



INFERENCIA Y MODELOS ESTADÍSTICOS

Jacqueline Köhler C. y José Luis Jara V.



CAPÍTULO 15. REGRESIÓN LOGÍSTICA

En los capítulos 13 y 14 estudiamos la regresión lineal, útil para modelar cómo varía la media de una variable respuesta numérica Y a medida que cambian los valores de una o más variables predictoras $\mathbf{X} = (X_1, X_2, \dots, X_k)$, cuando esta relación condicional sigue una distribución normal con varianza constante. Pero estas ideas se pueden **generalizar** considerando otras distribuciones de probabilidad.

Nelder y Wedderburn (1972) desarrollaron las bases de esta idea para distribuciones condicionales de la familia exponencial, como la binomial, la de Poisson, la gama y otras. De hecho, la regresión lineal que estudiamos en los capítulos anteriores puede verse como un caso especial de estos modelos, ya que la distribución normal también pertenece a la familia de distribuciones exponenciales.

En la **regresión lineal generalizada** (RLG), la respuesta depende de una función lineal de los predictores que puede tomar cualquier valor en $(-\infty, +\infty)$, llamado **predictor lineal**, como muestra la ecuación 15.1.

$$\eta(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k \quad (15.1)$$

En la regresión lineal, la relación entre el valor esperado de la variable respuesta (es decir, su media) y el predictor lineal es directa:

$$\mathbb{E}[y | \mathbf{x}] \equiv \mu_{y|\mathbf{x}} = \eta(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k$$

Pero cuando la media condicional de la variable de salida tiene restricciones, como que solo pueda tomar valores positivos o en un intervalo determinado, es necesario usar una **función de enlace** invertible, que pueda convertir valores medios de la variable de salida desde y hacia la escala del predictor lineal, como se representa en la ecuación 15.2.

$$g(\mu_{y|\mathbf{x}}) = \eta(\mathbf{x}) \quad (15.2)$$

En el caso de la regresión lineal, es evidente que para el enlace se utiliza la función identidad: $I(\mu_{y|\mathbf{x}}) = \eta(\mathbf{x})$. Cambiando la función de enlace, entonces, podemos definir modelos de regresión que relacionan variables de salida que siguen diferentes distribuciones.

Los modelos de RLG se ajustan a los datos por el método de máxima verosimilitud, usando un algoritmo iterativo de **mínimos cuadrados ponderados**. Este procedimiento permite estimar los valores de los coeficientes de la regresión:

$$\hat{\eta}(\mathbf{x}) = b_0 + b_1 x_1 + b_2 x_2 + \dots + b_k x_k \quad (15.3)$$

y, de esta forma, una estimación del valor medio para la variable de salida:

$$\hat{\mu}_{y|\mathbf{x}} = g^{-1}[\hat{\eta}(\mathbf{x})] \quad (15.4)$$

15.1 LA RELACIÓN LOGÍSTICA

Después de la regresión lineal, el modelo de RLG más relevante es la **regresión logística** (RLog) que usa como función de enlace la función **logit**, dada por la ecuación 15.5, cuya inversa es la **función logística estandarizada**, dada por la ecuación 15.6. La figura 15.1 presenta gráficamente estas funciones.

$$\text{logit}(p) = \ln \left(\frac{p}{1-p} \right) \quad \text{con } p \in (0, 1) \quad (15.5)$$

$$\text{logística}(z) = \text{logit}^{-1}(z) = \frac{1}{1 + e^{-z}} \quad \text{con } z \in (-\infty, +\infty) \quad (15.6)$$

Si nos fijamos en la gráfica de la función logística, podemos observar que el eje vertical describe una **transición de cero a uno** asociada al aumento de los valores en el eje horizontal. Esto resulta útil para representar **cómo cambia la probabilidad** de que algún evento ocurra en función del valor de una variable continua.

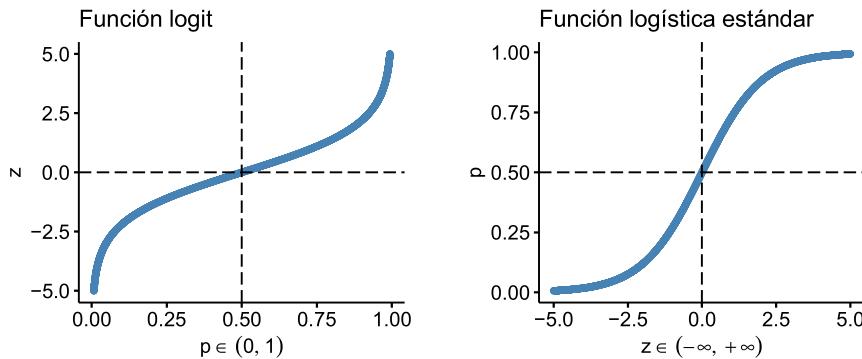


Figura 15.1: la función logit (izquierda) y, su inversa, la función logística (derecha).

De este modo, la regresión logística puede ser usada para modelar variables respuesta **binarias** que puedan tomar los valores 0 o 1, cuya media, en consecuencia, se encuentra en el intervalo $[0, 1]$ y que corresponde a la probabilidad de éxito: $\Pr(y = 1)$. Pero ahora, además, podemos mejorar la estimación de esta probabilidad considerando el valor observado para un conjunto de variables predictoras: $\Pr(y = 1 | \mathbf{x})$. La relación entre la media de esta probabilidad condicional ($\mu_{\Pr(y = 1 | \mathbf{x})}$) y los predictores queda definida por la ecuaciones:

$$\ln \left(\frac{\mu_{\Pr(y = 1 | \mathbf{x})}}{1 - \mu_{\Pr(y = 1 | \mathbf{x})}} \right) = \eta(\mathbf{x}) \quad (15.7)$$

$$\mu_{\Pr(y = 1 | \mathbf{x})} = \frac{1}{1 + e^{-\eta(\mathbf{x})}} \quad (15.8)$$

Si seleccionamos una división o **umbral** de probabilidad, podemos **predecir** la ocurrencia de un evento o, de forma equivalente, **clasificar** un caso en dos categorías posibles:

- Para valores menores que el umbral se predice que el evento “no ocurre”, otorgándose una clasificación cero ($\hat{y} = 0$) o negativa ($\hat{y} = -$).
- Mientras que para valores mayores o iguales que el umbral se predice que el evento “sí ocurre”, a lo que corresponde una clasificación uno ($\hat{y} = 1$) o positiva ($\hat{y} = +$).

Si bien es usual utilizar el valor $\mu_{\Pr(y = 1 | \mathbf{x})} = 0.5$ como umbral, esto no es obligatorio ni siempre conveniente, como veremos más adelante. De esta forma, un modelo de RLog puede utilizarse como un **clasificador**.

15.2 EVALUACIÓN DE UN CLASIFICADOR

Una forma de evaluar modelos de clasificación, entre ellos los de RLog, es de acuerdo a la cantidad de errores cometidos (Tharwat, 2021). Para ello, el primer paso consiste en construir una tabla de contingencia (también llamada **matriz de confusión**) para las respuestas predichas y observadas, como muestra la tabla 15.1, bastante similar a la que ya conocimos para explicar los errores de decisión en la prueba de hipótesis (tabla 4.1). Las cuatro celdas de la matriz de confusión contienen:

- **Verdaderos positivos (VP)**: cantidad de instancias correctamente clasificadas como pertenecientes a la clase positiva.
- **Falsos positivos (FP)**: cantidad de instancias erróneamente clasificadas como pertenecientes a la clase positiva.
- **Falsos negativos (FN)**: cantidad de instancias erróneamente clasificadas como pertenecientes a la clase negativa.
- **Verdaderos negativos (VN)**: cantidad de instancias correctamente clasificadas como pertenecientes a la clase negativa.

		Real		Total
		1 (+)		
Clasificación	1 (+)	VP	FP	VP + FP
	0 (-)	FN	VN	FN + VN
	Total	VP + FN	FP + VN	n

Tabla 15.1: tabla de contingencia para evaluar un clasificador.

