ILUSTRACIÓN PRÁCTICA EN R (*neuralnet*)

SOBRE REDES NEURONALES CLÁSICAS

En este documento vamos a ilustrar de manera sucinta cómo se pueden estimar modelos de redes neuronales clásicas en R a través del paquete *neuralnet*. Así, este documento permitirá asentar mejor algunos conocimientos básicos de la parte teórica sobre el funcionamiento y concepto de las redes neuronales clásicas.

El primer paso es cargar el paquete *neuralnet* de R que es necesario para reproducir este ejemplo (o instalarlo, si no se ha hecho aún). Para ello, utilizaremos la siguiente sintaxis:

```
install.packages("neuralnet", dependencies = T)

library(neuralnet)
```

En el caso de esta ilustración, vamos a utilizar un fragmento de una base de datos donde una serie de indicadores psicolingüísticos fueron extraídos de respuestas construidas para intentar predecir la personalidad de los participantes de un estudio[1]. Concretamente,

[1] Martínez-Huertas, J.A., Moreno, J.D., Olmos, R., Martínez-Mingo, A., & Jorge-Botana, G. (2022). A failed cross-validation study on the relationship between LIWC linguistic indicators and personality: Exemplifying the lack of generalizability of exploratory studies. *psych, 4,* 803–816. https://doi.org/10.3390/psych4040059.

vamos a trabajar con cinco indicadores lingüísticos (llamados “WC”, “Yo”, “Verbos”, “Trabajo” y “Comma”, que representan el conteo de palabras, la utilización de la primera persona, la utilización de verbos, la utilización de palabras relacionadas con trabajo y la utilización de comas, respectivamente) y con un rasgo de personalidad (llamado “bfq_est”, que representa la puntuación de estabilidad emocional de los participantes en el test BFQ). Todas estas variables estarían recogidas en un objeto que hemos denominado `bbdd`. Para facilitar la comprensión de la sintaxis, hemos renombrado las variables del estudio. Concretamente, los cinco predictores lingüísticos han sido denominados `x1-x5` y la variable dependiente (el rasgo de personalidad) ha sido denominado `output`. Si utilizamos la función `cor(bbdd)` podremos obtener la correlación que existe entre todas las variables de la base de datos (ver Tabla 1).

Tabla 1. Tabla de correlaciones entre las variables seleccionadas.

	x2	**x3**	**x4**	**x5**	**output**
x1	-0.215	-0.002	-0.095	0.044	0.024
x2	1	0.213	-0.073	-0.013	-0.104
x3		1	-0.053	-0.074	-0.050
x4			1	-0.023	-0.037
x5				1	-0.035
output					1

Como cabría esperar, la correlación entre los indicadores individuales y la variable de personalidad no es especialmente grande. Cabe entonces preguntarse si podemos generar un modelo predictivo que nos permita predecir la variable output a partir de estos indicadores con un modelo de red neuronal.

Para poder trabajar con estas variables en un modelo de red neuronal, la primera recomendación sería estandarizar las variables (ya que cada una de ellas tiene una métrica completamente diferente). Para ello, utilizaremos la función `scale()` y podemos guardar la estandarización en nuestro objeto `bbdd` utilizando este código: `bbdd<-scale(bbdd)`. Como estamos trabajando con variables cuantitativas, este paso sería opcional, pero en algunos contextos es muy importante estandarizar las variables para poder trabajar en una misma métrica.

Una vez que tenemos nuestra base de datos preparada para llevar a cabo los análisis pertinentes, vamos a dividir manualmente (luego veremos otros procedimientos más recomendables) nuestros datos en un data set para entrenar nuestro modelo de redes neuronales (`trainset`) y un data set para testarlo (`testset`). Podemos utilizar la sintaxis que se presenta a continuación. En este ejemplo, estamos seleccionado los primeros 400 participantes para el entrenamiento, y los 242 restantes para el test.

```
trainset <- bbdd[1:400, ]

testset <- bbdd[401:642, ] # validación o test
```

La sintaxis que utilizamos para estimar modelos de redes neuronales es muy sencilla. En primer lugar, podemos observar que la función es `neuralnet()`. Esta función recibe como argumento la fórmula de las variables que queremos introducir en el modelo.

