Guia 2

Grupo 3

10/19/2019

Ejercicio 1

Resolución del problema XOR con una red neuronal RBF

- Lectura de los patrones de entrenamiento

```
XOR_trn <- read_csv("../../PUBLICO/Encuentro 1/Práctica/data/XOR_trn.csv", col_names = FALSE)
XOR_tst <- read_csv("../../PUBLICO/Encuentro 1/Práctica/data/XOR_tst.csv", col_names = FALSE)
```

- Selección de parámetros y entrenamiento de perceptrón

En este caso se utilizan 4 gausianas por la distribución de los datos.

```
datos_x <- XOR_trn[,c(1,2)]
datos_y <- XOR_trn[,3]
modeloRBF <- redRBF(datos_x, datos_y, nroGausianas = 4, funcion = "sigmo")
```

```
## Epoca: 1 - Tasa: 0.7545 - Error: 0.897460645104851
## Epoca: 2 - Tasa: 1 - Error: 0.754493866515958
```

- Prueba con datos de test

```
datos_x <- XOR_tst[,c(1,2)]
datos_y <- XOR_tst[,3]
salida <- aplicarRedRBF(modeloRBF, datos_x, datos_y)
salida$tasa
```

```
## [1] 1
```

Resolución del problema Iris con una red neuronal RBF

- Lectura de los patrones de entrenamiento

```
irisbin <- read_csv("../../PUBLICO/Encuentro 1/Práctica/data/irisbin.csv", col_names = FALSE)
```

- Selección de parámetros y entrenamiento de perceptrón
- Prueba con datos

```
salida <- aplicarRedRBF(modeloRBF, datos_x, datos_y)
salida$tasa
```

```
## [1] 0.94
```

```
head(salida$salida)
```

```
##    1  1  1
## 1 -1 -1  1
## 2 -1 -1  1
## 3  1 -1 -1
## 4 -1 -1  1
## 5  1 -1 -1
## 6  1 -1 -1
```

Cantidad de parámetros:

En una red MLP con una estructura (3,1), tenemos los siguientes parámetros:

$numParamMLP = ParámetrosdeCapa1 + ParámetrosdeCapa2$

$numParamMLP = [(4entradas + 1) * 3neuronas] + [(3entradas + 1) * 1neurona]$

$numParamMLP = 5 * 3 + 4 * 1 = 19parámetros$

Una red RBF con 19 parámetros podría tener la siguiente distribución:

$numParamRBF = ParámetrosdeGausianas + ParámetrosdePerceptrones$

$numParamRBF = [3centros] + [(3entradas + 1) * 3neurona]$

$numParamRBF = 3 + 4 * 3 = 15parámetros$

- Prueba con datos

```
salida_2 <- aplicarRedRBF(modeloRBF_2, datos_x, datos_y)
salida_2$tasa
```

```
## [1] 0.7133333
```

```
head(salida_2$salida)
```

```
##    1  1  1
## 1 -1 -1  1
## 2 -1 -1  1
## 3  1 -1 -1
## 4 -1 -1  1
## 5  1 -1 -1
## 6  1 -1 -1
```

Ejercicio 2

- Lectura de datos

```
merval <- read_csv("../../PUBLICO/Encuentro 3/Práctica/data/merval.csv", col_names = FALSE)
```

