Learning Multivariate Adaptive Regression Splines (MARS) with Python

Authored by

Mohammed looti

November 6, 2025

The intricate world of statistical modeling frequently demands specialized techniques capable of accurately mapping complex, **nonlinear relationships** that prove elusive to standard linear approaches. A highly sophisticated and robust non-parametric regression methodology designed specifically to overcome these challenges is **Multivariate Adaptive Regression Splines** (MARS). MARS stands out due to its ability to model the connection between a comprehensive set of predictor variables and a **response variable**, particularly when the underlying data structure is unknown, highly fragmented, or involves complex interactions.

The inherent flexibility of MARS models is achieved through the systematic construction of a series of piecewise linear functions, known as basis functions, derived adaptively from the predictor variables. This modeling process is structured across three distinct and sequential conceptual stages, ensuring both comprehensive fit and subsequent parsimony:

The model initiates a **forward step**, characterized by an iterative addition of paired basis functions to the model. The objective of this phase is to aggressively minimize the residual sum of squares, often resulting in an intentionally overfitted model that captures all potential non-linearities. This continues until a predetermined maximum number of terms is incorporated.
Following the forward pass, the model executes a critical **backward step**, which involves pruning the overly complex structure. Terms are sequentially removed based on their contribution, guided by the **Generalized Cross-Validation** (GCV) criterion, ensuring the final model is both accurate and concise.
Finally, the selection of the optimal model complexity is finalized using a rigorous evaluation framework, frequently involving techniques like **k-fold cross-validation**, to validate the model's generalizability and guard against overfitting on the training data.

This comprehensive tutorial is designed to provide data scientists and analysts with a detailed, practical, and step-by-step methodology for implementing and evaluating a powerful **MARS model** within the widely used Python programming ecosystem, leveraging specialized libraries for a practical demonstration.

Understanding the Core MARS Algorithm

Multivariate Adaptive Regression Splines (MARS) were formally introduced by the distinguished statistician **Jerome H. Friedman** in 1991. What sets MARS apart from many other regression techniques is its inherent capability to automatically manage complex variable selection and detect intricate interaction effects simultaneously. At its core, MARS functions as an elegant extension of traditional linear models, utilizing localized, piecewise linear functions--known universally as **splines** or basis functions--to approximate the functional relationship between predictors and the target variable.

These powerful basis functions are defined by specific points called **knots** (or thresholds) located

within the predictor space. The strategic placement of these knots allows the MARS model to adapt its predictive structure locally, enabling it to model sharp shifts, curvatures, and discontinuities in the data that global models would miss. This adaptive quality ensures that the model can capture highly nuanced relationships without imposing rigid parametric assumptions.

The two-stage fitting process--the aggressive forward selection followed by the conservative backward pruning--is absolutely essential for achieving a balanced and robust final **regression model**. The initial forward pass is designed to intentionally maximize complexity, capturing every potential interaction and non-linearity. The subsequent backward phase is a regularization mechanism that uses the **Generalized Cross-Validation** (GCV) score to eliminate redundant terms, effectively steering the model toward a more parsimonious structure that maintains high predictive accuracy while minimizing the risk of **overfitting**.

The Mechanics of Basis Functions and Knot Selection

The mathematical foundation of MARS rests on its use of **hinge functions**, which constitute the primary basis set utilized for modeling. A hinge function is defined by a simple mathematical expression that allows for a change in the slope of the regression function at a specific point. Technically, a hinge function takes one of two forms: $max(0, x-t)$ or $max(0, t-x)$, where x represents a predictor variable and t denotes the location of a **knot**.

These piecewise linear functions introduce the necessary non-linearity into the model. When the input variable x is on one side of the knot t, the function outputs zero (or a constant slope), and when x crosses the knot, a new linear segment begins, allowing the relationship to bend. The power of the MARS algorithm lies in its intelligent, automated search capability, which systematically scans all variables and potential knot locations to identify the optimal set of basis functions that best approximate the observed data patterns.