En este ejemplo, podemos ver que queremos predecir la variable output a partir de los cinco indicadores antes mencionados: `output ~ x1 + x2 + x3 + x4 + x5`. La manera de expresar esta fórmula es estándar en R para todo tipo de modelos (e.g., modelos de efectos mixtos, modelos de ecuaciones estructurales, etc.). Esta función también recibe como argumento la base de datos que queremos utilizar para ajustar el modelo: `data=trainset` (recordemos que estamos en la fase de entrenamiento). El siguiente argumento que se presenta en la siguiente sintaxis es el que permite definir la arquitectura que queremos establecer en este modelo concreto (`hidden`). Básicamente, en este argumento estamos incluyendo tantos escalares (valores) como capas ocultas queramos incluir en el modelo (sin tener en cuenta la capa de salida), y específicamente cada uno de esos escalares (valores) define el número de nodos ocultos que queremos incluir en esa capa. En este caso concreto, hemos escrito `hidden=c(2,1)` porque queremos estimar un modelo de red neuronal con dos capas ocultas donde la primera capa tiene dos nodos ocultos y la segunda capa tiene un único nodo oculto. El argumento `linear.output=T` estaría indicando que nuestra variable dependiente (`output`) es una variable continua y, por lo tanto, queremos estimar un modelo de red neuronal lineal. Finalmente, el argumento `threshold` estaría definiendo un valor concreto (umbral) para las derivadas parciales de la función de error como criterio de parada de la estimación.

```
nn <- neuralnet(output ~ x1 + x2 + x3 + x4 + x5,
      data=trainset, hidden=c(2,1),
      linear.output=T, threshold=0.05)
```

Una vez que hemos ejecutado la sintaxis anterior, ya habremos estimado nuestro modelo de red neuronal que, recordemos, habrá aprendido a predecir la variable `output` a partir de los predictores `x1-x5` de la mejor manera posible en la base de datos `trainset`. Ahora vamos a ver cómo podemos visualizar los resultados de esta red neuronal. Para ello, vamos a ejecutar la siguiente sintaxis:

```
nn$result.matrix

plot(nn)
```

Para empezar, mostramos los resultados de ejecutar la función `nn$result.matrix`. Este output nos muestra cada una de las estimaciones del modelo de red neuronal que acabamos de estimar para los distintos parámetros que hemos establecido en la arquitectura. La estructura de estos resultados es relativamente sencilla porque estamos ante un ejemplo muy sencillo, aunque lo veremos de manera mucho más clara en el gráfico que veremos a continuación. Interpretemos las estimaciones del modelo:

- Los valores `Intercept.to.1layhid1` e `Intercept.to.1layhid2` nos muestran el valor del sesgo para los nodos 1 y 2 de la primera capa oculta. De manera similar, el valor `Intercept.to.2layhid1` nos muestra el sesgo del nodo de la segunda capa oculta, e `Intercept.to.output` nos muestra el sesgo del nodo de la capa de salida.
- Los valores que tiene el formato `x1.to.1layhid1` o `1layhid1.to.2layhid1`, representan los pesos que relacionan los nodos de

las capas ocultas. Es relevante tener en cuenta que los pesos que relacionan los nodos de la capa de entrada con la primera capa oculta sí identifican de manera explícita que su origen es una determinada variable predictora (e.g., `x1` en `x1.to.1layhid1`), mientras que los pesos que relacionan los nodos de una capa oculta (e.g., capa 1) con otra capa oculta (e.g., capa 2) ya están relacionando nodos no observables (e.g., `1layhid1` y `2layhid1` en `1layhid1.to.2layhid1`).

