imbalanceado

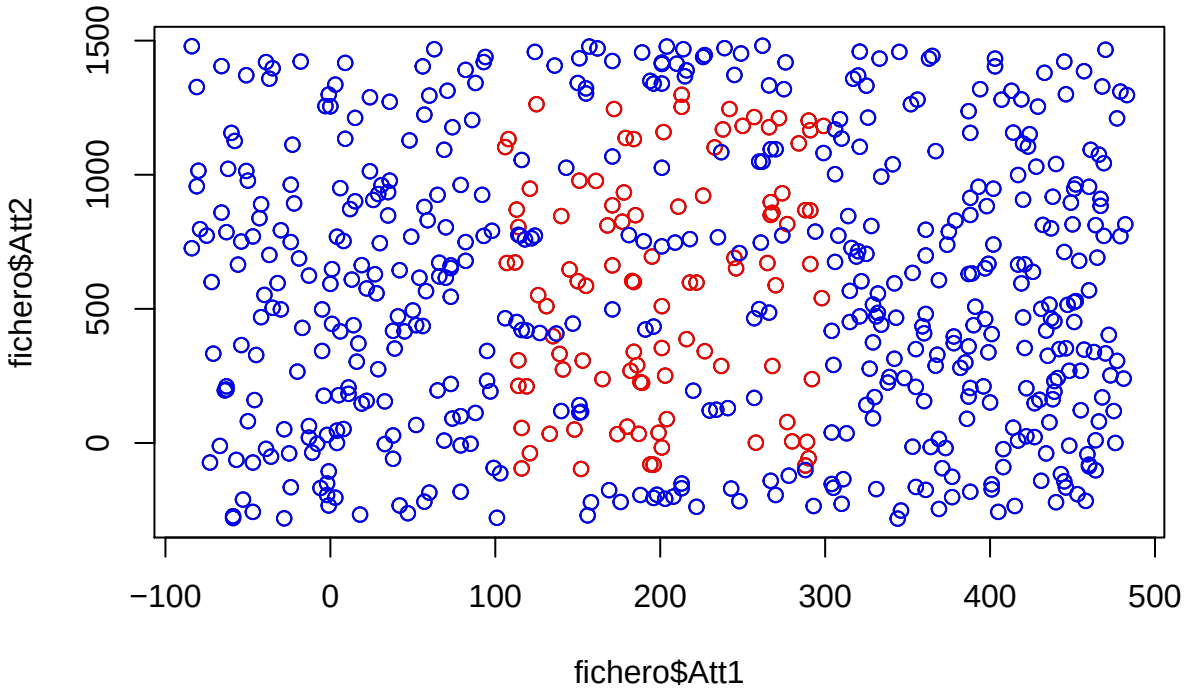
```
#ImplementaciÃ³n y evaluaciÃ³n de tÃ©cnicas de clasificaciÃ³n imbalanceada
#Leemos el dataset fichero
fichero <- read.table("subclus.txt", sep=",")
colnames(fichero) <- c("Att1", "Att2", "Class")

#Determinar el radio de imbalanceamiento
unique(fichero$Class)
```

```
## [1] 0 1
```

```
nClass0 <- sum(fichero$Class == 0)
nClass1 <- sum(fichero$Class == 1)
IR <- nClass1 / nClass0
IR #Por cada ejemplo positivo hay 5 negativos
```

```
## [1] 5
```

```
#Visualizamos la distribuciÃ³n de los datos
plot(fichero$Att1, fichero$Att2)
points(fichero[fichero$Class==0,1],fichero[fichero$Class==0,2],col="red")
points(fichero[fichero$Class==1,1],fichero[fichero$Class==1,2],col="blue")
```




```
#Dividimos el dataset en 5 partes para la validaciÃ³n cruzada
set.seed(1234)
```

```r
pos <- (1:dim(fichero)[1])[fichero$Class==0]
neg <- (1:dim(fichero)[1])[fichero$Class==1]
#Hacemos las divisiones en los 5 conjuntos de cada clase
CVperm_pos <- matrix(sample(pos,length(pos)), ncol=5, byrow=T)
CVperm_neg <- matrix(sample(neg,length(neg)), ncol=5, byrow=T)
#Unimos las dos clases
CVperm <- rbind(CVperm_pos, CVperm_neg)

#Llevamos a cabo el 3NN
library(class)
knn.pred = NULL
set.seed(1234)
for( i in 1:5){
  predictions <- knn(fichero[-CVperm[,i], -3], fichero[CVperm[,i], -3], fichero[-CVperm[,i], 3], k = 3)
  knn.pred <- c(knn.pred, predictions)
}
acc <- sum((fichero$Class[as.vector(CVperm)] == 0 & knn.pred == 1)
           | (fichero$Class[as.vector(CVperm)] == 1 & knn.pred == 2)) / (nClass0 + nClass1)
acc
```

```
## [1] 0.93
```

```r
#parece que se obitene un buen rendimiento ya que tenemos un accuracy del 0.93

tnr <- sum(fichero$Class[as.vector(CVperm)] == 1 & knn.pred == 2) / nClass1
tnr
```

```
## [1] 0.968
```

```r
#es bueno para la clase negativa al obtener un 0.968
tpr <- sum(fichero$Class[as.vector(CVperm)] == 0 & knn.pred == 1) / nClass0
tpr
```

```
## [1] 0.74
```

```r
#no es tan bueno para la case positiva ya que obtiene un 0.74
gmean <- sqrt(tpr * tnr) # no es tan bueno como decia el accuracy
gmean
```

```
## [1] 0.8463569
```

```r
#Obtenemos de media un 0.846

# 1. Vamos a aplicar ROS, de forma que replicaremos los ejemplos positivos
knn.pred = NULL
set.seed(1234)
aplicarROS <- function(){
  set.seed(1234)
  for( i in 1:5){
    train <- fichero[-CVperm[,i], -3]
    classes.train <- fichero[-CVperm[,i], 3]
    test  <- fichero[CVperm[,i], -3]
    #Aleatoriamente se hace oversample de la clase minoritaria (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)
  }
  return(knn.pred)
}
knn.pred <- aplicarROS()

tpr.ROS <- sum(fichero$Class[as.vector(CVperm)] == 0 & knn.pred == 1) / nClass0
tpr.ROS
```

```
## [1] 0.92
```

```
#Hemos pasado de un 0.74 a un 0.92 mejorando mucho en la clase positiva
tnr.ROS <- sum(fichero$Class[as.vector(CVperm)] == 1 & knn.pred == 2) / nClass1
tnr.ROS
```

```
## [1] 0.896
```

```
#Hemos pasado de un 0.968 a un 0.896 en la clase negativa
gmean.ROS <- sqrt(tpr.ROS * tnr.ROS)
gmean.ROS
```

```
## [1] 0.9079207
```

```
#La media ha pasado de 0.846 a 0.907, mejorando

# 2. Vamos a aplicar RUS, de forma que quito elementos de la clase mayoritaria
set.seed(1234)
knn.pred = NULL
aplicarRUS <- function(){
  for( i in 1:5){
    train <- fichero[-CVperm[,i], -3]
    classes.train <- fichero[-CVperm[,i], 3]
    test  <- fichero[CVperm[,i], -3]
    #Aleatoriamente aplicamos undersample a la clase minoritaria (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]
    #Uso el training set modificado para hacer las predicciones
    predictions <-  knn(train, test, classes.train, k = 3)
    knn.pred <- c(knn.pred, predictions)
  }
  return(knn.pred)
}
knn.pred = aplicarRUS()
tpr.RUS <- sum(fichero$Class[as.vector(CVperm)] == 0 & knn.pred == 1) / nClass0
tpr.RUS
```

```
## [1] 0.96
```

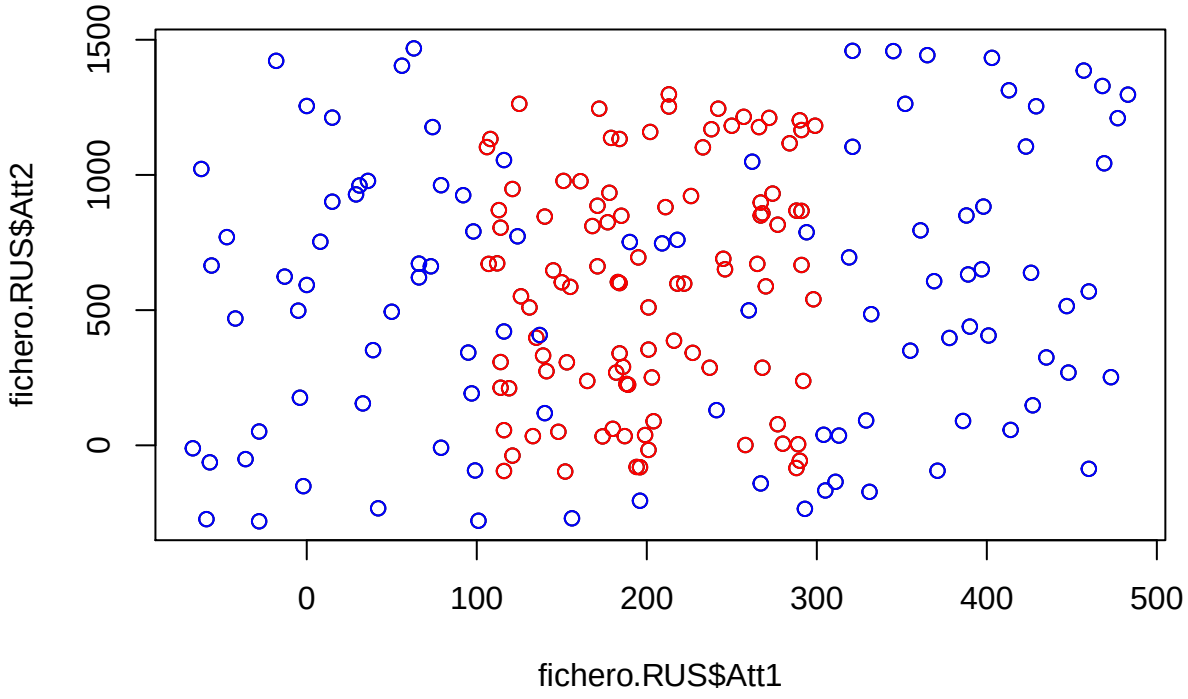
```
#Hemos pasado de 0.92 a 0.96
tnr.RUS <- sum(fichero$Class[as.vector(CVperm)] == 1 & knn.pred == 2) / nClass1
tnr.RUS
```

```
## [1] 0.804
```

```
#Hemos pasado de 0.896 a 0.804
gmean.RUS <- sqrt(tpr.RUS * tnr.RUS)
gmean.RUS
```

```
## [1] 0.8785443
```

```
#Hemos bajado el rendimiento con respecto a ROS, ya que ha bajado de 0.907 a 0.878,
#pero se ha mejorado al inicial sin aplicar ninguna tÃ©cnica

#Visualizamos como quedarÃ?a la tÃ©cnica RUS en el dataset
fichero.RUS <- fichero
majority.indices <- (1:dim(fichero.RUS)[1])[fichero.RUS$Class == 1]
to.remove <- 2 * length(majority.indices) - dim(fichero.RUS)[1]
remove <- sample(majority.indices, to.remove, replace = F)
fichero.RUS <- fichero.RUS[-remove,]

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




```
# 1.4.1 FunciÃ³n de distancia
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)
}

#FunciÃ³n GetNeighbors
getNeighbors <- function(x, minority.instances, train){
  #1 Por cada x, recorro todas las instancias minoritarias y busco sus 5 elementos mÃ¡s cercanos
  #1.1 Creo la funciÃ³n que devuelva la distancia de x a cada elemento de la clase minoritaria
  cercano = function(y){
    distance(x,y,fichero)
  }
  #1.2 Aplico para encontrar la distancia
  #points(fichero[x,],col="black")
  distancias <- sapply(minority.instances,cercano )
  #1.3 Busco los 5 elementos con menor distancia salvo el mismo
  matriz.elemento.distancia <- cbind(minority.instances,distancias)
  menores = function(matriz){
    #Ordeno las distancias
    a = matriz[,2]
    a <- a[order(a)]
    #Si la primera distancia es 0, es que es el mismo y lo elimino
    if (a[1]==0){
      a <- c(a[2:length(a)])
    }
    #Me quedo con las 5 mÃ¡s pequeÃ±as
    a <- c(a[1:5])
    return(a)
  }
  los.menores <- menores(matriz.elemento.distancia)
  #matriz.elemento.distancia[los.5.cercanos,1]
  los.5.cercanos <- matriz.elemento.distancia[,2] %in% los.menores
  valores.los.5.cercanos <- minority.instances[los.5.cercanos]
  #points(fichero[valores.los.5.cercanos,],col="blue")
  return(valores.los.5.cercanos)
}

#Prueba de funcionamiento de la distancia
for (i in 1:5){
  x = CVperm[1,i]
  t = as.matrix(CVperm[,i])
  #Busco los Ã?ndices de la clase minoritaria
  indices <- CVperm_pos[,i]
  minority.indices <- (1:length(pos))[-indices]
```

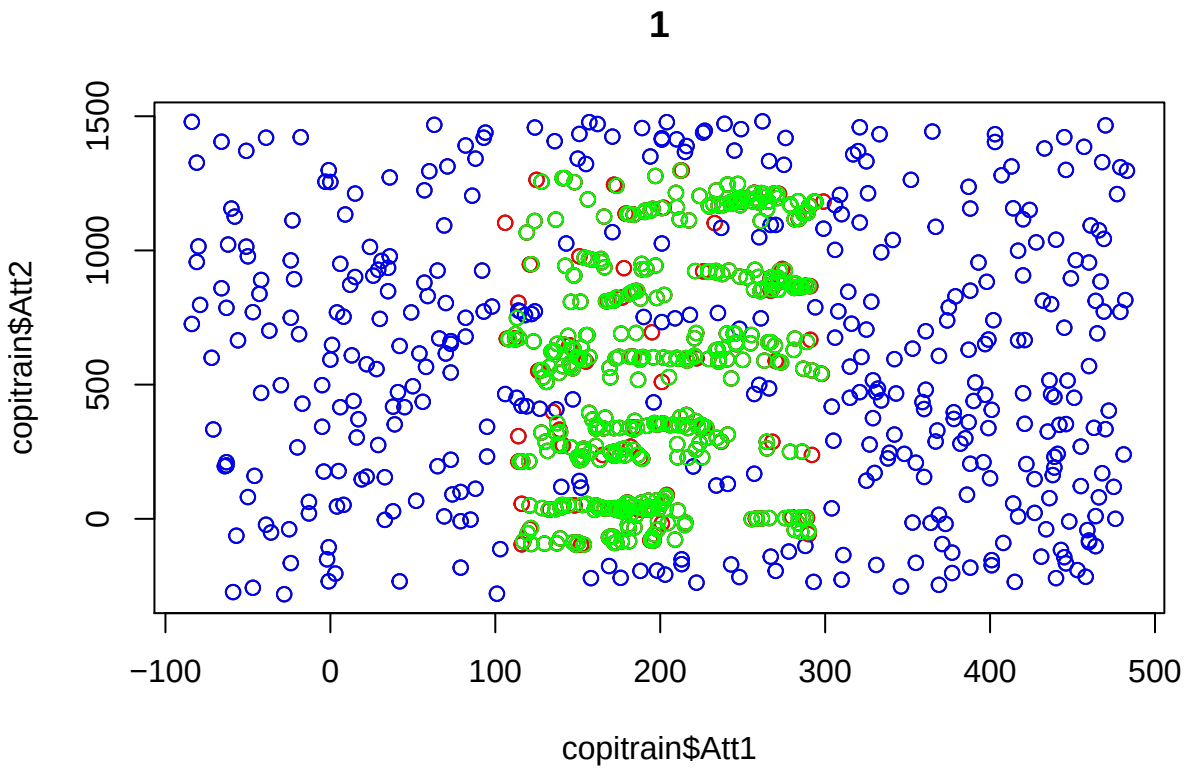
```
  cercanos <- getNeighbors(x,minority.indices,t)
  print("Elemento")
  print(x)
  print("Cercanos")
  print(cercanos)
}
```

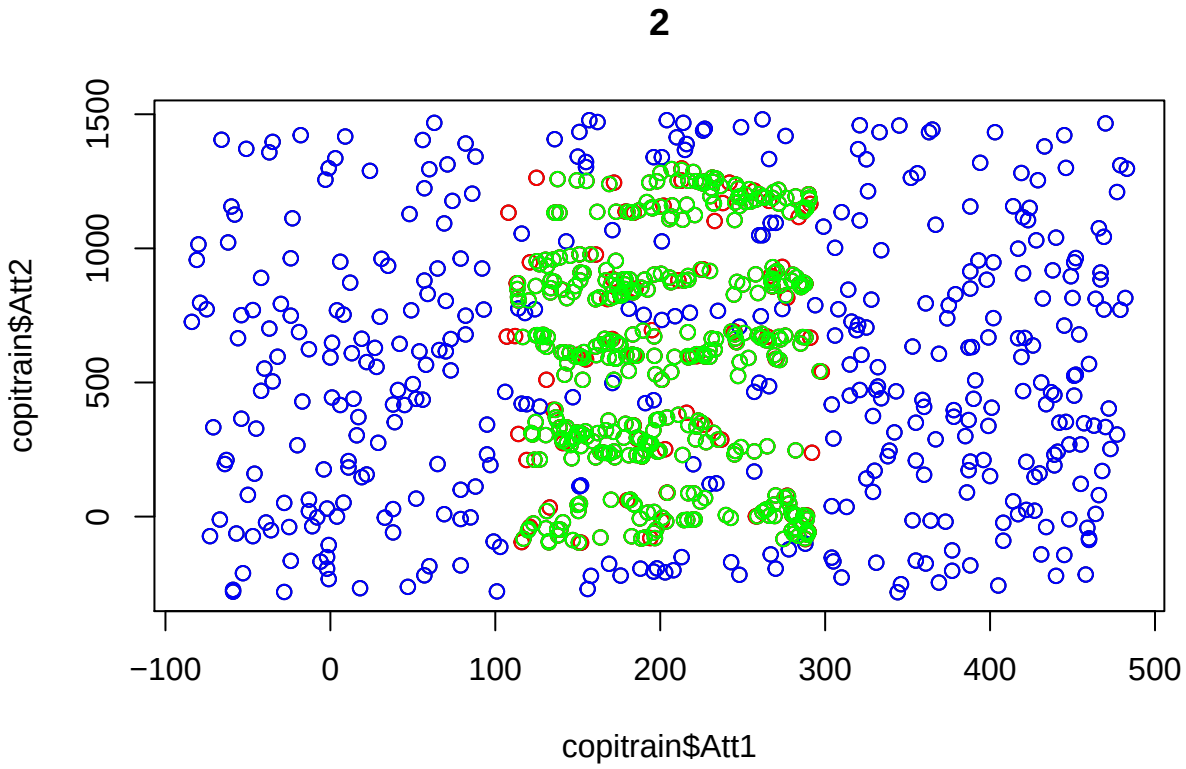
```
## [1] "Elemento"
## [1] 12
## [1] "Cercanos"
## [1]  5  8 10 16 17
## [1] "Elemento"
## [1] 62
## [1] "Cercanos"
## [1] 68 69 72 76 80
## [1] "Elemento"
## [1] 60
## [1] "Cercanos"
## [1] 41 46 48 53 58
## [1] "Elemento"
## [1] 61
## [1] "Cercanos"
## [1] 63 66 71 73 78
## [1] "Elemento"
## [1] 83
## [1] "Cercanos"
## [1] 85 88 90 96 98
```

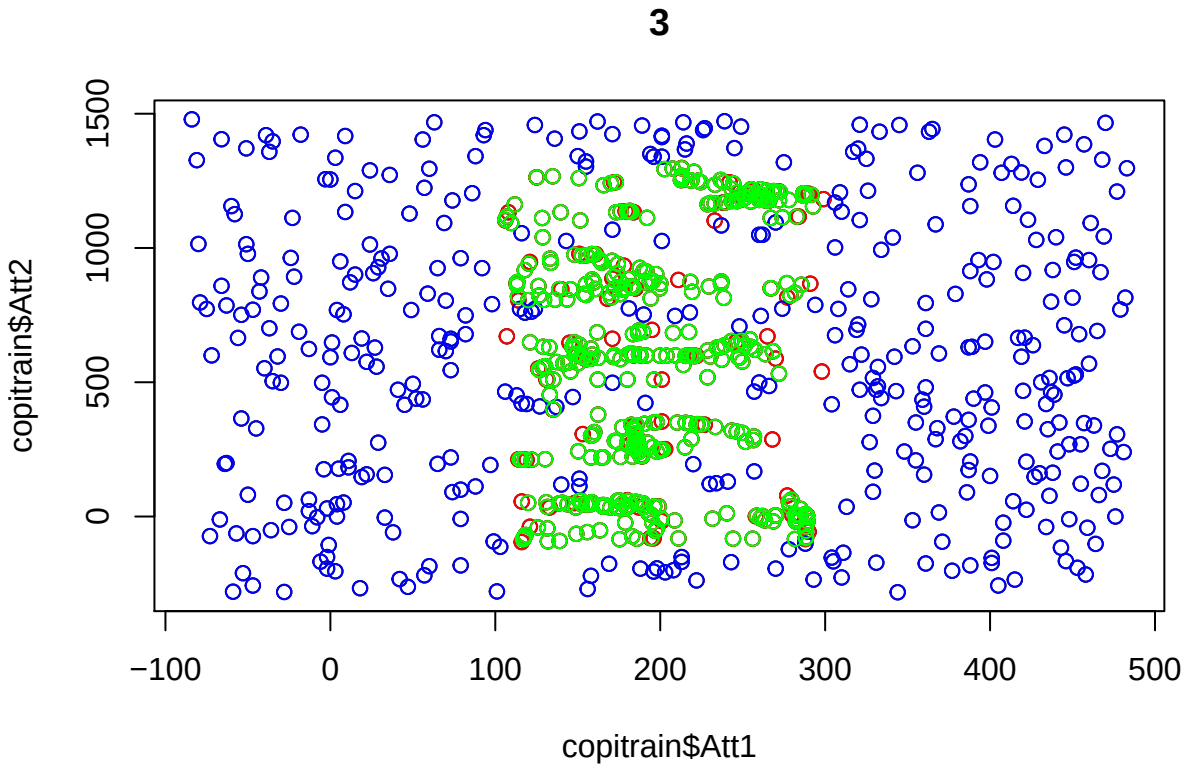
```
#FunciÃ³n SyntheticInstance
syntheticInstance <- function(elemento, cercanos){
  #Cojo un elemento de los cercanos aleatoriamente
  aleatorio.cercanos <- sample(1:5,1)
  valor.aleatorio.cercanos <- cercanos[aleatorio.cercanos]
  #Saco los valores att1 y att2 de mi elemento y del aleatorio cercano
  att1.x <- fichero[elemento,1]
  att1.y <- fichero[valor.aleatorio.cercanos,1]
  att1.z = att1.y - att1.x
  att2.x <- fichero[elemento,2]
  att2.y <- fichero[valor.aleatorio.cercanos,2]
  att2.z = att2.y - att2.x
  multiplicador = runif(1,0,1)
  #print("elemento,multiplciador,att1.x,att1.y,att1.z,att2.x,att2.y,att2.z,aleatoriodecercanos:")
  #print(elemento);print(multiplicador);print(att1.x);print(att1.y);print(att1.z);print(att2.x);print(a
  att1.z = att1.x + (multiplicador*att1.z)
  att2.z = att2.x + (multiplicador*att2.z)
  #print("Final z: ");print(att1.z);print(att2.z)
  salida = NULL
  salida$Att1 = att1.z
  salida$Att2 = att2.z
  return(list(salida,aleatorio.cercanos))
  #return(salida)
}
#Prueba de syntheticInstance
set.seed(1234)
```

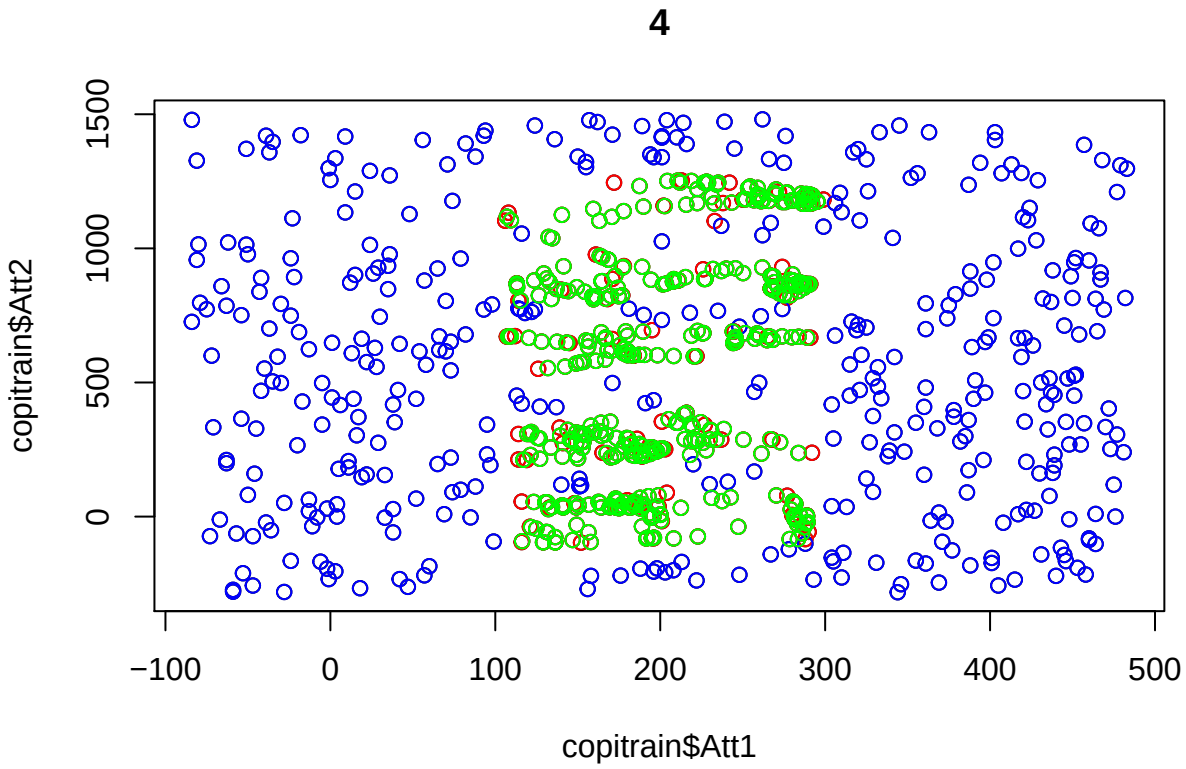
```
nueva.instancia <- syntheticInstance(x,cercanos)