- Preprocesamiento de los datos

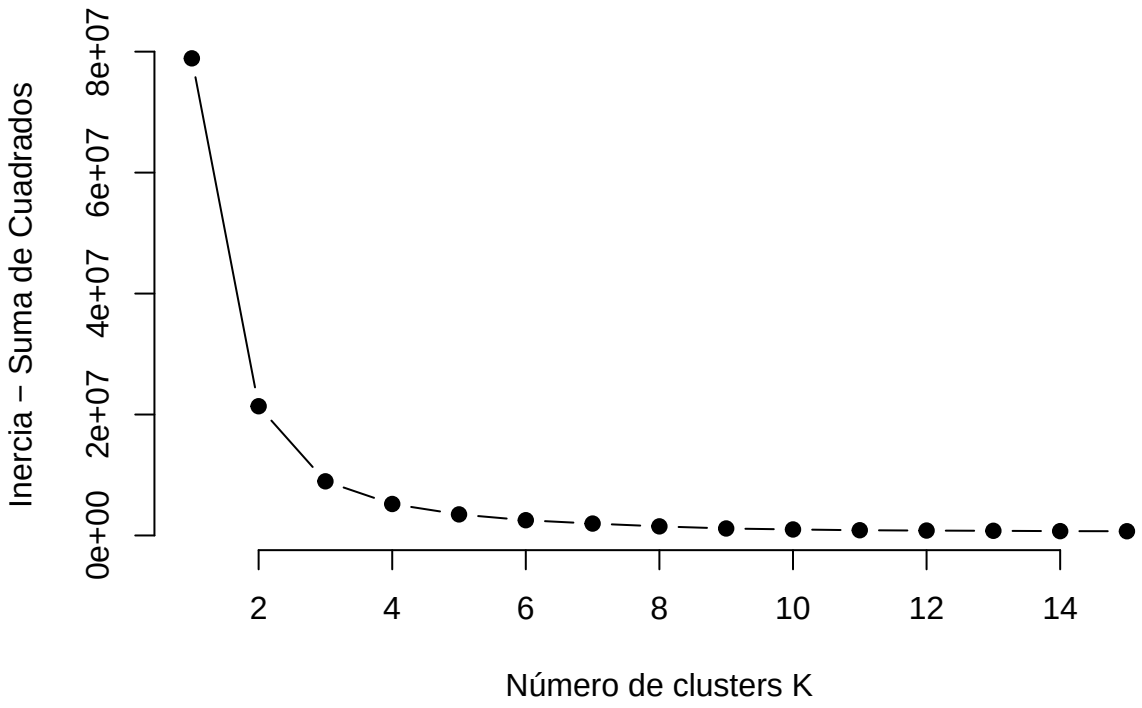
Generamos un dataset que contenga seis valores consecutivos en cada registro, cinco tomados como datos de entrada y un sexto valor tomado como clase.

```r
cantidadDatos <- nrow(merval)
datos_merval <- matrix(0,nrow = cantidadDatos-5, ncol = 6)

for (i in seq(1,cantidadDatos-5)) {
  datos_merval[i,] <- merval$X1[seq(i,i+5)]
}

datos_x <- datos_merval[,c(1,2,3,4,5)] %>% as.matrix()
datos_y <- datos_merval[,6] %>% as.matrix()

# Primeros seis registros del dataset
head(datos_merval)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]  215  212  229  253  254  235
## [2,]  212  229  253  254  235  239
## [3,]  229  253  254  235  239  241
## [4,]  253  254  235  239  241  252
## [5,]  254  235  239  241  252  253
## [6,]  235  239  241  252  253  257
```

Antes de generar el modelo, tenemos que definir el número de gausianas. Utilizamos la gráfica de Elbow para definir el k a utilizar en el modelo.

```r
set.seed(123)
wss <- function(k) {
  kmeans(datos_x, k, nstart = 10 )$tot.withinss
}
# Valores de k = 1 a k = 15
k.values <- 1:15
wss_values <- map_dbl(k.values, wss)
plot(k.values, wss_values,
       type="b", pch = 19, frame = FALSE,
       xlab="Número de clusters K",
       ylab="Inercia - Suma de Cuadrados")
```



Mirando la gráfica anterior tomamos un valor de k = 4, es donde la gráfica hace el codo y queda aproximadamente constante.

- Normalizamos los datos.

```
maximo <- 0
for (i in seq(1,ncol(datos_x))) {
  if (max(datos_x[,i]) > maximo) {maximo <- max(datos_x[,i])}
}
if (max(datos_y) > maximo) {maximo <- max(datos_y)}

datos_x <- datos_x / maximo
datos_y <- datos_y / maximo
```

- Dividimos los datos en Train y Test, utilizando un 70% para entrenamiento.

```
merval7030 <- generarParticionPorID(as.data.frame(datos_merval), porcEntrenamiento = 0.7,
                                    semilla = 1, clase = "V6")
```

- Generamos el modelo con los datos de entrenamiento.

