

[Start Here](#)[Blog](#)[Books](#)[About](#)[Contact](#)

Search...



Need help with XGBoost in Python? [Take the FREE Mini-Course](#)

Data Preparation for Gradient Boosting with XGBoost in Python

by **Jason Brownlee** on August 22, 2016 in **XGBoost**



XGBoost is a popular implementation of Gradient Boosting because of its speed and performance.

Internally, XGBoost models represent all problems as a regression predictive modeling problem that only takes numerical values as input. If your data is in a different form, it must be prepared into the expected format.

In this post, you will discover how to prepare your data for using with gradient boosting with the XGBoost library in Python.

After reading this post you will know:

- How to encode string output variables for classification.
- How to prepare categorical input variables using one hot encoding.
- How to automatically handle missing data

Get Your Start in Machine Learning

Let's get started.

- **Update Sept/2016:** I updated a few small typos in the impute example.
- **Update Jan/2017:** Updated to reflect changes in scikit-learn API version 0.18.1.
- **Update Jan/2017:** Updated breast cancer example to converted input data to strings.



Data Preparation for Gradient Boosting with XGBoost in Python
Photo by [Ed Dunens](#), some rights reserved.

Need help with XGBoost in Python?

Take my free 7-day email course and discover configuration, tuning and more (with sample code).

Click to sign-up now and also get a free PDF Ebook version of the course.

Start Your FREE Mini-Course Now!

Get Your Start in Machine Learning

Label Encode String Class Values

The iris flowers classification problem is an example of a problem that has a string class value.

This is a prediction problem where given measurements of iris flowers in centimeters, the task is to predict to which species a given flower belongs.

Below is a sample of the raw dataset. You can learn more about this dataset and download the raw data in CSV format from the [UCI Machine Learning Repository](#).

```
1 5.1,3.5,1.4,0.2,Iris-setosa
2 4.9,3.0,1.4,0.2,Iris-setosa
3 4.7,3.2,1.3,0.2,Iris-setosa
4 4.6,3.1,1.5,0.2,Iris-setosa
5 5.0,3.6,1.4,0.2,Iris-setosa
```

XGBoost cannot model this problem as-is as it requires that the output variables be numeric.

We can easily convert the string values to integer values using the [LabelEncoder](#). The three class values (Iris-setosa, Iris-versicolor, Iris-virginica) are mapped to the integer values (0, 1, 2).

```
1 # encode string class values as integers
2 label_encoder = LabelEncoder()
3 label_encoder = label_encoder.fit(Y)
4 label_encoded_y = label_encoder.transform(Y)
```

We save the label encoder as a separate object so that we can transform both the training and later the test and validation datasets using the same encoding scheme.

Below is a complete example demonstrating how to load the iris dataset. Notice that Pandas is used to load the data in order to handle the string class values.

```
1 # multiclass classification
2 import pandas
3 import xgboost
4 from sklearn import model_selection
5 from sklearn.metrics import accuracy_score
6 from sklearn.preprocessing import LabelEncoder
7 # load data
8 data = pandas.read_csv('iris.csv', header=None)
9 dataset = data.values
10 # split data into X and y
11 X = dataset[:,0:4]
12 Y = dataset[:,4]
13 # encode string class values as integers
14 label_encoder = LabelEncoder()
15 label_encoder = label_encoder.fit(Y)
16 label_encoded_y = label_encoder.transform(Y)
17 seed = 7
18 test_size = 0.33
```

Get Your Start in Machine Learning

```
19 X_train, X_test, y_train, y_test = cross_validation.train_test_split(X, label_encoded)
20 # fit model no training data
21 model = xgboost.XGBClassifier()
22 model.fit(X_train, y_train)
23 print(model)
24 # make predictions for test data
25 y_pred = model.predict(X_test)
26 predictions = [round(value) for value in y_pred]
27 # evaluate predictions
28 accuracy = accuracy_score(y_test, predictions)
29 print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

Running the example produces the following output:

