Guía para la obtención de modelos de regresión lineal con R

En esta breve guía se va a introducir cómo obtener el modelo de regresión lineal con el software R.

1. Obtención de modelos

La función de R que permite ajustar modelos de regresión lineal en general es *lm()*. Para poder utilizarla, debemos indicar qué modelo queremos ajustar y sobré qué conjunto de datos (opción *data=*). Alternativamente, se puede usar la función *glm()* con la opción *family=gaussian*.

Los modelos a ajustar se especifican a través de fórmular mediante las cuales indicamos la variable objetivo, seguido de '∼' y, a continuación, se indican las variables regresoras separadas por '+'. Además, si se desean incluir todas las variables que haya en el conjunto de datos como predictoras, se indicará con un punto.

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

Las interacciones y las variables transformadas se incluyen facilmente a partir de dos puntos entre las variables que forman dicha interacción (por ejemplo, *Sex:Height*) y a partir de la fórmula que lo define (por ejemplo, *log(Age)*), respectivamente.

La llamada a esta función sólo nos ofrece la fórmula introducida y el valor de los coeficientes estimados. Si queremos obtener información acerca del contraste general de regresión, de los contrastes de hipótesis sobre los parámetros, y los estadísticos R^2 y R^2_{ajust}, debemos utilizar la función *summary()*.

Por ejemplo,

```
> modelo<-lm(FEV~Height+Age,data=datosFEV)
> summary(modelo)

Call:
lm(formula = FEV ~ Height + Age, data = datosFEV)

Residuals:
     Min       1Q   Median       3Q      Max
-1.50533 -0.25657 -0.01184  0.24575  2.01914

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -4.610466   0.224271 -20.558  < 2e-16 ***
```

```
Height       0.109712   0.004716  23.263  < 2e-16 ***
Age          0.054281   0.009106   5.961 4.11e-09 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.4197 on 651 degrees of freedom
Multiple R-squared:  0.7664,        Adjusted R-squared:  0.7657
F-statistic:  1068 on 2 and 651 DF,  p-value: < 2.2e-16
```

Los objetos de tipo *lm* son listas que contienen toda la información del modelo. En particular, podemos destacar: *coef()*, contiene los coeficientes estimados; *residuals()*, contiene los residuos; y *fitted.values()*; contiene los valores predichos.

R también nos ofrece el cálculo de los índices AIC y SBC (aunque a este segundo lo denomina BIC) sin más que llamar a *AIC()* o *BIC()*, respectivamente.

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 *lm* y el segundo, los datos que se quieren predecir.

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.

Para obtener dichos resultados, podemos utilizar la función `modelEffectSizes` de la librería “lmSupport”. Esta función calcula como aumenta la suma de cuadrados de los errores (SSE) al eliminar las distintas variables, así como la disminución en el R^2. Adicionalmente, podemos obtener un gráfico que contenga dicha información:

```
modelEffectSizes(modelo)
barplot(sort(modelEffectSizes(modelo)$Effects[-1,4],decreasing =T),
        las=2,main="Importancia de las variables (R2)")
```

Por último, es importante conocer siempre el número de parámatros que componen el modelo para tener una idea de la complejidad del mismo. Una manera rápida de obtenerlo es contando el numero de parámatros que contiene: `length(coef(modelo))`.

2. Selección de variables

En R podemos llevar a cabo los métodos de selección de variables ya estudiados: hacia delante, hacia atrás y paso a paso.

2.1. Función *step*

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. Este proceso se realiza a partir de la función *step*:

```
null<-lm(y~1, data=datos) #Modelo mínimo
full<-lm(y~., data=datos) #Modelo máximo
modelo<-step(null, scope=list(lower=null, upper=full), direction="forward")
summary(modelo)
```

Como se puede ver, *null* y *full* son el "menor" y el "mayor" modelos posibles, respectivamente. En este ejemplo, en *full* sólo se han incluido los factores principales, pero se podrían incluir también las interacciones correspondientes.

