Reactividad avanzada en Shiny II

¿Qué es un evento?

- Un evento puede ser (generalmente) un botón (actionButton) o un evento de un output.
- El actionButton se puede usar en cualquier reactive, render u observer

```
library(shiny)

ui <- fluidPage(
  actionButton("go", "Go"),
  numericInput("n", "n", 50),
  plotOutput("plot")
)

server <- function(input, output) {

  randomVals <- reactive({
    input$go # el botón se usa en el reactive
    rnorm(isolate(input$n))
  })

  output$plot <- renderPlot({
    hist(randomVals())
  })
}

shinyApp(ui, server)
```

Eventos de botón

- input$go es el botón. Su valor es 0 por defecto y va aumentando de uno en uno cada click
- Esto hace que cada vez que se pulsa se invalide todo lo que dependa del botón. En este caso “randomVals”

```
library(shiny)

ui <- fluidPage(
  actionButton("go", "Go"),
  numericInput("n", "n", 50),
  plotOutput("plot")
)

server <- function(input, output) {

  randomVals <- reactive({
    input$go # el botón se usa en el reactive
    rnorm(isolate(input$n))
  })

  output$plot <- renderPlot({
    hist(randomVals())
  })
}

shinyApp(ui, server)
```

Eventos de output

- Los outputs (por ejemplo las gráficas) puedes ser también inputs haciendo eventos
- Los más comunes son: click, dblclick y hover
- El nombre que pones entre comillas (“click”) es el nombre del input: input$click

```
mainPanel(
    plotOutput("plot", click="click"),
)
```

Eventos de output

- Con este click se pueden hacer muchas cosas, las dos principales son:
  - Acceder a las coordenadas (input$click$x y input$click$y)
  - Acceder a los puntos más cercanos del dataset (con nearPoints)

```
ui <- fluidPage(
    titlePanel("Old Faithful Geyser Data"),
    sidebarLayout(
        sidebarPanel(
        ),
        mainPanel(
           plotOutput("distPlot", click="click"),
           verbatimTextOutput("resultadoClick"),
           dataTableOutput("resultadoFiltro")
        )
    )
)
```

Eventos de output

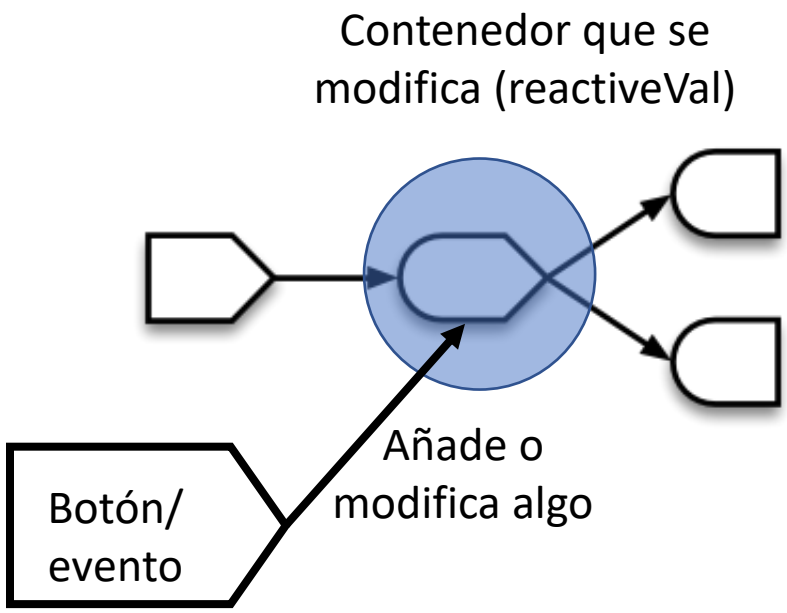
- Acceder a las coordenadas (input$click$x y input$click$y)
- Acceder a los puntos más cercanos del dataset (con nearPoints)