La **exactitud** (*accuracy*) del clasificador corresponde a la proporción de observaciones correctamente clasificadas, dada por la ecuación 15.9.

$$\text{exactitud} = \frac{VP + VN}{n} \quad (15.9)$$

A su vez, el **error** del clasificador corresponde a la proporción de observaciones clasificadas de manera equivocada (ecuación 15.10).

$$\text{error} = \frac{FP + FN}{n} = 1 - \text{exactitud} \quad (15.10)$$

La **sensibilidad** (*sensitivity* o *recall*, ecuación 15.11) indica cuán apto es el clasificador para detectar aquellas observaciones pertenecientes a la clase positiva.

$$\text{sensibilidad} = \frac{VP}{VP + FN} \quad (15.11)$$

De manera análoga, la **especificidad** (*specificity*, ecuación 15.12) permite evaluar la aptitud del clasificador para asignar observaciones correctamente a la clase negativa.

$$\text{especificidad} = \frac{VN}{FP + VN} \quad (15.12)$$

La **precisión** (*precision*) o valor predictivo positivo (*VPP*, ecuación 15.13) indica cuán exacta es la asignación de elementos a la clase positiva (la proporción de instancias clasificadas como positivas que realmente lo son).

$$VPP = \frac{VP}{VP + FP} \quad (15.13)$$

Asimismo, el **valor predictivo negativo** (*VPN*, ecuación 15.14) señala la proporción de instancias correctamente clasificadas como pertenecientes a la clase negativa.

$$VPN = \frac{VN}{FN + VN} \quad (15.14)$$

Otra herramienta útil es la **curva de calibración**, también llamada curva ROC por las siglas inglesas para *receiver-operating characteristic*, que muestra la relación entre la sensibilidad y la especificidad del modelo (Fawcett, 2006). Este gráfico permite evaluar visualmente la calidad del clasificador, puesto que mientras más se aleje la curva de la diagonal, mejor es su clasificación. Para ilustrar mejor la utilidad de este gráfico, la figura 15.2 muestra las curvas ROC para dos modelos diferentes, además de la diagonal (línea recta). Esta figura indica que el clasificador representado por la curva más oscura es mejor que el representado por la curva más clara, pues se aleja más de la diagonal.

Este “alejamiento” de la diagonal, a veces es representado numéricamente por el **área total bajo la curva ROC**, llamado AUC por sus inglesas para *area under the curve*, cuyo valor varía entre 0 y 1. Un AUC más alto indica un mejor desempeño del modelo en la clasificación. Un clasificador perfecto que asigna correctamente la clase a todas las instancias exhibe un AUC = 1, mientras que uno que no discrimina, es decir, su desempeño no es mejor que el de una clasificación aleatoria, se asocia a un AUC = 0,5.

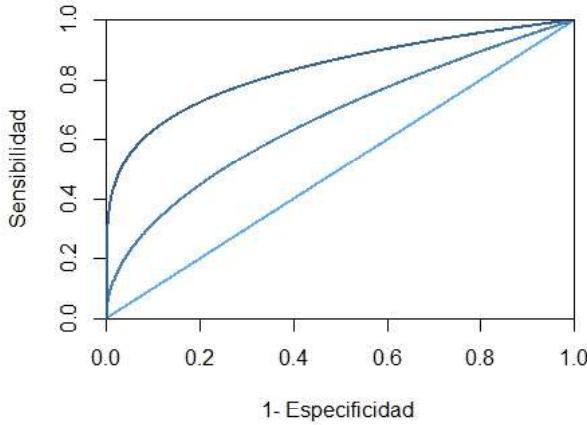


Figura 15.2: dos curvas ROC. Adaptado de Ayala (2020).

15.3 AJUSTE DE UN MODELO DE RLOG

Como se dijo anteriormente, el ajuste de un modelo de RLG se realiza mediante la resolución de un problema de optimización que busca minimizar las desviaciones entre las respuestas predichas y las observadas usando la función de verosimilitud. Si se tiene una muestra $D = \{(\mathbf{x}_j, y_j)\}$ con d observaciones, donde $j = \{1, 2, \dots, d\}$, $y_j \in \{0, 1\}$ y $\mathbf{x}_j = (x_{j1}, x_{j2}, \dots, x_{jn})$ son los valores observados de las variables predictoras en conjunto a y_j , la función de verosimilitud a optimizar en búsqueda del vector de parámetros $B = (\beta_0, \beta_1, \dots, \beta_n)$ es:

$$\begin{aligned}\mathcal{L}(B; D) &= \Pr(y_1, y_2, \dots, y_d | X_1, X_2, \dots, X_d; B) \\ &= \prod_{j=1}^d \left[\Pr(y_j = 1 | \eta_B(\mathbf{x}_j))^{y_j} [1 - \Pr(y_j = 1 | \eta_B(\mathbf{x}_j))]^{1-y_j} \right] \\ &= \prod_{j=1}^d \left[\text{logística}(\eta_B(\mathbf{x}_j))^{y_j} [1 - \text{logística}(\eta_B(\mathbf{x}_j))]^{1-y_j} \right]\end{aligned}$$

Por conveniencia numérica, se suele optimizar el logaritmo natural de la verosimilitud:

$$\ln \mathcal{L}(B; D) = \sum_{j=1}^d \left[y_j \ln \text{logística}(\eta_B(\mathbf{x}_j)) + (1 - y_j) \ln [1 - \text{logística}(\eta_B(\mathbf{x}_j))] \right] \quad (15.15)$$

Al igual que en el caso de la regresión lineal, existen diversos mecanismos para evaluar el ajuste de un modelo de regresión logística. El estadístico de **log-verosimilitud** ($\ln \mathcal{L}(B; D)$), dado por la ecuación 15.15, nos permite cuantificar la diferencia entre las probabilidades predichas y las observadas. Este estadístico se asemeja a la suma de los residuos cuadrados de la regresión lineal en el sentido de que cuantifica la cantidad de información que carece de explicación tras el ajuste del modelo. Así, mientras menor sea su valor, mejor es el ajuste del modelo.

Sin embargo, el estadístico **desviación** (*deviance* en inglés), a menudo denotada por $-2LL$ y en ocasiones traducida como *devianza*, suele usarse en lugar de la log-verosimilitud, y que se obtiene de forma directa a partir de esta última, como indica la ecuación 15.16.

$$-2LL = -2 \ln \mathcal{L}(B; D) \quad (15.16)$$

Para comparar dos modelos de RLog M_1 y M_2 cuando el segundo es una extensión del primero, es decir M_2 contiene todos los predictores de M_1 más otros k predictores, se puede usar la diferencia de sus desviaciones,

$(-2LL_1) - (-2LL_2)$, que **sigue asintóticamente** (a medida que el tamaño de la muestra crece al infinito) una distribución χ^2 con k grados de libertad. Esto permite calcular el nivel de significación de esta diferencia, que es usada en la prueba conocida como **Likelihood Ratio Test (LRT)**.

Por otro lado, los criterios de evaluación de modelos basados en el principio de parsimonia, que estudiamos en el capítulo 14, también están definidos para la regresión logística. El más sencillo es el criterio de información de Akaike (AIC), dado por la ecuación 15.17, donde k corresponde a la cantidad de predictores en el modelo.

$$\text{AIC} = -2LL + 2k \quad (15.17)$$

Similar al AIC, el criterio bayesiano de Schwarz (BIC) ajusta la penalización a la complejidad del modelo según el tamaño de la muestra, como se ve en la ecuación 15.18.

$$\text{BIC} = -2LL + 2k \ln n \quad (15.18)$$

15.4 REGRESIÓN LOGÍSTICA EN R

En R, la llamada `glm(formula, family = binomial(link = "logit"), data)` permite ajustar un modelo de regresión logística, donde:

- `formula` tiene la forma `<variable_respuesta> ~ <variable_predictora>`.
- `data`: matriz de datos.

El argumento `family = binomial(link = "logit")` indica que asumiremos una distribución binomial para la variable de respuesta y que usaremos la función `logit` como enlace.

