CASI Chap 7
MA 677

Yuyang Sun

May 7, 2024

1. James-Stein estimator

1.1 Main Points

Suppose we need to estimate a single parameter θ from observations x in a Bayesian setting, where $\theta \sim N(M, A)$ and $x|\theta \sim N(\theta, 1)$. The posterior distribution of θ is given by:

$$\theta|x \sim N\left(M + \frac{B(x-M)}{A+B}, \frac{1}{A+B}\right)$$

The Bayes estimator of θ, denoted as $\hat{\theta}_{\text{Bayes}}$, has an expected squared error of B, compared to 1 for the Maximum Likelihood Estimator (MLE) $\hat{\theta}_{\text{MLE}} = x$.

For N independent versions of the above scenario, where $\theta_i \sim N(M, A)$ and $x_i|\theta_i \sim N(\theta_i, 1)$, with $i = 1, 2, \ldots, N$, the total squared error risk of $\hat{\theta}_{\text{Bayes}}$ is NB, compared to N for $\hat{\theta}_{\text{MLE}}$.

The James-Stein estimator $\hat{\theta}_{\text{JS}}$ is a plug-in version of the Bayes estimator and is given by $\hat{\theta}_{\text{JS}} = M + B(x - M)$.

1.2 Computational Methods and Mathematics

The computations involved in implementing the James-Stein estimator can be efficiently carried out using R or Python packages for Bayesian inference, such as `rstan` in R or `PyMC3` in Python. These packages provide tools for fitting Bayesian models and estimating posterior distributions.

The mathematics underlying the material in this chapter involve Bayesian inference, conditional distributions, and decision theory.

1.3 Historical Context

The James-Stein estimator, introduced in 1961 by James and Stein, challenged the dominance of Maximum Likelihood Estimation (MLE) in statistical practice. The James-

Stein theorem demonstrated that the MLE is inadmissible in certain scenarios, leading to a paradigm shift in statistical estimation.

Fisher's "logic of inductive inference" initially promoted MLE as the preferred estimator, but the James-Stein result showed its limitations, particularly in high-dimensional problems. This theorem was a significant development in statistical theory and practice.

1.4 Statistical Practice Implications

The James-Stein estimator highlights the importance of shrinkage estimation in high-dimensional problems. While MLE remains relevant in low-dimensional scenarios, shrinkage methods are indispensable in modern statistical practice, especially when dealing with large datasets or complex models.

1.5 Theoretical Background

The classical James-Stein estimator can be expressed as:

$$\hat{\theta}_{JS} = \left(1 - \frac{(p-2)\sigma^2}{\|\hat{\theta}\|^2}\right)\hat{\theta}$$

where:

- $\hat{\theta}$ is the vector of sample means.
- p is the number of dimensions.
- σ^2 is the known variance of each dimension.
- $\|\hat{\theta}\|^2$ is the squared Euclidean norm of $\hat{\theta}$.

This formula shows that the James-Stein estimator pulls the sample means towards zero, which reduces estimation error when the true means are close to each other and/or close to zero.

1.6 Python Implementation

The following Python script demonstrates how to apply the James-Stein estimator:

```
import numpy as np

def james_stein_estimator(sample_means, sigma_squared):
    """
    Apply the James-Stein estimator to improve the estimation of means.

    Parameters:
```

```
    sample_means (numpy array): The vector of sample means.
    sigma_squared (float): The common variance of the observations.

    Returns:
    numpy array: The James-Stein adjusted means.
    """
    p = len(sample_means)
    if p <= 2:
        return sample_means  # James-Stein estimator is not effective when p <= 2
    norm_squared = np.sum(sample_means**2)
    shrinkage_factor = (p - 2) * sigma_squared / norm_squared
    return (1 - shrinkage_factor) * sample_means

# Example usage
sample_means = np.array([3.0, -2.5, 1.5, 0.5])
sigma_squared = 2.0  # Known common variance
adjusted_means = james_stein_estimator(sample_means, sigma_squared)
print("Adjusted Means:", adjusted_means)
```

2. Ridge Regression

Linear regression, perhaps the most widely used estimation technique, is based on a version of the ordinary maximum likelihood estimate (MLE). In the usual notation, we observe an n-dimensional vector $\mathbf{y} = (y_1, y_2, \ldots, y_n)^T$ from the linear model

$$\mathbf{y} = \mathbf{X}\beta + \epsilon$$

Here $\mathbf{X}$ is a known $n \times p$ structure matrix, β is an unknown p-dimensional parameter vector, while the noise vector $\epsilon = (\epsilon_1, \epsilon_2, \ldots, \epsilon_n)^T$ has its components uncorrelated and with constant variance σ^2,

$$\epsilon \sim (0, \sigma^2\mathbf{I})$$

where $\mathbf{I}$ is the $n \times n$ identity matrix. Often ϵ is assumed to be multivariate normal,

$$\epsilon \sim \mathcal{N}_n(0, \sigma^2\mathbf{I})$$

but that is not required for most of what follows.

James–Stein Estimation and Ridge Regression

The least squares estimate $\hat{\beta}_{\text{OLS}}$, going back to Gauss and Legendre in the early 1800s, is the minimizer of the total sum of squared errors,

$$\hat{\beta}_{\text{OLS}} = \arg\min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|^2$$

It is given by

$$\hat{\beta}_{\text{OLS}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

where $S = \mathbf{X}^T\mathbf{X}$ is the $p \times p$ inner product matrix

$$S = \mathbf{X}^T\mathbf{X}$$

$\hat{\beta}_{\text{OLS}}$ is unbiased for β and has covariance matrix $\sigma^2 S^{-1}$,

$$\hat{\beta}_{\text{OLS}} \sim \mathcal{N}_p(\beta, \sigma^2 S^{-1})$$

In the normal $\hat{\beta}_{\text{OLS}}$ is the MLE of β. Before 1950 a great deal of effort went into designing matrices $\mathbf{X}$ such that S^{-1} could be feasibly calculated, which is now no longer a concern.