```
1 XGBClassifier(base_score=0.5, colsample_bylevel=1, colsample_bytree=1,
2     gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=3,
3     min_child_weight=1, missing=None, n_estimators=100, nthread=-1,
4     objective='multi:softprob', reg_alpha=0, reg_lambda=1,
5     scale_pos_weight=1, seed=0, silent=True, subsample=1)
6 Accuracy: 92.00%
```

Notice how the XGBoost model is configured to automatically model the multiclass classification problem using the **multi:softprob** objective, a variation on the softmax loss function to model class probabilities. This suggests that internally, that the output class is converted into a one hot type encoding automatically.

One Hot Encode Categorical Data

Some datasets only contain categorical data, for example the breast cancer dataset.

This dataset describes the technical details of breast cancer biopsies and the prediction task is to predict whether or not the patient has a recurrence of cancer, or not.

Below is a sample of the raw dataset. You can learn more about this dataset at the [UCI Machine Learning Repository](#) and download it in CSV format from [mldata.org](#).

```
1 '40-49','premeno','15-19','0-2','yes','3','right','left_up','no','recurrence-events'
2 '50-59','ge40','15-19','0-2','no','1','right','central','no','no-recurrence-events'
3 '50-59','ge40','35-39','0-2','no','2','left','left_low','no','recurrence-events'
4 '40-49','premeno','35-39','0-2','yes','3','right','left_low','yes','no-recurrence-event'
5 '40-49','premeno','30-34','3-5','yes','2','left','right_up','no','recurrence-events'
```

We can see that all 9 input variables are categorical and described in string format. The problem is a binary classification prediction problem and the output class values are also described in string format.

We can reuse the same approach from the previous section and convert the string class values to integer values to model the prediction using

Get Your Start in Machine Learning

```
1 # encode string class values as integers
```

We can use this same approach on each input feature in *X*, but this is only a starting point.

```
1 # encode string input values as integers
2 features = []
3 for i in range(0, X.shape[1]):
4     label_encoder = LabelEncoder()
5     feature = label_encoder.fit_transform(X[:,i])
6     features.append(feature)
7 encoded_x = numpy.array(features)
8 encoded_x = encoded_x.reshape(X.shape[0], X.shape[1])
```

XGBoost may assume that encoded integer values for each input variable have an ordinal relationship. For example that 'left-up' encoded as 0 and 'left-low' encoded as 1 for the breast-quad variable have a meaningful relationship as integers. In this case, this assumption is untrue.

Instead, we must map these integer values onto new binary variables, one new variable for each categorical value.

For example, the breast-quad variable has the values:

```
1 left-up
2 left-low
3 right-up
4 right-low
5 central
```

We can model this as 5 binary variables as follows:

```
1 left-up, left-low, right-up, right-low, central
2 1,0,0,0,0
3 0,1,0,0,0
4 0,0,1,0,0
5 0,0,0,1,0
6 0,0,0,0,1
```

This is called [one hot encoding](#). We can one hot encode all of the categorical input variables using the [OneHotEncoder](#) class in scikit-learn.

We can one hot encode each feature after we have label encoded it. First we must transform the feature array into a 2-dimensional NumPy array where each integer value is a feature vector with a length 1.

```
1 feature = feature.reshape(X.shape[0], 1)
```

We can then create the OneHotEncoder and

Get Your Start in Machine Learning

```

1 onehot_encoder = OneHotEncoder(sparse=False)
2 feature = onehot_encoder.fit_transform(feature)

```

Finally, we can build up the input dataset by concatenating the one hot encoded features, one by one, adding them on as new columns (axis=2). We end up with an input vector comprised of 43 binary input variables.

