Practica1LBP

Francisco Pérez Hernández

29/3/2017

1 Trabajo en Matlab

Fichero "Practica1LBP.m"

1.1 Implementación de un descriptor

En esta primera función leeremos una imagen y aplicaremos la función LBPu con la que se transforma la imagen recibida, añadiéndole un marco de 0 para que mi implementación sea más sencilla. De esta forma recorremos cada pixel de la imagen y calculo el código LBP uniforme. Este código lo calculo sacando si los elementos al rededor son menores ó mayor-igual que el elemento examinado sacando el número en binario con el que calculamos cuantos cambios, o transacciones, hay en ese número y será cuando le demos valor a ese pixel. Una vez realizado esto, muestro la imagen LBPu y el histograma que se proporciona.

1.2 Cálculo de características

En esta función recorremos los 105 bloques de la imagen. En cada bloque calcularemos el histograma del bloque, lo normalizaremos con la raíz cuadrada de la sumatoria de los elementos al cuadrado. Por lo tanto, con la concatenación de los 105 histogramas, como cada uno tiene 59 elementos, tendremos un vector con 6195 características por cada imagen.

1.3 Cálculo para todas las imágenes

Lo que realizaremos en esta parte, será leer todas las imágenes proporcionadas y crearemos un csv para cada conjunto, en el que cada imagen tendrá un vector de 6195 características. Este csv, lo pasaremos a continuación a R para realizar un clasificador.

2 Trabajo en RStudio

Por motivos de facilidad y por acelerar el trabajo, ya que era mi primer contacto con Matlab, he decidido realizar el clasificador en R. Fichero "Practica1LBP.Rmd"

2.1 Lectura de las características

Primero leo las características de los csv creados con Matlab

```
train.pedestrian <- read.csv("matriz_pedestrians_train.csv", header = F)
train.background <- read.csv("matriz_background_train.csv", header = F)
test.pedestrian <- read.csv("matriz_pedestrians_test.csv", header = F)
test.background <- read.csv("matriz_background_test.csv", header = F)
```

2.2 Añadido de etiquetas

Creamos los vectores de etiquetas para cada dataset

```
pedestrians_train_eti <- as.data.frame(rep(1,1916))
colnames(pedestrians_train_eti) <- c("Y")
backgorund_train_eti <- as.data.frame(rep(0,2390))
colnames(backgorund_train_eti) <- c("Y")
```

Añadimos las etiquetas a los datasets de train

```
train.pedestrian.eti <- cbind(train.pedestrian,pedestrians_train_eti)
train.background.eti <- cbind(train.background,backgorund_train_eti)
train <- rbind(train.pedestrian.eti,train.background.eti)
```

Hacemos lo mismo para el test

```
test <- rbind(test.pedestrian,test.background)
aux <- as.data.frame(rep(1,500))
colnames(aux) <- c("Y")
aux2 <- (as.data.frame(rep(0,600)))
colnames(aux2) <- c("Y")
test.aux2 <- rbind(aux, aux2)
test <- rbind(test.pedestrian,test.background)
test <- cbind(test,test.aux2)
```

Pasamos la clase a factor para el correcto funcionamiento del clasificador

```
train$Y <- as.factor(train$Y)
test$Y <- as.factor(test$Y)
```

2.3 Elección del clasificador

Como clasificador para esta práctica, se ha decidido usar el clasificador Random Forest, ya que suele tener buenos resultados para conjuntos de datos similares. Por lo tanto voy a realizar varias pruebas con distintos valores de número de árboles para ver que tal funciona nuestro clasificador.

