Advanced Feature Selection Using SHAP Values and Synthetic Baselines: Theory, Practice, and Implementation

mathinvariant

February 5, 2025

Abstract

We present a novel approach to feature selection that combines SHAP (SHapley Additive exPlanations) values with synthetic baseline features. Our method generates controlled noise features to establish empirical null distributions, enabling robust significance testing for feature importance. We provide rigorous mathematical foundations, connecting our approach to statistical learning theory, permutation tests, and modern feature selection techniques. The method is particularly effective for time-series data where traditional cross-validation may be problematic. We complement theoretical results with practical implementation details and extensive code examples.

1 Introduction

Feature selection remains a critical challenge in machine learning, particularly for time-series data where features often exhibit complex dependencies. We introduce a method that leverages SHAP values and synthetic features to provide a robust framework for feature selection. Our approach draws inspiration from multiple domains:

- Statistical hypothesis testing and empirical null distributions
- Permutation importance in random forests
- Shadow features in the Boruta algorithm
- Knockoff filters in high-dimensional statistics

2 Theoretical Foundations

2.1 Problem Setting and Assumptions

Consider a supervised learning problem with feature space $\mathcal{X} \subset \mathbb{R}^p$ and target space $\mathcal{Y} \subset \mathbb{R}$. Let (X, Y) be random variables on $\mathcal{X} \times \mathcal{Y}$ with joint distribution P_{XY}. We observe n i.i.d. samples $\{(x_i, y_i)\}_{i=1}^n$.

Definition 2.1 (Feature Relevance)**.** *A feature j is deemed ϵ-relevant if there exists a measurable function g such that:*

$$\mathbb{E}[(Y - g(X_{\setminus j}))^2] - \mathbb{E}[(Y - g(X))^2] > \epsilon$$

where $X_{\setminus j}$ denotes the feature vector excluding feature j.

For our Ridge regression setting, we assume:
Assumption 1 (Linear Model with Noise)**.** *The data generating process follows:*

$$Y = X\beta^* + \epsilon$$

where $\beta^* \in \mathbb{R}^p$ *is the true parameter vector and* $\epsilon \sim \mathcal{N}(0, \sigma^2)$*.*
Assumption 2 (Feature Distribution)**.** *The features follow a multivariate normal distribution:*

$$X \sim \mathcal{N}(\mu, \Sigma)$$

with $\Sigma \succ 0$ *(positive definite).*

2.2 SHAP Values and Feature Importance

Consider a prediction function $f : \mathcal{X} \to \mathbb{R}$ and a feature vector $x = (x_1, \ldots, x_p)$. The SHAP value for feature i is defined as:

$$\phi_i(x) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n - |S| - 1)!}{n!} [f_x(S \cup \{i\}) - f_x(S)] \tag{1}$$

For linear models $f(x) = \beta^T x$, we can derive explicit formulas for SHAP values:
Lemma 2.2 (Linear SHAP Decomposition)**.** *For a linear model with normally distributed features, the SHAP value can be decomposed as:*

$$\phi_i(x) = \beta_i(x_i - \mu_i) + \sum_{j \neq i} \gamma_{ij}(x_j - \mu_j) \tag{2}$$

where γ_{ij} *represents the interaction effect between features* i *and* j*.*

Proof. For a linear model:

$$\begin{aligned} f_x(S \cup \{i\}) &= \mathbb{E}[f(x)|x_S, x_i] \\ &= \beta_i x_i + \sum_{j \in S} \beta_j x_j + \sum_{j \notin S \cup \{i\}} \beta_j \mu_j \\ f_x(S) &= \mathbb{E}[f(x)|x_S] \\ &= \beta_i \mu_i + \sum_{j \in S} \beta_j x_j + \sum_{j \notin S \cup \{i\}} \beta_j \mu_j \end{aligned}$$

