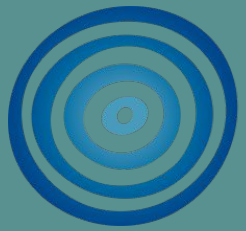


OPTUNA

Presentación realizada por:
Anahí Sulca y Adrián Zelaya

¿Qué es Optuna?

Optuna is an Open Source automatic hyperparameter optimization software framework, particularly designed for machine learning.

It features an imperative, *define-by-run*[1] style user API which ensures the code written with Optuna enjoys high modularity, and the user of Optuna can dynamically construct the search spaces for the hyperparameters.

Optuna is framework agnostic. You can use it with any machine learning or deep learning framework.

Autores: Preferred Networks, Inc.

Paper: Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta,and Masanori Koyama. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. In KDD.

Repo: optuna@github

[1]In the *define-by-run* style DL framework the user is allowed to directly program how each variables are to be manipulated in the network.

Key features

- **Platform agnostic architecture:** Handle a wide variety of tasks with a simple installation that has few requirements.
- **Pythonic search spaces:** Automated search for optimal hyperparameters using Python conditionals, loops, and syntax.
- **State-of-the-art optimization algorithms:** Efficiently search large spaces and prune unpromising trials for faster results.
- **Parallelization:** Parallelize hyperparameter searches over multiple threads or processes without modifying code. This can be scaled further using additional libraries like RayTune.
- **Visualization:** Inspect optimization histories from a variety of plotting functions.

Basic Concepts

Optuna is a framework designed for the automation and the acceleration of the optimization studies.

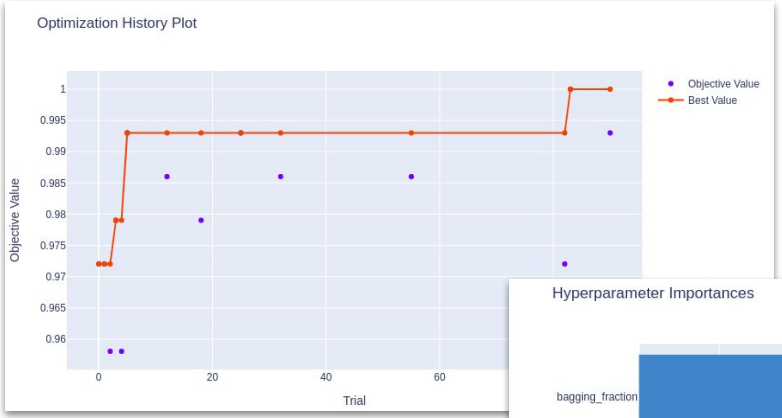
We use the terms study and trial as follows:

- **Study:** optimization based on an objective function. It's a set of trials.
- **Trial:** a single execution of the objective function.
- **Parameter:** A variable whose value is to be optimized.

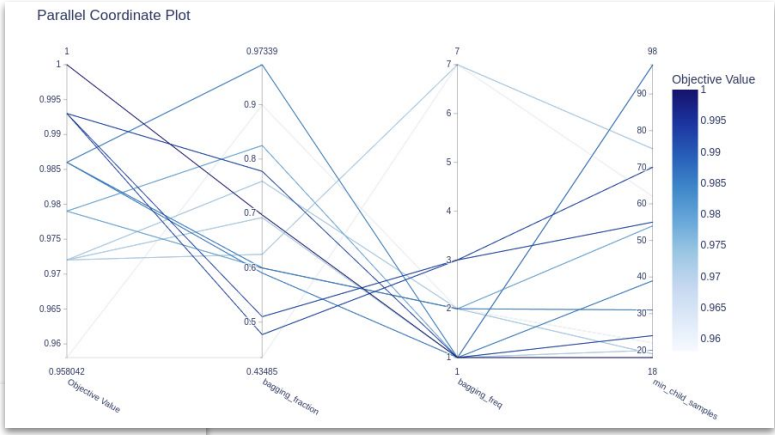
The goal of a study is to find out the optimal set of hyperparameter values (e.g., classifier and svm_c) through multiple trials (e.g., n_trials=100).

This study structure is used with any machine learning or deep learning framework, e.g. PyTorch, TensorFlow, Keras, scikit-learn, etc.

