

贝叶斯



2 Contributors

基本介绍

分类与回归（Classification and Regression）

我们知道，机器学习可以分为监督学习和无监督学习，其中我们接触到的绝大部分都是监督学习。类比于高等数学中求离散数值的和与对连续区间上的函数求积分，监督学习可以根据实际的数据标签（label）是否连续分为：连续情况下的回归（regression）和离散情况下的分类（classification），本质都是对有数据标签的数据进行处理。本章介绍的朴素贝叶斯模型，即是一种一定假设条件（条件概率独立性假设）下利用贝叶斯公式进行分类的模型。

贝叶斯定理

贝叶斯定理（Bayes' Theorem）是概率论中的一个重要定理，它描述了在已知某些条件下，如何更新事件的概率。贝叶斯定理在统计推断、机器学习、金融分析等领域都有广泛的应用。

设 A 和 B 是两个事件，且 $P(B) > 0$ ，则贝叶斯定理的公式为：

$$P(A_i | B) = \frac{P(B | A_i)P(A_i)}{P(B)}$$

其中：

- $P(A_i | B)$ ：在事件 B 发生的前提下，事件 A_i 发生的**后验概率**（我们希望计算的概率）。
- $P(B | A_i)$ ：在事件 A_i 发生的前提下，事件 B 发生的**似然概率**（似然指某种事件发生的可能性，和概率相似）。
- $P(A_i)$ ：事件 A_i 发生的**先验概率**，即在没有考虑事件 B 发生时的概率。
- $P(B)$ ：事件 B 发生的**全概率**，可以通过全概率公式计算：

$$P(B) = P(B | A_1) \cdot P(A_1) + P(B | A_2) \cdot P(A_2) + P(B | A_3) \cdot P(A_3)$$

其中 A_1, A_2, A_3 是事件 A 的完备事件组。

概率与似然

在统计学中，“似然性”和“概率”（或然性）有明确的区分：概率，用于在已知一些参数的情况下，预测接下来在观测上所得到的结果；似然性，则是用于在已知某些观测所得到的结果时，对有关事物之性质的参数进行估值，也就是说已观察到某事件后，对相关参数进行猜测。

与上面给出的贝叶斯定理公式相对应，我们来解释一下各个概率的含义。

- $P(A)$ ：**先验概率**，表示在没有观察到数据 B 之前，我们对事件 A 发生的信念。
- $P(B | A)$ ：**似然（Likelihood）**，表示在事件 A 发生的情况下，观察到数据 B 的概率，更多的是一种事后的概念。
- $P(B)$ ：**证据（Evidence）**，即事件 B 发生的总概率，通常用全概率公式计算。
- $P(A | B)$ ：**后验概率**，表示在观察到数据 B 之后，我们对事件 A 发生的更新后的信念。

直观理解

- 先验概率（Prior）**：
 - 代表我们在获取新信息之前对某个事件的原始信念。例如，一个医生知道某种疾病的患病率是 1%，那么 1% 就是先验概率。
- 后验概率（Posterior）**：
 - 代表在得到新数据（例如检测结果）之后，我们对该事件的修正信念。例如，如果一个病人检测结果呈阳性，我们根据检测的准确性（似然）调整对该病人患病的概率，从而得到后验概率。

贝叶斯定理的直观理解

贝叶斯定理的核心思想是**利用已有的信息（先验概率）结合新证据（似然概率）来更新概率**：

- 先验概率 $P(A)$ 代表在没有额外信息（ML算法中对应数据类别）时，我们对事件 A 发生的信念。
- 似然概率 $P(B | A)$ 表示如果 A 发生，事件 B 发生的可能性（这种概率通过样本数据可以估计出来）。
- 通过贝叶斯定理，我们可以计算**在观察到 B （某种数据类别）发生的情况下，事件 A 发生的概率 $P(B | A)$** 。

朴素贝叶斯算法和分类

朴素贝叶斯算法是一种基于贝叶斯定理的概率分类方法，其核心在于**特征条件独立假设**，即在给定类别的前提下，假设各个特征之间是相互独立的。下面详细介绍其原理和分类过程。

基础假定




设输入空间 $\chi \subseteq R^n$ 为 n 维向量的集合，输出空间为类标记集合 $\Upsilon = \{c_1, c_2, \dots, c_k\}$ 。输入为特征向量 $x \in \chi$ ，输出为类标记（class label） $y \in \Upsilon$ 。 X 是定义在输入空间 χ 上的随机向量， Y 是定义在输出空间 Υ 上的随机变量。 $P(X, Y)$ 是 X 和 Y 的联合概率分布。训练集