```

1 # encode string input values as integers
2 encoded_x = None
3 for i in range(0, X.shape[1]):
4     label_encoder = LabelEncoder()
5     feature = label_encoder.fit_transform(X[:,i])
6     feature = feature.reshape(X.shape[0], 1)
7     onehot_encoder = OneHotEncoder(sparse=False)
8     feature = onehot_encoder.fit_transform(feature)
9     if encoded_x is None:
10         encoded_x = feature
11     else:
12         encoded_x = numpy.concatenate((encoded_x, feature), axis=1)
13 print("X shape: ", encoded_x.shape)

```

Ideally, we may experiment with not one hot encode some of input attributes as we could encode them with an explicit ordinal relationship, for example the first column age with values like '40-49' and '50-59'. This is left as an exercise, if you are interested in extending this example.

Below is the complete example with label and one hot encoded input variables and label encoded output variable.

```

1 # binary classification, breast cancer dataset, label and one hot encoded
2 import numpy
3 from pandas import read_csv
4 from xgboost import XGBClassifier
5 from sklearn.model_selection import train_test_split
6 from sklearn.metrics import accuracy_score
7 from sklearn.preprocessing import LabelEncoder
8 from sklearn.preprocessing import OneHotEncoder
9 # load data
10 data = read_csv('datasets-uci-breast-cancer.csv', header=None)
11 dataset = data.values
12 # split data into X and y
13 X = dataset[:,0:9]
14 X = X.astype(str)
15 Y = dataset[:,9]
16 # encode string input values as integers
17 encoded_x = None
18 for i in range(0, X.shape[1]):
19     label_encoder = LabelEncoder()
20     feature = label_encoder.fit_transform(X[:,i])
21     feature = feature.reshape(X.shape[0], 1)
22     onehot_encoder = OneHotEncoder(sparse=False)
23     feature = onehot_encoder.fit_transform(feature)
24     if encoded_x is None:
25         encoded_x = feature
26     else:

```

Get Your Start in Machine Learning

```

27         encoded_x = numpy.concatenate((encoded_x, feature), axis=1)
28 print("X shape: ", encoded_x.shape)
29 # encode string class values as integers
30 label_encoder = LabelEncoder()
31 label_encoder = label_encoder.fit(Y)
32 label_encoded_y = label_encoder.transform(Y)
33 # split data into train and test sets
34 seed = 7
35 test_size = 0.33
36 X_train, X_test, y_train, y_test = train_test_split(encoded_x, label_encoded_y, test_s
37 # fit model no training data
38 model = XGBClassifier()
39 model.fit(X_train, y_train)
40 print(model)
41 # make predictions for test data
42 y_pred = model.predict(X_test)
43 predictions = [round(value) for value in y_pred]
44 # evaluate predictions
45 accuracy = accuracy_score(y_test, predictions)
46 print("Accuracy: %.2f%%" % (accuracy * 100.0))

```

Running this example we get the following output:

```

1 ('X shape: ', (285, 43))
2 XGBClassifier(base_score=0.5, colsample_bylevel=1, colsample_bytree=1,
3     gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=3,
4     min_child_weight=1, missing=None, n_estimators=100, nthread=-1,
5     objective='binary:logistic', reg_alpha=0, reg_lambda=1,
6     scale_pos_weight=1, seed=0, silent=True, subsample=1)
7 Accuracy: 71.58%

```

Again we can see that the XGBoost framework chose the **'binary:logistic'** objective automatically, the right objective for this binary classification problem.

Support for Missing Data

XGBoost can automatically learn how to best handle missing data.

In fact, XGBoost was designed to work with sparse data, like the one hot encoded data from the previous section, and missing data is handled the same way that sparse or zero values are handled, by minimizing the loss function.

For more information on the technical details for how missing values are handled in XGBoost, see Section 3.4 “*Sparsity-aware Split Finding*” in the paper [XGBoost: A Scalable Tree Boosting System](#).

The Horse Colic dataset is a good example to demonstrate this capability as it contains a large percentage of missing data, approximately 30%.

You can learn more about the Horse Colic d

Get Your Start in Machine Learning

Machine Learning repository.

The values are separated by whitespace and we can easily load it using the Pandas function `read_csv`.

