TP1 - Reconocimiento de Patrones

Elio Campitelli

4/17/2020

1 Generar set de datos

Para cada x, los datos a generear siguen una distribución $p(x,y) = \mathcal{N}(\mu = \sin(2\pi x), \sigma = 0.3)$. Esta función de densidad de probabilidad conjunta se muestra en la Figura 1. El mejor ajuste en el sentido de cuadrados mínimos está dado por $h(x) = \mathbb{E}(y|x) = \sin(2\pi x)$. Ambas funciones ($p(x,y)$ y $\mathbb{E}(y|x)$) se grafican en la Figura 1.

```
densidad <- function(x = seq(0, 1, length.out = 40),
                     y = seq(-2, 2, length.out = 40),
                     FUN = ~sin(2*pi*.x), sigma = 0.3) {
  FUN <-  purrr::as_mapper(FUN)
  grid <- data.table::CJ(x, y)
  grid[, d := dnorm(y, mean = FUN(x), sd = sigma), by = x]
  return(grid)
}
```

```
densidad() %>%
  ggplot(aes(x, y)) +
  geom_contour_filled(aes(z = d), bins = 9) +
  stat_function(fun = ~ sin(2*pi*.x)) +
  scale_fill_viridis_d(guide = "none")
```

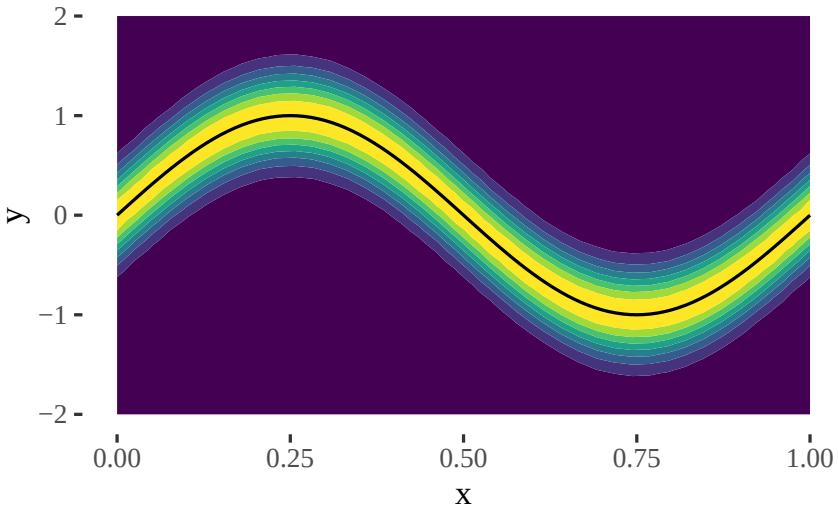


Figure 1: Densidad de probabilidad conjunta $p(x,y) = \mathcal{N}(\sin(2\pi x), 0.3)$. En negro, la línea $\mathbb{E}(y|x)$.

La función `D`[1] devuelve `L` sets de `n` datos. Éstos corresponden a la

[1] Definida en el Apéndice

función `FUN` (default: $\sin(2\pi x)$)) evaluada en `n` puntos elegidos a partir de una distribución uniforme en el intervalo `intervalo` (default: $(0, 1)$) a la que se le suma un ruido gausiano con media 0 y desvío `sigma` (default: 0.3).

```
datos <- D(n = 40, L = 4)
datos %>%
  ggplot(aes(x, t)) +
  stat_function(fun = ~sin(2*pi*.x)) +
  geom_point(size = 0.4) +
  scale_x_continuous(limits = c(0, 1)) +
  facet_wrap(~l, labeller = label_null)
```

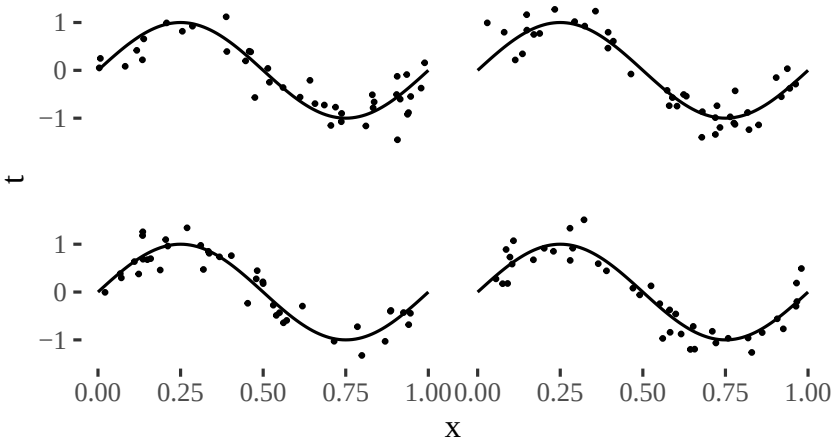


Figure 2: Cuatro ejemplos de conjuntos de datos generados por la función 'D' con 'n = 40'. En línea negra, la función $t = \sin(2\pi x)$

1.1 *Función para calcular la regresión*

`regresion_poly`[2] tiene argumentos `orden` y `lambda` y devuelve una función que realiza el ajuste polinomial correspondiente[3]. Los métodos `predictdf` y `predict`[4] aplican el ajuste a nuevos datos.

