Examples of local minima and maxima (Image by Author — Mayrhofen 2019)

Hyperparameter Tuning Methods - Grid, Random or Bayesian Search?

A practical guide to hyperparameter optimization using three methods: grid, random and bayesian search (with skopt)

Maria Gorodetski · Follow
Published in Towards Data Science
4 min read · Aug 28, 2021

When I was working on my last project, I got a new chunk of data **after** I trained

the first version of the model. That day, I felt a little lazy and tried to retrain my model with the new data using the same model type and hyperparameters. Unfortunately, it didn't follow my expectations. Instead of improving, due to the increase in data amount, it performed slightly worse. The added data was distributed differently than the data I had before, and its amount was not negligible comparing to the origin, contrary to my expectation.

So, I found myself in an unknown territory, without any hunch of which hyperparameters I should use. I tried some options manually, but there were too many possible combinations, and I had trouble managing my experiments. At that point, I decided to dive deeper into the hyperparameter tuning world.

What should you expect from this post?

1. Introduction to hyperparameter tuning.
2. Explanation about hyperparameter search methods.
3. Code examples for each method.
4. Comparison and conclusions.

For all the examples in the post, I used Kaggles' heart-attack-analysis-prediction-dataset.

```python
import pandas as pd

df_heart = pd.read_csv('heart.csv')

lables=['<45', '45-60', '60+']
df_heart['age_bins']=pd.cut(x=df_heart['age'],bins=[25,45,60,100], labels=lables, include_lo

numeric_var = ['age', 'trtbps', 'chol', 'thalachh', 'oldpeak']
categorical_var = ['sex','cp','fbs','restecg','exng','slp','caa','thall', 'age_bins']

df_heart[categorical_var] = df_heart[categorical_var].astype('category')

X = df_heart.drop('output', axis=1)
y = df_heart['output']
```

dataset.py hosted with ❤ by GitHub view raw

I have prepared a simple pipeline that is used in all the examples.

```
from sklearn.pipeline import Pipeline #sklearn==0.23.2
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import make_column_transformer
from lightgbm import LGBMClassifier

tuples = list()

tuples.append((Pipeline([
        ('scaler', StandardScaler()),
    ]), numeric_var))

tuples.append((Pipeline([
        ('onehot', OneHotEncoder()),
    ]), categorical_var))

preprocess = make_column_transformer(*tuples)

pipe = Pipeline([
    ('preprocess', preprocess),
    ('classifier', LGBMClassifier())
])
```

What is hyperparameter tuning, and why is it important?

Hyperparameters are the variables of the algorithm that control its whole behavior. It affects its speed, resolution, structure, and eventually performance. Sometimes it has only a small effect, but in others, it is crucial. A good example is the **learning rate**. When it is too large, the learning isn't sensitive enough, and the model results are unstable. But when it is too small, the model has trouble learning and might stuck.

When the algorithm has many parameters, it is very hard to try all the possible combinations to find the best set. For that reason, we would like to do hyperparameter tuning efficiently and in a manageable way.

Types of Hyperparameter Search

There are three main methods to perform hyperparameters search:

1. Grid search
2. Randomized search

3. Bayesian Search

Grid Search

The basic way to perform hyperparameter tuning is to try all the possible combinations of parameters. For example, if you want to tune the **learning_rate** and the **max_depth**, you need to specify all the values you think will be relevant for the search. Then, when we run the hyperparameter tuning, we try all the combinations from both lists.

In the following example, I tried to find the best values for **learning_rate** (5 values), **max_depth** (5 values), and **n_estimators** (5 values) — 125 iterations in Total.

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    "classifier__learning_rate": [0.0001, 0.0005, 0.001, 0.01, 0.1],
    "classifier__n_estimators": [100, 300, 600, 800, 1000],
    "classifier__max_depth": [ 4, 20, 100, 250, 400]
}

