Clasificación no balanceada

Laura del Pino Díaz

5/2/2017

Índice

Clasificación no balanceada

La clasificación no balanceada se produce cuando queremos diferenciar al menos dos clases pero de una de ellas tenemos muy pocos ejemplos en comparación con las restantes clases.

Objetivos

En esta práctica se persiguen los siguientes objetivos: * Preparar los datos para la clasificación. * Implementar la estrategia *Random Oversampling* (ROS) * Implementar la estrategia *Random Undersampling* (RUS) * Implementar la estrategia *Synthetic Minority Oversampli Technique* (SMOTE)

Carga de los conjuntos de datos

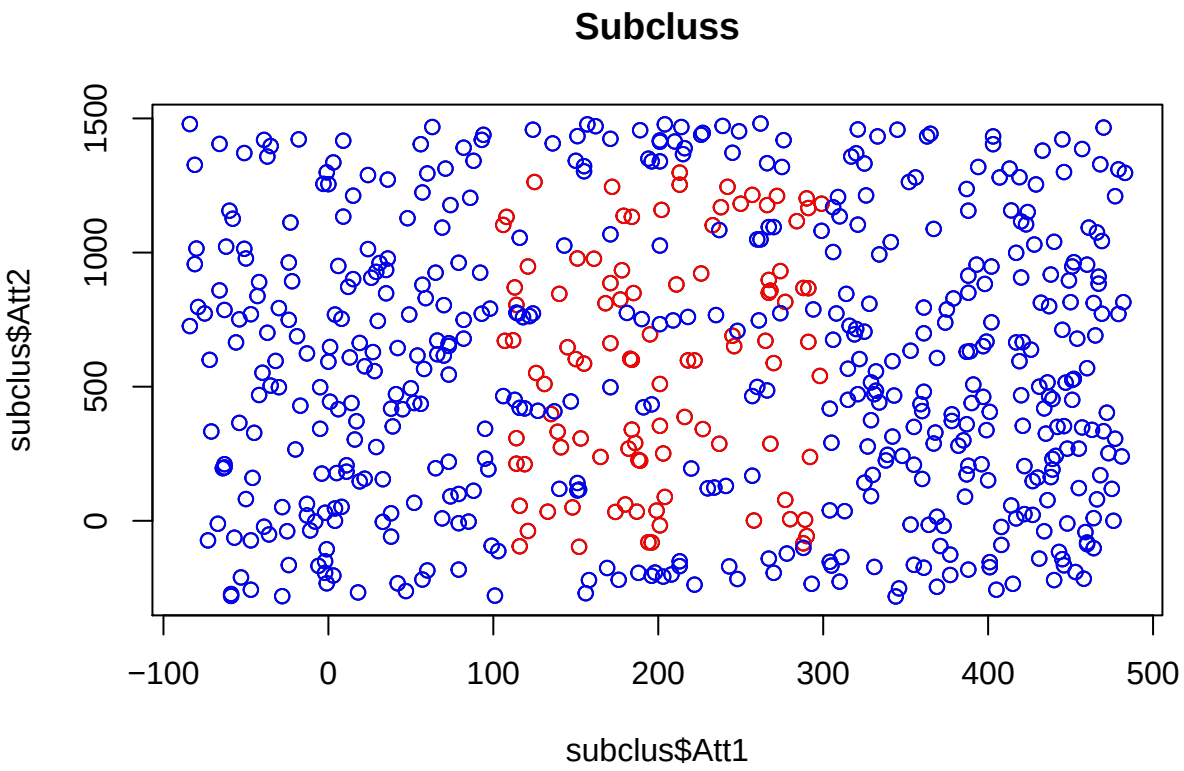
Los conjuntos de datos que vamos a utilizar en esta práctica son *sublus* y *circle*. Se procede a su carga:

```
subclus <- read.table("subclus.txt", sep = ",")
colnames(subclus) <- c("Att1", "Att2", "Class")

circle <- read.table("circle.txt", sep = ",")
colnames(circle) <- c("Att1", "Att2", "Class")
```

