Práctica 3

Carbonera Francesco Giada Cesaro

3/12/2019

Para esta tarea se utilizaran redes de neuronas. Para las redes de neuronas es conveniente que las entradas (y la salida, si es un problema de regresión) estén normalizadas. En este caso la normalización se hace aquì directamente usando `mlr`. También, se establece una conexión *Spark*, que llamamos `sc`, para poder trabajar con ella luego.

```
sc <- spark_connect(master = "local")
datos_train <- "trainst3.rds"
datos_test <- "testst3.rds"
train <- readRDS(datos_train)
test <- readRDS(datos_test)
train <- mlr::normalizeFeatures(train,method="range")
test <- mlr::normalizeFeatures(test,method="range")
saveRDS(train, datos_train)
saveRDS(test, datos_test)
train <- readRDS(datos_train)
test <- readRDS(datos_test)
```

Los datos de train se copian del driver al *cluster*.

El problema de predicción de radiación solar, con el que estamos trabajando, es de regresión. Lo vamos a transformar en uno de clasificación. Para ello vamos a transformar la columna de salida (que de momento es numérica) en dos clases: radiación normal y radiación alta. Consideraremos que la radiación es alta cuando supera el tercer cuantil.
Para calcular el tercer cuantil primero se debe usar el comando `collect()` para trae al ordenador local la columna `salida` que hemos seleccionado usando el comando `select` de `dplyr`, y solo una vez que los datos estan en local se pueden calcular el quantil (lo guardamos en la variable `c75`). Es necesario hacer todo esto porque *Spark* no tiene una función *quantile*, así como no tiene una función *median* implementada. Probablemente esto pasa porque los cuantiles (la mediana es un cuantil particular) son estadísticas que no pueden calcularse en paralelo, y aquì estamos trabajando con datos que en *Spark* estan almacenados (virtualmente) en dos partes. Por el contrario, la media podría calcularse directamente con *Spark* (o sea `d<- datos_train_tbl%>% select(salida) %>% summarize(media= mean())` no daría error).
Una vez que se ha obtenido el valor `c75`, la función `ft_binarizer()` de *Spark* se usa para convertir la respuesta en una variable binaria.

```
datos_train_tbl <- sdf_copy_to(sc, train, repartition = 2, overwrite = TRUE, name="TrainSpark")

d<- datos_train_tbl%>% select(salida) %>% collect()
d<- as.vector(d$salida)
c75<- quantile(d, probs=c(0.75))

datos_train_tbl<- datos_train_tbl %>% ft_binarizer(input_col = "salida",
                                        output_col = "salida_bin", threshold = c75)
```

Comprobamos que de hecho la variable `salida_bin` tenga el 25% de unos, calculando su media (vemos aquí,

como acabamos de decir, que la función `mean()` funciona, porque es paralelizable).

```
datos_train_tbl%>%  summarize(mean=mean(salida_bin))
```

```
# Source: spark<?> [?? x 1]
   mean
  <dbl>
1  0.25
```

Preparamos los datos en el formato que requiere *Spark* para aplicar sus modelos de aprendizaje automático, y subdividimos los datos de entrenamiento para tener un conjunto de validación que podamos usar para hacer un ajuste (manual) de los hiperparámetros. En este caso solo se probarán 3 valores para el número de capas intermedias de la red, y nos quedaremos con el mejor (error calculado con el conjunto de validación).

```
datos_train_tbl<- datos_train_tbl%>% select(-salida)
partitions <- datos_train_tbl %>%
  sdf_random_split(training = 2/3, validacion = 1/3, seed = 0)

nombres <- tbl_vars(partitions$training)

features <- nombres[-76]
response <- nombres[76]
```

```
tasa_error=rep(NA,3)
for (i in 3:5){

set.seed(0)
model_neuralnetwork <- ml_multilayer_perceptron_classifier(partitions$training,
                features = features, response = response, layers = c(75, i, 2))

predictions_NN_tbl <- ml_predict(model_neuralnetwork, partitions$validacion)
tasa_aciertos <- predictions_NN_tbl %>%
  mutate(salida_bin= as.double(salida_bin)) %>%
  ml_multiclass_classification_evaluator(prediction_col="prediction",
                            label_col="salida_bin", metric_name="accuracy")

tasa_error[i-2] <- 1 - tasa_aciertos
}

tasa_error
```

```
[1] 0.09215956 0.08459422 0.09147180
```

Una vez que hayamos encontramos el mejor número de capas intermedias (4) lo mantenemos fijo y intentamos ajustar el hiperparámetro `max_iter`. También en este caso solo se probarán solamente algunos valores.

```
maxiters<- c(105,110,115, 120)

for (i in 1:4) {
set.seed(0)
model_neuralnetwork <- ml_multilayer_perceptron_classifier(partitions$training,
                              features = features, response = response,
```

```
                              layers = c(75, 4, 2), max_iter = maxiters[i])

predictions_NN_tbl <- ml_predict(model_neuralnetwork, partitions$validacion)
tasa_aciertos <- predictions_NN_tbl %>%
  mutate(salida_bin= as.double(salida_bin)) %>%
  ml_multiclass_classification_evaluator(prediction_col="prediction",
                          label_col="salida_bin", metric_name="accuracy")
tasa_error[i] <- 1 - tasa_aciertos
}
tasa_error
```

```
[1] 0.08734525 0.08390646 0.08390646 0.08528198
```

Se ve que los mejores resultados se obtienen con valores de `max_iter` igual a 110 o 115 (se obtiene el mismo error).

Se construye un modelo final con 4 *layers* ocultos y con `max_iter` igual a 110, usandos todos los datos de *train*.

```
model_neuralnetwork <- ml_multilayer_perceptron_classifier(datos_train_tbl,
                                    features = features, response = response,
                                    layers = c(75, 4, 2), max_iter=110)
```

Finalmente se hace una predicción usando el conjunto de *test* (que está guardado en un fichero distinto, con lo cual inicialmente no tuvimos que dividir los datos en un subconjunto de *training* y en uno de *test*).

```
datos_test_tbl <- sdf_copy_to(sc, test, repartition = 2, overwrite = TRUE, name="TestSpark")
predictions_final<- ml_predict(model_neuralnetwork, datos_test_tbl)
ValPredichos<- predictions_final %>% select(prediction) %>% collect()
write.table(ValPredichos, file = "predicciones.txt",
            row.names = FALSE)
spark_disconnect(sc)
```