Optimizing Hyperparameters the right Way

Efficiently exploring the parameter-search through Bayesian Optimization with skopt in Python. TL;DR: my hyperparameters are always better than yours.

Richard Michael · Follow

Published in Towards Data Science

7 min read · Oct 1, 2020

Explore vast canyons of the problem space efficiently — Photo by Fineas Anton on Unsplash

In this post, we will build a machine learning pipeline using multiple optimizers and use the power of Bayesian Optimization to arrive at the **most optimal configuration for all our parameters.** All we need is the sklearn Pipeline and Skopt.

You can use your favorite ML models, as long as they have a sklearn wrapper (looking at you XGBoost or NGBoost).

About Hyperparameters

The critical point for finding the best models that can solve a problem are *not* just the models. We need to **find the optimal parameters** to make our model work optimally, given the dataset. This is called finding or searching hyperparameters. For example, we would like to implement a Random Forest in practice and its documentation states:

```
class sklearn.ensemble.RandomForestClassifier(n_estimators=100, *,
criterion='gini', max_depth=None, min_samples_split=2,
min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features='auto', max_leaf_nodes=None, ...
```

All of those parameters can be explored. This can include all possible number of estimators (n_estimators) in the forest from 1 to 10.000, you may try to split using *{"gini", "entropy"}*, or the maximum depth of your trees is another integer and there are many, many more options. **Each** of those **parameters can influence your model**'s performance and worst of all most of the time **you do not know the right configuration** when you're starting out with a new problem-set.

The Jack-Hammer aka Grid-Search

The brute-force way to find the optimal configuration is to perform a grid-search for example using sklearn's GridSearchCV. This means that you try out all possible combinations of parameters on your model.
On the bright side, you might find the desired values. The problem is that the **runtime is horrible** and over all **grid-searching does not scale well**. For every new parameter you want to try out, you will test all of the other previously specified parameters also.
This approach is very uninformative and feels like being blind in our problem-space and testing out all the possible buttons and configurations. In contrast to this is our understanding of hyperparameters:

> *We have the **intuition that some picks of parameters are more informative than others.***

Some parameters just make more sense and once we know that some ranges of parameters work better than others. Further once we know what direction works, we don't have to explore all values in the wrong domain.

We don't need to test for *all* parameters — especially not those of which we know that they are far off.
One step in the right direction is randomized search like RandomizedSearchCV, where we pick the parameters randomly while moving in the right direction.

Better Bayesian Search

Our tool of choice is BayesSearchCV. This approach uses stepwise Bayesian Optimization to explore the most promising hyperparameters in the problem-space.
Very briefly, Bayesian Optimization finds the minimum to an objective function in large problem-spaces and is very applicable to continuous values. To do this it uses Gaussian Process regression on the objective function under the hood. A thorough mathematical introduction can be found in [2]. In our case the **objective function** is to arrive at **the best model output given the model-parameters** that we specify.
The Bayesian Optimization approach gives the benefit that we can give a much **larger range of possible values**, since over time we automatically explore the most promising regions and discard the not so promising ones.
Plain grid-search would need ages to stupidly explore all possible values.
Since we move much more effectively, we can allow for a much larger playing field. Let's look at an example.

