Guía para la obtención de modelos de regresión logística con R

En esta breve guía se va a introducir cómo obtener modelos de regresión logística con R.

1. Obtención de modelos

La función de R que permite ajustar modelos de regresión logística es *glm()*, donde debemos incluir la opción *family=binomial* para indicar el tipo de modelo que queremos ajustar (se trata de una función que permite ajustar gran variedad de modelos). Para poder utilizarla, debemos indicar qué modelo queremos ajustar (a través de las fórmulas ya vistas para la función *lm*) y sobré qué conjunto de datos (opción *data=*).

```
modelo<-glm(y~x1+x2+x3,data=datos,family=binomial)
modelo<-glm(y~x1+x2+x3-1,data=datos,family=binomial) #Modelo sin constante
modelo<-glm(y~.,data=datos,family=binomial) #Modelo "con todo"
```

La llamada a esta función sólo nos ofrece la fórmula introducida y el valor de los coeficientes estimados. Si queremos obtener más información acerca del modelo, debemos utilizar la función *summary()*.

Por ejemplo,

```
> modelo<-glm(chd~.,datos,family=binomial)
> summary(modelo)

Call:
glm(formula = chd ~ ., family = binomial, data = datos)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.7781  -0.8213  -0.4387   0.8889   2.5435

Coefficients:
                 Estimate Std. Error z value Pr(>|z|)
(Intercept)    -6.1507209  1.3082600  -4.701 2.58e-06 ***
sbp             0.0065040  0.0057304   1.135 0.256374
tobacco         0.0793764  0.0266028   2.984 0.002847 **
ldl             0.1739239  0.0596617   2.915 0.003555 **
adiposity       0.0185866  0.0292894   0.635 0.525700
famhistPresent  0.9253704  0.2278940   4.061 4.90e-05 ***
typea           0.0395950  0.0123202   3.214 0.001310 **
```

```
obesity        -0.0629099  0.0442477  -1.422 0.155095
alcohol         0.0001217  0.0044832   0.027 0.978350
age             0.0452253  0.0121298   3.728 0.000193 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 596.11  on 461  degrees of freedom
Residual deviance: 472.14  on 452  degrees of freedom
AIC: 492.14

Number of Fisher Scoring iterations: 5
```

Como podemos observar, nos ofrece información acerca de los parámetros (y su significatividad), el AIC y los logaritmos de las verosimilitudes para que podamos evaluar la bondad del modelo. Lo que R denomina deviance es: $-2(\log(L))$, donde L es la verosimilitud. La *null deviance* se refiere al "no modelo", mientras que "residual deviance" se refiere al modelo ajustado. Por lo tanto, si quisiéramos obtener en $Pseudo - R^2$ de McFadden, podríamos utilizar la fórmula para calcularlo:

```
1-modelo$deviance/modelo$null.deviance
```

No obstante, de esta forma estaríamos obteniéndolo para el conjunto de datos de entrenamiento y, como ya se ha visto, no se trata de una estimación muy realista. Desgraciadamente, no existe ninguna función en R que calcule este indicador para el conjunto de datos de prueba, por lo que hemos de utilizar una función propia:

```
pseudoR2(modelo,data_train,"nombreVariableObjetivo")
pseudoR2(modelo,data_test,"nombreVariableObjetivo")
```

Recordemos que, a la hora de evaluar un modelo de regresión, es importante hacerse una idea de la importancia de las variables que lo componen. Para ello, una alternativa es calcular como se ve afectado el mismo al eliminar cada una de las variables individualmente. De esta manera, las variables más importantes harán que el modelo empeore mucho al eliminarlas mientras que las variables menos útiles apenas tendrán efecto sobre la calidad del mismo. En el caso de la regresión logística, podemos medir este "empeoramiento" en términos del $Pseudo - R^2$. Para obtener un gráfico que nos muestre la pérdida a la que dan lugar cada una de las variables, podemos usar la función *propia* siguiente:

```
impVariablesLog(modelo,"nombreVariableObjetivo")
```

Adicionalmente, si queremos obtener la predicción a través del modelo obtenido para un nuevo conjunto de datos, recurriremos a la función *predict()*, cuyo primer argumento ha de ser el objeto tipo *glm* y el segundo, los datos que se quieren predecir. En este caso se debe añadir, además, la opción *type = "response"* para indicar que se desean obtener las probabilidades, y no los valores de la función *logit*.

2. Selección de variables

De nuevo, en R podemos llevar a cabo los métodos de selección de variables ya estudiados: hacia delante, hacia atrás y paso a paso. Recuerda que en cada iteración de los métodos se evalúa el AIC/SBC

del modelo resultante. De esa forma, se incluyen/eliminan factores siempre y cuando mejore dicho estadístico.