$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

由 $P(X, Y)$ 独立同分布产生。

朴素贝叶斯算法（分类） 原理

朴素贝叶斯算法（分类）原理

- 概括来说，朴素贝叶斯算法通过训练集学习联合概率分布 $P(X, Y)$ 。朴素贝叶斯算法是一种分类算法，其目的在于当模型接收到给定的数据输入时，可以识别其它具体类别。
- 具体来说，任意给定一个n维向量 $x = (x_1, x_2, \dots, x_n)$ 作为自变量输入时，选取某一类别 c_k ，模型首先通过训练样本得到该类别下的先验概率分布 $P(Y = c_k)$ ，然后习得对应条件下的概率分布（以某种类别 c_k 为条件） $P(x | c_k)$ （ 这一步将用到条件独立性的特殊假定，下文有专门的介绍，这也是为什么该算法叫做朴素贝叶斯的原因），将二者相乘便得到以类别 c_k 为条件下的联合概率分布 $P(X = x, Y = c_k)$ ，将联合概率分布 $P(X = x, Y = c_k)$ 除以 $P(x)$ （ 通常由完备事件组条件概率公式求得），便得到在给出 $X = x$ 数据条件下，模型判定 $Y = c_k$ 的后验条件概率 $P(c_k | x)$ 。接着选取另一类别 c_l 重复计算对应类别 c_l 下的先验概率，条件概率以及 $P(x)$ （ 和 c_k 条件下数值相同，上文已计算过），计算出对应 c_l 条件下的后验概率。换取另一类别再次计算对应类别下的后验概率直到所有类别都已计算结束，选取所有后验概率结果当中的最大值，最大值对应的类别即为模型给出的分类结果。

-  数学上可以证明，后验概率最大化时即满足了损失函数的期望风险最小化准则，回到一般的模型评估步骤，我们知道，最大后验概率对应的类别即是我们需要的结果！

将 原理回归到数学公式来理解：

对于一个类别 c_k 和特征向量 $x = (x_1, x_2, \dots, x_n)$ ，利用贝叶斯定理可得：

$$P(c_k | x) = \frac{P(x | c_k) P(c_k)}{P(x)}$$

- $P(c_k | x)$ ：在观察到特征 x 后，样本属于类别 c_k 的后验概率。
- $P(x | c_k)$ ：在类别 c_k 下，观察到特征 x 的似然概率。
- $P(c_k)$ ：类别 c_k 的先验概率，即在未观察特征前对 c_k 的信念。
- $P(x)$ ：样本的总体概率，原理部分提到，对于所有类别来说均相等，因此在分类时可忽略分母而不影响后验概率大小的比较。

朴素贝叶斯算法核心

利用贝叶斯公式进行计算后验概率时，我们需要记得模型输入是一个 n 维向量 $x = (x_1, x_2, \dots, x_n)$ ，第 i 个分向量 x_i 假设可取 S_j 个值，且假设共有 K 个类别标签，则若估计每有的条件概率，共有 $K \cdot \prod_{j=1}^n S_j$ 个，实际分类中，这个乘积很大对应要求得条件概率的个数很多，不方便分别求出。故从实用性和简便性考虑，朴素贝叶斯算法假设在条件概率的下， n 向量 x 各维度的取值相互独立。

即：

$$P(X = x | Y = c_k) = P(X^{(1)} = x^{(1)}, \dots, X^{(n)} = x^{(n)} | Y = c_k) = \prod_{j=1}^n P(X^{(j)} = x^{(j)} | Y = c_k)$$

这样便大大简化了程序的计算量。但这一核心假定也可能导致学得模型产生误差。

朴素贝叶斯的参数估计

④ 叶斯分类中，我们需要计算先验概率 $P(C)$ 和条件概率 $P(x_j | C)$ 。这些概率通常使用**极大似然估计**（Maximum Likelihood Estimation, MLE）进行估计。

极大似然估计的基本思想

上文中提到 $P(c_k)$ 和 $P(x | c_k)$ 如何估计出来呢？我们运用极大似然原理，对训练集数据进行极大似然估计，并结合上文提到的条件概率下独立性假设便可算出对应概率。极大似然估计的核心思想是：在已知数据的情况下，找到使观测数据出现的概率最大的参数值。

对于朴素贝叶斯分类，假设我们有一个训练集：

$$D = (X_1, C_1), (X_2, C_2), \dots, (X_m, C_m)$$

其中：

- $X_i = (x_{i1}, x_{i2}, \dots, x_{in})$ 表示第 i 个样本的特征向量。
- C_i 表示第 i 个样本的类别标签。

先验概率的极大似然估计

先验概率 ($P(C)$) 代表类别 (C) 在训练集中出现的频率，即：

$$P(C_k) = \frac{\text{类别 } C_k \text{ 的样本数}}{\text{总样本数}}$$

即： $P(C_k) = \frac{N_k}{m}$

其中： N_k 是类别 C_k 的样本数； m 是总样本数。

条件概率的极大似然估计

对于特征 x_j ，条件概率 $P(x_j | C_k)$ 的计算方式取决于特征的类型：

(1) 离散变量

如果 (x_j) 是离散的（如文本分类中的单词是否出现），则可以用频率计算：

$$P(x_j = a | C_k) = \frac{\text{类别 } C_k \text{ 中 } x_j = a \text{ 的样本数}}{\text{类别 } C_k \text{ 的总样本数}}$$

