AutoML: Introduction

Overview of AutoML Problems

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Objectives of AutoML

Different metrics to measure the success of ML and AutoML:

- Accuracy on a validation dataset

Objectives of AutoML

Different metrics to measure the success of ML and AutoML:

- Accuracy on a validation dataset
- AUC-ROC

Objectives of AutoML

Different metrics to measure the success of ML and AutoML:

- Accuracy on a validation dataset
- AUC-ROC
- precision & recall
- ...

Objectives of AutoML

Different metrics to measure the success of ML and AutoML:

- Accuracy on a validation dataset
- AUC-ROC
- precision & recall
- ...
- Memory consumption
- Inference time
- ...

Objectives of AutoML

Different metrics to measure the success of ML and AutoML:

- Accuracy on a validation dataset
- AUC-ROC
- precision & recall
- ...
- Memory consumption
- Inference time
- ...

$\rightsquigarrow$ AutoML optimizes an arbitrary cost function, denoted as c.

Objectives of AutoML

Different metrics to measure the success of ML and AutoML:

- Accuracy on a validation dataset
- AUC-ROC
- precision & recall
- ...
- Memory consumption
- Inference time
- ...

$\rightsquigarrow$ AutoML optimizes an arbitrary cost function, denoted as c.

$\rightsquigarrow$ $c(\cdot)$ can also return several cost metrics

Hyperparameters of an SVM

Home | Installation | Documentation | Examples

Google Custom Search

Previous sklearn.svm.O... | Next sklearn.svm.SVR | Up API Reference

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'.

Hyperparameter Optimization

Definition

Let

- $\boldsymbol{\lambda}$ be the hyperparameters of an ML algorithm $\mathcal{A}$ with domain $\boldsymbol{\Lambda}$,

Hyperparameter Optimization

Definition

Let

- $\boldsymbol{\lambda}$ be the hyperparameters of an ML algorithm $\mathcal{A}$ with domain $\boldsymbol{\Lambda}$,
- $\mathcal{D}_{opt}$ be a dataset which is split into $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$

Hyperparameter Optimization

Definition

Let

- $\boldsymbol{\lambda}$ be the hyperparameters of an ML algorithm $\mathcal{A}$ with domain $\boldsymbol{\Lambda}$,
- $\mathcal{D}_{opt}$ be a dataset which is split into $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$
- $c(\mathcal{A}_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{valid})$ denote the cost of $\mathcal{A}_{\boldsymbol{\lambda}}$ trained on $\mathcal{D}_{\text{train}}$ and evaluated on $\mathcal{D}_{\text{val}}$.

Hyperparameter Optimization

Definition

Let

- $\boldsymbol{\lambda}$ be the hyperparameters of an ML algorithm $\mathcal{A}$ with domain $\boldsymbol{\Lambda}$,
- $\mathcal{D}_{opt}$ be a dataset which is split into $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$
- $c(\mathcal{A}_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{valid})$ denote the cost of $\mathcal{A}_{\boldsymbol{\lambda}}$ trained on $\mathcal{D}_{\text{train}}$ and evaluated on $\mathcal{D}_{\text{val}}$.

The *hyper-parameter optimization (HPO)* problem is to find a hyper-parameter configuration that minimizes this cost:

$$\boldsymbol{\lambda}^* \in \arg\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} c(\mathcal{A}_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{valid})$$

Hyperparameter Optimization

Definition

Let

- $\boldsymbol{\lambda}$ be the hyperparameters of an ML algorithm $\mathcal{A}$ with domain $\boldsymbol{\Lambda}$,
- $\mathcal{D}_{opt}$ be a dataset which is split into $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$
- $c(\mathcal{A}_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{valid})$ denote the cost of $\mathcal{A}_{\boldsymbol{\lambda}}$ trained on $\mathcal{D}_{\text{train}}$ and evaluated on $\mathcal{D}_{\text{val}}$.