```
1 dataframe = read_csv("horse-colic.csv", delim_whitespace=True, header=None)
```

Once loaded, we can see that the missing data is marked with a question mark character ('?'). We can change these missing values to the sparse value expected by XGBoost which is the value zero (0).

```
1 # set missing values to 0
2 X[X == '?'] = 0
```

Because the missing data was marked as strings, those columns with missing data were all loaded as string data types. We can now convert the entire set of input data to numerical values.

```
1 # convert to numeric
2 X = X.astype('float32')
```

Finally, this is a binary classification problem although the class values are marked with the integers 1 and 2. We model binary classification problems in XGBoost as logistic 0 and 1 values. We can easily convert the Y dataset to 0 and 1 integers using the LabelEncoder, as we did in the iris flowers example.

```
1 # encode Y class values as integers
2 label_encoder = LabelEncoder()
3 label_encoder = label_encoder.fit(Y)
4 label_encoded_y = label_encoder.transform(Y)
```

The full code listing is provided below for completeness.

```
1 # binary classification, missing data
2 from pandas import read_csv
3 from xgboost import XGBClassifier
4 from sklearn.model_selection import train_test_split
5 from sklearn.metrics import accuracy_score
6 from sklearn.preprocessing import LabelEncoder
7 # load data
8 dataframe = read_csv("horse-colic.csv", delim_whitespace=True, header=None)
9 dataset = dataframe.values
10 # split data into X and y
11 X = dataset[:,0:27]
12 Y = dataset[:,27]
13 # set missing values to 0
14 X[X == '?'] = 0
15 # convert to numeric
16 X = X.astype('float32')
17 # encode Y class values as integers
18 label_encoder = LabelEncoder()
19 label_encoder = label_encoder.fit(Y)
```

Get Your Start in Machine Learning


```

20 label_encoded_y = label_encoder.transform(Y)
21 # split data into train and test sets
22 seed = 7
23 test_size = 0.33
24 X_train, X_test, y_train, y_test = train_test_split(X, label_encoded_y, test_size=test
25 # fit model no training data
26 model = XGBClassifier()
27 model.fit(X_train, y_train)
28 print(model)
29 # make predictions for test data
30 y_pred = model.predict(X_test)
31 predictions = [round(value) for value in y_pred]
32 # evaluate predictions
33 accuracy = accuracy_score(y_test, predictions)
34 print("Accuracy: %.2f%%" % (accuracy * 100.0))

```

Running this example produces the following output.

```

1 XGBClassifier(base_score=0.5, colsample_bylevel=1, colsample_bytree=1,
2     gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=3,
3     min_child_weight=1, missing=None, n_estimators=100, nthread=-1,
4     objective='binary:logistic', reg_alpha=0, reg_lambda=1,
5     scale_pos_weight=1, seed=0, silent=True, subsample=1)
6 Accuracy: 83.84%

```

We can tease out the effect of XGBoost's automatic handling of missing values, by marking the missing values with a non-zero value, such as 1.

```

1 X[X == '?'] = 1

```

Re-running the example demonstrates a drop in accuracy for the model.

```

1 Accuracy: 79.80%

```

We can also impute the missing data with a specific value.

It is common to use a mean or a median for the column. We can easily impute the missing data using the scikit-learn [Imputer](#) class.

```

1 # impute missing values as the mean
2 imputer = Imputer()
3 imputed_x = imputer.fit_transform(X)

```

Below is the full example with missing data imputed with the mean value from each column.