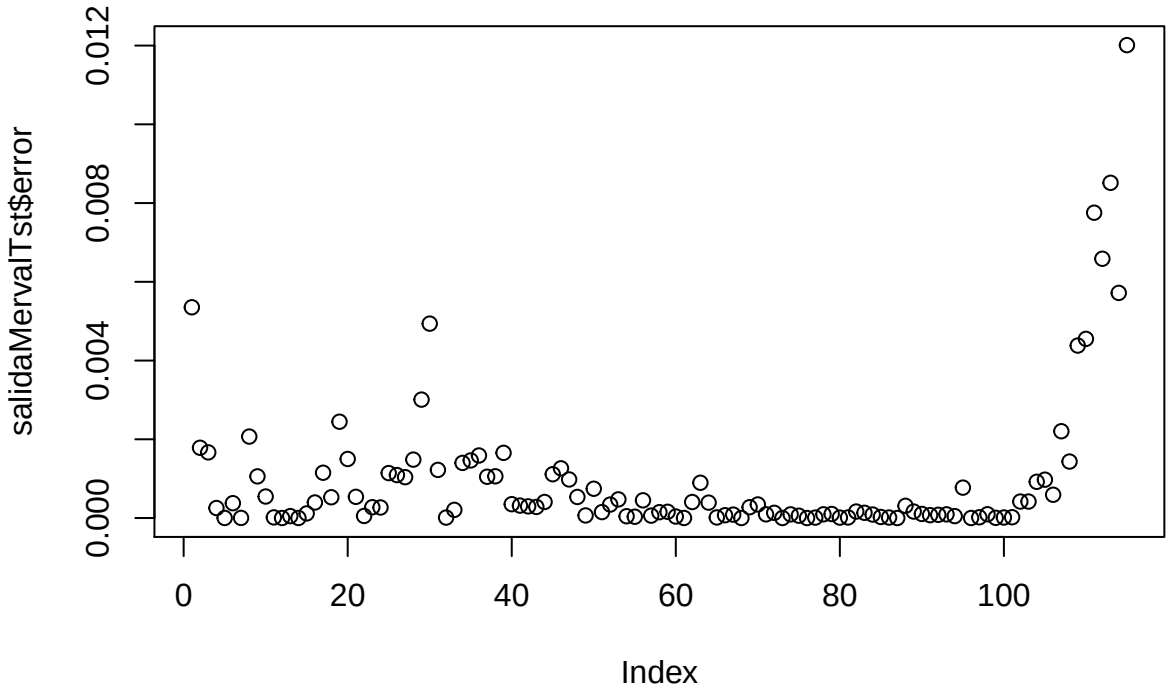
```
if (calcular) {
modeloMerval70 <- redRBF(datos_x[merval7030$trn,], (datos_y[merval7030$trn,] %>% as.matrix()),
                         nroGausianas = 4, funcion = "lineal", pnu = 0.01, pepoca = 200000,
                         pcritFinalizacion = 0.95, ptolerencia = 0.1)
}
```

- Aplicamos el modelo a los datos de Train y Test

```
salidaMervalTrn <- aplicarRedRBF(modeloMerval70, datos_x[merval7030$trn,],
                                 (datos_y[merval7030$trn,] %>% as.matrix()))
salidaMervalTst <- aplicarRedRBF(modeloMerval70, datos_x[merval7030$tst,],
                                 (datos_y[merval7030$tst,] %>% as.matrix()))
```

- Grafica de error en Test

```
#Grafica de error en cada registro
plot(salidaMervalTst$error)
```




```
#Error cuadrático medio
errorMedio <- sum(salidaMervalTst$error) / length(salidaMervalTst$error)
errorMedio
```

```
## [1] 0.0009814814
```

- Generamos el modelo a aplicar para realizar las predicciones con todos los datos.