Para poder llevar a cabo dicha selección, es necesario que indiquemos el rango de los modelos a evaluar, es decir, el mayor y el menor modelo posible (en este caso lo haremos a partir de la función *glm*. Este proceso se realiza a partir de la función *step*:

```
null<-glm(y~1,data=datos,family=binomial)
full<-glm(y~.,data=datos,family=binomial)
modelo<-step(null, scope=list(lower=null, upper=full), direction="forward")
modelo<-step(full, scope=list(lower=null, upper=full), direction="backward")
modelo<-step(null, scope=list(lower=null, upper=full), direction="both")
```

No olvides que en estos modelos de selección de variables existe una jerarquía entre los efectos y las interacciones de manera que una interacción no puede entrar al modelo si no han entrado previamente los efectos que la forman. De igual forma, no puede salir un efecto si no han salido previamente todas las interacciones de las que forma parte.

2.1. Lista de todas las posibles interacciones

Como en el tema anterior, se puede crear en R una función que genere automáticamente una fórmula que contenga todos los posibles efectos (variables e interacciones) para poderla usar en el proceso de selección de variables:

```
formulaInteracciones<-function(data,posicion){
  listaFactores<-c()
  lista<-paste(names(data)[posicion],'~')
  nombres<-names(data)
  for (i in (1:length(nombres))[-posicion]){
    lista<-paste(lista,nombres[i],'+')
    if (class(data[,i])=="factor"){
      listaFactores<-c(listaFactores,i)
      for (j in ((1:length(nombres))[-c(posicion,listaFactores)])){
        lista<-paste(lista,nombres[i],':',nombres[j],'+')
      }
    }
  }
  lista<-substr(lista, 1, nchar(lista)-1)
  lista
}

formInt<-formulaInteracciones(datos,posicionColumnaVariableObjetivo)
full<-glm(formInt, data=datos, family=binomial)
modelo<-step(null, scope=list(lower=null, upper=full), direction="forward")
```

Dado que se trata de la misma función, vuelve a ser necesario indicar el conjunto de datos, así como el número de la columna que contiene la variable objetivo (*posicionColumnaVariableObjetivo*).

3. Curva ROC

Para obtener la curva ROC, así como su área, existen varios paquetes. No obstante, nosotros vamos a usar la librería *pROC* que permite obtener el gráfico de la curva ROC y su área de la siguiente forma:

```
curvaRoc<-roc(datos$y, predict(modelo,datos,type = "response"), direction="<")
plot(curvaRoc)
```

Como se puede observar, es necesario indicarle a la función cuál es la variable objetivo, así como las predicciones correspondientes. Nótese que se puede obtener para el conjunto de datos de entrenamiento o test, que se generan de la misma forma que se vió en regresión lineal.

4. Validación cruzada

Recordemos que este método consiste en dividir el conjunto de datos en submuestras e iterativamente construir el modelo con todas las observaciones menos las de una submuestra y evaluarlo a continuación con las observaciones de dicha submuestra excluida.

De nuevo, podemos utilizar la función `train()` de la librería "caret". Debemos indicarle que el modelo es de tipo *glm binomial* y la métrica que queremos que obtenga, el área bajo la curva ROC en este caso.

```
modelo<- train(ecuacion, data= datos,
                method = "glm", family="binomial",
                metric = "ROC",
                trControl = trainControl(method = "cv",
                                         number = Ngroup,
                                         summaryFunction=twoClassSummary,
                                         classProbs=TRUE,
 returnResamp="all"
                )
)
modelo$results #Muestra el AUC, sensibilidad y especificidad, junto con su sd
```

Para intentar reducir lo máximo posible el efecto del azar a la hora de realizar las particiones de la validación cruzada es recomendable realizar dicho proceso para varias semillas, de manera que se pueda evaluar cómo se comportan los modelos para distintos conjuntos de datos. Para ello, simplemente se debe indicar `method = "repeatedcv"` junto con el número de grupos a realizar (`number`) y el número de repeticiones (`repeats`).

De nuevo, una forma de interpretar fácilmente los últimos resultados es construir un diagrama de cajas con todos los estadísticos obtenidos al repetir el proceso de validación cruzada. Si se representan estos diagramas de cajas para distintos modelos sobre la misma escala, podremos concluir qué modelo es preferible sobre el resto.