```

1 # binary classification, missing data, impute with mean
2 import numpy
3 from pandas import read_csv
4 from xgboost import XGBClassifier
5 from sklearn.model_selection import train_test_split
6 from sklearn.metrics import accuracy_score
7 from sklearn.preprocessing import LabelEncoder
8 from sklearn.preprocessing import Imputer
9 # load data

```

Get Your Start in Machine Learning

```
10 dataframe = read_csv("horse-colic.csv", delim_whitespace=True, header=None)
11 dataset = dataframe.values
12 # split data into X and y
13 X = dataset[:,0:27]
14 Y = dataset[:,27]
15 # set missing values to 0
16 X[X == '?'] = numpy.nan
17 # convert to numeric
18 X = X.astype('float32')
19 # impute missing values as the mean
20 imputer = Imputer()
21 imputed_x = imputer.fit_transform(X)
22 # encode Y class values as integers
23 label_encoder = LabelEncoder()
24 label_encoder = label_encoder.fit(Y)
25 label_encoded_y = label_encoder.transform(Y)
26 # split data into train and test sets
27 seed = 7
28 test_size = 0.33
29 X_train, X_test, y_train, y_test = train_test_split(imputed_x, label_encoded_y, test_s
30 # fit model no training data
31 model = XGBClassifier()
32 model.fit(X_train, y_train)
33 print(model)
34 # make predictions for test data
35 y_pred = model.predict(X_test)
36 predictions = [round(value) for value in y_pred]
37 # evaluate predictions
38 accuracy = accuracy_score(y_test, predictions)
39 print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

Running this example we see results equivalent to the fixing the value to one (1). This suggests that at least in this case we are better off marking the missing values with a distinct value of zero (0) rather than a valid value (1) or an imputed value.

```
1 Accuracy: 79.80%
```

It is a good lesson to try both approaches (automatic handling and imputing) on your data when you have missing values.

Summary

In this post you discovered how you can prepare your machine learning data for gradient boosting with XGBoost in Python.

Specifically, you learned:

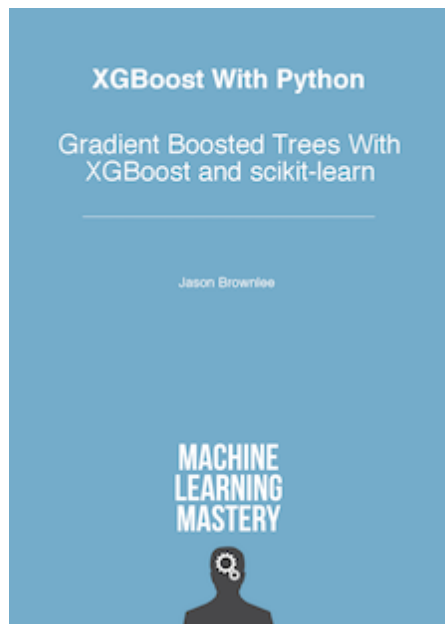
- How to prepare string class values for binary classification using label encoding.
- How to prepare categorical input variables using a one hot encoding to model them as binary variables.
- How XGBoost automatically handles mi

Get Your Start in Machine Learning

missing values.

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

Want To Learn The Algorithm Winning Competitions?



Develop Your Own XGBoost Models in Minutes

...with just a few lines of Python

Discover how in my new Ebook:

[XGBoost With Python](#)

It covers **self-study tutorials** like:

Algorithm Fundamentals, Scaling, Hyperparameters, and much more...

Bring The Power of XGBoost To Your Own Projects

Skip the Academics. Just Results.

[Click to learn more.](#)



About Jason Brownlee

Dr. Jason Brownlee is a husband, proud father, academic researcher, author, professional developer and a machine learning practitioner. He is dedicated to helping developers get started and get good at applied machine learning. [Learn more.](#)

[View all posts by Jason Brownlee](#) →

< [How to Develop Your First XGBoost Model in Python](#)

[Get Your Start in Machine Learning](#)

[How to Save Gradient Boosting Models with XGBoost in Python >](#)

20 Responses to *Data Preparation for Gradient Boosting with XGBoost in Python*



Ralph_adu August 28, 2016 at 1:24 am #

REPLY ↩

hi Jason, the train data for the last example should be `imputed_x`, but you use the original `X` which has missing data. I tried with data `imputed_x` and get the accuracy 79.8%



Jason Brownlee September 12, 2016 at 2:35 pm #

REPLY ↩

Thanks Ralph. Fixed.



