IECD 2C2023: Comentarios Entrega 1

Gonzalo Barrera

2023-09-26

A continuación, algunas observaciones generales sobre fragmentos de código silvestres encontrados en las entregas del TP1.

1. Almacenar resultados de for-loops

MUY peligroso es inicializar vectores pre-poblados: ¿cómo se que es un numero calculado y qué es el número inicial?

```
promedio <- c(1:n.h)
for(j in 1:n.h){
  promedio[j] <- mean(rta[j,])
}
```

Mejor: inicializar un vector vacío. Aún antes que `rep(NA, n)`, lo ideal es crear un vetor vacío del tipo atómico (`"logical"`, `"integer"`, `"numeric"`, `"complex"`, `"character"`, `"raw"`) adecuado:

```
promedio <- vector("numeric", length=n.h)
for(j in 1:n.h){
  promedio[j] <- mean(rta[j,])
}
```

Eficiente pero algo confuso es acumular en un escalar una operación que más naturalmente se entiende como operación sobre un vector.

```
sumatoria <- 0
for (i in 1:n.x) { #indice de los datos
  sumatoria <- sumatoria + f(x,i,h) #almacenamos los valores calculados
}
loglikes[j] <- sumatoria/n.x #promedio de lo obtenido
```

Recomiendo (para vectores "cortos", de longitud en el orden de millones), almacenar el resultado en un vector y operar al final.

```
sumatoria <- vector("numeric", n.x)
for (i in 1:n.x) { #indice de los datos
  sumatoria[i] <- f(x,i,h) #almacenamos los valores calculados
}
loglikes[j] <- mean(sumatoria)
```

2. `type="l"` transforma el "scatterplot" base de `plot` en un gráfico de líneas

```
plot(res$grilla.h, res$loglikes, xlab="h", ylab="log-verosimilitud promedio")
```

devuelve un scatterplot. Agregar la opción `type="l"` devuelve un gráfico de línea que une puntos consecutivos. *Si el argumento* x *está ordenado*, el resultado es un bello gráfico de línea.

```
plot(res$grilla.h, res$loglikes, xlab="h", ylab="log-verosimilitud promedio", type="l")
```

`geom_line` en lugar de `geom_point` logra su equivalente en gráficos basados en `ggplot2`.

3. Nombrar los argumentos de una lista

Una lista definida por argumentos posicionales, implícitamente indexa los elementos con enteros consecutivos `1, 2, ...`:

```
lista <- list(h.CV, grilla.h, loglikes)
```

Mejor darles nombres: ante la duda, los mismos nombres de las variables a guardar:

```
lista <- list(h.CV=h.CV, grilla.h=grilla.h, loglikes=loglikes)
```

Además, esto permite usar la sintaxis del signo `$` para acceder a los atributos `lista$h.cv`, menos abstracta que `lista[[1]]`.

4. Evitar las funciones crípticas

Es conveniente evitar los nombres de variables de un único caracter, aunque en estadística es una práctica muy común. Usadlos con cautela.

Las expresiones muy largas en una misma línea también son problemáticas: los humanos estamos acostumbrados a leer líneas de <= 80 caracteres en libros físicos, y varios estándares de código abogan por similares longitudes.

```
f <- function(x,i,n=length(x),h){
  valor <- ((1/((n-1)*h))*(sum(((1/(sqrt(2*pi))*(exp(((-1/2)*(((x[i]-x[-i])/h)**2)))))))));
  return(valor)
}
```

Un par de sugerencias sencillas: 1. En RStudio, "Tools -> Global Options -> Code -> Display", tildan "Show Margin" y en "Margin Column", ponen 80 u 88 caracteres. Una cómoda regla vertical va a aparecer en la consola para mostrarles hasta dónde es recomendable escribir. 2. Habiendo seleccionado un frangmento de código, en RStudio tocan "Code -> Reformat Code" y automágicamente lo indenta razonablemente:

```
f <- function(x, i, n = length(x), h) {
  valor <-
    ((1 / ((n - 1) * h)) * (sum(((
      1 / (sqrt(2 * pi)) * (exp(((
        -1 / 2
      ) * (((x[i] - x[-i]) / h) ** 2
      ))))
    )))))

  return(valor)
}
```

Aún mejor, abstraigan y denle nombre a algunas funciones intermedias. En este caso, `dnorm` es exactamente el núcleo buscado:

```
f <- function(x, i, n = length(x), h) {
  return(mean(dnorm((x[i] - x[-i]) / h)) / h)
}
```

5. La función de densidad generadora de los datos es conocida

Varios comentarios en las entregas fueron en esta línea:

> "*¿**Si conocieramos la distribución** real de nuestra muestra, cúal de las estimaciones aproximaría mejor?, ¿La de Silverman subestima la probabilidad, o el resto la sobreestima?, ¿Tenemos las herramientas suficientes para hacer estas comparaciones (si nos alcanza con los datos que tenemos)?.*"

> "***Intento tener** la funcion de la densidad real (. . .)*"

Pero efectivamente es conocida la función: la función `muestra` expone el hecho de que la densidad es una mezcla de dos normales.

6. Cada párrafo extra es un riesgo extra

En la medida de lo posible, dejen que el código y los elementos de los gráficos hablen por ustedes. Cuando necesiten agregar descripciones en prosa, consideren que también se exponen a cometer "errores no forzados", y háganlo únicamente si agregan información.

7. No incluir código que no se use

Algunas entregas tomaron el código de referencia de `iecd/resueltos/kde.R`, cosa que alentamos. En la misma línea que el comentario anterior, lo que *no* hay que hacer, es agregar código que luego no se usa: complica innecesariamente la lectura, y hace sospechar que no se entendió qué parte era necesaria y cuál superflua en lo incluido.

8. Grillas en escala logarítmica

Ojo con elegir secuencias lineales equiespaciadas para escalas logarítmicas: van a tener muy pobre resolución en los valores absolutos pequeños.

```
seq(silverman * 10^(-2), silverman * 10^2, length.out = 100)
```

Más razonable, es elegir linealmente espaciados los exponentes apra que la grilla esté equiespaciada en escala logarítmica:

```
silverman * 10 ^ seq(-2, 2, length.out = 100)
```