knn.pred = NULL
#FunciÃ³n de SMOTE
SMOTE <- function(slot,visualizacion=FALSE,visualizar.por.pasos=FALSE){
  #Preparo los conjuntos de train y test
  train <- fichero[-CVperm[,slot],-3]
  classes.train <- fichero[-CVperm[,slot], 3]
  test  <- fichero[CVperm[,slot], -3]
  #Busco los Ã?ndices de la clase minoritaria
  indices <- CVperm_pos[,slot]
  minority.indices <- (1:length(pos))[-indices]
  #Creo el nuevo train
  copitrain <- train
  instancias.nuevas <- NULL
  total.a.crear = dim(train)[1] - length(minority.indices)
  #Para cada Ã?ndice de la clase minoritaria creare un nuevo elemento
  j=0
  for (i in 1:total.a.crear){
    j=j+1
    if(j==(length(minority.indices)))
      j=1
    #Busco los elementos cercanos
    cercanos <- getNeighbors(minority.indices[j],minority.indices,fichero)
    #Creo la nueva instancia
    salida.synthetic <- syntheticInstance(minority.indices[j],cercanos)
    instancia.nueva = salida.synthetic[[1]]
    aleatorio.cercanos = salida.synthetic[[2]]
    #AÃ±ado la nueva instancia al nuevo train
    copitrain <- rbind(copitrain, instancia.nueva)
    instancias.nuevas <- rbind(instancias.nuevas,instancia.nueva)
    #AÃ±ado la clase del nuevo elemento
    classes.train <- c(classes.train,0)
    if (visualizar.por.pasos == TRUE){
      plot(train$Att1,train$Att2)
      points(train[classes.train==0,1],train[classes.train==0,2],col="grey")
      points(train[classes.train==1,1],train[classes.train==1,2],col="white")
      points(instancia.nueva$Att1,instancia.nueva$Att2,col="green")
      points(fichero[minority.indices[j],],col="red")
      points(fichero[cercanos[aleatorio.cercanos],],col="red")
      #points(fichero[cercanos,],col="blue")
    }

  }
  #Realizo knn para la prediccIÃ³n
  predictions <-  knn(copitrain, test, classes.train, k = 3)
  knn.pred <- c(knn.pred, predictions)
  #Si estÃ¡ activa, realizarÃ© la visualizaciÃ³n de SMOTE siendo verde los nuevos elementos
  if (visualizacion == TRUE){
    plot(copitrain$Att1,copitrain$Att2,title(slot))
    points(copitrain[classes.train==0,1],copitrain[classes.train==0,2],col="red")
    points(copitrain[classes.train==1,1],copitrain[classes.train==1,2],col="blue")
    points(instancias.nuevas,col="green")
```

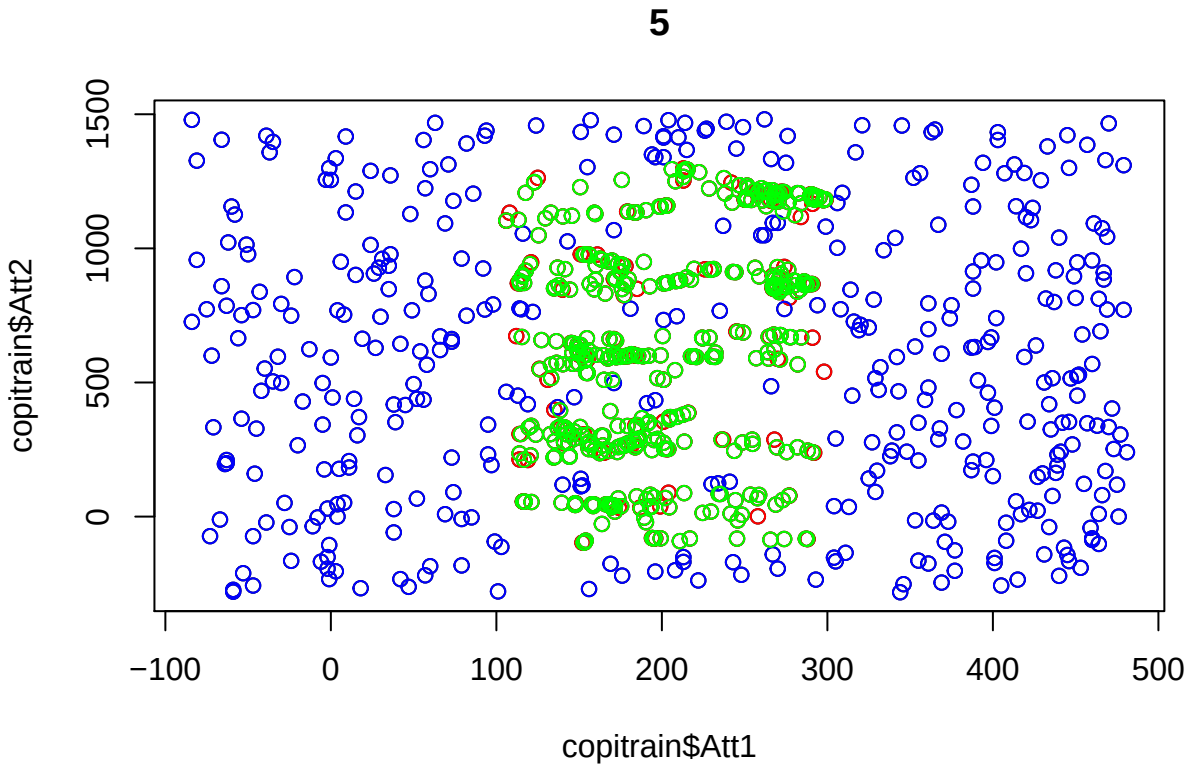
```
  }
  return(knn.pred)
}
set.seed(1234)
#Aplico SMOTE y visualizo los resultados
for (j in 1:5){
  knn.pred <- SMOTE(j,TRUE)
}
```




2
copitrain$Att2
1500
1000
500
0
−100
0
100
200
300
400
500
copitrain$Att1


3
copitrain$Att2
1500
1000
500
0
−100
0
100
200
300
400
500
copitrain$Att1


4
copitrain$Att2
1500
1000
500
0
−100
0
100
200
300
400
500
copitrain$Att1




```
tpr.SMOTE <- sum(fichero$Class[as.vector(CVperm)] == 0 & knn.pred == 1) / nClass0
tpr.SMOTE
```

```
## [1] 0.87
```

```
#Obtenemos un 0.87 en la clase positiva en la clase positiva
tnr.SMOTE <- sum(fichero$Class[as.vector(CVperm)] == 1 & knn.pred == 2) / nClass1
tnr.SMOTE
```

```
## [1] 0.93
```

```
#Obtenemos un 0.93 en la clase negativa
gmean.SMOTE <- sqrt(tpr.SMOTE * tnr.SMOTE)
gmean.SMOTE
```

```
## [1] 0.8994999
```

```
#La media es de un 0.899
```

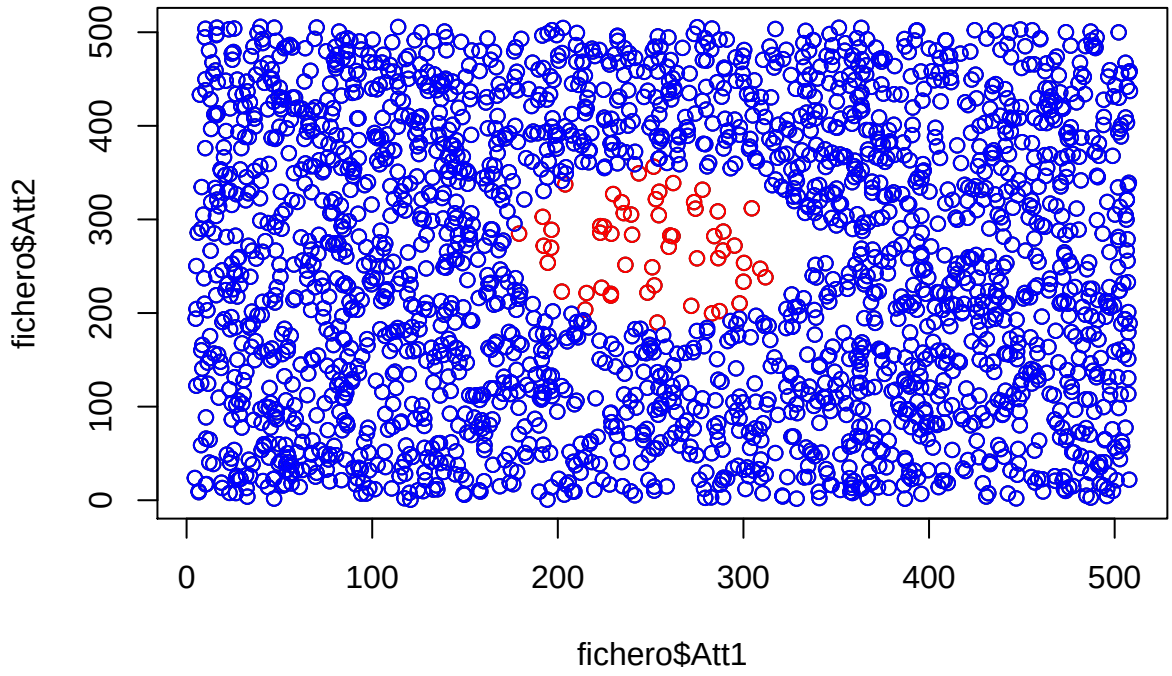
```
############################################################
######################  CIRCLE ############################
############################################################
fichero <- read.table("circle.txt", sep=",")
colnames(fichero) <- c("Att1", "Att2", "Class")
#Determinar el radio de imbalanceamiento
unique(fichero$Class)
```

```
## [1] 0 1
```

```r
nClass0 <- sum(fichero$Class == 0)
nClass1 <- sum(fichero$Class == 1)
IR <- nClass1 / nClass0
IR #Por cada ejemplo positivo hay 42,4 negativos
```

```
## [1] 42.45455
```

```r
#Visualizamos la distribuciÃ³n de los datos
plot(fichero$Att1, fichero$Att2)
points(fichero[fichero$Class==0,1],fichero[fichero$Class==0,2],col="red")
points(fichero[fichero$Class==1,1],fichero[fichero$Class==1,2],col="blue")
```




```r
#Dividimos el dataset en 5 partes para la validaciÃ³n cruzada
set.seed(1234)
pos <- (1:dim(fichero)[1])[fichero$Class==0]
neg <- (1:dim(fichero)[1])[fichero$Class==1]
#Hacemos las divisiones en los 5 conjuntos de cada clase
CVperm_pos <- matrix(sample(pos,length(pos)), ncol=5, byrow=T)
CVperm_neg <- matrix(sample(neg,length(neg)), ncol=5, byrow=T)
#Unimos las dos clases
CVperm <- rbind(CVperm_pos, CVperm_neg)

#Llevamos a cabo el 3NN
library(class)
knn.pred = NULL
set.seed(1234)
```

```
for( i in 1:5){
  predictions <- knn(fichero[-CVperm[,i], -3], fichero[CVperm[,i], -3], fichero[-CVperm[,i], 3], k = 3)
  knn.pred <- c(knn.pred, predictions)
}
acc <- sum((fichero$Class[as.vector(CVperm)] == 0 & knn.pred == 1)
           | (fichero$Class[as.vector(CVperm)] == 1 & knn.pred == 2)) / (nClass0 + nClass1)
acc
```

```
## [1] 0.9941423
```

```
#parece que se obitene un buen rendimiento ya que tenemos un accuracy del 0.994

tnr <- sum(fichero$Class[as.vector(CVperm)] == 1 & knn.pred == 2) / nClass1
tnr
```

```
## [1] 0.9991435
```

```
#es bueno para la clase negativa al obtener un 0.999
tpr <- sum(fichero$Class[as.vector(CVperm)] == 0 & knn.pred == 1) / nClass0
tpr
```

```
## [1] 0.7818182
```

```
#no es tan bueno para la case positiva ya que obtiene un 0.781
gmean <- sqrt(tpr * tnr) # no es tan bueno como decia el accuracy
gmean
```

```
## [1] 0.8838261
```

```
#Obtenemos de media un 0.883

# 1. Vamos a aplicar ROS, de forma que replicaremos los ejemplos positivos
knn.pred = NULL
knn.pred <- aplicarROS()

tpr.ROS <- sum(fichero$Class[as.vector(CVperm)] == 0 & knn.pred == 1) / nClass0
tpr.ROS
```

```
## [1] 0.8909091
```

```
#Hemos pasado de un 0.78 a un 0.89 mejorando mucho en la clase positiva
tnr.ROS <- sum(fichero$Class[as.vector(CVperm)] == 1 & knn.pred == 2) / nClass1
tnr.ROS
```

```
## [1] 0.9961456
```

```
#Hemos pasado de un 0.999 a un 0.996 en la clase negativa
gmean.ROS <- sqrt(tpr.ROS * tnr.ROS)
gmean.ROS
```

```
## [1] 0.942059
```

```
#La media ha pasado de 0.883 a 0.942, mejorando

# 2. Vamos a aplicar RUS, de forma que quito elementos de la clase mayoritaria
set.seed(1234)
knn.pred = NULL
knn.pred = aplicarRUS()
```

```
tpr.RUS <- sum(fichero$Class[as.vector(CVperm)] == 0 & knn.pred == 1) / nClass0
tpr.RUS
```

