RL: Introduction

What drives us?

Marius Lindauer

Leibniz Universität Hannover

Winter Term 2021

AutoML: 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.

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:

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

Reason I: AutoML for RL

- RL algorithms also have many hyperparameters
- Deep RL depends on the network architecture used

$\rightsquigarrow$ Performance of RL depends on both

[Henderson et al. 2019], [Engstrom et al. 2020]

Reason I: AutoML for RL

- RL algorithms also have many hyperparameters
- Deep RL depends on the network architecture used

$\rightsquigarrow$ Performance of RL depends on both
[Henderson et al. 2019], [Engstrom et al. 2020]

- Hard to apply AutoML to RL because
  - RL agents need a long time to really start learning
  - Learning of RL agents is very noisy $\rightsquigarrow$ very noisy signal for AutoML

Reason II: Dynamic Algorithm Configuration

- Often we assume that an algorithm runs with some single settings

Reason II: Dynamic Algorithm Configuration

- Often we assume that an algorithm runs with some single settings
- But some settings, e.g., learning rate, have to be dynamically adapted

Reason II: Dynamic Algorithm Configuration

- Often we assume that an algorithm runs with some single settings
- But some settings, e.g., learning rate, have to be dynamically adapted

Definition

Let

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

Reason II: Dynamic Algorithm Configuration

- Often we assume that an algorithm runs with some single settings
- But some settings, e.g., learning rate, have to be dynamically adapted

Definition

Let

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

Reason II: Dynamic Algorithm Configuration

- Often we assume that an algorithm runs with some single settings
- But some settings, e.g., learning rate, have to be dynamically adapted

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,

Reason II: Dynamic Algorithm Configuration

- Often we assume that an algorithm runs with some single settings
- But some settings, e.g., learning rate, have to be dynamically adapted

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,
- $c : \quad \times \mathbf{D} \rightarrow \mathbb{R}$ be a cost metric assessing the cost of a conf. policy $\pi \in \Pi$ on $\mathcal{D} \in \mathbf{D}$

Reason II: Dynamic Algorithm Configuration

- Often we assume that an algorithm runs with some single settings
- But some settings, e.g., learning rate, have to be dynamically adapted

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,
- $c : \quad \times \mathbf{D} \to \mathbb{R}$ 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:

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

RL for Dynamic Algorithm Configuration

$\leadsto$ We learn π via RL!

- We showed that:
  - Dynamic Algorithm Configuration can be formulated as a RL problem [Biedenkapp et al. 2020]
  - Heuristics of planning solvers can be automatically and dynamically selected [Speck et al. 2020]
  - We can use a teacher (i.e., existing heuristics) to efficiently learn step size settings of CMA-ES [Shala et al. 2020]
  - We can speed up learning by repeating actions [Biedenkapp et al. 2020]
  - We can speed up learning by learning an efficient schedule of task instances [Eimer et al. 2020]