Estudio del desbalanceo

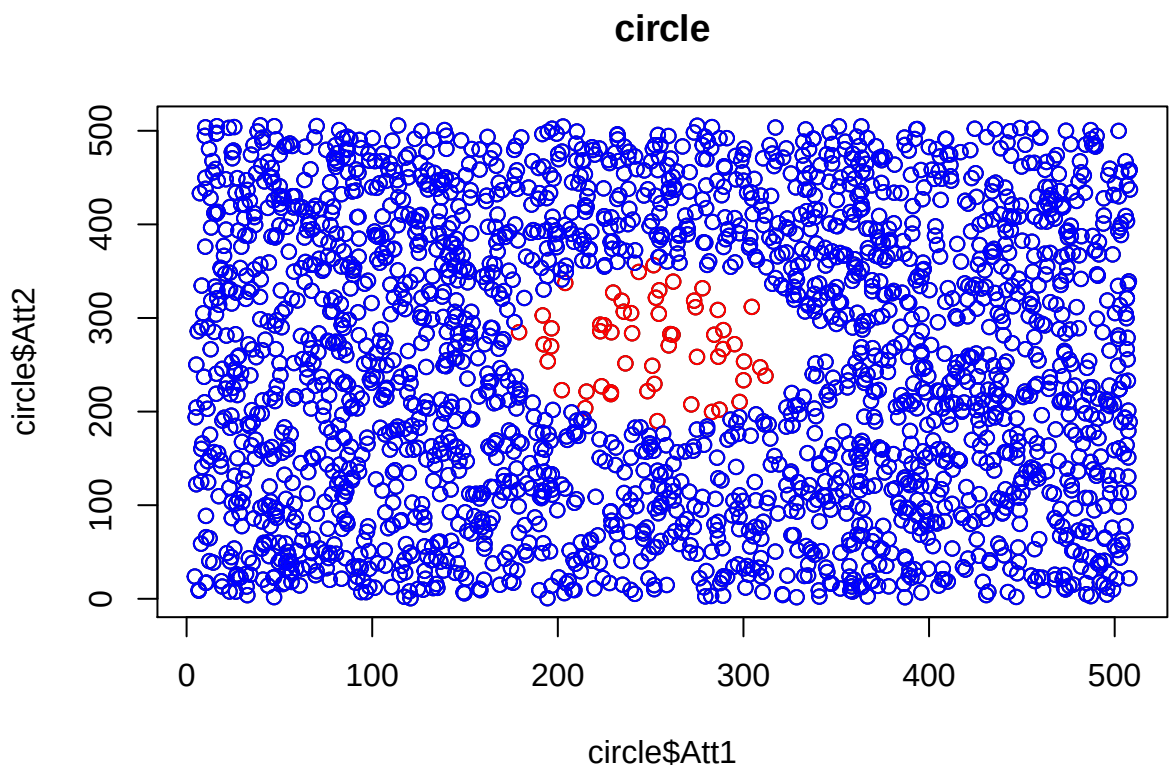
Descubrir que un conjunto de datos está desbalanceado puede darse en la fase de visualización. Visualizamos las dos bases de datos:

```
# subcluss
plot(subclus$Att1, subclus$Att2, main = "Subcluss")
points(subclus[subclus$Class == 0, 1], subclus[subclus$Class ==
    0, 2], col = "red")
points(subclus[subclus$Class == 1, 1], subclus[subclus$Class ==
    1, 2], col = "blue")
```



Como podemos ver tenemos una clase minoritaria coloreada en rojo en medio de todo el espacio de puntos y con solapamiento entre esta clase y la mayoría.

```
plot(circle$Att1, circle$Att2, main = "circle")
points(circle[circle$Class == 0, 1], circle[circle$Class == 0,
    2], col = "red")
points(circle[circle$Class == 1, 1], circle[circle$Class == 1,
    2], col = "blue")
```



En el caso de la base de datos *circle* tenemos un problema más sencillo ya que no tenemos el problema añadido del solapamiento entre clases.