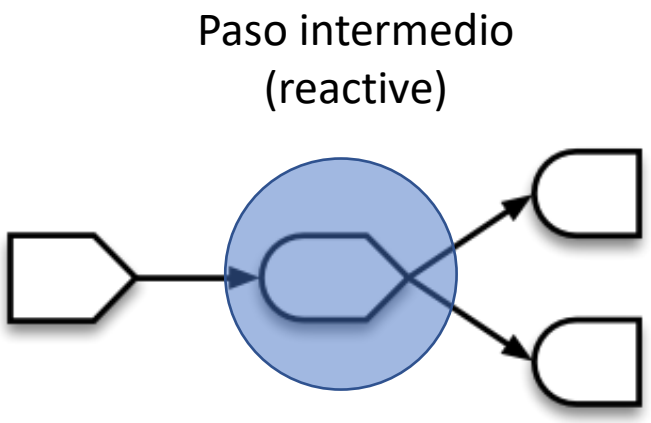
```
server <- function(input, output) {

    output$distPlot <- renderPlot({
        ggplot(mpg, aes(x=cyl, y=hwy)) + geom_point()
    })

    output$resultadoClick <- renderText({
        paste0("Has pulsado en las coordenadas ", input$click$x, ", ",
               input$click$y)
    })

    output$resultadoFiltro <- renderDataTable({
     nearPoints(mpg, input$click)
    })

}
```

reactive vs reactiveVal

reactive NO se puede modificar cuando el programador quiere, sólo reacciona ante los cambios.

reactiveVal permite modificar cuando uno quiera su valor. El caso más típico es usar eventos para modificar un valor. P.e: Añadir un modelo más de los que había en nuestro ejemplo de overfitting





Interfaces reactivas: renderUi y uiOutput

renderUi e uiOutput permiten programar la ui que queramos en la parte del server.

Vamos a usarlo para crear una interfaz que cambie. En concreto vamos a crear distribuciones aleatorias y para cada distribución pediremos al usuario distintos parámetros.

Los parámetros aparecerán y desaparecerán a medida que cambiemos la distribución elegida.

(distribuciones.R)

Interfaces reactivas: renderUI y uiOutput

En el UI ahora podemos añadir un nuevo elemento: “uiOutput”

Funciona como cualquier output:

- Tiene un identificador (“parameters”)
- Se rellena desde el server
- El render correspondiente se llama renderUI

```
ui <- fluidPage(
  selectInput("distribution",
              label = "Elige una distribución",
              choices = list("Uniforme" = "unif",
                             "Normal" = "norm",
                             "Poisson" = "pois"),
              selected = "unif"),
  uiOutput("parameters"),
  plotOutput("plot")
)
```

Interfaces reactivas: renderUI y uiOutput

En el server podemos añadir el renderUI correspondiente.

```
output$parameters <- renderUI({
  numericInput("mean", label = "Media", value = 0),
})
```

Dentro de ese renderUI puedes poner cualquier componente que hayamos usado en shiny. Por ejemplo un numericInput

Interfaces reactivas: renderUI y uiOutput

Si necesitas añadir varios elementos lo envuelves en un padre que nunca hemos usado: “tagList”.

```
output$parameters <- renderUI({
  tagList(
    numericInput("mean", label = "Media", value = 0),
    numericInput("sd", label = "Desv. Típica", value = 1)
  )
})
```

tagList no hace nada visualmente, sólo sirve para encapsular múltiples elementos y que se puedan devolver todos con las llaves {}

Interfaces reactivas: renderUI y uiOutput

La ventaja de usar esta técnica es que te permite crear programáticamente (es decir, programando) interfaces que cambien según algunas condiciones.

Dos casos típicos:

- Parte de los input dependen de otros (como el caso que vamos a resolver)
- El UI tiene elementos muy complicados de escribir a mano. P.e: un selector con todas las empresas del IBEX

Interfaces reactivas: renderUI y uiOutput

Al ser código programado podemos usar if o for o cualquier otra ténica de programación que conozcamos. En este caso if y else if para mostrar los parámetros correspondientes.

```
output$parameters <- renderUI({
  if (input$distribution == "norm") {
    tagList(
      numericInput("mean", label = "Media", value = 0),
      numericInput("sd", label = "Desv. Típica", value = 1)
    )
  } else if (input$distribution == "unif") {
    tagList(
      numericInput("min", label = "Mínimo", value = 0),
      numericInput("max", label = "Máximo", value = 1)
    )
  } else if (input$distribution == "pois") {
    numericInput("lambda", label = "Lambda", value = 1)
  }
})
```