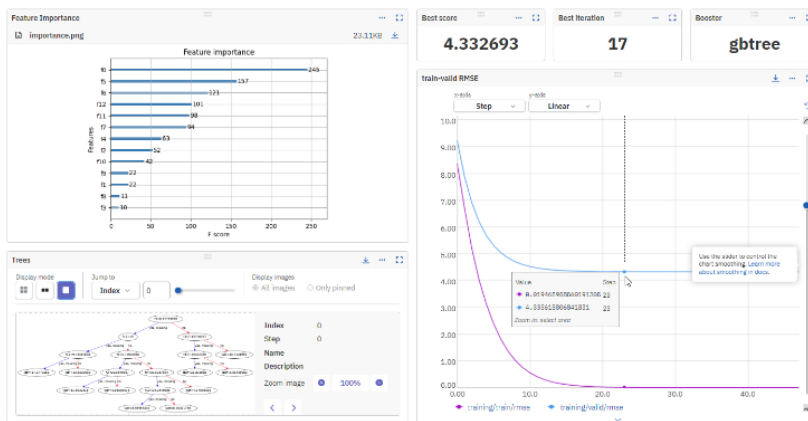


[PyTorch Lightning](#)[Sacred](#)[scikit-learn](#)[Seaborn](#)[skorch](#)[TensorBoard](#)[TensorFlow](#)[Weights & Biases](#)[XGBoost](#)[Integration guide](#)[API reference](#)[ZenML](#)[Community integrations](#)[Demo: Using multiple integrations together](#)

tabular-data

## XGBoost integration guide

[Open in Colab](#)

XGBoost is an optimized distributed library that implements machine learning algorithms under the Gradient Boosting framework. With the Neptune-XGBoost integration, the following metadata is logged automatically:

- Metrics
- Parameters
- The pickled model
- The feature importance chart
- Visualized trees
- Hardware consumption metrics
- stdout and stderr streams
- Training code and Git information

[See in Neptune](#)[Code examples](#)

### Before you start

- Sign up at [neptune.ai/register](https://neptune.ai/register).
- Create a [project](#) for storing your metadata.
- Ensure that you have at least version 1.3.0 of XGBoost installed:

```
pip      conda
pip install xgboost>=1.3.0
```

### Installing the integration

[Install integration only](#)[Install Neptune + integration](#)

To use your preinstalled version of Neptune together with the integration:

**pip**

```
pip install -U neptune-xgboost
```

**conda**

```
conda install -c conda-forge neptune-xgboost
```

[Table of contents](#)[Before you start](#)[Installing the integration](#)[XGBoost logging example](#)[Logging metadata during training](#)[Exploring results in Neptune](#)[More options](#)[Changing the base namespace](#)[Using Neptune callback with CV function](#)[Working with scikit-learn API](#)[Manually logging metadata](#)

If you want to log visualized trees after training (recommended), additionally install Graphviz:

```
pip conda
pip install -U graphviz
```

#### Note

The above installation is only for the pure Python interface to the Graphviz software. You need to install Graphviz separately.

For installation help, see the [Graphviz documentation](#).

If you'd rather follow the guide without any setup, you can [run the example in Colab](#).

## XGBoost logging example

This example walks you through logging metadata as you train your model with XGBoost.

You can log metadata during training with `NeptuneCallback`.

### Logging metadata during training

1. Start a run:

```
import neptune
run = neptune.init_run()
```

2. Initialize the Neptune callback:

```
from neptune.integrations.xgboost import NeptuneCallback
neptune_callback = NeptuneCallback(run=run, log_tree=[0, 1, 2, 3])
```

3. Prepare your data, parameters, and so on.

4. Pass the callback to the `train()` function and train the model:

```
xgb.train(
    params=model_params,
    dtrain=dtrain,
    callbacks=[neptune_callback],
)
```

5. To stop the connection to Neptune and sync all data, call the `stop()` method:

```
run.stop()
```

6. Run your script as you normally would.

To open the run, click the Neptune link that appears in the console output.