Medir el desbalanceo puede darnos una idea de cómo se distribuyen las clases en nuestro conjunto de datos. Una forma de representarlo es el índice de desbalanceo que procedemos a calcular:

```
getIR = function(classData) {
    nClass0 <- sum(classData == 0)
    nClass1 <- sum(classData == 1)
    IR <- nClass1/nClass0
    IR
}

IR_subcluss = getIR(subclus$Class)
IR_circle = getIR(circle$Class)

IR_subcluss
```

```
[1] 5
```

```
IR_circle
```

```
[1] 42.45455
```

Como podemos ver la clase 1 en ambos casos es la mayoritaria, siendo en el caso de la base de datos *subcluss* 5 veces mayor que la minoritaria y en el caso de la base de datos *circle* 42.45 veces más abundante los ejemplos de la clase mayoritaria.

Clasificador de control

Ahora que conocemos como se comportan los conjuntos de datos, vamos a lanzar un clasificador que nos muestre la problemática de clasificar los dos conjuntos en el estado actual.

Para mejorar las condiciones de entrenamiento del clasificador lo vamos a entrenar utilizando la técnica de validación cruzada de 5 particiones. Por ello creamos las particiones de cada conjunto de datos, respetando siempre el ratio de desbalanceo en cada una de las particiones.

```
# subcluss
pos <- (1:dim(subclus)[1])[subclus$Class == 0]
neg <- (1:dim(subclus)[1])[subclus$Class == 1]

SubclussCVperm_pos <- matrix(sample(pos, length(pos)), ncol = 5,
    byrow = T)
SubclussCVperm_neg <- matrix(sample(neg, length(neg)), ncol = 5,
    byrow = T)

SubclussCVperm <- rbind(SubclussCVperm_pos, SubclussCVperm_neg)

# circle

pos <- (1:dim(circle)[1])[circle$Class == 0]
neg <- (1:dim(circle)[1])[circle$Class == 1]

circleCVperm_pos <- matrix(sample(pos, length(pos)), ncol = 5,
    byrow = T)
circleCVperm_neg <- matrix(sample(neg, length(neg)), ncol = 5,
    byrow = T)

circleCVperm <- rbind(circleCVperm_pos, circleCVperm_neg)
```

Con las particiones preparadas entrenamos el clasificador

```
library(class)
knn.pred = NULL
for (i in 1:5) {
    predictions <- knn(subclus[-SubclussCVperm[, i], -3], subclus[SubclussCVperm[,
        i], -3], subclus[-SubclussCVperm[, i], 3], k = 3)
    knn.pred <- c(knn.pred, predictions)
}
acc <- sum((subclus$Class[as.vector(SubclussCVperm)] == 0 & knn.pred ==
    1) | (subclus$Class[as.vector(SubclussCVperm)] == 1 & knn.pred ==
    2))/(sum(subclus$Class == 0) + sum(subclus$Class == 1))
tpr <- sum(subclus$Class[as.vector(SubclussCVperm)] == 0 & knn.pred ==
    1)/sum(subclus$Class == 0)
tnr <- sum(subclus$Class[as.vector(SubclussCVperm)] == 1 & knn.pred ==
    2)/sum(subclus$Class == 1)
gmean <- sqrt(tpr * tnr)
gmean
```

```
[1] 0.8577179
```

Como se muestra el clasificador tienen una alta tasa de acierto perteneciente prácticamente en su totalidad al acierto de la clase mayoritaria. De hecho se espera que para el conjunto de datos *circle* sea mayor puesto que la clase mayoritaria es aún más abundante.

```
knn.pred = NULL
for (i in 1:5) {
    predictions <- knn(circle[-circleCVperm[, i], -3], circle[circleCVperm[,
        i], -3], circle[-circleCVperm[, i], 3], k = 3)
    knn.pred <- c(knn.pred, predictions)
}
acc <- sum((circle$Class[as.vector(circleCVperm)] == 0 & knn.pred ==
    1) | (circle$Class[as.vector(circleCVperm)] == 1 & knn.pred ==
    2))/(sum(circle$Class == 0) + sum(circle$Class == 1))
tpr <- sum(circle$Class[as.vector(circleCVperm)] == 0 & knn.pred ==
    1)/sum(circle$Class == 0)
tnr <- sum(circle$Class[as.vector(circleCVperm)] == 1 & knn.pred ==
    2)/sum(circle$Class == 1)
gmean <- sqrt(tpr * tnr)
gmean
```

```
[1] 0.8940441
```

Como se esperaba la tasa de acierto ha aumentado por el porcentaje de la clase mayoritaria, pero este crecimiento ha sido moderado puesto que la clase minoritaria no tiene solapamiento con la mayoritaria.