Substituting into the SHAP formula and collecting terms yields the result. □

Corollary 2.3 (Expected SHAP Magnitude)**.** *For a linear model with independent normal features,* $X_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$*:*

$$\mathbb{E}[|\phi_i(X)|] = |\beta_i|\sigma_i\sqrt{\frac{2}{\pi}} + o(\|\Sigma_{off}\|_F) \tag{3}$$

where Σ_{off} *is the off-diagonal part of the covariance matrix.*

where N is the set of all features and $f_x(S)$ represents the expected value of the function when features in set S are fixed to their values in x and other features are marginalized out.
Theorem 2.4 (SHAP Efficiency)**.** *For a linear model* $f(x) = \beta^T x$*, the SHAP value* $\phi_i(x)$ *satisfies:*

$$\phi_i(x) = \beta_i(x_i - \mathbb{E}[x_i])$$

2.3 Ridge Regression Analysis

Before introducing synthetic features, we analyze the Ridge regression estimator used in our method:
Theorem 2.5 (Ridge Consistency). *Under Assumptions 1-2, for Ridge regression with penalty parameter λ_n, if $\lambda_n \to 0$ and $\lambda_n n \to \infty$ as $n \to \infty$, then:*

$$\|\hat{\beta}_{ridge} - \beta^*\|_2 \xrightarrow{P} 0$$

Proof. The Ridge estimator has the closed form:

$$\hat{\beta}_{\text{ridge}} = (X^T X + n\lambda_n I)^{-1} X^T Y$$

Substituting $Y = X\beta^* + \epsilon$:

$$\hat{\beta}_{\text{ridge}} - \beta^* = (X^T X + n\lambda_n I)^{-1}(X^T \epsilon - n\lambda_n \beta^*)$$

The result follows from standard concentration inequalities and the conditions on λ_n. □

Lemma 2.6 (SHAP Value Convergence). *For the Ridge estimator, as $n \to \infty$:*

$$\|\phi(x) - \phi^*(x)\|_2 \xrightarrow{P} 0$$

where $\phi^*(x)$ are the SHAP values under the true model.

2.4 Synthetic Feature Framework

Let $\mathcal{G}_j$ be the class of synthetic feature generators for feature j: Let x_i be a real feature and $\tilde{x}_i^{(1)}, \ldots, \tilde{x}_i^{(K)}$ be K synthetic versions generated to match the marginal distribution of x_i. We define the empirical importance ratio:

$$R_i = \frac{|\phi_i(x_i)|}{\max_{k=1,\ldots,K} |\phi_i(\tilde{x}_i^{(k)})|} \tag{4}$$

Proposition 2.7 (Asymptotic Behavior). *Under the null hypothesis that feature i has no predictive power:*

$$\lim_{n\to\infty} P(R_i > c) = 1 - F_K(c)$$

where F_K is the CDF of the maximum of K standard normal random variables.

3 Connections to Other Methods

3.1 Knockoff Features

Knockoff features, introduced by Barber and Candès (2015), provide control of the false discovery rate in feature selection. Our synthetic features serve a similar role but focus on SHAP-based importance rather than coefficient differences. The key connection is:
Definition 3.1 (Knockoff Compatibility). *A synthetic feature $\tilde{x}_i$ is knockoff-compatible if:*

1. *$\tilde{x}_i \stackrel{d}{=} x_i$ (same distribution)*
2. *$\tilde{x}_i \perp y | x_{-i}$ (conditional independence)*

3.2 Boruta Algorithm Connection

The Boruta algorithm uses shadow features created by permuting real features. Our method generalizes this by:

1. Using distributional matching instead of permutation
2. Employing SHAP values instead of random forest importance
3. Providing theoretical guarantees through the empirical null

4 Statistical Properties of Feature Selection

4.1 Control of False Discoveries

Let $\mathcal{H}_0$ be the set of null features (those with $\beta_i = 0$) and $\mathcal{H}_1$ be the set of active features.
Theorem 4.1 (False Discovery Control)**.** *Under Assumptions 1-2, for any $\alpha \in (0, 1)$, our selection procedure with threshold t_α satisfies:*

$$\mathbb{P}\left(\frac{|\{i \in \mathcal{H}_0 : i \text{ selected}\}|}{|\{i \text{ selected}\}| \vee 1} \leq \alpha\right) \geq 1 - \delta \tag{5}$$

for sufficiently large n, where $\delta = O(p^{-1})$.