La opción *direction* permite seleccionar el método entre *forward, backward o both*. Además, por defecto la selección se basa en el estadístico AIC pero, si deseamos utilizar el SBC deberemos incluir la opción $k = log(nrow(datos))$ en la función *step*. Al ejecutar la sentencia correspondiente nos aparece en pantalla el proceso de selección de variables y se guarda el modelo final que después puede utilizarse como si hubiese sido generado a partir de la sentencia *lm*. Nótese que si se ha seleccionado el método hacia atrás, se debe poner como primer argumento el modelo "con todo" para que pueda eliminar efectos.

Es importante destacar que en los modelos de selección de variables en R 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.

2.2. Lista de todas las posibles interacciones

De cara a considerar todos los posibles efectos que puedan aportar información sobre la variable objetivo, sería interesante poder generar automáticamente una fórmula que los contenga todos para poderla usar en el proceso de selección de variables o de búsqueda exhaustiva.

Para ello, podemos crear en R una función que realice esa función de la siguiente forma:

```
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,columnaVariableObjetivo)
full<-lm(formInt, data=datos)
```

Nótese que es necesario indicar el conjunto de datos, así como el número de la columna que contiene

la variable objetivo (*posicion/columnaVariableObjetivo*). El modelo *full* se puede introducir como el modelo "máximo" de la función *step*.

Por último, es importante destacar que si se desean incluir todas las posibles interacciones (incluidas aquellas que se dan entre dos variables continuas, que no han sido estudiadas en esta asignatura debido a la complejidad de su interpretación), se puede hacer a partir de la fórmula: `y~.^2`

3. Comparación de modelos a partir de *Training-test*

Como ya hemos visto, una de las mejores formas de comparar modelos ya construidos es a través de un conjunto de datos test, que no forme parte de la construcción del modelo y que nos permita obtener una estimación *insesgada* del funcionamiento del mismo. Para ello, haremos uso de la libreria *caret* y las siguientes sentencias:

```
partitionIndex <- createDataPartition(varObj, p=0.8, list=FALSE)
data_train <- dataFrame[partitionIndex,]
data_test <- dataFrame[-partitionIndex,]
```

Una vez obtenidos estos subconjuntos, deberemos obtener los modelos únicamente con el conjunto de datos de entrenamiento y después evaluarlos y decidir el mejor de entre ellos a partir del conjunto de datos de prueba. Para ello, debemos predecir, utilizando el modelo, la variable objetivo para las observaciones de *Test* y comparar con los valores reales.

```
Rsq<-function(modelo,varObj,datos){
  testpredicted<-predict(modelo, datos)
  testReal<-datos[,varObj]
  sse <- sum((testpredicted - testReal) ^ 2)
  sst <- sum((testReal - mean(testReal)) ^ 2)
  1 - sse/sst
}
Rsq(modelos,"nombreVariableObjetivo",data_train)
Rsq(modelos,"nombreVariableObjetivo",data_test)
```

Es importante no olvidar dos cosas: 1) el valor del R^2 obtenido en test se trata de una estimación más realista del comportamiento futuro del modelo y, por tanto, nos permitirá hacernos una idea sobre cómo de bueno o malo será el modelo aplicado a datos futuros; y 2) la diferencia en los valores de R^2 en train y test nos permite saber cómo de estable es el modelo (cuando los valores se parecen, es esperable que para cualquier conjunto de datos futuro esto sea también así), así como de la posible presencia de sobreajuste (variables que mejoran el modelo en train, no así en test).

4. Comparación de modelos a partir de 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. Nótese que cada ejecución de validación cruzada da lugar a una única predicción de todas las observaciones por lo que se puede obtener la SSE correspondiente. Para evitar trabajar con cantidades tan grandes, la comparación de modelos se suele llevar a cabo a partir del R^2.

La función que nos permitirá realizar este proceso de validación cruzada vuelve a ser `train` de la librería *caret*.

```
set.seed(1234)
modelo <- train(ecuacion, data = datos,
    method = "lm",
    trControl = trainControl(method="cv", number=Ngroup, returnResamp="all")
)
modelo$results
```

Como se puede observar, hay que indicar el conjunto de datos, el modelo a evaluar, y el número de grupos a realizar. Si se quiere obtener resultados reproducibles, es necesario fijar la semilla previamente con la función `set.seed()`.