Random Oversampling (ROS)

La técnica *Random Oversampling* toma los valores que ya existen de la clase minoritaria y genera nuevos ejemplos duplicando los existentes. Aplicaremos esta técnica a los dos conjuntos de datos que ya tenemos y compararemos la gmean en ambos casos con la obtenida en el caso de control.

```
knn.pred = NULL
for (i in 1:5) {

    train <- subclus[-SubclussCVperm[, i], -3]
    classes.train <- subclus[-SubclussCVperm[, i], 3]
    test <- subclus[SubclussCVperm[, i], -3]

    # randomly oversample the minority class (class 0)
    minority.indices <- (1:dim(train)[1])[classes.train == 0]
    to.add <- dim(train)[1] - 2 * length(minority.indices)
    duplicate <- sample(minority.indices, to.add, replace = T)
    for (j in 1:length(duplicate)) {
        train <- rbind(train, train[duplicate[j], ])
        classes.train <- c(classes.train, 0)
    }

    # use the modified training set to make predictions
    predictions <- knn(train, test, classes.train, k = 3)
    knn.pred <- c(knn.pred, predictions)
}
tpr.ROS <- sum(subclus$Class[as.vector(SubclussCVperm)] == 0 &
    knn.pred == 1)/sum(subclus$Class == 0)
tnr.ROS <- sum(subclus$Class[as.vector(SubclussCVperm)] == 1 &
    knn.pred == 2)/sum(subclus$Class == 1)
gmean.ROS <- sqrt(tpr.ROS * tnr.ROS)
gmean.ROS
```

```
[1] 0.9118224
```

La gmean ha mejorado con respecto a la original aunque sigue manteniendo un margen de error debido al solapamiento entre clases.

```
knn.pred = NULL
for (i in 1:5) {

    train <- circle[-circleCVperm[, i], -3]
    classes.train <- circle[-circleCVperm[, i], 3]
    test <- circle[circleCVperm[, i], -3]

    # randomly oversample the minority class (class 0)
    minority.indices <- (1:dim(train)[1])[classes.train == 0]
    to.add <- dim(train)[1] - 2 * length(minority.indices)
    duplicate <- sample(minority.indices, to.add, replace = T)
    for (j in 1:length(duplicate)) {
        train <- rbind(train, train[duplicate[j], ])
        classes.train <- c(classes.train, 0)
    }

    # use the modified training set to make predictions
    predictions <- knn(train, test, classes.train, k = 3)
    knn.pred <- c(knn.pred, predictions)
}
tpr.ROS <- sum(circle$Class[as.vector(circleCVperm)] == 0 & knn.pred ==
    1)/sum(circle$Class == 0)
tnr.ROS <- sum(circle$Class[as.vector(circleCVperm)] == 1 & knn.pred ==
    2)/sum(circle$Class == 1)
gmean.ROS <- sqrt(tpr.ROS * tnr.ROS)
gmean.ROS
```

```
[1] 0.9418565
```

Al igual que en el caso anterior, la gmean ha mejorado al mejorar el balance de la clase minoritaria.

Random Undersampling (RUS)

La técnica de *Random Undersampling* equilibra las dos clases mediante el borrado aleatorio de registros de la clase mayoritaria. Aplicaremos esta técnica a ambos conjuntos de datos y compararemos la gmean resultante con la obtenida en los dos apartados anteriores.

```
knn.pred = NULL
for (i in 1:5) {

    train <- subclus[-SubclussCVperm[, i], -3]
    classes.train <- subclus[-SubclussCVperm[, i], 3]
    test <- subclus[SubclussCVperm[, i], -3]

    # randomly undersample the minority class (class 1)
    majority.indices <- (1:dim(train)[1])[classes.train == 1]
    to.remove <- 2 * length(majority.indices) - dim(train)[1]
    remove <- sample(majority.indices, to.remove, replace = F)
    train <- train[-remove, ]
    classes.train <- classes.train[-remove]
```

```
    # use the modified training set to make predictions
    predictions <- knn(train, test, classes.train, k = 3)
    knn.pred <- c(knn.pred, predictions)
}
tpr.RUS <- sum(subclus$Class[as.vector(SubclussCVperm)] == 0 &
    knn.pred == 1)/sum(subclus$Class == 0)
tnr.RUS <- sum(subclus$Class[as.vector(SubclussCVperm)] == 1 &
    knn.pred == 2)/sum(subclus$Class == 1)
gmean.RUS <- sqrt(tpr.RUS * tnr.RUS)
gmean.RUS
```

```
[1] 0.8826098
```