The *hyper-parameter optimization (HPO)* problem is to find a hyper-parameter configuration that minimizes this cost:

$$\boldsymbol{\lambda}^* \in \arg\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} c(\mathcal{A}_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{valid})$$

Remarks:

- $\arg\min$ returns a set of optimal points of a given function. It suffices to find one element of this set and thus we use $\in$ instead of $=$.

Hyperparameter Optimization

Definition

Let

- $\boldsymbol{\lambda}$ be the hyperparameters of an ML algorithm $\mathcal{A}$ with domain $\boldsymbol{\Lambda}$,
- $\mathcal{D}_{opt}$ be a dataset which is split into $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$
- $c(\mathcal{A}_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{valid})$ denote the cost of $\mathcal{A}_{\boldsymbol{\lambda}}$ trained on $\mathcal{D}_{\text{train}}$ and evaluated on $\mathcal{D}_{\text{val}}$.

The *hyper-parameter optimization (HPO)* problem is to find a hyper-parameter configuration that minimizes this cost:

$$\boldsymbol{\lambda}^* \in \arg\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} c(\mathcal{A}_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{valid})$$

Remarks:

- $\arg\min$ returns a set of optimal points of a given function. It suffices to find one element of this set and thus we use $\in$ instead of $=$.
- Sometimes, we want to optimize for different metrics, instead of one
  - $\rightsquigarrow$ multi-objective optimization and Pareto fronts

Choosing an Algorithm

- Over the years, many ML-algorithms were proposed

Choosing an Algorithm

- Over the years, many ML-algorithms were proposed
- Most of these still have a reason for existence

Choosing an Algorithm

- Over the years, many ML-algorithms were proposed
- Most of these still have a reason for existence
- Examples for classification:
  - logistic regression
  - k-nearest neighbor
  - naïve Bayes
  - support vector machine
  - decision tree
  - random forest
  - gradient boosting
  - multi-layer perceptron
  - residual networks
  - ...

Choosing an Algorithm

- Over the years, many ML-algorithms were proposed
- Most of these still have a reason for existence
- Examples for classification:
  - logistic regression
  - k-nearest neighbor
  - naïve Bayes
  - support vector machine
  - decision tree
  - random forest
  - gradient boosting
  - multi-layer perceptron
  - residual networks
  - ...
- [Fernández-Delgado et al. 2015] studied 179 classifiers on 121 datasets

Choosing an Algorithm

- Over the years, many ML-algorithms were proposed
- Most of these still have a reason for existence
- Examples for classification:
  - logistic regression
  - k-nearest neighbor
  - naïve Bayes
  - support vector machine
  - decision tree
  - random forest
  - gradient boosting
  - multi-layer perceptron
  - residual networks
  - ...
- [Fernández-Delgado et al. 2015] studied 179 classifiers on 121 datasets
- In practice, we actually want to jointly choose the best ML-algorithm and its hyperparameters

CASH: Combined Algorithm Selection and Hyperparameter Optimization

Definition

Let

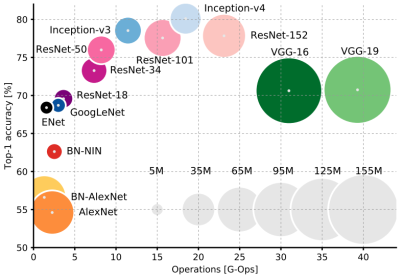
- $\mathbf{A} = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k\}$ be a set of algorithms (a.k.a. portfolio)
- $\mathbf{\Lambda}$ be a set of hyperparameters of each machine learning algorithm $\mathcal{A}_i$
- $\mathcal{D}_{opt}$ be a dataset which is split into $\mathcal{D}_{train}$ and $\mathcal{D}_{valid}$
- $c(\mathcal{A}_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{valid})$ denote the cost of $\mathcal{A}_{\boldsymbol{\lambda}}$ trained on $\mathcal{D}_{train}$ and evaluated on $\mathcal{D}_{valid}$.

