Código del Modelo Bayesiano de Actualización de Opiniones (MBAO-IV)

Calcina Aquino Dina Susan

Código en R

```
library(shiny)
library(igraph)

# UI
ui <- fluidPage(
  titlePanel("Modelo Bayesiano de Actualización de Opiniones (MBAO-IV)"),

  sidebarLayout(
    sidebarPanel(
      h4("Parámetros de la Red"),
      sliderInput("n_nodos", "Número de usuarios:",
                  min = 5, max = 30, value = 15),
      sliderInput("prob_conexion", "Probabilidad de conexión:",
                  min = 0.1, max = 0.5, value = 0.3, step = 0.05),

      hr(),
      h4("Parámetros del Modelo"),
      sliderInput("peso_vecinos", "Peso de influencia vecinal ():",
                  min = 0, max = 1, value = 0.6, step = 0.1),
      sliderInput("iteraciones", "Iteraciones:",
                  min = 1, max = 20, value = 5),

      hr(),
      actionButton("simular", "Simular", class = "btn-primary"),
      actionButton("reset", "Reiniciar")
    ),

    mainPanel(
      h4("Visualización de la Red Social"),
      plotOutput("grafico_red", height = "400px"),
      hr(),
      h4("Estadísticas"),
      verbatimTextOutput("stats")
    )
  )
)
```

```
# SERVER
server <- function(input, output, session) {

  # Variables reactivas
  valores <- reactiveValues(
    grafo = NULL,
    opiniones_inicial = NULL,
    opiniones_actual = NULL
  )

  # Función para crear la red
  crear_red <- function(n, p) {
    g <- erdos.renyi.game(n, p, type = "gnp")
    return(g)
  }

  # Función bayesiana de actualización
  actualizar_opinion_bayesiana <- function(opinion_actual, opiniones_vecinos, alpha) {
    if(length(opiniones_vecinos) == 0) {
      return(opinion_actual)
    }

    # Prior: opinión actual
    prior <- opinion_actual

    # Likelihood: promedio de vecinos (evidencia)
    evidencia <- mean(opiniones_vecinos)

    # Posterior bayesiano simple
    posterior <- alpha * evidencia + (1 - alpha) * prior

    # Mantener entre 0 y 1
    posterior <- max(0, min(1, posterior))

    return(posterior)
  }

  # Botón reset
  observeEvent(input$reset, {
    valores$grafo <- NULL
    valores$opiniones_inicial <- NULL
    valores$opiniones_actual <- NULL
  })

  # Botón simular
  observeEvent(input$simular, {
    # Crear red
    g <- crear_red(input$n_nodos, input$prob_conexion)
    valores$grafo <- g
```

```
  # Opiniones iniciales aleatorias
  n <- vcount(g)
  opiniones <- runif(n, 0, 1)
  valores$opiniones_inicial <- opiniones
  valores$opiniones_actual <- opiniones

  # Proceso iterativo de actualización
  for(iter in 1:input$iteraciones) {
    nuevas_opiniones <- numeric(n)

    for(i in 1:n) {
      # Obtener vecinos del nodo i
      vecinos <- neighbors(g, i)

      if(length(vecinos) > 0) {
        opiniones_vecinos <- valores$opiniones_actual[vecinos]

        # Actualizar con modelo bayesiano
        nuevas_opiniones[i] <- actualizar_opinion_bayesiana(
          valores$opiniones_actual[i],
          opiniones_vecinos,
          input$peso_vecinos
        )
      } else {
        nuevas_opiniones[i] <- valores$opiniones_actual[i]
      }
    }

    valores$opiniones_actual <- nuevas_opiniones
  }
})

# Gráfico de la red
output$grafico_red <- renderPlot({
  req(valores$grafo)

  g <- valores$grafo
  op_inicial <- valores$opiniones_inicial
  op_actual <- valores$opiniones_actual

  # Layout
  set.seed(123)
  layout <- layout_with_fr(g)

  # Colores según opinión
  colores <- colorRampPalette(c("red", "yellow", "green"))(100)
  indices <- round(op_actual * 99) + 1
  node_colors <- colores[indices]

  # Tamaño según cambio
  cambios <- abs(op_actual - op_inicial)
```

```
    tamaños <- 10 + cambios * 30

    # Graficar
    par(mar = c(1,1,2,1))
    plot(g,
         layout = layout,
         vertex.color = node_colors,
         vertex.size = tamaños,
         vertex.label = round(op_actual, 2),
         vertex.label.cex = 0.7,
         vertex.label.color = "black",
         edge.arrow.size = 0.3,
         main = "Red Social - Opiniones Actualizadas")

    legend("bottomright",
           legend = c("Negativa (0)", "Neutral (0.5)", "Positiva (1)"),
           fill = c("red", "yellow", "green"),
           cex = 0.8)
  })

  # Estadísticas
  output$stats <- renderPrint({
    req(valores$opiniones_inicial, valores$opiniones_actual)

    cat("=== ESTADÍSTICAS DEL MODELO ===\n\n")
    cat("Opinión promedio inicial:", round(mean(valores$opiniones_inicial), 3), "\n")
    cat("Opinión promedio final:", round(mean(valores$opiniones_actual), 3), "\n")
    cat("Desviación estándar inicial:", round(sd(valores$opiniones_inicial), 3), "\n")
    cat("Desviación estándar final:", round(sd(valores$opiniones_actual), 3), "\n")

    cambio_promedio <- mean(abs(valores$opiniones_actual - valores$opiniones_inicial))
    cat("\nCambio promedio de opinión:", round(cambio_promedio, 3), "\n")

    # Convergencia
    if(sd(valores$opiniones_actual) < sd(valores$opiniones_inicial)) {
      cat("\n¡Las opiniones CONVERGIERON! (menor dispersión)\n")
    } else {
      cat("\nLas opiniones se DISPERSARON\n")
    }
  })
}

# Ejecutar app
shinyApp(ui = ui, server = server)
```