Furthermore, MARS is adept at creating interaction terms by multiplying two or more basis functions together. For instance, a term might involve the product of a hinge function based on variable X_1 and another based on variable X_2. This mechanism allows the model to capture non-additive effects--situations where the effect of one variable on the response depends heavily on the value of another variable--without requiring the user to manually specify these complex interactions beforehand. The final model is simply a weighted sum of these selected basis functions and their interactions.

Step 1: Setting up the Python Environment and Dependencies

To successfully implement Multivariate Adaptive Regression Splines within the Python environment, we rely on the specialized third-party package known as **sklearn-contrib-py-earth**. This library is commonly referenced as **py-earth** and provides a high-performance implementation

of the MARS algorithm that is fully compatible with the widely accepted and standardized Scikit-learn API structure, making it intuitive for experienced Python machine learning practitioners.

Before initiating the modeling process, it is mandatory to ensure that this crucial package is installed within your active development environment. This can be accomplished swiftly using Python's package installer, **pip**.

Execute the following command directly within your terminal or development environment to install the necessary library, along with any required dependencies:

pip install sklearn-contrib-py-earth

Once the core MARS library is installed, the next stage involves importing all required modules for data handling, numerical computation, and model evaluation. We will integrate standard tools such as **pandas** for structured data manipulation, **numpy** for efficient numerical operations (like calculating the mean), and several critical utilities from the **sklearn** package, particularly those designated for cross-validation and convenient dataset generation.

```
import pandas as pd
from numpy import mean
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from sklearn.datasets import make_regression
from pyearth import Earth
```

In this sequence of imports, the **Earth()** function represents the central class that we will instantiate to construct and train the MARS model. Conversely, the **RepeatedKFold()** utility is essential for establishing a robust and statistically sound evaluation protocol, ensuring that our performance metrics are highly reliable and not susceptible to variations arising from arbitrary data partitioning.

Step 2: Creating a Sample Dataset for Modeling

For the purpose of clearly demonstrating the MARS implementation and its capabilities, it is highly advantageous to work with a controlled, synthetic dataset that embodies characteristics suitable for a challenging regression task. Leveraging the Scikit-learn function **make_regression()** enables us to quickly and programmatically generate a sufficiently large dataset with precise control over its complexity, the number of informative features, and the level of inherent noise, thereby simulating realistic data where underlying signals are often obscured by randomness.

In this specific example, we define the parameters to generate 5,000 distinct observations. The

resulting predictor matrix, denoted as **X**, will contain a total of 15 features. Crucially, we specify that only 10 of these features are genuinely informative, meaning the remaining 5 features are redundant or noise variables. This setup tests the MARS algorithm's inherent feature selection capabilities. Furthermore, we introduce a controlled amount of noise (set at 0.5) to mimic realistic measurement error and natural variability found in real-world data collection.

```
# Create synthetic regression data with specified complexity
X, y = make_regression(n_samples=5000, n_features=15, n_informative=10,
noise=0.5, random_state=5)
```

The resulting dataset pair (X, y) establishes a strong, controlled environment for rigorous testing. It specifically challenges the MARS algorithm to effectively detect the truly informative features, discard the noise variables, and accurately model the underlying non-linear signal despite the presence of confounding factors and inherent randomness.

Step 3: Building and Optimizing the MARS Model

With the data successfully generated and prepared, the next phase focuses on initializing the MARS model using the **Earth()** function and defining a highly resilient evaluation strategy. Given MARS's capacity for extreme flexibility and adaptation, it is paramount that its performance is assessed using resampling methods. This approach ensures that the model's accuracy metrics are reliable indicators of its ability to generalize effectively to completely unseen, out-of-sample data.