2.3.1 10 árboles

Vamos a probar el clasificador random forest primero con 10 árboles

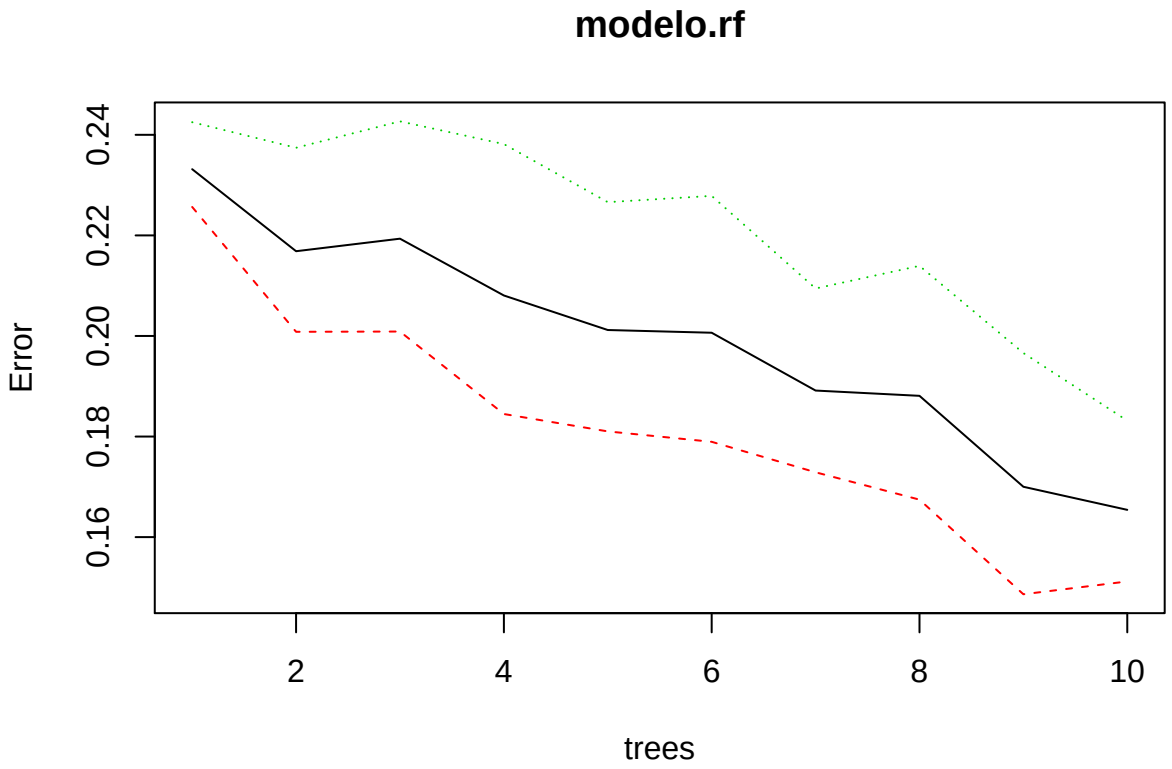
```
rm(list=setdiff(ls(), c("train","test")))
set.seed(1234)
modelo.rf <- randomForest::randomForest(Y~.,data = train, ntree=10)
```

```
print(modelo.rf)
```

```
##
## Call:
##  randomForest(formula = Y ~ ., data = train, ntree = 10)
##                Type of random forest: classification
##                      Number of trees: 10
## No. of variables tried at each split: 78
##
##         OOB estimate of  error rate: 16.54%
## Confusion matrix:
```

```
##      0    1 class.error
## 0 2010  358   0.1511824
## 1  348 1552   0.1831579
```

```
plot(modelo.rf)
```




```
prediccion <- predict(modelo.rf, newdata = test)
tabla <- table(prediccion, test$Y)
tabla
```

```
##
## prediccion   0   1
##          0 538  74
##          1  62 426
```

```
accuracy <- sum(prediccion == test$Y)/nrow(test)
accuracy
```

```
## [1] 0.8763636
```