La Figura 3 muestra el efecto de cambiar el orden del polinomio para un set de datos de `n = 10`. Un polinomio de orden cero es una constante, por lo que el valor predicho por ese ajuste coincide con el promedio muestral. Polinomio de orden 1 agrega una tendencia, y órdenes mayores van aumentando los grados de libertad del modelo. Para órdenes altos (cercanos a la cantidad de datos usados para realizar el ajuste), el modelo es lo suficientemente complejo para predecir los datos observados con gran exactitud, pero pierde poder de generalización para datos no observados.

```
datos <- D(n = 10, L = 1)
```

[2] Definida en el Apéndice

[3] Esto es una complicación extra pero vale la pena para poder usar la función como argumento de `method` en `geom_smooth()` como se ve en la figura siguiente.

[4] Definidos en el Apéndice

```
ggplot(datos, aes(x, t)) +
  stat_function(fun = ~sin(2*pi*.x)) +
  geom_smooth(method = regresion_poly(orden = c(0:3, 6, 8)),
              aes(color = ..orden.., group = ..orden..),
              size = 0.4, fullrange = TRUE, n = 120) +
  geom_point()  +
  scale_x_continuous(limits = c(0, 1)) +
  coord_cartesian(ylim = c(-2, 2)) +
  facet_wrap(~l, labeller = label_null)
```

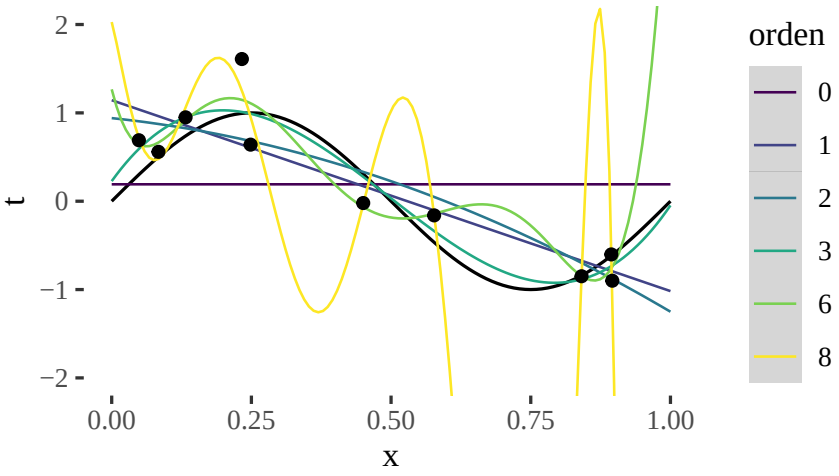


Figure 3: Ajustes polinomiales con distintos órdenes y lambda = 0 para 1 ejemplo. La línea negra representa la función real. Al aumentar el grado del polinomio, el ajuste se acerca más a los puntos observados pero oscila alocadamente lejos de ellos.

En cambio, la Figura **??** muestra el efecto de aumentar el factor de regularización lambda. Al aumentar, aumenta la penalización de coeficientes altos y el modelo deja de ajustar tan bien a los datos observados pero mejora la generalización.

```
ggplot(datos, aes(x, t)) +
  stat_function(fun = ~sin(2*pi*.x)) +
  geom_smooth(method = regresion_poly(orden = 8, lambda = c(0, 10^seq(-8, -1, length.out = 5))),
              aes(color = factor(log10(..lambda..), ordered = TRUE),
                  group = log10(..lambda..)),
              size = 0.4, fullrange = TRUE, n = 120) +
  geom_point()  +
  scale_x_continuous(limits = c(0, 1)) +
  scale_color_viridis_d("log(lambda)", labels = function(x) signif(as.numeric(x), 2)) +
  coord_cartesian(ylim = c(-2, 2)) +
  facet_wrap(~l, labeller = label_null)
```

El error cuadrático medio de entrenamiento se calcula como la diferencia cuadrática media entre los valores observados y los predichos por el modelo. En la Figura 5 se muestra un histograma de la

