Métodos Basados en Árboles

Random Forests y Boosting
2025

Hoja de Ruta

Hasta ahora:

- Clase 4: Regresión penalizada (Ridge/Lasso)
- Clase 5: Más allá de linealidad (Polynomials, Splines)

Hoy: Métodos basados en árboles

0. **Interludio: Bootstrap** (fundamentos + aplicación econométrica)
1. Árboles de decisión (CART)
2. Bootstrap aggregating (Bagging)
3. Random Forests
4. Boosting (incluyendo XGBoost)
5. Comparación práctica y guías de uso

Aplicaciones: Predicción de salarios (regresión) y crédito (clasificación)

Interludio: El Bootstrap

¿Qué es el Bootstrap?

Problema fundamental en estadística:

- Tenemos una muestra de datos de tamaño n
- Queremos estimar una cantidad (media, varianza, coeficiente de regresión, etc.)
- **¿Qué tan precisa es nuestra estimación?** (necesitamos error estándar)

Soluciones tradicionales:

- Fórmulas analíticas (requieren supuestos distribucionales)
- Teoría asintótica (puede no funcionar con n pequeño)

Bootstrap: Una alternativa computacional sin fórmulas

La Idea del Bootstrap

Intuición: Si pudiéramos obtener muchas muestras de la población, podríamos ver cómo varían nuestras estimaciones.

Problema: Solo tenemos **una** muestra.

Solución de Bootstrap: Tratar nuestra muestra como si fuera la población y **remuestrear de ella**.

Principio del Bootstrap (Efron, 1979)

"La relación entre la muestra y la población es análoga a la relación entre una muestra bootstrap y la muestra original."

En otras palabras: Usamos nuestra muestra para aproximar la población, y generamos nuevas muestras de ella.

Algoritmo de Bootstrap

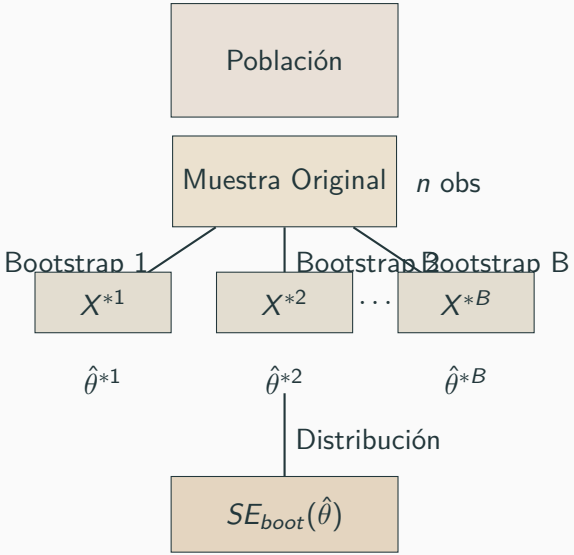
Para estimar el error estándar de un estadístico $\hat{\theta}$:

1. **Muestra original:** $X = \{x_1, x_2, \ldots, x_n\}$
2. **Para $b = 1, \ldots, B$ (típicamente $B = 100$ o más):**
   - Generar muestra bootstrap X^{*b} tomando n observaciones **con reemplazo** de X
   - Calcular el estadístico en esta muestra: $\hat{\theta}^{*b} = f(X^{*b})$
3. **Estimar error estándar:**

$$SE_{boot}(\hat{\theta}) = \sqrt{\frac{1}{B-1}\sum_{b=1}^{B}(\hat{\theta}^{*b} - \bar{\theta}^{*})^2}$$

donde $\bar{\theta}^{*} = \frac{1}{B}\sum_{b=1}^{B}\hat{\theta}^{*b}$

Visualización del Proceso



Ejemplo: Error Estándar de un Coeficiente OLS

Problema: Estimar β en $y = \alpha + \beta x + \varepsilon$

Método tradicional: Usar la fórmula:

$$SE(\hat{\beta}) = \sqrt{\frac{\hat{\sigma}^2}{\sum(x_i - \bar{x})^2}}$$

Requiere suponer homocedasticidad.

Bootstrap: No necesita este supuesto.

Demo: Bootstrap para Regresión

Ver código en: `tree_examples.R`

Lo que haremos:

1. Generar datos con heterocedasticidad
2. Estimar β con OLS
3. Calcular SE con fórmula tradicional (incorrecta bajo heterocedasticidad)
4. Calcular SE con bootstrap
5. Comparar con SE robusto (Huber-White)

Resultado esperado: Bootstrap captura la heterocedasticidad, fórmula tradicional no.