Para el ejemplo de la enfermedad coronaria visto anteriormente se ha utilizado la siguiente sentencia y obtenido los resultados que aparecen en la Figura 1:

```
set.seed(12345)
modelo1 <- train(chd ~ age + famhist + tobacco + typea + ldl,datos,
                 method = "glm", family="binomial", metric = "ROC",
```

```
                trControl = trainControl(method = "repeatedcv",
                                         number = 5, repeats = 100,
                                         summaryFunction=twoClassSummary,
                                         classProbs=TRUE,
                                         returnResamp="all"
                )
)
set.seed(12345)
modelo2 <- train(chd ~ age + famhist + tobacco + typea + ldl+obesity,datos,
                method = "glm", family="binomial", metric = "ROC",
                trControl = trainControl(method = "repeatedcv",
                                         number = 5, repeats = 100,
                                         summaryFunction=twoClassSummary,
                                         classProbs=TRUE,
                                         returnResamp="all"
                )
)
set.seed(12345)
modelo3 <- train(chd ~ age + famhist + tobacco + typea,datos,
                method = "glm", family="binomial", metric = "ROC",
                trControl = trainControl(method = "repeatedcv",
                                         number = 5, repeats = 100,
                                         summaryFunction=twoClassSummary,
                                         classProbs=TRUE,
                                         returnResamp="all"
                )
)
todo1<-data.frame(roc=modelo1r$resample[,1],modelo=rep("m1",nrow(modelo1r$resample)))
todo2<-data.frame(roc=modelo2r$resample[,1],modelo=rep("m2",nrow(modelo2r$resample)))
todo3<-data.frame(roc=modelo3r$resample[,1],modelo=rep("m3",nrow(modelo3r$resample)))
todo<-rbind(todo1,todo2,todo3)
boxplot(roc~modelo,data=todo,main="AUC")
```

Como se puede comprobar en la figura, el mejor modelo es el número 1 en términos de "bondad media". Se puede observar que la variabilidad de los modelos es muy parecida. Otra ventaja de estos gráficos es que permiten evaluar fácilmente la mejora conseguida al incluir/eliminar factores. Por ejemplo, la diferencia entre el primer y el segundo modelo pone de manifiesto el efecto de la variable *obesity*, mientras que la diferencia entre el primero y el tercero lo hace de la variable *ldl*. Podemos observar, que al eliminar esta última, el modelo empeora significativamente, demostrando la utilidad de la misma. Siguiendo esta misma idea, podemos concluir que la variable *obesity*, aunque útil, no lo es tanto como *ldl*.

5. Selección punto de corte

Como ya se ha comentado, la gran diferencia con los modelos de regresión lineal, es que para regresión logística es necesario determinar la probabilidad a partir de la cuál se considerará una observación como *evento*.

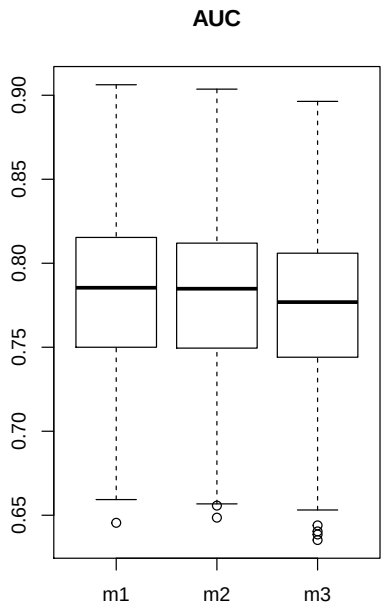

Figura 1: Salidas validación cruzada ejemplo chd

Para ello, vamos a obtener, para una rejilla de posibles puntos de corte y el conjunto de datos de prueba, el valor de la tasa de acierto, la sensibilidad, la especificidad y el índice de Youden y, a partir de ahí, tomar la decisión.

Para ello, se puede recurrir a la función `confusionMatrix` de la librería *caret*. Esta función requiere que le demos, para el conjunto de datos deseado, los valores predichos y los reales. A parte de las medidas anteriores, la función calcula otras que no hemos estudiado, por lo que podemos ignorarlas. Para automatizar el proceso de predicción a través del modelo y producir solo las medidas estudiadas, podemos crear la siguiente función:

```
sensEspCorte<-function(modelo,dd,nombreVar,ptoCorte,evento){
  probs <-predict(modelo,newdata=dd,type="response")
  cm<-confusionMatrix(data=factor(ifelse(probs>ptoCorte,1,0)),
                                reference=dd[,nombreVar],positive=evento)
  c(cm$overall[1],cm$byClass[1:4])
}
posiblesCortes<-seq(0,1,0.01)
rejilla<-data.frame(t(rbind(posiblesCortes,sapply(posiblesCortes,function(x)
                        sensEspCorte(modelo,data_test,"nombreVar",x,"evento")))))
rejilla$Youden<-rejilla$Sensitivity+rejilla$Specificity-1
plot(rejilla$posiblesCortes,rejilla$Youden)
plot(rejilla$posiblesCortes,rejilla$Accuracy)
rejilla$posiblesCortes[which.max(rejilla$Youden)]
rejilla$posiblesCortes[which.max(rejilla$Accuracy)]
```