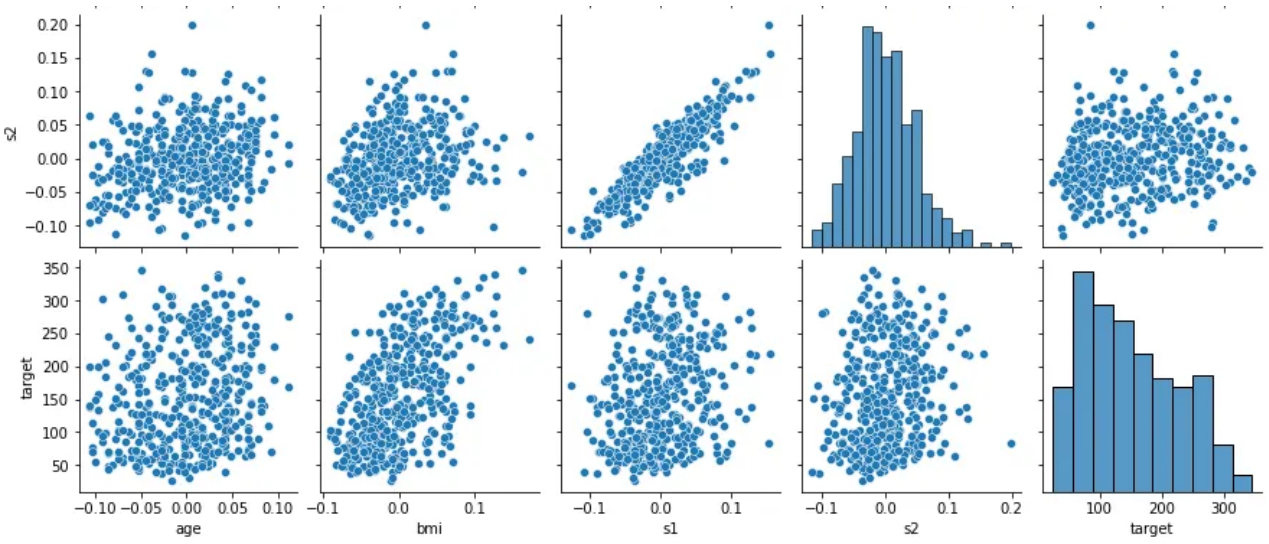
The Problemset

Today we use the diabetes dataset from sklearn for ease of use.
This saves us the trouble of loading and cleaning our data and the features are already neatly encoded.

```
diabetes_data = load_diabetes()
diabetes_df = pd.DataFrame(data=np.c_[diabetes_data['data'], diabetes_data['target']],
                           columns=diabetes_data['feature_names'] + ['target'])
diabetes_df
```

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	target
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019908	-0.017646	151.0
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068330	-0.092204	75.0
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.002592	0.002864	-0.025930	141.0
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022692	-0.009362	206.0
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031991	-0.046641	135.0
...	...	...	...	...	...	...	...	...	...	...	...

We have a set of encoded columns from age, sex, bmi, blood-pressure and serum values, numerically encoded. Our target value is a measure of the disease progression.
We can see some interactions on the s-columns (blood-serum values), indicating some level of correlation.



To build our pipeline we first split our dataset into training and testing respectively in an 80:20 split.

```
X_train, X_test, y_train, y_test =
train_test_split(diabetes_df.drop(columns="target"),
diabetes_df.target, test_size=0.2, random_state=21)
```

Build a Pipeline

We need three elements to build a pipeline: (1) the models to be optimized, (2) the sklearn Pipeline object, and (3) the skopt optimization procedure.

First, we choose **two boosting models**: AdaBoost and GradientBoosted regressors and for each we **define a search space over crucial hyperparameters**. Any other regressor from the depth of the sklearn library would do, but boosting might win you the next hackathon (…its 2015 isn't it?)
The search-space is a dictionary with the
key-value pair := { 'model__*parameter*' : skopt.space.Object}.
For each parameter we set a space from the skopt library in the range that we want. Categorical values are also included by passing them as a list of strings (see Categorical below):

```
ada_search = {
    'model': [AdaBoostRegressor()],
    'model__learning_rate': Real(0.005, 0.9, prior="log-uniform"),
    'model__n_estimators': Integer(1, 1000),
    'model__loss': Categorical(['linear', 'square',
'exponential'])
}

gb_search = {
    'model': [GradientBoostingRegressor()],
    'model__learning_rate': Real(0.005, 0.9, prior="log-uniform"),
    'model__n_estimators': Integer(1, 1000),
    'model__loss': Categorical(['ls', 'lad', 'quantile'])
}
```

Second, we select over which regression model to pick through another model, this is our pipeline element, where both optimizers (adaboost and gradientboost) come together for selection:

```
pipe = Pipeline([('model', GradientBoostingRegressor())])
```

Third, we optimize over our search-space. For this we invoke the BayesSearchCV. We also specify how the optimizer should call our search-space. In our case that is 100 invocations. Then we **fit the pipeline with a simple skopt .fit()** command:

```
opt = BayesSearchCV(
    pipe,
    [(ada_search, 100), (gb_search, 100)],
    cv=5
)

opt.fit(X_train, y_train)
```

