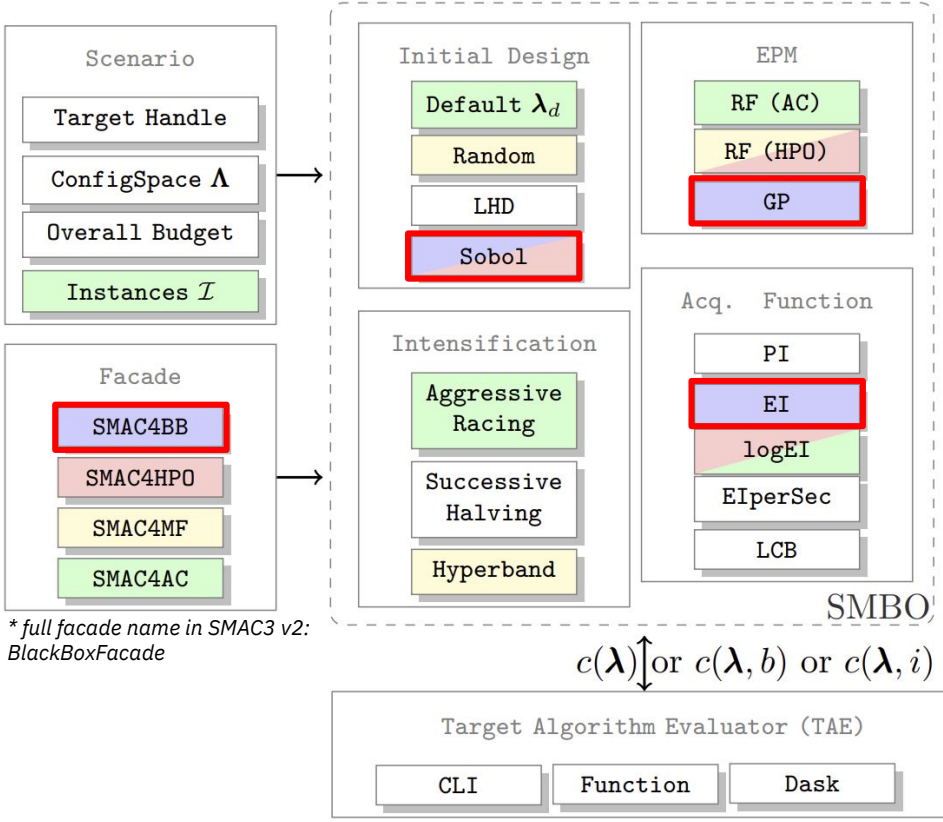
SMAC3: Modulares HPO



SMAC3 für Kontinuierliche Funktionen

$$\boldsymbol{\lambda}^* \in \arg\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} c(\boldsymbol{\lambda})$$



