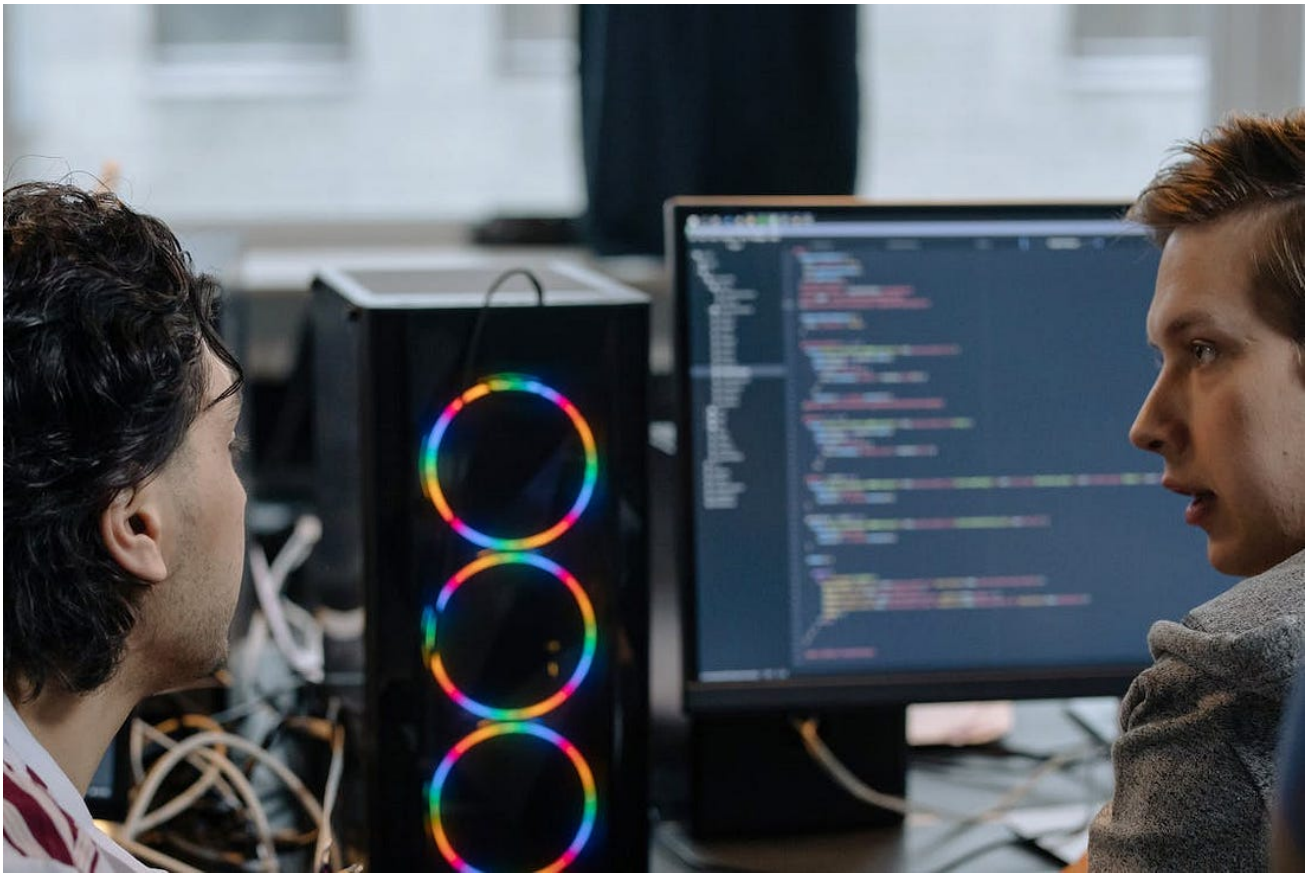
# Optimizing XGBoost: A Guide to Hyperparameter Tuning

RITHP · Follow

4 min read · Jan 17





Hyperparameter tuning is important because the performance of a machine learning model is heavily influenced by the choice of hyperparameters. Choosing the right set of hyperparameters can lead to better model performance, while choosing the wrong set can lead to poor performance. Additionally, when a model has too many hyperparameters, it can be difficult or even impossible to find the best set of hyperparameters manually.

