

code for variable selection in multiple linear regression

Published: September 27, 2026 | Index ID: #-

Understanding Variable Selection in Multiple Linear Regression

When building a predictive model with multiple predictors, the goal is to capture the underlying relationship between the outcome and the independent variables while keeping the model as simple and interpretable as possible. Variable selection—choosing a subset of predictors that best explain the variation in the response—is a foundational step in this process. In this guide we'll explore why variable selection matters, the most common techniques, and, most importantly, provide **code for variable selection in multiple linear regression** in both Python and R. By the end, you'll be equipped to implement these methods in your own projects.

Why Variable Selection Matters

Multiple linear regression can accommodate any number of predictors, but blindly including every available variable can lead to several problems:

- Overfitting – The model captures noise rather than signal, performing poorly on new data.
- Multicollinearity – Highly correlated predictors inflate standard errors, making coefficient estimates unstable.
- Interpretability – A model with dozens of variables is hard to explain to stakeholders.
- Computational Efficiency – More variables mean longer training times and larger memory footprints.

Variable selection helps mitigate these issues by identifying a parsimonious set of predictors that maintain predictive power while enhancing model stability and interpretability.

Common Variable Selection Strategies

There are three primary families of variable selection methods:

1. Filter Methods – Use statistical criteria (e.g., correlation, p-values) to rank predictors before modeling.
2. Wrapper Methods – Evaluate subsets of variables by training a model and measuring performance (e.g., forward/backward selection).
3. Embedded Methods – Perform variable selection during model training by incorporating regularization penalties (e.g., LASSO).

Filter Methods: Correlation and P-Values

Filter methods are quick and easy but may overlook interactions and non-linear effects. A typical approach is to compute the Pearson correlation between each predictor and the outcome, then discard variables below a threshold. Alternatively, you can fit a full model and inspect the p-values of coefficients, removing those that are not statistically significant.

Wrapper Methods: Stepwise Selection

Wrapper methods iterate over subsets of variables, training a model at each step and evaluating performance. Two popular variations are:

- Forward Selection – Start with no variables and add the one that improves model fit the most.
- Backward Elimination – Start with all variables and iteratively remove the least useful one.

Stepwise selection can be combined with information criteria such as Akaike Information Criterion (AIC) or Bayesian Information Criterion (BIC) to decide when to stop adding or removing variables.

Embedded Methods: Regularization Techniques

Regularization introduces a penalty term to the loss function, shrinking coefficient estimates toward zero. This can automatically set some coefficients to exactly zero, effectively performing variable selection:

- LASSO (Least Absolute Shrinkage and Selection Operator) – Uses an L1 penalty, encouraging sparsity.
- Ridge Regression – Uses an L2 penalty; it shrinks coefficients but rarely sets them to zero.
- Elastic Net – Combines L1 and L2 penalties, balancing sparsity and grouping effects.

Because embedded methods consider all variables simultaneously, they can be more efficient and less prone to overfitting than wrapper approaches.

Implementing Variable Selection in Python

Below we provide code snippets that demonstrate how to perform variable selection in Python using scikit-learn and statsmodels. Each example includes data preparation, model fitting, and interpretation of results.

1. Filter Method: Correlation Threshold

We'll use the popular pandas library to compute correlations and filter variables.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

# Load dataset (replace with your own)
df = pd.read_csv('data.csv')

# Assume 'target' is the dependent variable
X = df.drop('target', axis=1)
y = df['target']

# Compute correlation matrix
corr_matrix = X.corrwith(y).abs()

# Set a threshold (e.g., 0.3)
threshold = 0.3
selected_features = corr_matrix[corr_matrix > threshold].index.tolist()

print(f"Selected features: {selected_features}")

# Train a simple linear regression model
X_train, X_test, y_train, y_test = train_test_split(
    X[selected_features], y, test_size=0.2, random_state=42
)
model = LinearRegression()
```

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

```
# Evaluate
```

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

```
print(f'R²: {r2_score(y_test, y_pred):.4f}')
```

This snippet demonstrates a quick filter approach. However, it ignores multicollinearity and interactions.

2. Wrapper Method: Forward Stepwise Selection

Below is a custom implementation of forward selection that uses adjusted R^2 to evaluate each step.

```
import numpy as np
```

```
import pandas as pd
```

```
import statsmodels.api as sm
```

```
from sklearn.model_selection import train_test_split
```

```
def forward_selection(X, y, threshold_in=0.01):
```

```
    initial_features = []
```

```
    best_features = []
```

```
    while True:
```

```
        changed = False
```

```
        scores_with_candidates = []
```

```
        for candidate in X.columns.difference(initial_features):
```

```
            features = initial_features + [candidate]
```

```
            X_train = sm.add_constant(X[features])
```

```
            model = sm.OLS(y, X_train).fit()
```

```
            score = model.rsquared_adj
```

```
            scores_with_candidates.append((score, candidate))
```

```
        scores_with_candidates.sort(reverse=True)
```

```
        best_new_score, best_candidate = scores_with_candidates[0]
```

```
        if not best_features or best_new_score - best_features[-1][0] > threshold_in:
```