# grid
reg_grid = GridSearchCV(pipe,
                        param_grid=param_grid,
                        cv=5,
                        n_jobs=8,
                        scoring='roc_auc'
                       )

model_grid = reg_grid.fit(X, y)
```

gridsearch.py hosted with ❤ by GitHub view raw

Random Search

Unlike the Grid Search, in randomized search, only part of the parameter values are tried out. The parameter values are sampled from a given **list** or specified **distribution**. The number of parameter settings that are sampled is given by **n_iter**. Sampling without replacement is performed when the parameters are presented as a list (like the grid search). But if the parameter is given as a distribution, sampling **with replacement** is used (recommended).

The advantage of randomized search, in my experience, is that you can extend

your search limits without increasing the number of iterations (time-consuming). Also, you can use it to find narrow limits to continue a thorough search in a smaller area.

In the following example, I used parameters distributions for sampling with replacement.

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import loguniform, randint

n_iter = 70

param_grid = {
    "classifier__learning_rate": loguniform(1e-4, 0.1),
    "classifier__n_estimators": randint(100,1000),
    "classifier__max_depth": randint(4, 400)
}

# Random
reg_rand = RandomizedSearchCV(pipe,
                          param_distributions=param_grid,
                          n_iter=n_iter,
                          cv=5,
                          n_jobs=8,
                          scoring='roc_auc',
                          random_state=123)

model_rand = reg_rand.fit(X, y)
```

Bayesian Search

The main difference between Bayesian search and the other methods is that the tuning algorithm optimizes its parameter selection in each round according to the previous round score. Thus, instead of randomly choosing the next set of parameters, the algorithm optimizes the choice, and likely reaches the best parameter set faster than the previous two methods. Meaning, this method chooses only the relevant search space and discards the ranges that will most likely not deliver the best solution. Thus, it can be beneficial when you have a large amount of data, the learning is slow, and you want to **minimize the tuning time**.

Same as in the random search example, I used in this example parameter

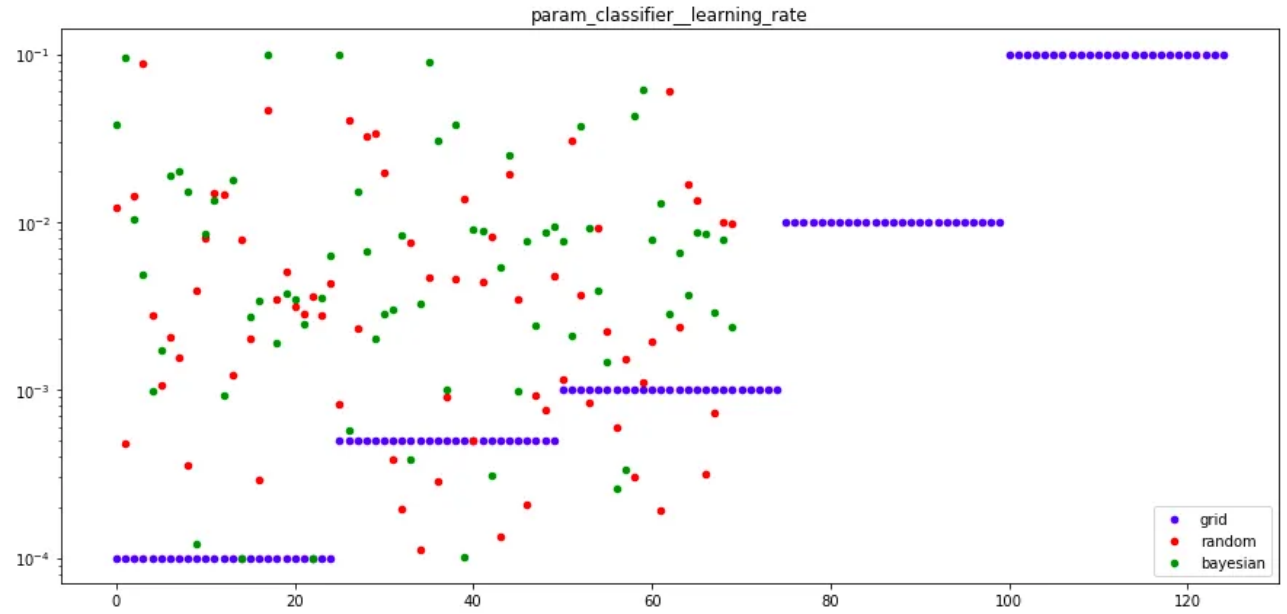
distributions for sampling:

```
from skopt import BayesSearchCV

# Bayesian
n_iter = 70

param_grid = {
    "classifier__learning_rate": (0.0001, 0.1, "log-uniform"),
    "classifier__n_estimators": (100,  1000) ,
    "classifier__max_depth": (4, 400)
}

reg_bay = BayesSearchCV(estimator=pipe,
                    search_spaces=param_grid,
                    n_iter=n_iter,
                    cv=5,
                    n_jobs=8,
                    scoring='roc_auc',
                    random_state=123)

model_bay = reg_bay.fit(X, y)
```