2.3.2 50 árboles

Vamos a probar el clasificador random forest con 50 árboles

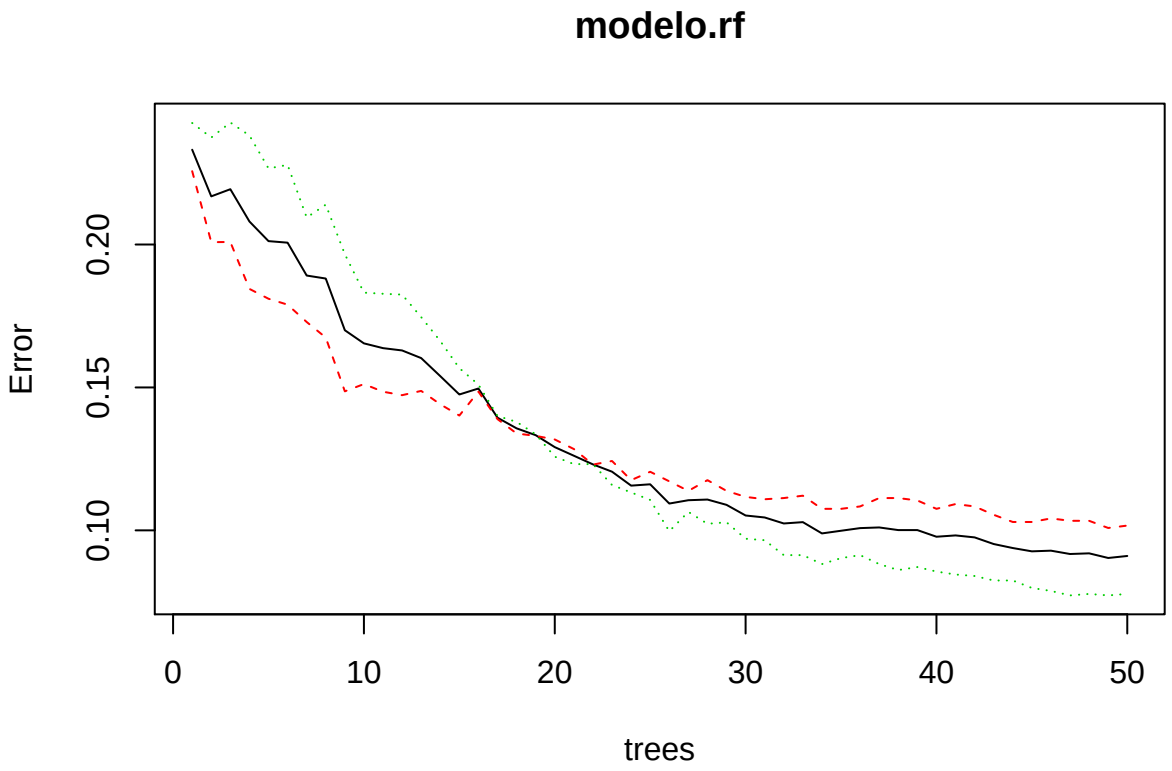
```
rm(list=setdiff(ls(), c("train","test","accuracy")))
set.seed(1234)
modelo.rf <- randomForest::randomForest(Y~.,data = train, ntree=50)
```

```
print(modelo.rf)
```

```
##
## Call:
```

```
##  randomForest(formula = Y ~ ., data = train, ntree = 50)
##                Type of random forest: classification
##                      Number of trees: 50
## No. of variables tried at each split: 78
##
##         OOB estimate of  error rate: 9.1%
## Confusion matrix:
##      0    1 class.error
## 0 2147  243  0.10167364
## 1  149 1767  0.07776618
```

```
plot(modelo.rf)
```




```
prediccion <- predict(modelo.rf, newdata = test)
tabla <- table(prediccion, test$Y)
tabla
```

```
##
## prediccion   0   1
##          0 552  47
##          1  48 453
```

```
accuracy_act <- sum(prediccion == test$Y)/nrow(test)
accuracy_act
```

```
## [1] 0.9136364
```

```
accuracy <- c(accuracy,accuracy_act)
```

