XGBoost With Python

7-Day Mini-Course

Jason Brownlee

MACHINE
LEARNING
MASTERY

Jason Brownlee

XGBoost With Python

7 Day Mini-Course

XGBoost With Python

Edition: v1.0

Find the latest version of this guide online at: http://MachineLearningMastery.com

Contents

Before We Get Started... 1

Lesson 01: Introduction to Gradient Boosting 4

Lesson 02: Introduction to XGBoost 5

Lesson 03: Develop Your First XGBoost Model 6

Lesson 04: Monitor Performance and Early Stopping 8

Lesson 05: Feature Importance with XGBoost 10

Lesson 06: How to Configure Gradient Boosting 11

Lesson 07: XGBoost Hyperparameter Tuning 12

Final Word Before You Go... 14

Before We Get Started...

XGBoost is an implementation of gradient boosting that is being used to win machine learning competitions. It is powerful but it can be hard to get started. In this guide you will discover a 7-part crash course on XGBoost with Python. This mini course is designed for Python machine learning practitioners that are already comfortable with scikit-learn and the SciPy ecosystem. Let's get started.

Who Is This Mini-Course For?

Before we get started, let's make sure you are in the right place. The list below provides some general guidelines as to who this course was designed for. Don't panic if you don't match these points exactly, you might just need to brush up in one area or another to keep up.

- **Developers that know how to write a little code**. This means that it is not a big deal for you to get things done with Python and know how to setup the SciPy ecosystem on your workstation (a prerequisite). It does not mean your a wizard coder, but it does mean you're not afraid to install packages and write scripts.
- **Developers that know a little machine learning**. This means you know about the basics of machine learning like cross-validation, some algorithms and the bias-variance trade-off. It does not mean that you are a machine learning PhD, just that you know the landmarks or know where to look them up.

This mini-course is not a textbook on XGBoost. It will take you from a developer that knows a little machine learning in Python to a developer who can get results and bring the power of XGBoost to your own projects.

Mini-Course Overview (what to expect)

This mini-course is divided into 7 parts. Each lesson was designed to take the average developer about 30 minutes. You might finish some much sooner and others you may choose to go deeper and spend more time. You can complete each part as quickly or as slowly as you like. A comfortable schedule may be to complete one lesson per day over a one week period. Highly recommended. The topics you will cover over the next 7 lessons are as follows:

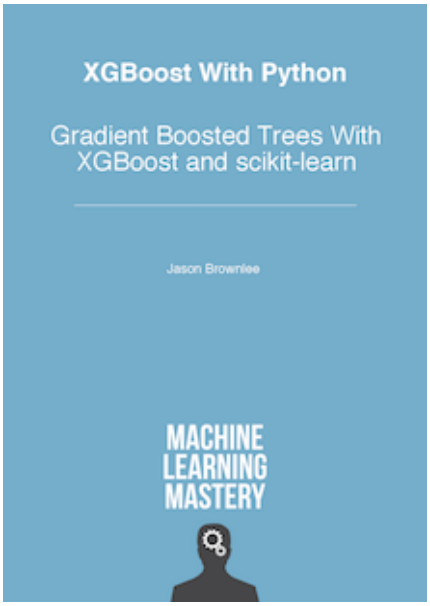
- **Lesson 01**: Introduction to Gradient Boosting.
- **Lesson 02**: Introduction to XGBoost.

- **Lesson 03**: Develop Your First XGBoost Model.
- **Lesson 04**: Monitor Performance and Early Stopping.
- **Lesson 05**: Feature Importance with XGBoost.
- **Lesson 06**: How to Configure Gradient Boosting.
- **Lesson 07**: XGBoost Hyperparameter Tuning.

This is going to be a lot of fun. You're going to have to do some work though, a little reading, a little research and a little programming. You want to learn XGBoost right? Here's a tip: All of the answers these lessons can be found on this blog http://MachineLearningMastery.com. Use the search feature.

Hang in there, don't give up!