After the fitting is done, we can then ask for the optimal found parameters. This includes using the score function on the unseen test-data.

```
print(f"validation score: {opt.best_score_}")
print(f"test score: {opt.score(X_test, y_test)}")
print(f"best parameters: {str(opt.best_params_)}")
```

```
validation score: 0.46989003593474976
test score: 0.34078180831257776
best parameters: OrderedDict([('model', AdaBoostRegressor(base_estimator=None, learning_rate=0.06407566831734247,
                loss='linear', n_estimators=259, random_state=None)), ('model__learning_rate', 0.06407566831734
```

We can see the validation and test-score as well as the parameters of the best fit. The model with the best results is the AdaBoost model with a linear loss and 259 estimators at a learning-rate of 0.064. Neat.
From the output we can also see that the optimizer uses a GP for optimization under the hood:

```
opt.optimizer_results_[0]
```

```
        -0.46116175, -0.46193422, -0.45870089, -0.46150441, -0.4333234 ])
        models: [GaussianProcessRegressor(alpha=1e-10, copy_X_train=True,
                        kernel=1**2 * Matern(length_scale=[1, 1, 1, 1], nu=2.5) + WhiteKernel(noise_level=1),
                        n_restarts_optimizer=2, noise='gaussian',
                        normalize_y=True, optimizer='fmin_l_bfgs_b'
```

GP Minimization

For illustrative purposes we can also call GP minimization specifically — we'll use the AdaBoost regressor for that — only focusing on a numerical hyperparameter space.

```
ada = AdaBoostRegressor(random_state=21)

# numerical space
space  = [Real(0.005, 0.9, "log-uniform", name='learning_rate'),
          Integer(1, 1000, name="n_estimators")]
```

The major change here is that we have to **define an objective function** over which we have to optimize. In order to see how the model performs over the parameter-space we use the named_args decorator.

```
@use_named_args(space)
def objective(**params):
    ada.set_params(**params)
    return -np.mean(cross_val_score(ada, X_train, y_train, cv=5,
n_jobs=-1,

scoring="neg_mean_absolute_error"))
```

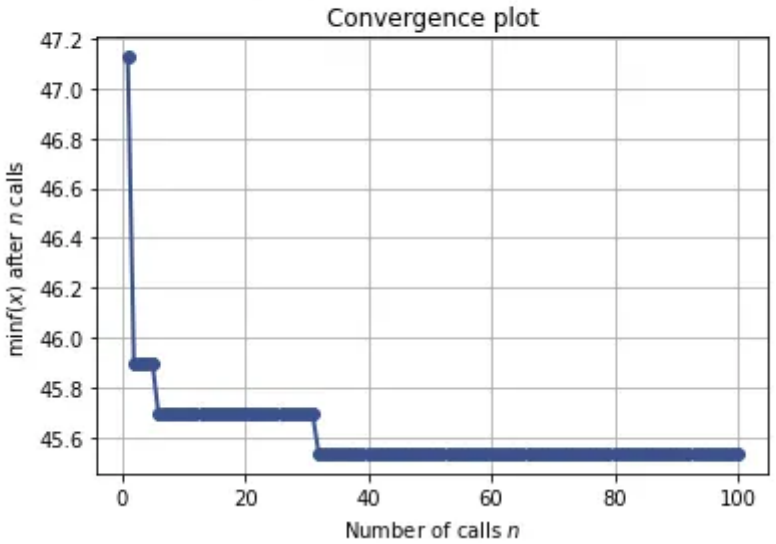
Therefore our optimization is the negative mean score that we get from the cross-validated fit while using the negative mean absolute error. Other scoring functions might suit you better.
We then use GP minimization to fit the most optimal parameters for our regressor. `gp_minimize(objective, space, n_calls=100, random_state=21)`

Visualize the problem space — post-optimization

Using GP optimization directly allows us to plot convergence over the minimization process.