```
## [1] 0.9818182
```

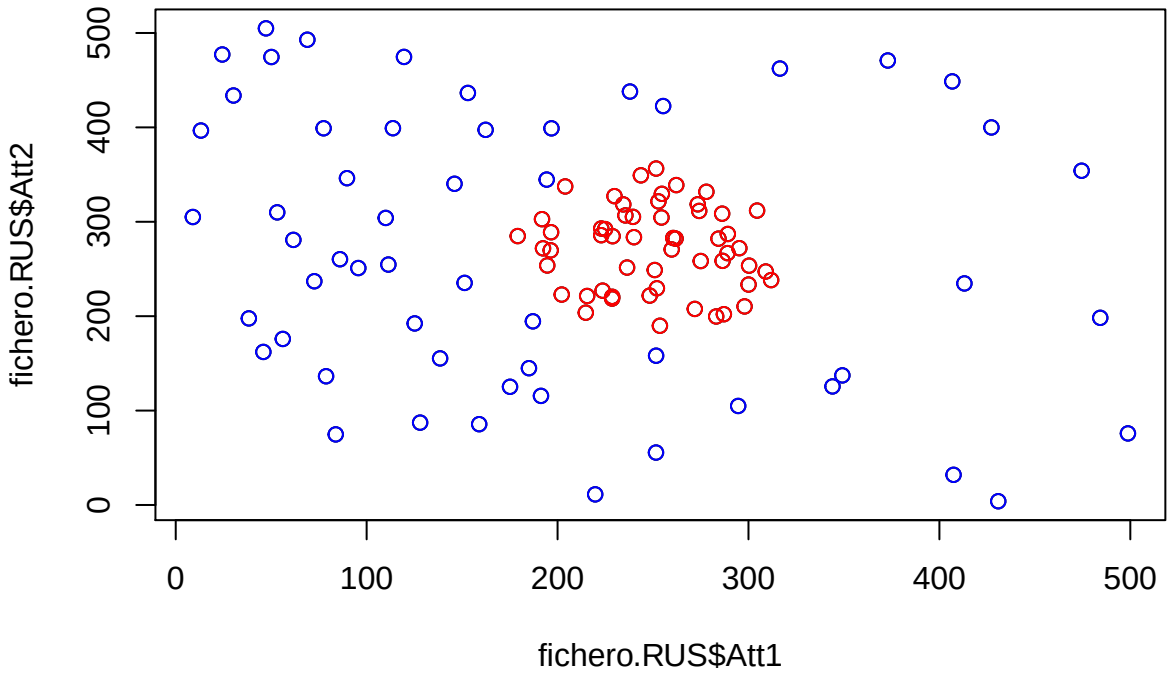
```
#Hemos pasado de 0.89 a 0.981
tnr.RUS <- sum(fichero$Class[as.vector(CVperm)] == 1 & knn.pred == 2) / nClass1
tnr.RUS
```

```
## [1] 0.909636
```

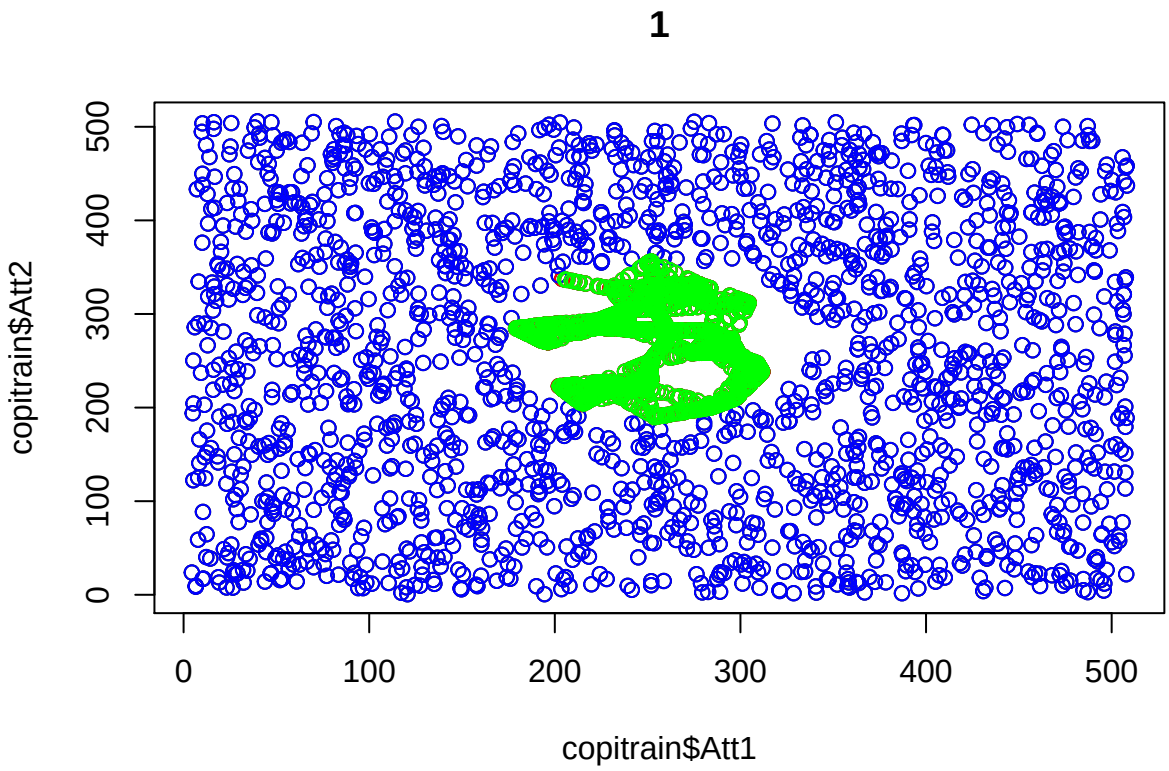
```
#Hemos pasado de 0.99 a 0.90
gmean.RUS <- sqrt(tpr.RUS * tnr.RUS)
gmean.RUS
```

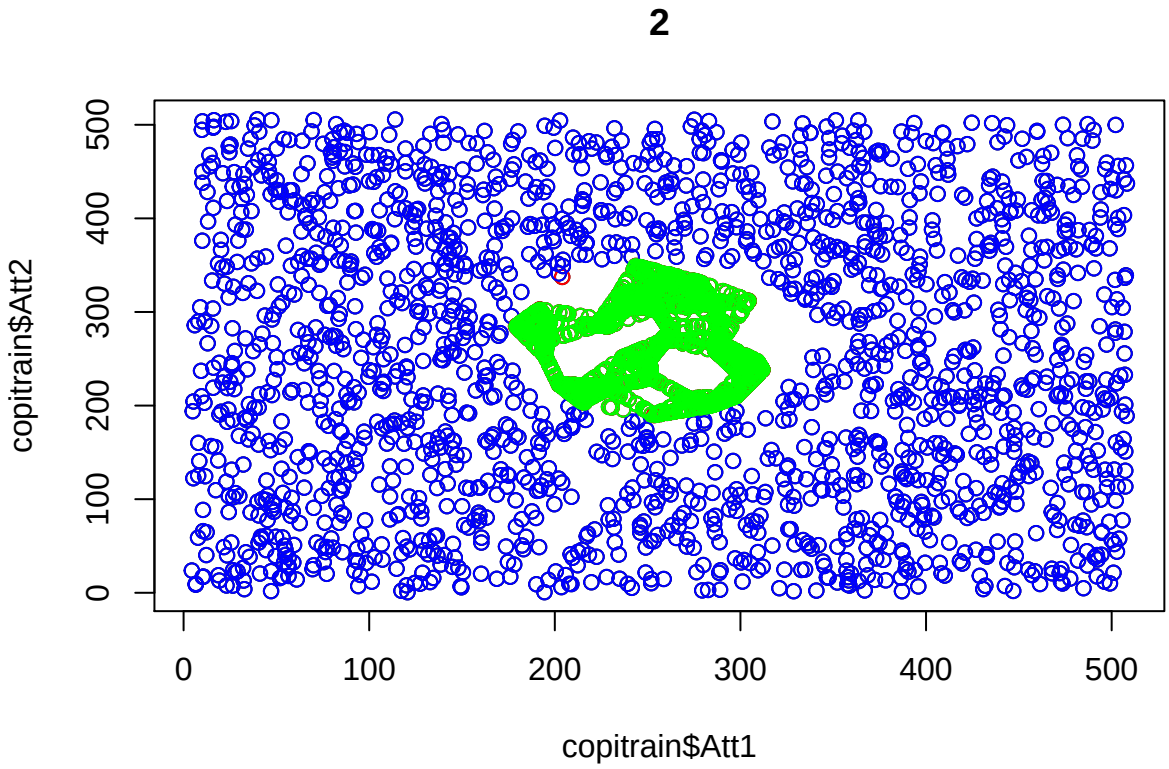
```
## [1] 0.9450382
```

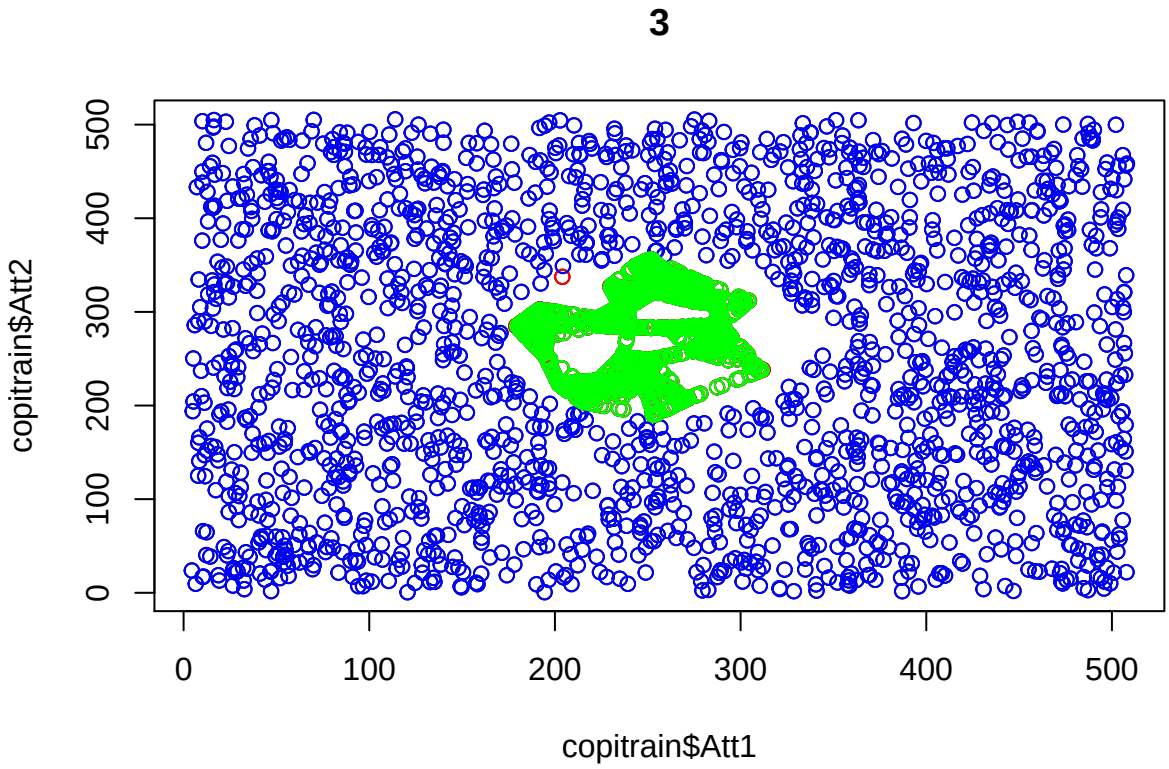
```
#En este caso, a diferencia de subclus, hemos aumentado un poco la media de 0.942 a 0.945

#Visualizamos como quedarÃ?a la tÃ©cnica RUS en el dataset
fichero.RUS <- fichero
majority.indices <- (1:dim(fichero.RUS)[1])[fichero.RUS$Class == 1]
to.remove <- 2 * length(majority.indices) - dim(fichero.RUS)[1]
remove <- sample(majority.indices, to.remove, replace = F)
fichero.RUS <- fichero.RUS[-remove,]

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

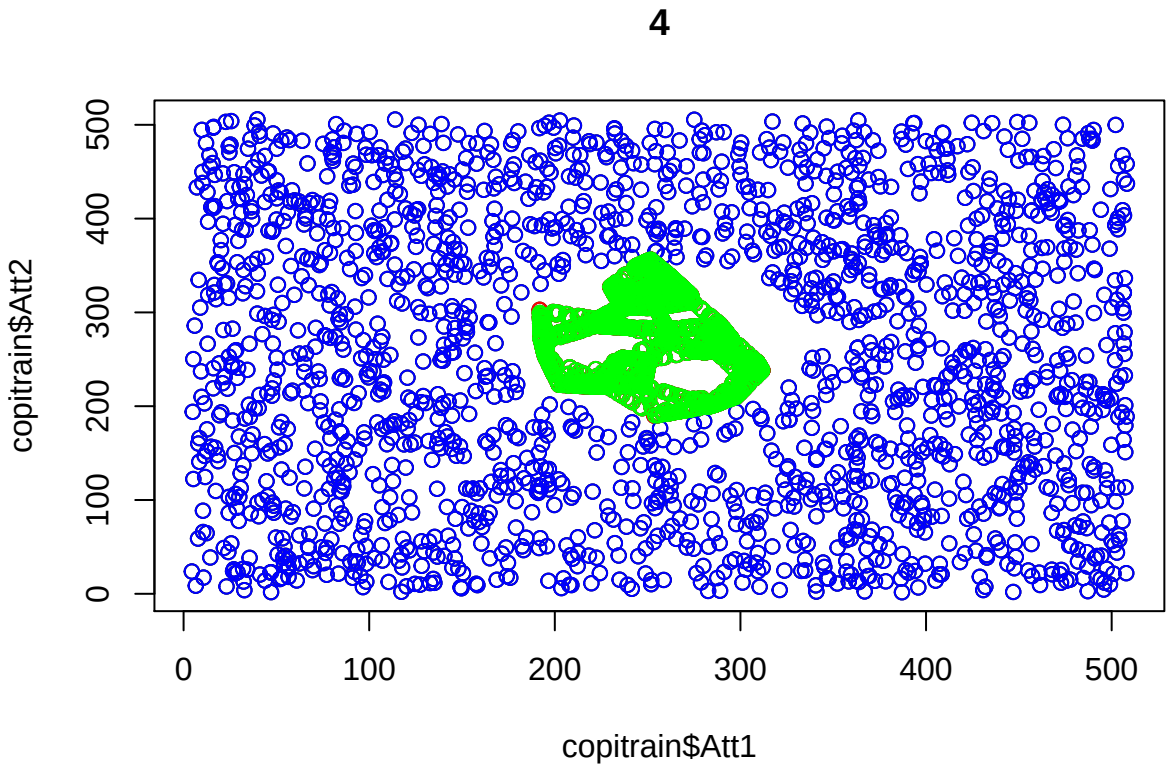


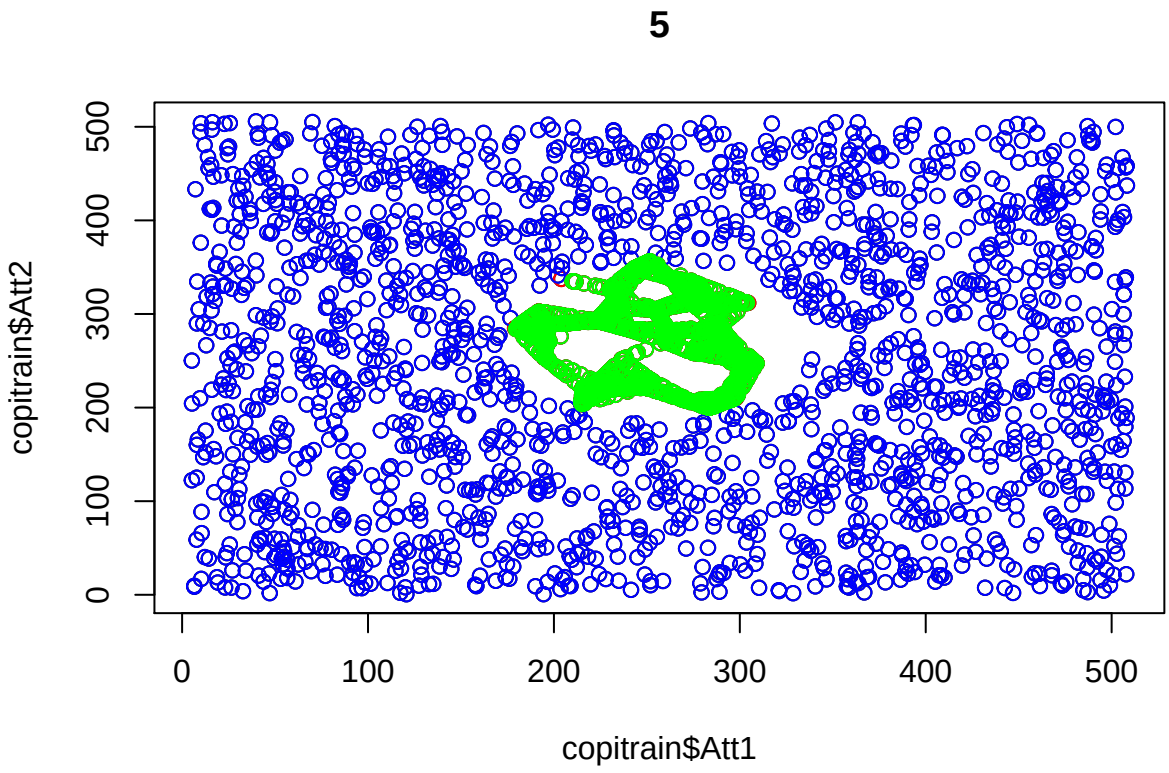

```
#SMOTE
set.seed(1234)
knn.pred = NULL
#Aplico SMOTE y visualizo los resultados
for (j in 1:5){
  knn.pred <- SMOTE(j,TRUE)
}
```


1
copitrain$Att2
0
100
200
300
400
500
copitrain$Att1
0
100
200
300
400
500


2
copitrain$Att2
0
100
200
300
400
500
copitrain$Att1


3
copitrain$Att2
0
100
200
300
400
500
copitrain$Att1
0
100
200
300
400
500


4
copitrain$Att2
0
100
200
300
400
500
copitrain$Att1
0
100
200
300
400
500




```
tpr.SMOTE <- sum(fichero$Class[as.vector(CVperm)] == 0 & knn.pred == 1) / nClass0
tpr.SMOTE
```

```
## [1] 0.9454545
```

```
#Obtenemos un 0.945 en la clase positiva en la clase positiva
tnr.SMOTE <- sum(fichero$Class[as.vector(CVperm)] == 1 & knn.pred == 2) / nClass1
tnr.SMOTE
```

```
## [1] 0.9982869
```

```
#Obtenemos un 0.998 en la clase negativa
gmean.SMOTE <- sqrt(tpr.SMOTE * tnr.SMOTE)
gmean.SMOTE
```

```
## [1] 0.9715117
```

```
#La media es de un 0.971
```

```
################################################
############  LibrerÃ?a Unbalanced  ############
################################################
library(unbalanced)
```

```
## Loading required package: mlr
```

```
## Loading required package: ParamHelpers
```

```
##
## Attaching package: 'mlr'
```

```
## The following objects are masked _by_ '.GlobalEnv':
##
##     acc, gmean, tnr, tpr
```

```
## Loading required package: foreach
```

```
## Loading required package: doParallel
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

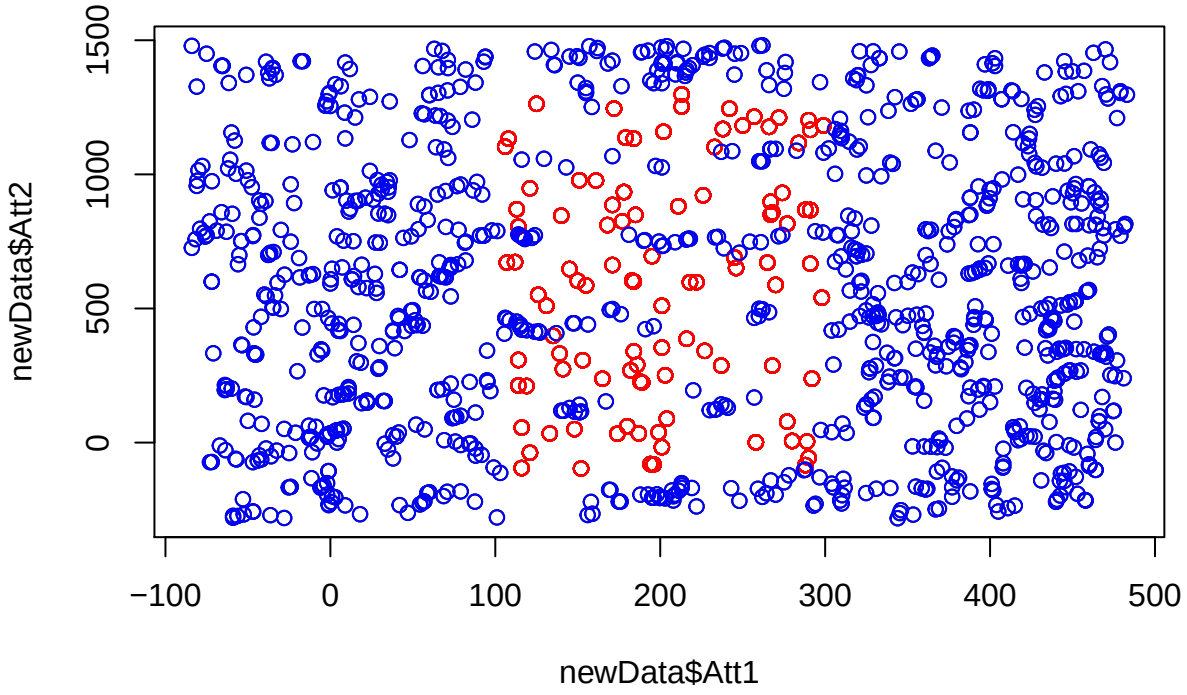
```
fichero <- read.table("subclus.txt", sep=",")
colnames(fichero) <- c("Att1", "Att2", "Class")
#Determinar el radio de imbalanceamiento
nClass0 <- sum(fichero$Class == 0)
nClass1 <- sum(fichero$Class == 1)
IR <- nClass1 / nClass0
IR #Por cada ejemplo positivo hay 5 negativos
```