- De manera similar, cabe destacar que la capa de salida tiene también su estimación del sesgo (`Intercept.to.output`) y del peso que relaciona el valor de la capa con el output observable (`2layhid1.to.output`). Aunque pudiéramos pensar que el valor de la capa oculta 2 es el output que estamos observando, realmente la relación entre las estimaciones de las capas ocultas y de la capa de salida es un poco más compleja (ya que este sesgo y este peso también se aprenden durante el proceso de entrenamiento).

Las interpretaciones de estas estimaciones son complicadas debido al arduo proceso de estimación de las redes neuronales, pero podemos considerar que estos pesos representan la fuerza de las asociaciones existentes entre los distintos nodos de la red. A modo de ejemplo, el predictor `x1` parece tener una mayor representatividad (peso) en el nodo 1 frente al nodo 2 de la primera capa oculta. Continuando con el ejemplo, podemos ver que el primer nodo oculto parece tener una mayor relación con el predictor `x3` y `x5` en

comparación con el predictor `x1`, `x2` y `x4`, por ejemplo[2]. Recomendamos a los lectores interesados en la interpretación de los pesos que recurran a leer sobre algunas técnicas ya clásicas como son los diagramas de Hinton[3] (en los modelos de redes neuronales clásicas, la "caja negra" ya no es tan incomprensible como se suele decir en ciertos círculos).

```
                                [,1]
error                  177.27344289
reached.threshold        0.04019341
steps                  968.00000000
Intercept.to.1layhid1    5.45153146
x1.to.1layhid1           1.30007477
x2.to.1layhid1          -3.01242999
x3.to.1layhid1          -9.68301490
x4.to.1layhid1           4.09291324
x5.to.1layhid1          -7.35458557
Intercept.to.1layhid2   -6.39455514
x1.to.1layhid2           0.37553615
x2.to.1layhid2          -1.97657464
```

[2] En la práctica habitual, no se suelen interpretar los pesos de las redes neuronales así, pero se hace este ejercicio para aprender. Además, se debe tener en cuenta que estas interpretaciones son posibles gracias a que hemos estandarizado los predictores del modelo, ya que todos estos predictores se encuentran en la misma métrica con la misma media y desviación típica (además, habría otras características de la distribución de las variables que podría ser interesante tener en cuenta).

[3] Bremner, F. J., Gotts, S. J., & Denham, D. L. (1994). Hinton diagrams: Viewing connection strengths in neural networks. *Behavior Research Methods, Instruments, & Computers, 26*, 215-218. https://doi.org/10.3758/BF03204624.

```
      x3.to.1layhid2          -0.13168305

      x4.to.1layhid2          -4.97614632

      x5.to.1layhid2           3.73917175

      Intercept.to.2layhid1   33.05762434

      1layhid1.to.2layhid1   -76.49303239

      1layhid2.to.2layhid1   -66.03664163

      Intercept.to.output      0.10446020

      2layhid1.to.output      -0.78136424
```

Utilizando la función `plot(nn)`, podemos tener una visión más clara de la arquitectura de la red estimada y observar exactamente la misma información en un gráfico.