Qichang September 19, 2016 at 9:29 am #

REPLY ↩

Hi Jason,

Thanks for the tutorial with such useful information! I have one question regarding the label encoding and the one hot encoding you applied on the breast cancer dataset.

You perform label encoding and one hot encoding for the whole dataset and then split into train and test set. This way it can be ensured that all the data are transformed with the same encoding configuration.

However, if we have new unseen data with the raw dataset type, how can we ensure that label encoding and one hot encoding is still transforming the unseen data in the same way? Do we need to save the encoders for the sake of processing unseen data?

Thanks in advance!



Jason Brownlee September 20, 2016 at 8:28 am #

REPLY ↩

Great question Qichang.

Get Your Start in Machine Learning

I would prepare the encodings on the training data, store the mappings (or pickle the objects), then reuse the encodings on the test data.

It means we must be confident that the training data is representative of the data we may need to predict in the future.



Qichang September 20, 2016 at 7:47 pm #

REPLY ↩

Thanks Jason for the prompt reply.

Besides this kinds of data transformation, do we need to consider scaling or normalisation of the input variables before passing to XGBoost? We know that it generally yields better result for SVM especially with kernel function.



Jason Brownlee September 21, 2016 at 8:28 am #

REPLY ↩

Generally no scaling. You may see some benefit by spreading out a univariate distribution to highlight specific features (e.g. with a square, log, square root, etc.)



JChen October 12, 2016 at 5:26 am #

REPLY ↩

Hi Jason, your post is very helpful. Thanks a lot!

I had a question around how to treat “default” values of continuous predictors for XGBoost. For example, let’s say attributer X may take continuous values as such (say in range of 1 -100). But certain records may have some default value (say 9999) which denotes certain segment of customers for whom that predictor X cannot be calculated or is unavailable. Can we directly use predictor X as input variable for an XGBoost model? Or, should we do some data treatment for X? If so, what would that be?

TIA



Jason Brownlee October 12, 2016 at 9:14 am #

REPLY ↩

Great question JChen.

I would try modeling the data as-is to sta

Get Your Start in Machine Learning

You could then try some feature engineering (maybe add a new flag variable for such cases) and see if you can further list performance.



JChen October 14, 2016 at 2:37 am #

REPLY ↩

Thanks for your reply! This is helpful



Jason Brownlee October 14, 2016 at 9:02 am #

REPLY ↩

I'm glad to hear it JChen.



Sargam Modak February 9, 2017 at 11:54 pm #

REPLY ↩

For your missing data part you replaced '?' with 0. But you have not mentioned while defining XGBClassifier model that in your dataset treat 0 as missing value. And by default 'missing' parameter value is none which is equivalent to treating NaN as missing value. So i don't think your model is handling missing values.



Jason Brownlee February 10, 2017 at 9:54 am #

REPLY ↩

Thanks for the note Sargam.



Tursun April 19, 2017 at 11:54 pm #

REPLY ↩

Hi Jason,
I came to this page from another : IRIS ,
source: <http://machinelearningmastery.com/multi-class-classification-tutorial-keras-deep-learning-library/#comment-396630>

In that webPage, your code classifies IRIS with 96.6% accuracy, which is very good.
In comments section, you told this guy (Abhilash Menon) to use gradient boosting with XGBoost. Which is this tutorial. Here is also IRIS classification.

Well I do not understand:

Here, the IRIS classification with gradient bo

Get Your Start in Machine Learning

(only).

Why should we use gradient boosting with XGBoost then?

Accuracy is too low.



Jason Brownlee April 20, 2017 at 9:28 am #

REPLY ↩

The tutorial is a demonstration of how to use the algorithm.

Your job as the machine learning practitioner is to try many algorithms on your problem and see what works best.