即： $P(x_j = a | C_k) = \frac{N_{j,a,k}}{N_k}$ ， 其中 $N_{j,a,k}$ 表示在类别 C_k 中，特征 x_j 取值为 a 的样本数。

(2) 连续变量

如果 x_j 是连续变量（如气温），通常假设其在每个类别 C_k 下服从**高斯分布（正态分布）**：

$$P(x_j | C_k) = \frac{1}{\sqrt{2\pi\sigma_{jk}^2}} \exp\left(-\frac{(x_j - \mu_{jk})^2}{2\sigma_{jk}^2}\right)$$

其中均值 μ_{jk} 和方差 σ_{jk}^2 由训练数据计算以下公式而来：

$$\mu_{jk} = \frac{1}{N_k} \sum_{i:C_i=C_k} x_{ij}$$
$$\sigma_{jk}^2 = \frac{1}{N_k} \sum_{i:C_i=C_k} (x_{ij} - \mu_{jk})^2$$

然后根据贝叶斯公式，计算在给定气温 x_j 的情况下属于每个类别的后验概率：

$$P(C_i = C_k | x_j) = \frac{P(x_j | C_i = C_k)P(C_i = C_k)}{P(x_j)}$$

比较多个后验概率 $P(C_i | x_j)$ ，选择后验概率较大的类别作为决策结果。

特殊情况：零概率问题

在离散特征的情况下，如果某个特征值 x_j 在训练集中从未在某个类别 C_k 下出现，则 MLE 会导致：

$$P(x_j | C_k) = 0$$

这会使得整个后验概率 $P(C | X)$ 变为 0，从而导致错误分类。

为了解决这个问题，通常使用贝叶斯估计，又名**拉普拉斯平滑（Laplacian Smoothing）**：

$$P(x_j = a | C_k) = \frac{N_{j,a,k} + \alpha}{N_k + \alpha V}$$

其中 $\alpha > 0$ 是平滑系数（通常取 $\alpha = 1$ ）； V 是特征 x_j 可能的取值个数。

最大后验估计

(Maximum a posteriori, 简称MAP)

在贝叶斯推断中，我们经常需要从这个后验分布中选出一个具体的数值，作为参数的点估计 (point estimator)。所谓点估计，就是用单一数值来代表整个分布，以便我们更直观地理解和使用。常见的贝叶斯点估计方法都是选取后验分布中某种“中心势”(central tendency)的统计量，比如：

① MAP估计是指选择后验分布中概率密度最大的那个点作为参数的估计值。换句话说，MAP估计就是在观察到数据后，参数最有可能取到的那个值。

关于MAP估计与最大似然估计 (MLE) 的比较：

最大似然估计 (Maximum Likelihood Estimator, MLE) 是一种经典频率学派的方法，它通过寻找使观测数据概率最大的参数来进行估计。但MLE在一些复杂问题上可能会遇到困难，比如当样本量较小时、或模型较复杂时，MLE可能会不稳定或难以计算。

相比之下，MAP估计由于引入了**先验信息**，相当于在MLE基础上增加了正则化 (regularization) 效果，这种正则化能够有效地降低过拟合风险，使得模型在新数据上的表现更稳定、更可靠。

通常具备更好的渐近性质 (**asymptotic properties**)。所谓渐近性质好，是指当数据量越来越大时，MAP估计通常会逐渐趋向于真实参数，并且表现更加稳定可靠。即使在一些MLE难以处理的问题上 (例如样本较少、模型复杂或存在多解情况)，MAP也能表现出更稳健、更可靠的性能。

MAP方法允许研究者根据具体问题灵活选择不同形式的先验分布，从而更好地适应不同场景下的需求。这种灵活性使得MAP方法可以广泛应用于各种复杂模型和实际问题中，例如机器学习中的深度神经网络、图像处理中的图像重建、生物学中的高维数据分析等领域。