Se ha obtenido un valor de gmean situado entre el obtenido en el clasificador de control y el de ROS, esto es debido a que al eliminar registros de la clase mayoritaria para igualar el número de observaciones de ambas clases ha perdido información sobre la clase mayoritaria.

```
knn.pred = NULL
for (i in 1:5) {

    train <- circle[-circleCVperm[, i], -3]
    classes.train <- circle[-circleCVperm[, i], 3]
    test <- circle[circleCVperm[, i], -3]

    # randomly undersample the minority class (class 1)
    majority.indices <- (1:dim(train)[1])[classes.train == 1]
    to.remove <- 2 * length(majority.indices) - dim(train)[1]
    remove <- sample(majority.indices, to.remove, replace = F)
    train <- train[-remove, ]
    classes.train <- classes.train[-remove]

    # use the modified training set to make predictions
    predictions <- knn(train, test, classes.train, k = 3)
    knn.pred <- c(knn.pred, predictions)
}
tpr.RUS <- sum(circle$Class[as.vector(circleCVperm)] == 0 & knn.pred ==
    1)/sum(circle$Class == 0)
tnr.RUS <- sum(circle$Class[as.vector(circleCVperm)] == 1 & knn.pred ==
    2)/sum(circle$Class == 1)
gmean.RUS <- sqrt(tpr.RUS * tnr.RUS)
gmean.RUS
```

```
[1] 0.9433641
```

En este caso ha bajado ligeramente el valor de la gmean por el mismo motivo que en el caso anterior, porque pierde parte de la información de la clase mayoritaria, pero como en esta basa de datos los datos no están solapados esta diferencia no se nota demasiado.

Synthetic Minority Oversampling Technique (SMOTE)

La técnica de Synthetic Minority Oversampling aumenta el número de registros de la clase minoritaria creando observaciones entre los puntos ya existentes de la clase minoritaria.

El primer paso en el desarrollo de esta técnica es definir la distancia entre los puntos. Para ello tomamos la

distancia euclídea entre los atributos numéricos, y para los atributos categóricos tomamos como distancia 1 siempre que no tengan la misma categoría.

```
distance <- function(i, j, data) {
    sum <- 0
    for (f in 1:dim(data)[2]) {
        if (is.factor(data[, f])) {
            # nominal feature
            if (data[i, f] != data[j, f]) {
                sum <- sum + 1
            }
        } else {
            sum <- sum + (data[i, f] - data[j, f]) * (data[i,
                f] - data[j, f])
        }
    }
    sum <- sqrt(sum)
    return(sum)
}
```

Una vez tenemos la función de distancia, tenemos que seleccionar aquellos puntos que nos ayudarán a crear los nuevos ejemplos, es decir los vecinos al punto que estamos considerando.

```
getNeighbours = function(x, minorityIndices, train) {
    minority = train[minorityIndices, ]
    distances = NULL
    for (i in 1:(dim(minority)[1])) {
        distances = c(distances, distance(x, rownames(minority[i,
            ]), train))
    }
    indexInMinority = which(distances %in% sort(distances)[1:5])
    return(minority[indexInMinority, ])
}
```

Con la selección de los vecinos más cercanos podemos crear nuevas instacias entre ellos.

```
syntheticInstance = function(x, nearestNeighbours) {
    chosenNeighbourIndex = ceiling(runif(1) * 5)
    chosenNeighbour = nearestNeighbours[chosenNeighbourIndex,
        ]
    result = NULL
    for (i in 1:length(x)) {
        if (is.factor(x[i])) {
            result = c(result, ifelse(runif(1) > 0.5, x[i], chosenNeighbour[i]))
        } else {
            increment = abs(as.numeric(x[i]) - as.numeric(chosenNeighbour[i])) *
                runif(1)
            minimum = min(as.numeric(x[i]), as.numeric(chosenNeighbour[i]))
            result = c(result, minimum + increment)
        }
    }
    return(result)
}
```

