Machine Learning Mastery
Making Developers Awesome at Machine Learning

How to Develop Multi-Output Regression Models with Python

by **Jason Brownlee** on March 27, 2020 in **Python Machine Learning**

Multioutput regression are regression problems that involve predicting two or more numerical values given an input example.

An example might be to predict a coordinate given an input, e.g. predicting x and y values. Another example would be multi-step time series forecasting that involves predicting multiple future time series of a given variable.

Many machine learning algorithms are designed for predicting a single numeric value, referred to simply as regression. Some algorithms do support multioutput regression inherently, such as linear regression and decision trees. There are also special workaround models that can be used to wrap and use those algorithms that do not natively support predicting multiple outputs.

In this tutorial, you will discover how to develop machine learning models for multioutput regression.

After completing this tutorial, you will know:

- The problem of multioutput regression in machine learning.
- How to develop machine learning models that inherently support multiple-output regression.
- How to develop wrapper models that allow algorithms that do not inherently support multiple outputs to be used for multiple-output regression.

Let's get started.

Tutorial Overview

This tutorial is divided into three parts; they are:

1. Problem of Multioutput Regression
    1. Check Scikit-Learn Version
    2. Multioutput Regression Test Problem
2. Inherently Multioutput Regression Algorithms
    1. Linear Regression for Multioutput Regression
    2. k-Nearest Neighbors for Multioutput Regression
    3. Random Forest for Multioutput Regression

Problem of Multioutput Regression

Regression refers to a predictive modeling problem that involves predicting a numerical value.

For example, predicting a size, weight, amount, number of sales, and number of clicks are regression problems. Typically, a single numeric value is predicted given input variables.

Some regression problems require the prediction of two or more numeric values. For example, predicting an x and y coordinate.

These problems are referred to as multiple-output regression, or multioutput regression.

- **Regression**: Predict a single numeric output given an input.
- **Multioutput Regression**: Predict two or more numeric outputs given an input.

In multioutput regression, typically the outputs are dependent upon the input and upon each other. This means that often the outputs are not independent of each other and may require a model that predicts both outputs together or each output contingent upon the other outputs.

Multi-step time series forecasting may be considered a type of multiple-output regression where a sequence of future values are predicted and each predicted value is dependent upon the prior values in the sequence.

There are a number of strategies for handling multioutput regression and we will explore some of them in this tutorial.

Check Scikit-Learn Version

First, confirm that you have a modern version of the scikit-learn library installed.

This is important because some of the models we will explore in this tutorial require a modern version of the library.

You can check the version of the library with the following code example:

```
# check scikit-learn version
import sklearn
print(sklearn.__version__)
```

Running the example will print the version of the library.

At the time of writing, this is about version 0.22. You need to be using this version of scikit-learn or higher.

```
0.22.1
```

Multioutput Regression Test Problem

We can define a test problem that we can use to demonstrate the different modeling strategies.

We will use the make_regression() function to create a test dataset for multiple-output regression. We will generate 1,000 examples with 10 input features, five of which will be redundant and five that will be informative. The problem will require the prediction of two numeric values.

- **Problem Input**: 10 numeric variables.
- **Problem Output**: 2 numeric variables.

The example below generates the dataset and summarizes the shape.

```
# example of multioutput regression test problem
from sklearn.datasets import make_regression
# create datasets
X, y = make_regression(n_samples=1000, n_features=10, n_informative=5, n_targets=2, random_state
# summarize dataset
print(X.shape, y.shape)
```

Running the example creates the dataset and summarizes the shape of the input and output elements of the dataset for modeling, confirming the chosen configuration.

```
(1000, 10) (1000, 2)
```

Next, let's look at modeling this problem directly.

Inherently Multioutput Regression Algorithms

Some regression machine learning algorithms support multiple outputs directly.

This includes most of the popular machine learning algorithms implemented in the scikit-learn library, such as:

- LinearRegression (and related)
- KNeighborsRegressor
- DecisionTreeRegressor
- RandomForestRegressor (and related)

Let's look at a few examples to make this concrete.

