# PEC 2: Gene expression patterns of phenotypes subclasses using gene expression profiling and artificial neural networks

María Ajenjo Bauzá

15/12/2020

## Contents

# 1 Algoritmo Red Neuronal Artificial

## 1.1 Funcionamiento y características

Las redes neuronales artificiales se inspiran en las redes neuronales biológicas, las del cerebro. Así como el cerebro utiliza una red de neuronas interconectadas entre sí, el algoritmo ANN utiliza una red de neuronas artificiales, también denominadas "nodos", para resolver problemas de aprendizaje.

Cada neurona tiene conexiones de entrada, a través de las que recibe valores de entrada. Con estos valores, la neurona realizará un cálculo y generará un valor de salida. Por tanto, una neurona es como una función matemática. Esa función consiste en una suma ponderada de los valores de entrada, donde la ponderación correrá a cargo de los pesos asignados a cada variable o parámetros de nuestro modelo.

La red neuronal se crea mediante diferente capas interconectadas. De esta manera se logra un aprendizaje jerarquizado. En cada capa hay neuronas que reciben la información de la capa anterior, la procesan, y la transmiten a la capa siguiente. A la primera capa se le denomina "capa de entrada", a la última, "capa de salida", y a las intermedias, "capas ocultas".

**CARACTERÍSTICAS DE LAS REDES NEURONALES ARTIFICIALES:**

- Algoritmo de entrenamiento: Establece cómo se distribuyen los pesos de las conexiones para inhibir o estimulas las neuronas en proporción de la señal de entrada.
- Topología o arquitectura de la red: número de capas y nodos, así como la dirección en la que se transmite la información dentro de las capas o entre capas.
- Función de activación: distorsiona el valor de salida, añadiéndole deformaciones no lineales. Hay diferentes tipos de funciones de activación:
  - Función escalonada: para un valor de entrada mayor al umbral el output es 1. Para un valor inferior, será igual a 0.
  - Función sigmoide: hace que los valores muy grandes se saturen en 1 y que los muy pequeños se saturen en 0.
  - Función Tanh: similar a la sigmoide pero con un rango de -1 a 1.
  - Función RELU: Unidad Rectificada Lineal. Se comporta como una función lineal positiva cuando el valor de entrada es positivo y como constante a 0 cuando el valor de entrada es negativo.

## 1.2 Tabla de fortalezas y debilidades

Table 1: Strengths and Weaknesses of the Artificial Neural Network algorithm

| Fortalezas | Debilidades |
|---|---|
| Adaptable a clasificación o problemas de predicción numérica | Requiere de gran potencia computacional y en general es de aprendizaje lento, particularmente si la topología es compleja |
| Capaz de modelar patrones más complejos que casi cualquier otro algoritmo | Propenso a sobreajustar los datos de entrenamiento |
| No necesita muchas restricciones acerca de las relaciones subyacentes de los datos | Es un modelo de caja negra complejo que es difícil, si no imposible, de interpretar. |

## 1.3 Implementación del algoritmo

```
# Lectura de los ficheros de partida
class <- read.csv(file = file.path(params$folder.data, params$file.class))
data <- read.csv(file = file.path(params$folder.data, params$file.data))
head(data)[,1:6]
```

```
##            V1          V2          V3         V4         V5           V6
## 1  0.10172920  0.59423550 -1.12635568  0.9723793 -0.2171365 -0.370574508
## 2  0.12897473 -0.33259945  0.91374049  0.6013183 -0.2171365 -0.006958959
## 3  0.74419297 -0.33259945  0.17397538 -0.2795754 -0.2171365  0.031994203
## 4  0.48339568  0.68787110 -2.11496708  0.3055419 -0.2171365  0.115864874
## 5 -0.03216349 -0.07408875 -0.02699948 -0.2501134 -0.2171365  0.405275555
## 6  1.27170501 -0.33259945  1.48865401  0.7502087 -0.2171365 -0.699182804
```

### 1.3.1 Análisis de Componentes Principales o PCA

En primer lugar, exploraremos los datos.

```
# Exploración de los datos
dim(data)
```

```
## [1]  102 5506
```

```
# Media de la expresión de cada gen (muestra de los 10 primeros).
# (MARGIN = 2 para que se aplique la función a las columnas)
apply(X = data, MARGIN = 2, FUN = mean)[1:10]
```

```
##           V1           V2           V3           V4           V5           V6
## -0.010837100 -0.019573348 -0.022971285 -0.014820683  0.002128789  0.021826720
##           V7           V8           V9          V10
## -0.017818765  0.008242983  0.001263431 -0.008287230
```

```
# Varianza de la expresión de cada gen (muestra de los 10 primeros)
apply(X = data, MARGIN = 2, FUN = var)[1:10]
```

```
##         V1         V2         V3         V4         V5         V6         V7
## 0.13915773 0.22074686 1.09335007 0.20684945 0.29483689 0.22178778 0.18339033
##         V8         V9        V10
## 0.26437904 0.09066323 0.62855988
```

Posteriormente, hemos de estandarizar las variables para que tengan desviación estándar de 1 y media de 0 y aplicar el PCA. Con el parámetro scale = TRUE de la función prcomp() indicamos que queremos escalar las variables para que tengan desviación estándar de 1.

```
data_pca <- prcomp(data, scale = TRUE)
head(data_pca$rotation)[, 1:5]
```

```
##             PC1          PC2           PC3           PC4          PC5
## V1 -0.009586979  0.035586731 -0.0162914487 -0.0004262009 -0.009823806
## V2 -0.017132197  0.007035199  0.0002991786 -0.0050140778 -0.001264078
## V3 -0.003042461 -0.004995705 -0.0135563876 -0.0140423035  0.027107233
## V4 -0.020792156  0.002621273 -0.0084894110  0.0017120682  0.005332373
## V5 -0.006312430 -0.008128174 -0.0005817980 -0.0130820819  0.006673765
## V6  0.013745178 -0.007584544  0.0115878940  0.0126923878 -0.004853183
```

```
dim(data_pca$rotation)
```

```
## [1] 5506  102
```

Hay 102 componentes principales distintas. Observamos el vector de los scores y la desviación estándar de cada componente principal:

```
# Scores
head(data_pca$x)[, 1:5]
```

```
##             PC1      PC2        PC3       PC4        PC5
## [1,] -40.210185 19.23744  13.885727 10.110266   6.682348
## [2,] -36.921936 14.37209  -4.903639 26.651474  30.377865
## [3,] -27.590151 40.08005  -1.768691 19.150609   3.200707
## [4,] -19.419953 32.17562  16.248369 17.559142 -11.279857
## [5,]   1.390606 11.47379  21.910047  9.583781 -16.907643
## [6,] -46.621655 62.94365 -11.829080 -1.732755  -8.525425
```

```
# Desviación estándar
data_pca$sdev
```