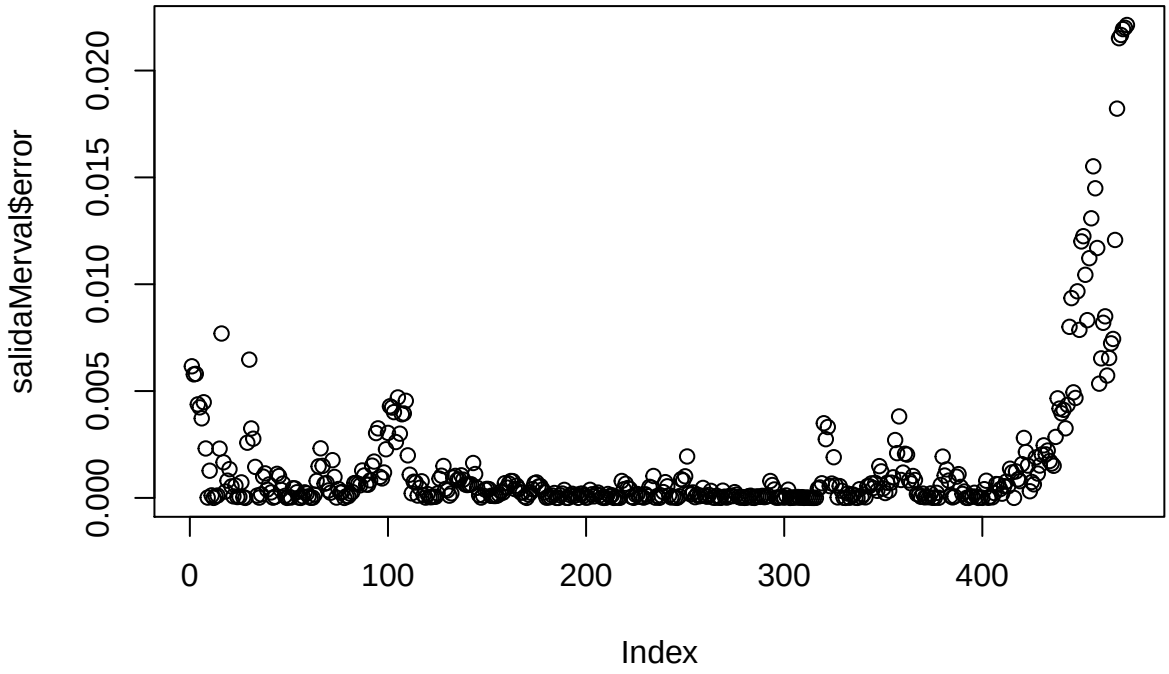
```
# Generamos el modelo
if (calcular) {
modeloMerval <- redRBF(datos_x, datos_y, nroGausianas = 4, funcion = "lineal", pnu = 0.01,
                    pepoca = 200000, pcritFinalizacion = 0.95, ptolerencia = 0.1)
}
```

Aplicamos el modelo a los mismos datos de entrenamiento para graficar error en train.

```
salidaMerval <- aplicarRedRBF(modeloMerval, datos_x, datos_y)
```

Grafica de error

```
#Grafica de error en cada registro
plot(salidaMerval$error)
```

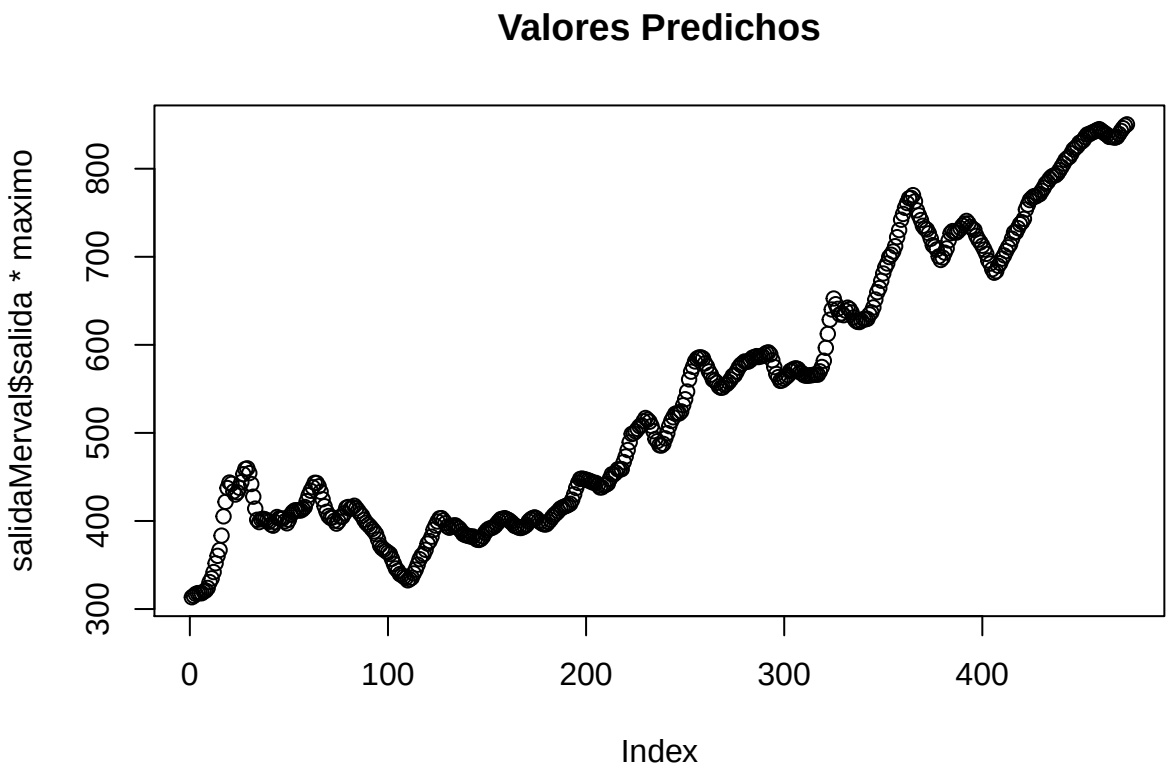



```
#Error cuadrático medio
errorMedio <- sum(salidaMerval$error) / length(salidaMerval$error)
errorMedio
```