Linear Regression for Multioutput Regression

The example below fits a linear regression model on the multioutput regression dataset, then makes a single prediction with the fit model.

```
# linear regression for multioutput regression
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
```

```
# create datasets
X, y = make_regression(n_samples=1000, n_features=10, n_informative=5, n_targets=2, random_stat
# define model
model = LinearRegression()
# fit model
model.fit(X, y)
# make a prediction
data_in = [[-2.02220122, 0.31563495, 0.82797464, -0.30620401, 0.16003707, -1.44411381, 0.876168
yhat = model.predict(data_in)
# summarize prediction
print(yhat[0])
```

Running the example fits the model and then makes a prediction for one input, confirming that the model predicted two required values.

```
[-93.147146 23.26985013]
```

k-Nearest Neighbors for Multioutput Regression

The example below fits a k-nearest neighbors model on the multioutput regression dataset, then makes a single prediction with the fit model.

```
# k-nearest neighbors for multioutput regression
from sklearn.datasets import make_regression
from sklearn.neighbors import KNeighborsRegressor
# create datasets
X, y = make_regression(n_samples=1000, n_features=10, n_informative=5, n_targets=2, random_stat
# define model
model = KNeighborsRegressor()
# fit model
model.fit(X, y)
# make a prediction
data_in = [[-2.02220122, 0.31563495, 0.82797464, -0.30620401, 0.16003707, -1.44411381, 0.876168
yhat = model.predict(data_in)
# summarize prediction
print(yhat[0])
```

Running the example fits the model and then makes a prediction for one input, confirming that the model predicted two required values.

```
[-109.74862659 0.38754079]
```

Random Forest for Multioutput Regression

The example below fits a random forest model on the multioutput regression dataset, then makes a single prediction with the fit model.

```
# random forest for multioutput regression
from sklearn.datasets import make_regression
from sklearn.ensemble import RandomForestRegressor
# create datasets
X, y = make_regression(n_samples=1000, n_features=10, n_informative=5, n_targets=2, random_stat
# define model
model = RandomForestRegressor()
# fit model
model.fit(X, y)
# make a prediction
data_in = [[-2.02220122, 0.31563495, 0.82797464, -0.30620401, 0.16003707, -1.44411381, 0.876168
yhat = model.predict(data_in)
```

```
# summarize prediction
print(yhat[0])
```

Running the example fits the model and then makes a prediction for one input, confirming that the model predicted two required values.

```
[-76.79505796 27.16551641]
```

Evaluate Multioutput Regression With Cross-Validation

We may want to evaluate a multioutput regression using k-fold cross-validation.

This can be achieved in the same way as evaluating any other machine learning model.

We will fit and evaluate a *DecisionTreeRegressor* model on the test problem using 10-fold cross-validation with three repeats. We will use the mean absolute error (MAE) performance metric as the score.

The complete example is listed below.

```
# evaluate multioutput regression model with k-fold cross-validation
from numpy import absolute
from numpy import mean
from numpy import std
from sklearn.datasets import make_regression
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
# create datasets
X, y = make_regression(n_samples=1000, n_features=10, n_informative=5, n_targets=2, random_stat
# define model
model = DecisionTreeRegressor()
# evaluate model
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, y, scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1, er
# summarize performance
n_scores = absolute(n_scores)
print('Result: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

Running the example evaluates the performance of the decision tree model for multioutput regression on the test problem. The mean and standard deviation of the MAE is reported calculated across all folds and all repeats.

Importantly, error is reported across both output variables, rather than separate error scores for each output variable.

```
Result: 51.659 (3.455)
```

Wrapper Multioutput Regression Algorithms

Not all regression algorithms support multioutput regression.

One example is the support vector machine, although for regression, it is referred to as support vector regression, or SVR.

This algorithm does not support multiple outputs for a regression problem and will raise an error. We can demonstrate this with an example, listed below.

```
# failure of support vector regression for multioutput regression
from sklearn.datasets import make_regression
from sklearn.svm import LinearSVR
# create datasets
X, y = make_regression(n_samples=1000, n_features=10, n_informative=5, n_targets=2, random_state
# define model
model = LinearSVR()
# fit model
model.fit(X, y)
```

Running the example reports an error message indicating that the model does not support multioutput regression.