Visualization of parameter search — learning rate

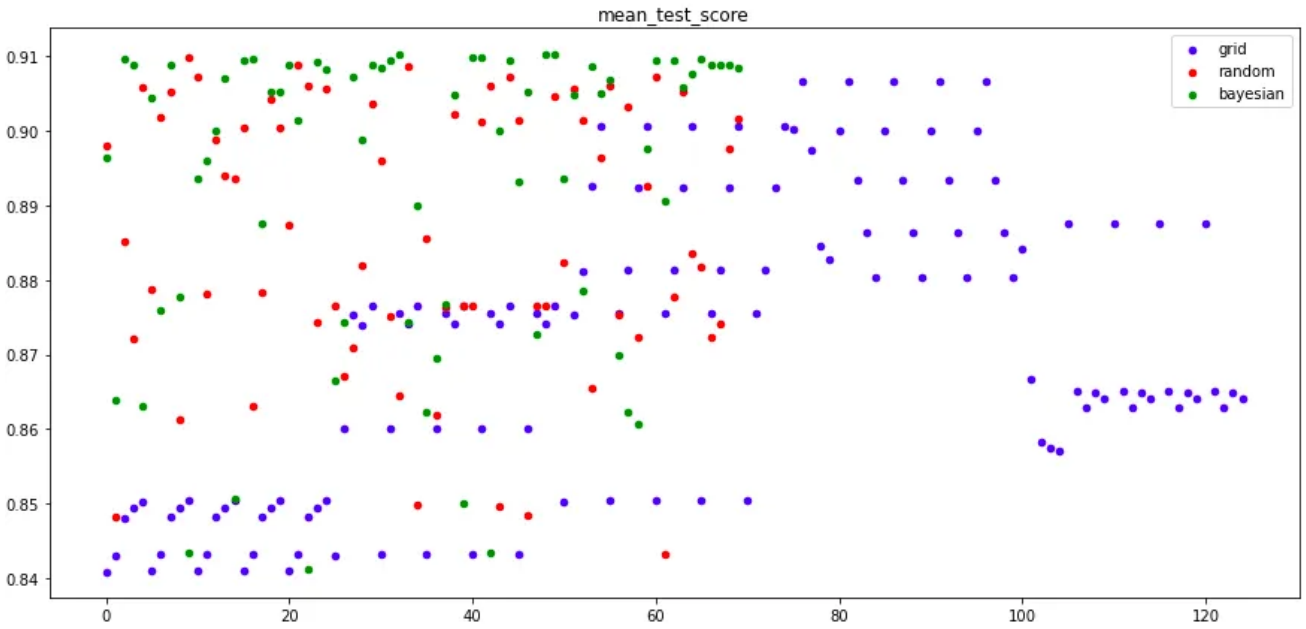
```
param = 'param_classifier__learning_rate'

grid = model_grid.cv_results_[param]
rand = model_rand.cv_results_[param]
bay = model_bay.cv_results_[param]

fig = plt.figure(figsize=(15, 7))

ax = plt.gca()
ax.scatter(np.arange(len(grid)), grid.data, c='b', s=20, label='grid');
ax.scatter(np.arange(len(rand)), rand.data, c='r', s=20, label='random');
ax.scatter(np.arange(len(bay)), bay, c='g', s=20, label='bayesian');
ax.set_yscale('log')

plt.legend();
plt.title(param);
```

lr_visualisation.py hosted with ❤ by GitHub view raw



The best learning rate parameter in this comparison is 0.008 (found by the bayesian search).

Visualization of the mean score for each iteration



We can see that the bayesian search outperforms the other methods by a little. This effect is much more noticeable in larger datasets and more complex models.

Discussion and Conclusions

I ran the three search methods on the same parameter ranges. The grid-search ran 125 iterations, the random and the bayesian ran 70 iterations each. This data

set is relatively simple, so the variations in scores are not that noticeable. Still, the random search and the bayesian search performed better than the grid-search, with fewer iterations. The **bayesian** search found the hyperparameters to achieve the **best score**.