we want to find the best combination of algorithm $\mathcal{A} \in \mathbf{A}$ and its hyperparameter configuration $\boldsymbol{\lambda} \in \mathbf{\Lambda}$ minimizing:

$$(\mathcal{A}^*, \boldsymbol{\lambda}^*) \in \mathop{\arg\min}_{\mathcal{A} \in \mathbf{A}, \boldsymbol{\lambda} \in \mathbf{\Lambda}} c(\mathcal{A}_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{valid})$$

Architectures of Neural Networks

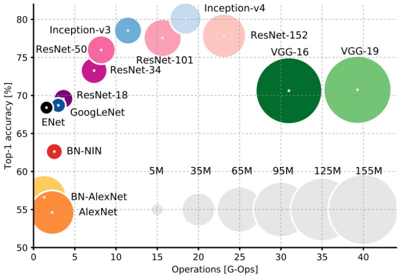
- Many architecture were proposed
- Differences in
  - Depth
  - Resolution
  - Width
  - Operators
  - Connections
  - ...



on Imagenet [Canzian et al. 2017]

Architectures of Neural Networks

- Many architecture were proposed
- Differences in
  - Depth
  - Resolution
  - Width
  - Operators
  - Connections
  - ...

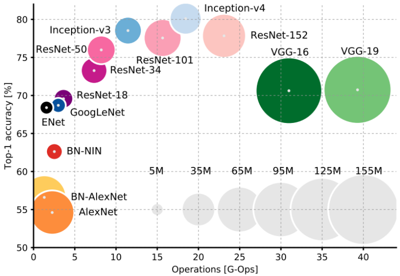


on Imagenet [Canzian et al. 2017]

- Already on a single dataset such as ImageNet, it is not obvious which architeture to choose

Architectures of Neural Networks

- Many architecture were proposed
- Differences in
  - Depth
  - Resolution
  - Width
  - Operators
  - Connections
  - ...

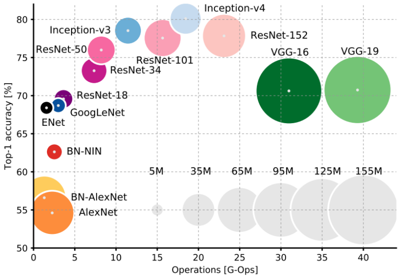


on Imagenet [Canzian et al. 2017]

- Already on a single dataset such as ImageNet, it is not obvious which architeture to choose
- For other datasets, you might need different architectures to achieve top-performance

Architectures of Neural Networks

- Many architecture were proposed
- Differences in
  - Depth
  - Resolution
  - Width
  - Operators
  - Connections
  - ...



on Imagenet [Canzian et al. 2017]

- Already on a single dataset such as ImageNet, it is not obvious which architeture to choose
- For other datasets, you might need different architectures to achieve top-performance
  - For similar datasets, you might use scaled versions of known architectures (e.g., CIFAR10 and Imagenet)

Neural Architecture Search (NAS)

Definition

Let

- $\boldsymbol{\lambda}$ be an architecture for a deep neural network N with domain $\boldsymbol{\Lambda}$,
- $\mathcal{D}_{opt}$ be a dataset which is split into $\mathcal{D}_{train}$ and $\mathcal{D}_{valid}$
- $c(N_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{valid})$ denote the cost of $N_{\boldsymbol{\lambda}}$ trained on $\mathcal{D}_{train}$ and evaluated on $\mathcal{D}_{valid}$.

The *neural architecture search (NAS)* problem is to find an architecture that minimizes this cost:

$$\boldsymbol{\lambda}^* \in \arg\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} c(N_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{valid})$$

Neural Architecture Search (NAS)

Definition

Let

- $\boldsymbol{\lambda}$ be an architecture for a deep neural network N with domain $\boldsymbol{\Lambda}$,
- $\mathcal{D}_{opt}$ be a dataset which is split into $\mathcal{D}_{train}$ and $\mathcal{D}_{valid}$
- $c(N_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{valid})$ denote the cost of $N_{\boldsymbol{\lambda}}$ trained on $\mathcal{D}_{train}$ and evaluated on $\mathcal{D}_{valid}$.

