Problemas no estándar de clasificación supervisada

Exploración y análisis de datos

Jerónimo Hernández González

4 de octubre de 2017

Máster Universitario en Ingeniería
Computacional y Sistemas Inteligentes

1. Clasificación multi etiqueta

En este tutorial introduciremos el estudio de problemas de clasificación multi etiqueta. Para ello, usaremos el paquete **mlr** de R y el dataset Yeast:

```
library(mlr)

dataset <- getTaskData(yeast.task)
tarea <- makeMultilabelTask(id = "multi", data = dataset,
    target = colnames(dataset)[1:14])
tarea
```

Código 1: Cargamos el dataset

El dominio "Yeast" está formado por datos de expresión de micro-arrays, hasta 103 variables para caracterizar un total de 2417 genes. Cada gen está asociado a un conjunto de clases funcionales (etiquetas). La función **makeMultilabelTask**, nativa del paquete **mlr**, transforma los datos facilitados a un formato propio del paquete.

Para caracterizar el dataset, atenderemos al conteo de etiquetas (número de apariciones de cada etiqueta), distribución del número de etiquetas por instancia o los cómo se agrupan habitualmente las etiquetas. Esta última medida da una impresión de la correlación entre etiquetas.

```
labelCount <- colSums(dataset[,1:14])
labelCount

nLabelsPerInst <- rowSums(dataset[, 1:14])
nLabelsPerInst
table(nLabelsPerInst)

labelsets <- do.call(paste, c(dataset[, 1:14]+0, sep = ""))
labelsets <- table(as.factor(labelsets))
labelsets
```

Código 2: Caracterizando los datos

Otras medidas tradicionales requieren cierta computación. Programaremos, por ejemplo, la cardinalidad, que sigue la siguiente fórmula:

$$card = \frac{1}{N}\sum_{i=1}^{N}|\mathbf{y}_i|$$

donde N es el número de instancias (casos) del dataset, $\mathbf{y}_i$ el conjunto de etiquetas asociado a la instancia $\mathbf{x}_i$ y el operador $|\cdot|$ calcula el número de elementos en el conjunto.

```
# Label cardinality : Media de etiquetas por instancia
labelCardinality <- function(dataset) {
  lc <- 0
```

```
  return(lc)
}

labelCardinality(dataset)
```

Código 3: Caracterizando los datos: cardinalidad

También podemos calcular la densidad, que sigue la siguiente fórmula:

$$card = \frac{1}{N}\sum_{i=1}^{N}\frac{|\mathbf{y}_i|}{|L|}$$

donde L es el conjunto completo de posibles etiquetas.

```
# Label density : Media de la proporción de etiquetas por
    instancia
labelDensity <- function(dataset) {
  lc <- 0

  return(lc)
}

labelDensity(dataset)
```

Código 4: Caracterizando los datos: densidad

A parte de los algoritmos expresamente adaptados al aprendizaje multi etiqueta, las mayorías de estrategias de aprendizaje están basadas en *ensembles* (conjuntos) de clasificadores que tratan sólo con un subproblema. Estos clasificadores, conocidos como clasificadores base, pueden ser de cualquier tipo. No obstante, suelen usarse clasificadores que devuelvan una probabilidad junto a la predicción para tener más información a la hora de combinar las diferentes predicciones. Entre los clasificadores implementados en este paquete encontramos:

- classif.rpart : Arboles de decision
- classif.nnet : Redes Neuronales
- classif.logreg : Regresion Logistica
- classif.knn : K vecinos mas cercanos

Se puede usar la función **listLearners()** para conocer todos los clasificadores base que se pueden usar. Seleccionamos así el tipo de clasificador base:

```
clasifBase <- makeLearner("classif.rpart", predict.type="prob")
clasifBase
```

Código 5: Seleccionar clasificador base

A continuación, debemos seleccionar la técnica de aprendizaje. Son varias las implementadas en este paquete. Las aquí seleccionadas son las principales:

