Fundamentos de la programación estadística y Data Mining en R

Unidad 3. Regresión logística y Árboles de Decisión en R

Dr. Germán Rosati (Digital House - UNTREF - UNSAM)

19 julio, 2017

Implementación de una regresión logística en R

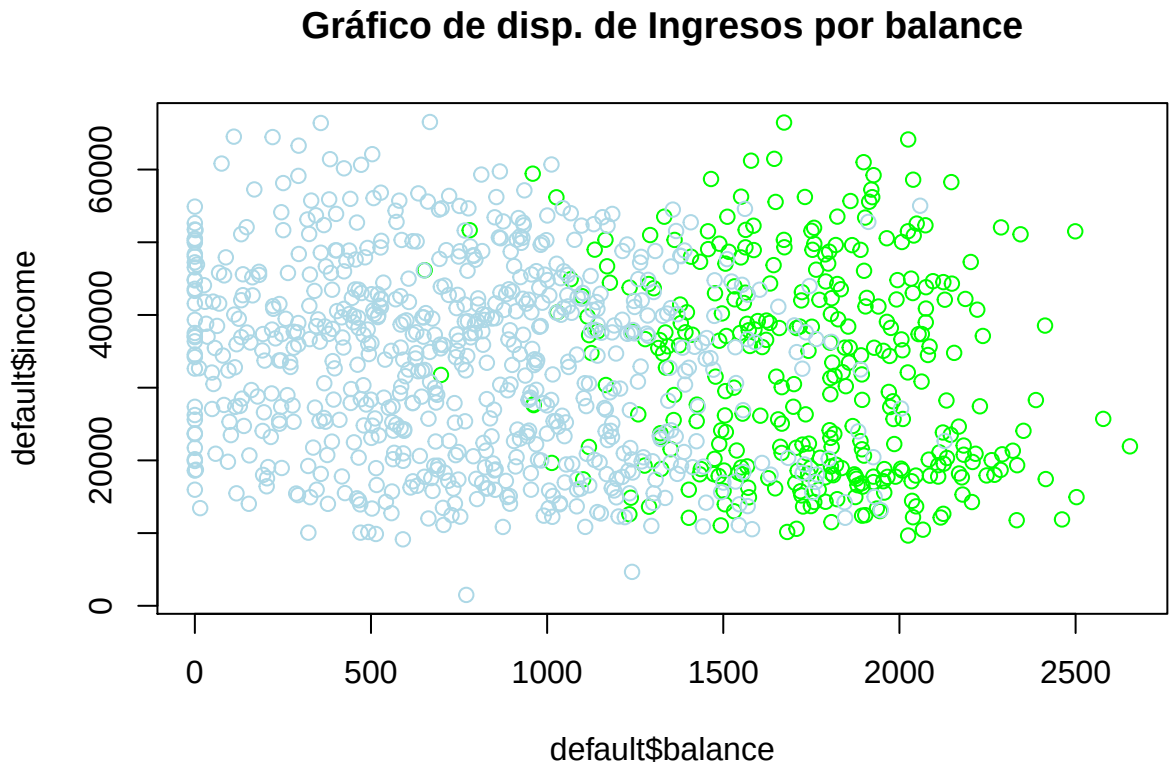
- Para variables dependientes binarias o categóricas el modelo lineal no suele ser recomentable
- La línea de regresión puede tomar valores negativos o mayores a 1
- Solución: usar una función logística
- En lugar de intentar predecir $E(Y|X)$ intentaremos predecir $P(Y = 1|X)$
- El modelo será: $P(Y = 1|X) = \frac{e_0^{\beta}+\beta_1 X}{1+e_0^{\beta}+\beta_1 X}$

Implementación de una regresión logística en R: `glm()`

- Para eso usaremos la función `glm()` que sirve para ajustar una gran familia de modelos llamada "Modelos Lineales Generalizados" de los cuales la regresión logística es uno de ellos.
- La sintaxis de `glm()` es muy similar a la de `lm()` con la única diferencia de que debemos pasar un argumento '`family=binomial` para decirle a `R` que queremos correr una regresión logística en lugar de algún otro GLM.
- Carguemos los datos y hagamos algunos gráficos para refrescar conceptos vistos en la clase pasada.