2.3.3 100 árboles

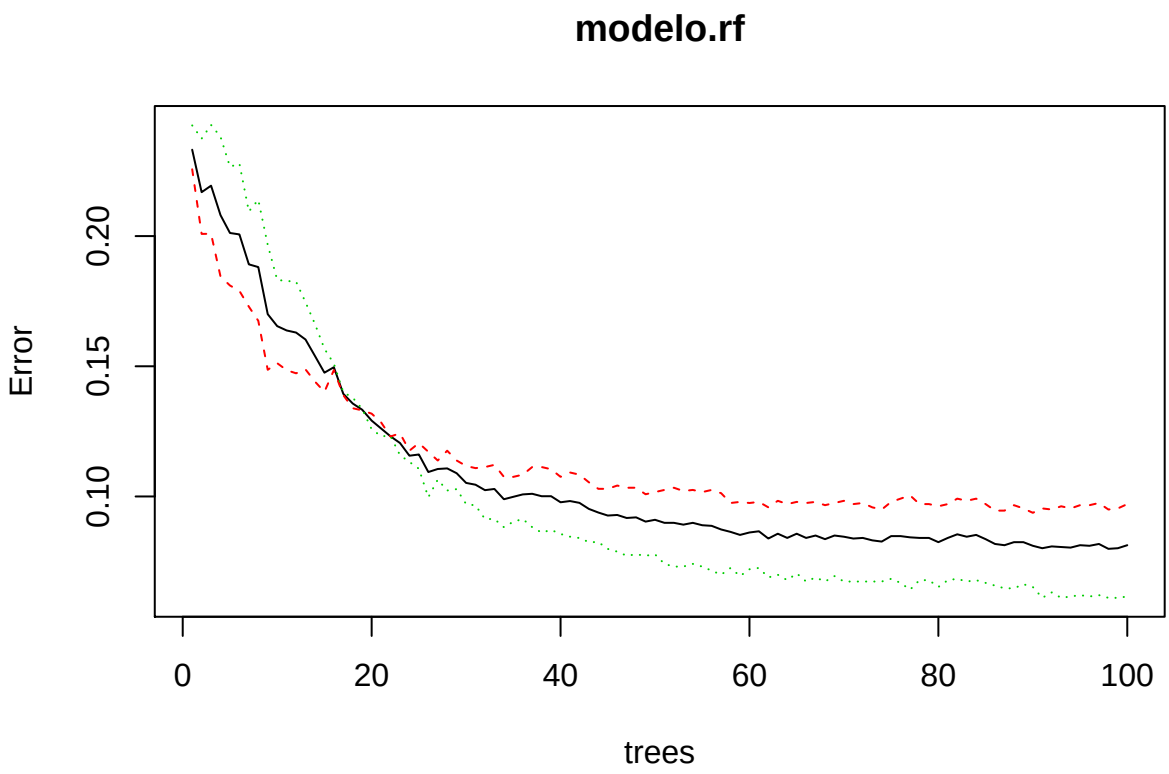
Vamos a probar el clasificador random forest con 100 árboles

```
rm(list=setdiff(ls(), c("train","test","accuracy")))
set.seed(1234)
modelo.rf <- randomForest::randomForest(Y~.,data = train, ntree=100)
```

```
print(modelo.rf)
```

```
##
## Call:
##  randomForest(formula = Y ~ ., data = train, ntree = 100)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 78
##
##         OOB estimate of  error rate: 8.13%
## Confusion matrix:
##      0    1 class.error
## 0 2158  232  0.09707113
## 1  118 1798  0.06158664
```

```
plot(modelo.rf)
```




```
prediccion <- predict(modelo.rf, newdata = test)
tabla <- table(prediccion, test$Y)
tabla
```

```
##
## prediccion   0   1
##          0 553  45
```

```
##          1  47 455
```

```
accuracy_act <- sum(prediccion == test$Y)/nrow(test)
accuracy_act
```

```
## [1] 0.9163636
```

```
accuracy <- c(accuracy,accuracy_act)
```