Una vez que tenemos todas las funciones solamente queda ensamblar para tener el método SMOTE. Este método asume que solo hay dos clases, una mayoritaria y otra minoritaria y a parte de los datos se le pasa el

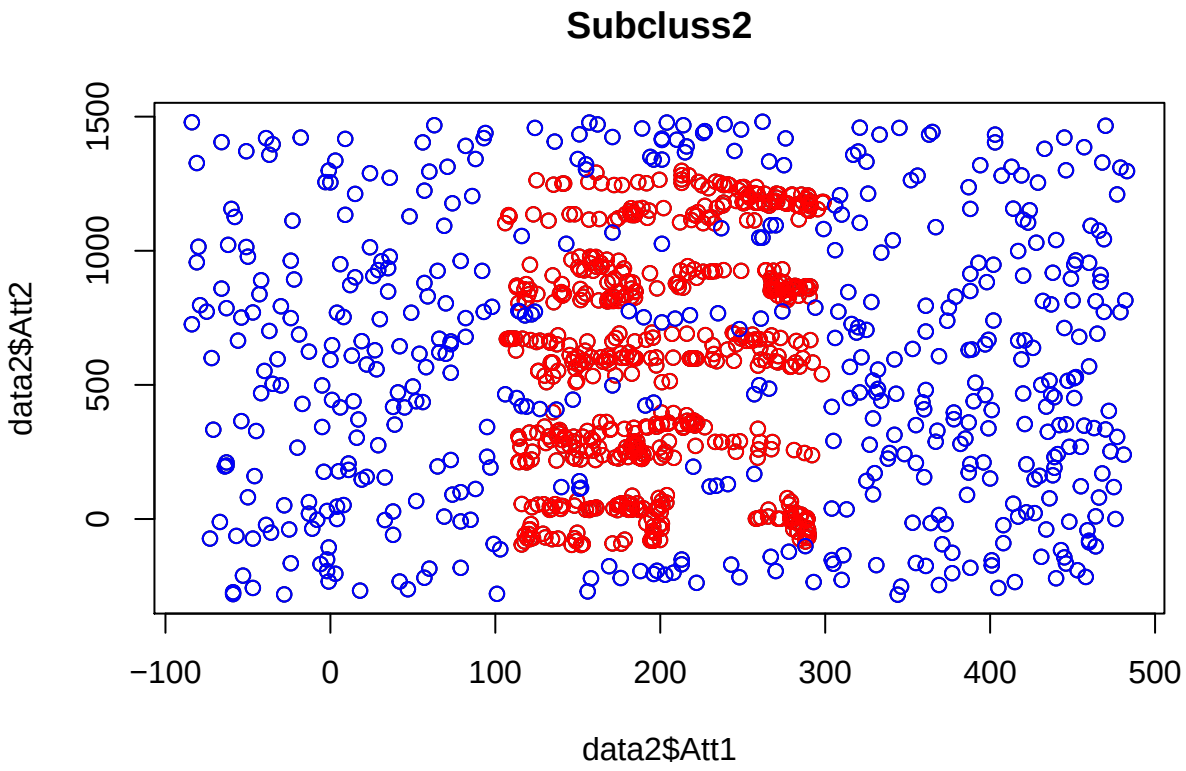
número de la columna que contiene las etiquetas.