Tipos de Bootstrap

1. **Bootstrap de casos (pairs bootstrap):**
   - Remuestrear **pares** (x_i, y_i) completos
   - Preserva la relación entre X e Y
   - Más general, no requiere supuestos
2. **Bootstrap de residuos:**
   - Ajustar modelo, guardar residuos $\hat{\varepsilon}_i$
   - Remuestrear residuos, generar nuevos $y_i^* = \hat{\alpha} + \hat{\beta} x_i + \hat{\varepsilon}_i^*$
   - Requiere que los errores sean i.i.d.
3. **Wild bootstrap:**
   - Para heterocedasticidad severa
   - Multiplica residuos por pesos aleatorios

Propiedades del Bootstrap

Ventajas:

1. **No paramétrico:** No requiere supuestos distribucionales
2. **General:** Funciona para cualquier estadístico
3. **Fácil de implementar:** Solo requiere remuestreo
4. **Captura heterocedasticidad:** Automáticamente

Limitaciones:

1. **Falla con muestras pequeñas:** Necesitas n razonablemente grande ($n > 30$)
2. **Observaciones dependientes:** Requiere bootstrap en bloques
3. **Computacionalmente intensivo:** B re-estimaciones
4. **No es magic:** Si tu muestra es sesgada, bootstrap también lo será

Conexión con Bagging

Bootstrap Aggregating (Bagging): Misma idea, aplicada a predicción

En inferencia (bootstrap estándar):

- Remuestreo para estimar variabilidad
- Reportamos **un** estimador + su SE

En predicción (bagging):

- Remuestreo para reducir varianza
- **Promediamos** múltiples modelos

Clave común: Explotar la variabilidad del remuestreo para mejorar inferencia/predicción

La Regla de 37%

Recordatorio matemático:

En cada muestra bootstrap de tamaño n (con reemplazo):

$$P(\text{obs } i \text{ no aparece}) = \left(1 - \frac{1}{n}\right)^n \to \frac{1}{e} \approx 0.37$$

Implicaciones:

- Cada muestra continene en média, un 63% de las observaciones, algunas repetidas. Las otras 37% no hacen parte.

Parte 1: Árboles de Decisión

Motivación: Límites de los Métodos Lineales

Problema: Queremos predecir salarios usando educación, experiencia, etc.

Métodos anteriores:

- Regresión lineal: $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \varepsilon$
- Polinomios: $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \varepsilon$
- Splines: regiones conectadas suavemente

Limitaciones:

- Necesitas especificar interacciones manualmente
- Difícil capturar relaciones complejas
- ¿Qué pasa si hay interacciones de orden superior?

Solución: Métodos que aprenden la estructura automáticamente

Ejemplo Motivador

Pregunta: ¿Cómo varía el salario con educación y experiencia?

Realidad: La relación es compleja

- Alta educación + alta experiencia = salarios muy altos
- Alta educación + baja experiencia = salarios moderados
- Baja educación + alta experiencia = salarios moderados
- Baja educación + baja experiencia = salarios bajos

Desafío: Capturar estas interacciones sin especificarlas manualmente

Árboles: Particionan el espacio de características recursivamente

¿Qué es un Árbol de Decisión?

Idea básica: Dividir el espacio de predictores en regiones rectangulares

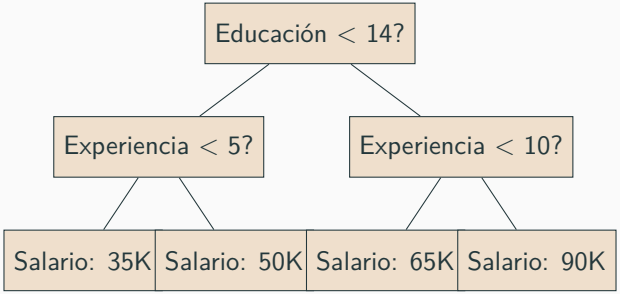
Proceso:

1. Comenzar con todos los datos
2. Encontrar la mejor variable y punto de corte
3. Dividir los datos en dos grupos
4. Repetir para cada grupo
5. Detenerse cuando se cumple un criterio

Predicción: Usar el promedio (regresión) o mayoría (clasificación) en cada región

Ejemplo Visual: Árbol Simple

Datos: Salario vs Experiencia y Educación



Interpretación: 4 regiones, 4 predicciones diferentes

Cómo Crecer un Árbol: El Algoritmo

Algoritmo de partición recursiva (greedy):

Para cada nodo:

1. **Considerar todas las variables** X_j
2. **Para cada variable, considerar todos los puntos de corte** s
3. **Dividir:**
   - Región 1: $X_j < s$
   - Región 2: $X_j \geq s$
4. **Elegir** la división que más reduce la suma de cuadrados de residuos (RSS):

$$\text{RSS} = \sum_{i \in R_1} (y_i - \bar{y}_{R_1})^2 + \sum_{i \in R_2} (y_i - \bar{y}_{R_2})^2$$

5. **Repetir** recursivamente en cada región

Árboles de Regresión: Criterio de División

Objetivo: Minimizar la varianza dentro de cada región

RSS total:

$$\sum_{m=1}^{|T|} \sum_{i \in R_m} (y_i - \bar{y}_{R_m})^2$$

donde:

- $|T|$ = número de nodos terminales (hojas)
- R_m = región correspondiente al nodo m
- $\bar{y}_{R_m}$ = promedio de y en la región m

Intuición: Queremos regiones homogéneas (baja varianza interna)

Árboles de Clasificación: Criterios Alternativos

Para problemas de clasificación, usamos:

1. Índice de Gini:

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

donde $\hat{p}_{mk}$ = proporción de observaciones de clase k en nodo m

2. Entropía:

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk})$$

Ambos miden "pureza":

- Valor bajo = nodo puro (una sola clase domina)
- Valor alto = nodo mezclado (clases distribuidas uniformemente)

¿Cuándo Detener el Crecimiento?