2.3.4 200 árboles

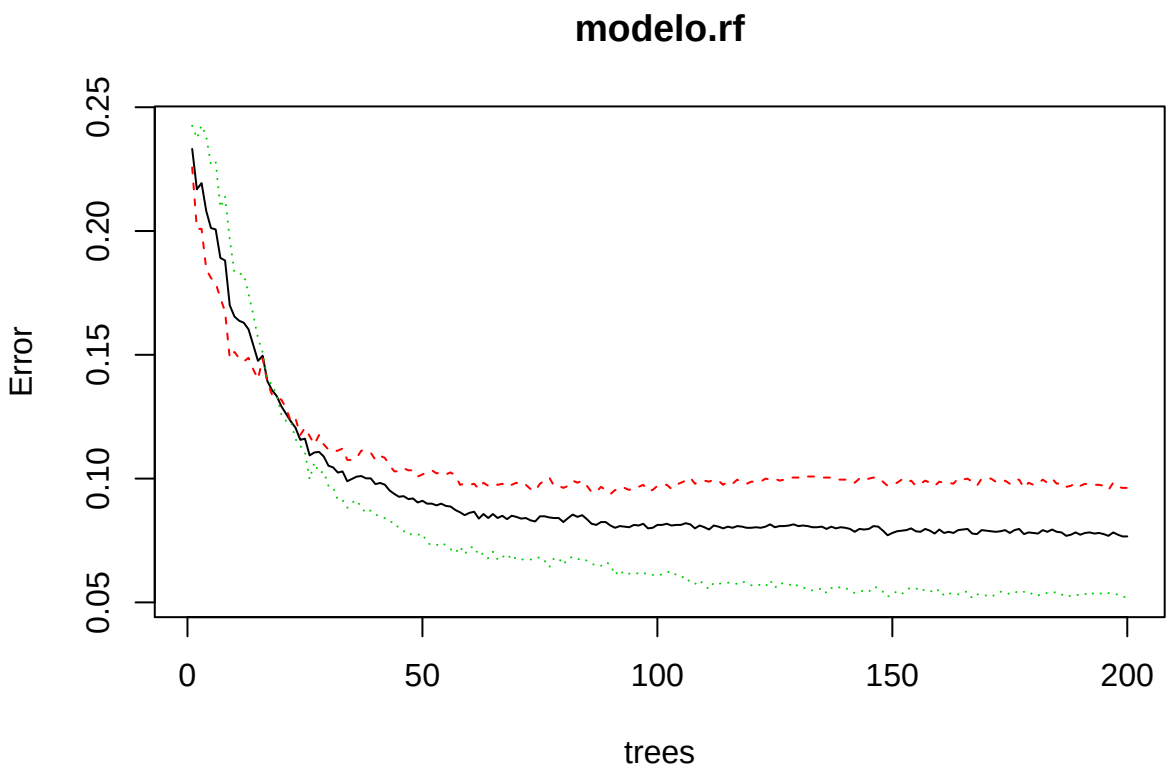
Vamos a probar el clasificador random forest con 200 árboles

```
rm(list=setdiff(ls(), c("train","test","accuracy")))
set.seed(1234)
modelo.rf <- randomForest::randomForest(Y~.,data = train, ntree=200)
```

```
print(modelo.rf)
```

```
##
## Call:
##  randomForest(formula = Y ~ ., data = train, ntree = 200)
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 78
##
##         OOB estimate of  error rate: 7.66%
## Confusion matrix:
##      0    1 class.error
## 0 2160  230  0.09623431
## 1  100 1816  0.05219207
```

```
plot(modelo.rf)
```




```
prediccion <- predict(modelo.rf, newdata = test)
tabla <- table(prediccion, test$Y)
tabla
```

```
##
## prediccion   0   1
##          0 557  46
##          1  43 454
```

```
accuracy_act <- sum(prediccion == test$Y)/nrow(test)
accuracy_act
```

```
## [1] 0.9190909
```

```
accuracy <- c(accuracy,accuracy_act)
```

