Taller

Implementación del Lasso y Selección de λ vía Validación Cruzada

Objetivo

Implementar el estimador Lasso utilizando el algoritmo de descenso por coordenadas para ajustar modelos de regresión penalizados, y seleccionar el parámetro de regularización λ óptimo mediante validación cruzada K-fold.

Parte 1: Preparación de datos

- Simular un conjunto de datos de regresión con matriz de diseño $X \in \mathbb{R}^{n\times p}$ y respuesta $y \in \mathbb{R}^n$, donde $n = 100$ y $p = 20$. Incluir correlación entre las variables predictoras.
- Generar un vector β^{true} esparso (algunos coeficientes exactamente cero).
- Generar $y = X\beta^{\text{true}} + \varepsilon$, con $\varepsilon \sim \mathcal{N}(0, \sigma^2 I)$.

Parte 2: Implementación del Lasso

Implementar en R o Python el Algoritmo 1 de descenso por coordenadas para resolver el problema:

$$\hat{\beta}^{\lambda} = \arg\min_{\beta} \left\{ \frac{1}{2n}\|y - X\beta\|_2^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}$$

Algoritmo 1. Descenso por coordenadas para el estimador Lasso

Entrada: Matriz de diseño $X \in \mathbb{R}^{n\times p}$, vector de respuesta $y \in \mathbb{R}^n$, parámetro de regularización $\lambda > 0$

Preprocesamiento:

Centrar $y \leftarrow y - \bar{y}$

Para cada columna j: centrar y escalar $X_j \leftarrow \frac{X_j - \bar{X}_j}{\|X_j\|_2}$

Inicializar: $\beta^{(0)} = 0$, tolerancia ϵ, $t = 0$

repeat

  for cada coordenada $j = 1, \ldots, p$ **do**

    Calcular el residuo parcial:

$$r_i^{(j)} = y_i - \sum_{k\neq j} x_{ik}\beta_k^{(t)}$$

    Calcular:

$$\langle x, r_i^{(j)} \rangle = \frac{1}{n}\sum_{i=1}^{n} x_{ij} r_i^{(j)}$$

    Actualizar coeficiente:

$$\beta_j^{(t+1)} = S_\lambda(\langle x, r_i^{(j)} \rangle), \quad \text{donde } S(z,\lambda) = \text{sign}(z)\cdot \text{máx}(|z| - \lambda, 0)$$

  end for

  $t \leftarrow t + 1$

until convergencia: $\|\beta^{(t)} - \beta^{(t-1)}\|_2 < \epsilon$

Recuperar la media original de y: $\bar{y}$, y de cada columna X_j: $\bar{x}_j$

Calcular el intercepto:

$$\hat{\beta}_0 = \bar{y} - \sum_{j=1}^{p} \bar{x}_j \cdot \beta_j^{(t)}$$

Salida: $(\hat{\beta}_0, \beta^{(t)})$

Parte 3: Validación cruzada

- Implementar en R o Python el Algoritmo 2 de validación cruzada K-fold
- Usar $K = 10$ y una grilla de al menos 100 valores de λ para seleccionar el valor óptimo de λ.
- Seleccionar el λ^* que minimice el error cuadrático medio de validación.

Algoritmo 2. Validación cruzada K-fold para seleccionar λ óptimo en Lasso

Entrada: Datos (X, y), número de folds K, número de puntos L en la grilla de λ

Calcular:

$$\lambda_{\text{máx}} = \max_j \left| \frac{1}{n} X_j^\top y \right|$$

Construir grilla logarítmica:

$$\lambda_1, \ldots, \lambda_L \in [0{,}0001 \cdot \lambda_{\text{máx}}, \lambda_{\text{máx}}]$$

equispaciada en escala log:

$$\lambda_\ell = \lambda_{\text{máx}} \cdot (0{,}0001)^{\frac{\ell-1}{L-1}} \quad \text{para } \ell = 1, \ldots, L$$

Dividir los datos en K *folds* $D_1, \ldots, D_K$

for cada $\lambda_\ell \in \{\lambda_1, \ldots, \lambda_L\}$ **do**

  Inicializar error acumulado $E_\ell = 0$

  for cada fold $k = 1, \ldots, K$ **do**

    Definir entrenamiento $D_{\text{train}} = \bigcup_{j \neq k} D_j$, validación $D_{\text{val}} = D_k$

    Ejecutar **Algoritmo 1** con $\lambda = \lambda_\ell$ sobre $D_{\text{train}} \to$ obtener $\hat{\beta}^{(\ell,k)}$

    Calcular error en validación:

$$\text{ECM}^{(\ell,k)} = \frac{1}{|D_k|} \sum_{(x_i, y_i) \in D_k} (y_i - x_i^\top \hat{\beta}^{(\ell,k)})^2$$

    Acumular $E_\ell \leftarrow E_\ell + \text{ECM}^{(\ell,k)}$

  end for

  Promediar: $E_\ell \leftarrow \frac{1}{K} E_\ell$

end for

Seleccionar $\lambda^* = \arg\min_{\lambda_\ell} E_\ell$

Salida: λ^*

Parte 4: Resultados y visualización

- Graficar el error de validación promedio contra $\log(\lambda)$.
- Mostrar la trayectoria de los coeficientes β_j conforme varía λ (trayectoria regularizada).
- Comparar el estimador final con β^{true} y discutir:
  - ¿Se identificaron correctamente las variables relevantes?
  - ¿Qué ocurre si no se estandariza X? ¿Y si no se centra y?

Bonus (opcional) Comparar su implementación con `glmnet` (en R) o `LassoCV` (en Python).

Entregables

- Código bien comentado (en R o Python).
- Breve informe explicando cada paso, los resultados obtenidos y reflexiones finales.