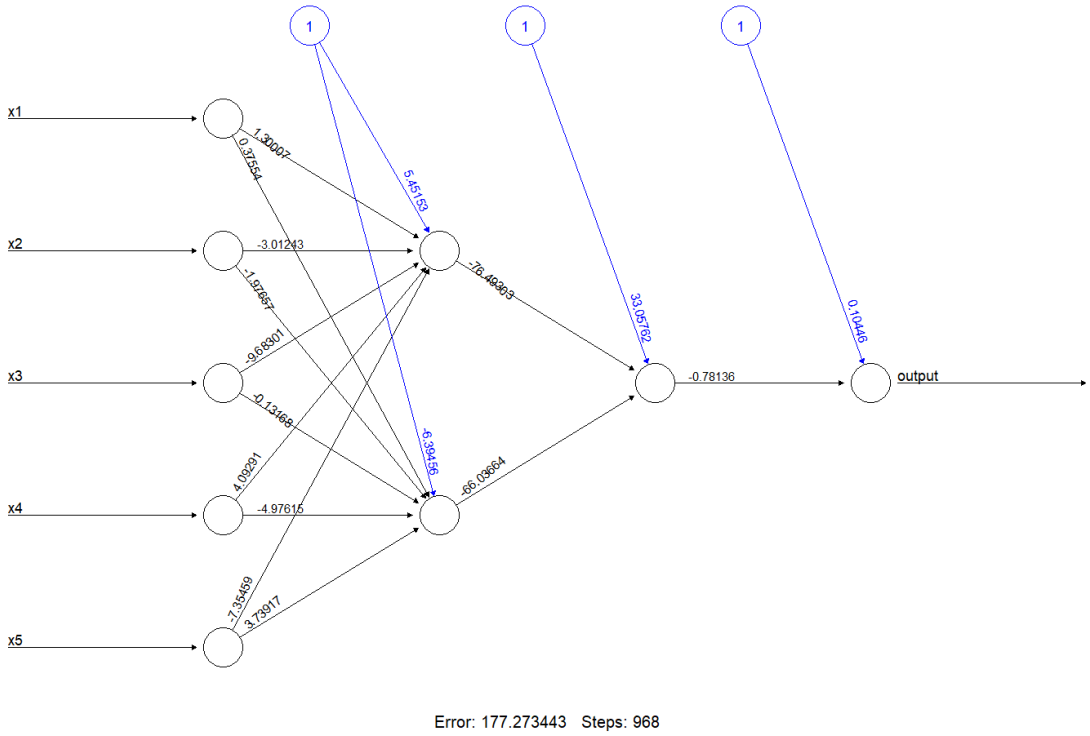

Figura 1. Red neuronal estimada.

Una vez que tenemos nuestro modelo estimado, queremos evaluar qué tal funciona para predecir nuevos casos. Para ello, vamos a utilizar el data set llamado `testset`. Aplicando la función `head(testset)` podemos ver que el formato del inicio de la base de datos es equivalente al del data set de entrenamiento.

```
           x1          x2          x3          x4          x5          output
[1,] -1.2586170 -0.83347561 -0.58379292  0.9412951 -0.5825650  0.40277200
[2,] -1.4201516  2.55459313  1.31434041 -0.1027190  0.1472218 -0.35389173
[3,]  1.3079888 -0.03351494  0.02924057 -0.4873557 -0.5375165  1.02218321
[4,] -0.2176161  0.32529095 -2.12772913 -0.2675633 -1.0150313 -0.07879506
[5,] -0.1518056 -0.12174589  0.85570054 -1.2173806 -0.5555359 -1.52349627
[6,]  0.6977469  0.04295189  0.60594616 -0.7463968 -0.2717299 -0.14742133
```

Como se habrán fijado, el modelo de redes neuronales que hemos entrenado se ha guardado en el objeto `nn`. Si ejecutamos el código `compute(nn, testset)`, estaremos utilizando los valores estimados por nuestro modelo de redes neuronales para obtener las predicciones correspondientes en el `testset`. Para facilitar la comprensión de esta sintaxis, estamos grabando las predicciones en un objeto llamado `nn.results`. La siguiente sintaxis nos permite grabar las predicciones de nuestro modelo junto a la variable dependiente original (aquella variable que queremos predecir) para los casos que componen el `testset`. Concretamente, con esta función estaríamos generando un data frame en R que se llamará `results`. En la primera columna de `results`, vamos a tener las puntuaciones originales del `testset` (que son las variables que queremos predecir). En la

segunda columna, vamos a tener las predicciones del modelo de red neuronal para el `testset`.

```
nn.results <- compute(nn, testset)
results <- data.frame(actual = testset[,6], prediction =
      nn.results$net.result)
```

