Clasificación monotónica

Juan Ignacio Isern Ghosn
Universidad de Granada
Minería de datos: Aspectos avanzados

07/02/2019

Contents

Clasificación monotónica

En este documento se presenta la primera parte de la práctica del tema *Non-standard problems* de la asignatura de Minería de datos: Aspectos avanzados. Concretamente, en este documento se trata la clasificación monotónica.

Carga de librerías necesarias

Se cargan aquellas librerías necesarias para nuestro análisis:

```
library(RWeka)
library(plyr)
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library(xgboost)
```

Carga de los conjuntos de datos a utilizar

Se cargan el dataset para el cual se realiza el análisis en este documento:

```
dataset_esl <- RWeka::read.arff("../Material/esl.arff")
```

Etiquetas de las clases

Se guardan las etiquetas de cada una de las clases:

```
labels_esl <- range(dataset_esl[,ncol(dataset_esl)])[1]:range(dataset_esl[,ncol(dataset_esl)])[2]
labels_esl
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

Se establecen como factor la variable correspondiente a las etiquetas

```
dataset_esl[,ncol(dataset_esl)] <- factor(dataset_esl[,ncol(dataset_esl)])
```

Creación de conjuntos de train y test

Para evaluar nuestros modelos, se divide nuestro dataset en train y test

```
# Para conjunto esl
index_esl <- createDataPartition(dataset_esl[,ncol(dataset_esl)], p = 0.7, list = FALSE)
train_esl <- dataset_esl[index_esl,]
test_esl <- dataset_esl[-index_esl,]
```

Etiquetas binarias para clasificación monotónica

Se deben generar nuevas etiquetas binarias para entrenar cada uno de los modelos, pues actualmente tenemos una única variable que guarda todas las etiquetas, así, generamos la siguiente función:

```r
ordinal_labels <- function(data, class_indx){
    data[,class_indx] <- as.integer(revalue(as.factor(data[,class_indx])))
    classes <- sort(unique(data[,class_indx]))
    data_aux <- data[,-class_indx]
    for(class in classes[-length(classes)]){
        data_aux <- cbind(data_aux, "Target" = ifelse(data[, class_indx] <= class,0,1))
        names(data_aux)[length(names(data_aux))] <- paste0("Target", class)
    }
    return(data_aux)
}
```

El resultado obtenido es el siguiente:

```r
data_lab_esl <- ordinal_labels(train_esl, ncol(train_esl))
head(data_lab_esl)
```

```
##   in1 in2 in3 in4 Target1 Target2 Target3 Target4 Target5 Target6 Target7
## 1   6   5   6   6       1       1       1       1       1       0       0
## 2   5   4   5   5       1       1       1       1       0       0       0
## 3   5   3   4   5       1       1       1       0       0       0       0
## 4   6   5   6   7       1       1       1       1       1       0       0
## 5   4   3   3   5       1       1       0       0       0       0       0
## 6   9   6   6   6       1       1       1       1       1       0       0
##   Target8
## 1       0
## 2       0
## 3       0
## 4       0
## 5       0
## 6       0
```

Modelos de clasificación binarios

Del mismo modo, es necesario generar modelos para clasificar cada una de las nuevas etiquetas binarias generadas. En nuestro caso, hacemos uso del xgboost del paquete xgboost, al cual le podemos hacer cumplir las restricciones de monotonía, teniendo en cuenta que es para modelos de clasificación binarios. Así, generamos un conjunto de modelos xgboost que pueda clasificar cada una de las etiquetas binarias generadas anteriormente, haciéndole cumplir a cada uno de estos las restricciones de monotonía:

```r
mono_xbg <- function(data, nlab){
    models <- list()
    lab_vars <- paste0("Target",1:(nlab-1))
    for(lab_var in lab_vars){
        data_aux <- cbind(data[,!(names(data) %in% lab_vars)],
                          "C"=as.numeric(as.character(data[,lab_var])))
        dtrain <- xgboost::xgb.DMatrix(data = as.matrix(data_aux[,-ncol(data_aux)]),
                                       label = data_aux[,ncol(data_aux)])
        model <- xgboost::xgb.train(data = dtrain, objective = "binary:logistic",
                                    params = list(monotone_constraints=rep(1,ncol(data_aux)-1)),
                                    nrounds = 200)
        models[[lab_var]] <-  model
    }
    models
}
```

Se generan los conjuntos de modelos necesarios:

```
models_esl <- mono_xbg(data_lab_esl, length(labels_esl))
```

Predicción

Para llevar a cabo la predicción de esta clasificación monotónica, se sigue el algoritmo facilitado en el material de la asignatura. Así, la función resultante se muestra a continuación:

```
mono_predict <- function(models, newdata, labels=c(1:(length(models)+1))){
    dtest <- xgb.DMatrix(data = as.matrix(newdata))
    res <- data.frame(matrix(0, ncol = 0, nrow = nrow(newdata)))
    for(i in 1:(length(models))){
        prediction <- predict(models[[i]], dtest)
        prediction <- as.numeric(prediction > 0.50)
        res <- cbind(res, prediction)
    }
    rowSums(res)+1
}
```

Así, se predice:

```
pred_esl <- mono_predict(models_esl, test_esl[,-ncol(test_esl)], labels = labels_esl)
head(pred_esl, 30)
```

```
##  [1] 3 5 4 2 6 3 6 4 5 7 4 6 5 7 6 6 5 5 6 7 6 6 7 6 6 5 4 6 5 7
```

Resultados

Los resultados concretos de la ejecución de todos los pasos anteriores se muestran a continuación para el dataset **esl**:

```
sum(pred_esl == test_esl[,ncol(test_esl)])/nrow(test_esl)
```

```
## [1] 0.7042254
```

Cabe decir que con respecto a la primera parte de la práctica correspondiente a clasificación ordinal, el resultado mejora en una cuantía significativa para este conjunto de datos.