How to Develop Your First XGBoost Model in Python

XGBoost is an implementation of gradient boosted decision trees designed for speed and performance that is dominative competitive machine learning.

In this post you will discover how you can install and create your first XGBoost model in Python.

After reading this post you will know:

- How to install XGBoost on your system for use in Python.
- How to prepare data and train your first XGBoost model.
- How to make predictions using your XGBoost model.

Let’s get started.

How to Develop Your First XGBoost Model in Python with scikit-learn
Photo by Justin Henry, some rights reserved.

Tutorial Overview

This tutorial is broken down into the following 6 sections:

1. Install XGBoost for use with Python.
2. Problem definition and download dataset.
3. Load and prepare data.
4. Train XGBoost model.
5. Make predictions and evaluate model.
6. Tie it all together and run the example.

1. Install XGBoost for Use in Python

Assuming you have a working SciPy environment, XGBoost can be installed easily using pip.

For example:

```
sudo pip install xgboost
```

To update your installation of XGBoost you can type:

```
sudo pip install --upgrade xgboost
```

An alternate way to install XGBoost if you cannot use pip or you want to run the latest code from GitHub requires that you make a clone of the XGBoost project and perform a manual build and installation.

For example to build XGBoost without multithreading on Mac OS X (with GCC already installed via macports or homebrew), you can type:

```
git clone --recursive https://github.com/dmlc/xgboost
cd xgboost
cp make/minimum.mk ./config.mk
make -j4
cd python-package
sudo python setup.py install
```

You can learn more about how to install XGBoost for different platforms on the XGBoost Installation Guide. For up-to-date instructions for installing XGBoost for Python see the XGBoost Python Package.

For reference, you can review the XGBoost Python API reference.

2. Problem Description: Predict Onset of Diabetes

In this tutorial we are going to use the Pima Indians onset of diabetes dataset.

This dataset is comprised of 8 input variables that describe medical details of patients and one output variable to indicate whether the patient will have an onset of diabetes within 5 years.

You can learn more about this dataset on the UCI Machine Learning Repository website.

This is a good dataset for a first XGBoost model because all of the input variables are numeric and the problem is a simple binary classification problem. It is not necessarily a good problem for the XGBoost algorithm because it is a relatively small dataset and an easy problem to model.

Download this dataset and place it into your current working directory with the file name "**pima-indians-diabetes.csv**" (update: download from here).

3. Load and Prepare Data

In this section we will load the data from file and prepare it for use for training and evaluating an XGBoost model.

We will start off by importing the classes and functions we intend to use in this tutorial.

```
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

Next, we can load the CSV file as a NumPy array using the NumPy function **loadtext()**.

```
# load data
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=",")
```

We must separate the columns (attributes or features) of the dataset into input patterns (X) and output patterns (Y). We can do this easily by specifying the column indices in the NumPy array format.

```
# split data into X and y
X = dataset[:,0:8]
Y = dataset[:,8]
```

Finally, we must split the X and Y data into a training and test dataset. The training set will be used to prepare the XGBoost model and the test set will be used to make new predictions, from which we can evaluate the performance of the model.

For this we will use the **train_test_split()** function from the scikit-learn library. We also specify a seed for the random number generator so that we always get the same split of data each time this example is executed.

```
# split data into train and test sets
seed = 7
test_size = 0.33
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size,
random_state=seed)
```

We are now ready to train our model.

4. Train the XGBoost Model

XGBoost provides a wrapper class to allow models to be treated like classifiers or regressors in the scikit-learn framework.

This means we can use the full scikit-learn library with XGBoost models.

The XGBoost model for classification is called **XGBClassifier**. We can create and and fit it to our training dataset. Models are fit using the scikit-learn API and the **model.fit()** function.

Parameters for training the model can be passed to the model in the constructor. Here, we use the sensible defaults.

```
# fit model no training data
model = XGBClassifier()
model.fit(X_train, y_train)
```

You can see the parameters used in a trained model by printing the model, for example:

```
print(model)
```

You can learn more about the defaults for the **XGBClassifier** and **XGBRegressor** classes in the XGBoost Python scikit-learn API.

You can learn more about the meaning of each parameter and how to configure them on the XGBoost parameters page.

We are now ready to use the trained model to make predictions.

5. Make Predictions with XGBoost Model

We can make predictions using the fit model on the test dataset.

To make predictions we use the scikit-learn function **model.predict()**.

By default, the predictions made by XGBoost are probabilities. Because this is a binary classification problem, each prediction is the probability of the input pattern belonging to the first class. We can easily convert them to binary class values by rounding them to 0 or 1.

```
# make predictions for test data
y_pred = model.predict(X_test)
predictions = [round(value) for value in y_pred]
```

Now that we have used the fit model to make predictions on new data, we can evaluate the performance of the predictions by comparing them to the expected values. For this we will use the built in **accuracy_score()** function in scikit-learn.

```
# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

6. Tie it All Together

We can tie all of these pieces together, below is the full code listing.

```
# First XGBoost model for Pima Indians dataset
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# load data
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=",")
# split data into X and y
X = dataset[:,0:8]
Y = dataset[:,8]
# split data into train and test sets
seed = 7
test_size = 0.33
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size,
random_state=seed)
# fit model no training data
model = XGBClassifier()
model.fit(X_train, y_train)
# make predictions for test data
y_pred = model.predict(X_test)
predictions = [round(value) for value in y_pred]
# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

Running this example produces the following output.

```
Accuracy: 77.95%
```

This is a good accuracy score on this problem, which we would expect, given the capabilities of the model and the modest complexity of the problem.

Summary

In this post you discovered how to develop your first XGBoost model in Python.

Specifically, you learned:

- How to install XGBoost on your system ready for use with Python.
- How to prepare data and train your first XGBoost model on a standard machine learning dataset.
- How to make predictions and evaluate the performance of a trained XGBoost model using scikit-learn.

Do you have any questions about XGBoost or about this post? Ask your questions in the comments and I will do my best to answer.