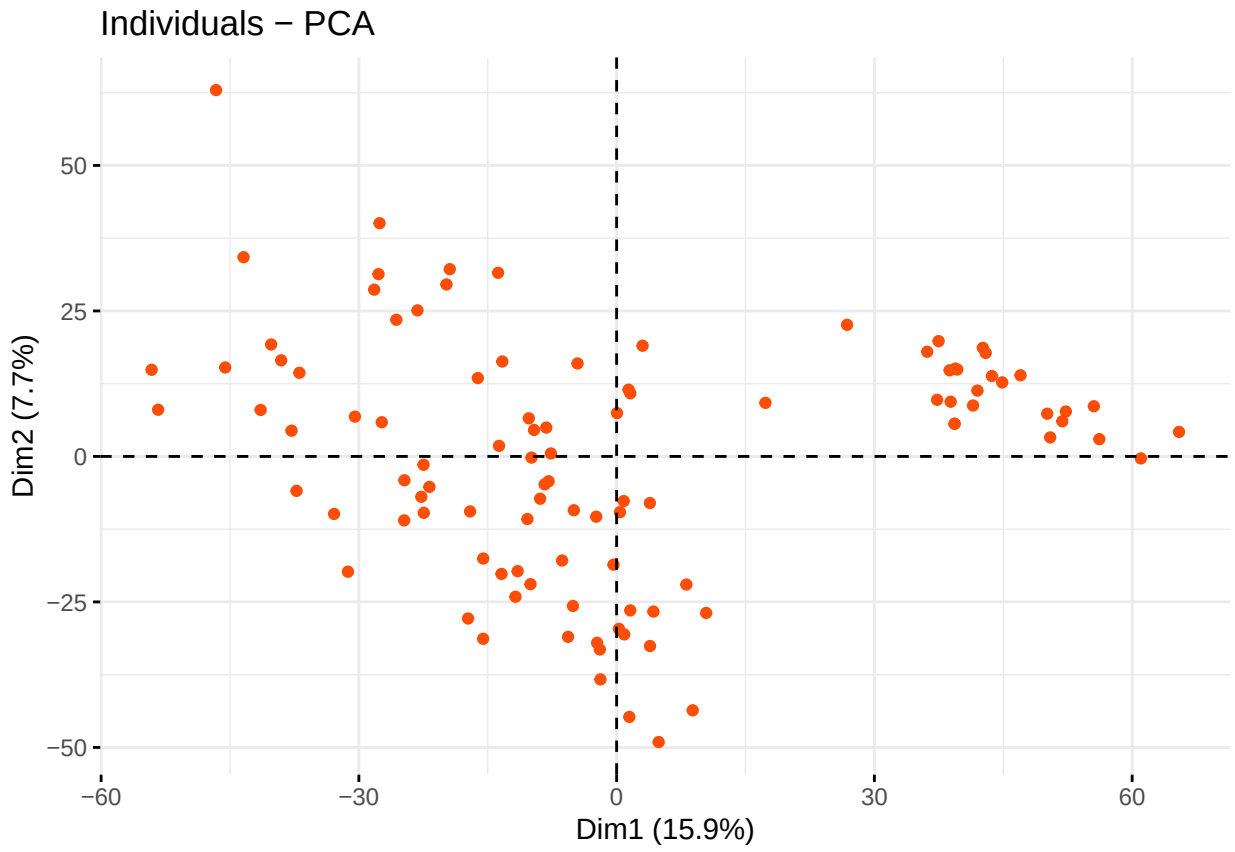
```
##   [1] 2.955867e+01 2.055621e+01 1.852980e+01 1.715335e+01 1.389642e+01
##   [6] 1.158840e+01 1.104160e+01 1.092238e+01 1.023150e+01 9.840876e+00
##  [11] 9.505690e+00 9.137285e+00 8.665456e+00 8.460454e+00 8.305310e+00
##  [16] 7.961401e+00 7.808630e+00 7.681629e+00 7.543959e+00 7.316296e+00
##  [21] 7.115128e+00 7.060800e+00 6.988537e+00 6.924282e+00 6.805358e+00
##  [26] 6.774266e+00 6.564867e+00 6.543767e+00 6.371217e+00 6.355354e+00
##  [31] 6.272482e+00 6.244309e+00 6.186007e+00 6.079629e+00 6.055186e+00
##  [36] 6.015908e+00 5.874652e+00 5.857791e+00 5.809276e+00 5.749216e+00
##  [41] 5.716554e+00 5.703445e+00 5.629194e+00 5.617234e+00 5.526026e+00
##  [46] 5.484521e+00 5.465560e+00 5.454085e+00 5.434961e+00 5.376265e+00
##  [51] 5.327046e+00 5.290513e+00 5.254826e+00 5.236592e+00 5.194609e+00
##  [56] 5.161901e+00 5.149500e+00 5.086373e+00 5.075484e+00 5.009619e+00
##  [61] 4.995136e+00 4.967619e+00 4.932654e+00 4.889227e+00 4.853164e+00
##  [66] 4.835426e+00 4.808668e+00 4.747228e+00 4.730308e+00 4.693657e+00
##  [71] 4.666636e+00 4.613927e+00 4.599579e+00 4.561862e+00 4.547517e+00
##  [76] 4.505508e+00 4.490840e+00 4.472626e+00 4.441162e+00 4.392766e+00
##  [81] 4.378469e+00 4.339464e+00 4.245965e+00 4.191312e+00 4.141484e+00
##  [86] 4.100275e+00 4.049426e+00 3.985776e+00 3.868419e+00 3.836812e+00
##  [91] 3.767020e+00 3.698406e+00 3.612251e+00 3.474092e+00 3.412885e+00
##  [96] 3.342070e+00 3.307123e+00 3.185979e+00 3.057477e+00 3.005014e+00
## [101] 1.731371e-14 6.829275e-15
```

```r
summary(data_pca)
```

```
## Importance of components:
##                             PC1      PC2      PC3      PC4      PC5      PC6
## Standard deviation     29.5587 20.55621 18.52980 17.15335 13.89642 11.58840
## Proportion of Variance  0.1587  0.07674  0.06236  0.05344  0.03507  0.02439
## Cumulative Proportion   0.1587  0.23543  0.29779  0.35123  0.38630  0.41069
##                             PC7      PC8      PC9    PC10    PC11    PC12
## Standard deviation     11.04160 10.92238 10.23150 9.84088 9.50569 9.13729
## Proportion of Variance  0.02214  0.02167  0.01901 0.01759 0.01641 0.01516
## Cumulative Proportion   0.43283  0.45450  0.47351 0.49110 0.50751 0.52268
##                            PC13   PC14    PC15    PC16    PC17    PC18    PC19
## Standard deviation     8.66546 8.4605 8.30531 7.96140 7.80863 7.68163 7.54396
## Proportion of Variance 0.01364 0.0130 0.01253 0.01151 0.01107 0.01072 0.01034
## Cumulative Proportion  0.53631 0.5493 0.56184 0.57335 0.58443 0.59515 0.60548
##                            PC20    PC21    PC22    PC23    PC24    PC25    PC26
## Standard deviation     7.31630 7.11513 7.06080 6.98854 6.92428 6.80536 6.77427
## Proportion of Variance 0.00972 0.00919 0.00905 0.00887 0.00871 0.00841 0.00833
## Cumulative Proportion  0.61520 0.62440 0.63345 0.64232 0.65103 0.65944 0.66778
##                            PC27    PC28    PC29    PC30    PC31    PC32    PC33
## Standard deviation     6.56487 6.54377 6.37122 6.35535 6.27248 6.24431 6.18601
## Proportion of Variance 0.00783 0.00778 0.00737 0.00734 0.00715 0.00708 0.00695
## Cumulative Proportion  0.67560 0.68338 0.69075 0.69809 0.70523 0.71232 0.71927
##                            PC34    PC35    PC36    PC37    PC38    PC39   PC40
## Standard deviation     6.07963 6.05519 6.01591 5.87465 5.85779 5.80928 5.7492
## Proportion of Variance 0.00671 0.00666 0.00657 0.00627 0.00623 0.00613 0.0060
## Cumulative Proportion  0.72598 0.73264 0.73921 0.74548 0.75171 0.75784 0.7638
##                            PC41    PC42    PC43    PC44    PC45    PC46    PC47
## Standard deviation     5.71655 5.70344 5.62919 5.61723 5.52603 5.48452 5.46556
## Proportion of Variance 0.00594 0.00591 0.00576 0.00573 0.00555 0.00546 0.00543
## Cumulative Proportion  0.76978 0.77569 0.78144 0.78717 0.79272 0.79818 0.80361
##                            PC48    PC49    PC50    PC51    PC52    PC53    PC54
## Standard deviation     5.4541 5.43496 5.37626 5.32705 5.29051 5.25483 5.23659
## Proportion of Variance 0.0054 0.00536 0.00525 0.00515 0.00508 0.00502 0.00498
## Cumulative Proportion  0.8090 0.81438 0.81962 0.82478 0.82986 0.83488 0.83986
##                           PC55    PC56    PC57   PC58    PC59    PC60    PC61
## Standard deviation     5.1946 5.16190 5.14950 5.0864 5.07548 5.00962 4.99514
## Proportion of Variance 0.0049 0.00484 0.00482 0.0047 0.00468 0.00456 0.00453
## Cumulative Proportion  0.8448 0.84960 0.85441 0.8591 0.86379 0.86835 0.87288
##                            PC62    PC63    PC64    PC65    PC66   PC67    PC68
## Standard deviation     4.96762 4.93265 4.88923 4.85316 4.83543 4.8087 4.74723
## Proportion of Variance 0.00448 0.00442 0.00434 0.00428 0.00425 0.0042 0.00409
## Cumulative Proportion  0.87736 0.88178 0.88612 0.89040 0.89465 0.8989 0.90294
##                            PC69  PC70    PC71    PC72    PC73    PC74    PC75
## Standard deviation     4.73031 4.694 4.66664 4.61393 4.59958 4.56186 4.54752
## Proportion of Variance 0.00406 0.004 0.00396 0.00387 0.00384 0.00378 0.00376
## Cumulative Proportion  0.90700 0.911 0.91496 0.91883 0.92267 0.92645 0.93020
##                            PC76    PC77    PC78    PC79   PC80    PC81    PC82
## Standard deviation     4.50551 4.49084 4.47263 4.44116 4.3928 4.37847 4.33946
## Proportion of Variance 0.00369 0.00366 0.00363 0.00358 0.0035 0.00348 0.00342
## Cumulative Proportion  0.93389 0.93755 0.94119 0.94477 0.9483 0.95176 0.95518
##                            PC83    PC84    PC85    PC86    PC87    PC88    PC89
## Standard deviation     4.24596 4.19131 4.14148 4.10028 4.04943 3.98578 3.86842
```

```
## Proportion of Variance 0.00327 0.00319 0.00312 0.00305 0.00298 0.00289 0.00272
## Cumulative Proportion  0.95845 0.96164 0.96476 0.96781 0.97079 0.97367 0.97639
##                           PC90    PC91    PC92    PC93    PC94    PC95    PC96
## Standard deviation     3.83681 3.76702 3.69841 3.61225 3.47409 3.41288 3.34207
## Proportion of Variance 0.00267 0.00258 0.00248 0.00237 0.00219 0.00212 0.00203
## Cumulative Proportion  0.97906 0.98164 0.98413 0.98650 0.98869 0.99080 0.99283
##                           PC97    PC98   PC99   PC100     PC101     PC102
## Standard deviation     3.30712 3.18598 3.0575 3.00501 1.731e-14 6.829e-15
## Proportion of Variance 0.00199 0.00184 0.0017 0.00164 0.000e+00 0.000e+00
## Cumulative Proportion  0.99482 0.99666 0.9984 1.00000 1.000e+00 1.000e+00
```

```r
fviz_pca_ind(data_pca, geom.ind = "point",
             col.ind = "#FC4E07",
             axes = c(1, 2),
             pointsize = 1.5)
```