Criterios comunes:

1. **Profundidad máxima:** Limitar niveles del árbol
2. **Tamaño mínimo de nodo:** No dividir si $n < n_{min}$
3. **Mejora mínima:** Detener si la reducción en RSS es pequeña
4. **Validación cruzada:** Elegir tamaño óptimo

Trade-off:

- Árbol grande (profundo): Bajo sesgo, alta varianza (overfitting)
- Árbol pequeño (poco profundo): Alto sesgo, baja varianza (underfitting)

Estrategia común: Crecer árbol grande, luego podar usando CV

Poda de Árbol: Cost-Complexity Pruning

Idea: Penalizar el tamaño del árbol

Función objetivo:

$$\sum_{m=1}^{|T|} \sum_{i \in R_m} (y_i - \bar{y}_{R_m})^2 + \alpha |T|$$

donde:

- $|T|$ = número de nodos terminales
- α = parámetro de complejidad (tuning parameter)

Proceso:

1. Crecer árbol grande
2. Para cada α, encontrar el subárbol que minimiza la función
3. Usar CV para elegir α óptimo

Ejemplo: Datos de Salarios (ISLR)

Dataset: `Wage` del paquete ISLR

Variables:

- Outcome: `wage` (salario en miles de dólares)
- Predictores: `age`, `education`, `year`, `jobclass`, etc.

Pregunta: ¿Podemos predecir salarios usando un árbol?

Demostración: Árbol Simple

Ver código en: `tree_examples.R`

Lo que haremos:

1. Ajustar árbol de regresión a datos Wage
2. Visualizar el árbol
3. Examinar las divisiones elegidas
4. Comparar predicciones con regresión lineal
5. Evaluar en conjunto de prueba

Resultado esperado: El árbol captura algunas interacciones automáticamente

Interpretación del Árbol

Ventajas de los árboles:

1. **Fáciles de explicar:** Reglas simples if-then
2. **Visualización intuitiva:** Diagramas de árbol
3. **Capturan interacciones:** Automáticamente
4. **Manejan variables categóricas:** Sin crear dummies
5. **Robustos a outliers:** Divisiones basadas en rangos

Limitaciones:

1. **Alta varianza:** Pequeños cambios en datos $\rightarrow$ árbol muy diferente
2. **No suaves:** Predicciones en escalones
3. **Menos precisos:** Que otros métodos en muchos casos

El Problema de Alta Varianza

Experimento mental:

- Datos originales $\rightarrow$ Árbol A
- Remover 5% de observaciones $\rightarrow$ Árbol B completamente diferente

Causa: Algoritmo greedy + cambios tempranos afectan todo lo demás

Solución: Promediar muchos árboles (ensemble methods)

Parte 2: Bootstrap Aggregating (Bagging)

La Idea de Bagging

Intuición: Si una estimación tiene alta varianza, promediar muchas estimaciones reduce la varianza

Recordatorio de estadística:

$$\text{Var}\left(\frac{1}{B}\sum_{b=1}^{B} Z_b\right) = \frac{\sigma^2}{B}$$

si las Z_b son independientes e idénticamente distribuidas.

Idea de Bagging:

1. Generar B muestras bootstrap de los datos
2. Entrenar un árbol en cada muestra
3. Promediar las predicciones

Algoritmo de Bagging

Para $b = 1, \ldots, B$:

1. Generar muestra bootstrap: tomar n observaciones con reemplazo
2. Entrenar árbol T_b en esta muestra
3. Guardar el modelo

Predicción para nueva observación x:

- **Regresión:** $\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} T_b(x)$
- **Clasificación:** Voto mayoritario de los B árboles

Típicamente: $B = 500$ o $B = 1000$

Estimación Out-of-Bag (OOB)

Observación clave: Cada muestra bootstrap omite ~37% de los datos originales

Error OOB:

1. Para cada observación i, identificar árboles que no usaron i en entrenamiento
2. Predecir y_i promediando solo esos árboles
3. Calcular error cuadrático medio sobre todas las observaciones

Ventaja: Estimación de error de validación gratis, sin necesidad de conjunto de validación separado

En la práctica: Error OOB $\approx$ error de validación cruzada

Bagging: Pros y Contras

Ventajas:

- Reduce varianza significativamente
- Mejora precisión de predicción
- Error OOB para validación gratis

Desventajas:

- **Pérdida de interpretabilidad:** Ya no tenemos un solo árbol
- **Correlación entre árboles:** Si hay un predictor muy fuerte, todos los árboles lo usan primero
- Aún podemos calcular importancia de variables