Vamos a imprimir las cinco primeras filas del objeto `results` para ejemplificar los resultados del modelo.

```
          actual   prediction
1    0.40277200  0.137688913
2   -0.35389173 -0.617046090
3    1.02218321 -0.118855541
4   -0.07879506  0.504474859
5   -1.52349627 -0.118856041
```

Para poder visualizar los resultados, podríamos ejecutar la función `plot(results)` y obtendremos un gráfico de dispersión de puntos como el que se presenta en la Figura 2. En este caso concreto, se presentan las predicciones en el eje Y (i.e., *prediction*) y los valores a predecir en el eje X (i.e., *actual*). Podemos observar que,

aparentemente, el rendimiento del modelo no es muy bueno. Es decir, que las variables que hemos incluido como predictores de esta variable dependiente en esta arquitectura de red neuronal no parecen explicar adecuadamente los valores de la variable a predecir.

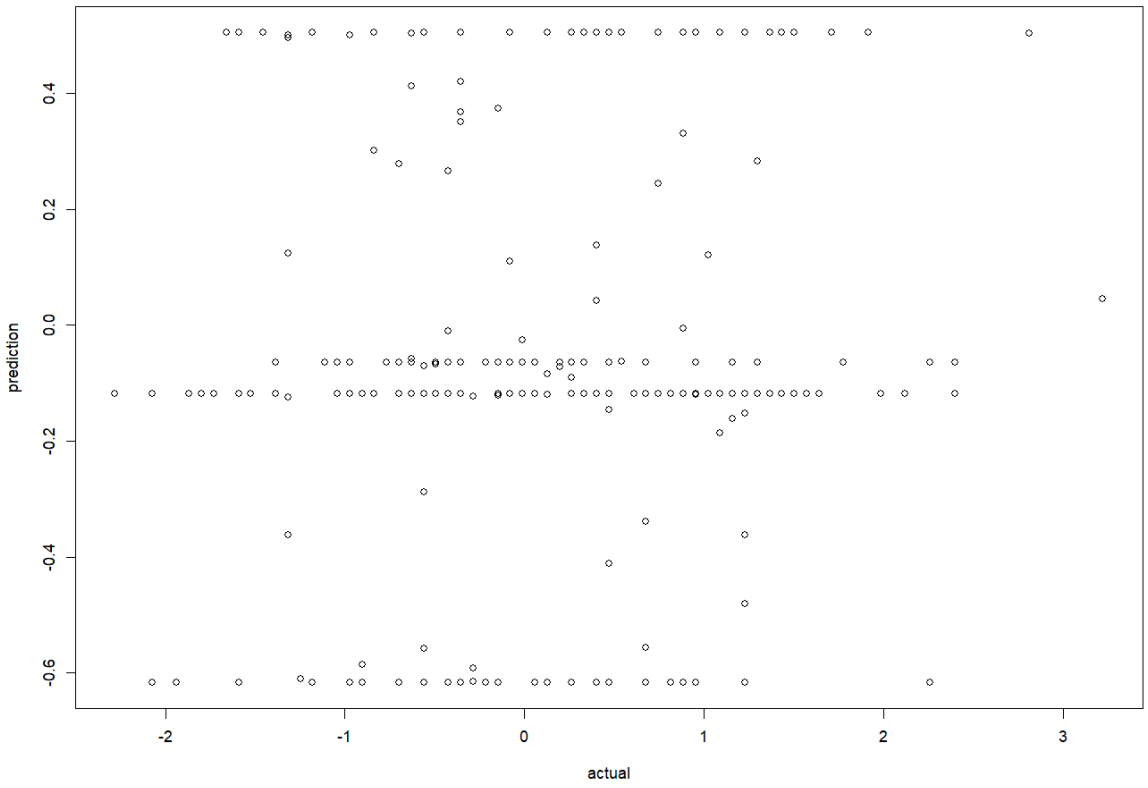

Figura 2. Gráfico de dispersión de puntos.