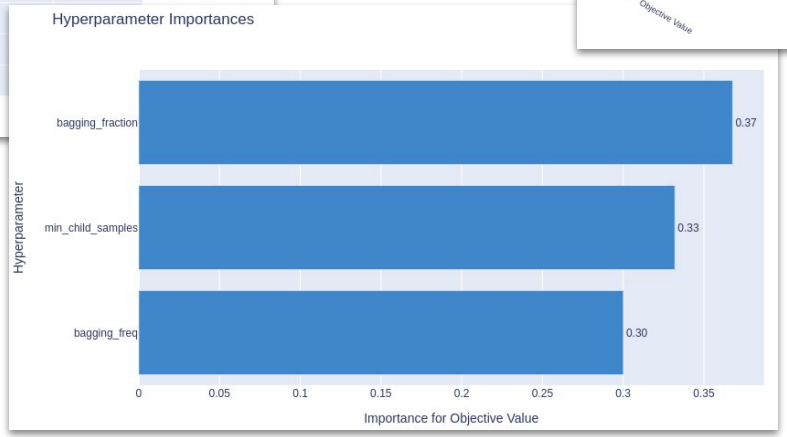
Visualizations



`plot_optimization_history(study)`



`plot_parallel_coordinate(study)`



`plot_param_importances(study)`

Observaciones

- **Easiness of use:** is simple to use and provides flexibility, imperative approach to sampling parameters.
- **Optimization algorithms:** there are a lot of options when it comes to optimization functions right now. However there are some important ones, like Hyperband or BOHB missing.
- **Maintenance:** active contribution and release updates (latest: 3.0.2 - Dec-1-2022).
- **Documentation:** complete and very easy-to-understand documentation on read-the-docs, containing: tutorials with both simple and advanced examples and API Reference with all the functions containing beautiful docstrings.

Library		Optuna	Hyperopt
Ease of use and API		**10**	**9**
Options methods and hyper(hyperparameters)		**9.5**	**8.5**
	Search Space	*10*	*10*
	Optimization Methods	*8*	*8*
	Callbacks	*10*	*6*
	Persisting and restarting	*10*	*10*
Documentation		**10**	**6**
Visualizations		**10**	**3**
Speed and parallelization		**10**	**9**
Features in-common total		**49.5**	**35.5**
Additional Features		**10**	**0**
	Run Pruning	*10*	
	Handling Exceptions	*10*	
With bonus features total		**59.5**	**35.5**
Experimental results*(one problem, one run)		**0.854 / 10**	**0.850 / 6**
Total		**69.5**	**41.5**

Ref: *Optuna vs Hyperopt: Which Hyperparameter Optimization Library Should You Choose?*

Appendix

Basic Concepts: search spaces

In Optuna, we define search spaces using familiar Python syntax including conditionals and loops.

For hyperparameter sampling, Optuna provides the following features:

- optuna.trial.Trial.suggest_categorical() for **categorical** parameters
- optuna.trial.Trial.suggest_int() for **integer** parameters
- optuna.trial.Trial.suggest_float() for **floating** point parameters

With optional arguments of **step** and **log**, we can discretize or take the logarithm of integer and floating point parameters.

Basic Concepts: optimization algorithms

Optuna enables efficient hyperparameter optimization by adopting state-of-the-art algorithms for **sampling** hyperparameters and **pruning** efficiently unpromising trials.

Sampling:

- Grid Search implemented in **GridSampler**
- Random Search implemented in **RandomSampler**
- Tree-structured Parzen Estimator algorithm implemented in **TPESampler**
- CMA-ES based algorithm implemented in **CmaEsSampler**
- Algorithm to enable partial fixed parameters implemented in **PartialFixedSampler**
- Nondominated Sorting Genetic Algorithm II implemented in **NSGAIISampler**
- A Quasi Monte Carlo sampling algorithm implemented in **QMCSampler**

Pruning:

- Median pruning algorithm implemented in **MedianPruner**
- Non-pruning algorithm implemented in **NopPruner**
- Algorithm to operate pruner with tolerance implemented in **PatientPruner**
- Algorithm to prune specified percentile of trials implemented in **PercentilePruner**
- Asynchronous Successive Halving algorithm implemented in **SuccessiveHalvingPruner**
- Hyperband algorithm implemented in **HyperbandPruner**
- Threshold pruning algorithm implemented in **ThresholdPruner**

References

- https://optuna.readthedocs.io/en/stable/
- https://optuna.org/
- https://medium.com/optuna/scaling-up-optuna-with-ray-tune-88f6ca87b8c7
- https://neptune.ai/blog/optuna-vs-hyperopt