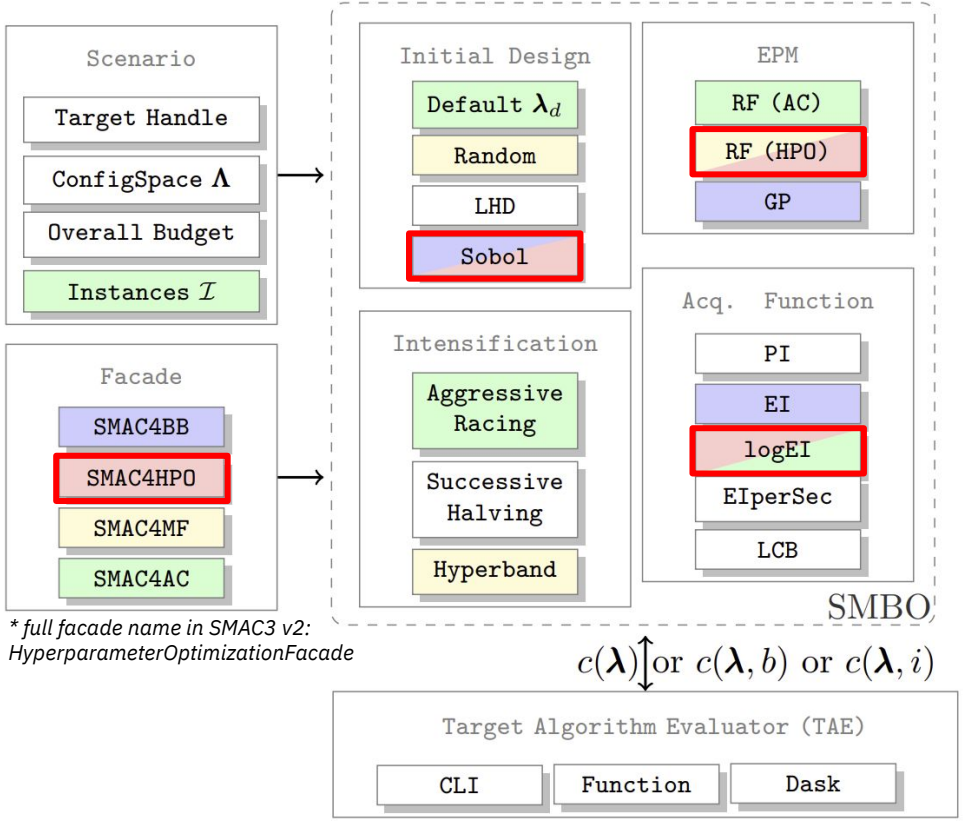


** full facade name in SMAC3 v2: BlackBoxFacade*

SMAC für HPO

$$(A^*, \boldsymbol{\lambda}^*) \in \underset{A_i \in \mathbf{A}, \boldsymbol{\lambda} \in \boldsymbol{\Lambda}_i}{\arg\min} \; c(A_i, \boldsymbol{\lambda}) =$$

$$\underset{A_i \in \mathbf{A}, \boldsymbol{\lambda} \in \boldsymbol{\Lambda}_i}{\arg\min} \; \mathcal{L}(\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}; A_i(\boldsymbol{\lambda})).$$

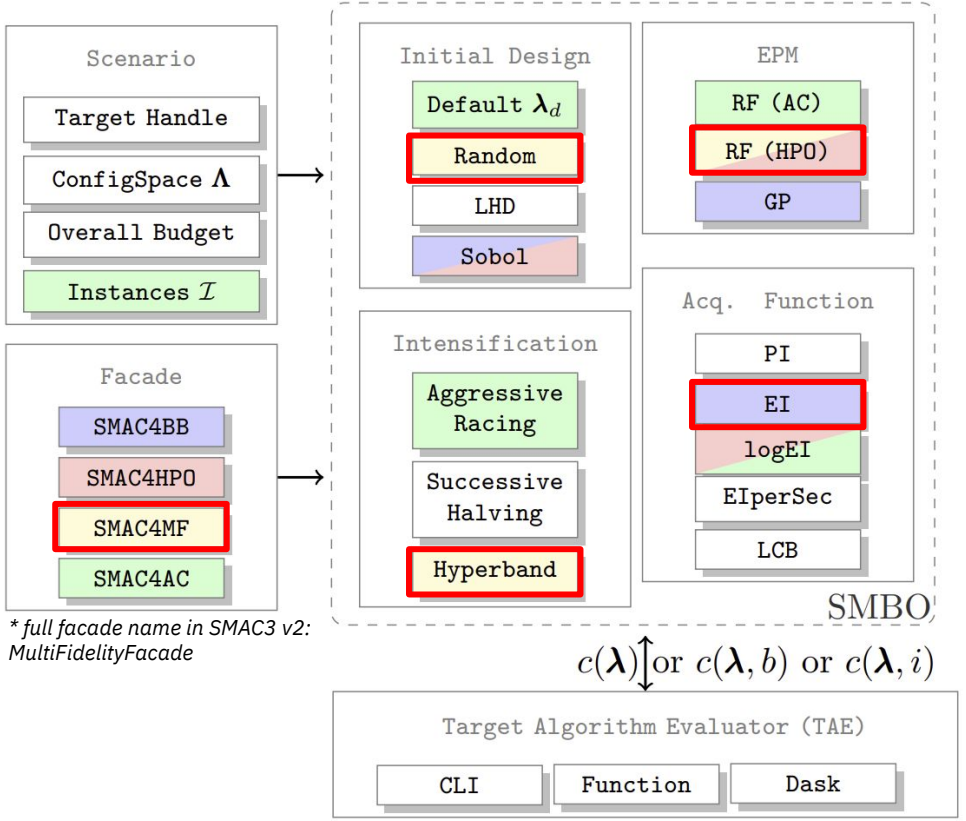


** full facade name in SMAC3 v2: HyperparameterOptimizationFacade*

SMAC für teures HPO – Multi-Fidelity

$$\boldsymbol{\lambda}^* \in \arg\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} c(\boldsymbol{\lambda}, b_{max}) =$$

$$\arg\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} \mathcal{L}(\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}; \boldsymbol{\lambda}, b_{max}).$$