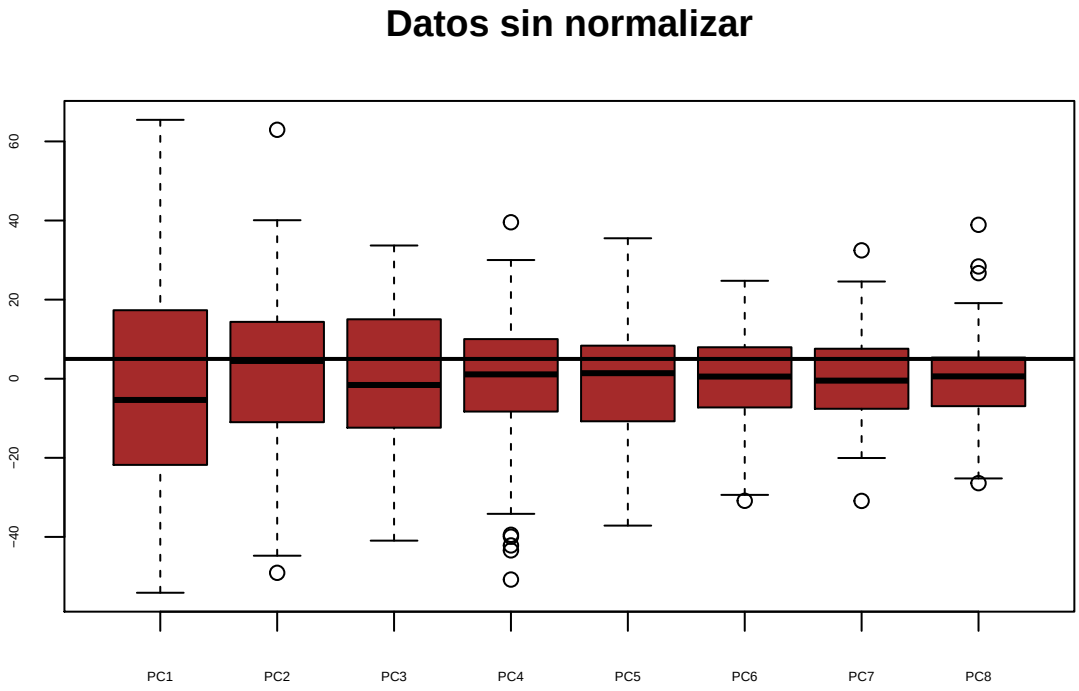

Por último, pasaremos a un nuevo dataframe los datos que hemos obtenido tras el PCA.

```r
pca_data <- data_pca$x
pca_data <- as.data.frame(pca_data)
```

### 1.3.2 Normalización de las variables

En primer lugar observamos algunas de las primeras componentes del análisis, en concreto las 8 primeras.

```r
# Observamos las 8 primeras componentes principales
boxplot(pca_data[,1:8],
        main='Datos sin normalizar',
        col='brown',cex.axis=0.4,subset=pca_data)
abline(h=5,lwd=2)
```




Hay que normalizar las variables para que tomen valores entre 0 y 1. Definimos la función normalize() para realizar esta operación.

```r
# Definimos función de normalización
normalize <- function(x) {
return((x - min(x)) / (max(x) - min(x)))
}

# Normalizamos los datos
data_norm <- as.data.frame(lapply(pca_data,normalize))
```

Confirmamos que el rango de valores esta entre 0 y 1.

```r
summary(data_norm)[,1:8]
```

```
##       PC1               PC2               PC3               PC4
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.2745   1st Qu.:0.3405   1st Qu.:0.3828   1st Qu.:0.4750
##  Median :0.4077   Median :0.4782   Median :0.5274   Median :0.5743
```

```
##  Mean   :0.4526   Mean   :0.4381   Mean   :0.5486   Mean   :0.5621
##  3rd Qu.:0.5830   3rd Qu.:0.5655   3rd Qu.:0.7498   3rd Qu.:0.6723
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##       PC5              PC6              PC7              PC8
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.3651   1st Qu.:0.4249   1st Qu.:0.3683   1st Qu.:0.3022
##  Median :0.5305   Median :0.5647   Median :0.4802   Median :0.4133
##  Mean   :0.5112   Mean   :0.5550   Mean   :0.4877   Mean   :0.4042
##  3rd Qu.:0.6260   3rd Qu.:0.6979   3rd Qu.:0.6028   3rd Qu.:0.4852
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
```