El método SMOTE hará que por cada observación de la clase minoritaria, se creen tantos puntos como el valor de IR que se calculó al principio, es decir si para el conjunto de datos *subclus* la clase mayoritaria era 4 veces mayor que la clase minoritaria, se crearán 4 nuevas observaciones para igualar el número de observaciones.

```
smote = function(data, labelIndex) {
    minorityIndexes = which(data[, labelIndex] == 0)

    for (ir in 1:ceiling(getIR(data[, labelIndex]))) {
        for (i in 1:length(minorityIndexes)) {
            nnI = getNeighbours(minorityIndexes[i], minorityIndexes[-i],
                data)
            siI = syntheticInstance(data[minorityIndexes[i],
                ], nnI)
            data = rbind(data, siI)
        }
    }

    return(data)
}
```

Aplicamos SMOTE a la base de datos de *subclus* y obtenemos la siguiente gráfica.

```
data2 = smote(subclus, 3)
plot(data2$Att1, data2$Att2, main = "Subcluss2")
points(data2[data2$Class == 0, 1], data2[data2$Class == 0, 2],
    col = "red")
points(data2[data2$Class == 1, 1], data2[data2$Class == 1, 2],
    col = "blue")
```



Como podemos ver el método ha colocado los puntos cerca de sus vecinos, reforzando así la clase. Pasamos a

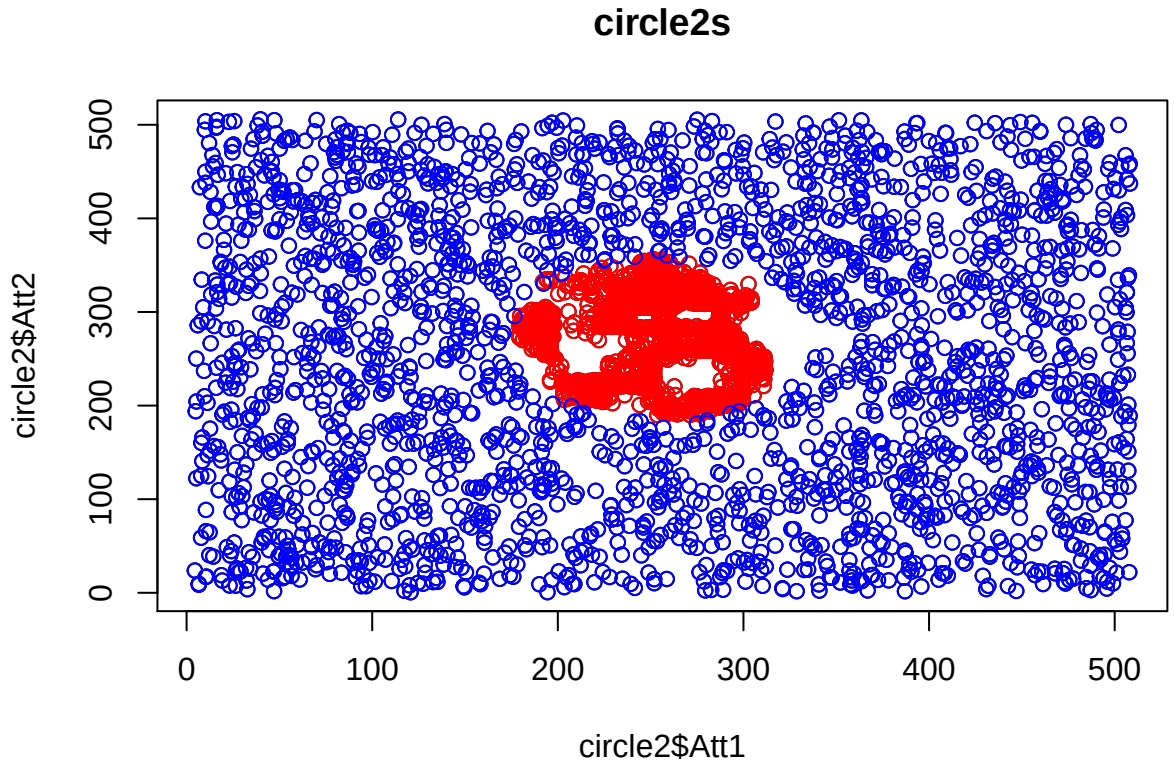
obtener otra vez las particiones y la gmean:

```
pos <- (1:dim(data2)[1])[data2$Class == 0]
neg <- (1:dim(data2)[1])[data2$Class == 1]

d2CVperm_pos <- matrix(sample(pos, length(pos)), ncol = 5, byrow = T)
d2CVperm_neg <- matrix(sample(neg, length(neg)), ncol = 5, byrow = T)

d2CVperm <- rbind(SubclussCVperm_pos, SubclussCVperm_neg)


knn.pred = NULL
for (i in 1:5) {
    predictions <- knn(data2[-d2CVperm[, i], -3], data2[d2CVperm[,
        i], -3], data2[-d2CVperm[, i], 3], k = 3)
    knn.pred <- c(knn.pred, predictions)
}
acc <- sum((data2$Class[as.vector(d2CVperm)] == 0 & knn.pred ==
    1) | (data2$Class[as.vector(d2CVperm)] == 1 & knn.pred ==
    2))/(sum(data2$Class == 0) + sum(data2$Class == 1))
tpr <- sum(data2$Class[as.vector(d2CVperm)] == 0 & knn.pred ==
    1)/sum(data2$Class == 0)
tnr <- sum(data2$Class[as.vector(d2CVperm)] == 1 & knn.pred ==
    2)/sum(data2$Class == 1)
gmean <- sqrt(tpr * tnr)
gmean
```

```
[1] 0.3887673
```