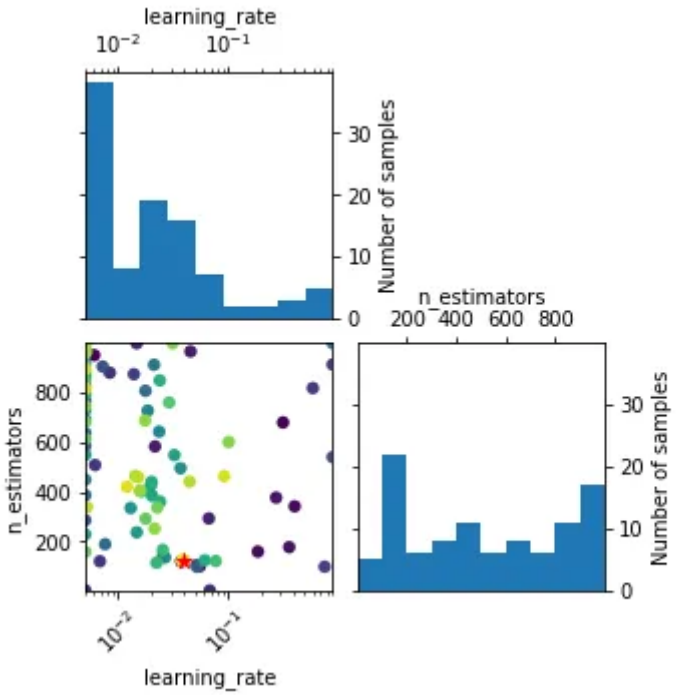
```
[28] plot_convergence(ada_gp)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe64710dda0>
```



Convergence of GP minimization while finding the optimal hyperparameters of the AdaBoost regressor with respect to the target column in the dataset.

We can see that the min in the function value has already been reached after around 40 iterations.
The last excellent feature is **visualizing the explored problem space**. Since we used only numerical input at this point, we can then evaluate the problem space by plotting it using skopt, like so:



Visualizing the invoked learning_rate and n_estimator values over the problem space as histograms. The problem-space is displayed as a scatterplot such that each GP-run (out of the 100) uncovers one function value in that space. The minimum is marked by the red star in the scatterplot.

From the figure above we can see that a lot of exploration was done on the lower end of our learning-rate spectrum, and the peaks of the tried and tested estimator number was 200 and over 800.
The scatterplot paints a picture of how difficult the problem space is that we are trying to optimize — with the minimum marked with a red star. Each GP-minimize run uncovers a function value, given the parameter that it has as input. Though we have reached convergence not a lot of values are evident in the region around the minimum and increasing the number of evaluations can be considered.

Now that we have the best model given our parameter space we can implement that model accordingly and go into its specific analysis.

Conclusion

We have shown that we can explore a vast space of possible model-parameters in an efficient way, saving both time and computational resources. We do this by formulating the search for hyperparameters as an objective function for which we find the optimal values using Bayesian Optimization. There is no need to test useless parameters anymore by trying out every single one of them.
As you have seen skopt is a library that offers a lot in the realm of optimizations and I encourage you to use them in your daily ML engineering.

Happy Exploring!

The complete code can be found in a Notebook here:

Data Science Machine Learning Scikit Learn Optimization Hyperparameter Tuning

ub is home to over 50 million developers working together to host and review code, manage...

github.com

References

1. Marc Claesen, Bart De Moor. *Hyperparameter Search in Machine Learning*. 2015 on arXiv.

I. Frazier. *A Tutorial on Bayesian Optimization*. 2018 on arXiv.

ptimize contributors (BSD License). *Scikit-learn hyperpar ai wrapper*.