## 1.4 Separación de los datos en train y test

Antes de partir los datos en train y test, añadiremos los datos del fichero class6.csv al dataframe con el que estamos trabajando. Para ello creamos 4 variables binarias, una para cada uno de los fenotipos marcados en el fichero class6.csv.

```
# Creamos variables binarias para cada categoría
data_ann <- cbind(data_norm, class)
data_ann$FNT1 <- class$x == 1
data_ann$FNT2 <- class$x == 2
data_ann$FNT3 <- class$x == 3
data_ann$FNT4 <- class$x == 4
data_ann$x <- factor(data_ann$x, levels = c(1,2,3,4),
                labels = c("FNT1","FNT2","FNT3","FNT4"))
```

Dado que la semilla aleatoria (12345) ha sido indicada en los parámetros de este archivo, separaremos ahora los datos en dos grupos, el 67% (2/3) de los datos irá para el entrenamiento del modelo (train) y el otro 33% (1/3) para probar el modelo (test).

```
# n será el número de filas del conjunto total de datos
n <- nrow(data_ann)

# Fijamos la semilla de aleatoriedad
set.seed(params$seed.train)

# Partimos los datos en train y test
# n_train = 2/3 = params$p.train
train <- sample(n,floor(n*params$p.train))
data_ann.train <- data_ann[train,]
data_ann.test  <- data_ann[-train,]
```

```
# Comprobamos que hemos partido bien los datos
dim(data_ann.train)
```

```
## [1]  68 107
```

```
dim(data_ann.test)
```

```
## [1]  34 107
```

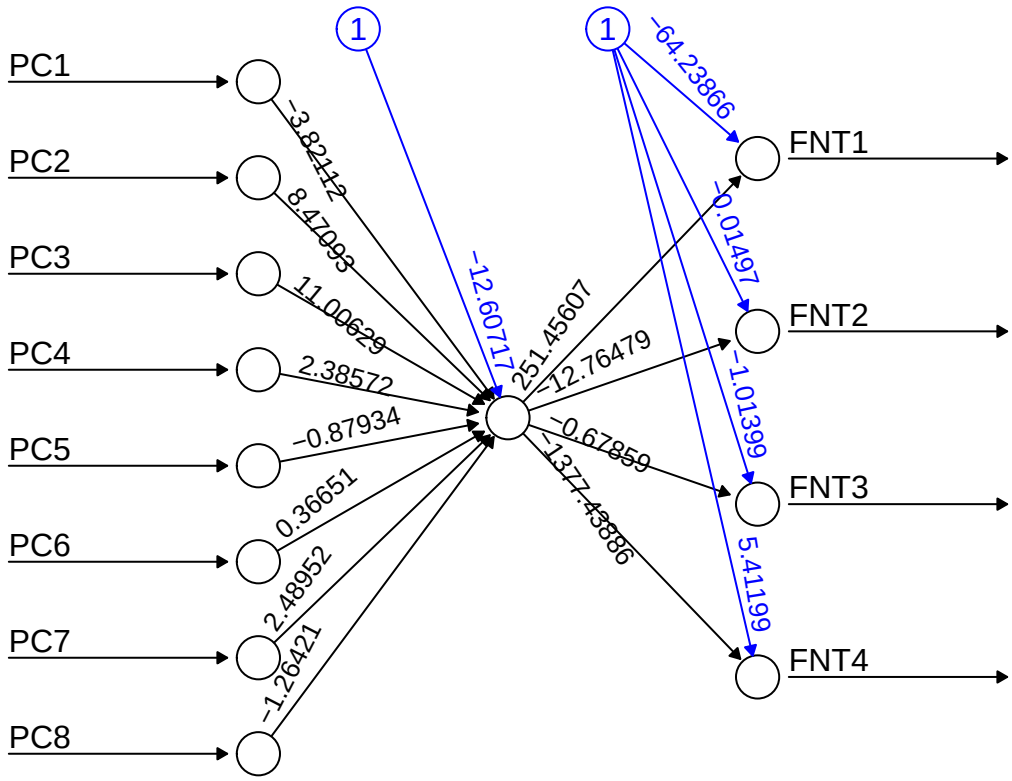
## 1.5 Modelos de red neuronal artificial de una sola capa oculta

Antes de la creación de los modelos, fijaremos la semilla generadora:

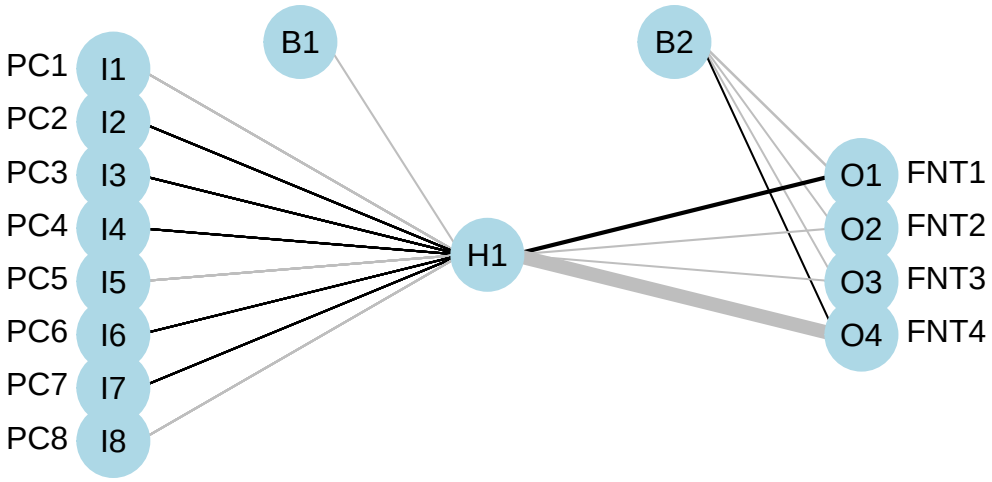
```
set.seed(params$seed.clsfier)
```

### 1.5.1 De un nodo

```
# especificamos la fórmula
formula_ann <- FNT1+FNT2+FNT3+FNT4 ~ PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8
# construimos el modelo
model_ann1 <- neuralnet(formula_ann,
                        data = data_ann.train, hidden = 1,
                        linear.output = FALSE)
# visualizamos el modelo
plot(model_ann1, rep = "best")
```




```
# visualizamos con el paquete NeuralNetTools
plotnet(model_ann1)
```