See this post:

<http://machinelearningmastery.com/a-data-driven-approach-to-machine-learning/>



daniel May 10, 2017 at 1:43 am #

REPLY ↩

Hi, Jason. First of all i would like to thank you for the wonderful material. I have been trying the XGBoost algorithm, it seems its acting weird on my pc. The first iris species dataset i got score of 21.2 %. Now this one of the breast cancer my accuracy is 2.1%. I really dont know wats wrong, can you please, please help me



Jason Brownlee May 10, 2017 at 8:51 am #

REPLY ↩

I'm sorry to hear that. Perhaps the library is not installed correctly?

I would recommend posting a question to stackoverflow or even the xgboost users group.

Let me know how you go.



Giulio June 15, 2017 at 3:50 am #

REPLY ↩

Hello Jason, I know that for regression models we should drop the first dummy variable to avoid the "dummy trap". I don't see you doing that in this case. Is it because the dummy variable trap only applies to linear regression models and not gradient boosting algorithms?

Get Your Start in Machine Learning

Jason Brownlee June 15, 2017 at 8:50 am #

REPLY ↩



I believe the dummy variable trap applies with linear models and when the variables are multicollinear.



Eran August 25, 2017 at 12:22 am #

REPLY ↩

Hello Jason.
What happens if I have a string column with NA's?
Should I first put 0's instead of the NA's and then use the OneHotEncoder ?



Jason Brownlee August 25, 2017 at 6:44 am #

REPLY ↩

XGBoost can handle the NAs. No need to do anything.
If you do want to impute or similar, see this post:
<https://machinelearningmastery.com/handle-missing-data-python/>

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

Get Your Start in Machine Learning

SUBMIT COMMENT

Welcome to Machine Learning Mastery



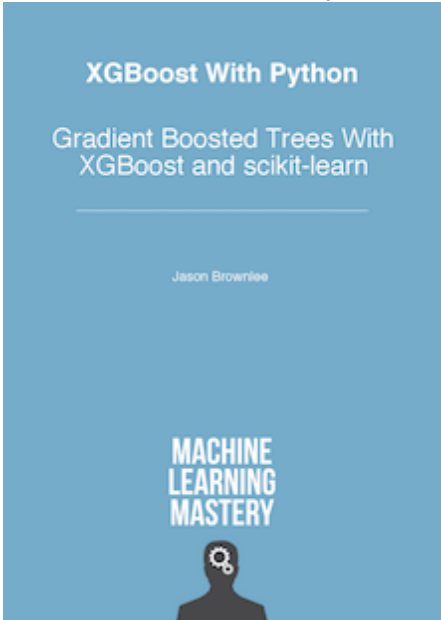
Hi, I'm Dr. Jason Brownlee.
My goal is to make practitioners like YOU awesome at applied machine learning.

[Read More](#)

The Algorithm that is Winning Competitions

Discover the machine learning algorithm
that is Winning Competitions.

[Get Started With XGBoost in Python Today!](#)












POPULAR

Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras

JULY 21, 2016

Your First Machine Learning Project in Python

Get Your Start in Machine Learning

	JUNE 10, 2016
	Develop Your First Neural Network in Python With Keras Step-By-Step MAY 24, 2016
	How to Setup a Python Environment for Machine Learning and Deep Learning with Anaconda MARCH 13, 2017
	Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras JULY 26, 2016
	Time Series Forecasting with the Long Short-Term Memory Network in Python APRIL 7, 2017
	Multi-Class Classification Tutorial with the Keras Deep Learning Library JUNE 2, 2016
	Multivariate Time Series Forecasting with LSTMs in Keras AUGUST 14, 2017
	Regression Tutorial with the Keras Deep Learning Library in Python JUNE 9, 2016
	How to Implement the Backpropagation Algorithm From Scratch In Python NOVEMBER 7, 2016

© 2017 Machine Learning Mastery. All Rights Reserved.

[Privacy](#) | [Contact](#) | [About](#)

Get Your Start in Machine Learning