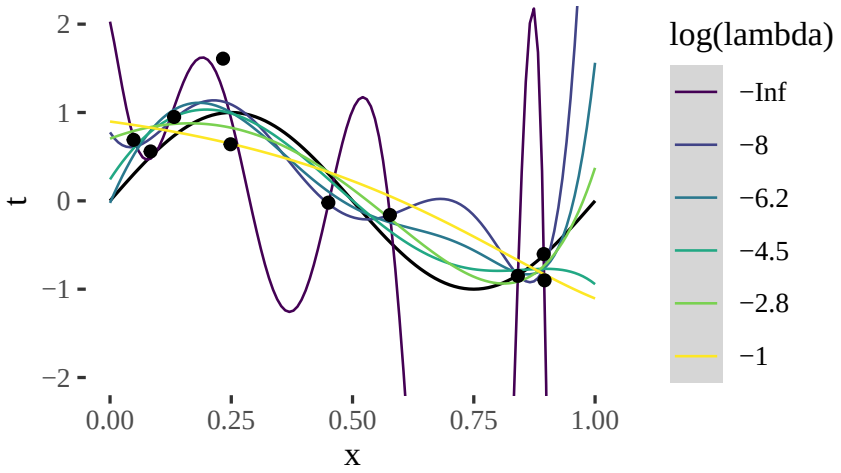


Figure 4: Igual que la figura anterior, pero con orden fijo = 8 y lambda variable. Al aumentar el factor de regularización, el modelo se simplifica. Aumenta la diferencia con los datos usados para el ajuste, pero mejora la generalización.

raiz cuadrada del error cuadrático medio[5] para 200 muestras de `n = 10` haciendo un ajuste con `orden = 3` y `lambda = 1e-3`. En el recuadro, el valor medio del RMSE y su desvío estándar.

[5] RMSE: Root Mean Square Error

```
datos <- D(n = 10, L = 200)
datos[, pred := predict(regresion_poly(orden = 3, lambda = 1e-3)(t ~ x)), by = l] %>%
  .[, .(error_medio = sqrt(mean((t - pred)^2))), by = l] %>%
  ggplot(aes(error_medio)) +
  geom_histogram(binwidth = 0.025, fill = NA, color = "black") +
  geom_label(data = ~.x[, .(mu = mean(error_medio), s = sd(error_medio))],
             aes(label = glue::glue("Media = {signif(mu, 2)}\nSD = {signif(s, 2)}")),
             x = 0.35, y = 30, hjust = 0) +
  scale_x_continuous("RSME") +
  scale_y_continuous(NULL) +
  geom_rug()
```

1.2 *Determinando M y lambda*

Para elegir el orden y el lambda se puede usar validación cruzada. Para cada combinación de los hiperpaŕametros se separa los datos en un conjunto de *entrenamiento* que se usa para ajustar un modelo y uno de *validación*, que se usa para evaluar el error del modelo a datos nuevos. Se busca la combinación que minimice el error de validación y finalmente se estima el error esperado con el conjunto de *test*.

Esta es la matriz de parámetros donde voy a buscar. Lambda entre 10^-10 y 1, y el orden del polinomio entre 0 y 11

```
params <- CJ(lambda = 10^seq(-10, 0, length.out = 15), orden = 0:15)
```

Defino una función para calcular el RSME de validación cruzada[6].

[6] Definida en el Apéndice

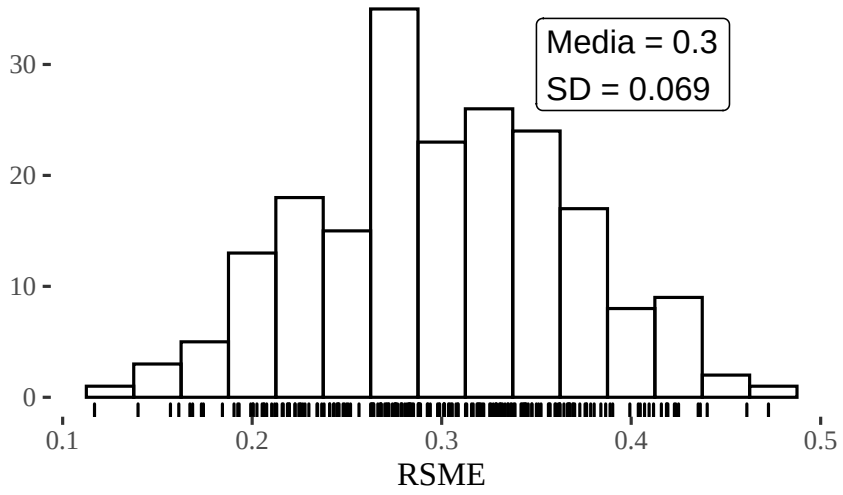


Figure 5: Histograma de la raiz del error cuadrático medio computado para 200 muestras de 'n = 10'.