Al reforzar la clase minoritaria que está solapada con la clase mayoritaria, el modelo de clasificación de los k-vecinos más cercanos está dando problemas a la hora de determinar si pertenece a una clase o a otra, por lo que aumenta la tasa de error.

```
circle2 = smote(circle, 3)
plot(circle2$Att1, circle2$Att2, main = "circle2s")
points(circle2[circle2$Class == 0, 1], circle2[circle2$Class ==
    0, 2], col = "red")
points(circle2[circle2$Class == 1, 1], circle2[circle2$Class ==
    1, 2], col = "blue")
```



Al igual que en el caso anterior obtenemos que la clase minoritaria está reforzada con nuevos ejemplos similares a los originales. Si ahora obtenemos la gmean tras haber hecho la validación cruzada tenemos como resultado:

```
pos <- (1:dim(circle2)[1])[circle2$Class == 0]
neg <- (1:dim(circle2)[1])[circle2$Class == 1]

circle2CVperm_pos <- matrix(sample(pos, length(pos)), ncol = 5,
    byrow = T)
circle2CVperm_neg <- matrix(sample(neg, length(neg)), ncol = 5,
    byrow = T)

circle2CVperm <- rbind(circle2CVperm_pos, circle2CVperm_neg)

knn.pred = NULL
for (i in 1:5) {
    predictions <- knn(circle2[-circle2CVperm[, i], -3], circle2[circle2CVperm[,
        i], -3], circle2[-circle2CVperm[, i], 3], k = 3)
    knn.pred <- c(knn.pred, predictions)
}
acc <- sum((circle2$Class[as.vector(circle2CVperm)] == 0 & knn.pred ==
    1) | (circle2$Class[as.vector(circle2CVperm)] == 1 & knn.pred ==
    2))/(sum(circle2$Class == 0) + sum(circle2$Class == 1))
tpr <- sum(circle2$Class[as.vector(circle2CVperm)] == 0 & knn.pred ==
    1)/sum(circle2$Class == 0)
tnr <- sum(circle2$Class[as.vector(circle2CVperm)] == 1 & knn.pred ==
    2)/sum(circle2$Class == 1)
gmean <- sqrt(tpr * tnr)
gmean
```

```
[1] 0.9967828
```

Por el contrario reforzar la clase minoritaria en la base de datos *circle* ayuda al clasificador a diferenciar mejor las dos clases, permitiendole tener casi el 100 % de aciertos.

Conclusión.

El método ROS mejora la clasificación creando nuevos ejemplos en puntos donde ya hay puntos, lo que es malo si las clases se solapan porque no aporta nada nuevo a la clasificación y bueno si están bien definidas porque refuerzan la zona del espacio en la que se encuentran las observaciones y las fronteras del mismo.

El método RUS al eliminar ejemplos de la clase mayoritaria, haciendo que el conjunto de datos sea menor, no favorece la clasificación puesto que pierde información sobre la clase mayoritaria.

El método SMOTE al generar nuevos ejemplos de la clase minoritaria, similares pero no iguales a los ya existentes ayuda a generar nueva información sobre esta clase y a diferenciarla de la mayoritaria. No obstante si las clases están solapadas, puede disminuir el número de ejemplos bien clasificados.