```
            initial_features.append(best_candidate)
```

```
            best_features.append((best_new_score, best_candidate))
```

```
            changed = True
```

```
        if not changed:
```

```
            break
```

```
    return best_features, initial_features
```

```
# Example usage
```

```
df = pd.read_csv('data.csv')
```

```
X = df.drop('target', axis=1)
```

```
y = df['target']
```

```
best_scores, selected = forward_selection(X, y)
```

```
print("Selected variables in order of addition:")
for score, var in best_scores:
    print(f"{var} (Adj R²: {score:.4f})")
```

Forward selection can be computationally expensive for many predictors, but it often yields a good trade-off between model complexity and performance.

3. Embedded Method: LASSO Regularization

LASSO automatically shrinks some coefficients to zero. We'll use scikit-learn's LassoCV for cross validated alpha selection.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LassoCV
from sklearn.metrics import r2_score

# Load data
df = pd.read_csv('data.csv')
X = df.drop('target', axis=1)
y = df['target']

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train LASSO with cross validation
lasso = LassoCV(cv=5, random_state=42).fit(X_scaled, y)

print(f"Optimal alpha: {lasso.alpha_:.4f}")

# Identify non zero coefficients
coef = pd.Series(lasso.coef_, index=X.columns)
selected_features = coef[coef != 0].index.tolist()
print(f"Selected features: {selected_features}")

# Evaluate on a hold out set
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)
lasso_final = LassoCV(alphas=[lasso.alpha_], cv=5, random_state=42).fit(X_train, y_train)
y_pred = lasso_final.predict(X_test)
print(f"R² on test set: {r2_score(y_test, y_pred):.4f}")
```

Because LASSO works in the standardized feature space, the magnitude of coefficients reflects variable

importance. You can also use ElasticNetCV if you suspect groups of correlated predictors.

4. Cross Validated Stepwise Selection with statsmodels

While statsmodels doesn't provide built in stepwise selection, you can wrap it with cross validation to prevent overfitting. Here's a concise example using statsmodels's OLS and cross_val_score from sklearn.model_selection.

```
import pandas as pd
import statsmodels.api as sm
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import make_scorer
```

```
df = pd.read_csv('data.csv')
X = df.drop('target', axis=1)
y = df['target']
```

```
# Define a custom scorer that returns adjusted R2
```

```
def adj_r2_scorer(estimator, X, y):
    X_const = sm.add_constant(X)
    model = sm.OLS(y, X_const).fit()
    return model.rsquared_adj
```

```
adj_r2 = make_scorer(adj_r2_scorer, greater_is_better=True)
```

```
# Simple forward selection with cross validation
```

```
def forward_cv(X, y, cv=5):
    remaining = set(X.columns)
    selected = []
    best_score = -np.inf
    while remaining:
        scores = []
        for candidate in remaining:
            trial = selected + [candidate]
            X_trial = X[trial]
            score = cross_val_score(
                sm.OLS(y, sm.add_constant(X_trial)),
                X_trial, y, cv=cv, scoring=adj_r2
            ).mean()
            scores.append((score, candidate))
        scores.sort(reverse=True)
        best_candidate_score, best_candidate = scores[0]
        if best_candidate_score > best_score:
            selected.append(best_candidate)
            remaining.remove(best_candidate)
            best_score = best_candidate_score
        else:
```

```
break
return selected
```

```
selected_vars = forward_cv(X, y, cv=5)
print(f"Variables selected via CV based forward selection: {selected_vars}")
```

Although this example is more involved, it demonstrates how to combine statsmodels's statistical output with scikit-learn's cross validation utilities.

Implementing Variable Selection in R

R's statistical ecosystem offers a wealth of functions for variable selection. Below we present several popular methods, each accompanied by concise code.

1. Filter Method: Correlation Matrix

Using the `cor()` function, you can compute correlations and filter predictors.

```
# Load dataset
df <- read.csv('data.csv')

# Compute correlation with target
corrs <- abs(cor(df$target, df[-'target']))
threshold <- 0.3
selected <- names(corrs)[corrs > threshold]
print(selected)
```

2. Wrapper Method: Stepwise Selection via `step()`

The `step()` function implements both forward and backward selection based on AIC.

```
# Full model
full_model <- lm(target ~ ., data = df)

# Null model
null_model <- lm(target ~ 1, data = df)

# Stepwise selection
step_model <- step(null_model, scope = list(lower = null_model, upper = full_model),
```

Baca Juga (Artikel Terkait)

- [class 6 math solution in bangladesh](http://www.bbgglobalsolutions.com/class-6-math-solution-in-bangladesh.pdf)

URL: <http://www.bbgglobalsolutions.com/class-6-math-solution-in-bangladesh.pdf>

- [class 6 math solution bd ntclan](http://www.bbgglobalsolutions.com/class-6-math-solution-bd-ntclan.pdf)

URL: <http://www.bbgglobalsolutions.com/class-6-math-solution-bd-ntclan.pdf>

- [class 5 question papers in bd psc](http://www.bbgglobalsolutions.com/class-5-question-papers-in-bd-psc.pdf)

URL: <http://www.bbgglobalsolutions.com/class-5-question-papers-in-bd-psc.pdf>

- [class 6 books pdf dav](http://www.bbgglobalsolutions.com/class-6-books-pdf-dav.pdf)

URL: <http://www.bbgglobalsolutions.com/class-6-books-pdf-dav.pdf>

Tags: `#code` `#variable` `#selection` `#multiple` `#linear` `#regression`