Siguiendo el ejemplo de los capítulos anteriores con el conjunto de datos `mtcars` (descrito en la tabla 13.1), el script 15.1 muestra la construcción de un modelo de RLog que predice el tipo de transmisión de un automóvil a partir de su peso.

La variable “tipo de transmisión” (`am`) es dicotómica, por lo que es ideal para este ejemplo. Si bien en el conjunto de datos viene etiquetada con los valores numéricos 0 para automóviles automáticos y 1 para los mecánicos, el script se encarga de convertirla en un factor con etiquetas más intuitivas (líneas 8–9).

Script 15.1: ejemplo de la construcción de un modelo de regresión logística en R.

```

1 library(caret)
2 library(dplyr)
3 library(ggpubr)
4 library(pROC)
5
6 # Cargar y filtrar los datos, teniendo cuidado de dejar "automático" como el
7 # 2do nivel de la variable "am" para que sea considerada como la clase positiva.
8 datos <- mtcars |> filter(wt > 2 & wt < 5) |>
9   mutate(am = factor(am, levels = c(1, 0), labels = c("manual", "automático")))
10
11 # Separar conjuntos de entrenamiento y prueba, fijando una semilla para asegurar
12 # la reproducibilidad de los resultados.
13 set.seed(101)
14 n <- nrow(datos)
15 i_muestra <- sample.int(n = n, size = floor(0.7 * n), replace = FALSE)
16 datos_ent <- datos[i_muestra, ]
17 datos_pru <- datos[-i_muestra, ]
18
19 # Ajustar y mostrar modelo
20 modelo <- glm(am ~ wt, family = binomial(link = "logit"), data = datos_ent)
21 print(summary(modelo), signif.stars = FALSE)
22 cat("---\n")
23 print(anova(modelo, test = "LRT"), signif.stars = FALSE)

```

```

Call:
glm(formula = am ~ wt, family = binomial(link = "logit"), data = datos_ent)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -18.711     8.790  -2.129  0.0333
wt           6.003      2.741   2.191  0.0285

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 23.0348 on 16 degrees of freedom
Residual deviance: 9.9658 on 15 degrees of freedom
AIC: 13.966

Number of Fisher Scoring iterations: 6

---
Analysis of Deviance Table

Model: binomial, link: logit

Response: am

Terms added sequentially (first to last)

          Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
NULL          16    23.0348
wt           1     13.069   15    9.9658 0.0003002

```

Figura 15.3: ajuste de un modelo de regresión logística.

Luego, siguiendo lo aprendido en los capítulos anteriores sobre construcción de modelos, se selecciona aleatoriamente un conjunto de entrenamiento, para construir el modelo, con el 70 % de las observaciones disponibles, dejando el restante 30 % para la validación (líneas 13–17). A continuación, las líneas 20–21 del script muestran el uso de la función `glm()` para ajustar y mostrar en pantalla, respectivamente, el modelo deseado. Finalmente, la línea 23 muestra cómo comparar el modelo obtenido con el modelo nulo para determinar si es significativamente mejor aplicando la prueba LRT.

El resultado se muestra en la figura 15.3, donde podemos apreciar, en el resumen del modelo, que el peso del vehículo (`wt`) aporta significativamente al modelo ($z = 2,191$; $p = 0,029$) si consideramos un nivel de significación $\alpha = 0,05$, que la desviación para este modelo con un predictor (15 grados de libertad) se redujo a 9,966 del valor 23,035 exhibido por el modelo nulo, y que su valor para el criterio de información de Akaike es $AIC = 13,966$. En la segunda parte, vemos que la reducción en la desviación conseguida (13,069) es significativa ($\chi^2(15) = 13,069$; $p < 0,001$). Podemos concluir, entonces, que el modelo con un predictor presenta un mejor ajuste que el modelo nulo.

El script 15.2 muestra código R que permite evaluar la calidad predictiva del modelo ajustado en el conjunto de entrenamiento. La línea 31 muestra el uso de la función `roc(response, predictor, levels, direction)` del paquete `ROCR`, donde:

- `response`: factor con la clase observada (real o de referencia) para cada caso a considerar.
- `predictor`: vector con la puntuación que utiliza el modelo para predecir la clase de cada caso.
- `levels`: vector con dos strings que indican, respectivamente, la clase negativa y positiva en `response`.
- `direction`: indica si una mayor probabilidad de éxito (la clase positiva definida anteriormente) está asociada a valores más altos (`direction = ">"`) o más bajos (`direction = "<"`) del predictor .

La función `roc()` devuelve un objeto que representa la ROC que se genera al ir cambiando el umbral del valor del `predictor` usado para asignar la clase positiva a un caso. Existen otros argumentos optativos que permiten suavizar la curva, cambiar las escalas, entre otras cosas, que pueden revisarse en la documentación de R.

Script 15.2: (continuación del script 15.1) evaluación del modelo de RLog en los datos de entrenamiento.

```

25 #
26 # Evaluar el modelo con el conjunto de entrenamiento
27 #
28 probs_ent <- fitted(modelo)
29
30 # Graficar curva ROC, indicando AUC obtenido.
31 ROC_ent <- roc(datos_ent[["am"]], probs_ent,
32                   levels = c("manual", "automático"), direction = "<")
33 g_ROC_ent <- ggroc(ROC_ent, color = "steelblue")
34 g_ROC_ent <- g_ROC_ent + geom_abline(intercept = 1, slope = 1,
35                                         colour = "steelblue1", linetype = "dashed")
36 g_ROC_ent <- g_ROC_ent + xlab("Especificidad") + ylab("Sensibilidad")
37 texto_ent <- sprintf("AUC = %.2f", ROC_ent[["auc"]])
38 g_ROC_ent <- g_ROC_ent + annotate("text", x = 0.3, y = 0.3, label = texto_ent)
39 g_ROC_ent <- g_ROC_ent + theme_pubr()
40 print(g_ROC_ent)
41
42 # Obtener las predicciones.
43 umbral <- 0.5
44 preds_ent <- sapply(probs_ent,
45                      function(p) ifelse(p >= umbral, "automático", "manual"))
46 preds_ent <- factor(preds_ent, levels = c("manual", "automático"))
47
48 # Obtener y mostrar estadísticas de clasificación en datos de entrenamiento.
49 mat_conf_ent <- confusionMatrix(preds_ent, datos_ent[["am"]],
50                                   positive = "automático")
51 cat("\n\nEvaluación de la calidad predictora (cjto. de entrenamiento):\n")
52 cat("-----\n")
53 print(mat_conf_ent[["table"]])
54 cat("\n")
55 cat(sprintf("    Exactitud: %.3f\n", mat_conf_ent[["overall"]][["Accuracy"]]))
56 cat(sprintf(" Sensibilidad: %.3f\n", mat_conf_ent[["byClass"]][["Sensitivity"]]))
57 cat(sprintf("Especificidad: %.3f\n", mat_conf_ent[["byClass"]][["Specificity"]]))

```

En la llamada a la función `roc()` utilizamos los valores ajustados por el modelo, recuperados en la línea 28. Recordemos que estos corresponden a las probabilidades de éxito para cada caso (ecuación 15.8). Las líneas 33–40 permiten obtener (y adornar) el gráfico de la curva, la que es mostrada al lado izquierdo de la figura 15.4. Podemos notar que esta se aleja bastante de la diagonal, por lo que al parecer se trata de un buen clasificador.

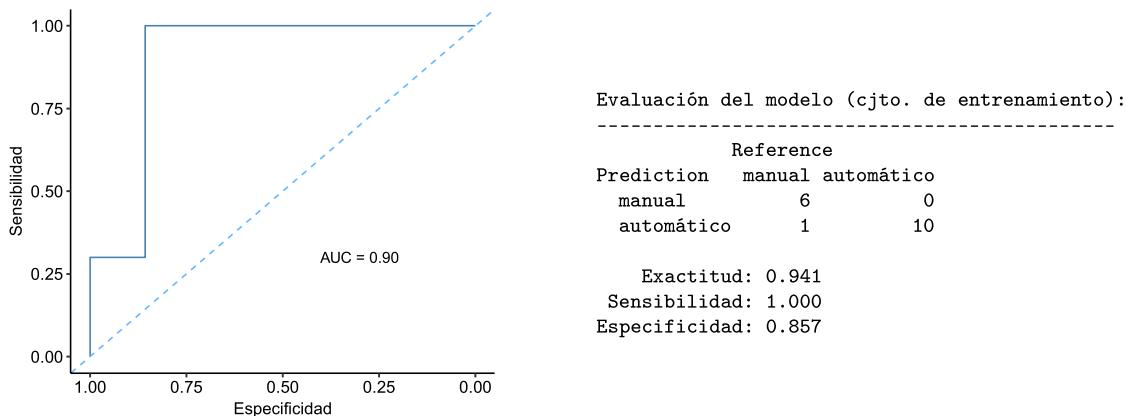


Figura 15.4: evaluación del clasificador en los datos usados para construir el modelo.