Python Machine Learning Data Science Hyperparameter Tuning Bayesian Optimization

ng excellent atascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f.

Good luck!

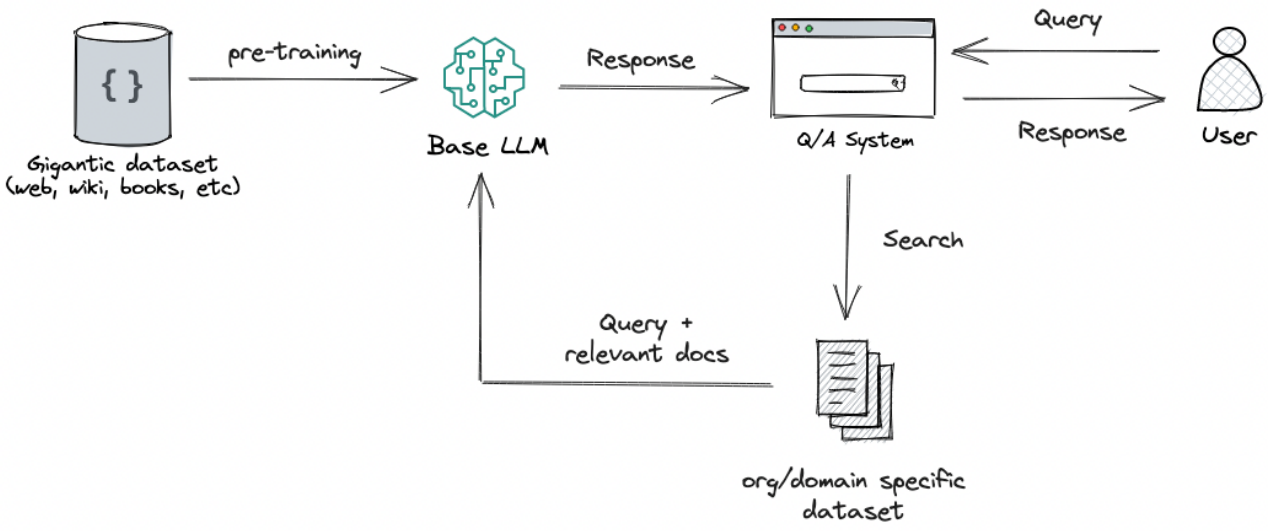
Follow

Written by Maria Gorodetski

30 Followers · Writer for Towards Data Science

I'm a data scientist at DayTwo, where I combine my programming and bioinformatical skills for solving healthcare problems.

More from Maria Gorodetski and Towards Data Science



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

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

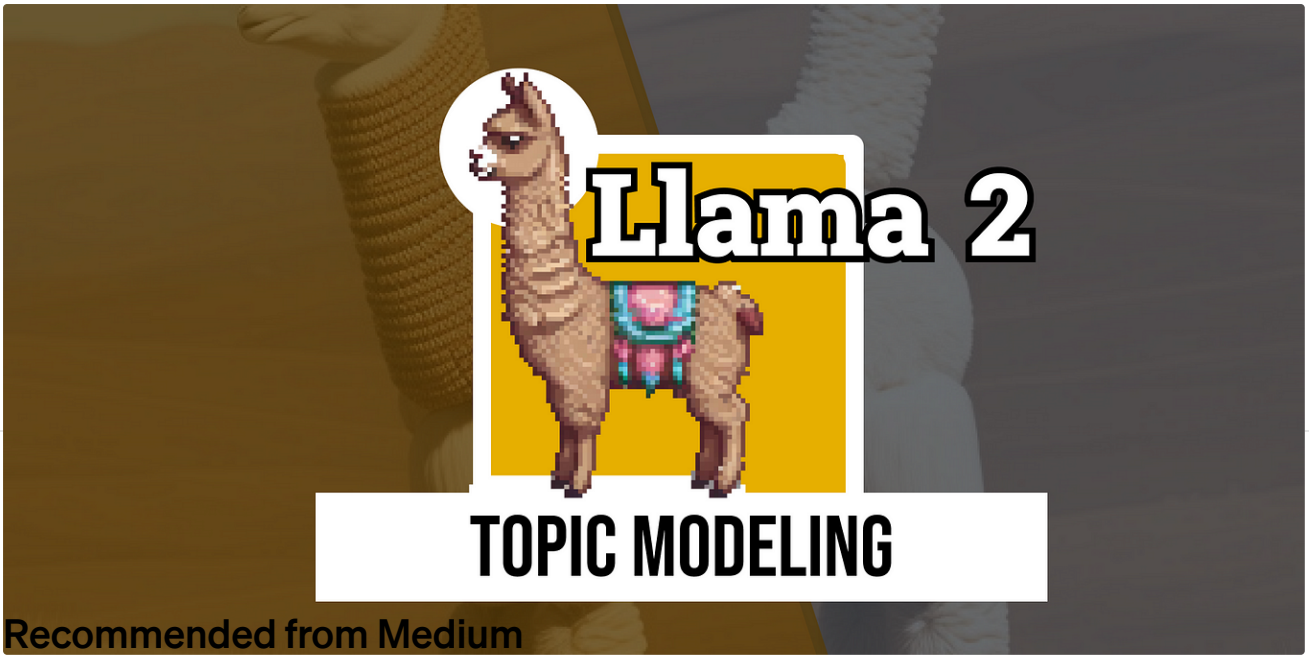
Giuseppe Scalamogna in Towards Data Science