The *neural architecture search (NAS)* problem is to find an architecture that minimizes this cost:

$$\boldsymbol{\lambda}^* \in \arg\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} c(N_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{valid})$$

Remark:

- very similar to the HPO definition

Neural Architecture Search (NAS)

Definition

Let

- $\boldsymbol{\lambda}$ be an architecture for a deep neural network N with domain $\boldsymbol{\Lambda}$,
- $\mathcal{D}_{opt}$ be a dataset which is split into $\mathcal{D}_{train}$ and $\mathcal{D}_{valid}$
- $c(N_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{valid})$ denote the cost of $N_{\boldsymbol{\lambda}}$ trained on $\mathcal{D}_{train}$ and evaluated on $\mathcal{D}_{valid}$.

The *neural architecture search (NAS)* problem is to find an architecture that minimizes this cost:

$$\boldsymbol{\lambda}^* \in \operatorname*{arg\,min}_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} c(N_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{valid})$$

Remark:

- very similar to the HPO definition
- In practice, you want jointly optimize HPO and NAS [Zela et al. 2018]

Per-Instance Algorithm Selection

- For solving the HPO, CASH and NAS problem, we use some kind of optimization approach to search for a solution

Per-Instance Algorithm Selection

- For solving the HPO, CASH and NAS problem, we use some kind of optimization approach to search for a solution
- A different view: Can we learn from data and thus predict an algorithm/ hyperparameter configuration/ neural architecture?

Per-Instance Algorithm Selection

- For solving the HPO, CASH and NAS problem, we use some kind of optimization approach to search for a solution
- A different view: Can we learn from data and thus predict an algorithm/ hyperparameter configuration/ neural architecture?

Definition

Let

- $p(\mathcal{D})$ be a probability distribution over datasets $\mathcal{D} \in \mathbf{D}$,

Per-Instance Algorithm Selection

- For solving the HPO, CASH and NAS problem, we use some kind of optimization approach to search for a solution
- A different view: Can we learn from data and thus predict an algorithm/ hyperparameter configuration/ neural architecture?

Definition

Let

- $p(\mathcal{D})$ be a probability distribution over datasets $\mathcal{D} \in \mathbf{D}$,
- $\mathbf{P}$ a portfolio of algorithms $\mathcal{A} \in \mathbf{P}$, and

Per-Instance Algorithm Selection

- For solving the HPO, CASH and NAS problem, we use some kind of optimization approach to search for a solution
- A different view: Can we learn from data and thus predict an algorithm/ hyperparameter configuration/ neural architecture?

Definition

Let

- $p(\mathcal{D})$ be a probability distribution over datasets $\mathcal{D} \in \mathbf{D}$,
- $\mathbf{P}$ a portfolio of algorithms $\mathcal{A} \in \mathbf{P}$, and
- $c : \mathbf{P} \times \mathbf{D} \to \mathbb{R}$ be a cost metric

Per-Instance Algorithm Selection

- For solving the HPO, CASH and NAS problem, we use some kind of optimization approach to search for a solution
- A different view: Can we learn from data and thus predict an algorithm/ hyperparameter configuration/ neural architecture?