```
## [1] 5
```

```
#Dividimos el dataset en 5 partes para la validaciÃ³n cruzada
set.seed(1234)
pos <- (1:dim(fichero)[1])[fichero$Class==0]
neg <- (1:dim(fichero)[1])[fichero$Class==1]
#Hacemos las divisiones en los 5 conjuntos de cada clase
CVperm_pos <- matrix(sample(pos,length(pos)), ncol=5, byrow=T)
CVperm_neg <- matrix(sample(neg,length(neg)), ncol=5, byrow=T)
#Unimos las dos clases
CVperm <- rbind(CVperm_pos, CVperm_neg)

input <- fichero[,-3]
output <- as.factor(fichero[,3])
data<-ubSMOTE(X=input, Y= output, perc.over = 100, perc.under = 200)
newData<-cbind(data$X, data$Y)
nClass0nuevos <- sum(newData$`data$Y` == 0)
nClass1nuevos <- sum(newData$`data$Y` == 1)
IR <- nClass1nuevos / nClass0nuevos
IR #Por cada ejemplo positivo hay 1 negativos
```

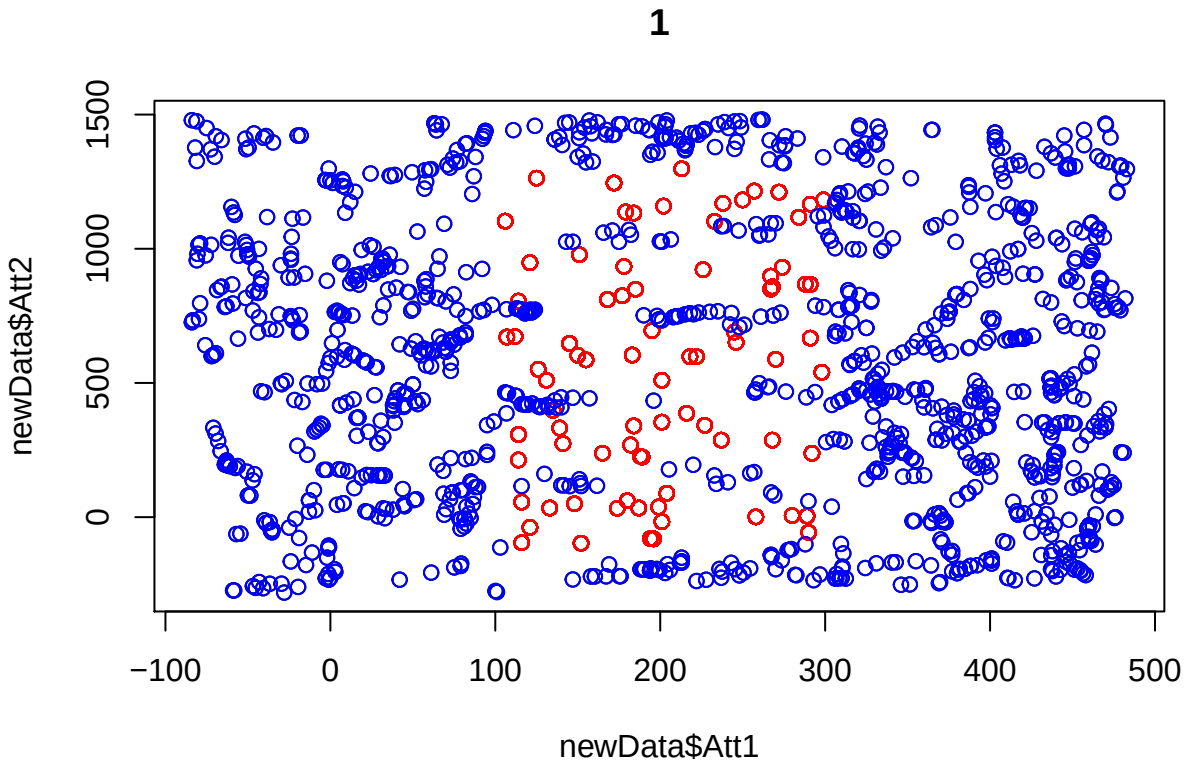
```
## [1] 1
```

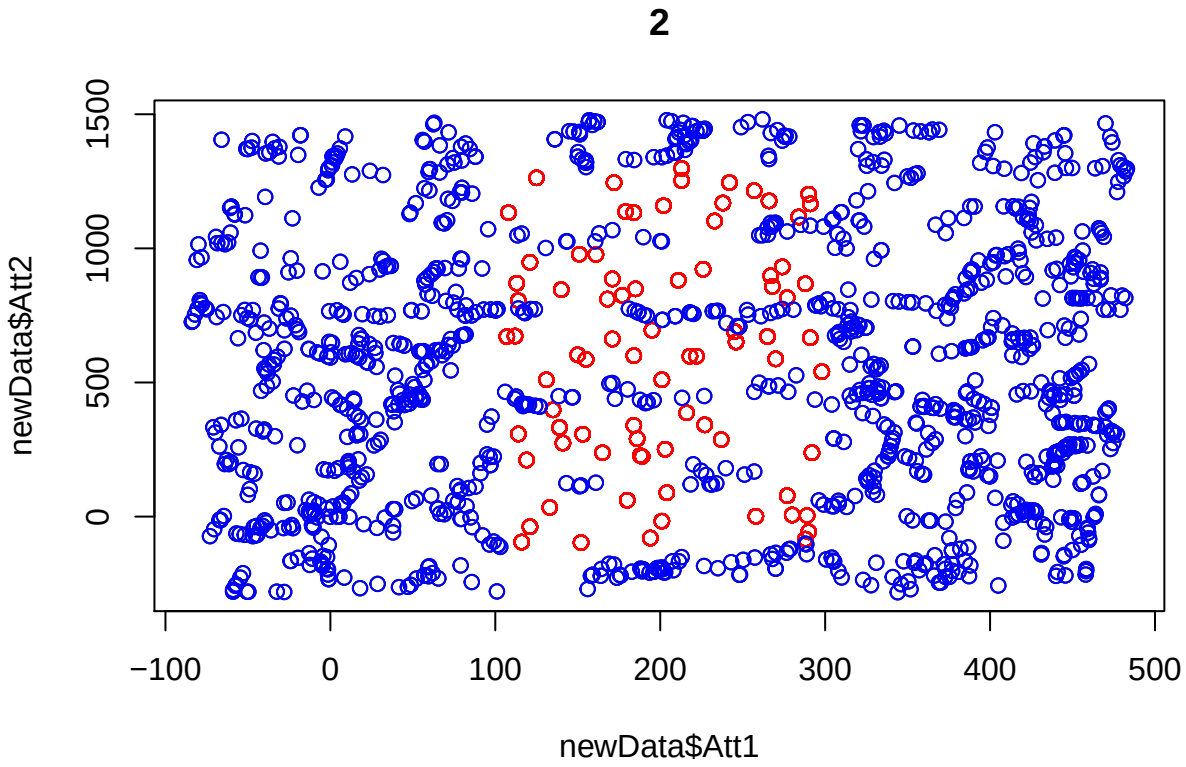
```
#Visualizamos la distribuciÃ³n de los datos
plot(newData$Att1, newData$Att2)
points(newData[newData$`data$Y`==0,1],newData[newData$`data$Y`==0,2],col="red")
points(newData[newData$`data$Y`==1,1],newData[newData$`data$Y`==1,2],col="blue")
```

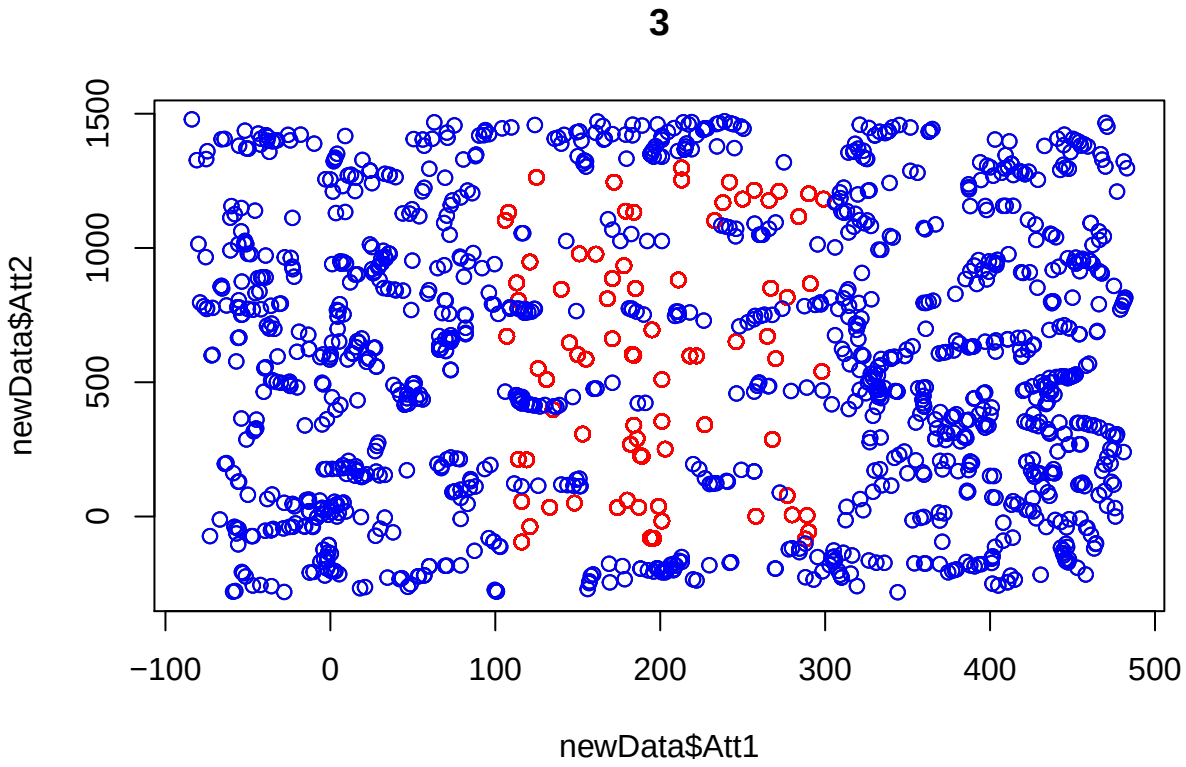


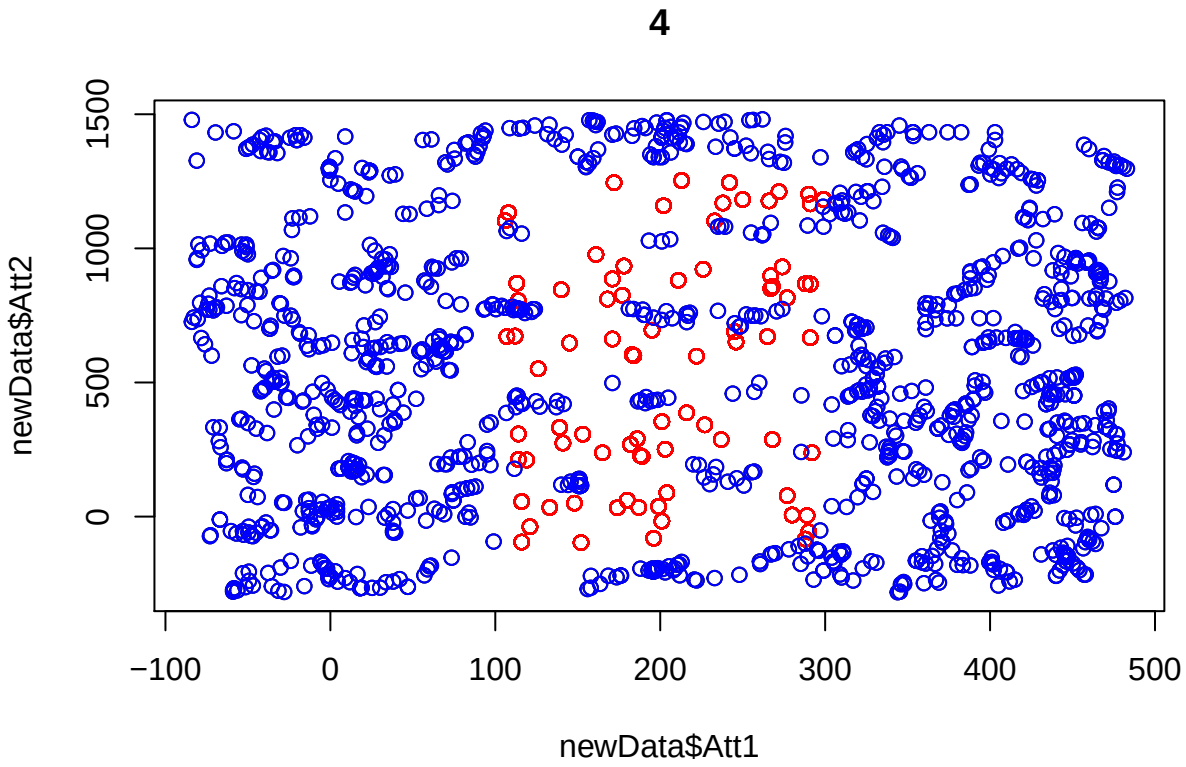

```
knn.pred <- NULL
tomelinks=FALSE; enn=FALSE
SMOTEUnbalanced = function(tomelinks=FALSE,enn=FALSE){
  for (i in 1:5){
    input <- fichero[-CVperm[,i],-3]
    output <- as.factor(fichero[-CVperm[,i], 3])
    data<-ubSMOTE(X=input, Y= output, perc.over = 200, perc.under = 200)
    newData <- cbind(data$X, data$Y)
    classes.train <- newData$`data$Y`
    test <- fichero[CVperm[,i],-3]
    copitrain <- newData[,-3]
    if (tomelinks==TRUE){
      #Tomelinks
      output <- newData$`data$Y`
      input <- newData[,-3]
      data<-ubTomek(X=input, Y= output)
      newData <- cbind(data$X, data$Y)
      classes.train <- newData$`data$Y`
      copitrain <- newData[,-3]
    } else if (enn==TRUE){
      data<-ubENN(X=input, Y= output)
      newData<-cbind(data$X, data$Y)
      classes.train <- newData$`data$Y`
      copitrain <- newData[,-3]
    }
    plot(newData$Att1, newData$Att2,title(i))
```

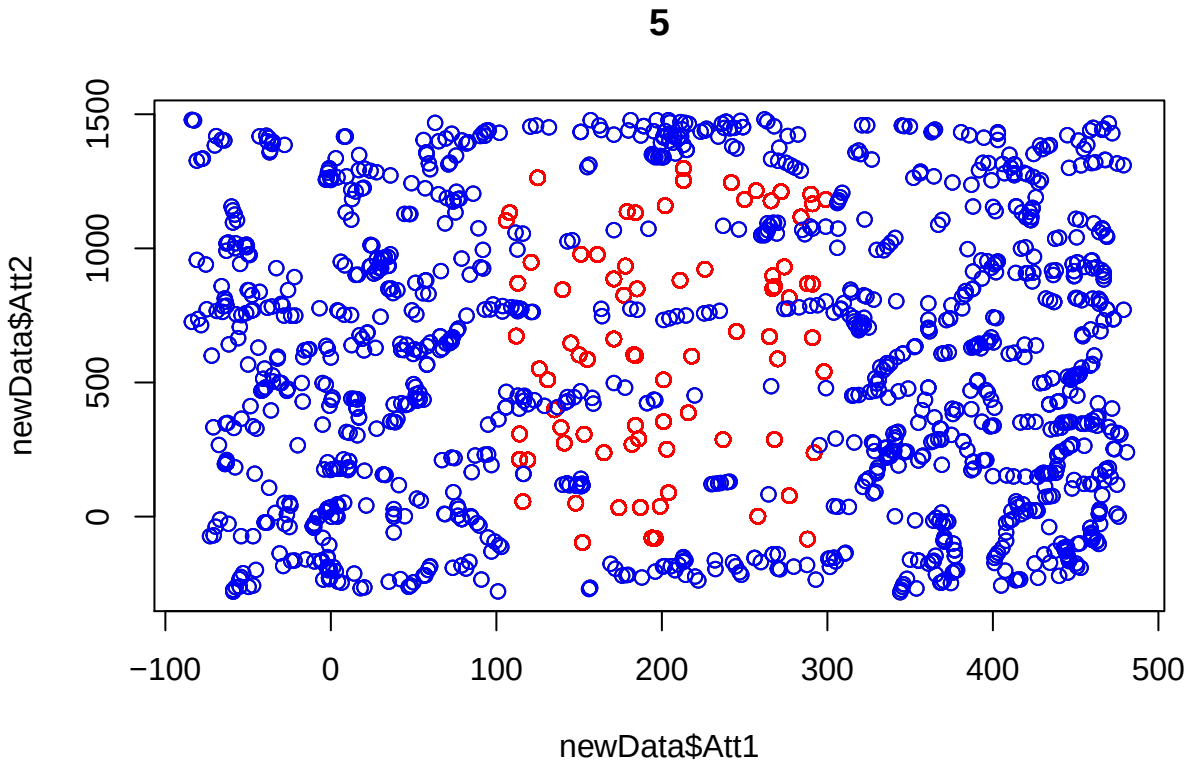
```
    points(newData[newData$`data$Y`==0,1],newData[newData$`data$Y`==0,2],col="red")
    points(newData[newData$`data$Y`==1,1],newData[newData$`data$Y`==1,2],col="blue")

    predictions <-  knn(copitrain, test, classes.train, k = 3)
    knn.pred <- c(knn.pred, predictions)
  }
  return(knn.pred)
}
knn.pred=NULL
set.seed(1234)
knn.pred <- SMOTEUnbalanced()
```




2
newData$Att2
1500
1000
500
0
−100
0
100
200
300
400
500
newData$Att1


3
newData$Att2
1500
1000
500
0
−100
0
100
200
300
400
500
newData$Att1


4
newData$Att2
1500
1000
500
0
−100
0
100
200
300
400
500
newData$Att1




```
tpr.SMOTEun <- sum(fichero$Class[as.vector(CVperm)] == 0 & knn.pred == 1) / nClass0
tpr.SMOTEun
```