Importancia de Variables en Bagging

Problema: No podemos interpretar B árboles simultáneamente

Solución: Medir cuánto contribuye cada variable

Método: Para cada variable X_j, calcular:

$$\text{Importancia}(X_j) = \frac{1}{B}\sum_{b=1}^{B} \text{Reducción total en RSS por } X_j \text{ en árbol } b$$

Interpretación: Variables con alta importancia son las más útiles para predicción

Limitación: Árboles siguen correlacionados (usan predictores similares)

Parte 3: Random Forests

El Problema con Bagging

Situación común:

- Un predictor muy fuerte domina
- Todos los árboles bagged hacen primera división en ese predictor
- Los árboles son muy similares entre sí (correlacionados)

Consecuencia: Promediar árboles correlacionados reduce menos la varianza

Intuición:

$$\text{Var}\left(\frac{1}{B}\sum_{b=1}^{B} Z_b\right) = \frac{\sigma^2}{B} + \frac{B-1}{B}\rho\sigma^2$$

donde ρ = correlación entre los Z_b

Queremos reducir ρ (decorrelacionar los árboles)

Random Forests: Decorrelación Forzada

Idea clave: En cada división, considerar solo un subconjunto aleatorio de predictores

Algoritmo:

Para $b = 1, \ldots, B$**:**

1. Generar muestra bootstrap
2. Entrenar árbol, pero en **cada división**:
   - Seleccionar aleatoriamente m predictores de los p totales
   - Solo considerar esos m para la división
   - Típicamente: $m \approx \sqrt{p}$ (clasificación) o $m \approx p/3$ (regresión)
3. Crecer árbol completamente (sin poda)

Predicción: Igual que bagging (promedio o voto mayoritario)

¿Por Qué Funciona Random Forest?

Mecanismo:

- Forzar al algoritmo a considerar predictores más débiles
- Algunos árboles no usan el predictor más fuerte
- Árboles son menos correlacionados

Resultado: Menor varianza en el promedio

Parámetro clave: m (mtry)

- $m = p$: Igual que bagging
- $m = 1$: Máxima decorrelación (pero quizás mucho sesgo)
- $m = \sqrt{p}$ o $m = p/3$: Balance típico

Random Forest: Algoritmo Completo

Hiperparámetros principales:

1. **ntree (B):** Número de árboles (típicamente 500-2000)
2. **mtry (m):** Número de predictores a considerar en cada división
3. **nodesize:** Tamaño mínimo de nodos terminales
4. **maxnodes:** Número máximo de nodos terminales

Ventajas sobre árboles individuales:

- Mucha mejor precisión
- Más estable
- Menos overfitting
- Importancia de variables

Importancia de Variables en Random Forest

Dos medidas principales:

1. **Reducción promedio en RSS (o Gini):**
   - Para cada variable, sumar reducción en RSS en todas las divisiones (todos los árboles)
   - Normalizar
2. **Importancia por permutación:**
   - Para cada árbol: permutar variable j en datos OOB
   - Calcular aumento en error OOB
   - Promediar sobre todos los árboles

Interpretación: Variables con alta importancia son más útiles para predicción

Random Forest en Práctica

Ventajas:

1. **Funciona "out of the box":** Poco tuning necesario
2. **Robusto:** Difícil de sobreajustar
3. **Maneja muchas variables:** Incluso cuando $p > n$
4. **No necesita preprocesamiento:** No escalar, no crear dummies para variables categóricas
5. **Error OOB:** Validación gratis

Desventajas:

1. **Caja negra:** Difícil de interpretar
2. **Computacionalmente intensivo:** B árboles completos
3. **Problemas con extrapolación:** No predice fuera del rango de entrenamiento

Demostración: Wage Prediction con RF

Ver código en: `tree_examples.R`

Comparación:

1. Árbol individual
2. Bagging
3. Random Forest (diferentes valores de mtry)
4. Comparar errores de predicción
5. Examinar importancia de variables

Pregunta: ¿Qué variables son más importantes para predecir salarios?

Parte 4: Boosting

Una Filosofía Diferente

Bagging/Random Forest: Árboles independientes en paralelo

- Generar muestras bootstrap
- Entrenar árboles simultáneamente
- Promediar predicciones

Boosting: Árboles secuenciales, cada uno corrigiendo errores del anterior

- Empezar con modelo simple
- Identificar errores
- Entrenar nuevo árbol enfocándose en los errores
- Repetir

Intuición: Aprendizaje Iterativo

Analogía: Aprender un tema difícil

Mal método (bagging):

- Estudiar todo el material 100 veces de forma independiente

Buen método (boosting):

1. Primera pasada: aprender lo básico
2. Identificar lo que no entendimos
3. Enfocarnos en las partes difíciles
4. Repetir hasta dominar todo

Boosting hace esto con datos:

- Enfocarse en observaciones mal predichas
- Cada nuevo árbol corrige errores previos

Gradient Boosting: La Idea Central

Pregunta clave: ¿Cómo mejorar iterativamente un modelo?

Respuesta: Entrenar nuevos modelos para **corregir los errores** del modelo actual.