En las líneas 43–46 del script se clasifica como un vehículo automático a todos los casos de entrenamiento cuya probabilidad de pertenecer a esta categoría sea igual o superior al tradicional umbral del 50 %. Con esta clasificación realizada, se puede obtener una matriz de confusión y calcular las métricas de evaluación del modelo. Esta tarea se facilita enormemente si se usa la función `confusionMatrix(data, reference, positive)` del paquete `caret`, donde `data` corresponde a las clases predichas, `reference` a las clases observadas y `positive` al nombre de la clase positiva. Las líneas 49–50 del script 15.2 utiliza esta función con la clasificación obtenida anteriormente, y las líneas 51–57 despliegan en pantalla algunas de las métricas de interés. El resultado de estas últimas se muestra en el lado derecho de la figura 15.4, donde podemos ver que el modelo tiene una exactitud de 94,1 %. La sensibilidad de 100 % y la especificidad de 85,70 % muestran que el modelo se desempeña un poco mejor identificando elementos de la clase positiva, correspondiente en este caso a los vehículos de transmisión automática.

Pero, como ya hemos estudiado en capítulos anteriores, esta evaluación puede estar sesgada y debemos evaluar el modelo con un conjunto de datos diferente al que usamos para su construcción. El script 15.3 realiza esta evaluación. En la línea 62, al indicar `type = "response"` en la función `predict()`, se obtienen las predicciones del modelo de la probabilidad de pertenecer a la clase positiva para cada caso de prueba. Luego, las líneas 65–74 obtienen y despliegan la ROC asociada a estas probabilidades. A continuación, las líneas 77–79 generan la clasificación de estos datos en las categorías de la variable de salida, la que es evaluada en las líneas 82–90.

El resultado del script puede verse en la figura 15.5, donde observamos una caída importante de la exactitud y la especificidad con respecto al rendimiento visto con el conjunto de entrenamiento. Esto podría ser una indicación de sobreajuste del modelo al conjunto de entrenamiento, pero también podría ser una señal de que el conjunto de prueba puede ser muy pequeño para obtener una evaluación confiable. Esta última idea gana fuerza si usamos el AUC como criterio de calidad, puesto que el valor con los datos de entrenamiento (0,90) es menor al obtenido con los datos de prueba (0,92).

Script 15.3: (continuación del script 15.2) evaluación del modelo de RLog en los datos de prueba.

```

59 #
60 # Evaluar el modelo con el conjunto de prueba.
61 #
62 probs_pru <- predict(modelo, datos_pru, type = "response")
63
64 # Graficar curva ROC, indicando AUC obtenido.
65 ROC_pru <- roc(datos_pru[["am"]], probs_pru,
66                   levels = c("manual", "automático"), direction = "<")
67 g_ROC_pru <- ggroc(ROC_pru, color = "steelblue")
68 g_ROC_pru <- g_ROC_pru + geom_abline(intercept = 1, slope = 1,
69                                         colour = "steelblue1", linetype = "dashed")
70 g_ROC_pru <- g_ROC_pru + xlab("Especificidad") + ylab("Sensibilidad")
71 texto_pru <- sprintf("AUC = %.2f", ROC_pru[["auc"]])
72 g_ROC_pru <- g_ROC_pru + annotate("text", x = 0.3, y = 0.3, label = texto_pru)
73 g_ROC_pru <- g_ROC_pru + theme_pubr()
74 print(g_ROC_pru)
75
76 # Obtener las predicciones (con el mismo umbral).
77 preds_pru <- sapply(probs_pru,
78                      function(p) ifelse(p >= umbral, "automático", "manual"))
79 preds_pru <- factor(preds_pru, levels = c("manual", "automático"))
80
81 # Obtener y mostrar estadísticas de clasificación en datos de prueba.
82 mat_conf_pru <- confusionMatrix(preds_pru, datos_pru[["am"]],
83                                   positive = "automático")
84 cat("\n\nEvaluación del modelo (cjto. de prueba):\n")
85 cat("-----\n")
86 print(mat_conf_pru[["table"]])
87 cat("\n")
88 cat(sprintf("    Exactitud: %.3f\n", mat_conf_pru[["overall"]][["Accuracy"]]))
89 cat(sprintf(" Sensibilidad: %.3f\n", mat_conf_pru[["byClass"]][["Sensitivity"]]))
90 cat(sprintf(" Especificidad: %.3f\n", mat_conf_pru[["byClass"]][["Specificity"]]))

```

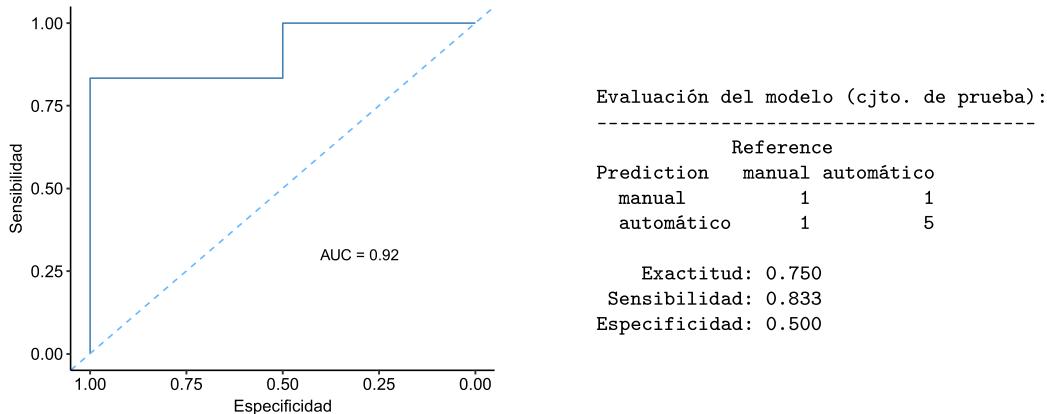


Figura 15.5: evaluación del clasificador con datos de prueba, no utilizados al construir el modelo.

En capítulos anteriores conocimos la validación cruzada como herramienta para conseguir una estimación más confiable y robusta del rendimiento de un modelo, la cual podemos usar de manera análoga para regresión logística.

El script 15.4 mejora la evaluación realizada en el script 15.3 incorporando el uso de validación cruzada de 4 pliegues (líneas 12–16). Notemos que la llamada a la función `train()` también solicita que “se guarden” los valores predichos, lo que nos permite calcular el rendimiento promedio del modelo, como si se repitiera la evaluación del script 15.3, seleccionando aleatoriamente un conjunto de entrenamiento y otro de prueba, cuatro veces. Los coeficientes del modelo que se consigue con este procedimiento se muestran en pantalla en las líneas 19–22 del script.

Las líneas 25–26 del script recuperan las predicciones hechas en cada pliegue. Luego se obtienen las respectivas matrices de confusión y métricas de evaluación para cada uno, utilizando la función `confusionMatrix()`, en las líneas 29–30. Como esta función devuelve muchas métricas, las líneas 33–36 extraen las que nos interesan en este caso: exactitud, sensibilidad y especificidad. Estas se ordenan en una matriz de datos, y se agregan las medias y desviaciones estándar de cada métrica en las líneas 39–42. En las líneas 45–47, esta tabla con valores numéricos se lleva a una tabla de strings para su presentación en pantalla, lograda en las líneas 51–58.