## Different types of hyperparameters in XGBoost

In XGBoost, there are two main types of hyperparameters: tree-specific and learning task-specific.

#### Tree-specific hyperparameters control the construction and complexity of the decision trees:

- `max_depth` : maximum depth of a tree. Deeper trees can capture more complex patterns in the data, but may also lead to overfitting.
- `min_child_weight` : minimum sum of instance weight (hessian) needed in a child. This can be used to control the complexity of the decision tree by preventing the creation of too small leaves.
- `subsample` : percentage of rows used for each tree construction. Lowering this value can prevent overfitting by training on a smaller subset of the data.
- `colsample_bytree` : percentage of columns used for each tree construction. Lowering this value can prevent overfitting by training on a subset of the features.

#### Learning task-specific hyperparameters control the overall behavior of the model and the learning process:

- `eta` (also known as learning rate): step size shrinkage used in updates to prevent overfitting. Lower values make the model more robust by taking smaller steps.
- `gamma` : minimum loss reduction required to make a further partition on a leaf node of the tree. Higher values increase the regularization.
- `lambda` : L2 regularization term on weights. Higher values increase the regularization.
- `alpha` : L1 regularization term on weights. Higher values increase the regularization.

## Overview of different techniques for tuning hyperparameters

*Grid search* is one of the most widely used techniques for hyperparameter tuning. It involves specifying a set of possible values for each hyperparameter, and then training and evaluating the model for each combination of hyperparameter

values. Grid search is simple to implement and can be efficient when the number of hyperparameters and their possible values is small. However, it can become computationally expensive as the number of hyperparameters and possible values increases.

```
import xgboost as xgb
from sklearn.model_selection import GridSearchCV

# Define the hyperparameter grid
param_grid = {
    'max_depth': [3, 5, 7],
    'learning_rate': [0.1, 0.01, 0.001],
    'subsample': [0.5, 0.7, 1]
}

# Create the XGBoost model object
xgb_model = xgb.XGBClassifier()

# Create the GridSearchCV object
grid_search = GridSearchCV(xgb_model, param_grid, cv=5, scoring='accuracy')

# Fit the GridSearchCV object to the training data
grid_search.fit(X_train, y_train)

# Print the best set of hyperparameters and the corresponding score
print("Best set of hyperparameters: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

*Random search* is a variation of grid search that randomly samples from the set of possible hyperparameter values instead of trying all combinations. This can be more efficient than grid search because it does not need to evaluate all possible combinations. However, it can still be computationally expensive, especially when the number of hyperparameters and possible values is large.

```
import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV
import scipy.stats as stats

# Define the hyperparameter distributions
param_dist = {
    'max_depth': stats.randint(3, 10),
    'learning_rate': stats.uniform(0.01, 0.1),
```

```
    'subsample': stats.uniform(0.5, 0.5),
    'n_estimators':stats.randint(50, 200)
}

# Create the XGBoost model object
xgb_model = xgb.XGBClassifier()

# Create the RandomizedSearchCV object
random_search = RandomizedSearchCV(xgb_model, param_distributions=param_dist

# Fit the RandomizedSearchCV object to the training data
random_search.fit(X_train, y_train)

# Print the best set of hyperparameters and the corresponding score
print("Best set of hyperparameters: ", random_search.best_params_)
print("Best score: ", random_search.best_score_)
```

*Bayesian optimization* is a more sophisticated technique that uses Bayesian methods to model the underlying function that maps hyperparameters to the model performance. It tries to find the optimal set of hyperparameters by making smart guesses based on the previous results. Bayesian optimization is more efficient than grid or random search because it attempts to balance exploration and exploitation of the search space. It can also deal with the cases of large number of hyperparameters and large search space. However, it can be more difficult to implement than grid search or random search and may require more computational resources.