Intuición

1. Empezar con predicción simple (e.g., promedio)
2. Calcular errores: $\text{error}_i = y_i - \hat{f}(x_i)$
3. Entrenar nuevo árbol para predecir estos errores
4. Actualizar: $\hat{f}_{\text{nuevo}} = \hat{f}_{\text{viejo}} + \text{corrección}$
5. Repetir

Cada árbol intenta corregir lo que los árboles anteriores no capturaron.

Gradient Boosting: Algoritmo Paso a Paso

Gradient Boosting: Algoritmo Formal

- Función de pérdida: $L(y, f(x))$
- Objetivo: minimizar $\sum_{i=1}^{n} L(y_i, f(x_i))$

1. **Inicializar** con modelo constante:

$$f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$$

2. **Para** $m = 1, \ldots, M$**:**

   a. Calcular **pseudo-residuos** (explicación siguiente)

   b. Entrenar árbol $h_m(x)$ para predecir pseudo-residuos

   c. **Actualizar** modelo:

$$f_m(x) = f_{m-1}(x) + \lambda h_m(x)$$

donde λ es el learning rate (constante para todos los árboles)

Pseudo-Residuos: ¿Qué Son?

Nombre formal: "Pseudo-residuos" o "gradiente negativo"

Fórmula general:

$$r_{im} = -\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\bigg|_{f=f_{m-1}}$$

En palabras: La dirección en la que debemos mover nuestra predicción $f(x_i)$ para reducir la pérdida.

¿Por qué "pseudo"?

No son los residuos verdaderos $(y_i - \hat{y}_i)$ para funciones de pérdida generales. Son la **dirección de máximo descenso** en el espacio de funciones.

Pseudo-Residuos: Caso Simple

Para regresión con pérdida cuadrática:

$$L(y, f) = \frac{1}{2}(y - f)^2$$

Derivada:

$$-\frac{\partial L(y, f)}{\partial f} = -\frac{\partial}{\partial f}\left[\frac{1}{2}(y - f)^2\right] = y - f$$

Resultado: Los pseudo-residuos son exactamente los **residuos ordinarios**:

$$r_{im} = y_i - f_{m-1}(x_i)$$

¡En regresión es intuitivo!

Entrenar nuevo árbol para predecir los errores actuales = entrenar para predecir residuos.

Pseudo-Residuos: Otros Casos

Para clasificación binaria con pérdida logística:

$$L(y, f) = \log(1 + e^{-yf}) \quad (y \in \{-1, +1\})$$

Pseudo-residuos:

$$r_{im} = \frac{y_i}{1 + e^{y_i f_{m-1}(x_i)}}$$

Estos NO son residuos simples, pero siguen siendo la dirección para mejorar.

Para otras pérdidas (MAE, Huber, etc.): Cada una tiene su propia forma de pseudo-residuos.

Intuición: Descenso de Gradiente en Espacio de Funciones

Analogía con optimización estándar:

En optimización numérica:

$$\theta_{t+1} = \theta_t - \lambda \cdot \nabla_\theta L(\theta_t)$$

En boosting:

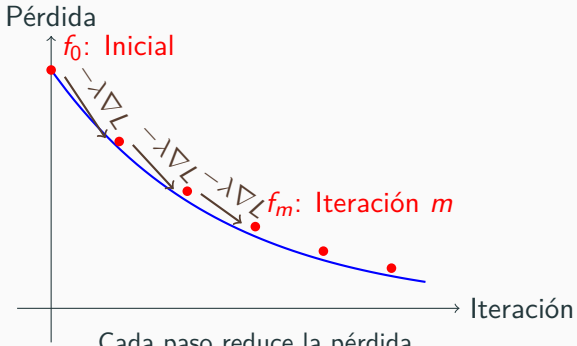
$$f_m(x) = f_{m-1}(x) - \lambda \cdot [-\text{gradiente}]$$

Diferencia clave:

- En optimización: actualizamos **parámetros** θ
- En boosting: actualizamos una **función** $f(x)$

El árbol $h_m(x)$ **aproxima el gradiente negativo** (pseudo-residuos), moviéndonos en la dirección de máximo descenso.

Visualización del Proceso



Cada árbol h_m nos mueve en la dirección que más reduce la pérdida.

Ejemplo Concreto: Regresión

Datos: (x_i, y_i), queremos predecir y

Iteración 0: $f_0(x) = \bar{y} = 50$ (predecir la media)

Iteración 1:

- Residuos: $r_i = y_i - 50$
- Entrenar árbol $h_1(x)$ para predecir r_i
- Actualizar: $f_1(x) = 50 + 0.1 \cdot h_1(x)$

Iteración 2:

- Nuevos residuos: $r_i = y_i - f_1(x_i)$ (basados en modelo actualizado)
- Entrenar árbol $h_2(x)$ para predecir estos nuevos r_i
- Actualizar: $f_2(x) = f_1(x) + 0.1 \cdot h_2(x)$

Continuar: Cada árbol corrige los errores que quedan.