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:

```
set.seed(1234)
modelo <- train(ecuacion, data = datos,
    method = "lm",
    trControl = trainControl(method="repeatedcv", number=Ngroup, repeats=Nrep,
    returnResamp="all")
)
modelo$results
```

De nuevo, hay que indicar el conjunto de datos, la fórmula del modelo, el número de grupos de la validación cruzada, el número de repeticiones, la semilla.

Una forma de interpretar fácilmente los últimos resultados es construir un diagrama de cajas con todos los R^2 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.

```
set.seed(1234)
modelo1 <- train(ecuacion1, data = datos,
    method = "lm",
    trControl = trainControl(method="repeatedcv", number=Ngroup, repeats=Nrep,
    returnResamp="all")
)
set.seed(1234)
modelo2 <- train(ecuacion2, data = datos,
    method = "lm",
    trControl = trainControl(method="repeatedcv", number=Ngroup, repeats=Nrep,
    returnResamp="all")
)

todo<-rbind(cbind(modelo1$resample[,2],modelo=rep("Modelo1",nrow(modelo1$resample))),
    cbind(modelo2$resample[,2],modelo=rep("Modelo2",nrow(modelo2$resample))))
boxplot(Rsquared~modelo,data=todo,main="R-Square")
```

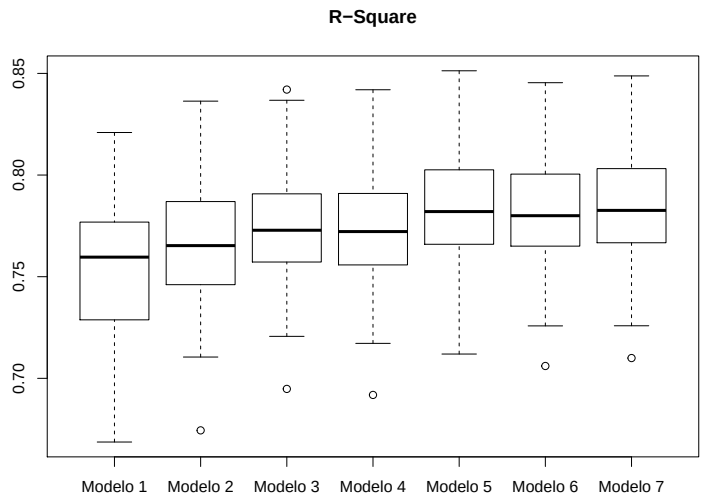

Figura 1: Salidas validación cruzada ejemplo FEV

Para el ejemplo del funcionamiento pulmonar visto anteriormente se ha utilizado la siguiente sentencia y obtenido los resultados que aparecen en la Figura 1:

```
modelos<-c("FEV~Height","FEV~Height+Age","FEV~Height+Age+Sex",
           "FEV~Height+Age+Sex+gen","FEV~Height+Age+Sex+Height*Sex",
           "FEV~Height+Age+Sex+Age*Sex","FEV~Height+Age+Sex+Height*Sex+Age*Sex")
total<-c()
for (i in 1:length(modelos)){
  set.seed(1712)
  vcr<-train(as.formula(modelos[i]), data = fevdata,
    method = "lm",
    trControl = trainControl(method="repeatedcv", number=5, repeats=20,
    returnResamp="all")
  )
  total<-rbind(total,cbind(vcr$resample[,2],modelo=rep(paste("Modelo",i),
    nrow(vcr$resample))))
}
boxplot(Rsquared~modelo,data=total,main="R-Square")
```

Como se puede comprobar en la figura, el mejor modelo es el número 5 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 que el efecto de la variable *Age* es significativo. Por el contrario, si comparamos el tercero y el cuarto, que se diferencian en la inclusión de la variable *Gen*, comprobamos que dicha variable no aporta información al modelo.

En ocasiones, no obstante, no queda claro visualmente qué modelo es preferible. Para esos casos, podemos calcular la media y desviación típica de los R^2 de las repeticiones de la validación cruzada como complemento a los diagramas de cajas anteriores, utilizando el siguiente código:

```
aggregate(Rsquared~modelo, data = total, mean)
aggregate(Rsquared~modelo, data = total, sd)
```