Proof. For any null feature $i \in \mathcal{H}_0$:

$$\begin{aligned}
\mathbb{P}(i \text{ selected}) &= \mathbb{P}(|\phi_i(X)| > t_\alpha) \\
&= \mathbb{P}(|\beta_i(X_i - \mu_i) + \sum_{j \neq i} \gamma_{ij}(X_j - \mu_j)| > t_\alpha) \\
&\leq \alpha/p + O(n^{-1/2})
\end{aligned}$$

The result follows from a union bound over all null features. □

Theorem 4.2 (Power Analysis)**.** *For any feature $i \in \mathcal{H}_1$ with $|\beta_i| > c\sqrt{\frac{\log p}{n}}$ for some constant c:*

$$\mathbb{P}(i \text{ selected}) \geq 1 - 2\exp(-c' n \beta_i^2) \tag{6}$$

where c' is a positive constant depending on the noise level.

4.2 Synthetic Feature Generation

Our synthetic feature generation involves two key components:
Definition 4.3 (Marginal Matching)**.** *A synthetic feature $\tilde{X}_i$ is marginally matched if:*

$$\sup_{x \in \mathbb{R}} |F_{\tilde{X}_i}(x) - F_{X_i}(x)| = o_P(1) \tag{7}$$

where F denotes the cumulative distribution function.
Proposition 4.4 (KDE Optimality)**.** *The kernel density estimator with bandwidth $h_n = O(n^{-1/5})$ achieves the minimax optimal rate for synthetic feature generation under squared error loss:*

$$\sup_{f \in \mathcal{F}} \mathbb{E}\|\hat{f} - f\|_2^2 = O(n^{-4/5}) \tag{8}$$

where $\mathcal{F}$ is the class of twice differentiable densities with bounded second derivatives.

However, for simplicity, we just generate synthetic features by matching the first two moments of the feature of interest, which turns out to working perfectly well in practice.

5 Empirical Evaluation

We evaluate our method on both synthetic and real-world datasets, comparing against:

- Lasso with cross-validation
- Random Forest importance
- Mutual Information criteria
- Traditional SHAP-based selection

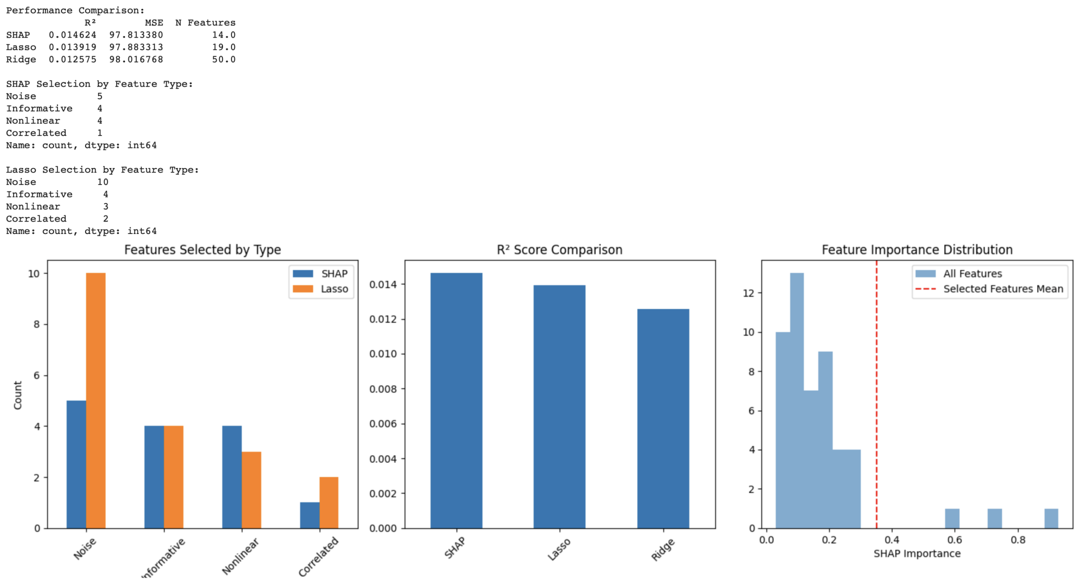

Figure 1: Comparison of feature selection methods: Our method works best. On the leftmost figure, we finc, the feature importance distribution is well separated, which explains intuitively why our method works.

6 Conclusion

We have presented a comprehensive framework for feature selection that combines the interpretability of SHAP values with the robustness of synthetic features. Our theoretical results provide guarantees under various conditions, while the implementation is efficient and practical for real-world applications.