Definition

Let

- $p(\mathcal{D})$ be a probability distribution over datasets $\mathcal{D} \in \mathbf{D}$,
- $\mathbf{P}$ a portfolio of algorithms $\mathcal{A} \in \mathbf{P}$, and
- $c : \mathbf{P} \times \mathbf{D} \to \mathbb{R}$ be a cost metric

the *per-instance algorithm selection problem* is to obtain a mapping $s : \mathcal{D} \mapsto \mathcal{A}$ such that

$$\arg\min_{s} \int_{\mathbf{D}} c(s(\mathcal{D}), \mathcal{D}) p(\mathcal{D}) \, \mathrm{d}\mathcal{D}$$

Dynamic Algorithm Configuration

- So far, we assume that an algorithm runs with some given settings

Dynamic Algorithm Configuration

- So far, we assume that an algorithm runs with some given settings
- However, some settings, such as learning rate, have to be adapted over time

Dynamic Algorithm Configuration

- So far, we assume that an algorithm runs with some given settings
- However, some settings, such as learning rate, have to be adapted over time

Definition

Let

- $\boldsymbol{\lambda}$ be a hyperparameter configuration of an algorithm $\mathcal{A}$,

Dynamic Algorithm Configuration

- So far, we assume that an algorithm runs with some given settings
- However, some settings, such as learning rate, have to be adapted over time

Definition

Let

- $\boldsymbol{\lambda}$ be a hyperparameter configuration of an algorithm $\mathcal{A}$,
- $p(\mathcal{D})$ be a probability distribution over datasets $\mathcal{D} \in \mathbf{D}$,

Dynamic Algorithm Configuration

- So far, we assume that an algorithm runs with some given settings
- However, some settings, such as learning rate, have to be adapted over time

Definition

Let

- $\boldsymbol{\lambda}$ be a hyperparameter configuration of an algorithm $\mathcal{A}$,
- $p(\mathcal{D})$ be a probability distribution over datasets $\mathcal{D} \in \mathbf{D}$,
- s_t be a state description of $\mathcal{A}$ solving $\mathcal{D}$ at time point t,

Dynamic Algorithm Configuration

- So far, we assume that an algorithm runs with some given settings
- However, some settings, such as learning rate, have to be adapted over time

Definition

Let

- $\boldsymbol{\lambda}$ be a hyperparameter configuration of an algorithm $\mathcal{A}$,
- $p(\mathcal{D})$ be a probability distribution over datasets $\mathcal{D} \in \mathbf{D}$,
- s_t be a state description of $\mathcal{A}$ solving $\mathcal{D}$ at time point t,
- $c : \mathbf{\Pi} \times \mathbf{D} \to \mathbb{R}$ be a cost metric assessing the cost of a conf. policy $\pi \in \Pi$ on $\mathcal{D} \in \mathbf{D}$

Dynamic Algorithm Configuration

- So far, we assume that an algorithm runs with some given settings
- However, some settings, such as learning rate, have to be adapted over time

Definition

Let

- $\boldsymbol{\lambda}$ be a hyperparameter configuration of an algorithm $\mathcal{A}$,
- $p(\mathcal{D})$ be a probability distribution over datasets $\mathcal{D} \in \mathbf{D}$,
- s_t be a state description of $\mathcal{A}$ solving $\mathcal{D}$ at time point t,
- $c : \mathbf{\Pi} \times \mathbf{D} \to \mathbb{R}$ be a cost metric assessing the cost of a conf. policy $\pi \in \Pi$ on $\mathcal{D} \in \mathbf{D}$

the *dynamic algorithm configuration problem (DAC)* is to obtain a configuration policy $\pi^* : s_t \times \mathcal{D} \mapsto \boldsymbol{\lambda}$ by optimizing its cost across a distribution of datasets:

$$\pi^* \in \arg\min_{\pi \in \mathbf{\Pi}} \int_{\mathbf{D}} p(\mathcal{D}) c(\pi, \mathcal{D}) \, \mathrm{d}\mathcal{D}$$

Summary

HPO	Search for the best hyperparameter configuration of a ML algorithm
CASH	Search for the best combination of algorithm and hyperparameter configuration
NAS	Search for the architecture of neural network
Selection	Predict the best algorithm (and its hyperparameter configuration)
DAC	Predict the best hyperparameter configuration for an algorithm state at a given time point