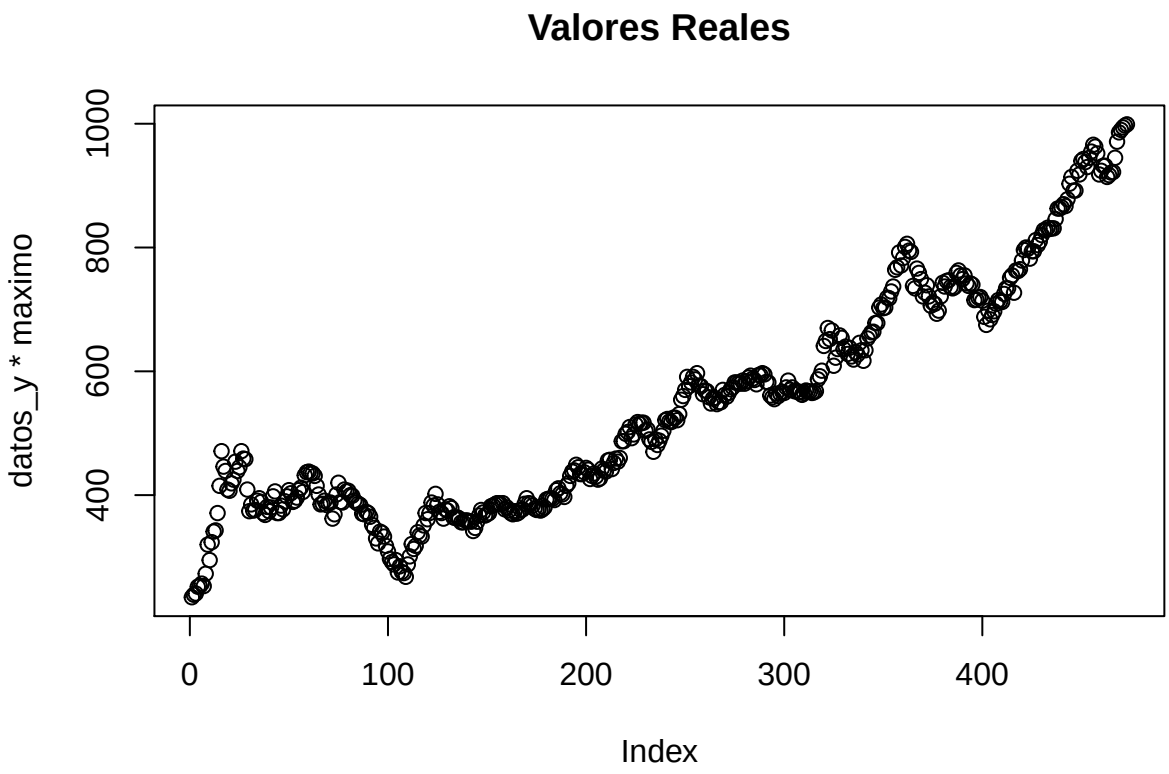
```
## [1] 0.001437009
```

- Gráfica del valor predicho y el valor real

```
plot(salidaMerval$salida * maximo, main = "Valores Predichos")
```




```
plot(datos_y * maximo, main = "Valores Reales")
```



- Predecimos un nuevo valor

Tomamos los últimos 5 valores del dataset y predecimos cual será el próximo valor.

```
ultimosDatos <- (merval[seq(nrow(merval)-4,nrow(merval)),] / maximo) %>% as.matrix()
ultimosDatos <- t(ultimosDatos)
ultimosDatos <- rbind(ultimosDatos,ultimosDatos) %>% as.matrix()
#usamos dos registros por el tipo de datos.
salidaUno <- as.matrix(c(1,1))
salidaMervalNew <- aplicarRedRBF(modeloMerval, ultimosDatos, salidaUno)
#Nuevo valor predicho
salidaMervalNew$salida[1] * maximo
```

```
##       X1
## 851.0265
```

```
# Guardamos los modelos generados
if (calcular) {
save(modeloMerval, modeloMerval70,file = "resultadosG2.RData")
}
```