```
import xgboost as xgb
from hyperopt import fmin, tpe, hp

# Define the hyperparameter space
space = {
    'max_depth': hp.quniform('max_depth', 2, 8, 1),
    'learning_rate': hp.loguniform('learning_rate', -5, -2),
    'subsample': hp.uniform('subsample', 0.5, 1)
}

# Define the objective function to minimize
def objective(params):
    xgb_model = xgb.XGBClassifier(**params)
    xgb_model.fit(X_train, y_train)
    y_pred = xgb_model.predict(X_test)
    score = accuracy_score(y_test, y_pred)
    return {'loss': -score, 'status': STATUS_OK}
```

```
# Perform the optimization
best_params = fmin(objective, space, algo=tpe.suggest, max_evals=100)
print("Best set of hyperparameters: ", best_params)
```

There are several techniques that can be used to tune the hyperparameters of an XGBoost model including grid search, random search and Bayesian optimization. Grid search is simple to implement but can be computationally expensive when the number of hyperparameters and possible values is large. Random search is more efficient than grid search but still can be computationally expensive. Bayesian optimization is the most sophisticated technique, which balances

Statistics Machine Learning Data Science Artificial Intelligence

nt and require more computational resources.

Data Analysis

Follow

## Written by RITHP

81 Followers

Data scientist with a passion for using data-driven insights to solve complex problems. On a mission to make data science accessible and impactful for all.

