Published in Towards Data Science

Corey Wade Follow

Nov 10, 2020 · 6 min read ★ · Listen

GETTING STARTED

Getting Started with XGBoost in scikit-learn

Every journey starts with a single boost

This article explains what XGBoost is, why XGBoost should be your go-to machine learning algorithm, and the code you need to get XGBoost up and running in Colab or Jupyter Notebooks. Basic familiarity with machine learning and Python is assumed.

What XGBoost is

In machine learning, ensemble models perform better than individual models with high probability. An ensemble model combines different machine learning models into one. The Random Forest is a popular ensemble that takes the average of many Decision Trees via bagging. Bagging is short for "bootstrap aggregation," meaning that samples are chosen with replacement (bootstrapping), and combined (aggregated) by taking their average.

Boosting is a strong alternative to bagging. Instead of aggregating predictions, boosters turn weak learners into strong learners by focusing on where the individual models (usually Decision Trees) went wrong. In Gradient Boosting, individual models train upon the residuals, the difference between the prediction and the actual results. Instead of aggregating trees, gradient boosted trees learns from errors during each boosting round.

XGBoost is short for "eXtreme Gradient Boosting." The "eXtreme" refers to speed enhancements such as parallel computing and cache awareness that makes XGBoost approximately 10 times faster than traditional Gradient Boosting. In addition, XGBoost includes a unique split-finding algorithm to optimize trees, along with built-in regularization that reduces overfitting. Generally speaking, XGBoost is a faster, more accurate version of Gradient Boosting.

Boosting performs better than bagging on average, and Gradient Boosting is arguably the best boosting ensemble. Since XGBoost is an advanced version of Gradient Boosting, and its results are unparalleled, it's arguably the best machine learning ensemble that we have.

Why XGBoost should be your go-to machine learning algorithm

Starting with the Higgs boson Kaggle competition in 2014, XGBoost took the machine


XGBoost is likely your best place to start when making predictions from tabular data for the following reasons:

- XGBoost is easy to implement in scikit-learn.
- XGBoost is an ensemble, so it scores better than individual models.
- XGBoost is regularized, so default models often don't overfit.
- XGBoost is very fast (for ensembles).
- XGBoost learns form its mistakes (gradient boosting).
- XGBoost has extensive hyperparameters for fine-tuning.
- XGBoost includes hyperparameters to scale imbalanced data and fill null values.

Now that you have a better idea of what XGBoost is, and why XGBoost should be your go-to machine learning algorithm when working with tabular data (as contrasted with unstructured data such as images or text where neural networks work better), let's build some models.

Installing XGBoost

If you're running Colab Notebooks, XGBoost is included as an option. If you're running Anaconda in Jupyter Notebooks, you may need to install it first. Open your terminal and running the following to install XGBoost with Anaconda:

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

If you want to verify installation, or your version of XGBoost, run the following:

```
import xgboost; print(xgboost.__version__)
```

For additional options, check out the XGBoost Installation Guide.

Next, let's get some data to make predictions

Scikit-learn comes with several built-in datasets that you may access to quickly score models. The following code loads the scikit-learn Diabetes Dataset, which measures how much the disease has spread after one year. See the scikit-learn dataset loading page for more info.

```
from sklearn import datasets
X,y = datasets.load_diabetes(return_X_y=True)
```

The measure of how much diabetes has spread may take on continuous values, so we need a machine learning regressor to make predictions. The XGBoost regressor is called **XGBRegressor** and may be imported as follows:

```
from xgboost import XGBRegressor
```

We can build and score a model on multiple folds using cross-validation, which is always a good idea. An advantage of using cross-validation is that it splits the data (5 times by default) for you. First, import **cross_val_score**.

```
from sklearn.model_selection import cross_val_score
```

To use XGBoost, simply put the XGBRegressor inside of cross_val_score along with X, y, and your preferred scoring metric for regression. I prefer the root mean squared error, but this requires converting the negative mean squared error as an additional step.

```
scores = cross_val_score(XGBRegressor(), X, y,
scoring='neg_mean_squared_error')
```


```
# Alternative code to silence potential errors
scores = cross_val_score(XGBRegressor(objective='reg:squarederror'),
X, y, scoring='neg_mean_squared_error')
```

To find the root mean squared error, just take the negative square root of the five scores.