Follow

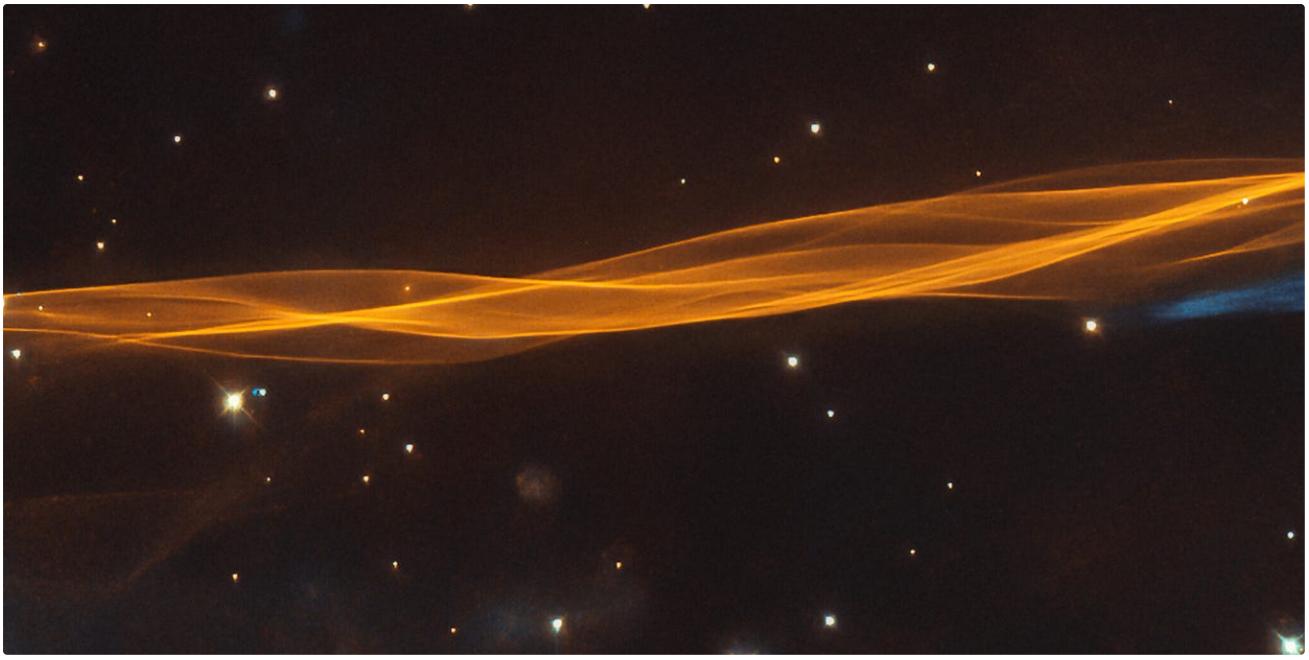
Written by Richard Michael

76 Followers · Writer for Towards Data Science

I am a Data Scientist and M.Sc. student in Bioinformatics at the University of Copenhagen. You can find more content on my weekly blog http://laplaceml.com/blog

---

More from Richard Michael and Towards Data Science

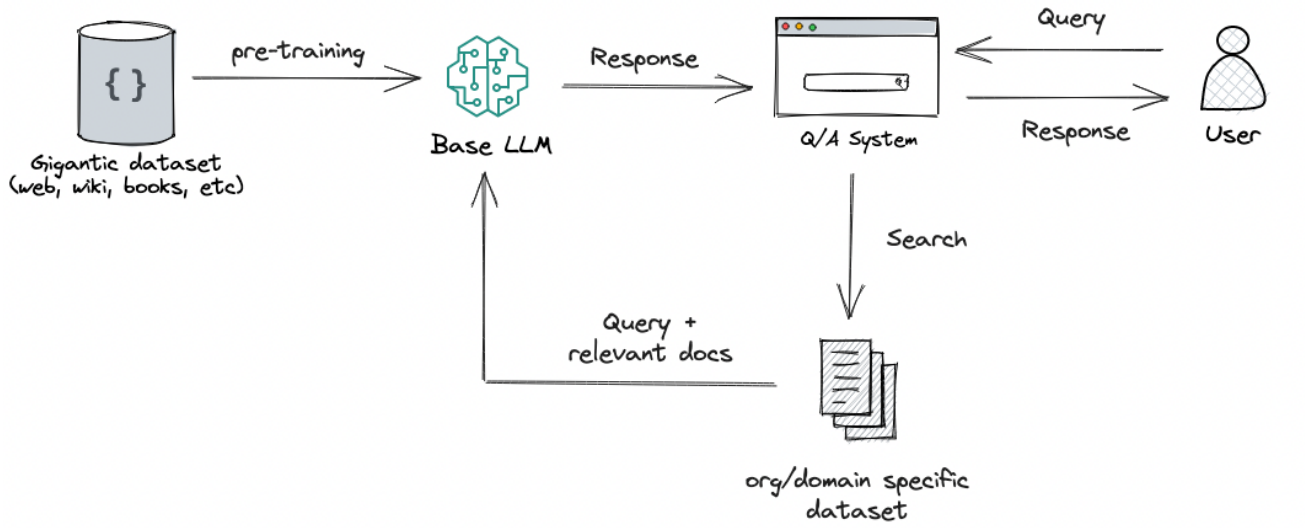


Richard Michael in Towards Data Science

Infinitely Wide Neural-Networks | Neural Tangents Explained

use Python and JAX to efficiently build infinitely wide networks for deeper network insights on your finite machine.

11 min read · Dec 14, 2020



Heiko Hotz in Towards Data Science

RAG vs Finetuning — Which Is the Best Tool to Boost Your LLM Application?