¿Por Qué Funciona el Learning Rate?

$$f_m(x) = f_{m-1}(x) + \lambda h_m(x)$$

Con $\lambda = 1$:

- Cada árbol se agrega completamente
- Riesgo: sobreajustar, "pasarse de largo"

Con $\lambda = 0.1$:

- Movimiento pequeño, cauteloso
- Resultado: más suave, menos overfitting

Trade-off:

$$\text{Complejidad final} \approx \lambda \times M$$

Pequeño λ requiere más árboles M, pero da mejor generalización.

Hiperparámetros de Boosting

Tres parámetros clave:

1. **Número de árboles (M):**
   - Más árboles = más flexible, pero puede sobreajustar
   - Típicamente: 1000-5000, usar early stopping
   - A diferencia de RF, boosting SÍ puede sobreajustar con demasiados árboles
2. **Learning rate (λ):**
   - Controla cuánto contribuye cada árbol: $f_m = f_{m-1} + \lambda h_m$
   - λ pequeño (0.01-0.1) = aprendizaje lento, más estable
   - λ grande (0.3-1.0) = aprendizaje rápido, más riesgo de overfitting
   - Trade-off: λ pequeño requiere más árboles (M grande)
3. **Profundidad de árboles (d):**
   - Controla complejidad de cada árbol individual
   - Típicamente: $d = 1$ (stumps) a $d = 6$
   - d = orden máximo de interacciones capturadas
   - Árboles en boosting suelen ser poco profundos ("weak learners")

Regularización en Boosting

Problema: Boosting puede sobreajustar

Estrategias de regularización:

1. **Learning rate (λ):** Más pequeño = más regularización
2. **Early stopping:**
   - Monitorear error en conjunto de validación
   - Detener cuando error empieza a aumentar
3. **Subsampling (Stochastic Gradient Boosting):**
   - En cada iteración, usar solo fracción de los datos (e.g., 50%)
   - Reduce varianza, acelera cómputo
4. **Shrinkage adicional:** Penalizar complejidad del árbol

XGBoost: Estado del Arte

XGBoost (Extreme Gradient Boosting) es la implementación más popular

Mejoras sobre gradient boosting estándar:

1. **Regularización L1 y L2:** En la función objetivo
2. **Sparsity awareness:** Maneja missing values inteligentemente
3. **Parallel computing:** Entrenamiento paralelo de árboles
4. **Tree pruning:** Poda más agresiva
5. **Built-in CV:** Para early stopping

Función objetivo de XGBoost:

$$\text{Obj} = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

donde $\Omega(f_k) = \gamma T + \frac{1}{2}\lambda||\omega||^2$ penaliza complejidad

Hiperparámetros de XGBoost

Principales a tunear:

1. **nrounds:** Número de árboles (boosting iterations)
2. **eta:** Learning rate (típicamente 0.01-0.3)
3. **max_depth:** Profundidad máxima de árboles (3-10)
4. **subsample:** Fracción de observaciones (0.5-1.0)
5. **colsample_bytree:** Fracción de variables (0.5-1.0)
6. **lambda, alpha:** Penalizaciones L2 y L1

Estrategia:

- Empezar con defaults
- Tunear learning rate y nrounds primero
- Luego profundidad y sampling

Demostración: XGBoost en Práctica

Ver código en: `tree_examples.R`

Aplicación: Predicción de salarios

1. Preparar datos para XGBoost
2. Tunear hiperparámetros con validación cruzada
3. Entrenar modelo final
4. Comparar con Random Forest
5. Examinar importancia de variables
6. Visualizar curvas de aprendizaje

Parte 5: Comparación y Aplicaciones

Ejemplo Completo: Credit Default

Problema: Predecir si un cliente hará default en un préstamo

Datos: German Credit Data

- **Outcome:** Default (1) o No default (0)
- **Predictores:** Historial crediticio, monto del préstamo, empleo, edad, etc.
- $n = 1000$, $p \approx 20$

Métodos a comparar:

1. Regresión logística (baseline)
2. Árbol de decisión individual
3. Random Forest
4. XGBoost

Demostración: Credit Scoring

Ver código en: `tree_examples.R`

Flujo de trabajo:

1. **Exploración:** Distribución de variables, tasa de default
2. **Train/Test split:** 70/30
3. **Entrenar modelos:**
   - Logit con variables importantes
   - Árbol con CV para profundidad
   - RF con tuning de mtry
   - XGBoost con tuning completo
4. **Comparar:** Curvas ROC, AUC, matriz de confusión
5. **Interpretar:** Importancia de variables

Resultados Esperados: Credit Default

Rendimiento típico (AUC):

- Regresión logística: ~0.75
- Árbol individual: ~0.70
- Random Forest: ~0.78
- XGBoost: ~0.80

Importancia de variables (típicamente):

1. Historial crediticio
2. Duración del préstamo
3. Monto del préstamo
4. Saldo en cuenta
5. Propósito del préstamo

Trade-offs Entre Métodos

	Árbol	**Bagging**	**RF**	**Boosting**
Precisión	Baja	Media	Alta	Muy alta*
Interpretabilidad	Alta	Baja	Baja	Baja
Velocidad	Rápido	Lento	Medio	Lento
Tuning	Mínimo	Mínimo	Poco	Mucho
Overfitting	Alto	Bajo	Muy bajo	Medio

{*Con tuning apropiado. Sin tuning, boosting puede ser peor que métodos más simples.}

Lección Importante: XGBoost No Es Magia

Observación común: XGBoost con parámetros por defecto puede ser **peor** que Random Forest o incluso regresión logística.