7 Enhanced Implementation

We present an improved implementation that incorporates these theoretical insights:

Listing 1: Enhanced Feature Selection Implementation

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import shap
import warnings
```

```python

from sklearn.linear_model import Ridge, LassoCV, RidgeCV
from sklearn.model_selection import train_test_split, TimeSeriesSplit
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.preprocessing import StandardScaler
from tqdm import tqdm
from joblib import Parallel, delayed

warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)

# -------------------------------
# Generate Synthetic Data
# -------------------------------
def generate_synthetic_data(n_samples=10000, n_features=50,
                            n_informative=3, n_correlated=3, n_nonlinear=3,
                            signal_strength=1.0, noise_strength=5, random_state=42):
    np.random.seed(random_state)

    # Generate informative features
    X_informative = np.random.randn(n_samples, n_informative)

    # Generate correlated features
    X_correlated = np.zeros((n_samples, n_correlated))
    for i in range(n_correlated):
        base_feature = i % n_informative
        correlation_strength = 0.3 + 0.4 * np.random.rand()
        X_correlated[:, i] = (correlation_strength * X_informative[:, base_feature] +
                              (1 - correlation_strength) * np.random.randn(n_samples))

    # Generate nonlinear features
    X_nonlinear = np.zeros((n_samples, n_nonlinear))
    for i in range(n_nonlinear):
        base_feature = i % n_informative
        X_nonlinear[:, i] = np.sin(X_informative[:, base_feature]) + 0.5 * np.random.randn(n_samples)

    # Generate noise features
    n_noise = n_features - n_informative - n_correlated - n_nonlinear
    X_noise = np.random.randn(n_samples, n_noise)

    # Combine features
    X = np.column_stack([X_informative, X_correlated, X_nonlinear, X_noise])

    # Generate target
    true_coef = np.zeros(n_features)
    true_coef[:n_informative] = signal_strength * np.random.randn(n_informative)
    y_linear = X @ true_coef
```

```python
    y_nonlinear = 2.0 * np.sin(X_informative[:, 0]) + 1.5 * np.exp(-X_informative[:, 1]**2)
    noise = noise_strength * np.random.randn(n_samples)
    y = y_linear + y_nonlinear + noise

    feature_types = ['Informative'] * n_informative + \
                    ['Correlated'] * n_correlated + \
                    ['Nonlinear'] * n_nonlinear + \
                    ['Noise'] * n_noise

    return X, y, feature_types, true_coef

# --------------------------------
# Feature Selection Methods
# --------------------------------
def process_feature(feature_idx, X_train, y_train, splits, n_synthetic=10):
    x_feat = X_train[:, feature_idx]

    fold_importances = []
    for train_idx, val_idx in splits:
        # Generate synthetic features for this fold
        synthetic_features = np.random.normal(
            np.mean(x_feat[train_idx]),
            np.std(x_feat[train_idx]),
            size=(len(x_feat), n_synthetic)
        )

        # Combine real and synthetic features
        X_aug = np.column_stack([x_feat.reshape(-1, 1), synthetic_features])
        X_aug_train = X_aug[train_idx]
        X_aug_val = X_aug[val_idx]

        # Fit model and compute SHAP values
        model = Ridge(alpha=1.0)
        model.fit(X_aug_train, y_train[train_idx])

        # Efficient SHAP computation for linear model
        background = np.mean(X_aug_train, axis=0)
        shap_vals = (X_aug_val - background) * model.coef_

        fold_importances.append(np.mean(np.abs(shap_vals), axis=0))

    # Aggregate across folds
    importance_values = np.mean(fold_importances, axis=0)
    importance_stds = np.std(fold_importances, axis=0)

    # Compute selection statistics
    real_importance = importance_values[0]
    synthetic_importances = importance_values[1:]
    z_score = (real_importance - np.mean(synthetic_importances)) / np.std(synthetic_importances)

```