```
## [1] 0.88
```

```
#Obtenemos un 0.88 en la clase positiva en la clase positiva
tnr.SMOTEun <- sum(fichero$Class[as.vector(CVperm)] == 1 & knn.pred == 2) / nClass1
tnr.SMOTEun
```

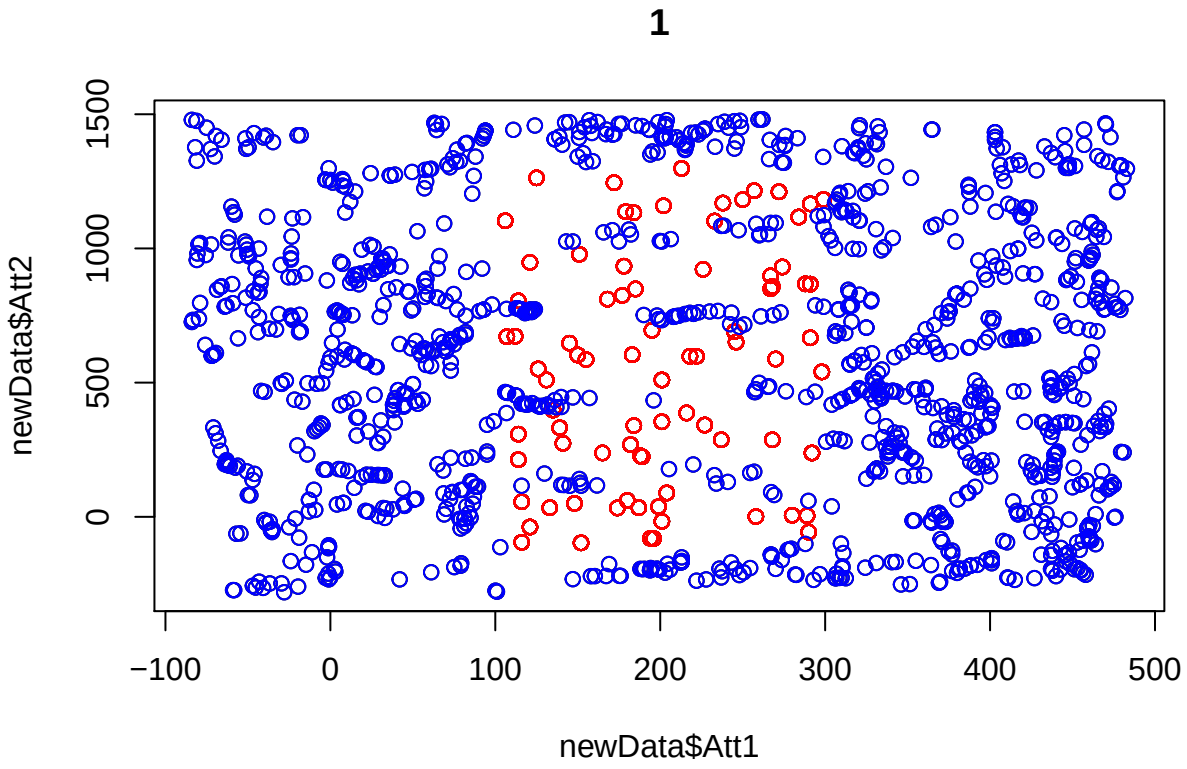
```
## [1] 0.948
```

```
#Obtenemos un 0.948 en la clase negativa
gmean.SMOTEun <- sqrt(tpr.SMOTEun * tnr.SMOTEun)
gmean.SMOTEun
```

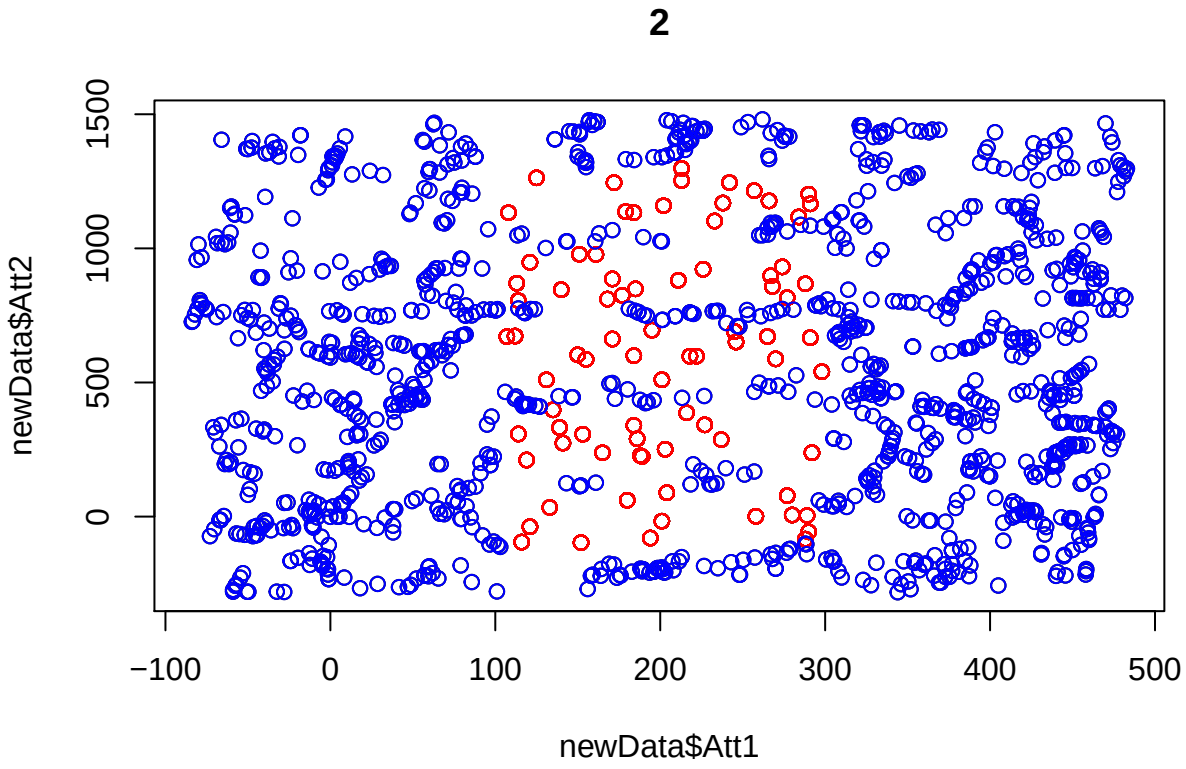
```
## [1] 0.9133674
```

```
#La media es de un 0.913

knn.pred=NULL
set.seed(1234)
knn.pred <- SMOTEUnbalanced(TRUE)
```

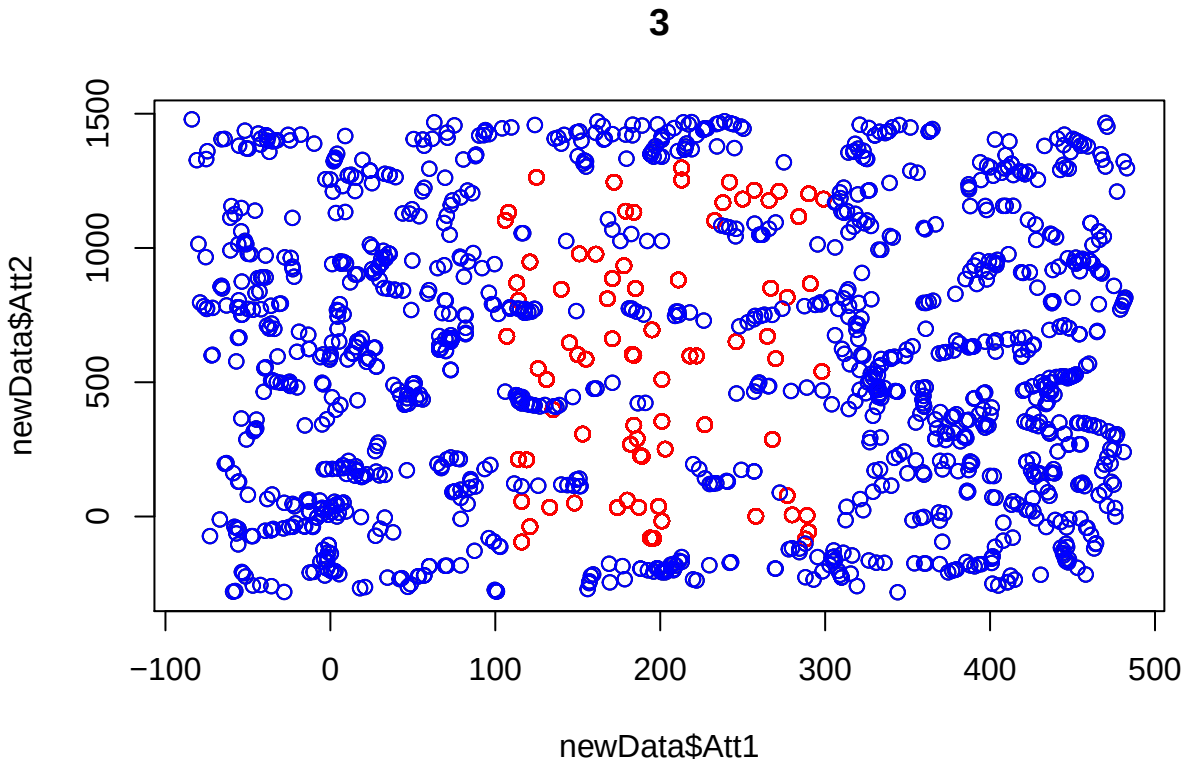
```
## Instances removed 3 : 0.19 % of 0 class ; 0.11 % of training ; Time needed 0.02
```



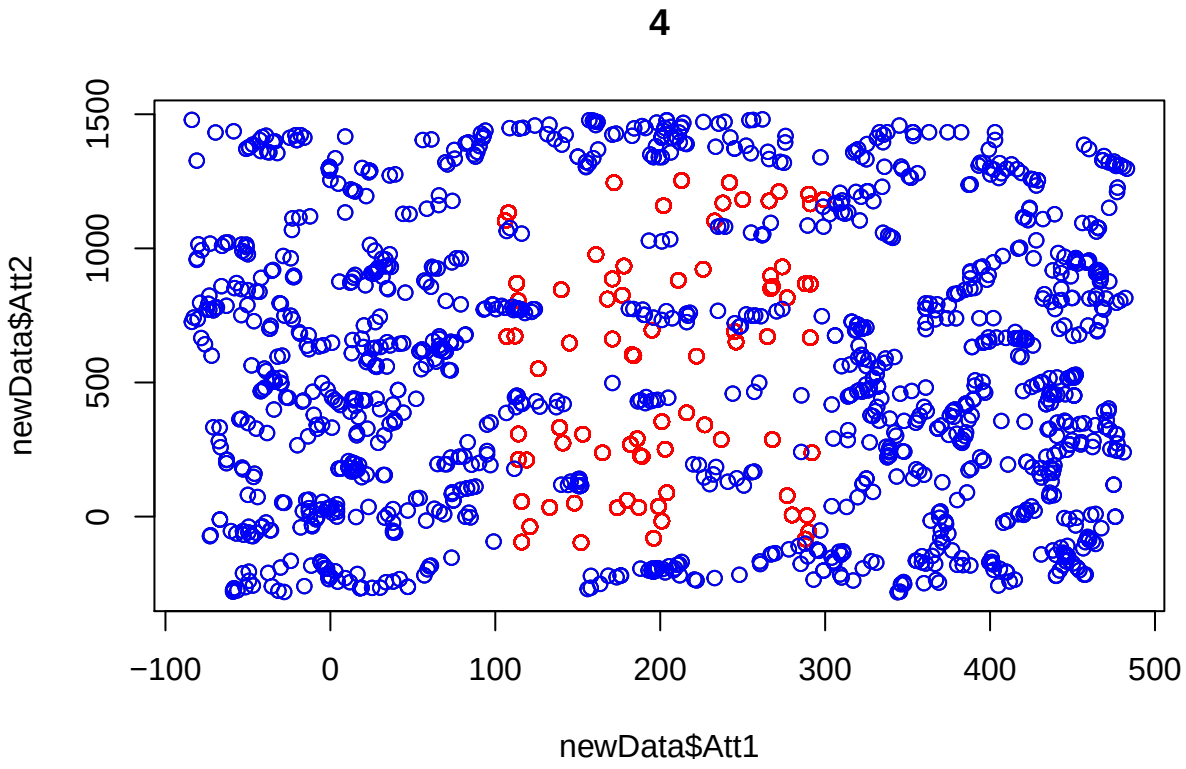

```
## Instances removed 3 : 0.19 % of 0 class ; 0.11 % of training ; Time needed 0
```



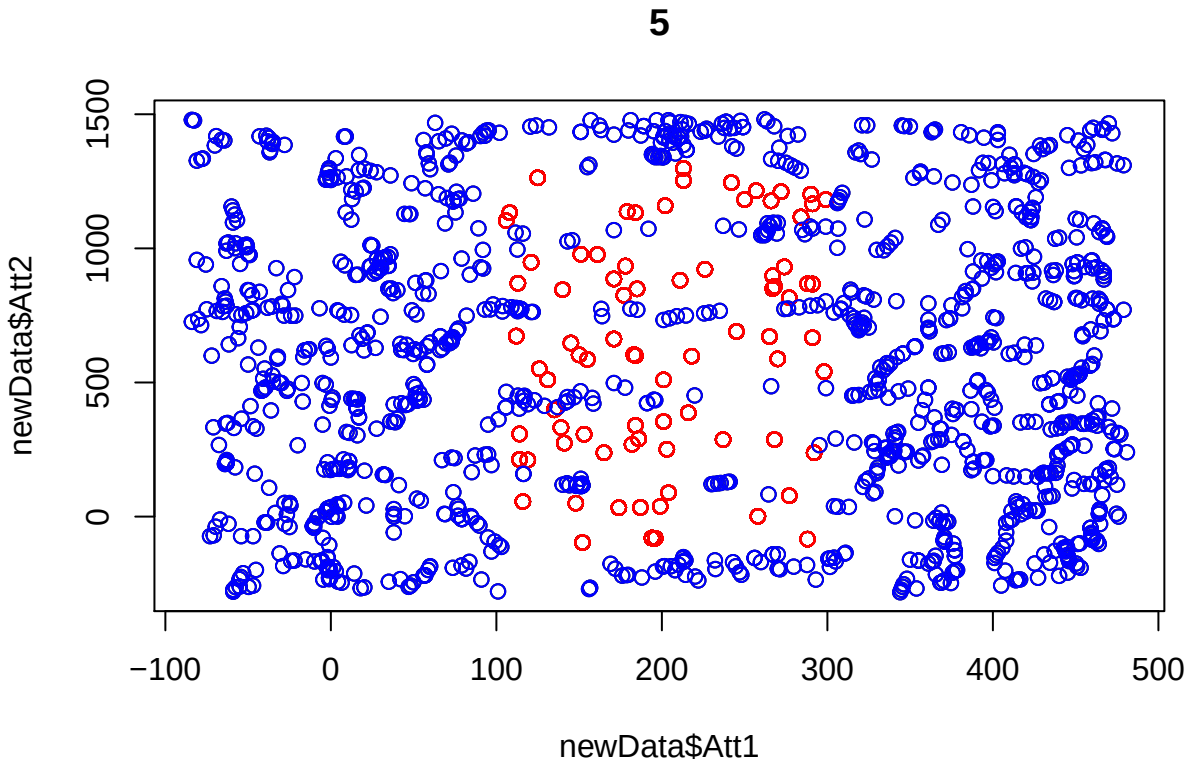

```
## Instances removed 7 : 0.44 % of 0 class ; 0.25 % of training ; Time needed 0
```




```
## Instances removed 3 : 0.19 % of 0 class ; 0.11 % of training ; Time needed 0
```




```
## Instances removed 2 : 0.12 % of 0 class ; 0.07 % of training ; Time needed 0
```




```
tpr.SMOTEun <- sum(fichero$Class[as.vector(CVperm)] == 0 & knn.pred == 1) / nClass0
tpr.SMOTEun
```

```
## [1] 0.88
```

```
#Obtenemos un 0.88 en la clase positiva en la clase positiva
tnr.SMOTEun <- sum(fichero$Class[as.vector(CVperm)] == 1 & knn.pred == 2) / nClass1
tnr.SMOTEun
```

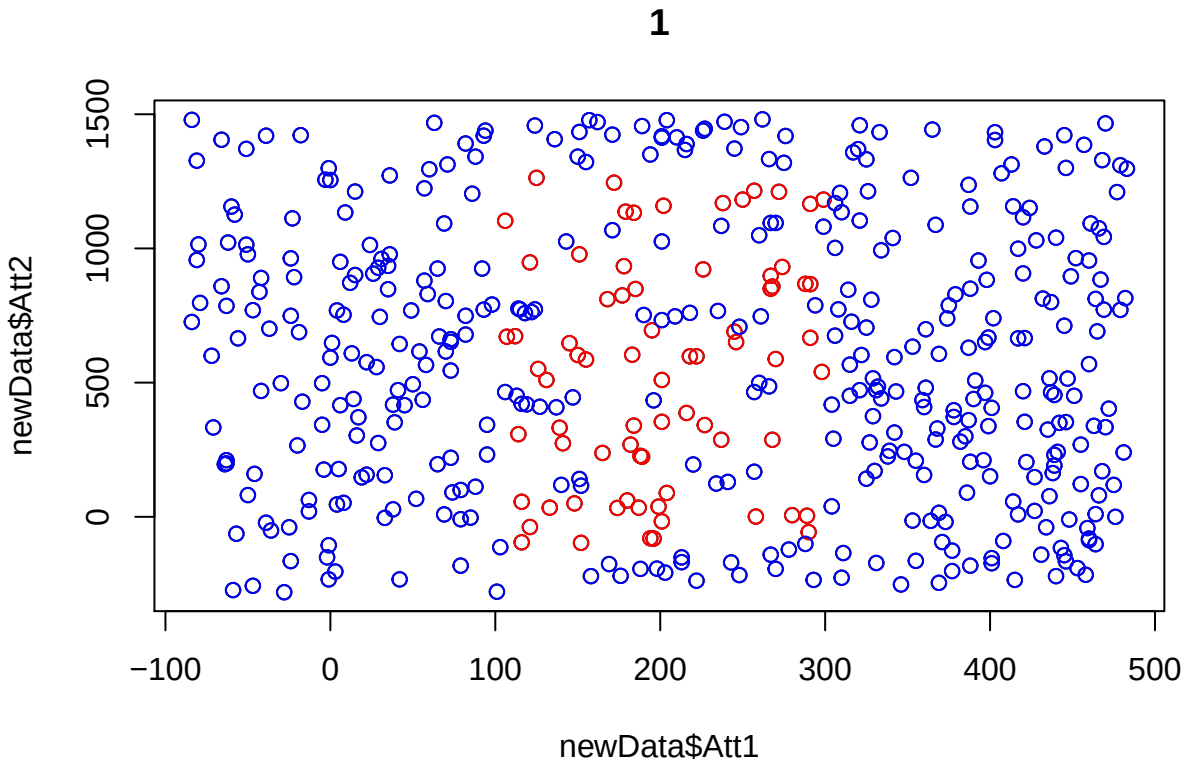
```
## [1] 0.948
```

```
#Obtenemos un 0.948 en la clase negativa
gmean.SMOTEun <- sqrt(tpr.SMOTEun * tnr.SMOTEun)
gmean.SMOTEun
```

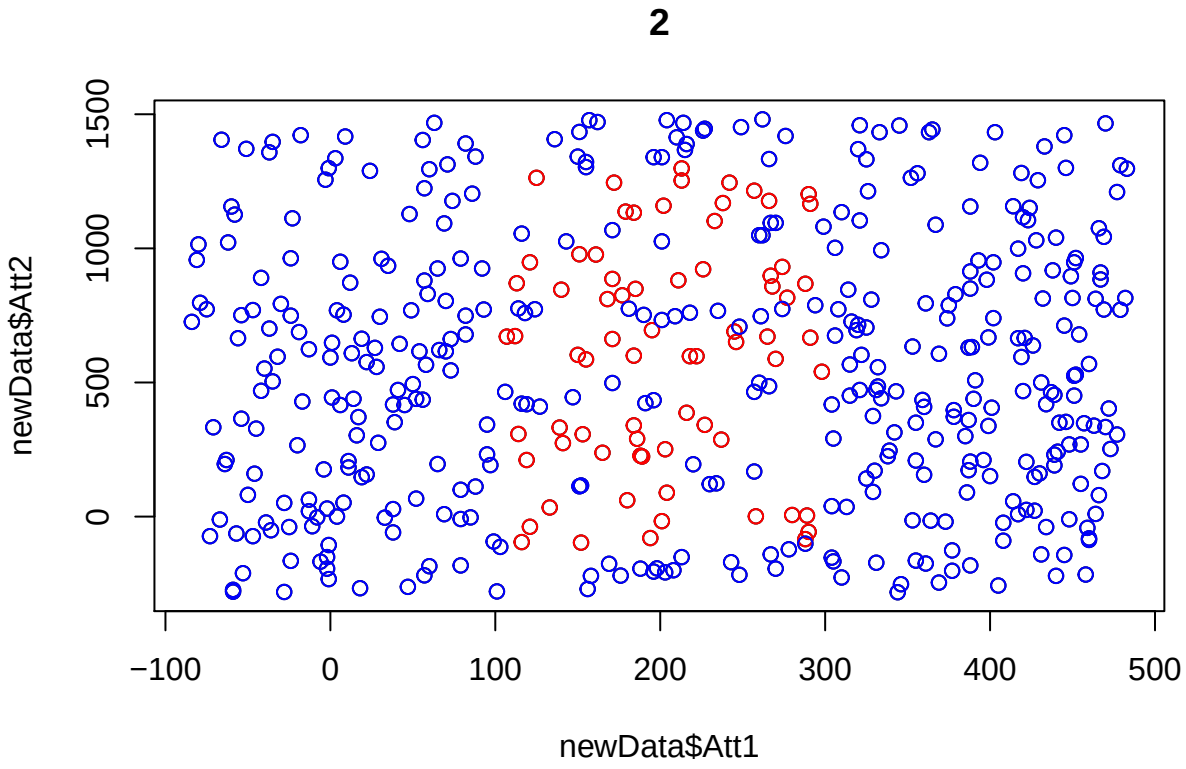
```
## [1] 0.9133674
```