### More from RITHP

RITHP

## The main parameters in XGBoost and their effects on model performance

Parameter tuning is an essential step in achieving high model performance in machine learning. By adjusting the values of the various...

4 min read · Jan 29

54 2

RITHP

## XGBoost and imbalanced datasets: Strategies for handling class imbalance

XGBoost is a powerful and widely used gradient boosting library for machine learning. It is known for its high predictive power and...

3 min read · Dec 28, 2022

26 1

RITHP

## Logistic Regression for Feature Selection: Selecting the Right Features for Your Model

Logistic regression is a popular classification algorithm that is commonly used for feature selection in machine learning. It is a simple...

4 min read · Jan 1

16


RITHP

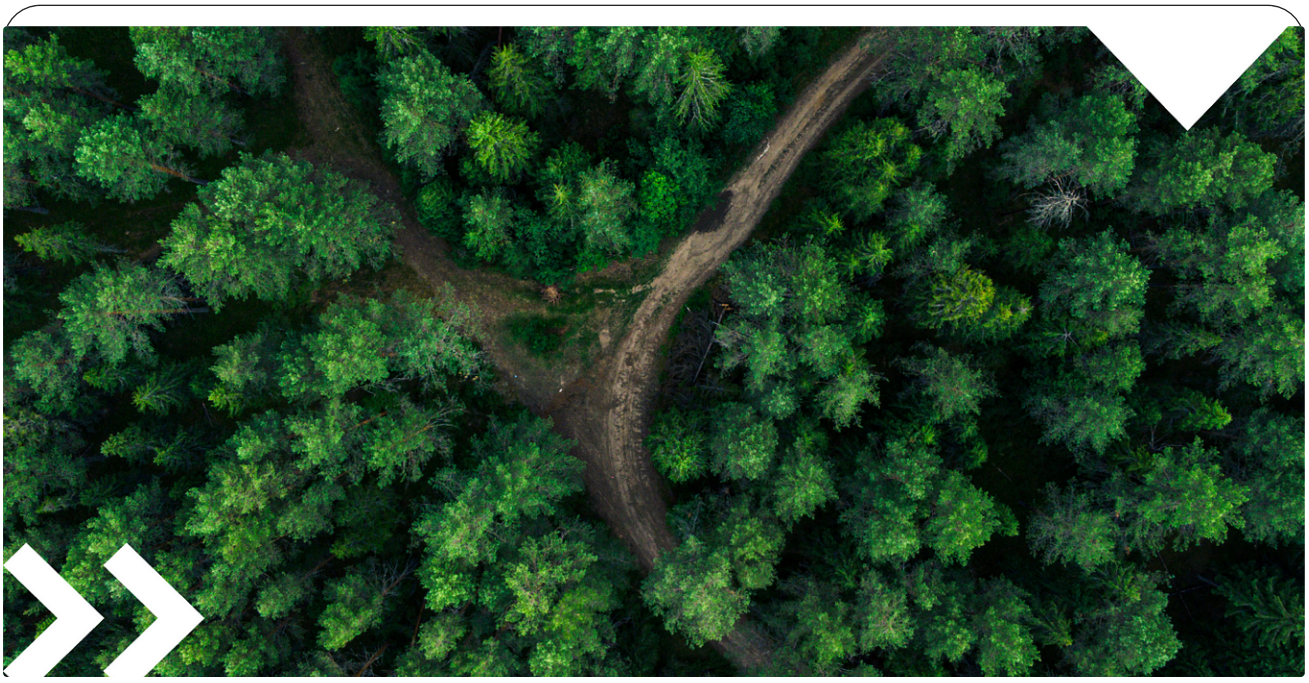
## Choosing the Right Size: A Look at the Differences between Upsampling and Downsampling Methods

In machine learning, having a sufficient amount of data is critical for building accurate models. A large dataset allows for the extraction...

4 min read · Feb 19

42

## Recommended from Medium



Ido Finder in AppsFlyer Engineering

## Building a tunable and configurable custom objective function for XGBoost

Learn how to take your XGBoost custom objective functions to the next level.

11 min read · Mar 16

139 3

Gülsüm Budakoğlu in MLearning.ai