For this evaluation, we employ the **RepeatedKFold** cross-validation technique. This methodology involves splitting the entire dataset into k distinct folds (we use 10 folds here) and then repeating this entire cross-validation process multiple times (3 repeats). This yields a total of 30 independent performance estimates. This critical repetition helps to significantly mitigate the variance inherently associated with single, random splits of the data, providing a far more stable and trustworthy estimate of the model's true predictive power.

We define the cross-validation object and then utilize Scikit-learn's powerful **cross_val_score()** function to execute the full evaluation across all splits. The chosen performance measure is the **neg_mean_absolute_error** metric. The Negative Mean Absolute Error is utilized because the Scikit-learn cross-validation framework standardizes all scoring such that higher values consistently denote better performance. Therefore, the conventional **Mean Absolute Error** (MAE), which is an error metric where lower is better, must be negated to align with this standard convention.

```
# Define the MARS model using the Earth() class
model = Earth()
```

```
# Specify repeated k-fold cross-validation (10 splits, repeated 3 times)
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)

# Evaluate model performance using MAE across all splits
scores = cross_val_score(model, X, y, scoring='neg_mean_absolute_error',
cv=cv, n_jobs=-1)

# Calculate and print the average MAE across all 30 evaluations
mean(scores)
```

```
-1.745345918289
```

Interpreting Results and Comparing Model Performance

The resulting numerical output, which is the average **neg_mean_absolute_error** calculated across all 30 independent cross-validation folds, provides the crucial performance metric. Since we are fundamentally interested in the actual magnitude of the predictive error, we interpret this result by simply ignoring the negative sign. Thus, the resulting **Mean Absolute Error** (MAE) for this specific MARS model implementation is approximately **1.7453**. This value quantifies the average absolute distance between the response values predicted by the model and the true, observed values in the dataset.

A fundamental step in any statistical modeling project is the comparison of multiple candidate models to determine the most effective predictor. The MARS model, having achieved a robust MAE of 1.7453, can now be rigorously benchmarked against a variety of other established regression techniques. These comparison models might encompass simpler, parametric models such as **Multiple Linear Regression**, advanced regularization methods like Ridge or Lasso Regression, or alternative non-linear approaches such as Random Forests or Gradient Boosting Machines.

The model that consistently yields the lowest MAE when subjected to the identical cross-validation procedure is generally deemed the most accurate and reliable predictor for the specific underlying data structure. However, it is important to note that MAE is rarely the sole metric used for decision-making. Practitioners frequently employ a comprehensive suite of metrics to gain a complete understanding of model performance, including the **Adjusted R-squared** (which provides a balance between model fit and complexity) or the Mean Squared Error (MSE), which applies a greater penalty to large prediction errors. The definitive choice of metric is ultimately dictated by the specific scientific or business objectives of the analysis.

The complete, executable Python code used for this example, including the setup, data generation, and robust execution of the MARS model, is made available for detailed review and further

experimentation here.

Conclusion: The Strategic Value of MARS

The Multivariate Adaptive Regression Splines methodology offers a powerful and highly interpretable alternative to overly complex "black-box" machine learning models. By intelligently and automatically selecting the most relevant variables, identifying optimal knot locations, and constructing tailored **basis functions**, MARS achieves a high degree of predictive flexibility without sacrificing the critical interpretability often lost in algorithms such as neural networks or complex ensemble methods.

Its innate capability to model non-additive effects and interactions directly within the structure of its basis functions makes MARS exceptionally valuable in disciplines where relationships are frequently non-linear and complex. Fields such as **econometrics**, environmental engineering, and biostatistics regularly benefit from the transparent yet flexible modeling capabilities that MARS provides, allowing researchers to not only predict outcomes but also understand the specific mechanism driving those predictions.

Mastering the implementation and rigorous evaluation of MARS models using Python's robust statistical libraries is thus an essential and advanced skill set for modern data analysis professionals, providing the necessary tools to effectively tackle the most sophisticated non-linear regression challenges encountered in real-world data science applications.