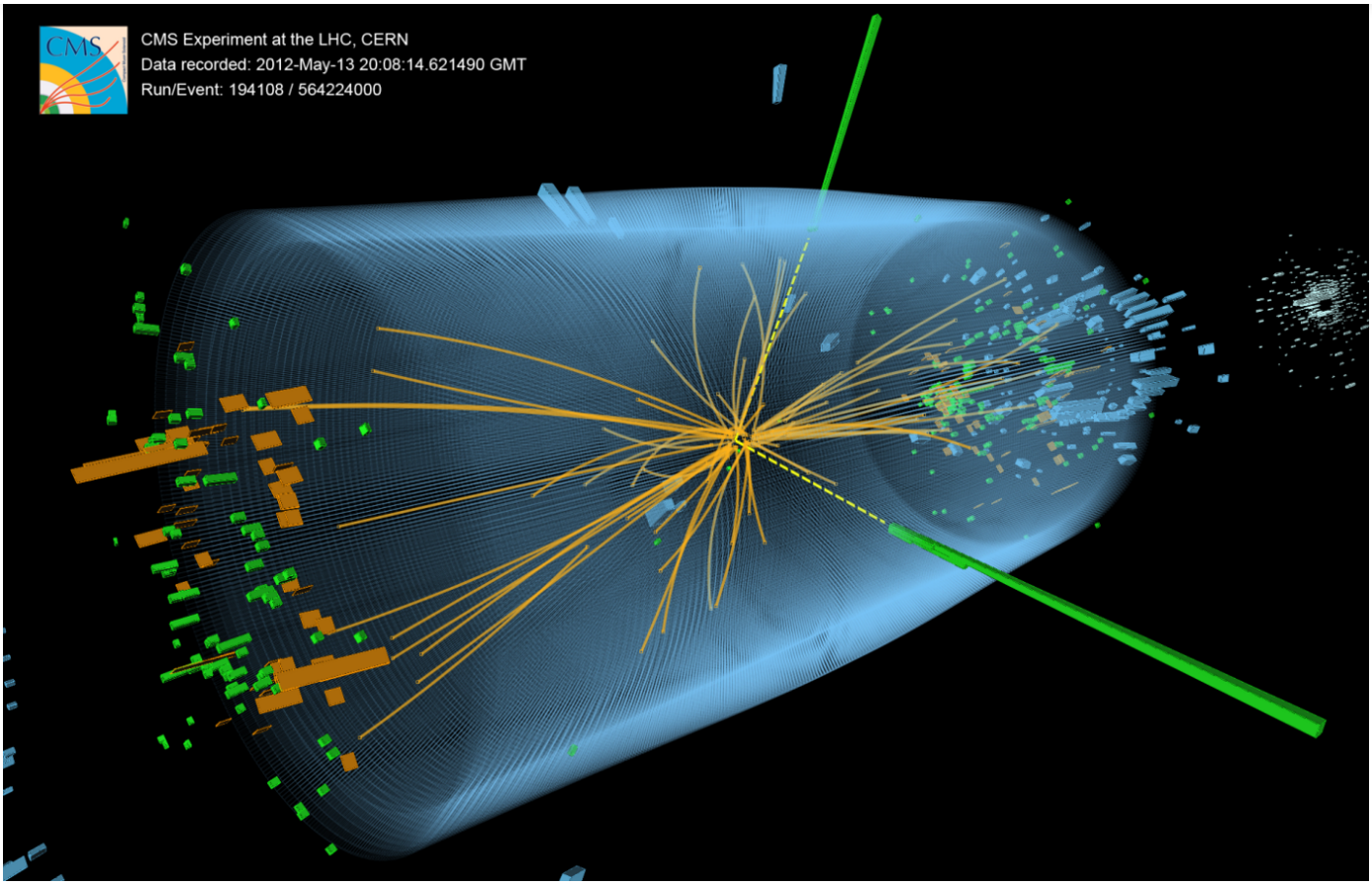
```
(-scores)**0.5
```

Recall that in Python, the syntax x**0.5 means x to the 1/2 power which is the square root.