El resultado del script 15.4 puede verse en la figura 15.6. Debemos fijarnos en que el modelo obtenido, es idéntico al anterior (script 15.1). Esto se debe a que la función `train()` **recalcula** el modelo del pliegue que obtuvo mejor rendimiento con **todos los datos disponibles**. En el caso de la RLog (como con la lineal), los pliegues solo se diferencian en los datos que utilizan, **manteniendo** los predictores, por lo que siempre se llega al mismo modelo. Esto no sería así si la validación cruzada se usara, por ejemplo, para seleccionar las variables predictoras a incluir en el modelo, variando el conjunto de predictores usado en cada pliegue.

Coeficientes del modelo final:					
	Estimate	Std. Error	z value	Pr(> z)	
(Intercept)	-11.752	4.682	-2.510	0.01207	
wt	3.935	1.487	2.646	0.00815	

Detalle por pliegue:				
Exactitud	Sensibilidad	Especificidad	Pliegue	
1.000	1.000	1.000	Fold1	
0.857	0.667	1.000	Fold2	
0.667	0.500	0.750	Fold3	
1.000	1.000	1.000	Fold4	
<hr/>				
0.881	0.792	0.938	Media	
0.158	0.250	0.125	D.E.	

Figura 15.6: evaluación de un modelo de RLog usando validación cruzada.

Script 15.4: ajuste y evaluación de un modelo de regresión logística usando validación cruzada.

```

1 library(caret)
2 library(dplyr)
3 library(purrr)
4
5 # Cargar y filtrar los datos, teniendo cuidado de dejar "automático" como el
6 # 2do nivel de la variable "am" para que sea considerada como la clase positiva.
7 datos <- mtcars |> filter(wt > 2 & wt < 5) |>
8   mutate(am = factor(am, levels = c(1, 0), labels = c("manual", "automático")))
9
10 # Ajustar modelo usando validación cruzada de 4 pliegues, asegurando que
11 # se guardan las predicciones de cada pliegue.
12 set.seed(113)
13 modelo_ent <- train(am ~ wt, data = datos, method = "glm",
14   family = binomial(link = "logit"),
15   trControl = trainControl(method = "cv", number = 4,
16                           savePredictions = TRUE))
17
18 # Mostrar los coeficientes del modelo obtenido
19 modelo_final <- modelo_ent[["finalModel"]]
20 modelo_final_str <- capture.output(print(summary(modelo_final), signif.stars = FALSE))
21 cat("Coeficientes del modelo final:\n")
22 write.table(modelo_final_str[6:9], quote = FALSE, row.names = FALSE, col.names = FALSE)
23
24 # Obtener las predicciones por pliegue
25 preds <- modelo_ent[["pred"]] |> mutate(pliegue = factor(Resample)) |>
26   select(pred, obs, pliegue)
27
28 # Construir las matrices de confusión de cada pliegue
29 conf_mat_list <- preds |> group_split(pliegue) |>
30   map(~ confusionMatrix(.x[["pred"]], .x[["obs"]]))
31
32 # Extraer las métricas de evaluación de interés
33 metricas_tab <- conf_mat_list |>
34   map_df(~ data.frame(Exactitud = .$overall["Accuracy"],
35     Sensibilidad = .\$byClass["Sensitivity"],
36     Especificidad = .\$byClass["Specificity"]))
37
38 # Agregar el pliegue, los promedios y las desviaciones estándar
39 metricas_tab <- cbind(metricas_tab, Pliegue = levels(preds[["pliegue"]]))
40 medias_tab <- data.frame(t(apply(metricas_tab[, -4], 2, mean)), Pliegue = "Media")
41 desv_tab <- data.frame(t(apply(metricas_tab[, -4], 2, sd)), Pliegue = "D.E. ")
42 metricas_tab <- rbind(metricas_tab, medias_tab, desv_tab)
43
44 # Formatear las columnas
45 formatea_col <- function(cn) format(metricas_tab[[cn]], digits = 3,
46                                       width = nchar(cn), justify = "right")
47 metricas_str_tab <- sapply(colnames(metricas_tab), formatea_col,
48                           USE.NAMES = FALSE, simplify = TRUE)
49
50 # Mostrar las métricas obtenidas
51 encab <- paste(colnames(metricas_tab), collapse = " ")
52 cat("Detalle por pliegue:\n", encab, "\n")
53 cat(strrep("-", nchar(encab)), "\n")
54 write.table(metricas_str_tab[1:4, ], sep = " ", 
55             row.names = FALSE, col.names = FALSE, quote = FALSE)
56 cat(strrep("-", nchar(encab)), "\n")
57 write.table(metricas_str_tab[5:6, ], sep = " ", 
58             row.names = FALSE, col.names = FALSE, quote = FALSE)

```

Analizando los resultados, se ve que en dos de los pliegues se obtuvo clasificación perfecta, mientras que los otros el rendimiento fue más parecido a lo observado en el modelo anterior (figura 15.5). Pero en promedio, repitiendo el experimento cuatro veces, el rendimiento del modelo es mayor que lo estimado anteriormente, por lo que con una única mirada lo estabamos subestimando.

15.5 MÚLTIPLES PREDICTORES

Cuando tenemos múltiples predictores potenciales, debemos decidir cuáles de ellos incorporar en el modelo de RLog. Una vez más, y tal como detallamos en el capítulo 14, el ideal es usar la regresión jerárquica para escoger los predictores de acuerdo a evidencia disponible en la literatura.

Sin embargo, si la intención es explorar los datos, podemos emplear los demás métodos ya descritos: selección hacia adelante, eliminación hacia atrás, regresión escalonada o todos los subconjuntos, usando para ello las mismas funciones de R descritas en el capítulo 14.

En páginas previas ajustamos un modelo de RLog para determinar el tipo de transmisión de un automóvil a partir de su peso. Sin embargo, el predictor fue seleccionado de manera arbitraria, simplemente para ilustrar el proceso, por lo que podríamos encontrar un mejor modelo usando algún método de selección de predictores.

Las líneas 22–23 del script 15.5 llevan a cabo esta tarea usando regresión escalonada, que busca golosamente el mejor modelo entre uno sin ningún predictor (el modelo nulo) hasta uno que considera todos los posibles predictores (el modelo completo). El resultado del script es presentado en la figura 15.7. En ella podemos ver que, además del peso del automóvil (`wt`), el mejor modelo encontrado considera además la potencia del motor (`hp`) como predictor.

Sin embargo, debemos notar que aparecen varios mensajes de advertencia indicando que se encontraron **problemas de convergencia** al realizar la búsqueda de este modelo. Es más, si nos fijamos en la evaluación de las estimaciones de los coeficientes, vemos que estos tienen *estadísticos gigantes* y valores *p* igual a 1! Una clara indicación de que el modelo tiene problemas.

```
Modelo RLog conseguido con regresión escalonada:
-----
Call:
glm(formula = am ~ wt + hp, family = binomial(link = "logit"),
     data = datos_ent)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -3.730e+02  6.464e+05  -0.001      1
wt           1.409e+02  2.437e+05   0.001      1
hp          -4.994e-01  1.960e+03   0.000      1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 2.3035e+01 on 16 degrees of freedom
Residual deviance: 4.6315e-10 on 14 degrees of freedom
AIC: 6

Number of Fisher Scoring iterations: 25

Warning messages:
1: glm.fit: fitted probabilities numerically 0 or 1 occurred
2: glm.fit: fitted probabilities numerically 0 or 1 occurred
3: glm.fit: algorithm did not converge
4: glm.fit: fitted probabilities numerically 0 or 1 occurred
5: glm.fit: fitted probabilities numerically 0 or 1 occurred
6: glm.fit: fitted probabilities numerically 0 or 1 occurred
7: glm.fit: fitted probabilities numerically 0 or 1 occurred
8: glm.fit: fitted probabilities numerically 0 or 1 occurred
9: glm.fit: fitted probabilities numerically 0 or 1 occurred
10: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

Figura 15.7: salida obtenida al buscar un modelo de RLog con regresión escalonada.

Script 15.5: búsqueda de un modelo de RLog usando regresión escalonada.