New ChatGPT Prompt Engineering Technique: Program Simulation

A potentially novel technique for turning a ChatGPT prompt into a mini-app.

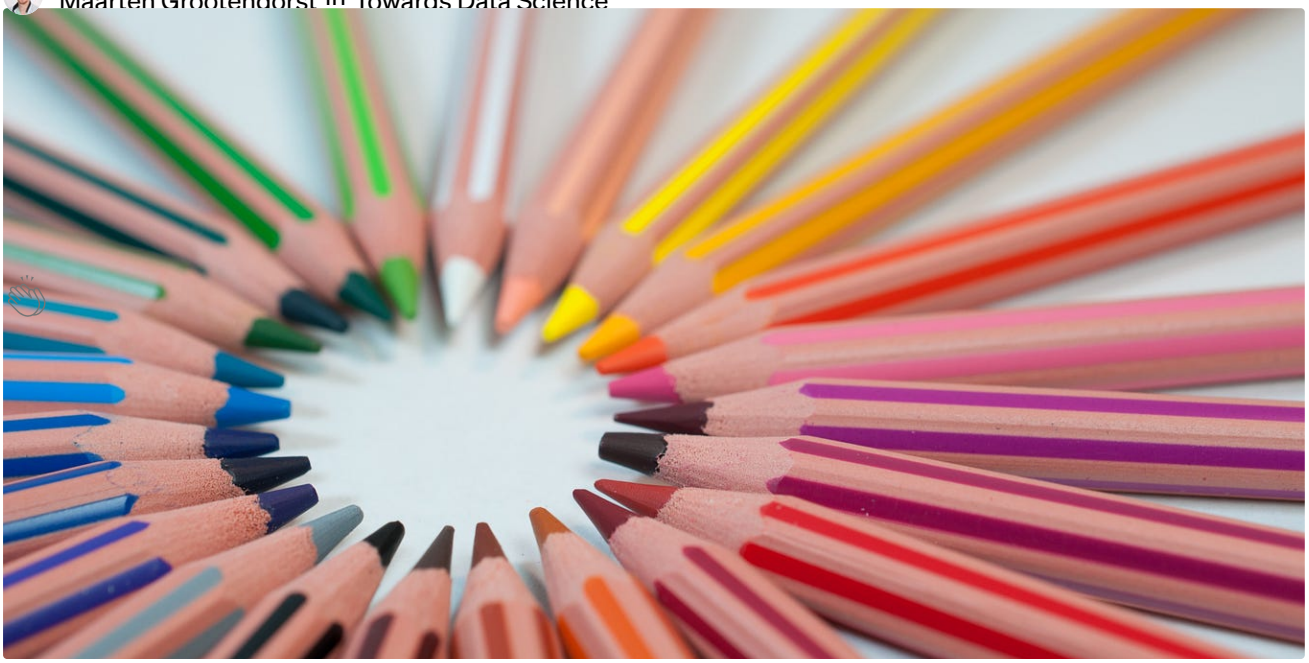
9 min read · Sep 4

742 6