The definitive guide for choosing the right method for your use case

· 19 min read · Aug 25

1.6K 16

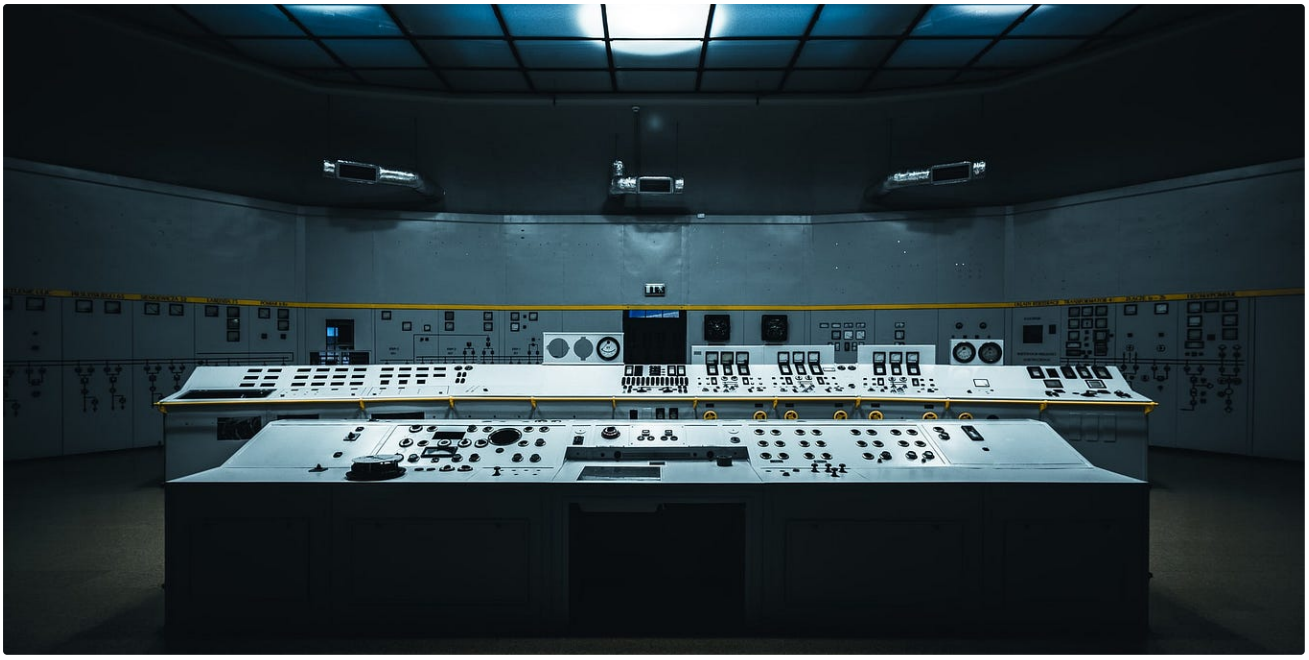
Cameron R. Wolfe, Ph.D. in Towards Data Science

Advanced Prompt Engineering

What to do when few-shot learning isn't enough...