```

1 library(ggpubr)
2 library(dplyr)
3
4 # Cargar y filtrar los datos (solo predictores numéricicos)
5 datos <- mtcars |> filter(wt > 2 & wt < 5) |>
6   select(-c("cyl", "vs", "gear", "carb")) |>
7   mutate(am = factor(am, levels = c(1, 0), labels = c("manual", "automático")))
8
9 # Separar los conjuntos de entrenamiento y prueba
10 set.seed(101)
11 n <- nrow(datos)
12 i_muestra <- sample.int(n = n, size = floor(0.7 * n), replace = FALSE)
13 datos_ent <- datos[i_muestra, ]
14 datos_pru <- datos[-i_muestra, ]
15
16 # Construir los Modelos nulo y completo
17 nulo <- glm(am ~ 1, family = binomial(link = "logit"), data = datos_ent)
18 comp <- glm(am ~ ., family = binomial(link = "logit"), data = datos_ent)
19
20 # Ajustar y mostrar un modelo con regresión paso a paso escalonada
21 modelo <- step(nulo, scope = list(upper = comp),
22                 direction = "both", trace = FALSE)
23
24 cat("Modelo RLog conseguido con regresión escalonada:\n")
25 cat("-----\n")
26 print(summary(modelo))

```

En un comienzo no es fácil determinar la raíz de los problemas. Si se grafican cómo serían los modelos de RLog asociados a cada predictor suponiendo constante el otro, obtenemos los gráficos (a) y (b) mostrados en la figura 15.8. Nada extraño es visible en ellos.

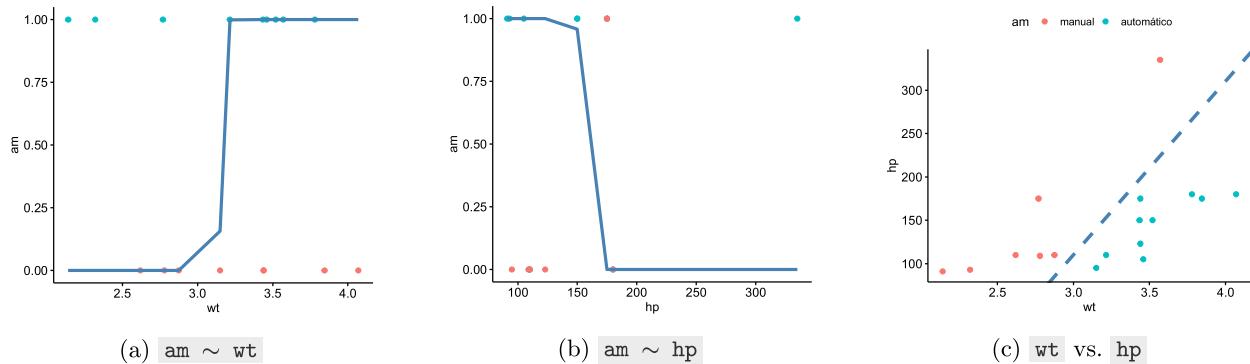


Figura 15.8: gráficos de las predicciones parciales de los predictores peso (a) y potencia (b) y una visión de las clases en el gráfico de dispersión formado por estas variables (c).

Sin embargo, cuando vemos cómo interactúan estas variables en el gráfico (c), podemos darnos cuenta que una simple recta puede hacer una **separación perfecta** de los vehículos con transmisión manual de los que tienen transmisión automática. Es decir, ¡no necesitamos un clasificador logístico! El algoritmo de optimización iterativo descrito en la sección 15.3 no puede converger a coeficientes estables este caso.

El script 15.6 muestra una regresión paso a paso hacia adelante, de manera que podamos ir evitando caer en este problema de separación perfecta. Primero se construyen los modelos nulo y completo para ser usados, respectivamente, como modelo inicial y límite superior de la búsqueda. Notemos que el ajuste del modelo completo genera la primera advertencia de problemas de convergencia, como se aprecia en la figura 15.9 muestra la salida que genera esta búsqueda en pantalla.

Script 15.6: búsqueda de un modelo de RLog usando regresión paso a paso hacia adelante.

```

1 library(ggpubr)
2 library(dplyr)
3
4 # Imprimir mensajes de advertencia a medida que ocurren, más cortos
5 opt <- options(warn = 1, width = 26)
6
7 # Cargar y filtrar los datos (solo predictores numéricicos)
8 datos <- mtcars |> filter(wt > 2 & wt < 5) |>
9   select(-c("cyl", "vs", "gear", "carb")) |>
10  mutate(am = factor(am, levels = c(1, 0), labels = c("manual", "automático")))
11
12 # Separar conjuntos de entrenamiento y prueba
13 set.seed(101)
14 n <- nrow(datos)
15 i_muestra <- sample.int(n = n, size = floor(0.7 * n), replace = FALSE)
16 datos_ent <- datos[i_muestra, ]
17 datos_pru <- datos[-i_muestra, ]
18
19 # Definir modelos inicial y máximo
20 nulo <- glm(am ~ 1, family = binomial(link = "logit"), data = datos_ent)
21 maxi <- glm(am ~ ., family = binomial(link = "logit"), data = datos_ent)
22
23 # Revisar un paso hacia adelante
24 cat("\nPaso 1:\n-----\n")
25 print(add1(nulo, scope = maxi))
26
27 # Actualizar el modelo
28 modelo1 <- update(nulo, . ~ . + wt)
29
30 # Revisar un paso hacia adelante
31 cat("\nPaso 2:\n-----\n")
32 print(add1(modelo1, scope = maxi))
33
34 # Actualizar el modelo
35 modelo2 <- update(modelo1, . ~ . + mpg)
36
37 # Revisar un paso hacia adelante
38 cat("\nPaso 3:\n-----\n")
39 print(add1(modelo2, scope = maxi))
40
41 # Reestablecer la opción para warnings y ancho de la pantalla
42 options(warn = opt[[1]], width = opt[[2]])
43
44 # Mostrar el modelo obtenido
45 modelo2_str <- capture.output(print(summary(modelo2)))
46 cat("\nModelo RLog conseguido con regresión hacia adelante:\n")
47 cat("-----\n")
48 write.table(modelo2_str[6:10], quote = FALSE, row.names = FALSE, col.names = FALSE)
49
50 # Comparar los modelos generados
51 cat("\nComparación de los modelos considerados:\n")
52 cat("-----\n")
53 print(anova(nulo, modelo1, modelo2, test = "LRT"))

```

A partir del modelo nulo, en el primer paso (columna izquierda) vemos que la mayor reducción en AIC se obtiene agregando el peso del vehículo como predictor. La línea 29 del script actualiza el modelo en concordancia. En el segundo paso (columna central) aparecen varias advertencias sobre problemas de convergencia. La función `add1()` reporta desviación nula si agregamos como predictores las variables `hp` o `qsec`. Ya sabíamos de la primera, ahora sabemos que tampoco podemos usar la segunda. Además, debemos observar que la única reducción del AIC se consigue agregando el rendimiento del vehículo (`mpg`) como predictor. La línea 37 del script 15.6 realiza esta actualización. El siguiente paso (columna derecha de la figura 15.9) genera más advertencias de problemas de convergencia. Vemos que la mejor opción es detener la búsqueda y quedarnos con el modelo que incluye dos predictores.

<pre> Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred Paso 1: ----- Single term additions Model: am ~ 1 Df Deviance AIC <none> 23.0348 25.035 mpg 1 18.9591 22.959 disp 1 15.1710 19.171 hp 1 23.0306 27.031 drat 1 12.5165 16.517 wt 1 9.9658 13.966 qsec 1 18.4951 22.495 </pre>	<pre> Paso 2: ----- Warning: glm.fit: fitted probabilities numerically Warning: glm.fit: algorithm did not converge Warning: glm.fit: fitted probabilities numerically Single term additions Model: am ~ wt Df Deviance AIC <none> 9.9658 13.966 mpg 1 6.2682 12.268 disp 1 9.7704 15.770 hp 1 0.0000 6.000 drat 1 9.2023 15.202 qsec 1 0.0000 6.000 </pre>	<pre> Paso 3: ----- Warning: glm.fit: fitted probabilities numerically Warning: glm.fit: algorithm did not converge Warning: glm.fit: fitted probabilities numerically Warning: glm.fit: algorithm did not converge Warning: glm.fit: fitted probabilities numerically Single term additions Model: am ~ wt + mpg Df Deviance AIC <none> 6.2682 12.268 disp 1 6.0578 14.058 hp 1 0.0000 8.000 drat 1 0.0000 8.000 qsec 1 0.0000 8.000 </pre>
---	---	--

Figura 15.9: salida obtenida al construir un modelo de RLog manualmente con regresión hacia adelante.

La figura 15.10 presenta la salida a pantalla que genera la segunda parte del script 15.6. Vemos que, tanto al evaluar los coeficientes del modelo obtenido como al comparar el este modelo con los precedentes, pareciera que la inclusión del predictor `mpg` no hace un aporte importante al ajuste del modelo, y lo mejor sería evaluar su eliminación. Sin embargo, con lo pequeña que es esta muestra, no es tan aconsejable tomar decisiones cuando la evidencia está cerca del borde.