## PREDICCIÓN Y EVALUACIÓN DEL MODELO

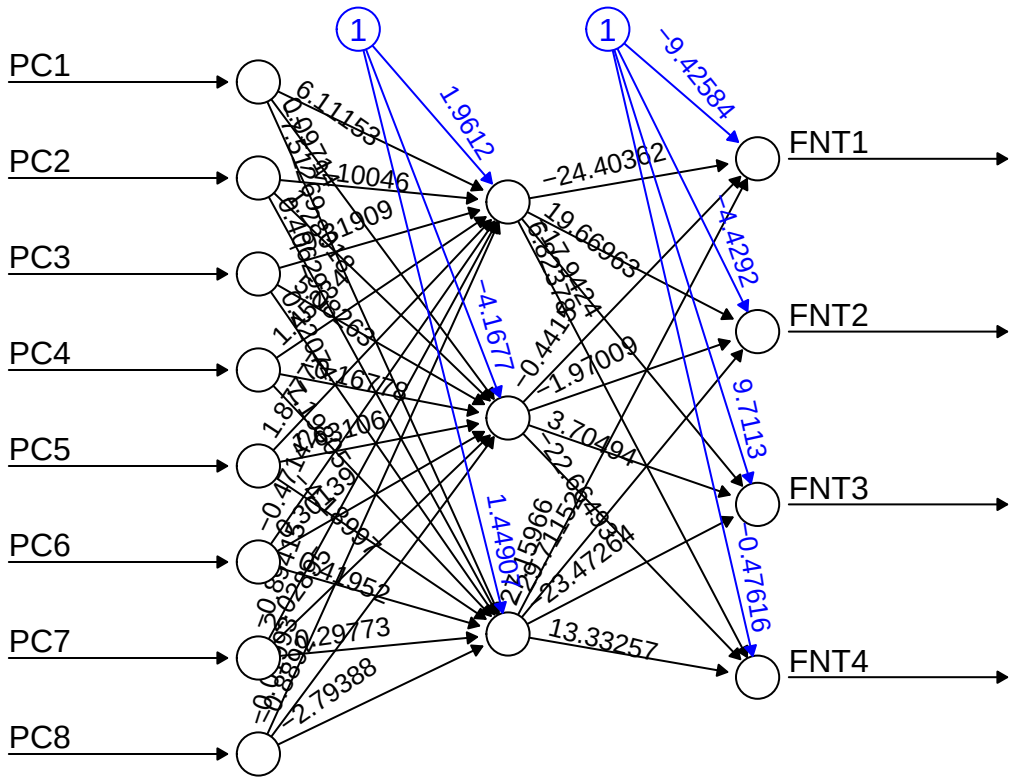
```
model_ann1_results <- neuralnet::compute(model_ann1,
                                         data_ann.test[,1:8])$net.result

# Transformamos el output binario a categórico
maxidx <- function(arr){
  return(which(arr == max(arr)))
}

idx <- apply(model_ann1_results[,-107], 1, maxidx)
prediction_1 <- c("FNT1","FNT2","FNT3","FNT4")[idx]
results1 <- table(prediction_1, data_ann.test$x)

# Confusion matrix
cmatrix1 <- confusionMatrix(results1)
cmatrix1
```

```
## Confusion Matrix and Statistics
##
##
## prediction_1 FNT1 FNT2 FNT3 FNT4
##         FNT1    6    0    4    0
##         FNT2    0    8    3    0
##         FNT3    0    0    5    0
##         FNT4    0    1    0    7
```

```
##
## Overall Statistics
##
##                Accuracy : 0.7647
##                  95% CI : (0.5883, 0.8925)
##     No Information Rate : 0.3529
##     P-Value [Acc > NIR] : 1.151e-06
##
##                   Kappa : 0.6913
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: FNT1 Class: FNT2 Class: FNT3 Class: FNT4
## Sensitivity               1.0000      0.8889      0.4167      1.0000
## Specificity               0.8571      0.8800      1.0000      0.9630
## Pos Pred Value            0.6000      0.7273      1.0000      0.8750
## Neg Pred Value            1.0000      0.9565      0.7586      1.0000
## Prevalence                0.1765      0.2647      0.3529      0.2059
## Detection Rate            0.1765      0.2353      0.1471      0.2059
## Detection Prevalence      0.2941      0.3235      0.1471      0.2353
## Balanced Accuracy         0.9286      0.8844      0.7083      0.9815
```

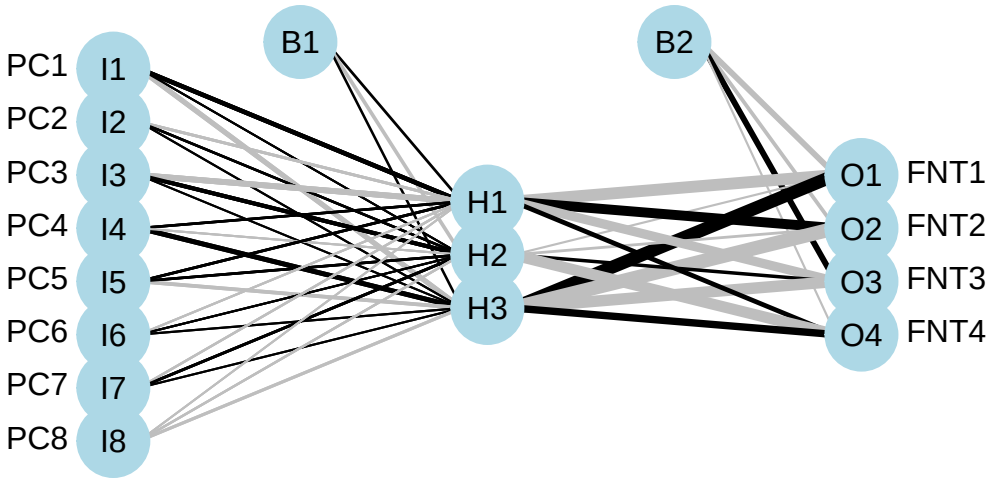
### 1.5.2 De tres nodos

```r
# construimos el modelo
model_ann3 <- neuralnet(formula_ann,
                        data = data_ann.train, hidden = 3,
                        linear.output = FALSE)
# visualizamos el modelo
plot(model_ann3, rep = "best")
```




Error: 0.039741 Steps: 337

```
# visualizamos con el paquete NeuralNetTools
plotnet(model_ann3)
```




### PREDICCIÓN Y EVALUACIÓN DEL MODELO

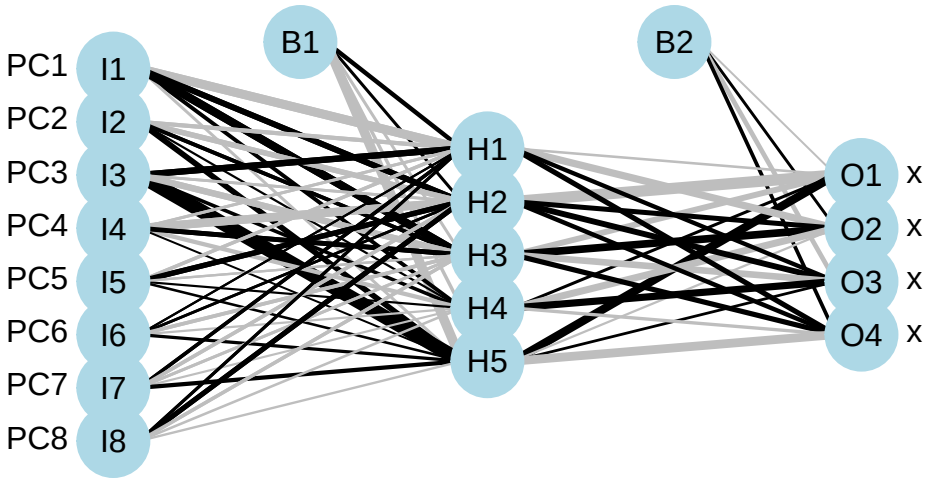
```
model_ann3_results <- neuralnet::compute(model_ann3,
                                         data_ann.test[,1:8])$net.result

# Transformamos el output binario a categórico con la función de antes
idx <- apply(model_ann3_results[,-107], 1, maxidx)
prediction_3 <- c("FNT1","FNT2","FNT3","FNT4")[idx]
results3 <- table(prediction_3, data_ann.test$x)

# Confusion matrix
cmatrix3 <- confusionMatrix(results3)
cmatrix3
```

```
## Confusion Matrix and Statistics
##
##
## prediction_3 FNT1 FNT2 FNT3 FNT4
##         FNT1    6    0    1    0
##         FNT2    0    9    0    0
##         FNT3    0    0   11    0
##         FNT4    0    0    0    7
##
## Overall Statistics
##
##                Accuracy : 0.9706
```

```
##                  95% CI : (0.8467, 0.9993)
##     No Information Rate : 0.3529
##     P-Value [Acc > NIR] : 2.652e-14
##
##                   Kappa : 0.9601
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: FNT1 Class: FNT2 Class: FNT3 Class: FNT4
## Sensitivity               1.0000      1.0000      0.9167      1.0000
## Specificity               0.9643      1.0000      1.0000      1.0000
## Pos Pred Value            0.8571      1.0000      1.0000      1.0000
## Neg Pred Value            1.0000      1.0000      0.9565      1.0000
## Prevalence                0.1765      0.2647      0.3529      0.2059
## Detection Rate            0.1765      0.2647      0.3235      0.2059
## Detection Prevalence      0.2059      0.2647      0.3235      0.2059
## Balanced Accuracy         0.9821      1.0000      0.9583      1.0000
```