My Colab Notebook results are as follows.

```
array([56.04057166, 56.14039793, 60.3213523 , 59.67532995, 60.7722925
])
```

If you prefer one score, try **scores.mean()** to find the average.

Expected Higgs boson decay. Photograph produced by CERN. Commons Creative License. Link to wikipedia.

XGBRegressor code

Here is all the code to predict the progression of diabetes using the XGBoost regressor in scikit-learn with five folds.

```
from sklearn import datasets
X,y = datasets.load_diabetes(return_X_y=True)
from xgboost import XGBRegressor
from sklearn.model_selection import cross_val_score
scores = cross_val_score(XGBRegressor(objective='reg:squarederror'),
X, y, scoring='neg_mean_squared_error')
(-scores)**0.5
```

As you can see, XGBoost works the same as other scikit-learn machine learning algorithms thanks to the new scikit-learn wrapper introduced in 2019.

The following url contains a heart disease dataset that may be used to predict whether a patient has a heart disease or not.

```
url =
'https://media.githubusercontent.com/media/PacktPublishing/Hands-On-
Gradient-Boosting-with-XGBoost-and-Scikit-
learn/master/Chapter02/heart_disease.csv'
```

This dataset contains 13 predictor columns like cholesterol level and chest pain. The last column, labeled 'target', determines whether the patient has a heart disease or not. The source of the original dataset is located at the UCI Machine Learning Repository.

Import pandas to read the csv link and store it as a DataFrame, df.

```
import pandas as pd
df = pd.read_csv(url)
```

Since the target column is the last column and this dataset has been pre-cleaned, you can split the data into X and y using index location as follows:

```
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
```

Finally, import the XGBClassifier and score the model using cross_val_score, leaving accuracy as the default scoring metric.

```
from xgboost import XGBClassifier
from sklearn.model_selection import cross_val_score
cross_val_score(XGBClassifier(), X, y)
```


```
array([0.85245902, 0.85245902, 0.7704918 , 0.78333333, 0.76666667])
```

XGBClassifier code

Here is all the code together to predict whether a patient has a heart disease using the XGBClassifier in scikit-learn on five folds:

```
url =
'https://media.githubusercontent.com/media/PacktPublishing/Hands-On-
Gradient-Boosting-with-XGBoost-and-Scikit-
learn/master/Chapter02/heart_disease.csv'

import pandas as pd
df = pd.read_csv(url)
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

from xgboost import XGBClassifier
from sklearn.model_selection import cross_val_score
cross_val_score(XGBClassifier(), X, y)
```

You know understand how to build and score XGBoost classifiers and regressors in scikit-learn with ease.

XGBoost next steps

Getting more out of XGBoost requires fine-tuning hyperparameters. XGBoost consist of many Decision Trees, so there are Decision Tree hyperparameters to fine-tune along with ensemble hyperparameters. Check out this Analytics Vidhya article, and the official XGBoost Parameters documentation to get started.

If you are looking for more depth, my book *Hands-on Gradient Boosting with XGBoost and scikit-learn* from Packt Publishing is a great option. In addition to extensive hyperparameter fine-tuning, you will learn the historical context of XGBoost within the machine learning landscape, details of XGBoost case studies like the Higgs boson Kaggle

Corey Wade is the founder and director of Berkeley Coding Academy where he teaches Machine Learning to students from all over the world. In addition, Corey teaches math and programming at the Independent Study Program of Berkeley High School. He is the author of two books, Hands-on Gradient Boosting with XGBoost and scikit-learn and The Python Workshop.

Code in this article may be directly copied from Corey's Colab Notebook.

Photo by Alexander Sinn: https://unsplash.com/photos/KgLtFCgfC28