If you would like me to step you through each lesson in great detail (and much more), take a look at my book: **XGBoost With Python:**



Learn more here:
https://machinelearningmastery.com/xgboost-with-python

Lesson 01: Introduction to Gradient Boosting

Gradient boosting is one of the most powerful techniques for building predictive models. The idea of boosting came out of the idea of whether a weak learner can be modified to become better. The first realization of boosting that saw great success in application was Adaptive Boosting or AdaBoost for short. The weak learners in AdaBoost are decision trees with a single split, called decision stumps for their shortness.

AdaBoost and related algorithms were recast in a statistical framework and became known as Gradient Boosting Machines. The statistical framework cast boosting as a numerical optimization problem where the objective is to minimize the loss of the model by adding weak learners using a gradient descent like procedure, hence the name. The Gradient Boosting algorithm involves three elements:

- **A loss function to be optimized**, such as cross entropy for classification or mean squared error for regression problems.
- **A weak learner to make predictions**, such as a greedily constructed decision tree.
- **An additive model**, used to add weak learners to minimize the loss function.

New weak learners are added to the model in an effort to correct the residual errors of all previous trees. The result is a powerful predictive modeling algorithm, perhaps more powerful than random forest. In the next lesson we will take a closer look at the XGBoost implementation of gradient boosting.

Lesson 02: Introduction to XGBoost

XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. XGBoost stands for eXtreme Gradient Boosting. It was developed by Tianqi Chen and is laser focused on computational speed and model performance, as such there are few frills. In addition to supporting all key variations of the technique, the real interest is the speed provided by the careful engineering of the implementation, including:

- **Parallelization** of tree construction using all of your CPU cores during training.
- **Distributed Computing** for training very large models using a cluster of machines.
- **Out-of-Core Computing** for very large datasets that don't fit into memory.
- **Cache Optimization** of data structures and algorithms to make best use of hardware.

Traditionally, gradient boosting implementations are slow because of the sequential nature in which each tree must be constructed and added to the model. The on performance in the development of XGBoost has resulted in one of the best predictive modeling algorithms that can now harness the full capability of your hardware platform, or very large computers you might rent in the cloud. As such, XGBoost has been a cornerstone in competitive machine learning, being the technique used to win and recommended by winners. For example, here is what some recent Kaggle competition winners have said:

> As the winner of an increasing amount of Kaggle competitions, XGBoost showed us again to be a great all-round algorithm worth having in your toolbox.
>
> – Dato Winners' Interview[1]

> When in doubt, use xgboost.
>
> – Avito Winner's Interview[2]

In the next lesson, we will develop our first XGBoost model in Python.

[1] http://goo.gl/AHkmWx
[2] http://goo.gl/sGyGtu

Lesson 03: Develop Your First XGBoost Model

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

```
sudo pip install xgboost
```

Listing 1: Install XGBoost using pip.

You can learn more about installing and building XGBoost on your platform in the XGBoost Installation Instructions[3]. XGBoost models can be used directly in the scikit-learn framework using the wrapper classes, `XGBClassifier` for classification and `XGBRegressor` for regression problems. This is the recommended way to use XGBoost in Python. Download the Pima Indians onset of diabetes dataset from the UCI Machine Learning repository[4]. It is a good test dataset for binary classification as all input variables are numeric, meaning the problem can be modeled directly with no data preparation. We can train an XGBoost model for classification by constructing it and calling the `model.fit()` function:

```
model = XGBClassifier()
model.fit(X_train, y_train)
```

Listing 2: Train an XGBoost Model.

This model can then be used to make predictions by calling the `model.predict()` function on new data.

```
y_pred = model.predict(X_test)
```

Listing 3: Make Predictions with an XGBoost Model.

We can tie this all together as follows:

```
# 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]
```

[3] http://xgboost.readthedocs.io/en/latest/build.html
[4] https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes

```
# 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 on 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))
```

Listing 4: First XGBoost Model.