```

Modelo RLog conseguido con regresión hacia adelante:
-----
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -66.4664   39.3397 -1.690   0.0911 .
wt          14.4710    8.0062  1.807   0.0707 .
mpg         1.0849    0.8049  1.348   0.1777

Comparación de los modelos considerados:
-----
Analysis of Deviance Table

Model 1: am ~ 1
Model 2: am ~ wt
Model 3: am ~ wt + mpg
Resid. Df Resid. Dev Df Deviance Pr(>Chi)
1       16   23.0348
2       15   9.9658  1 13.0690 0.0003002 ***
3       14   6.2682  1  3.6976 0.0544915 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

Figura 15.10: modelo de RLog que resulta de aplicar manualmente regresión hacia adelante y su comparación con otros modelos evaluados durante la búsqueda.

15.6 CONFIABILIDAD DE UN MODELO DE RLOG

Aprovechando las dificultades encontradas con el ejemplo, es importante recordar que no basta con realizar una búsqueda automática de predictores y evaluar el desempeño de un clasificador, sino que también es

esencial verificar el cumplimiento de las condiciones necesarias para que un modelo de regresión logística sea válido:

1. Debe existir una relación lineal entre los predictores y la respuesta transformada.
2. Los residuos deben ser independientes entre sí.

Si bien son menos condiciones que las vistas para la regresión lineal, debemos además agregar situaciones en que puede ocurrir que el método de optimización no converja:

3. Multicolinealidad entre los predictores, que en este caso se evalúa y aborda del mismo modo que para RLM (por ejemplo, mediante el factor de inflación de la varianza o la tolerancia).
4. Información incompleta, que se produce cuando no contamos con observaciones suficientes para todas las posibles combinaciones de predictores, en especial para algún nivel de una variable categórica.
5. Separación perfecta, que ocurre cuando no hay superposición entre las clases (es decir, como vimos, cuando los predictores separan ambas clases completamente).

Y finalmente, también debemos descartar la presencia de casos problemáticos:

6. Las estimaciones de los coeficientes del modelo no están dominadas por casos influentes.

Para verificar la condición 1 podemos usar los gráficos de residuos vistos para la RLM, provistos por la función `residualPlots()` del paquete `car`, aunque el gráfico de residuos versus los valores predichos pierde sentido en la RLog (y en general en los modelos de RLG), por lo que agregamos el argumento `fitted = FALSE`. El mismo paquete también permite revisar linealidad usando gráficos de los **residuos parciales**, es decir, cuánto contribuye a los residuos cada uno de los predictores, por medio de la función `crPlots(modelo)`.

Estos gráficos aplicados al ejemplo del modelo con dos predictores de la figura 15.10 se muestran, respectivamente, en la parte superior e inferior de la figura 15.11.

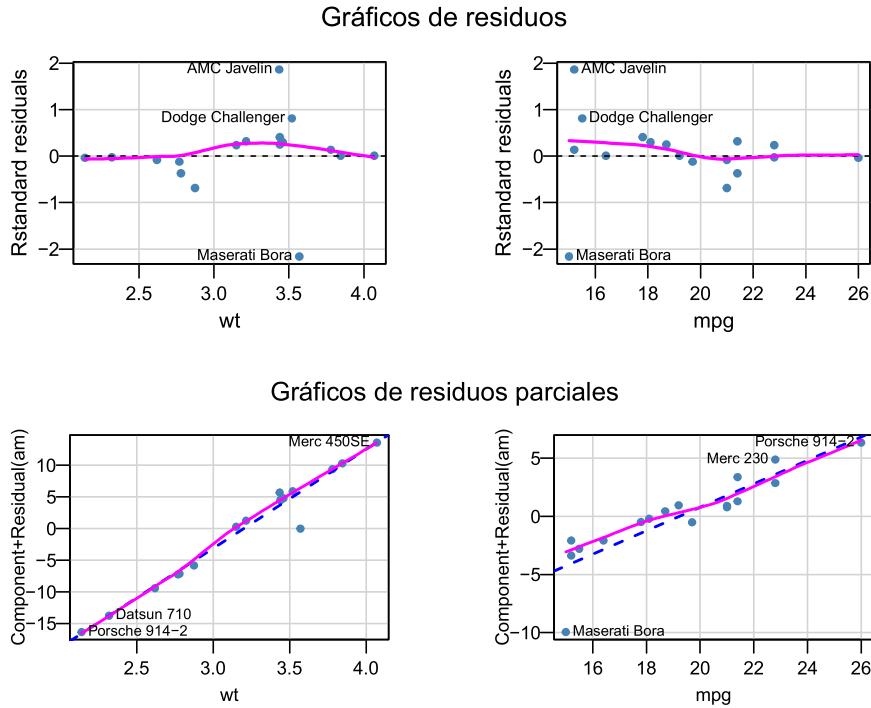


Figura 15.11: gráficos de residuos y gráficos de residuos parciales para evaluar la condición de linealidad en el modelo de RLog ejemplo.

En ambos gráficos, observamos que el supuesto de linealidad para cumplirse relativamente bien. De hecho, el resultado de las pruebas de curvatura que también realiza y reporta la función `residualPlots()` confirman esta idea:

```
Test stat Pr(>|Test stat|)
wt      1.3077      0.2528
mpg     0.3709      0.5425
```

Pasando a la segunda condición, nuevamente podemos utilizar lo hecho para RLM y aplicar la prueba de Durbin-Watson implementada en la función `durbinWatsonTest` del paquete `car`. Para el ejemplo se obtiene:

```
lag Autocorrelation D-W Statistic p-value
 1      -0.1794747    2.339983  0.532
Alternative hypothesis: rho != 0
```

Con esto descartamos que exista evidencia para sospechar que no se esté cumpliendo la condición de independencia de los residuos.

Pasemos entonces a revisar que no existe multicolinealidad fuerte, utilizando la ya conocida función `vif(modelo)` para obtener los factores de inflación de la varianza. Para el modelo ejemplo se consigue:

```
wt      mpg
3.846939 3.846939
```

Vemos que reporta valores iguales ($VIF = 3,847$) para ambos predictores, lo que es esperable en modelos con dos predictores continuos sin interacción. Si bien son un poco altos, pues hay autores que sugieren que valores superiores a 3 pueden ser problemáticos, no sobrepasan el umbral más preocupante fijado en 5 ni el umbral crítico de 10. Si bien se mantienen estos umbrales, debemos notar que los valores de VIF provienen de modelos lineales auxiliares, y no directamente del modelo de regresión logística, por lo que debemos ser especialmente cuidadosos.