### 1.6 Modelo nnet del paquete caret

```
model_nnet <- train(x ~ PC1+PC2+PC3+PC4+PC5+PC6+PC7+PC8,
                    data = data_ann.train,
                    method = "nnet",
                    trControl = trainControl(method = "cv", number = 3),
                    tuneGrid = NULL, tuneLength = 3, trace = FALSE)
plotnet(model_nnet)
```




```
summary(model_nnet)
```

```
## a 8-5-4 network with 69 weights
## options were - softmax modelling
##  b->h1 i1->h1 i2->h1 i3->h1 i4->h1 i5->h1 i6->h1 i7->h1 i8->h1
##   9.58 -26.61  -8.24  16.80  -3.67  -5.67   0.78   6.20   4.30
##  b->h2 i1->h2 i2->h2 i3->h2 i4->h2 i5->h2 i6->h2 i7->h2 i8->h2
##   4.24  14.92 -13.29  -3.02 -25.04  12.97   2.93  -6.24  11.92
##  b->h3 i1->h3 i2->h3 i3->h3 i4->h3 i5->h3 i6->h3 i7->h3 i8->h3
##  -3.99  23.81   7.24 -16.87  11.10  -1.26  -6.54  -3.70  -5.73
##  b->h4 i1->h4 i2->h4 i3->h4 i4->h4 i5->h4 i6->h4 i7->h4 i8->h4
##  -5.80   7.60   0.99   6.23  -8.98   0.75  -0.93  -0.70  -2.37
##  b->h5 i1->h5 i2->h5 i3->h5 i4->h5 i5->h5 i6->h5 i7->h5 i8->h5
## -29.66  -3.86  14.39  46.88   0.04   2.28   4.88   8.77  -1.52
##  b->o1 h1->o1 h2->o1 h3->o1 h4->o1 h5->o1
##  -0.03  -2.19 -36.41 -15.31   4.07  24.91
##  b->o2 h1->o2 h2->o2 h3->o2 h4->o2 h5->o2
##   3.82 -20.59  11.74  22.20 -19.72  -1.03
##  b->o3 h1->o3 h2->o3 h3->o3 h4->o3 h5->o3
## -10.62   9.18  15.27 -18.07  20.21   3.00
##  b->o4 h1->o4 h2->o4 h3->o4 h4->o4 h5->o4
##   6.82  12.54   9.85  12.54  -5.21 -26.56
```

```
prediction_nnet <- predict(model_nnet, data_ann.test)
results_nnet <- table(prediction_nnet, data_ann.test$x)
```

```
results_nnet
```

```
##
## prediction_nnet FNT1 FNT2 FNT3 FNT4
##            FNT1    6    0    1    0
##            FNT2    0    9    0    0
##            FNT3    0    0   11    0
##            FNT4    0    0    0    7
```

```
cmatrixnet <- confusionMatrix(results_nnet)
cmatrixnet
```

```
## Confusion Matrix and Statistics
##
##
## prediction_nnet FNT1 FNT2 FNT3 FNT4
##            FNT1    6    0    1    0
##            FNT2    0    9    0    0
##            FNT3    0    0   11    0
##            FNT4    0    0    0    7
##
## Overall Statistics
##
##                Accuracy : 0.9706
##                  95% CI : (0.8467, 0.9993)
##     No Information Rate : 0.3529
##     P-Value [Acc > NIR] : 2.652e-14
##
##                   Kappa : 0.9601
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: FNT1 Class: FNT2 Class: FNT3 Class: FNT4
## Sensitivity               1.0000      1.0000      0.9167      1.0000
## Specificity               0.9643      1.0000      1.0000      1.0000
## Pos Pred Value            0.8571      1.0000      1.0000      1.0000
## Neg Pred Value            1.0000      1.0000      0.9565      1.0000
## Prevalence                0.1765      0.2647      0.3529      0.2059
## Detection Rate            0.1765      0.2647      0.3235      0.2059
## Detection Prevalence      0.2059      0.2647      0.3235      0.2059
## Balanced Accuracy         0.9821      1.0000      0.9583      1.0000
```

## 1.7 Resultados y discusión

```
# Modelo de 1 capa oculta con 1 neurona
cmatrix1
```

```
## Confusion Matrix and Statistics
```

```
##
##
## prediction_1 FNT1 FNT2 FNT3 FNT4
##         FNT1    6    0    4    0
##         FNT2    0    8    3    0
##         FNT3    0    0    5    0
##         FNT4    0    1    0    7
##
## Overall Statistics
##
##                Accuracy : 0.7647
##                  95% CI : (0.5883, 0.8925)
##     No Information Rate : 0.3529
##     P-Value [Acc > NIR] : 1.151e-06
##
##                   Kappa : 0.6913
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: FNT1 Class: FNT2 Class: FNT3 Class: FNT4
## Sensitivity               1.0000      0.8889      0.4167      1.0000
## Specificity               0.8571      0.8800      1.0000      0.9630
## Pos Pred Value            0.6000      0.7273      1.0000      0.8750
## Neg Pred Value            1.0000      0.9565      0.7586      1.0000
## Prevalence                0.1765      0.2647      0.3529      0.2059
## Detection Rate            0.1765      0.2353      0.1471      0.2059
## Detection Prevalence      0.2941      0.3235      0.1471      0.2353
## Balanced Accuracy         0.9286      0.8844      0.7083      0.9815
```

```
# Modelo de 1 capa oculta con 3 neuronas
cmatrix3
```

```
## Confusion Matrix and Statistics
##
##
## prediction_3 FNT1 FNT2 FNT3 FNT4
##         FNT1    6    0    1    0
##         FNT2    0    9    0    0
##         FNT3    0    0   11    0
##         FNT4    0    0    0    7
##
## Overall Statistics
##
##                Accuracy : 0.9706
##                  95% CI : (0.8467, 0.9993)
##     No Information Rate : 0.3529
##     P-Value [Acc > NIR] : 2.652e-14
##
##                   Kappa : 0.9601
##
##  Mcnemar's Test P-Value : NA
##
```

```
## Statistics by Class:
##
##                      Class: FNT1 Class: FNT2 Class: FNT3 Class: FNT4
## Sensitivity               1.0000      1.0000      0.9167      1.0000
## Specificity               0.9643      1.0000      1.0000      1.0000
## Pos Pred Value            0.8571      1.0000      1.0000      1.0000
## Neg Pred Value            1.0000      1.0000      0.9565      1.0000
## Prevalence                0.1765      0.2647      0.3529      0.2059
## Detection Rate            0.1765      0.2647      0.3235      0.2059
## Detection Prevalence      0.2059      0.2647      0.3235      0.2059
## Balanced Accuracy         0.9821      1.0000      0.9583      1.0000
```