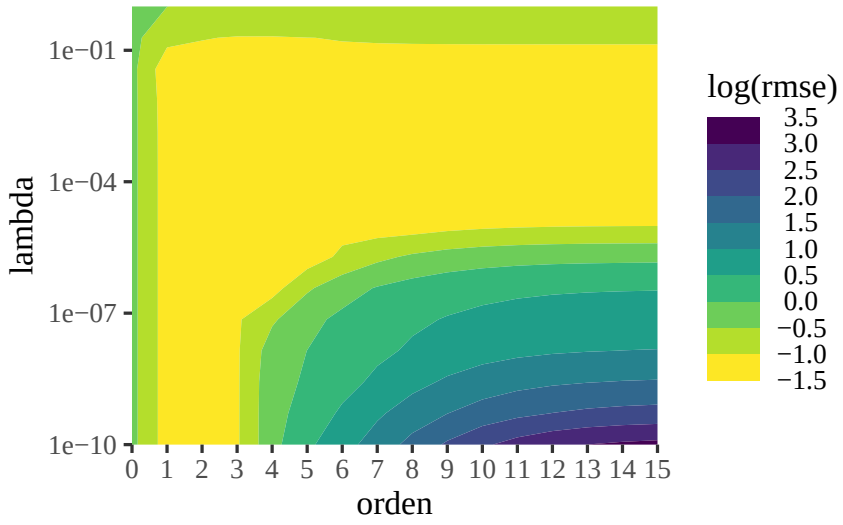
Ésta toma un set de datos, una formula que determina el modelo y una función de regresión (lo que devuelve `regresion_poly()`). Tiene un argumento `k_fold` (default: 10) que controla la cantidad de cachos. Si `k_fold = n`, el algoritmo se reduce a LOOCV[7]

Corriendo todo. Para cada lambda y orden, calculo el RMSE.

[7] Leave One Out Cross-Validation. Es decir, que se ajusta el modelo usando todos los datos menos uno.

```
cv <- params[, .(rmse = rmse_cv(t ~ x,
                                datos = datos[l == 1],
                                fit_fun = regresion_poly(orden = orden, lambda = lambda))),
             by = .(lambda, orden)]

ggplot(cv, aes(orden, lambda)) +
  geom_contour_filled(aes(z = log(rmse))) +
  scale_y_log10(expand = c(0, 0)) +
  scale_x_continuous(breaks = unique(cv$orden), expand = c(0, 0)) +
  scale_fill_viridis_d("log(rmse)", direction = -1,
                       guide = guide_colorsteps(show.limits = TRUE))
```



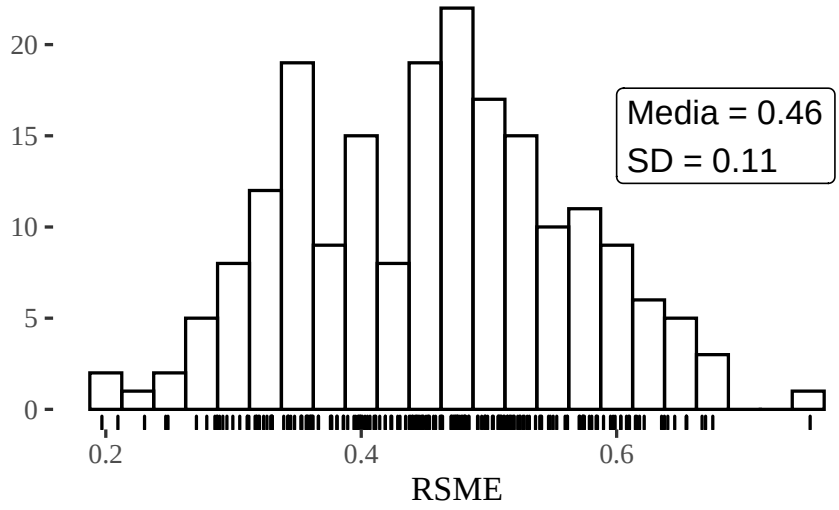
¿Cuál es la "mejor" combinación de hiperparámetros?