因此MAP估计器特别适合以下几种场景：

1. 数据量较少或数据质量较差的场景
2. 存在明确先验知识或领域经验的场景
3. 模型复杂、高维参数空间或容易过拟合的问题

递归贝叶斯

问题设定与数学推导

步骤1: 模型设定

- 线性模型:

$$y = wx + \epsilon$$

其中 $\epsilon \sim \mathcal{N}(0, \sigma^2)$, w 为待估计的斜率参数。

- 先验分布: 假设先验为高斯分布

$$w \sim \mathcal{N}(\mu_0, \sigma_0^2)$$

假设当前已处理 $n - 1$ 个数据, 得到的先验为 $w \sim \mathcal{N}(\mu_{n-1}, \sigma_{n-1}^2)$, 现输入第 n 个数据点 (x_n, y_n) 后验分布推导如下:

步骤2: 写出似然与先验

似然函数:

$$p(y_n \mid w) = \mathcal{N}(y_n \mid wx_n, \sigma^2).$$

先验分布:

$$p(w) = \mathcal{N}(w \mid \mu_{n-1}, \sigma_{n-1}^2).$$

步骤3: 合并指数项

后验分布 $p(w \mid y_n) \propto p(y_n \mid w) p(w)$ 的指数部分为:

$$\exp\left(-\frac{(y_n - wx_n)^2}{2\sigma^2} - \frac{(w - \mu_0)^2}{2\sigma_0^2}\right)$$

展开平方项:

$$\text{指数} = -\frac{1}{2} \left[\frac{(y_n - wx_n)^2}{\sigma^2} + \frac{(w - \mu_0)^2}{\sigma_0^2} \right] = -\frac{1}{2} \left[\frac{w^2x_n^2 - 2y_nwx_n + y_n^2}{\sigma^2} + \frac{w^2 - 2\mu_0w + \mu_0^2}{\sigma_0^2} \right] + \text{常数项}.$$

步骤4: 提取关于 w 的二次型 保留含 w^2 和 w 的项:

$$\text{指数} = -\frac{1}{2} \left[\underbrace{\left(\frac{x_n^2}{\sigma^2} + \frac{1}{\sigma_0^2}\right)}_{(1/\sigma_n^2)} w^2 - \underbrace{\left(\frac{2y_nx_n}{\sigma^2} + \frac{2\mu_0}{\sigma_0^2}\right)}_{(2\mu_n/\sigma_n^2)w} w \right] + \text{常数项}.$$

将此式与标准高斯形式 对比,

$$\exp\left(-\frac{(w - \mu_n)^2}{2\sigma_n^2}\right)$$

可以解得:

- 精度 (方差的倒数):

$$\frac{1}{\sigma_n^2} = \frac{x_n^2}{\sigma^2} + \frac{1}{\sigma_0^2}$$

- 均值:

$$\mu_n = \sigma_n^2 \left(\frac{y_nx_n}{\sigma^2} + \frac{\mu_0}{\sigma_0^2} \right)$$

最终参数更新公式:

$$\sigma_n^{-2} = \sigma_{n-1}^{-2} + \frac{x_n^2}{\sigma^2}, \quad \mu_n = \sigma_n^2 \left(\frac{\mu_{n-1}}{\sigma_{n-1}^2} + \frac{x_ny_n}{\sigma^2} \right).$$