Al observar los resultados de los modelos realizados mediante la función neuralnet(), parece que el modelo con 3 neuronas en la capa oculta clasifica mejor los 4 fenotipos. En concreto, este modelo tiene una tasa de éxito o *accuracy* del 97%, que difiere del 76.5% del modelo con una sola neurona. Además el modelo con 3 neuronas presenta mejor sensibilidad y especificidad que el modelo de una neurona. Esto tiene sentido, ya que si tenemos más neuronas, se podrán aprender a clasificar mejor los datos y por tanto ajustar mejor las predicciones.

```
# Modelo con nnet del paquete caret
cmatrixnet
```

```
## Confusion Matrix and Statistics
##
##
## prediction_nnet FNT1 FNT2 FNT3 FNT4
##            FNT1    6    0    1    0
##            FNT2    0    9    0    0
##            FNT3    0    0   11    0
##            FNT4    0    0    0    7
##
## Overall Statistics
##
##                Accuracy : 0.9706
##                  95% CI : (0.8467, 0.9993)
##     No Information Rate : 0.3529
##     P-Value [Acc > NIR] : 2.652e-14
##
##                   Kappa : 0.9601
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: FNT1 Class: FNT2 Class: FNT3 Class: FNT4
## Sensitivity               1.0000      1.0000      0.9167      1.0000
## Specificity               0.9643      1.0000      1.0000      1.0000
## Pos Pred Value            0.8571      1.0000      1.0000      1.0000
## Neg Pred Value            1.0000      1.0000      0.9565      1.0000
## Prevalence                0.1765      0.2647      0.3529      0.2059
## Detection Rate            0.1765      0.2647      0.3235      0.2059
## Detection Prevalence      0.2059      0.2647      0.3235      0.2059
## Balanced Accuracy         0.9821      1.0000      0.9583      1.0000
```

En cuanto al modelo con la función nnet() paquete caret, 3-fold cross validation y 3 nodos en la capa oculta, obtenemos prácticamente las mismas estadísticas que en el modelo con la función neuralnet() y 3 neuronas en la capa oculta. En ambos casos se obtienen valores similares de especificidad y sensibilidad en las 4 clases y un buen valor de accuracy (97%).

Para este número de datos no obtenemos diferencias entre el modelo creado con la función neuralnet() y el modelo nnet() del paquete caret con 3-fold cross validation. Es posible que con un mayor número de datos sí que se observaran diferencias.

# 2 Algoritmo Support Vector Machine

## 2.1 Funcionamiento y características

Se trata de un algoritmo de aprendizaje supervisado que se suele utilizar como clasificador discriminatorio, ya que separa los datos creando un hiperplano. Dados los datos de entrenamiento etiquetados, el algoritmo genera un hiperplano óptimo para clasificar los nuevos ejemplos.

Se fundamentan en el *Mazimal Margin Classifier*, que a su vez se basa en el concepto de hiperplano.

Si los datos no pueden separarse de manera lineal se utilizan los *kernels* y se especifica un parámetro C para minimizar la función de coste. Existen varios tipos de kernels:

- Lineal
- Polinomial
- Gaussiano
- Tangente hiperbólica sigmoide

## 2.2 Tabla de fortalezas y debilidades

Table 2: Strengths and Weaknesses of the SVM algorithm

| Fortalezas | Debilidades |
|---|---|
| Puede utilizarse para clasificación o problemas de predicción numérica | Encontrar el mejor modelo requiere probar varias combinaciones de kernels y parámetros del modelo |
| No está excesivamente influenciado por datos de ruido y datos que no son propensos al sobreajuste | El entrenamiento puede ser lento, particularmente si el conjunto de datos de partida tiene un gran número de características |
| Puede ser más fácil de usar que las redes neuronales, paricularmente debido a la existencia de varios algoritmos SVM bien soportados | Es un modelo complejo de caja negra que es difícil, si no imposible, de interpretar |
| Están ganando popularidad debido a su alta precisión y si alto perfil de victorias en competiciones de minería de datos | |

## 2.3 Implementación del algoritmo

```
# Lectura de los ficheros de partida
class <- read.csv(file = file.path(params$folder.data, params$file.class))
data <- read.csv(file = file.path(params$folder.data, params$file.data))
head(data)[,1:6]
```

```
##            V1          V2          V3         V4         V5           V6
## 1  0.10172920  0.59423550 -1.12635568  0.9723793 -0.2171365 -0.370574508
## 2  0.12897473 -0.33259945  0.91374049  0.6013183 -0.2171365 -0.006958959
## 3  0.74419297 -0.33259945  0.17397538 -0.2795754 -0.2171365  0.031994203
## 4  0.48339568  0.68787110 -2.11496708  0.3055419 -0.2171365  0.115864874
```

```
## 5 -0.03216349 -0.07408875 -0.02699948 -0.2501134 -0.2171365  0.405275555
## 6  1.27170501 -0.33259945  1.48865401  0.7502087 -0.2171365 -0.699182804
```

```
# Añadimos la columna class al dataset data
data <- cbind(data,class)
dim(data)
```

```
## [1]  102 5507
```

```
# Pasamos a factor la nueva columna añadida
data$x <- factor(data$x, levels = c(1,2,3,4),
                 labels = c("FNT1","FNT2","FNT3","FNT4"))
data$x
```

```
##   [1] FNT1 FNT1 FNT1 FNT1 FNT1 FNT1 FNT1 FNT1 FNT1 FNT1 FNT1 FNT1 FNT1 FNT1 FNT1 FNT1
##  [16] FNT1 FNT1 FNT1 FNT1 FNT1 FNT1 FNT1 FNT1 FNT1 FNT1 FNT2 FNT2 FNT2 FNT2 FNT2
##  [31] FNT2 FNT2 FNT2 FNT2 FNT2 FNT2 FNT2 FNT2 FNT2 FNT2 FNT2 FNT2 FNT2 FNT2 FNT2
##  [46] FNT2 FNT2 FNT2 FNT2 FNT2 FNT2 FNT3 FNT3 FNT3 FNT3 FNT3 FNT3 FNT3 FNT3 FNT3
##  [61] FNT3 FNT3 FNT3 FNT3 FNT3 FNT3 FNT3 FNT3 FNT3 FNT3 FNT3 FNT3 FNT3 FNT3 FNT3
##  [76] FNT3 FNT3 FNT3 FNT3 FNT4 FNT4 FNT4 FNT4 FNT4 FNT4 FNT4 FNT4 FNT4 FNT4 FNT4
##  [91] FNT4 FNT4 FNT4 FNT4 FNT4 FNT4 FNT4 FNT4 FNT4 FNT4 FNT4 FNT4
## Levels: FNT1 FNT2 FNT3 FNT4
```

```
str(data$x)
```

```
##  Factor w/ 4 levels "FNT1","FNT2",..: 1 1 1 1 1 1 1 1 1 1 ...
```

### 2.3.1 Separación de los datos en train y test

Dado que la semilla aleatoria (12345) ha sido indicada en los parámetros de este archivo, separaremos ahora los datos en dos grupos, el 67% (2/3) de los datos irá para el entrenamiento del modelo (train) y el otro 33% (1/3) para probar el modelo (test).

```
# n será el número de filas del conjunto total de datos
n <- nrow(data)

# Fijamos la semilla de aleatoriedad
set.seed(params$seed.train)

# Partimos los datos en train y test
# n_train = 2/3 = params$p.train
train <- sample(n,floor(n*params$p.train))
data_svm.train <- data[train,]
data_svm.test  <- data[-train,]
```

```
# Comprobamos que hemos partido bien los datos
dim(data_svm.train)
```

```
## [1]   68 5507
```

```r
dim(data_svm.test)
```

```
## [1]   34 5507
```