- Binary relevance (**makeMultilabelBinaryRelevanceWrapper**): Convertir el problema multi-etiqueta (L etiquetas) en L subproblemas de clasificación binaria. Cada subproblema se resuelve independientemente con un clasificador estándar binario.
- Classifier chains (**makeMultilabelClassifierChainsWrapper**): Igualmente, tenemos L clasificadores. Difiere en que la prediccion del clasificador 'l' se usa como variable predictora de los clasificadores subsecuentes ('l+1',...,L). Por lo tanto, es necesario establecer un orden entre las etiquetas y sus respectivos clasificadores. Esta técnica es capaz de modelar dependencias entre etiquetas.
- Stacking (**makeMultilabelStackingWrapper**): Igual que Dependent Binary Relevance, pero no usa tampoco las etiquetas reales en fase de aprendizaje. Usa un clasificador inicial basado en Binary Relevance para obtener una predicción de las etiquetas y con esas predicciones aprende el Dependent Binary Relevance.
- Nested stacking (**makeMultilabelNestedStackingWrapper**): Igual que Classifier Chains, pero no usa tampoco las etiquetas reales en fase de aprendizaje. Aprende el primer clasificador, obtiene una predicción de las etiquetas y con esas predicciones crea la columna de los subsecuentes clasificadores.

Una de las debilidades de este paquete es la falta de técnicas basadas en subconjuntos de etiquetas (ej., *Label Powerset* o *Random k-labelsets*). La selección de la técnica de aprendizaje se realiza de la siguiente manera:

```
tecnica <- makeMultilabelBinaryRelevanceWrapper(clasifBase)
tecnica
```

Código 6: Selección de la técnica de aprendizaje

Una vez están seleccionadas la técnica y el clasificador, podemos proceder a aprender el modelo y predecir la(s) etiqueta(s) de los casos que hemos dejado separados para test:

```
# Aprendizaje (con 1500 casos)
modelo = train(tecnica, tarea, subset = 1:1500)
modelo

# Prediccion (con 917 casos)
prediccion <- predict(modelo, newdata = dataset[1501:nrow(
    dataset),])
t(prediccion$data)
```

Código 7: Aprendizaje y clasificación

NOTA: Podemos intentar usar otros clasificadores base y técnicas de aprendizaje.

A la hora de evaluar un clasificador aprendido para un problema multi etiqueta, se pueden elegir entre dos estrategias: evaluar a nivel global o a nivel de etiqueta. Entre las medidas de evaluación especificas de clasificación multi etiqueta podemos elegir, usando la función **performance**, las siguientes:

- *multilabel.subset01*: Proporción de casos donde se acierta
- *multilabel.hamloss*: Hamming Loss - Proporción de etiquetas fuera de la intersección con respecto al total de etiquetas
- *multilabel.acc*: Media de la intersección entre la union de los sets de etiquetas reales y predicho (jaccard)
- *multilabel.f1*: Micro-F1 - Media armonica de precision y recall

La lista completa de medidas específicas de evaluación puede ser consultada usando la función **listMeasures("multilabel")**. Al evaluar a nivel de etiqueta podemos usar medidas "genéricas" de clasificación supervisada aplicadas al acierto en cada etiqueta individualmente. Entre este tipo de medidas de evaluación podemos elegir, usando la función **getMultilabelBinaryPerformances**, las siguientes:

- *acc*: Porcentaje de acierto
- *mmce*: Error de clasificación
- *tpr*: Recall
- *ppv*: Precisión
- *f1*: Media armónica de precisión y recall

La lista completa de medidas de evaluación puede ser consultada usando la función **listMeasures()**. La evaluación se realiza de la siguiente manera:

```
# Evaluación específica multi etiqueta
performance(prediccion, measures = list(multilabel.subset01,
    multilabel.hamloss, multilabel.acc,multilabel.f1))


# Evaluación tradicional aplicada a los clasificadores base
# (independientemente por etiqueta)
getMultilabelBinaryPerformances(prediccion, measures=list(acc,
    mmce,tpr,ppv,f1))
```