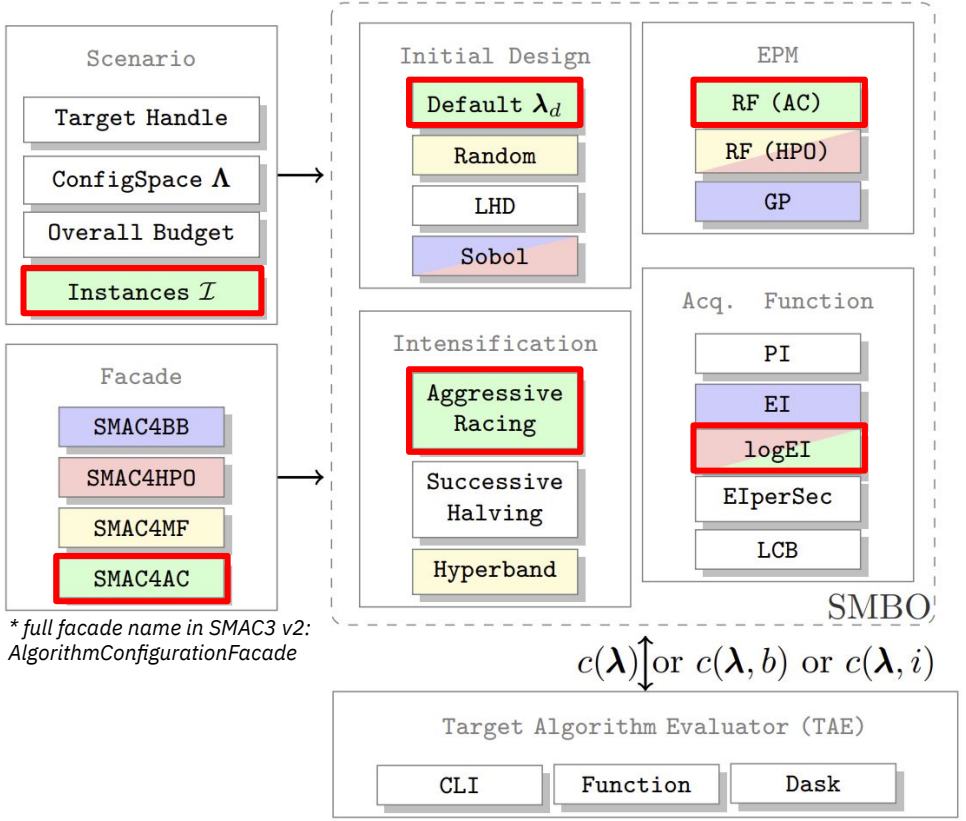


** full facade name in SMAC3 v2: MultiFidelityFacade*

SMAC für Optimierung von mehreren Aufgaben

$$\boldsymbol{\lambda}^* \in \arg\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} c(\boldsymbol{\lambda}) =$$

$$\arg\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} \sum_{i \in \mathcal{I}} c'(\boldsymbol{\lambda}, i)$$



Vergleich mit anderen Paketen

Package	Complex Hyperparameter Space	Multi-Objective	Multi-Fidelity	Instances	Command-Line Interface	Parallelism
HyperMapper	✅	✅	❌	❌	❌	❌
Optuna	✅	✅	✅	❌	✅	✅
Hyperopt	✅	❌	❌	❌	✅	✅
BoTorch	❌	✅	✅	❌	❌	✅
OpenBox	✅	✅	❌	❌	❌	✅
HpBandSter	✅	❌	✅	❌	❌	✅
SMAC	✅	✅	✅	✅	✅	✅