Ridge regression is a shrinkage method designed to improve the estimation of β in linear models. By transformations, we can standardize the columns of $\mathbf{X}$ each have mean 0 and sum of squares 1, that is,

$$S_{ii} = 1 \quad \text{for } i = 1, 2, \ldots, p$$

For convenience, we also assume $\bar{y} = 0$. A ridge regression estimate $\hat{\beta}(\lambda)$ is defined, for $\lambda \geq 0$, to be

$$\hat{\beta}(\lambda) = (S + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} = (S + \lambda\mathbf{I})^{-1}S\hat{\beta}_{\text{OLS}}$$

$\hat{\beta}(\lambda)$ is a shrunken version of $\hat{\beta}_{\text{OLS}}$, the bigger λ the more extreme the shrinkage: $\hat{\beta}(0) = \hat{\beta}_{\text{OLS}}$ while $\hat{\beta}(1)$ equals the vector of zeros.

2.1 Dataset and Model

We consider a dataset where each entry represents a house with features such as area, number of bedrooms, age of the house, and proximity to essential amenities. The target variable is the house price. We will use the 'sklearn' library in Python to implement Ridge Regression.

2.2 Ridge Regression Example

Here is a Python example implementing Ridge Regression with the sklearn library, demonstrating how to predict house prices from a dataset:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error

# Load the dataset
data = pd.read_csv('housing_data.csv')
```

```
# Features and target
X = data[['area', 'bedrooms', 'age']].values
y = data['price'].values

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create a Ridge Regression model
ridge_model = Ridge(alpha=1.0)  # Alpha is the regularization strength

# Fit the model
ridge_model.fit(X_train, y_train)

# Predict house prices
y_pred = ridge_model.predict(X_test)

# Calculate the mean squared error
mse = mean_squared_error(y_test, y_pred)
print(f' Mean Squared Error: {mse:.2f}')
```

2.3 The Downside to Ridge Regression

While Ridge Regression is beneficial for addressing multicollinearity in linear regression models by adding a penalty to the loss function, it introduces several issues:

1. **Bias Introduction:** By shrinking the regression coefficients, Ridge Regression inherently introduces bias into the model estimates, potentially undermining predictive accuracy when large coefficients are actually needed.
2. **Parameter Sensitivity:** The model's performance is highly sensitive to the selection of the regularization parameter, λ. This parameter must be carefully chosen, typically via cross-validation, which can be computationally expensive.
3. **Computation Complexity:** Especially in the context of large datasets, the computational demand of Ridge Regression, which involves matrix inversion, becomes a significant challenge.

To illustrate alternative approaches, below are Python examples implementing Lasso Regression and XGBoost:

```
from sklearn.linear_model import Lasso
import xgboost as xgb
import numpy as np
```

```
# Generating synthetic data
X = np.random.rand(100,3)  # 100 samples, 3 features
y = X @ np.array([1.5, -2.0, 1.0]) + np.random.rand(100)  # True relationship

# Applying Lasso Regression
lasso_model = Lasso(alpha=0.1)  # alpha corresponds to lambda in Lasso
lasso_model.fit(X, y)
print("Lasso Coefficients:", lasso_model.coef_)

# Applying XGBoost
xg_model = xgb.XGBRegressor(objective ='reg:squarederror', colsample_bytree = 0.3,

learning_rate = 0.1,
                max_depth = 5, alpha = 10, n_estimators = 10)
xg_model.fit(X, y)
print("XGBoost Feature Importance:", xg_model.feature_importances_)
```

2.4 Real life Application for James-Stein Estimator

```
import numpy as np

def james_stein_estimator(sample_means, sigma_squared):
    """
    Apply the James-Stein estimator to improve the estimation of means.

    Parameters:
    sample_means (numpy array): Vector of sample means (e.g., batting averages).
    sigma_squared (float): The common variance of the observations.

    Returns:
    numpy array: The James-Stein adjusted means.
    """
    p = len(sample_means)
    if p <= 2:
        return sample_means  # Not effective when p <= 2
    overall_mean = np.mean(sample_means)
    norm_squared = np.sum((sample_means - overall_mean) ** 2)
    shrinkage_factor = (p - 3) * sigma_squared / norm_squared
    return overall_mean + (1 - shrinkage_factor) * (sample_means - overall_mean)

# Example data: Batting averages of 5 players early in the season
sample_means = np.array([0.200, 0.250, 0.300, 0.350, 0.400])
```

```
sigma_squared = 0.002  # Estimated variance of batting averages

# Apply the James-Stein Estimator
adjusted_means = james_stein_estimator(sample_means, sigma_squared)
print("Adjusted Means:", adjusted_means)
```

References

[1] ChatGPT, https://www.openai.com/

[2] Wikipedia contributors. James-Stein estimator. Wikipedia, The Free Encyclopedia.

[3] Penn State Eberly College of Science. (n.d.). *The James-Stein Estimator*. Retrieved from https://online.stat.psu.edu/stat857/node/155/

[4] X. Author, Y. Author. (2023). *Title of the Article*. Journal Name, volume(issue), pages. https://link.springer.com/article/10.1007/s42081-023-00209-y