La siguiente condición es problemática para construir **cualquier modelo**. Recordemos que cuando estudiámos la RLM, se dijo que, si bien **no** existe un criterio único del tamaño de la muestra requerido, se ha recomendado contar con 10 o 15 observaciones por cada predictor numérico y nivel de las variables categóricas. Esto no suele ser simple, puesto que, por ejemplo, la variable asociada al número de carburadores presentes (`carb`) tiene 4 niveles, por lo que si se considera este potencial predictor, se necesitarían más de 50 datos distribuidos más o menos balanceadamente en estos niveles. Además, habría que considerar que estas observaciones, en cada nivel, cubriera el rango de valores de cada predictor numérico. En el ejemplo, tenemos dos predictores numéricos y, si bien los casos en la muestra se distribuyen bien a lo largo de sus valores, solo tenemos 17 observaciones, muy pocas como para conseguir un modelo confiable.

Por último, queda revisar la condición relacionada a casos influyentes. Cuando revisamos los residuos parciales respecto de la variable `car` (gráfico inferior derecho de la figura 15.11), debemos notar que la recta ajustada (línea punteada en color azul-acero) esta algo desviada de la curva de ajuste local de los datos (línea sólida en tono rojizo). Esto parece deberse por la influencia de un caso muy alejado hacia los valores negativos, correspondiente al (ya sospechoso de siempre) modelo Maserati Bora. Sin embargo, esta influencia no se observa en los gráficos de residuos, tal vez moderada por la presencia de otro caso atípico, pero positivo, correspondiente al modelo AMC Javelin.

Como con los diagnósticos anteriores, aquí también podemos aplicar las herramientas usadas para RLM, como el gráfico burbuja con tres medidas de influencia para el modelo que se obtiene con la función `influencePlot(modelo)` del paquete `car`, que para el ejemplo se presenta en la figura 15.12. Recordando que esta función también reporta los estadísticos para casos notorios, nos fijamos en lo que muestra en pantalla:

	StudRes	Hat	CookD
AMC Javelin	1.7553843	0.5185520	0.97532006
Mazda RX4 Wag	-0.6005575	0.5005603	0.08357138
Maserati Bora	-2.2497803	0.2826453	0.79483991

Vemos tres observaciones que sobrepasan dos veces el apalancamiento promedio (0,353 aproximadamente para el modelo de RLog del ejemplo) y que dos de ellos también presentan distancias de Cook altas (sobre el umbral $4/17 = 0,235$) pero que no sobrepasan el valor crítico de 1 (aunque el modelo AMC Javelin está muy cerca de hacerlo). El modelo Maserati Bora también constituye claramente un valor atípico.

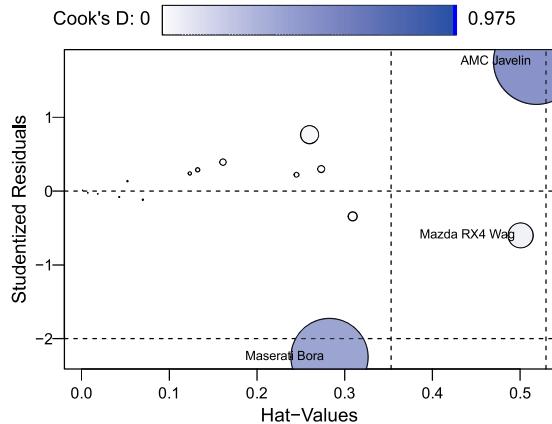


Figura 15.12: gráfico de influencia para el modelo de RLog del ejemplo.

Como hemos dicho anteriormente, correspondería hacer un análisis riguroso de los datos para decidir si eliminar estas observaciones problemáticas sin violar los principios de la ética profesional de un analista de datos. Si así se decidiera, se debe reconstruir (rehacer la búsqueda de predictores y coeficientes) y reevaluar el modelo que se consiga sin datos influyentes. Y si vuelven a aparecer otros casos con demasiada influencia, se debe iterar nuevamente en la búsqueda de un modelo confiable. En este ejemplo, el problema puede deberse al tamaño insuficiente de la muestra de observaciones, por lo que si el estudio fuera nuestro, los esfuerzos deberían concentrarse en extender este conjunto.

15.7 EJERCICIOS PROPUESTOS

- 15.1 Investiga cómo solicitar a la función `train()` del paquete `caret` que use una métrica distinta a la exactitud en su búsqueda de un modelo de regresión logística, y adapta el script 15.4 para evaluar el poder predictivo de un modelo que optimice el AUC.
- 15.2 Explica con mayor detalle el código presentado en el script 15.4 y adáptalo para conseguir una evaluación con 20 repeticiones de validación cruzada de 4 pliegues.
- 15.3 Evalúa la calidad predictiva del clasificador mostrado en la figura 15.10 en datos no vistos y compara el desempeño con los datos de entrenamiento. ¿Hay indicios de sobreajuste?.
- 15.4 Evalúa la calidad predictiva del clasificador mostrado en la figura 15.10 usando 20 repeticiones de validación cruzada de 4 pliegues con los datos de entrenamiento y compáralo con el desempeño en el conjunto de prueba. ¿Hay indicios de sobreajuste?.
- 15.5 Evalúa la calidad predictiva del clasificador mostrado en la figura 15.10 usando validación cruzada dejando uno fuera con los datos de entrenamiento y compáralo con el desempeño en el conjunto de prueba. ¿Hay indicios de sobreajuste?.
- 15.6 Investiga cómo se puede indicar a la función `train()` que utilice bootstrapping en la búsqueda de un modelo. Evalúa la calidad predictiva del clasificador con dos predictores de la figura 15.10 usando 1.000 repeticiones de bootstrapping. ¿Cómo evalúas la generalidad del modelo?.
- 15.7 Grafica la ROC asociada al clasificador de la figura 15.10 en los datos de entrenamiento. Determina el AUC y el umbral óptimo. Con este último, aplica el modelo a los datos de prueba y agrega al gráfico la ROC correspondiente. ¿Cómo se comparan los desempeños?
- 15.8 Investiga cómo usar la función `roc.test()` del paquete `pROC` y verifica si la curva ROC generada por el clasificador de la figura 15.10 para los datos de entrenamiento es significativamente mejor que la generada para los datos de prueba.

- [15.9] Reconstruye el clasificador de la figura 15.10, pero ahora sin considerar los dos casos con distancia de Cook sobre 0,5. ¿Se obtiene ahora un modelo sin la presencia de datos influyentes? ¿Cómo se comparan ambos modelos (coeficientes, calidad predictiva).
- [15.10] Para el clasificador de la pregunta anterior, evalúa su capacidad predictiva usando validación cruzada dejando uno fuera con los datos de entrenamiento y luego en el conjunto de prueba. ¿Hay indicios de sobreajuste?.
- [15.11] Construye un clasificador para otra variable binaria del conjunto de datos `mtcars` usando el método de todos los subconjuntos considerando modelos con dos o tres predictores. ¿Resulta confiable? ¿Qué calidad predictiva consigue? ¿Es generalizable?
- [15.12] Tome una muestra de 300 observaciones del conjunto de datos `titanic_train` del paquete `titanic` que tengan datos completos (no vacíos ni nulos), y construya un modelo de RLog con tres o cuatro predictores para predecir la variable de salida `Survived` utilizando regresión paso a paso hacia adelante. Luego evalúa su confiabilidad y calidad predictiva usando 10 repeticiones de validación cruzada de 6 pliegues.
- [15.13] Tome una muestra de 100 observaciones del conjunto de datos `titanic_test` del paquete `titanic` que tengan datos completos (no vacíos ni nulos), y aplique el modelo obtenido en la pregunta anterior. ¿Logró una buena generalización?
- [15.14] Investiga de qué se trata el método Recursive Feature Elimination implementado en el paquete `caret`. Bosqueja cómo se podría buscar un clasificador utilizando este algoritmo y algún tipo de validación cruzada.

15.8 BIBLIOGRAFÍA DEL CAPÍTULO

Ayala, J. (2020). *Minería de datos*.

Consultado el 23 de junio de 2021, desde <https://rpubs.com/JairoAyala/592802>

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern recognition letters*, 27(8), 861-874.

Nelder, J. A., & Wedderburn, R. W. M. (1972). Generalized linear models.

Journal of the Royal Statistical Society Series A: Statistics in Society, 135(3), 370-384.

Tharwat, A. (2021). Classification assessment methods. *Applied computing and informatics*, 17(1), 168-192.