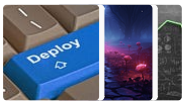
## Hyper-parameter Tuning Through Grid Search and Optuna

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

5 min read · Mar 26

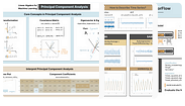
156 2

### Lists




**Predictive Modeling w/ Python**

20 stories · 369 saves



**Practical Guides to Machine Learning**

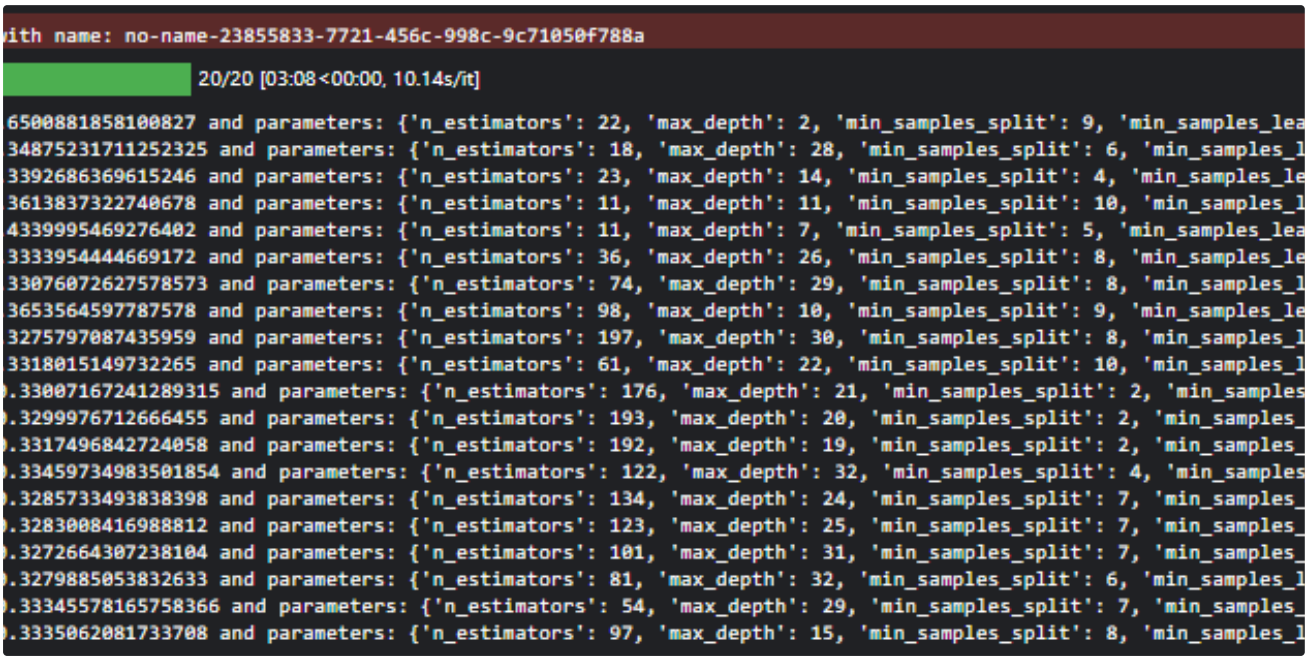
10 stories · 415 saves

## Natural Language Processing

596 stories · 210 saves

## ChatGPT prompts

24 stories · 360 saves




Mustafa Germec in AI Mind

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

8 min read · Jul 21

129

Rukshan Pramoditha in Towards Data Science

# Addressing Overfitting 2023 Guide — 13 Methods

Your one-stop place to learn 13 effective methods to prevent overfitting in machine learning and deep learning models

· 14 min read · Nov 23, 2022



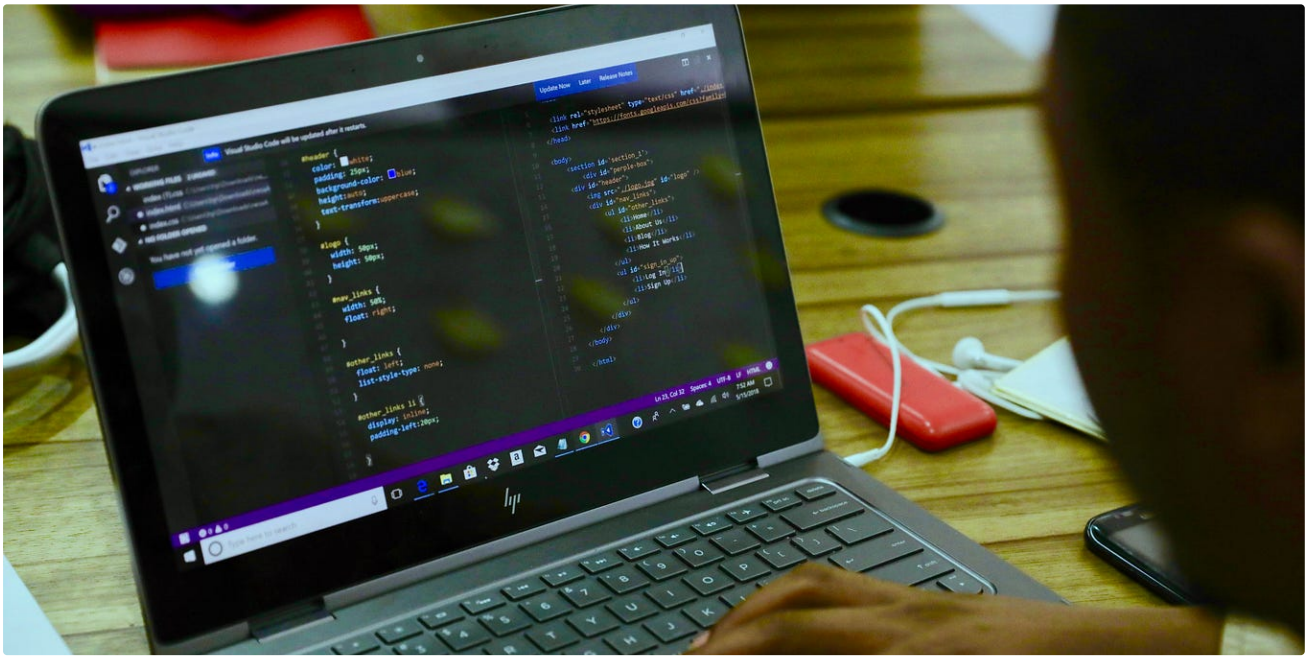
Shunya Vichar

## XGBoost: Giving Your Models an Incremental Boost, One Gradient at a Time!

Let’s unravel the idea of incremental learning in XGBoost through a fictional story. Assume in a fictional world inspired by “The Matrix,”...

4 min read · May 18

33



Peter Maina

## A Guide to Building Your First Machine Learning Model in Python: An Introduction

Building machine learning models is COOL! Your first model should not be complex and fancy. No! It is your first attempt; have fun doing...

6 min read · Aug 22

8 1

See more recommendations