¿Por qué?

- XGBoost tiene muchos hiperparámetros interdependientes
- Datasets pequeños ($n < 5000$) son particularmente sensibles
- Sin regularización apropiada, puede sobreajustar

Estrategia recomendada:

1. **Empezar con RF:** Funciona bien "out of the box"
2. **Si necesitas más precisión:** Invertir tiempo en tuning de XGBoost
3. **Comparar siempre:** XGBoost tuneado vs RF default

En competiciones: XGBoost gana porque la gente invierte días tuneando.
En producción: RF puede ser mejor por robustez y menor mantenimiento.

Guía de Decisión: ¿Cuál Usar?

Árbol individual:

- Necesitas interpretabilidad máxima
- Baseline rápido
- Conjuntos de datos pequeños

Random Forest:

- Quieres buena precisión sin mucho tuning
- "Default choice" para problemas complejos
- Importancia de variables robusta

Boosting/XGBoost:

- Necesitas la mejor precisión posible
- Tienes tiempo para tunear
- Competiciones de predicción

Parte 6: Consideraciones Prácticas

Sesgo-Varianza Revisitado

Árboles individuales:

- **Sesgo:** Puede ser bajo (si árbol profundo)
- **Varianza:** Muy alta
- **Resultado:** Inestable, sobreajusta fácilmente

Bagging/Random Forest:

- **Sesgo:** Similar a árbol individual
- **Varianza:** Mucho menor (promediando)
- **Resultado:** Reduce overfitting significativamente

Boosting:

- **Sesgo:** Se reduce iterativamente
- **Varianza:** Controlada por learning rate y early stopping
- **Resultado:** Puede reducir ambos si se tunea bien

Validación y Tuning

Estrategias recomendadas:

Random Forest:

1. Usar error OOB (no necesitas CV)
2. Probar diferentes valores de mtry: $\{\sqrt{p}, p/3, p/2\}$
3. Aumentar ntree si el error OOB no se estabiliza

XGBoost:

1. **Separar conjunto de validación** (no usar en entrenamiento)
2. Usar early stopping con `watchlist`
3. Grid search o random search para hiperparámetros
4. Empezar con pocas iteraciones, aumentar gradualmente

Paquete recomendado: `caret` o `tidymodels` para pipeline completo

Ventajas de Métodos de Árboles

1. **Manejan diferentes tipos de datos:**
   - Numéricas y categóricas juntas
   - No necesitas crear variables dummy
2. **Robusto a outliers:**
   - Divisiones basadas en orden, no en valores absolutos
3. **Selección automática de variables:**
   - Variables irrelevantes son ignoradas
4. **Capturan interacciones:**
   - Sin necesidad de especificarlas manualmente
5. **No necesitan scaling:**
   - A diferencia de Ridge/Lasso

Limitaciones de Métodos de Árboles

1. **Menos interpretables (ensembles):**
   - Excepto árbol individual
   - Importancia de variables ayuda pero no es causal
2. **Problemas con extrapolación:**
   - No pueden predecir fuera del rango de entrenamiento
   - Predicciones planas en extremos
3. **Computacionalmente intensivos:**
   - RF: B árboles completos
   - XGBoost: muchas iteraciones
4. **Tuning de boosting es delicado:**
   - Fácil sobreajustar si no se tiene cuidado
   - Requiere validación cuidadosa

Interpretación: Más Allá de Importancia

Métodos para interpretar modelos complejos:

1. **Partial Dependence Plots (PDP):**
   - Efecto marginal de una variable manteniendo las demás constantes
   - Muestra relación funcional promedio
2. **Individual Conditional Expectation (ICE):**
   - PDP para cada observación individual
   - Muestra heterogeneidad en efectos
3. **SHAP values:**
   - Teoría de juegos cooperativos
   - Atribuye contribución de cada variable a cada predicción

Exploraremos estos en más detalle en Clase 7

Partial Dependence Plot: Ejemplo

Pregunta: ¿Cómo afecta la edad al salario predicho?