Recommended from Medium

Maarten Grootendorst in Towards Data Science



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

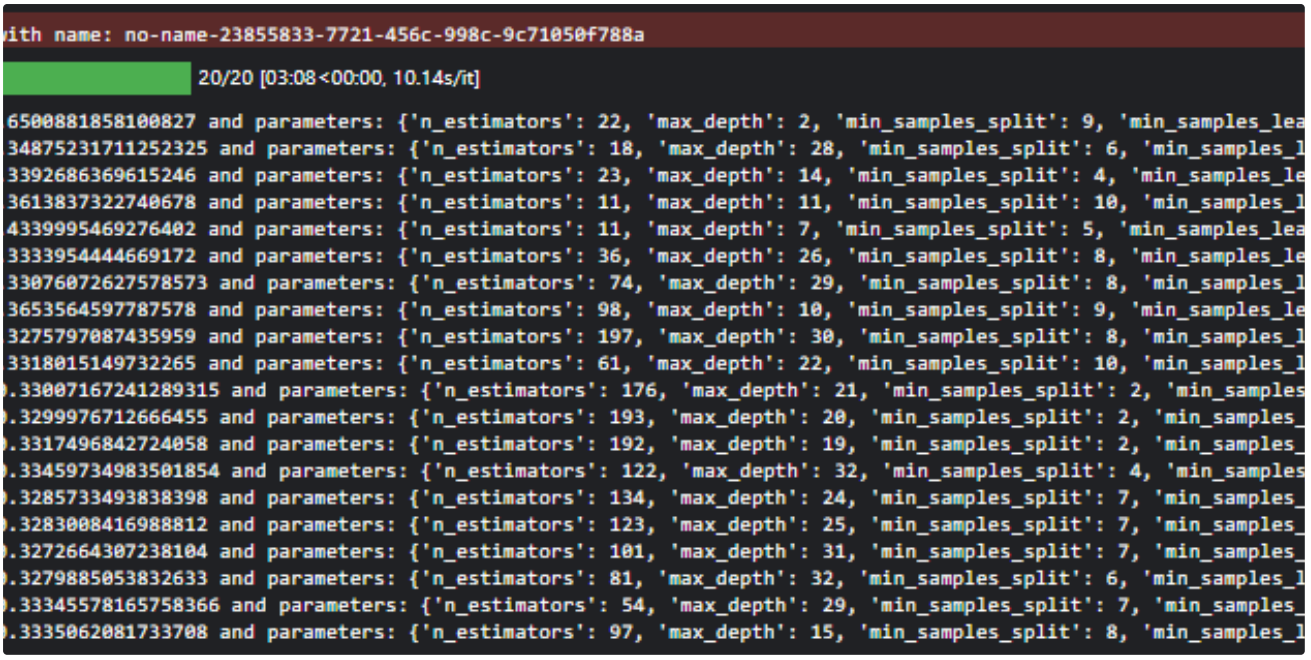
156 2

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.