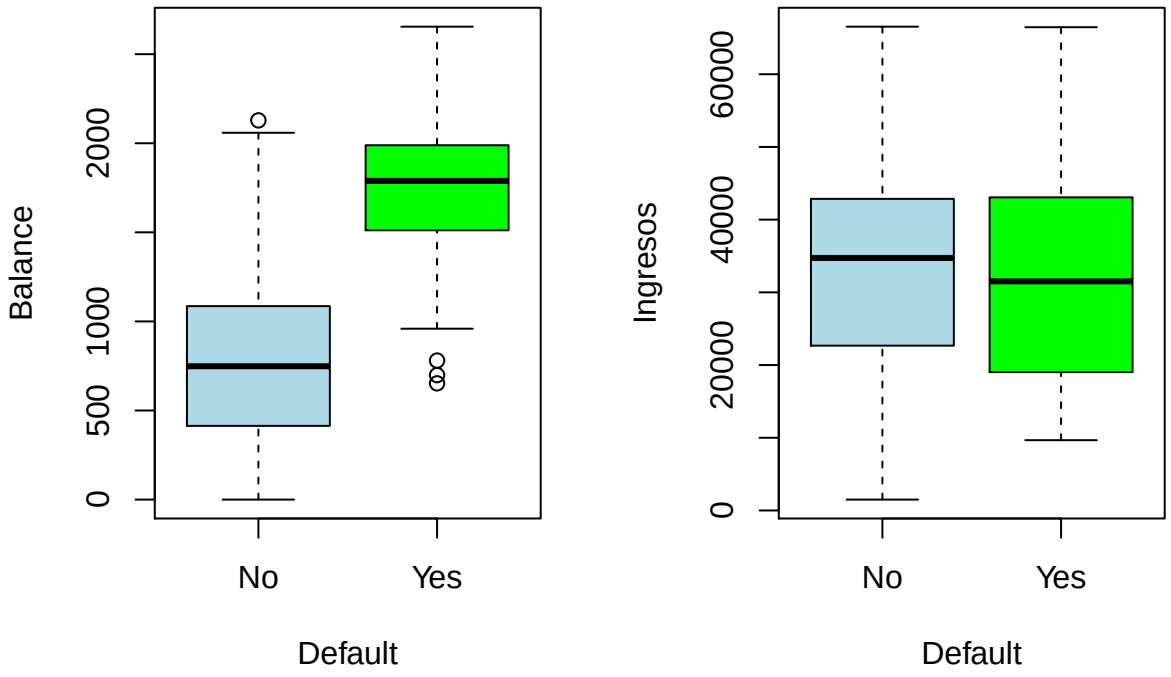
Gráficos de dispersión

```
setwd("/media/digitalhouse/Elements6/PEN/KINGSTON/PEN2/Cursos/REPO_Curso_Fund_Prog_R/Data")
default <- read.csv("default.csv")
plot(default$balance, default$income, col = c("lightblue", "green")[default$default],
    main = "Gráfico de disp. de Ingresos por balance")
```



Gráficos Boxplot

```
par(mfrow = c(1, 2))
boxplot(default$balance ~ default$default, col = c("lightblue", "green"), xlab = "Default",
    ylab = "Balance")
boxplot(default$income ~ default$default, col = c("lightblue", "green"), xlab = "Default",
    ylab = "Ingresos")
```



Implementación de una regresión logística en R: `glm()`

```
glm_fit<-glm(default~.
         ,data=default
         ,family = binomial)
summary(glm_fit)
##
## Call:
## glm(formula = default ~ ., family = binomial, data = default)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.76870  -0.34482  -0.09078   0.29302   2.86443
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -8.247e+00  7.342e-01 -11.233   <2e-16 ***
## studentYes  -4.059e-01  3.620e-01  -1.121    0.262
## balance      5.540e-03  3.569e-04  15.524   <2e-16 ***
## income       1.185e-05  1.299e-05   0.912    0.362
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1298.76  on 1032  degrees of freedom
## Residual deviance:  542.63  on 1029  degrees of freedom
## AIC: 550.63
##
## Number of Fisher Scoring iterations: 6
```