```
ValueError: bad input shape (1000, 2)
```

There are two workarounds that we can adopt in order to use an algorithm like SVR for multioutput regression.

They are to create a separate model for each output and to create a linear sequence of models, one for each output, where the output of each model is dependent upon the output of the previous models.

Thankfully, the scikit-learn library supports both of these cases. Let's take a closer look at each.

Separate Model for Each Output (MultiOutputRegressor)

We can create a separate model for each output of the problem.

This assumes that the outputs are independent of each other, which might not be a correct assumption. Nevertheless, this approach can provide surprisingly effective predictions on a range of problems and may be worth trying, at least as a performance baseline.

You never know. The outputs for your problem may, in fact, be mostly independent, if not completely independent, and this strategy can help you find out.

This approach is supported by the MultiOutputRegressor class that takes a regression model as an argument. It will then create one instance of the provided model for each output in the problem.

The example below demonstrates using the *MultiOutputRegressor* class with linear SVR for the test problem.

```
# example of linear SVR with the MultiOutputRegressor wrapper for multioutput regression
from sklearn.datasets import make_regression
from sklearn.multioutput import MultiOutputRegressor
from sklearn.svm import LinearSVR
# create datasets
X, y = make_regression(n_samples=1000, n_features=10, n_informative=5, n_targets=2, random_stat
# define model
model = LinearSVR()
wrapper = MultiOutputRegressor(model)
# fit model
```

```
wrapper.fit(X, y)
# make a prediction
data_in = [[-2.02220122, 0.31563495, 0.82797464, -0.30620401, 0.16003707, -1.44411381, 0.876168
yhat = wrapper.predict(data_in)
# summarize prediction
print(yhat[0])
```

Running the example fits a separate LinearSVR for each of the outputs in the problem using the *MultiOutputRegressor* wrapper class.

This wrapper can then be used directly to make a prediction on new data, confirming that multiple outputs are supported.

```
[-93.147146 23.26985013]
```

Chained Models for Each Output (RegressorChain)

Another approach to using single-output regression models for multioutput regression is to create a linear sequence of models.

The first model in the sequence uses the input and predicts one output; the second model uses the input and the output from the first model to make a prediction; the third model uses the input and output from the first two models to make a prediction, and so on.

This can be achieved using the RegressorChain class in the scikit-learn library.

The order of the models may be based on the order of the outputs in the dataset (the default) or specified via the "*order*" argument. For example, *order=[0,1]* would first predict the 0th output, then the 1st output, whereas an *order=[1,0]* would first predict the last output variable and then the first output variable in our test problem.

The example below uses the *RegressorChain* with the default output order to fit a linear SVR on the multioutput regression test problem.

```
# example of fitting a chain of linear SVR for multioutput regression
from sklearn.datasets import make_regression
from sklearn.multioutput import RegressorChain
from sklearn.svm import LinearSVR
# create datasets
X, y = make_regression(n_samples=1000, n_features=10, n_informative=5, n_targets=2, random_stat
# define model
model = LinearSVR()
wrapper = RegressorChain(model)
# fit model
wrapper.fit(X, y)
# make a prediction
data_in = [[-2.02220122, 0.31563495, 0.82797464, -0.30620401, 0.16003707, -1.44411381, 0.876168
yhat = wrapper.predict(data_in)
# summarize prediction
print(yhat[0])
```

Running the example first fits a linear SVR to predict the first output variable, then a second linear SVR to predict the second output variable using the input and the output of the first model. These models are fit on the entire dataset.

The fit chain of models is then used directly to make a prediction on a new test instance, predicting the required two output variables.

```
[-93.147146 23.26938475]
```

Further Reading

This section provides more resources on the topic if you are looking to go deeper.

APIs

- Multiclass and multilabel algorithms, API.
- sklearn.datasets.make_regression API.
- sklearn.multioutput.MultiOutputRegressor API.
- sklearn.multioutput.RegressorChain API.

Summary

In this tutorial, you discovered how to develop machine learning models for multioutput regression.

Specifically, you learned:

- The problem of multioutput regression in machine learning.
- How to develop machine learning models that inherently support multiple-output regression.
- How to develop wrapper models that allow algorithms that do not inherently support multiple outputs to be used for multiple-output regression.

Do you have any questions?
Ask your questions in the comments below and I will do my best to answer.