Table 1: Combinación de valores de lambda y orden que minimiza el RMSE de validación cruzada

lambda	orden	rmse
0.0000100	3	0.2564637
0.0000518	5	0.2610190
0.0372759	1	0.2673524
0.0000518	6	0.2745014
0.0000518	4	0.2750070

```
modelo <- datos[l == 1, regresion_poly(orden = mejor$orden, lambda = mejor$lambda)(t ~ x)]

datos[l != 1] %>%
  .[, pred := predict(modelo, newdata = x), by = l] %>%
  .[, .(error_medio = sqrt(mean((t - pred)^2))), by = l] %>%
  ggplot(aes(error_medio)) +
  geom_histogram(binwidth = 0.025, fill = NA, color = "black") +
  geom_label(data = ~.x[, .(mu = mean(error_medio), s = sd(error_medio))],
             aes(label = glue::glue("Media = {signif(mu, 2)}\nSD = {signif(s, 2)}")),
             x = 0.6, y = 15, hjust = 0) +
  scale_x_continuous("RSME") +
  scale_y_continuous(NULL) +
  geom_rug()
```



2 Apéndice

2.1 Paquetes y setup

```
library(ggplot2)
library(data.table)
library(magrittr)
library(patchwork)
theme_set(ggthemes::theme_tufte())
knitr::opts_chunk$set(tidy = FALSE, message = FALSE)

label_null <- as_labeller(function(x) "")

set.seed(42)
```

2.2 Definición de D

```
D <- function(n = 10, L = 1, intervalo = c(0, 1), FUN = ~sin(2*pi*.x), sigma = 0.3) {
  datos <- lapply(seq_len(L), function(l) {
    x <- runif(n, intervalo[1], intervalo[2])
    FUN <- purrr::as_mapper(FUN)
    real <- FUN(x)
    t <- real + rnorm(n, sd = sigma)
    return(data.table::data.table(x, t))
  })

  return(data.table::rbindlist(datos, idcol = "l"))
}
```

2.3 *Definición de regresion_poly y métodos*

```
regresion_poly <- function(orden = 1, lambda = 0) {
  force(orden)
  force(lambda)

  modelos <- data.table::CJ(orden, lambda)

  function(formula, data = NULL, weights) {
    datos <- model.frame(formula, data = data)
    y <- datos[, 1]
    x <- datos[, 2]
    Ws <- lapply(seq_len(nrow(modelos)), function(i) {
      orden <- modelos$orden[i]
      lambda <- modelos$lambda[i]
      # Matriz de diseño

      if (orden == 0) {
        A <- cbind(rep(1, length(x)))
      } else {
        A <- cbind(1, poly(x, degree = orden, raw = TRUE))
      }

      if (lambda != 0) {
        L <- diag(1, nrow = ncol(A)) * lambda
        w <- solve(t(A) %*% A + L) %*% t(A) %*% y   # Forma a lo bruto.
      } else {
        w <- qr.coef(qr(A), y)   # Forma eficiente de invertir la matriz
      }

      modelo <- list(orden = orden,
                     lambda = lambda,
                     w = w)
      return(modelo)
    })

    attr(Ws, "x") <- x
    class(Ws) <- c("regression_models", class(Ws))
    return(Ws)
  }
}

# Métodos para predecir nuevos valores usando la regresion.
predict.regression_models <- function(object, newdata = NULL, which = 1) {
  # browser()
```

```
  if (is.null(newdata)) {
    newdata <- attr(object, "x", exact = TRUE)
  }

  model <- object[[which]]

  if (model$orden == 0) {
    A <- cbind(rep(1, length(newdata)))
  } else {
    A <- cbind(1, poly(newdata, degree = model$orden, raw = TRUE))
  }
  return((A %*% model$w)[, 1])
}

predictdf.regression_models <- function(object, xseq, se, level) {
  fits <- lapply(seq_along(object), function(o) {
    y <- predict(object, newdata = xseq, which = o)
    return(data.frame(orden = object[[o]]$orden,
                      lambda = object[[o]]$lambda,
                      x = xseq,
                      y = y))
  })

  data <- do.call(rbind, fits)
  data$orden <- factor(data$orden, ordered = TRUE)
  return(data)
}
```

2.4 *Definición de rmse_cv*

```
rmse_cv <- function(formula, datos, fit_fun, k_fold = 10) {
  N <- nrow(datos)

  grupos <- ggplot2::cut_number(seq_len(N), k_fold)

  rmses <- vapply(seq_len(k_fold), function(k){
    train_index <- grupos != levels(grupos)[k]
    train <- datos[train_index == TRUE, ]
    validation <- datos[train_index == FALSE, ]
    model <- fit_fun(formula, data = train)
    validation[, sqrt(mean((t - predict(model, newdata = x))^2))]
  }, numeric(1))
  return(mean(rmses))
}
```