Código 8: Cálculo de varias distancia entre dos series

2. EM

Como práctica final, vamos a programar el algoritmo EM para aprender un sencillo Naive Bayes. Empezaremos, entonces, por programar este clasificador. Guardaremos en una lista el número de variables predictoras y los parámetros que usa: $p(c = 0)$, $p(c = 1)$, $p(x_i = 0|c = 0)$, $p(x_i = 1|c = 0)$, $p(x_i = 0|c = 1)$ y $p(x_i = 1|c = 1)$ (para todas las variables predictoras, i):

```
naiveBayes <- function(nVarPreds) {
  modelo <- list()

  modelo$nPredictors <- nVarPreds

  modelo$Pc <- vector("numeric",length=2)

  modelo$Px_c <- matrix(0,nVarPreds,2*2)

  return(modelo)
}
```

Código 9: Inicializar Naive Bayes

Una vez diseñado, vamos a lo importante. Un clasificador ha de poder ser aprendido y usado para predecir. Por ello, diseñaremos una función para cada procedimiento. La función de predicción sería así:

```
predecir <- function(instancia, modelo) {
  probs <- vector("numeric",length=2)

  probs[1] <- modelo$Pc[1]
  probs[2] <- modelo$Pc[2]

  for (i in 1:modelo$nPredictors) {
    probs[1] <- probs[1]*modelo$Px_c[i,1+instancia[i]]
    probs[2] <- probs[2]*modelo$Px_c[i,3+instancia[i]]
  }

  return(probs/sum(probs))
}
```

Código 10: Inicializar Naive Bayes

Donde se calculan las siguiente probabilidades:

$$p(c = 0) \prod_{i=1}^{v} p(x_i|c = 0)$$

$$p(c = 1) \prod_{i=1}^{v} p(x_i|c = 1)$$

con v siendo el número de variables predictoras (2 en este caso). Finalmente, normalizamos para que la suma de $p(c = 0|x)$ y $p(c = 1|x)$ sea 1.

Considerando la estructura definida para guardar los parámetros y lo visto en clase sobre aprendizaje de parámetros, diseñaremos la siguiente función:

```
aprender <- function(dataset, modelo) {

  modelo$Pc <- vector("numeric",length=2)
  modelo$Px_c <- matrix(0,modelo$nPredictors,2*2)

  ####################
  ## Tu código aquí ##
  ####################

  return(modelo)
}
```

Código 11: Inicializar Naive Bayes

Por último, el esqueleto del algoritmo EM es el siguiente:

```
EM <- function(datos, epsilon) {
  cDataset <- inicializar(datos)


  modelo <- aprender(cDataset,modelo)

  convergencia <- FALSE
  while (!convergencia) {
    cDataset <- EStep(cDataset,modelo)

    antModelo <- modelo
    modelo <- MStep(cDataset, antModelo)

    convergencia <- testConvergencia(modelo,antModelo,epsilon)
  }

  return(list(modelo,cDataset))
}
```

Código 12: Esqueleto EM

Faltarían por programar las tres funciones fundamentales del EM: **EStep**, **MStep** y **testConvergencia**. Mediremos la convergencia atendiendo al siguiente criterio: la distancia euclídea entre los parámetros del paso anterior y del actual es menor que *epsilon* o no.

```
EStep <- function(dataset, modelo){

  dataset$peso <- vector("numeric",length=length(dataset$peso))

  ####################
  ## Tu código aquí ##
  ####################
```

```
  return(dataset)
}


MStep <- function(dataset, modelo){

  ####################
  ## Tu código aquí ##
  ####################

  return(modelo)
}

testConvergencia <- function(modeloA, modeloB, epsilon) {

  resultado <- FALSE

  ####################
  ## Tu código aquí ##
  ####################

  return ( resultado )
}
```

Código 13: Ejecución de la práctica

Con el objetivo de hacerlo comparativo, todos ejecutaremos el algoritmo realizando la siguiente llamada:

```
# Inicializamos el model con 2 variables predictoras
modelo <- naiveBayes(2)
# Simulamos un dataset
datos <- cbind(c(0,1,0,0,1,1,1,1,0,1),
               c(0,0,1,1,0,0,0,1,1,1),
               c(1,NA,NA,NA,1,0,NA,NA,0,NA))

EM(datos, 0.001)
```

Código 14: Ejecución de la práctica