Python 代码实现

```
##### 生成数据与初步可视化
import numpy as np
import matplotlib.pyplot as plt

# True parameters
w_true = 10
sigma_noise = 30
num_samples = 80

# Generate data
np.random.seed(42)
x = np.random.uniform(-10, 10, num_samples)
y = w_true * x + np.random.normal(0, sigma_noise, num_samples)

# -----
# Plot 1: Show raw data and true model
# -----
plt.figure(figsize=(8, 5))
plt.scatter(x, y, alpha=0.6, label='Noisy measurements')
plt.plot([-10, 10], [-10*w_true, 10*w_true],
         color='r', linestyle='--', label='True model: $y=3x$')
plt.title('Generated Training Data')
plt.xlabel('Feature $x$'), plt.ylabel('Target $y$')
plt.grid(True)
plt.legend()
plt.show()

##### 贝叶斯参数估计
# Initialize prior
mu_prior = -10.0
var_prior = 1000.0
sigma_sq_noise = sigma_noise**2 # Known noise variance

# Store parameter evolution
mus = [mu_prior]
variances = [var_prior]

# Recursive Bayesian update
for xi, yi in zip(x, y):
    # Posterior variance update (precision addition)
    new_precision = 1/variances[-1] + xi**2 / sigma_sq_noise
    new_variance = 1 / new_precision

    # Posterior mean update (weighted sum)
    new_mean = new_variance * (mus[-1]/variances[-1] + xi*yi/sigma_sq_noise)

    # Record parameters
    mus.append(new_mean)
    variances.append(new_variance)

##### 最小二乘法估计
# Closed-form OLS solution for y = wx
X = x.reshape(-1, 1) # Ensure column vector
w_ols = np.linalg.lstsq(X, y, rcond=None)[0][0]

# Scikit-learn: LinearRegression(fit_intercept=False)

# Alternatively: w_ols = (x @ y) / (x @ x)

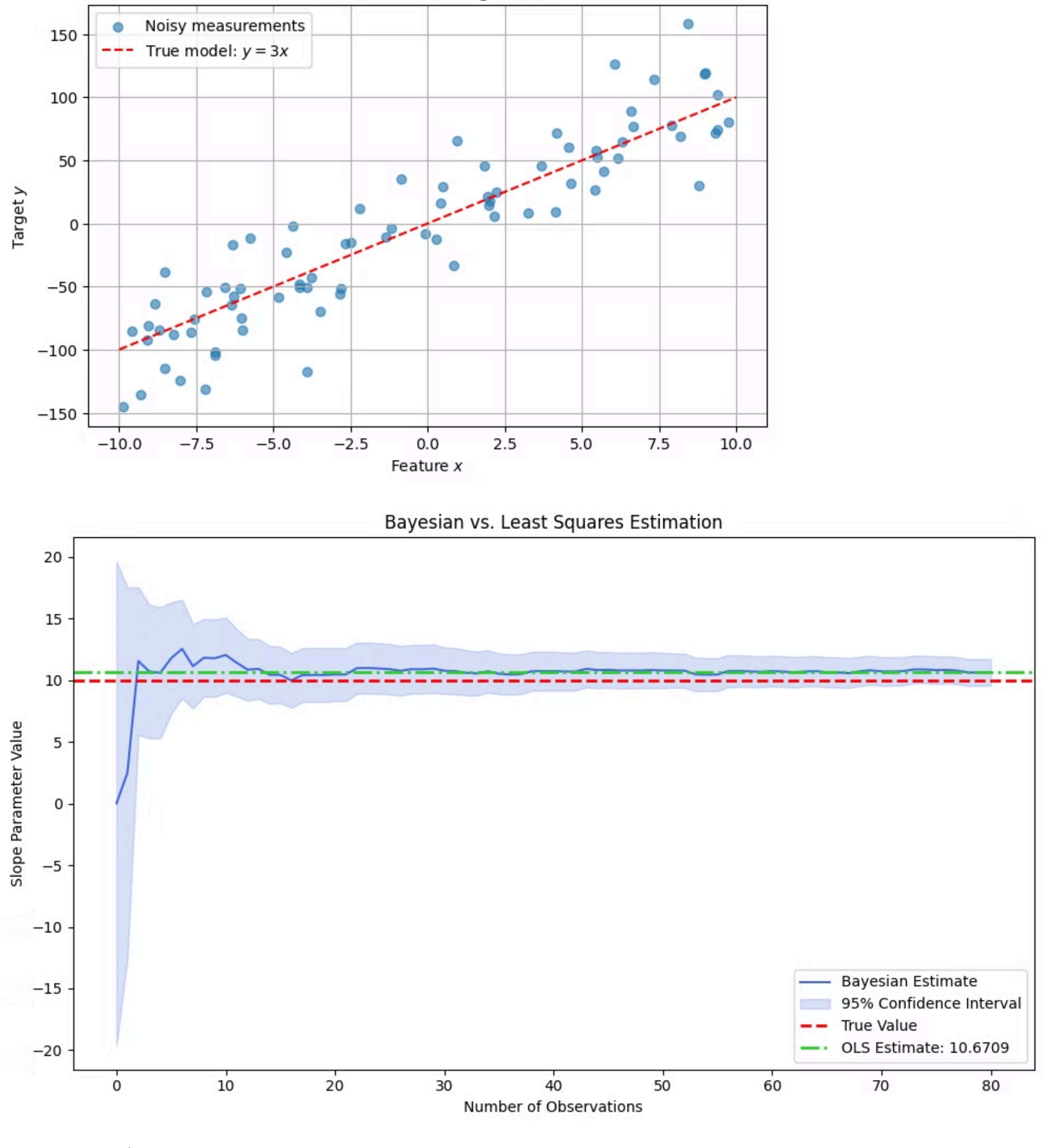
##### 可视化对比
plt.figure(figsize=(10, 6))

# Bayesian trajectory
plt.plot(mus, color='royalblue', label='Bayesian Estimate')
plt.fill_between(
    range(len(mus)),
    np.array(mus) - 1.96*np.sqrt(np.array(variances)),
    np.array(mus) + 1.96*np.sqrt(np.array(variances)),
    color='royalblue',
    alpha=0.2,
    label='95% Confidence Interval'
)

# Ground truth and OLS
plt.axhline(w_true, color='r', linestyle='--',
            linewidth=2, label='True Value')
plt.axhline(w_ols, color='limegreen', linestyle='-',
            linewidth=2, label=f'OLS Estimate: {w_ols:.4f}')

plt.xlabel('Number of Observations'),
plt.ylabel('Slope Parameter Value')
plt.title('Bayesian vs. Least Squares Estimation')
plt.legend(loc='lower right')
plt.tight_layout()
plt.show()

##### 数值对比
print(f"""
{' Method ':'^25'} | {' Value ':'^10'}
{'='*25} | {'='*10'}
{'Bayesian (final)':<25} | {'mus[-1]:>10.4f}
{'Ordinary Least Squares':<25} | {'w_ols:>10.4f}
{'Ground Truth':<25} | {'w_true:>10.4f}
""")
```