Método:

1. Para valores de edad $= \{20, 25, 30, \ldots, 60\}$:
2. Fijar edad en ese valor para todas las observaciones
3. Predecir salario con RF
4. Promediar predicciones
5. Graficar edad vs salario promedio predicho

Interpretación: Relación no-lineal capturada por el modelo

Ejemplo Visual: PDP

Lo que veríamos:

- Salario aumenta con edad hasta ~45
- Luego se estabiliza o decrece ligeramente
- Captura relación no-lineal automáticamente

Importancia de Variables: Interpretación

Lo que NO significa:

- Causalidad
- Que otras variables no importan
- Relación funcional específica

Lo que SÍ significa:

- Utilidad para predicción
- Priorizar qué variables investigar
- Comparar modelos

Ejemplo en credit scoring:

- "Historial crediticio" tiene alta importancia
- $\rightarrow$ Investigar más profundo esta variable
- $\rightarrow$ Políticas de préstamos deben considerarla

Al entrenar modelos de árboles:

1. **Siempre separar train/test** (o usar CV)
2. **Empezar simple:** Árbol individual como baseline
3. **RF primero:** Antes de intentar boosting
4. **Monitorear overfitting:** Error de entrenamiento vs validación
5. **Examinar importancia:** ¿Tiene sentido económico?
6. **Comparar múltiples modelos:** No confiar en uno solo
7. **Documentar hiperparámetros:** Reproducibilidad

Recursos Computacionales

Paquetes de R recomendados:

- `rpart`: Árboles individuales (CART)
- `rpart.plot`: Visualización de árboles
- `randomForest`: RF (clásico)
- `ranger`: RF (más rápido)
- `xgboost`: Gradient boosting
- `caret`: Framework unificado para todo
- `pdp`: Partial dependence plots
- `vip`: Variable importance plots

Python:

- `scikit-learn`: Todo básico
- `xgboost`, `lightgbm`, `catboost`: Boosting avanzado

Conexión con Clases Anteriores

Clase 4 (Ridge/Lasso):

- Regularización explícita vs implícita
- Ridge penaliza β, RF promedia árboles

Clase 5 (Splines):

- Ambos capturan no-linealidad
- Splines son suaves, árboles son escalones
- Splines mejores para univariados, árboles para interacciones

Clase 7 (próxima):

- Selección de modelos
- Validación cruzada más profunda
- Interpretación avanzada (SHAP)

Resumen: Puntos Clave

1. **Bootstrap:** Remuestreo para estimar variabilidad (SE, IC)
2. **Árboles individuales:** Interpretables pero inestables
3. **Bagging:** Reduce varianza promediando (usa bootstrap)
4. **Random Forest:** Decorrelaciona árboles, mejor predicción
5. **Boosting:** Aprendizaje secuencial, estado del arte
6. **Trade-off fundamental:** Interpretabilidad vs precisión
7. **RF es "free lunch":** Buena precisión con poco tuning
8. **XGBoost para competiciones:** Mejor precisión si se tunea bien
9. **Importancia de variables:** Para interpretación económica

Aplicaciones en Economía

Áreas donde árboles son útiles:

1. **Credit scoring:** Predicción de default
2. **Labor economics:** Predicción de salarios, clasificación de ocupaciones
3. **Política pública:** Targeting de programas sociales
4. **Finanzas:** Predicción de retornos, detección de fraude
5. **Health economics:** Predicción de costos médicos
6. **Marketing:** Segmentación de clientes
7. **Macroeconomía:** Nowcasting, predicción de recesiones

Ventaja clave: Capturan heterogeneidad en efectos automáticamente

Para la Próxima Clase

Clase 7: Model Selection y Overview

- Validación cruzada avanzada
- Comparación de modelos formalmente
- Interpretación con SHAP
- Cuándo usar cada método
- Checklist de ML en economía

Preparación:

- Revisar todos los métodos vistos (Ridge $\rightarrow$ Boosting)
- Pensar en aplicaciones a sus propios datos

Práctica Recomendada

Para consolidar:

1. Ejecutar código de `tree_examples.R`
2. Experimentar con:
   - Diferentes valores de mtry en RF
   - Diferentes learning rates en XGBoost
   - Diferentes profundidades de árbol
3. Aplicar a dataset propio
4. Comparar RF vs XGBoost
5. Interpretar importancia de variables

Pregunta para pensar:

¿En qué situaciones preferirías un árbol simple interpretable sobre un RF más preciso?

Lecturas Recomendadas

Libros:

- James et al. (2013), ISLR, Capítulo 8: Tree-Based Methods
- Hastie et al. (2009), ESL, Capítulo 10 y 15: Boosting

Papers:

- Breiman (2001): Random Forests (original)
- Chen & Guestrin (2016): XGBoost (KDD)
- Athey & Imbens (2016): Recursive Partitioning for Heterogeneous Causal Effects

Aplicaciones:

- Kleinberg et al. (2018): Human Decisions and Machine Predictions (QJE)
- Mullainathan & Spiess (2017): Machine Learning: An Applied Econometric Approach (JEP)

Fin de Clase 6

Hoy aprendimos:

- Bootstrap: remuestreo para inferencia
- Árboles de decisión (CART)
- Bagging y Random Forests
- Gradient Boosting y XGBoost
- Comparación práctica de métodos
- Aplicaciones en credit scoring

Próxima clase:

- Model selection
- Validación formal
- Interpretación avanzada
- Integración de todo lo visto

Preguntas

¿Preguntas sobre árboles, random forests o boosting?