Learning Lasso Regression with Python: A Step-by-Step Guide

Authored by

Mohammed looti

November 6, 2025

Lasso regression (Least Absolute Shrinkage and Selection Operator) is a powerful statistical technique primarily utilized in machine learning and statistics for fitting generalized linear models. Its key strength lies in its ability to perform both variable selection and regularization simultaneously, making it an indispensable tool when facing datasets plagued by high dimensionality or multicollinearity.

In traditional least squares regression, the goal is straightforward: determine the coefficient estimates that minimize the overall error between the observed data and the predicted values. This error is quantified by the Sum of Squared Residuals (RSS):

RSS = Σ(yi - ?i)2

Where these components represent the core of model fitting:

Σ: This Greek symbol denotes the mathematical operation of summation.
yi: The actual observed response value for the ith observation in the dataset.
?i: The predicted response value generated by the multiple linear regression model.

Conversely, Lasso regression introduces a constraint, known as an L1 penalty, to the minimization process. This constraint encourages the model to shrink the coefficient estimates, potentially forcing some coefficients exactly to zero. This distinct approach aims to minimize the following objective function:

RSS + λΣ|βj|

In this augmented equation, the index *j* spans from 1 to *p* predictor variables, and the parameter λ (lambda) must satisfy $\lambda \geq 0$. The second term, **λΣ|βj|**, is commonly referred to as the shrinkage penalty or regularization term.

The magnitude of the penalty is controlled by the hyperparameter λ. A larger value of λ results in greater shrinkage, potentially eliminating redundant features entirely. In practice, the primary task in utilizing Lasso regression is selecting the optimal λ value that yields the lowest possible test Mean Squared Error (MSE), ensuring the model generalizes well to unseen data. This tutorial provides a meticulous, step-by-step guide for implementing and optimizing Lasso regression using Python and the Scikit-learn library.

Step 1: Preparing the Environment and Importing Packages

Before we can begin the modeling process, it is essential to set up the Python environment by importing the necessary libraries. These packages provide the foundational structures for data manipulation, numerical operations, and machine learning algorithms.

For this specific implementation of Lasso regression, we rely heavily on the powerful ecosystems provided by Pandas and Scikit-learn (sklearn). Pandas facilitates efficient data handling, while sklearn provides the specialized regression tools and cross-validation utilities required for hyperparameter tuning.

The following code snippet imports four core components: Pandas for data frames, NumPy's `arange` function for generating the alpha sequence, `LassoCV` for the regression model, and `RepeatedKFold` for robust cross-validation:

```
import pandas as pd
from numpy import arange
from sklearn.linear_model import LassoCV
from sklearn.model_selection import RepeatedKFold
```

Step 2: Data Acquisition and Preparation

To demonstrate the practical application of Lasso regression, we will utilize the well-known **mtcars** dataset. This dataset compiles critical performance metrics for 33 distinct automobile models. Our modeling objective is to predict the car's horsepower (**hp**) based on several other vehicular characteristics.

We define the **hp** variable as our dependent or response variable. The independent variables, or predictors, chosen for this demonstration are intended to represent a combination of fuel economy, weight, and engine performance specifications:

mpg (Miles per Gallon)
wt (Weight)
drat (Rear Axle Ratio)
qsec (1/4 Mile Time)

The following Python code loads the data directly from a public URL using the Pandas library. After acquisition, we select the specific subset of variables required for our regression analysis and display the initial rows to confirm successful loading and structure:

```
#define URL where data is located
url = "https://raw.githubusercontent.com/Statology/Python-Guides/main/mtcars.csv"

#read in data
data_full = pd.read_csv(url)

#select subset of data
```

data = data_full]

#view first six rows of data
data

mpg wt drat qsec hp
0 21.0 2.620 3.90 16.46 110
1 21.0 2.875 3.90 17.02 110
2 22.8 2.320 3.85 18.61 93
3 21.4 3.215 3.08 19.44 110
4 18.7 3.440 3.15 17.02 175
5 18.1 3.460 2.76 20.22 105

Step 3: Implementing and Tuning the Lasso Regression Model

The core of this process involves using the `LassoCV()` function from sklearn. The "CV" suffix indicates that this function automatically incorporates cross-validation to search for the optimal regularization parameter. This automation significantly streamlines the hyperparameter tuning process compared to manually iterating through various alpha values.

In Python's Scikit-learn documentation, the regularization parameter λ (lambda) is typically referred to as **alpha**. The choice of alpha is critical, as it dictates the strength of the shrinkage penalty applied to the model coefficients. We must define a robust method for evaluating different alpha candidates to ensure we select the best model complexity.

We employ the `RepeatedKFold()` function to perform robust k-fold cross-validation. This technique involves splitting the training data into *k* partitions (or folds), training the model on *k*-1 folds, and validating on the remaining fold. We specify **k = 10 folds** and repeat this entire cross-validation procedure **3 times**. This repetition helps stabilize the results and mitigate the impact of random data partitioning, leading to a more reliable estimate of the model's performance.

By default, `LassoCV()` tests only a limited, broad range of alpha values. To ensure a granular search for the optimal penalty, we explicitly define a sequence of alpha values ranging from 0 (equivalent to standard least squares regression) up to 1, with precise increments of 0.01. This thorough search maximizes our chances of finding the alpha that minimizes the validation MSE.

#define predictor and response variables
X = data]
y = data

#define cross-validation method to evaluate model

```
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)

#define model, searching for optimal alpha across the defined range
model = LassoCV(alphas=arange(0, 1, 0.01), cv=cv, n_jobs=-1)

#fit model to the data
model.fit(X, y)

#display lambda (alpha) that produced the lowest test MSE
print(model.alpha_)
```

0.99

Upon execution, the model determines that the optimal regularization parameter, the one that minimizes the estimated test MSE across all cross-validation folds, is **0.99**. This value represents the best balance between model fit (low RSS) and coefficient shrinkage (L1 penalty).

Step 4: Utilizing the Optimized Model for Predictions

Once the Lasso regression model has been trained and optimized using the ideal alpha value determined through cross-validation, it is ready to be deployed. The final step involves using this rigorously tuned model to predict the response variable (horsepower) for new, unseen observations.

Consider a hypothetical new car for which we know the predictor attributes but need an estimate of its horsepower. We define a new observation vector based on the following specifications:

mpg: 24
wt: 2.5
drat: 3.5
qsec: 18.5

The following code demonstrates how to pass these new attributes into our fitted `LassoCV` model to generate a prediction for the horsepower (*hp*) value:

```
#define new observation vector (must match order of predictors used in training)
new =

#predict hp value using lasso regression model
model.predict()
```

array()

The output array indicates that, based on the input values and the relationships learned during the Lasso regression process, the model predicts this specific vehicle to have a horsepower (*hp*) value of approximately **105.63442071**. This concludes the demonstration of fitting and applying a Lasso model in Python.

Conclusion and Further Resources

Lasso regression provides a superior alternative to standard least squares regression when model simplicity and feature parsimony are priorities. By integrating the L1 regularization term, Lasso automatically handles feature selection, often resulting in more interpretable and robust models, especially in high-dimensional settings where multicollinearity might obscure true relationships.

The implementation using Scikit-learn's `LassoCV` simplifies the complex task of hyperparameter optimization, ensuring that the model selected is highly effective at minimizing the prediction MSE. Mastery of regularization techniques like Lasso is crucial for any data scientist building predictive models.

For those interested in reviewing the complete, executable Python script used throughout this tutorial, it is available via the following link: Lasso Regression Python Code.