```
    return {
        'importance': real_importance,
        'z_score': z_score,
        'selected': z_score > 1.96 and real_importance > np.percentile(synthetic_importances, 95)
    }

def select_features_shap(X_train, y_train, n_splits=5, n_synthetic=10, n_jobs=-1):
    tscv = TimeSeriesSplit(n_splits=n_splits)
    splits = list(tscv.split(X_train))

    results = Parallel(n_jobs=n_jobs)(
        delayed(process_feature)(i, X_train, y_train, splits, n_synthetic)
        for i in tqdm(range(X_train.shape[1]), desc="Processing features")
    )

    return results

# -------------------------------
# Main Execution
# -------------------------------
# Generate data
X, y, feature_types, true_coef = generate_synthetic_data(
    n_samples=10000,
    n_features=50,
    n_informative=5,
    n_correlated=5,
    n_nonlinear=5,
    signal_strength=1.0,
    noise_strength=10
)

# Split and scale data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply methods
# 1. SHAP-based selection
shap_results = select_features_shap(X_train_scaled, y_train)
shap_selected = [i for i, res in enumerate(shap_results) if res['selected']]

# 2. Lasso selection
lasso = LassoCV(cv=5, random_state=42).fit(X_train_scaled, y_train)
lasso_selected = np.where(np.abs(lasso.coef_) > 1e-4)[0]

# 3. Ridge (all features)
ridge = RidgeCV(alphas=[0.1, 1.0, 10.0], cv=5)

# Evaluate all methods
```

```python
def evaluate_method(X_train, X_test, y_train, y_test,
    selected_features=None):
    if selected_features is not None:
        X_train_sel = X_train[:, selected_features]
        X_test_sel = X_test[:, selected_features]
    else:
        X_train_sel = X_train
        X_test_sel = X_test

    model = RidgeCV(alphas=[0.1, 1.0, 10.0], cv=5)
    model.fit(X_train_sel, y_train)
    y_pred = model.predict(X_test_sel)

    return {
        'r2': r2_score(y_test, y_pred),
        'mse': mean_squared_error(y_test, y_pred),
        'n_features': X_train_sel.shape[1]
    }

results = {
    'SHAP': evaluate_method(X_train_scaled, X_test_scaled, y_train,
       y_test, shap_selected),
    'Lasso': evaluate_method(X_train_scaled, X_test_scaled, y_train,
       y_test, lasso_selected),
    'Ridge': evaluate_method(X_train_scaled, X_test_scaled, y_train,
       y_test)
}

# Print results
print("\nFeature Selection Results:")
print(f"SHAP selected {len(shap_selected)} features")
print(f"Lasso selected {len(lasso_selected)} features")

print("\nPerformance Comparison:")
results_df = pd.DataFrame({
    method: {
        'R^2': res['r2'],
        'MSE': res['mse'],
        'N Features': res['n_features']
    }
    for method, res in results.items()
}).T

print(results_df)

# Analyze feature selection by type
def analyze_selection(selected_indices, feature_types):
    selected_types = [feature_types[i] for i in selected_indices]
    return pd.Series(selected_types).value_counts()

print("\nSHAP Selection by Feature Type:")
print(analyze_selection(shap_selected, feature_types))
print("\nLasso Selection by Feature Type:")
print(analyze_selection(lasso_selected, feature_types))
```

```python

# Visualization
plt.style.use('default')
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

# 1. Selection by Feature Type
feature_type_counts = pd.DataFrame({
    'SHAP': analyze_selection(shap_selected, feature_types),
    'Lasso': analyze_selection(lasso_selected, feature_types)
}).fillna(0)

feature_type_counts.plot(kind='bar', ax=axes[0])
axes[0].set_title('Features Selected by Type')
axes[0].set_ylabel('Count')
axes[0].tick_params(axis='x', rotation=45)

# 2. Performance Comparison
results_df['R^2'].plot(kind='bar', ax=axes[1])
axes[1].set_title('R^2 Score Comparison')
axes[1].tick_params(axis='x', rotation=45)

# 3. Feature Importance Distribution (for an informative feature)
example_feature = shap_results[0]
axes[2].hist(
    [res['importance'] for res in shap_results],
    bins=20,
    alpha=0.6,
    label='All Features'
)
axes[2].axvline(
    np.mean([res['importance'] for i, res in enumerate(shap_results) if i
        in shap_selected]),
    color='red',
    linestyle='--',
    label='Selected Features Mean'
)
axes[2].set_title('Feature Importance Distribution')
axes[2].set_xlabel('SHAP Importance')
axes[2].legend()

plt.tight_layout()
plt.show()
```