· 17 min read · Aug 7

1K 8

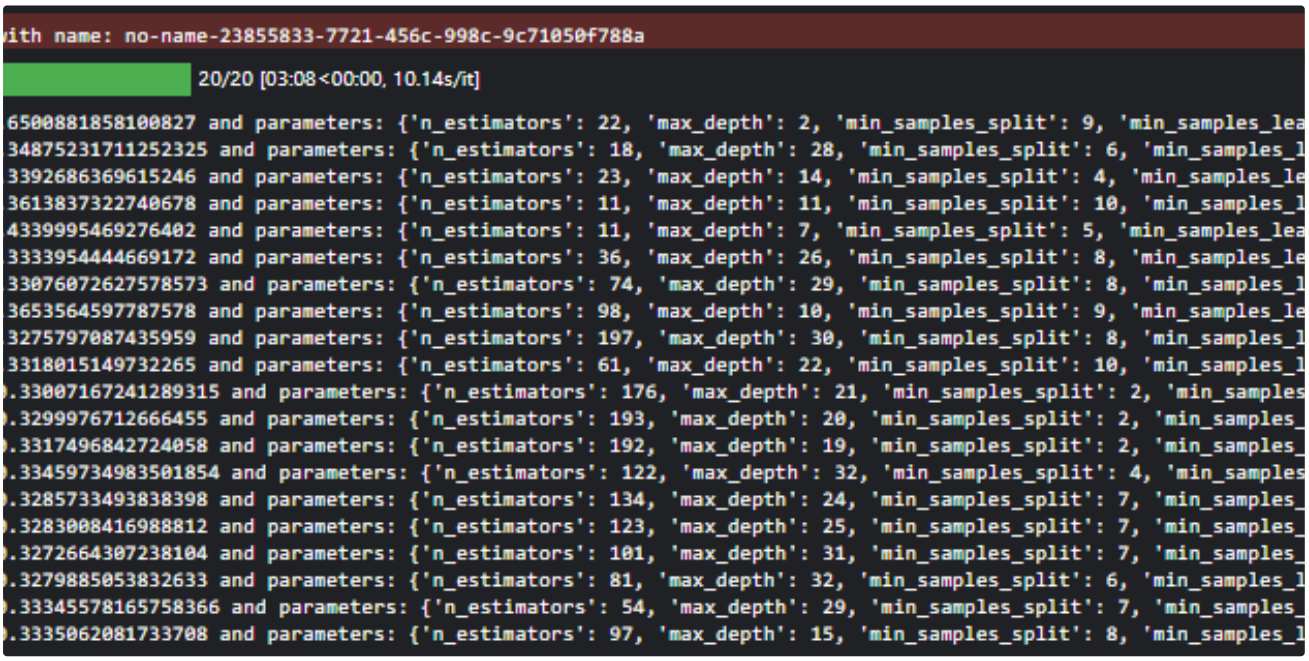


Richard Michael in Towards Data Science

3 Probabilistic Frameworks You should know | The Bayesian Toolkit

Build better Data Science workflows with probabilistic programming languages and counter the shortcomings of classical ML.

4 min read · Jul 21, 2020



Mustafa Germec in AI Mind

Hyperparameter optimization of random forest model using Optuna for a regression problem

8 min read · Jul 21

129

Gülsüm Budakoğlu in MLearning.ai

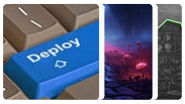
Hyper-parameter Tuning Through Grid Search and Optuna

Giving Information about Hyper-parameter Tuning Methods and Code Analysis of Grid Search ann Optuna

5 min read · Mar 26

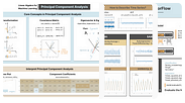
156 2

Lists



Predictive Modeling w/ Python

20 stories · 369 saves



Practical Guides to Machine Learning

10 stories · 415 saves

Natural Language Processing

596 stories · 210 saves

New_Reading_List

174 stories · 101 saves

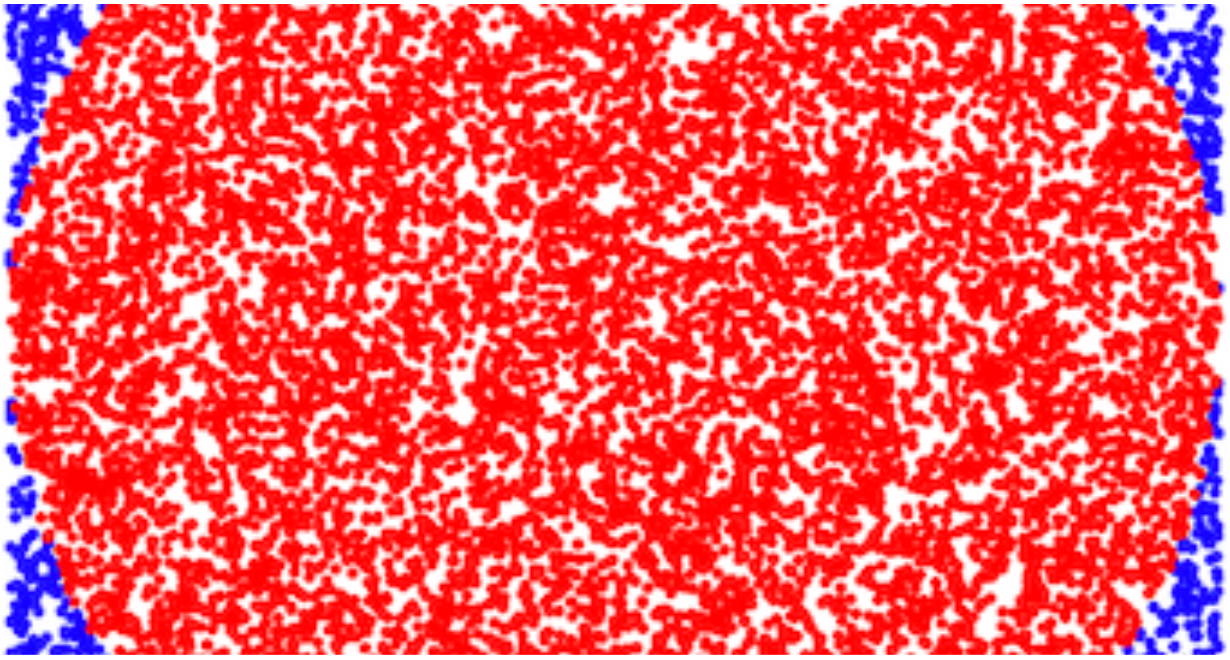
Farzad Mahmoodinobar in Towards Data Science