In the next lesson we will look at how we can use early stopping to limit overfitting.

Lesson 04: Monitor Performance and Early Stopping

The XGBoost model can evaluate and report on the performance on a test set for the model during training. It supports this capability by specifying both a test dataset and an evaluation metric on the call to `model.fit()` when training the model and specifying verbose output (`verbose=True`). For example, we can report on the binary classification error rate (`error`) on a standalone test set (`eval_set`) while training an XGBoost model as follows:

```
eval_set = [(X_test, y_test)]
model.fit(X_train, y_train, eval_metric="error", eval_set=eval_set, verbose=True)
```

Listing 5: Monitor Performance During Training.

We can use this evaluation to stop training once no further improvements have been made to the model. We can do this by setting the `early_stopping_rounds` parameter when calling `model.fit()` to the number of iterations that no improvement is seen on the validation dataset before training is stopped. The full example using the Pima Indians Onset of Diabetes dataset is provided below.

```
# exmaple of early stopping
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 on training data
model = XGBClassifier()
eval_set = [(X_test, y_test)]
model.fit(X_train, y_train, early_stopping_rounds=10, eval_metric="logloss",
    eval_set=eval_set, verbose=True)
# 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))
```

Listing 6: Example of Early Stopping.

In the next lesson we will look at how we calculate the importance of features using XGBoost.

Lesson 05: Feature Importance with XGBoost

A benefit of using ensembles of decision tree methods like gradient boosting is that they can automatically provide estimates of feature importance from a trained predictive model. A trained XGBoost model automatically calculates feature importance on your predictive modeling problem. These importance scores are available in the `feature_importances_` member variable of the trained model. For example, they can be printed directly as follows:

```
print(model.feature_importances_)
```

Listing 7: Print Feature Importance.

The XGBoost library provides a built-in function to plot features ordered by their importance. The function is called `plot_importance()` and can be used as follows:

```
plot_importance(model)
pyplot.show()
```

Listing 8: Plot Feature Importance.

These importance scores can help you decide what input variables to keep or discard. They can also be used as the basis for automatic feature selection techniques. The full example of plotting feature importance scores using the Pima Indians Onset of Diabetes dataset is provided below.

```
# plot feature importance using built-in function
from numpy import loadtxt
from xgboost import XGBClassifier
from xgboost import plot_importance
from matplotlib import pyplot
# load data
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=",")
# split data into X and y
X = dataset[:,0:8]
y = dataset[:,8]
# fit model on training data
model = XGBClassifier()
model.fit(X, y)
# plot feature importance
plot_importance(model)
pyplot.show()
```

Listing 9: Example of Plotting Feature Importance.

In the next lesson we will look at heuristics for best configuring the gradient boosting algorithm.

Lesson 06: How to Configure Gradient Boosting

Gradient boosting is one of the most powerful techniques for applied machine learning and as such is quickly becoming one of the most popular. But how do you configure gradient boosting on your problem? A number of configuration heuristics were published in the original gradient boosting papers. They can be summarized as:

- Learning rate or shrinkage (`learning_rate` in XGBoost) should be set to 0.1 or lower, and smaller values will require the addition of more trees.
- The depth of trees (`tree_depth` in XGBoost) should be configured in the range of 2-to-8, where not much benefit is seen with deeper trees.
- Row sampling (`subsample` in XGBoost) should be configured in the range of 30% to 80% of the training dataset, and compared to a value of 100% for no sampling.

These are a good starting points when configuring your model. A good general configuration strategy is as follows:

1. Run the default configuration and review plots of the learning curves on the training and validation datasets.
2. If the system is overlearning, decrease the learning rate and/or increase the number of trees.
3. If the system is underlearning, speed the learning up to be more aggressive by increasing the learning rate and/or decreasing the number of trees.

Owen Zhang, the former #1 ranked competitor on Kaggle and now CTO at Data Robot proposes an interesting strategy to configure XGBoost[5]. He suggests to set the number of trees to a target value such as 100 or 1000, then tune the learning rate to find the best model. This is an efficient strategy for quickly finding a good model. In the next and final lesson, we will look at an example of tuning the XGBoost hyperparameters.