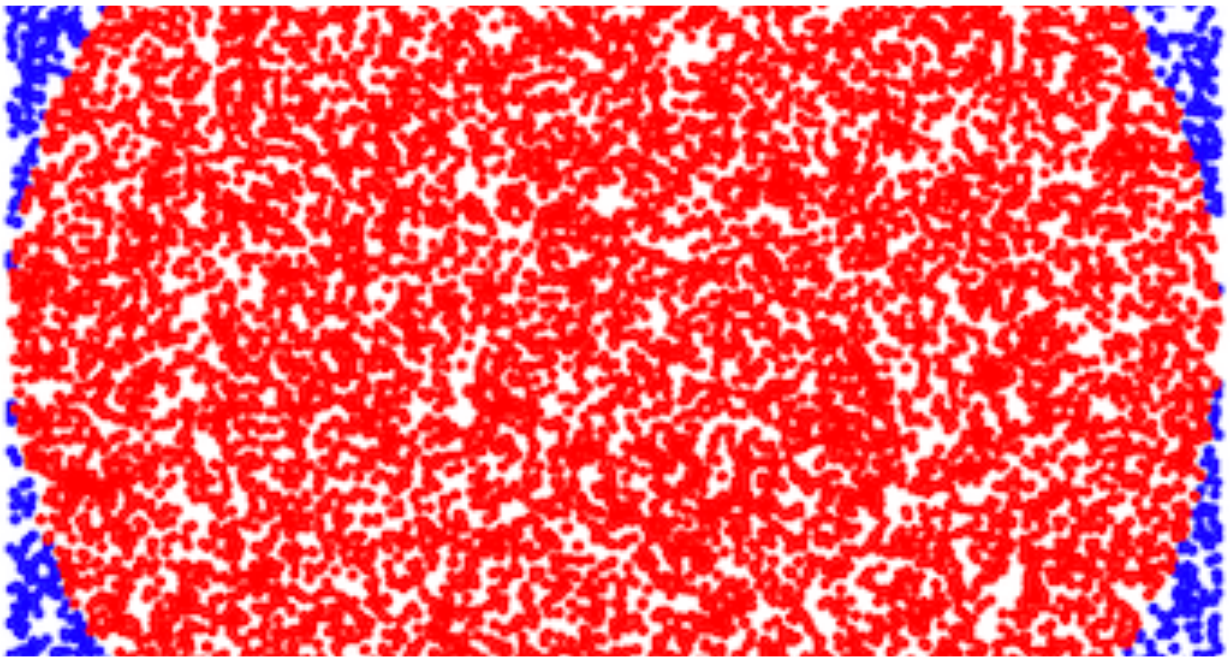
· 10 min read · Mar 13



Mustafa Germec in AI Mind

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

8 min read · Jul 21

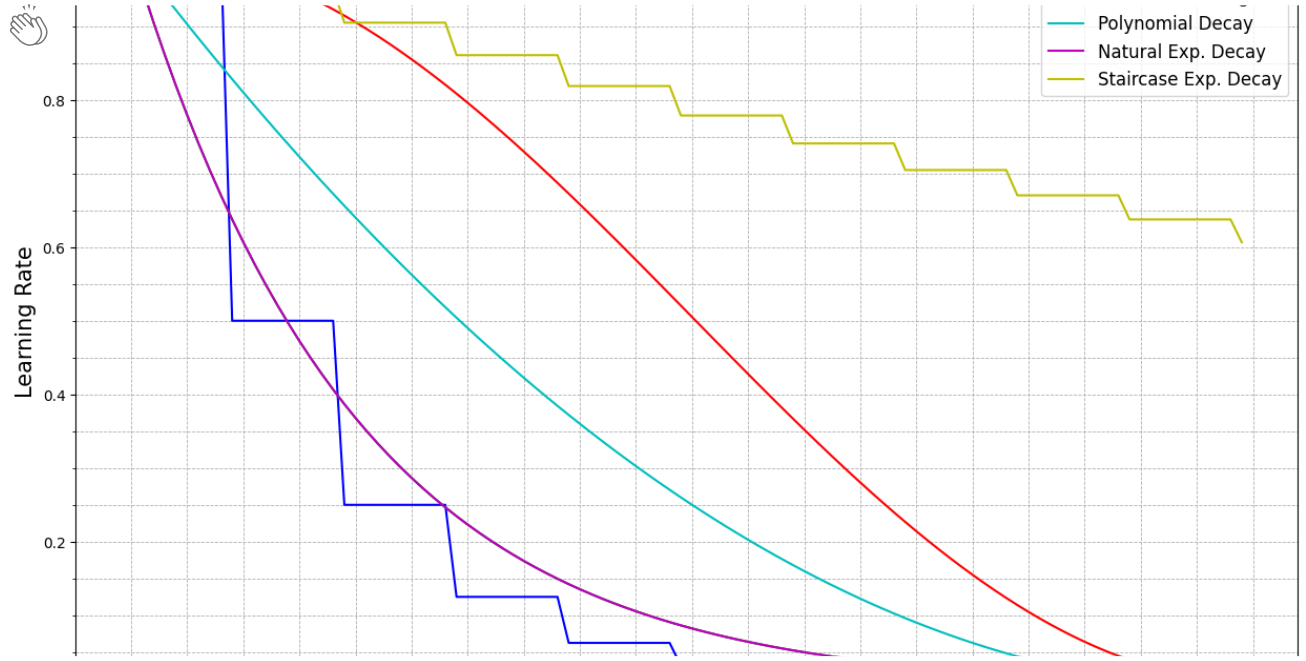


Wendy Hu

Monte Carlo Simulation with Python

Introduction

5 min read · Apr 5



Théo Martin

A (Very Short) Visual Introduction to Learning Rate Schedulers (With Code)

Learning rate is one of the most important hyperparameters in the training of neural networks, impacting the speed and effectiveness of the...

6 min read · Jul 10

51

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