Hyperparameter Optimization — Intro and Implementation of Grid Search, Random Search and Bayesian...

Most common hyperparameter optimization methodologies to boost machine learning outcomes.

· 10 min read · Mar 13

136



Wendy Hu

Monte Carlo Simulation with Python

Introduction

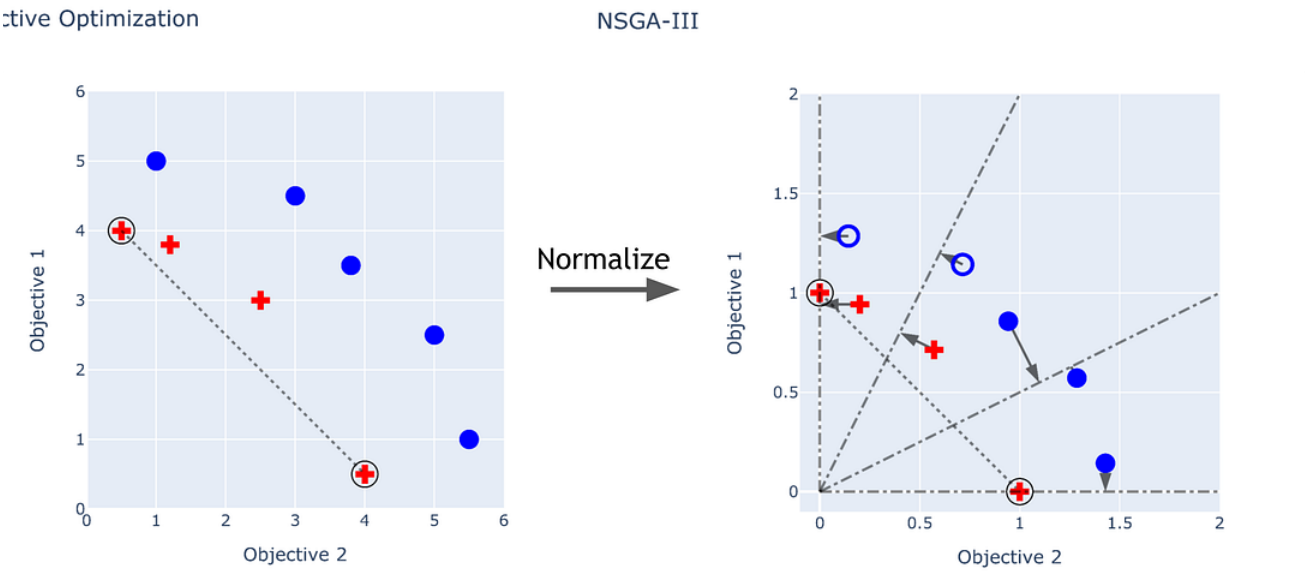
5 min read · Apr 5

Maxim Gusarov in CodeX

Do I need to tune logistic regression hyperparameters?

Aren't we over-committed to optimizing the data science work we do? We are often trying to find the best combination of x, y, z variables...

9 min read · Apr 9, 2022



Shinichi Hemmi in Optuna

NSGA-III: New Sampler for Many Objective Optimization

Optuna v3.2 was released at the end of May. In this article, we will explain one of its new features, the NSGAIIISampler. NSGA-III [1,4] is...

6 min read · Jul 14

16

See more recommendations