- Pareciera, entonces, que el `balance` y la condición de `student` son las variables que más influyen en la probabilidad de no pagar un crédito.
- Al igual que en `lm()` podemos usar la función `coef()`

```
coef(glm_fit)
##   (Intercept)    studentYes       balance        income
## -8.2472931607 -0.4059228872  0.0055402157  0.0000118527
```

Implementación de una regresión logística en R: `predict()`

- La función `predict()` puede ser usada para generar predicciones acerca de la variable dependiente. Debemos usar el argumento '`type=response` para decirle a `R` que genere las probabilidades de que $P(Y = 1|X)$ en lugar de otra información (como por ejemplo el logit)

- Sabemos que esas son las probabilidades de que una persona dada no pague su crédito por la funcion 'contrasts()

```
glm_probs<-predict(glm_fit ,type="response")
contrasts(default$default)
##     Yes
## No    0
## Yes   1
```

Implementación de una regresión logística en R: generando predicciones

```
glm_pred<-rep("No", 1033)
glm_pred[glm_probs>0.5]<-"Yes"
table(glm_pred,default$default)
##
## glm_pred  No Yes
##      No  645  68
##      Yes  55 265
(645+265)/1033
## [1] 0.8809293
mean(glm_pred==default$default)
## [1] 0.8809293
```

- La primera línea genera 1033 observaciones con el valor `No` por defecto.
- La segunda transforma en `Sí` a aquellas observaciones cuya `glm_probs > 0.5`
- Una vez armadas las predicciones podemos usar `table()` para generar una matriz de confusión de modelo.
- Y calculando la media de las observaciones que en `glm_pred` son iguales a las de `default$default` (la variable dependiente original), tenemos una medida del error de clasificación del modelo.

Implementación de una regresión logística en R: training error y test error

- A primera vista, el modelo parece funcionar divinamente... 88% de las observaciones son clasificadas bien. Pero hay un problema: la enorme mayoría de las observaciones no generaron un cese de pagos... Por lo cual, solamente observando eso el modelo aporta poca información... usando como modelo esa proporción estaríamos bien.
- Hemos estimado el error sobre TODOS nuestros datos.
- Lo cual nos lleva a plantearnos el problema del "Test-Error" y el "Training-Error"

(...)

Implementación de una regresión logística en R: training error y test error -volviendo-

- Entonces, ¿cómo lo implementamos en R?

```
set.seed(7)
train<-sample(x = 1:nrow(default)
          , size = 600
          , replace = FALSE)
glm_fit<-glm(default~.
         , data = default
             , subset = train
         , family = binomial)
```

- La lógica es generar 600 números aleatorios sin reposición con la función `sample()` que varíen entre 1 y el total de filas del dataset `default`.
- Luego, usamos ese vector como un índice de posición para poder hacer subsetting de `default`.
- En tercer lugar, estimamos el mismo modelo de regresión logística pero pasando como argumento `subset = train`.
- También podríamos haberlo hecho de esta forma:
- Luego, nos queda realizar la predicción sobre el test set.

```
glm_probs<-predict(glm_fit
           , newdata = default[-train,]
           , type ="response")
glm_pred<-rep("No",433)
glm_pred[glm_probs>0.5]="Yes"
table(glm_pred, default$default[-train])
##
## glm_pred  No Yes
##      No  259  28
##      Yes  33 113

mean(glm_pred==default$default[-train])
## [1] 0.8591224
```

- Básicamente, replicamos el mismo procedimiento que al principio, solamente que para el uso del `predict()` en el argumento `data = default[-train,]` negamos el vector `train`.
- ¿Qué pasa con el test error? Es más alto... No tanto... pero es para tener en cuenta.