### 2.3.2 Modelo SVM lineal

Antes de nada, fijamos la nueva semilla de aleatoriedad.

```r
# Fijamos la nueva semilla de aleatoriedad
set.seed(params$seed.clsfier)
```

**ENTRENAMIENTO MODELO**

```r
clasific_lineal <- ksvm(x ~ ., data = data_svm.train,
                        kernel = "vanilladot")
```

```
##  Setting default kernel parameters
```

```r
clasific_lineal
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 58
##
## Objective Function Value : -6e-04 -0.0014 -0.001 -6e-04 -5e-04 -0.001
## Training error : 0
```

**EVALUACIÓN MODELO**

```r
predictions_lineal <- predict(clasific_lineal, data_svm.test)
prop.table(table(predictions_lineal, data_svm.test$x))
```

```
##
## predictions_lineal       FNT1       FNT2       FNT3       FNT4
##               FNT1 0.17647059 0.00000000 0.00000000 0.00000000
##               FNT2 0.00000000 0.26470588 0.00000000 0.00000000
##               FNT3 0.00000000 0.00000000 0.32352941 0.00000000
##               FNT4 0.00000000 0.00000000 0.02941176 0.20588235
```

### 2.3.3 Modelo SVM RBF

**ENTRENAMIENTO MODELO**

```
clasific_rbf <- ksvm(x ~ ., data = data_svm.train,
                         kernel = "rbfdot")
clasific_rbf
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  8.67067839624197e-05
##
## Number of Support Vectors : 62
##
## Objective Function Value : -6.1195 -13.8653 -10.6871 -6.4468 -5.5085 -10.9498
## Training error : 0
```

**EVALUACIÓN MODELO**

```
predictions_rbf <- predict(clasific_rbf, data_svm.test)
prop.table(table(predictions_rbf, data_svm.test$x))
```

```
##
## predictions_rbf       FNT1       FNT2       FNT3       FNT4
##            FNT1 0.17647059 0.00000000 0.02941176 0.00000000
##            FNT2 0.00000000 0.26470588 0.00000000 0.00000000
##            FNT3 0.00000000 0.00000000 0.32352941 0.00000000
##            FNT4 0.00000000 0.00000000 0.00000000 0.20588235
```

### 2.3.4 Modelo svmLinear paquete caret

```
# Configuración 3-fold Cross Validation
train_control <- trainControl(method = "cv", number = 3)

# Ajuste del modelo
model_svmLinear <- train(x ~ ., data = data, method = "svmLinear",
                   trControl = train_control,
                   preProcess = c("center", "scale"))
model_svmLinear
```

```
## Support Vector Machines with Linear Kernel
##
##  102 samples
## 5506 predictors
##    4 classes: 'FNT1', 'FNT2', 'FNT3', 'FNT4'
##
## Pre-processing: centered (5506), scaled (5506)
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 69, 66, 69
## Resampling results:
```

```
##
##   Accuracy  Kappa
##   1         1
##
## Tuning parameter 'C' was held constant at a value of 1
```

Mediante la utilización del parámetro preProcess dentro de la función train() hemos normalizado los datos y los hemos puesto en una escala para que sean comparables entre sí.

El parámetro C, que por defecto tiene valor 1, hace referencia al "coste" y determina los posibles errores de clasificación.

### 2.3.5 Resultados y discusión

En primer lugar, observaremos las matrices de confusión para ver cómo se comportan los diferentes modelos.

```
# Modelo SVM lineal
matrix_lineal <- confusionMatrix(predictions_lineal, data_svm.test$x)
matrix_lineal
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FNT1 FNT2 FNT3 FNT4
##       FNT1    6    0    0    0
##       FNT2    0    9    0    0
##       FNT3    0    0   11    0
##       FNT4    0    0    1    7
##
## Overall Statistics
##
##                Accuracy : 0.9706
##                  95% CI : (0.8467, 0.9993)
##     No Information Rate : 0.3529
##     P-Value [Acc > NIR] : 2.652e-14
##
##                   Kappa : 0.96
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: FNT1 Class: FNT2 Class: FNT3 Class: FNT4
## Sensitivity               1.0000      1.0000      0.9167      1.0000
## Specificity               1.0000      1.0000      1.0000      0.9630
## Pos Pred Value            1.0000      1.0000      1.0000      0.8750
## Neg Pred Value            1.0000      1.0000      0.9565      1.0000
## Prevalence                0.1765      0.2647      0.3529      0.2059
## Detection Rate            0.1765      0.2647      0.3235      0.2059
## Detection Prevalence      0.1765      0.2647      0.3235      0.2353
## Balanced Accuracy         1.0000      1.0000      0.9583      0.9815
```

```r
# Modelo SVM RBF
matrix_rbf <- confusionMatrix(predictions_rbf, data_svm.test$x)
matrix_rbf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FNT1 FNT2 FNT3 FNT4
##       FNT1    6    0    1    0
##       FNT2    0    9    0    0
##       FNT3    0    0   11    0
##       FNT4    0    0    0    7
##
## Overall Statistics
##
##                Accuracy : 0.9706
##                  95% CI : (0.8467, 0.9993)
##     No Information Rate : 0.3529
##     P-Value [Acc > NIR] : 2.652e-14
##
##                   Kappa : 0.9601
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: FNT1 Class: FNT2 Class: FNT3 Class: FNT4
## Sensitivity               1.0000      1.0000      0.9167      1.0000
## Specificity               0.9643      1.0000      1.0000      1.0000
## Pos Pred Value            0.8571      1.0000      1.0000      1.0000
## Neg Pred Value            1.0000      1.0000      0.9565      1.0000
## Prevalence                0.1765      0.2647      0.3529      0.2059
## Detection Rate            0.1765      0.2647      0.3235      0.2059
## Detection Prevalence      0.2059      0.2647      0.3235      0.2059
## Balanced Accuracy         0.9821      1.0000      0.9583      1.0000
```

```r
# Modelo svmLinear
model_svmLinear
```

```
## Support Vector Machines with Linear Kernel
##
##  102 samples
## 5506 predictors
##    4 classes: 'FNT1', 'FNT2', 'FNT3', 'FNT4'
##
## Pre-processing: centered (5506), scaled (5506)
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 69, 66, 69
## Resampling results:
##
##   Accuracy  Kappa
##   1         1
##
```

```
## Tuning parameter 'C' was held constant at a value of 1
```

En los tres modelos, la tasa de éxito es prácticamente idéntica, de en torno al 97%. El valor kappa también es muy bueno, del 96% en los tres. En los modelos SVM de los que obtenemos más características para ayudarnos a analizar su comportamiento observamos que se comportan prácticamente igual, diferenciándose en que los dos se equivocan en una predicción de la clase 3. Por tanto, parece que los tres modelos clasifican de igual manera los datos de los que disponemos, siendo esta una clasificación bastante buena.

# Referencias


Hassoun, Mohamad H, and others. 1995. *Fundamentals of Artificial Neural Networks*. MIT press.

Lantz, Brett. 2019. *Machine Learning with R: Expert Techniques for Predictive Modeling*. Packt Publishing Ltd.

Noble, William S. 2006. "What Is a Support Vector Machine?" *Nature Biotechnology* 24 (12): 1565–7.

Xie, Yihui, Joseph J Allaire, and Garrett Grolemund. 2018. *R Markdown: The Definitive Guide*. CRC Press.