```
#La media es de un 0.913

knn.pred=NULL
set.seed(1234)
knn.pred <- SMOTEUnbalanced(FALSE,TRUE)
```

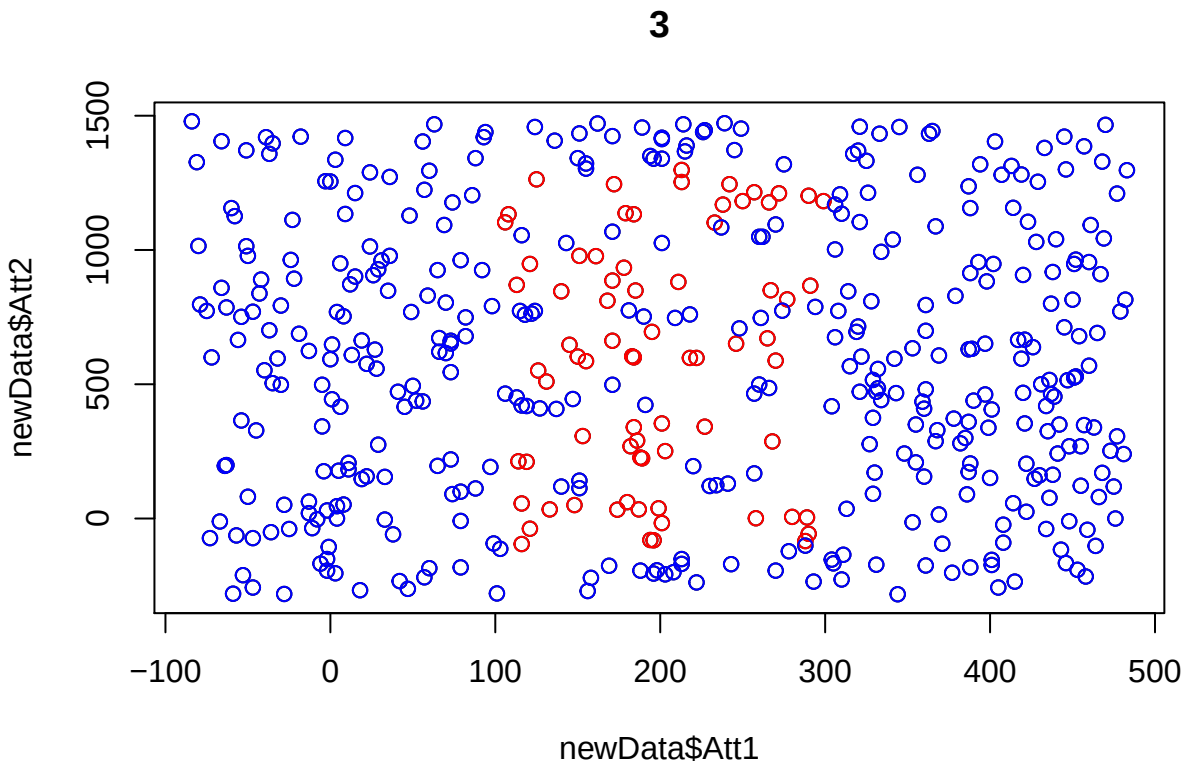
```
## Number of instances removed from majority class with ENN: 7   Time needed: 0.02
```



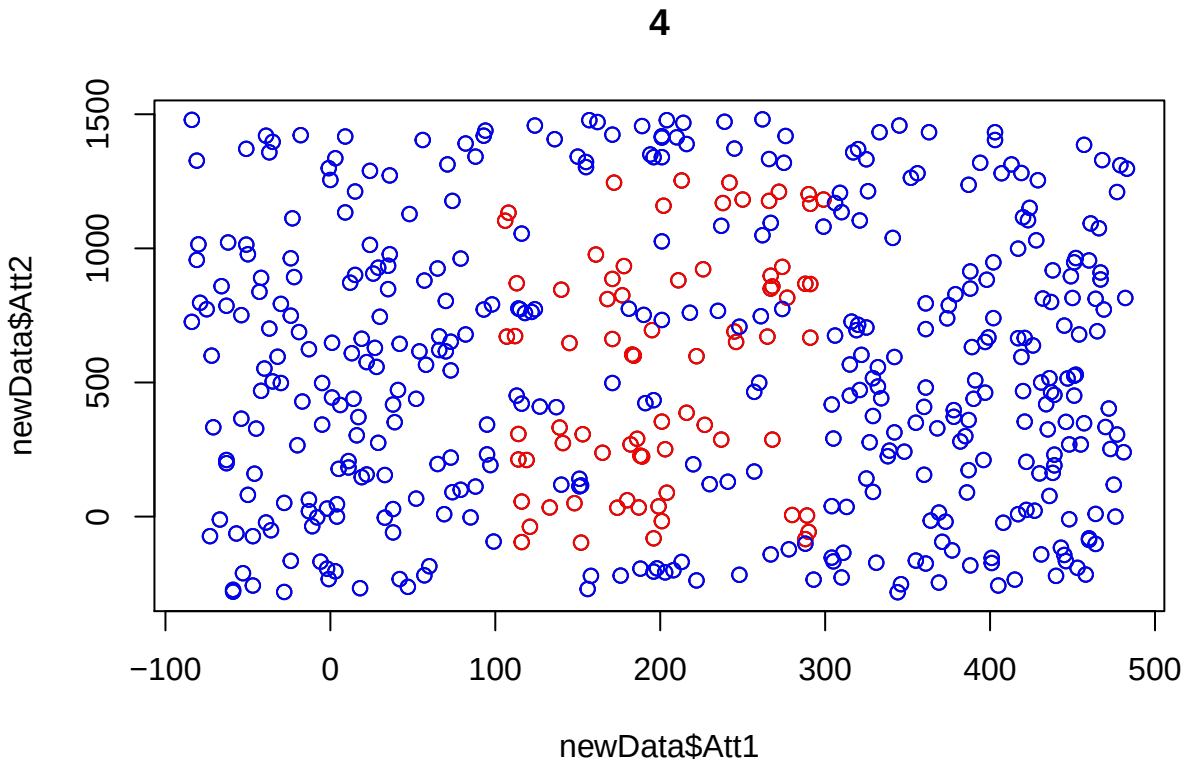

```
## Number of instances removed from majority class with ENN: 8   Time needed: 0
```



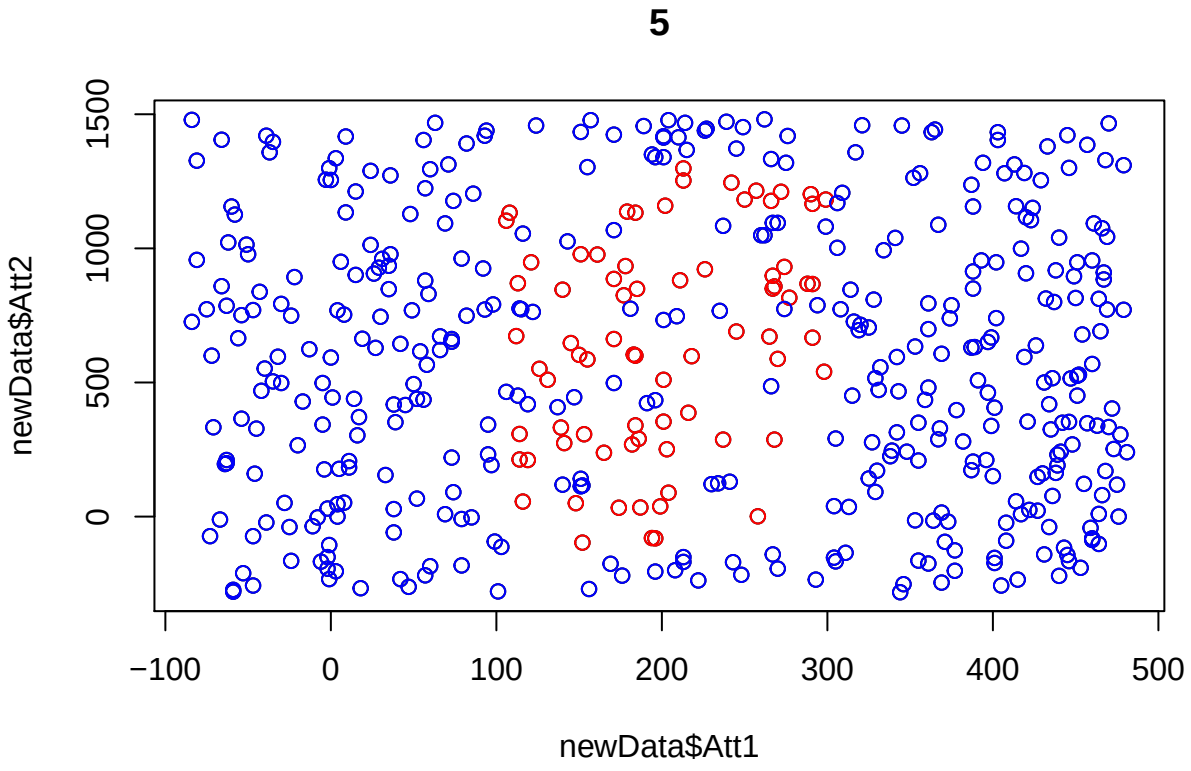

```
## Number of instances removed from majority class with ENN: 7   Time needed: 0
```




```
## Number of instances removed from majority class with ENN: 5   Time needed: 0.02
```




```
## Number of instances removed from majority class with ENN: 6   Time needed: 0
```




```
tpr.SMOTEun <- sum(fichero$Class[as.vector(CVperm)] == 0 & knn.pred == 1) / nClass0
tpr.SMOTEun
```

```
## [1] 0.73
```

```
#Obtenemos un 0.73 en la clase positiva en la clase positiva
tnr.SMOTEun <- sum(fichero$Class[as.vector(CVperm)] == 1 & knn.pred == 2) / nClass1
tnr.SMOTEun
```

```
## [1] 0.97
```

```
#Obtenemos un 0.97 en la clase negativa
gmean.SMOTEun <- sqrt(tpr.SMOTEun * tnr.SMOTEun)
gmean.SMOTEun
```

```
## [1] 0.8414868
```

```
#La media es de un 0.841
```

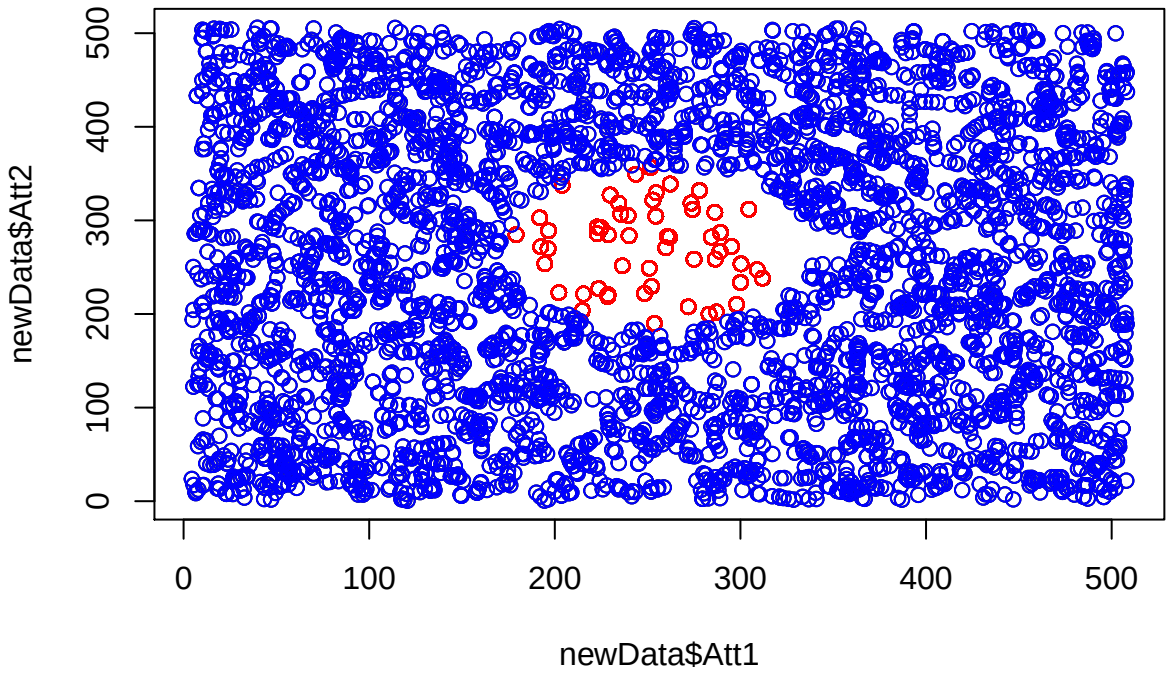
```
####
library(unbalanced)
fichero <- read.table("circle.txt", sep=",")
colnames(fichero) <- c("Att1", "Att2", "Class")
#Determinar el radio de imbalanceamiento
nClass0 <- sum(fichero$Class == 0)
nClass1 <- sum(fichero$Class == 1)
IR <- nClass1 / nClass0
IR #Por cada ejemplo positivo hay 42,4 negativos
```

```
## [1] 42.45455
```

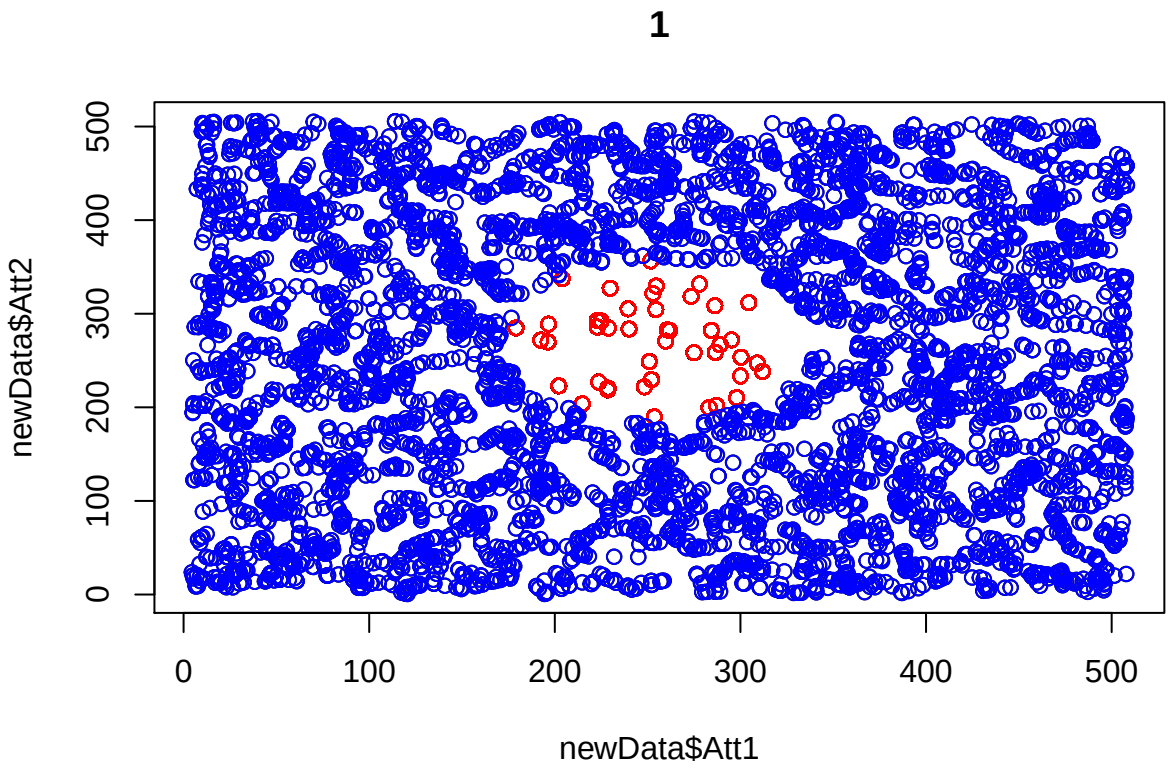
```
#Dividimos el dataset en 5 partes para la validaciÃ³n cruzada
set.seed(1234)
pos <- (1:dim(fichero)[1])[fichero$Class==0]
neg <- (1:dim(fichero)[1])[fichero$Class==1]
#Hacemos las divisiones en los 5 conjuntos de cada clase
CVperm_pos <- matrix(sample(pos,length(pos)), ncol=5, byrow=T)
CVperm_neg <- matrix(sample(neg,length(neg)), ncol=5, byrow=T)
#Unimos las dos clases
CVperm <- rbind(CVperm_pos, CVperm_neg)