Bayesian (final) | 10.6389

Ordinary Least Squares | 10.6709

Ground Truth | 10.0000

贝叶斯后验与最小二乘法

贝叶斯估计中的后验方差代表着参数估计的不确定性，而最小二乘估计的P值则是频率主义框架下用来检验“参数为零”假设的统计量。理论上，当样本量足够大时，贝叶斯的后验方差和频率主义的置信区间有对应关系，特别是在高斯模型中，两者的结果会趋向一致。

④ 注意不要混淆两者的概率解释。贝叶斯方差直接反映参数的不确定性，而P值涉及假设检验中的错误率控制。但在数值比较层面，可能找到对应的数学关系。

理论关系推导

(1) 普通最小二乘的方差估计

对于线性模型 $y = wx + \epsilon$ ， $\epsilon \sim \mathcal{N}(\mu_0, \sigma_0^2)$ ，OLS估计量为

$$\begin{aligned} S(w) &= \sum_{i=1}^N (y_i - wx_i)^2 \\ \Rightarrow \frac{\partial S(w)}{\partial w} &= -2 \sum_{i=1}^N x_i (y_i - wx_i) = 0 \\ \Rightarrow \sum_{i=1}^N x_i y_i &= w \sum_{i=1}^N x_i^2 \\ \Rightarrow \hat{w}_{\text{OLS}} &= \frac{\sum x_i y_i}{\sum x_i^2} \end{aligned}$$

其估计方差（即标准误平方）为：

$$\begin{aligned} \hat{w}_{\text{OLS}} &= \frac{\sum x_i (wx_i + \epsilon_i)}{\sum x_i^2} = w + \frac{\sum x_i \epsilon_i}{\sum x_i^2} \\ \text{Var}(\hat{w}_{\text{OLS}}) &= \text{Var}\left(\frac{\sum x_i \epsilon_i}{\sum x_i^2}\right) \\ &= \frac{\sigma^2 \sum x_i^2}{(\sum x_i^2)^2} \quad \triangleright \text{(误差项独立同分布)} \\ &= \frac{\sigma^2}{\sum x_i^2} \\ \text{Var}(\hat{w}_{\text{OLS}}) &= \frac{\sigma^2}{\sum x_i^2}. \end{aligned}$$

假设 σ^2 已知，则假设检验的P值通过以下统计量计算：

$$t = \frac{\hat{w}_{\text{OLS}} - w_0}{\sqrt{\text{Var}(\hat{w}_{\text{OLS}})}}$$

其中 w_0 为原假设下的值（通常取0）。

(2) 贝叶斯后验方差

后验方差与OLS方差关系：

当 $\sigma_0^2 \rightarrow \infty$ （无信息先验）时：

$$\lim_{\sigma_0^2 \rightarrow \infty} \frac{1}{\frac{1}{\sigma_0^2} + \sum \frac{x_i^2}{\sigma^2}} = \frac{\sigma^2}{\sum x_i^2}$$

贝叶斯后验方差公式退化为：

$$\sigma_{\text{post}}^2 = \frac{\sigma^2}{\sum x_i^2}.$$

这与OLS的方差估计完全一致。

P值计算公式：

$$p = 2\Phi\left(-\left|\frac{\hat{w}}{\sqrt{\sigma^2 / \sum x_i^2}}\right|\right)$$

（ Φ 为标准正态CDF）

核心结论

- 当使用**无信息先验**时，贝叶斯后验方差 ↔ OLS方差**严格等价**
- 两类方法的**置信/可信区间**在数值上重合

工程应用启发

- 大数据场景下两种方法可交叉验证
- 贝叶斯方法允许通过**先验融合领域知识**
- OLS可作为贝叶斯结果的快速**估算基准**

递归贝叶斯因果分析

问题设定

我们希望通过递归贝叶斯方法估计线性回归模型的参数，从而进行因果分析。假设模型如下：

$$Y_i = \beta_0 + \beta_1 X_i + \beta_2 Z_{1i} + \beta_3 Z_{2i} + \varepsilon_i$$