2.3.5 500 árboles

Vamos a probar el clasificador random forest con 500 árboles

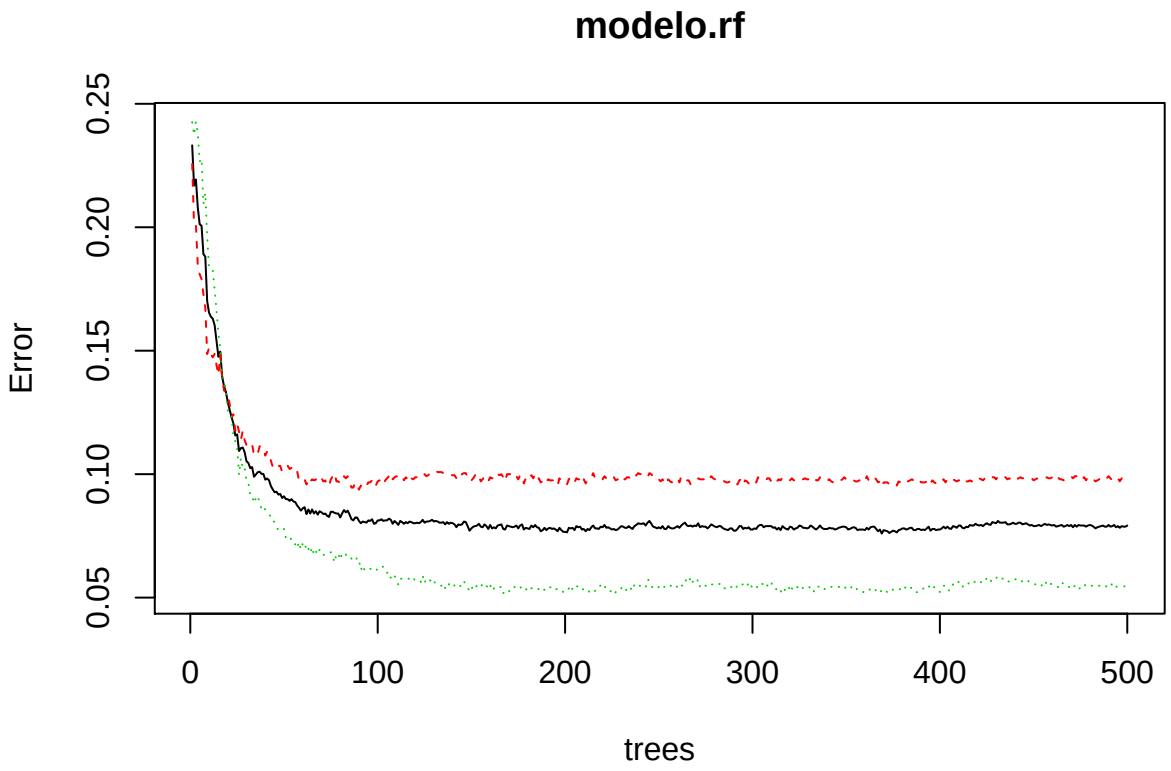
```
rm(list=setdiff(ls(), c("train","test","accuracy")))
set.seed(1234)
modelo.rf <- randomForest::randomForest(Y~.,data = train, ntree=500)
```

```
print(modelo.rf)
```

```
##
## Call:
##  randomForest(formula = Y ~ ., data = train, ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
```

```
## No. of variables tried at each split: 78
##
##         OOB estimate of  error rate: 7.92%
## Confusion matrix:
##      0    1 class.error
## 0 2154  236  0.09874477
## 1  105 1811  0.05480167
```

```
plot(modelo.rf)
```




```
prediccion <- predict(modelo.rf, newdata = test)
tabla <- table(prediccion, test$Y)
tabla
```

```
##
## prediccion   0   1
##          0 555  42
##          1  45 458
```

```
accuracy_act <- sum(prediccion == test$Y)/nrow(test)
accuracy_act
```

```
## [1] 0.9209091
```

```
accuracy <- c(accuracy,accuracy_act)
```

3 Conclusiones

Veamos los accuracy de cada clasificador

```r
names(accuracy) = c("10","50","100","200","500")
accuracy
```

```
##        10        50       100       200       500
## 0.8763636 0.9136364 0.9163636 0.9190909 0.9209091
```

Vemos el número de árboles y su accuracy proporcionado. Como hemos podido ver en las gráficas obtenidas y en este vector, el número de árboles que mejor nos ha funcionado ha sido el de 500 árboles. Pero viendo el comportamiento convergente se ha decidido no hacer más pruebas. Por lo tanto, aceptamos este resultado como el de un muy buen clasificador ya que tiene una tasa de error muy baja. Por lo tanto, hemos montado, a partir de un conjunto de train de background y pedestrians, un clasificador con el que podemos predecir si una nueva imagen es de una de las dos clases.