input <- fichero[,-3]
output <- as.factor(fichero[,3])
data<-ubSMOTE(X=input, Y= output, perc.over = 100, perc.under = 200)
newData<-cbind(data$X, data$Y)
nClass0nuevos <- sum(newData$`data$Y` == 0)
nClass1nuevos <- sum(newData$`data$Y` == 1)
IR <- nClass1nuevos / nClass0nuevos
IR #Por cada ejemplo positivo hay 1 negativos
```

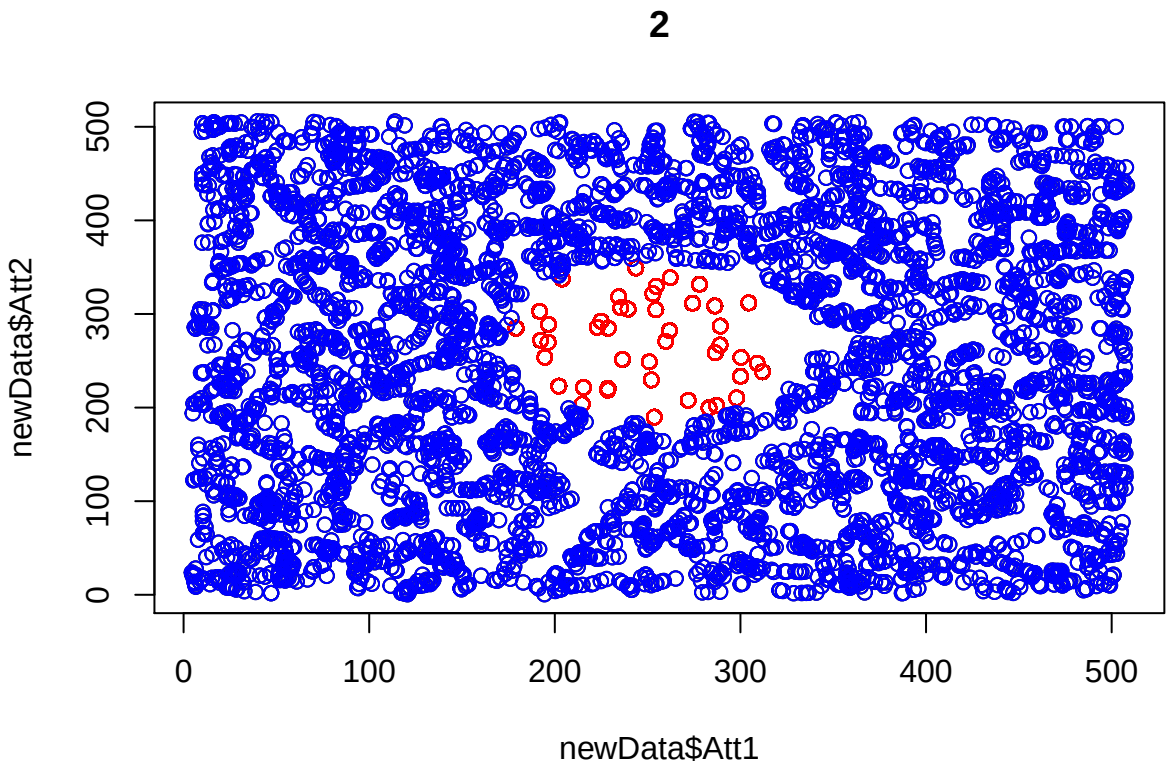
```
## [1] 1
```

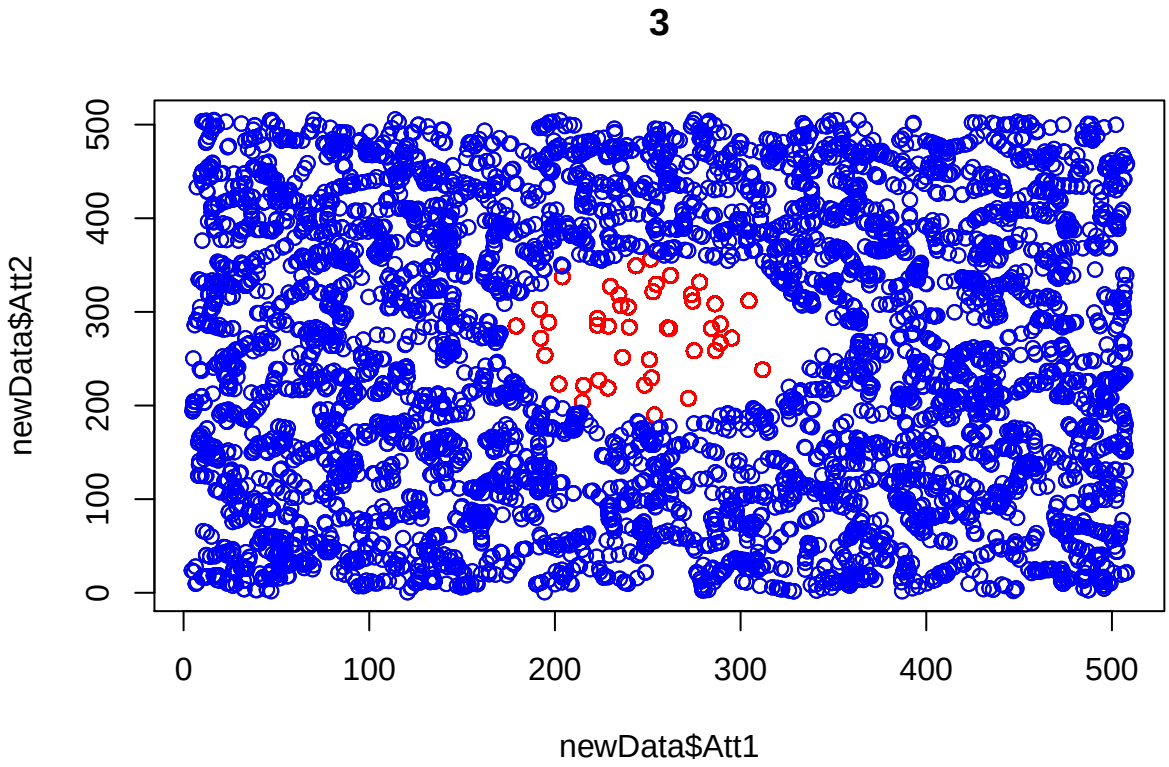
```
#Visualizamos la distribuciÃ³n de los datos
plot(newData$Att1, newData$Att2)
points(newData[newData$`data$Y`==0,1],newData[newData$`data$Y`==0,2],col="red")
points(newData[newData$`data$Y`==1,1],newData[newData$`data$Y`==1,2],col="blue")
```

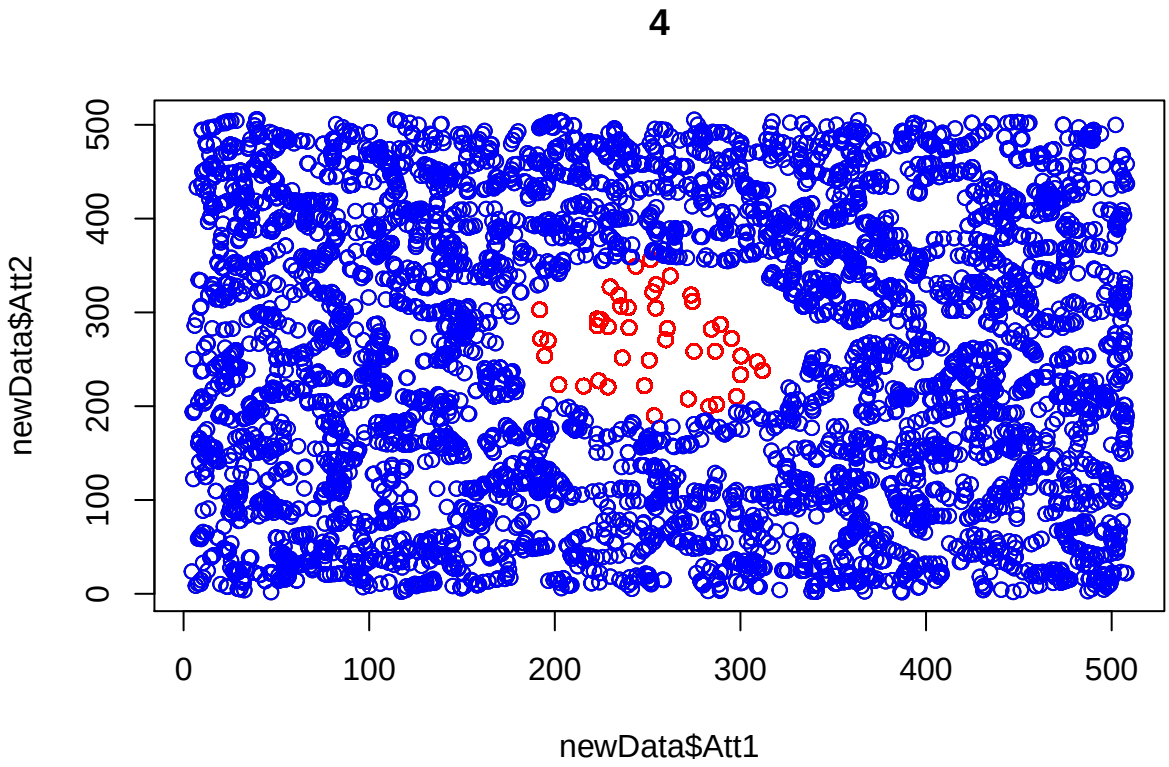


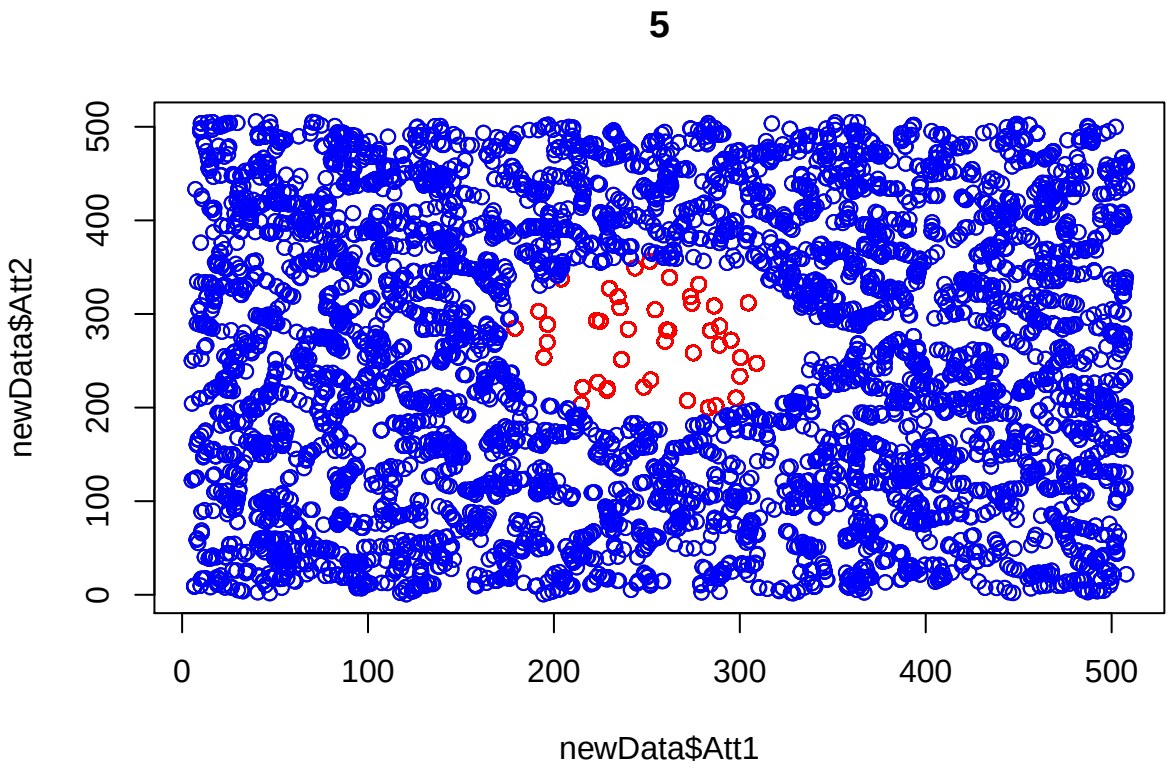
```
knn.pred=NULL
set.seed(1234)
knn.pred <- SMOTEUnbalanced()
```




2
newData$Att2
0
100
200
300
400
500
newData$Att1


3
newData$Att2
0
100
200
300
400
500
newData$Att1


4
newData$Att2
0
100
200
300
400
500
newData$Att1




```
tpr.SMOTEun <- sum(fichero$Class[as.vector(CVperm)] == 0 & knn.pred == 1) / nClass0
tpr.SMOTEun
```

```
## [1] 0.8363636
```

```
#Obtenemos un 0.836 en la clase positiva en la clase positiva
tnr.SMOTEun <- sum(fichero$Class[as.vector(CVperm)] == 1 & knn.pred == 2) / nClass1
tnr.SMOTEun
```

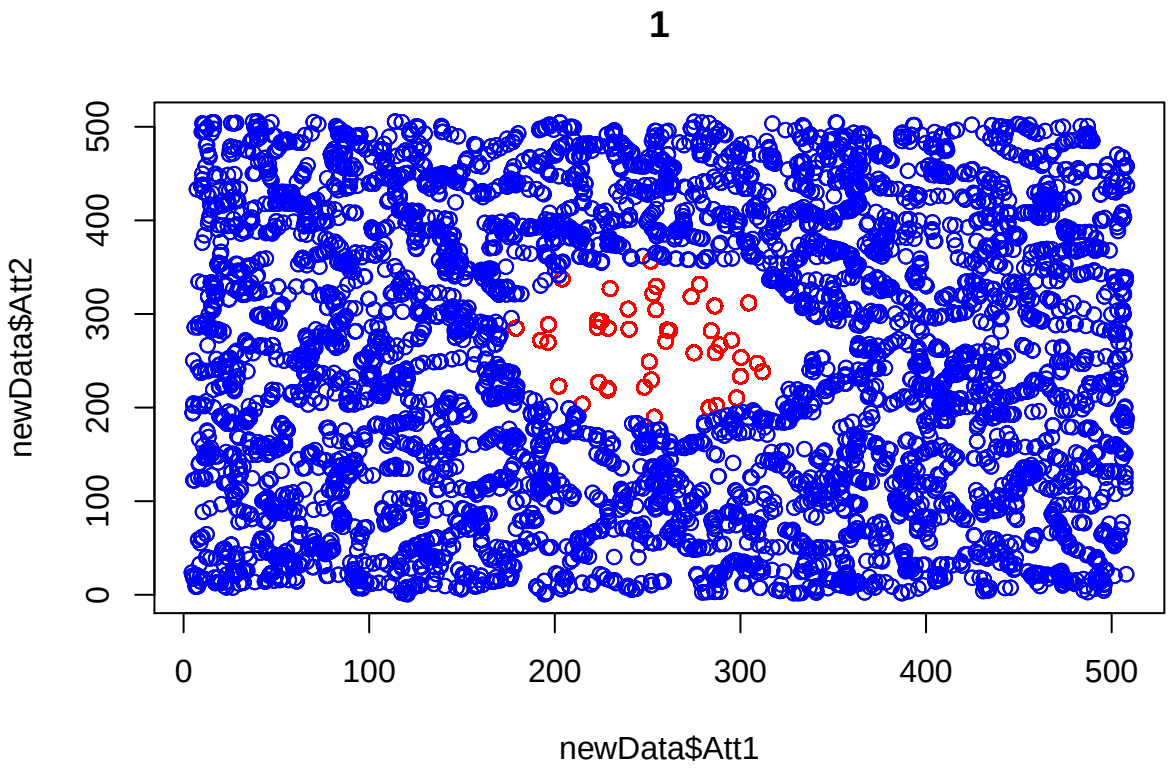
```
## [1] 0.9982869
```

```
#Obtenemos un 0.998 en la clase negativa
gmean.SMOTEun <- sqrt(tpr.SMOTEun * tnr.SMOTEun)
gmean.SMOTEun
```

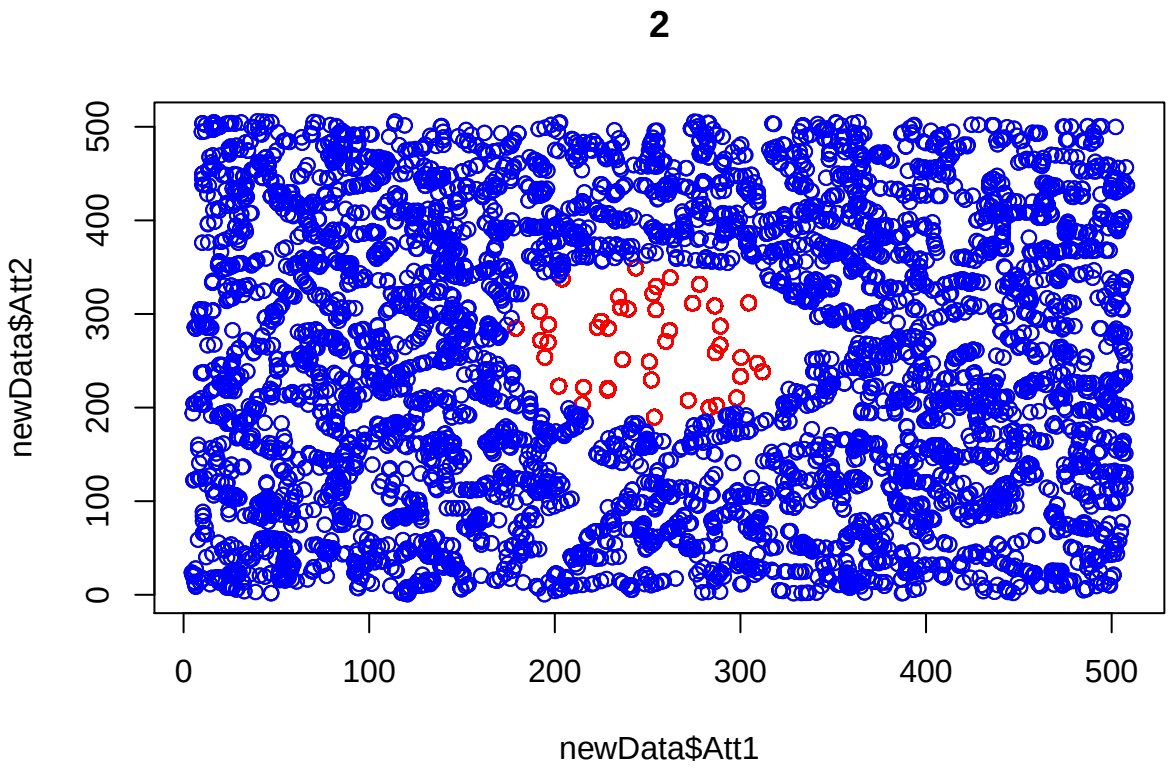
```
## [1] 0.9137455
```

```
#La media es de un 0.913

knn.pred=NULL
set.seed(1234)
knn.pred <- SMOTEUnbalanced(TRUE)
```

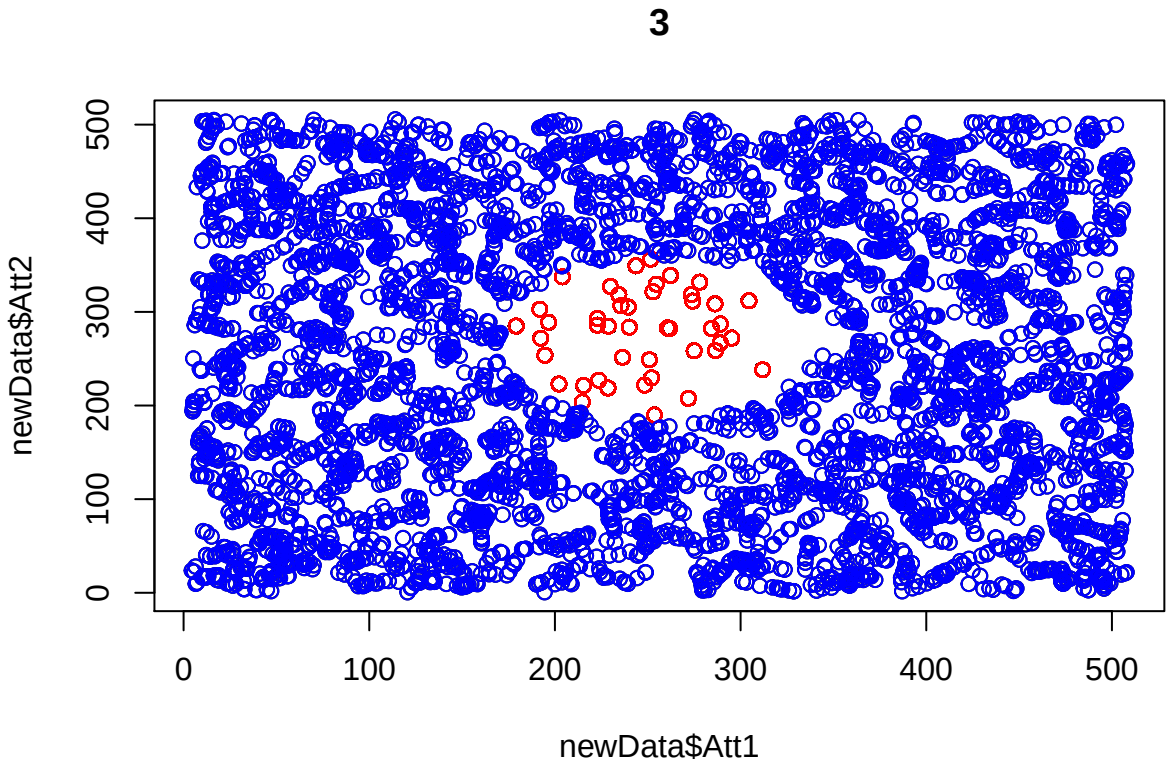
```
## Instances removed 0 : 0 % of 0 class ; 0 % of training ; Time needed 0
```



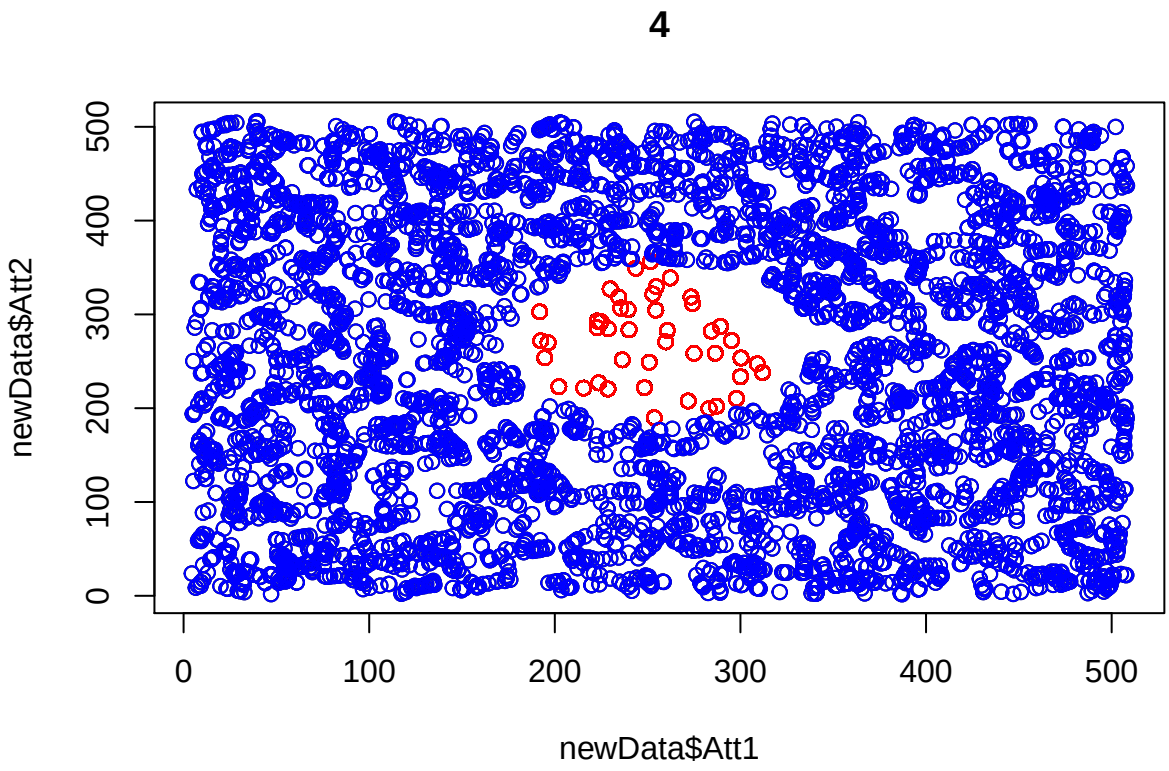