其中：

- Y_i 是被解释变量；
- X_i 是解释变量；
- Z_{1i}, Z_{2i} 是控制变量；
- $\varepsilon_i \sim N(0, \sigma^2)$ 是误差项。

我们希望估计参数向量

$$\boldsymbol{\beta} = [\beta_0, \beta_1, \beta_2, \beta_3]^\top$$

贝叶斯框架

先验分布

为参数 $\boldsymbol{\beta}$ 设置先验分布：

$$\boldsymbol{\beta} | \sigma^2 \sim N(\mathbf{0}, V_0 \sigma^2), \quad V_0 = 1000I$$

即：

- 初始均值为 0，表示对参数没有任何偏好；
- 初始协方差矩阵为 $1000I$ ，表示初始不确定性较大。

后验分布更新

根据贝叶斯公式，后验分布通过结合先验分布和观测数据得到：

$$p(\boldsymbol{\beta} | D_t) = \frac{p(Y_t | X_t, Z_{1t}, Z_{2t}, \boldsymbol{\beta}) p(\boldsymbol{\beta} | D_{t-1})}{p(Y_t | D_{t-1})}$$

其中：

$$D_t = (Y_j, X_j, Z_{1j}, Z_{2j}), j = 1, ..., t$$

是前 t 个数据点。

递归贝叶斯更新公式

输入矩阵

定义输入矩阵：

$$\mathbf{X}'_t = [1, X_t, Z_{1t}, Z_{2t}]^\top$$

模型预测为：

$$Y_t | X_t, Z_{1t}, Z_{2t}, \boldsymbol{\beta} \sim N(\mathbf{X}_t'^\top \boldsymbol{\beta}, \sigma^2)$$

预测方差推导

预测方差是指在当前后验分布下，预测数据点的误差方差。它由以下两部分组成：

- 参数的不确定性（协方差矩阵）导致的预测误差: $\boldsymbol{\beta} \sim N(\boldsymbol{\mu}_{t-1}, V_{t-1})$
- 模型本身的噪声（误差项方差）。

$$\text{Var} [\mathbf{x}_t^\top \boldsymbol{\beta}] = \mathbf{x}_t^\top V_{t-1} \mathbf{x}_t$$

因为噪声 ϵ_t 相对于 $\boldsymbol{\beta}$ 独立, y_t 的预测分布为:

$$y_t \sim N(\mathbf{X}_t^\top \boldsymbol{\mu}t - 1, \underbrace{\mathbf{X}_t^\top V_{t-1} \mathbf{X}_t + \sigma^2}_{R_t})$$

因此，预测方差可以写为：

$$R_t = \mathbf{X}_t^\top V_{t-1} \mathbf{X}_t + \sigma^2$$

- 第一项 $\mathbf{X}_t^\top V_{t-1} \mathbf{X}_t$ 表示模型参数 $\boldsymbol{\beta}$ 的不确定性对预测的贡献；
- 第二项 σ^2 表示模型固有噪声。

卡尔曼增益推导

卡尔曼增益是递归贝叶斯更新中的关键，它衡量了新数据对参数估计的影响程度。卡尔曼增益定义为：

$$K_t = V_{t-1} \mathbf{X}_t R_t^{-1}$$

卡尔曼增益的作用是调整后验均值，使其向新数据提供的信息靠拢。具体来说：

- 如果新数据提供的信息较多（即预测残差较大），卡尔曼增益会加大，后验均值向新数据靠拢；
- 如果新数据提供的信息较少（即预测残差较小），卡尔曼增益会减小，后验均值变化较小。

后验均值更新公式

利用卡尔曼增益更新后验均值：

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + K_t(Y_t - \mathbf{X}_t'^\top \boldsymbol{\mu}_{t-1})$$