**Example link:** <https://app.neptune.ai/common/xgboost-integration/e/XGBOOST-84>

### Exploring results in Neptune

In the run view, you can see the logged metadata organized into folder-like namespaces.

Name	Description
<code>booster_config</code>	All parameters for the booster.
<code>early_stopping</code>	<code>best_score</code> and <code>best_iteration</code> (logged if early stopping was activated)
<code>epoch</code>	Epochs (visualized as a chart from first to last epoch).
<code>learning_rate</code>	Learning rate visualized as a chart.
<code>pickled_model</code>	Trained model logged as a pickled file.
<code>plots</code>	Feature importance and visualized trees.
<code>train</code>	Training metrics.
<code>valid</code>	Validation metrics.

[See example in Neptune](#) 

## More options


### Changing the base namespace

By default, the metadata is logged under the namespace `training`.

You can change the namespace when creating the Neptune callback:

```
neptune_callback = NeptuneCallback(  
    run=run,  
    base_namespace="my_custom_name",  
)
```

### Using Neptune callback with CV function

You can use `NeptuneCallback` in the [xgboost.cv](#)  function. Neptune will log additional metadata for each fold in CV.

Pass the Neptune callback to the `callbacks` argument of `lgb.cv()`:

**Core code**    Full script

```
import neptune  
from neptune.integrations.xgboost import NeptuneCallback  
  
# Create run  
run = neptune.init_run()  
  
# Create neptune callback  
neptune_callback = NeptuneCallback(run=run, log_tree=[0, 1, 2, 3])  
  
# Prepare data, params, etc.  
...  
  
# Run cross validation and log metadata to the run in Neptune  
xgb.cv(  
    params=model_params,  
    dtrain=dtrain,  
    callbacks=[neptune_callback],  
)  
  
# Stop run  
run.stop()
```

In the **All metadata** section of the run view, you can see a `fold_n` namespace for each fold in an n-fold CV:


```
fold_n  
├─ booster_config  
├─ pickled_model  
└─ plots  
    ├─ importance  
    └─ trees
```

Namespaces inside the `fold_n` namespace:

Name	Description
<code>booster_config</code>	All parameters for the booster.
<code>pickled_model</code>	Trained model logged as a pickled file.
<code>plots</code>	Feature importance and visualized trees.

[See in Neptune](#) 

## Working with scikit-learn API

You can use `NeptuneCallback` in the [scikit-learn API](#)  of XGBoost.

Pass the Neptune callback while creating the regressor object:

**Core code**    Full script

```

import neptune
from neptune.integrations.xgboost import NeptuneCallback

# Create run
run = neptune.init_run()

# Create neptune callback
neptune_callback = NeptuneCallback(run=run)

# Prepare data, params, etc.
X_train = ...
y_train = ...
model_params = {...}

# Create regressor object and pass the Neptune callback
reg = xgb.XGBRegressor(**model_params, callbacks=[neptune_callback])

# Fit the model
reg.fit(X_train, y_train)

# Stop run
run.stop()

```

[See in Neptune](#) 

## Manually logging metadata

If you have other types of metadata that are not covered in this guide, you can still log them using the Neptune client library.

When you initialize the run, you get a `run` object, to which you can assign different types of metadata in a structure of your own choosing.

```

import neptune

# Create a new Neptune run
run = neptune.init_run()

# Log metrics inside loops
for epoch in range(n_epochs):
    # Your training loop

    run["train/epoch/loss"].append(loss) # Each append() call appends a value
    run["train/epoch/accuracy"].append(acc)


# Track artifact versions and metadata
run["train/images"].track_files("./datasets/images")

# Upload entire files
run["test/preds"].upload("path/to/test_preds.csv")

# Log text or other metadata, in a structure of your choosing
run["tokenizer"] = "regex_tokenizer"

```

### Related

- [Add Neptune to your code](#)
- [What you can log and display](#)
- [Resume a run](#)
- **API reference** >> [XGBoost integration](#)
- [neptune-xgboost repo on GitHub](#) 
- [XGBoost on GitHub](#) 