Continuando con esta ilustración, podemos incrementar la complejidad de la arquitectura de este modelo de redes neuronales manteniendo el mismo objetivo. En este caso concreto, queremos estimar un modelo de redes neuronales que tenga tres capas ocultas (sin tener en cuenta la capa de salida) y que éstas tengan 5, 1 y 5 nodos. Para ello, podríamos utilizar la siguiente sintaxis para grabar el modelo en el objeto `nn` utilizando la base de datos `trainset`. Esto daría lugar al modelo mostrado en la Figura 3.

```
nn <- neuralnet(output ~ x1 + x2 + x3 + x4 + x5,
      data=trainset, hidden=c(5,1,5), linear.output=T,
      threshold=0.05)
```

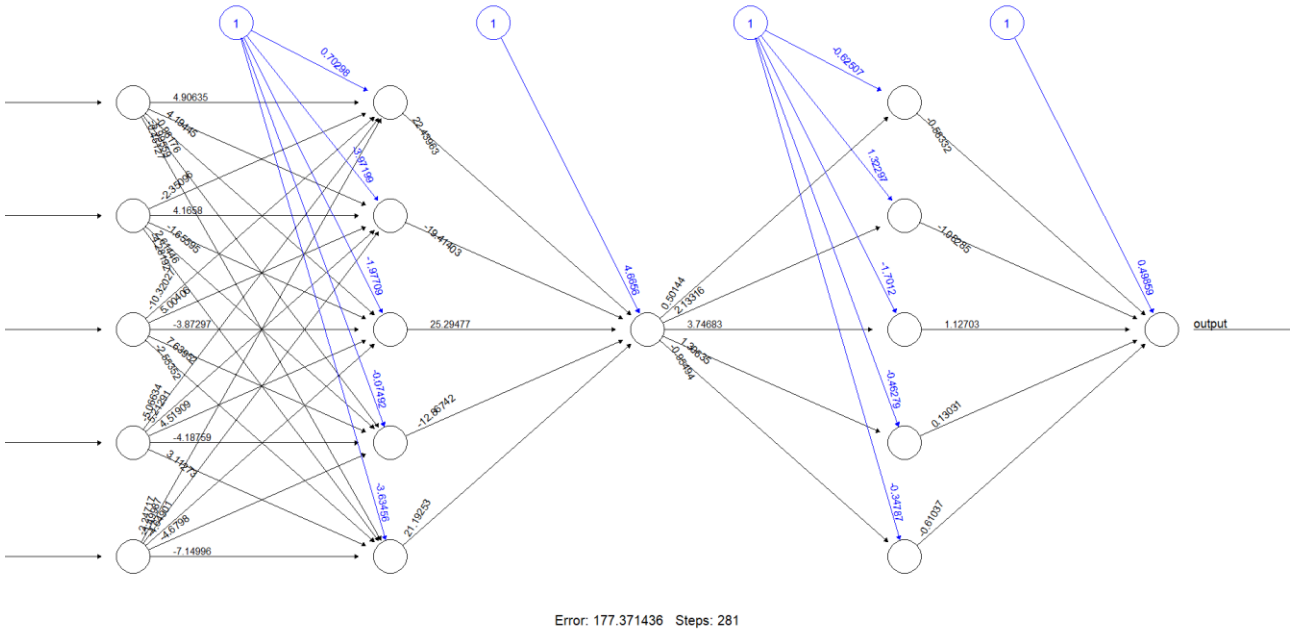

Figura 3. Red neuronal estimada.

INFORMACIÓN IMPORTANTE

En este documento hemos trabajado con modelos de redes neuronales clásicos desde el punto de vista del análisis de datos desde una perspectiva puramente exploratoria para facilitar la comprensión de su aplicación. Sin embargo, nosotros recomendamos la utilización de los modelos de redes neuronales desde el punto de vista del modelado formal de procesos psicológicos. Es decir, no recomendamos utilizar las redes neuronales únicamente como "cajas negras" que predicen un output, sino como una herramienta de trabajo más para el modelado de procesos psicológicos.