其中：

- 第一项是先验均值；
- 第二项是根据新数据调整后的修正量。

后验协方差更新公式

利用卡尔曼增益更新后验协方差矩阵：

$$V_t = (I - K_t \mathbf{X}_t'^\top) V_{t-1}$$

该公式表示：随着新数据的引入，协方差矩阵逐渐减小，即参数的不确定性逐步减少。

Python实现

以下是递归贝叶斯方法的完整实现：

```
import numpy as np
import matplotlib.pyplot as plt

# --- Step 1: Simulate Data ---
# Set the random seed for reproducibility
np.random.seed(42)

# Number of observations
n = 50

# Generate independent variables from a standard normal distribution.
X = np.random.normal(size=n)
Z1 = np.random.normal(size=n)
Z2 = np.random.normal(size=n)

# Define the true coefficients: [intercept, coefficient for X, coefficient for Z1, coefficient for Z2]
true_beta = np.array([3, 2, -1, 4]) # True parameter values

# Generate the dependent variable Y using the linear model:
# Y = beta0 + beta1*X + beta2*Z1 + beta3*Z2 + noise, where noise ~ N(0, sigma_sq)
noise_std = 1.0 # Standard deviation of the noise
Y = true_beta[0] + true_beta[1]*X + true_beta[2]*Z1 + true_beta[3]*Z2 + np.random.normal(scale=noise_std, size=n)

# --- Step 2: Initialize the Prior Distribution ---
dim = 4 # The number of parameters (intercept + three coefficients)
# Set an initial prior mean vector (all zeros, indicating an initial guess)
mu_prior = np.zeros(dim)

# Set an initial prior covariance matrix. Using a large variance (1000*I) reflects high initial uncertainty.
V_prior = np.eye(dim) * 1000

# The noise variance is assumed known
sigma_sq = noise_std ** 2

# Initialize lists to store the posterior means and the diagonal elements (variances) of the posterior covariance at each step.
posterior_means = []
posterior_vars = []

# --- Step 3: Recursive Bayesian Update via Kalman Filter Equations ---
# For each observation, update the posterior of the parameters.
for t in range(n):
    # Form the design vector Xt corresponding to the current observation.
    # Use 1 for intercept, and the values from X, Z1, and Z2.
    Xt = np.array([1, X[t], Z1[t], Z2[t]]); None] # Reshape to a column vector of shape (dim, 1)
    Yt = Y[t] # The observed response for the t-th data point

    # -- Prediction Step --
    # Compute the predictive variance Rt, which is the variance of the new observation.
    # Rt = Xt^T * V_prior * Xt + sigma_sq
    Rt = Xt.T @ V_prior @ Xt + sigma_sq

    # -- Update Step: Compute the Kalman Gain --
    # The Kalman gain determines the weight given to the new information (observation).
    # Kt = V_prior * Xt / Rt (this is a vector of shape (dim, 1))
    Kt = V_prior @ Xt / Rt

    # -- Compute the Residual --
    # (mu_t - 1, X[t], Z1[t], Z2[t]]); None] # Reshape to a column vector of shape (dim, 1)
    # residual = Yt - Xt^T * mu_prior
    residual = Yt - Xt.T @ mu_prior

    # -- Update the Posterior Mean --
    # Incorporate the new measurement into the prior mean. The update is proportional to the Kalman gain and the residual.
    # mu_post = mu_prior + Kt * residual
    mu_post = mu_prior + Kt.flatten() * residual # Use flatten() to convert Kt to a 1D array

    # -- Update the Posterior Covariance --
    # The updated covariance is given by the reduction in uncertainty after incorporating the new observation.
    # V_post = (I - Kt * Xt^T) * V_prior
    V_post = (np.eye(dim) - Kt @ Xt.T) @ V_prior

    # Store the updated posterior mean and the diagonal of the posterior covariance (variances)
    posterior_means.append(mu_post)
    posterior_vars.append(np.diag(V_post))

    # Update the prior with the current posterior for the next iteration.
    mu_prior, V_prior = mu_post, V_post

# --- Step 4: Output the Final Posterior Estimates ---
print("Final parameter estimates (posterior means):", posterior_means[-1])
print("Final parameter estimates (posterior variances on diagonal):", posterior_vars[-1])

# --- Step 5: Visualizing the Evolution of the Posterior Estimates ---
# For creating the plot, we convert the lists to numpy arrays
posterior_means_array = np.array(posterior_means) # Shape: (n, dim)
posterior_vars_array = np.array(posterior_vars) # Shape: (n, dim)

# Define labels for each parameter for clarity in the plots.
param_names = ['Intercept', 'Coefficient for X', 'Coefficient for Z1', 'Coefficient for Z2']

# Create a figure with two subplots.
# The top subplot will show the evolution of the posterior means.
# The bottom subplot will show the evolution of the posterior variances.
fig, axs = plt.subplots(2, 1, figsize=(12, 10))

# Plot the evolution of each parameter's posterior mean across iterations.
for i in range(dim):
    axs[0].plot(range(1, n+1), posterior_means_array[:, i], label=param_names[i])
axs[0].set_title('Evolution of Posterior Means')
axs[0].set_xlabel('Iteration')
axs[0].set_ylabel('Posterior Mean Value')
axs[0].legend()
axs[0].grid(True)

# Plot the evolution of each parameter's posterior variance (diagonal elements) across iterations.
for i in range(dim):
    axs[1].plot(range(1, n+1), posterior_vars_array[:, i], label=param_names[i])
axs[1].set_title('Evolution of Posterior Variances')
axs[1].set_xlabel('Iteration')
axs[1].set_ylabel('Posterior Variance Value')
axs[1].legend()
axs[1].grid(True)

# Adjust layout for better spacing and display the plots.
plt.tight_layout()
plt.show()
```