[5] http://goo.gl/OqIRIc

Lesson 07: XGBoost Hyperparameter Tuning

The scikit-learn framework provides the capability to search combinations of parameters. This capability is provided in the GridSearchCV class and can be used to discover the best way to configure the model for top performance on your problem. For example, we can define a grid of the number of trees (`n_estimators`) and tree sizes (`max_depth`) to evaluate by defining a grid as:

```
n_estimators = [50, 100, 150, 200]
max_depth = [2, 4, 6, 8]
param_grid = dict(max_depth=max_depth, n_estimators=n_estimators)
```

Listing 10: Define a Grid Search.

And then evaluate each combination of parameters using 10-fold cross-validation as:

```
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=7)
grid_search = GridSearchCV(model, param_grid, scoring="neg_log_loss", n_jobs=-1, cv=kfold,
    verbose=1)
result = grid_search.fit(X, label_encoded_y)
```

Listing 11: Perform a Grid Search.

We can then review the results to determine the best combination and the general trends in varying the combinations of parameters. This is the best practice when applying XGBoost to your own problems. The parameters to consider tuning are:

- The number and size of trees (`n_estimators` and `max_depth`).
- The learning rate and number of trees (`learning_rate` and `n_estimators`).
- The row and column subsampling rates (`subsample`, `colsample_bytree` and `colsample_bylevel`).

Below is a full example of tuning just the `learning_rate` on the Pima Indians Onset of Diabetes dataset.

```
# Tune learning_rate
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
# load data
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=",")
# split data into X and y
```

```
X = dataset[:,0:8]
Y = dataset[:,8]
# grid search
model = XGBClassifier()
learning_rate = [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3]
param_grid = dict(learning_rate=learning_rate)
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=7)
grid_search = GridSearchCV(model, param_grid, scoring="neg_log_loss", n_jobs=-1, cv=kfold)
grid_result = grid_search.fit(X, Y)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
  print("%f (%f) with: %r" % (mean, stdev, param))
```

Listing 12: Example of a Grid Search.

Final Word Before You Go...

You made it. Well done! Take a moment and look back at how far you have come:

- You learned about the gradient boosting algorithm and the XGBoost library.
- You developed your first XGBoost model.
- You learned how to use advanced features like early stopping and feature importance.
- You learned how to configure gradient boosted models and how to design controlled experiments to tune XGBoost hyperparameters.

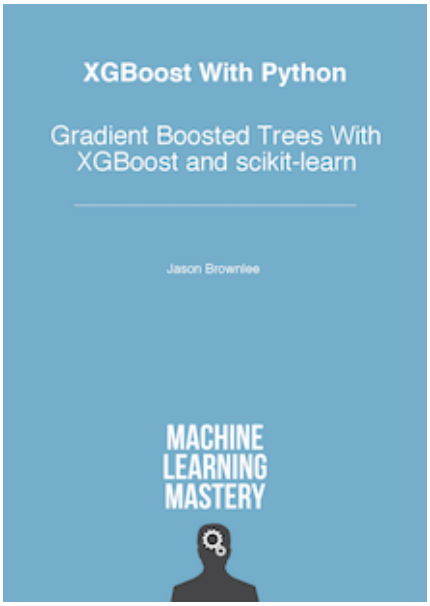
Don't make light of this, you have come a long way in a short amount of time. This is just the beginning of your journey with XGBoost in Python. Keep practicing and developing your skills.

How Did You Go With The Mini-Course?

Did you enjoy this mini-course?
Do you have any questions or sticking points?

Let me know, send me an email at: **jason@MachineLearningMastery.com**

If you would like me to step you through each lesson in great detail (and much more), take a look at my book: **XGBoost With Python:**



Learn more here:
https://machinelearningmastery.com/xgboost-with-python