```
## Instances removed 1 : 0.01 % of 0 class ; 0.01 % of training ; Time needed 0.02
```



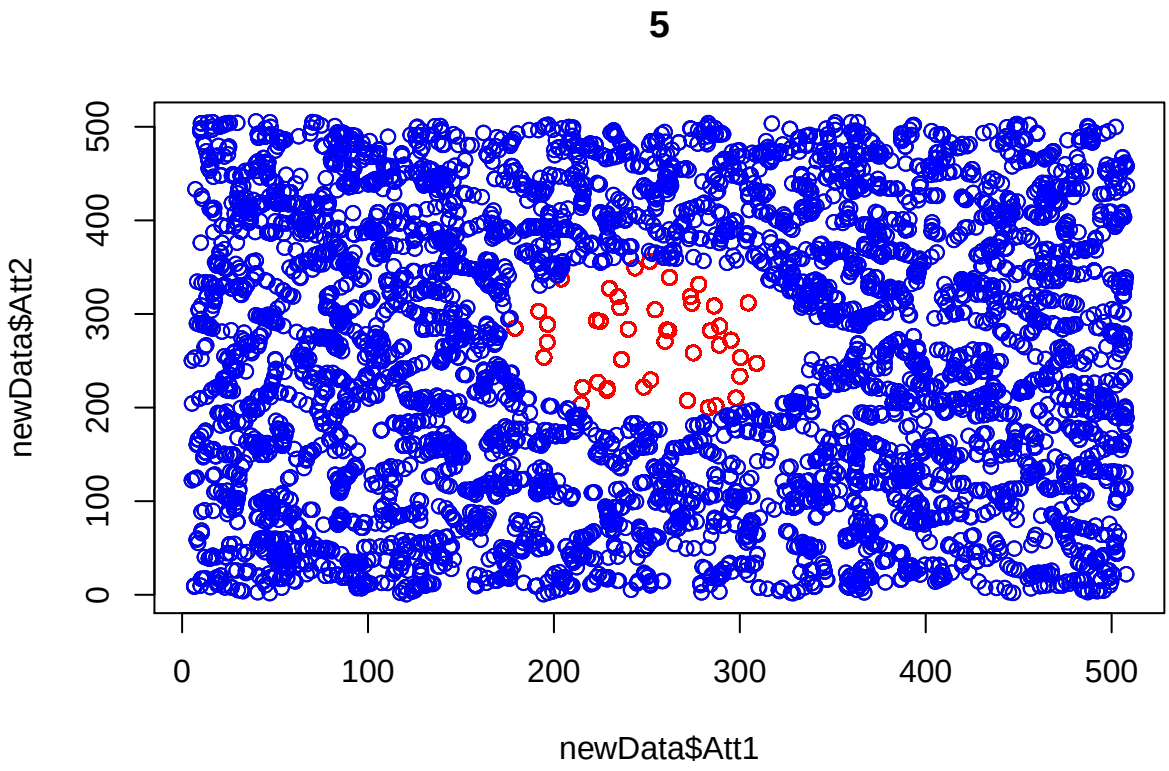

```
## Instances removed 0 : 0 % of 0 class ; 0 % of training ; Time needed 0
```




```
## Instances removed 0 : 0 % of 0 class ; 0 % of training ; Time needed 0
```




```
## Instances removed 0 : 0 % of 0 class ; 0 % of training ; Time needed 0
```




```
tpr.SMOTEun <- sum(fichero$Class[as.vector(CVperm)] == 0 & knn.pred == 1) / nClass0
tpr.SMOTEun
```

```
## [1] 0.8363636
```

```
#Obtenemos un 0.836 en la clase positiva en la clase positiva
tnr.SMOTEun <- sum(fichero$Class[as.vector(CVperm)] == 1 & knn.pred == 2) / nClass1
tnr.SMOTEun
```

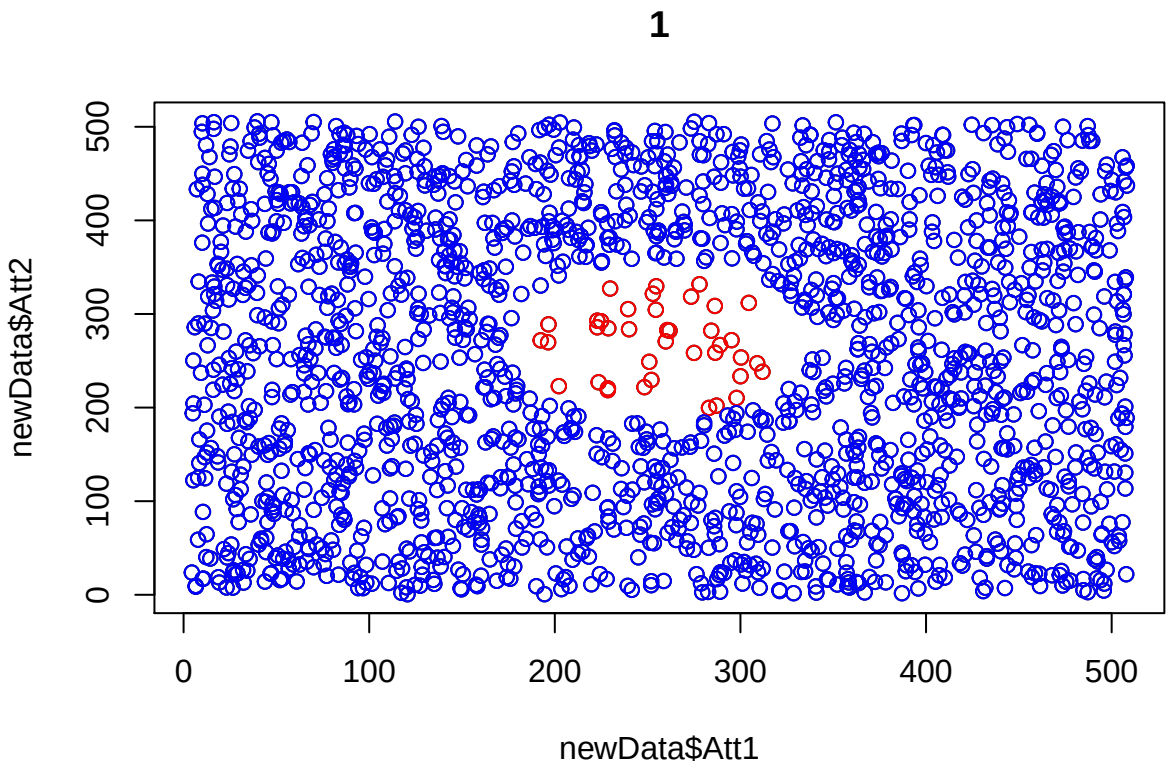
```
## [1] 0.9982869
```

```
#Obtenemos un 0.998 en la clase negativa
gmean.SMOTEun <- sqrt(tpr.SMOTEun * tnr.SMOTEun)
gmean.SMOTEun
```

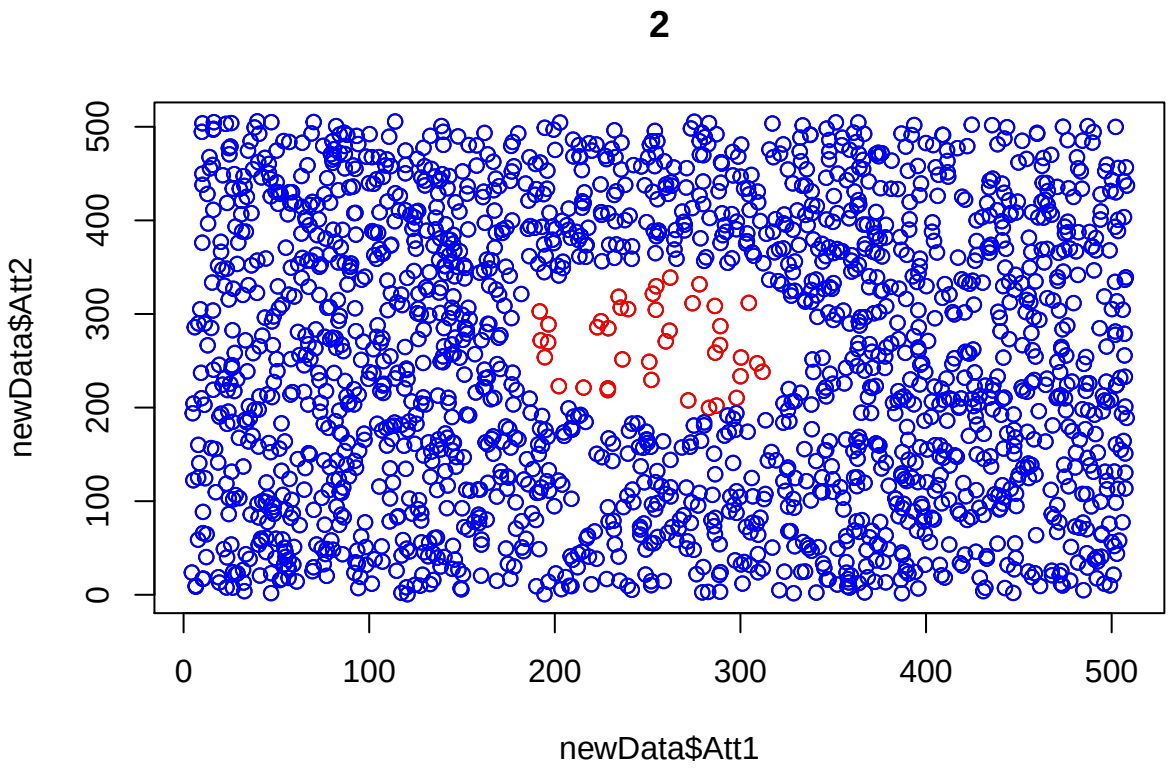
```
## [1] 0.9137455
```

```
#La media es de un 0.913

knn.pred=NULL
set.seed(1234)
knn.pred <- SMOTEUnbalanced(FALSE,TRUE)
```

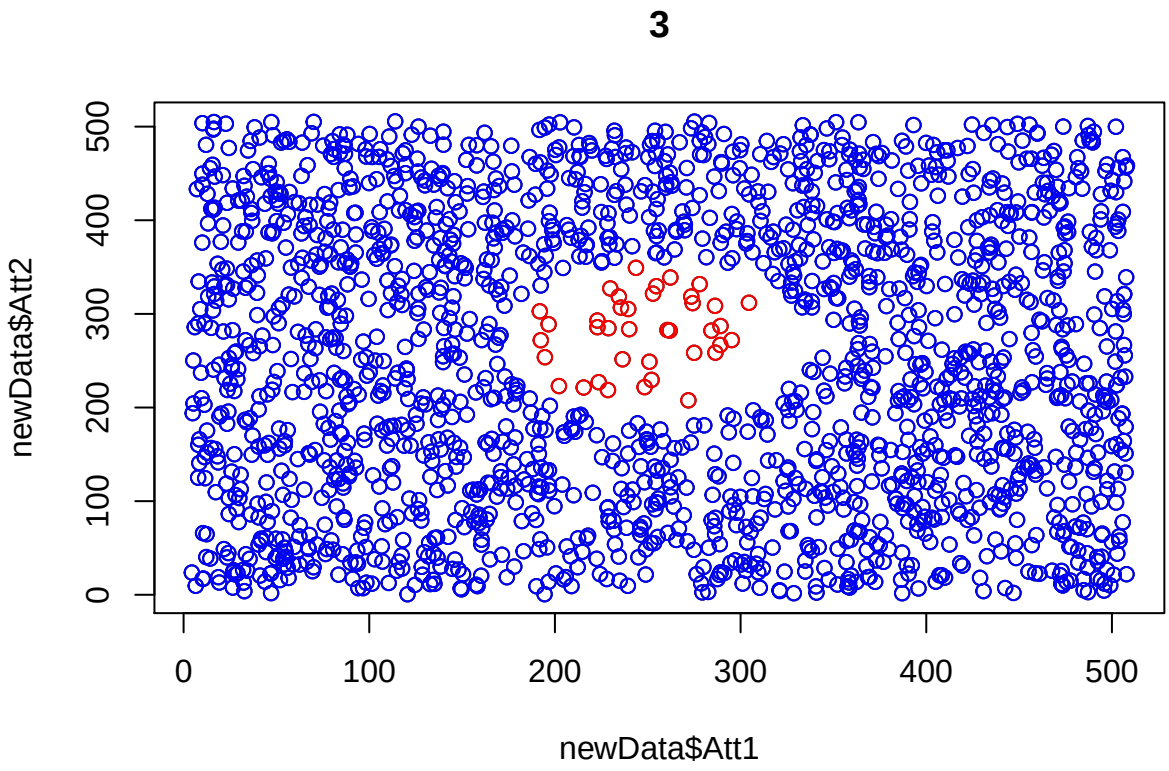
```
## Number of instances removed from majority class with ENN: 5   Time needed: 0
```



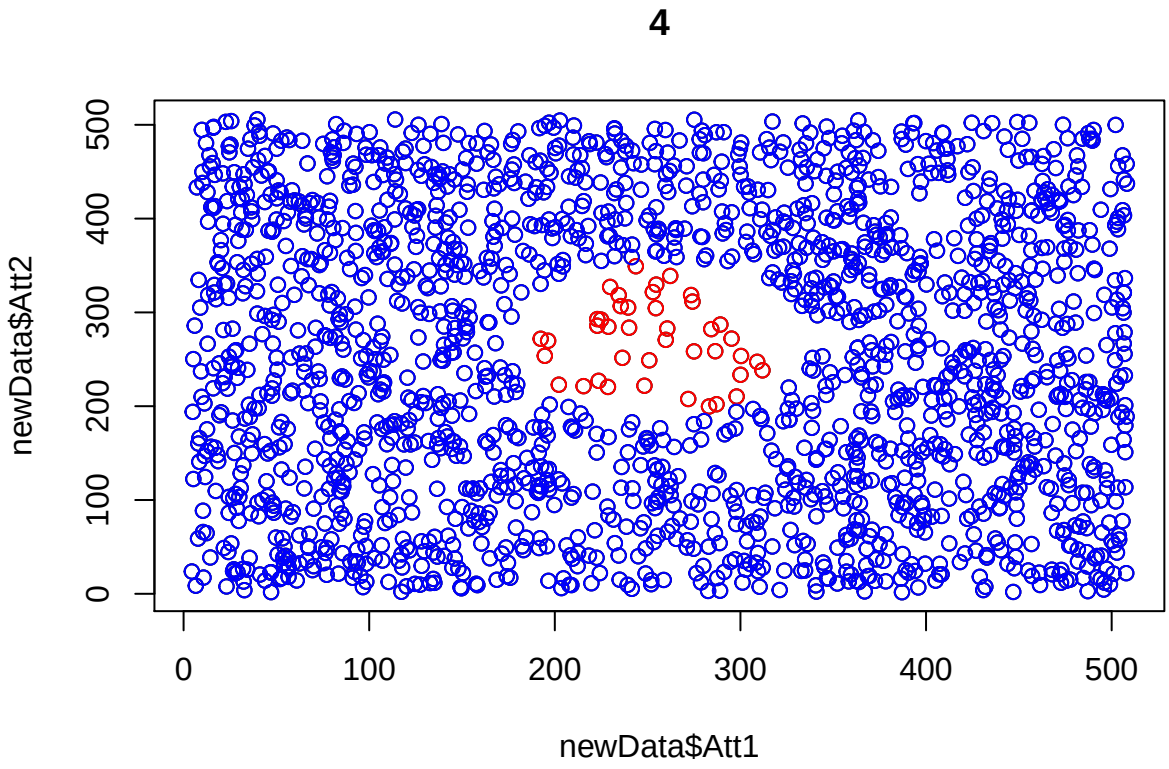

```
## Number of instances removed from majority class with ENN: 5   Time needed: 0
```




```
## Number of instances removed from majority class with ENN: 6   Time needed: 0.02
```

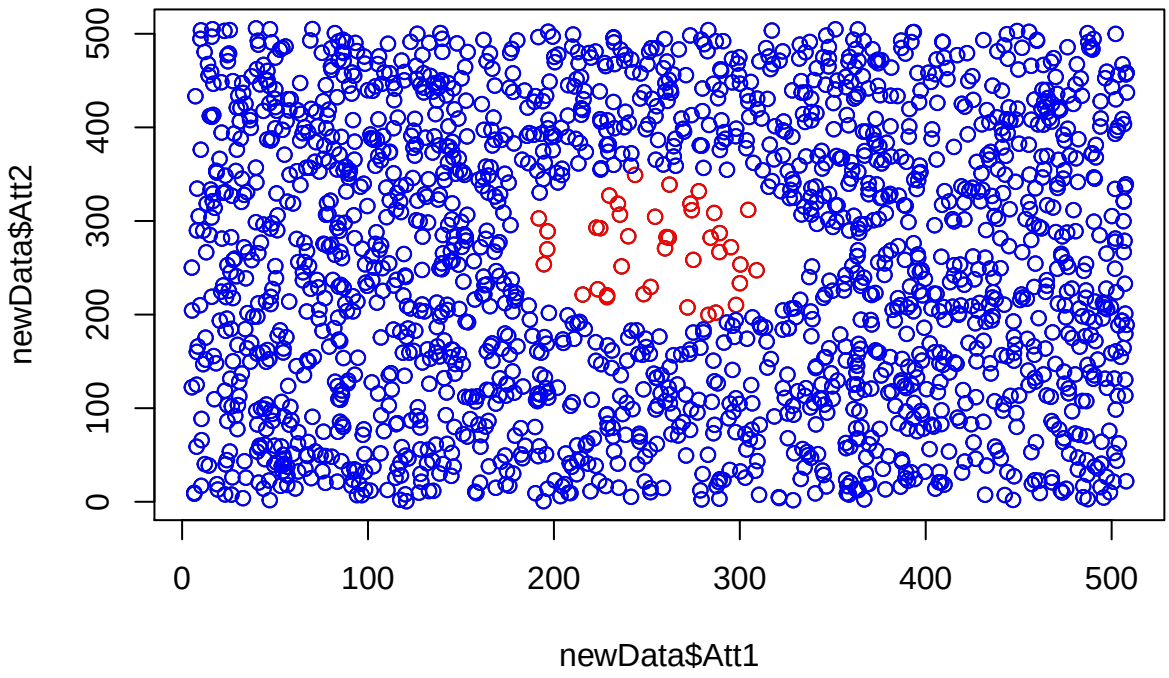



```
## Number of instances removed from majority class with ENN: 3   Time needed: 0
```




```
## Number of instances removed from majority class with ENN: 4   Time needed: 0
```

5




```
tpr.SMOTEun <- sum(fichero$Class[as.vector(CVperm)] == 0 & knn.pred == 1) / nClass0
tpr.SMOTEun
```

```
## [1] 0.7818182
```

```
#Obtenemos un 0.781 en la clase positiva en la clase positiva
tnr.SMOTEun <- sum(fichero$Class[as.vector(CVperm)] == 1 & knn.pred == 2) / nClass1
tnr.SMOTEun
```

```
## [1] 0.9995717
```

```
#Obtenemos un 0.999 en la clase negativa
gmean.SMOTEun <- sqrt(tpr.SMOTEun * tnr.SMOTEun)
gmean.SMOTEun
```

```
## [1] 0.8840155
```

```
#La media es de un 0.884
```

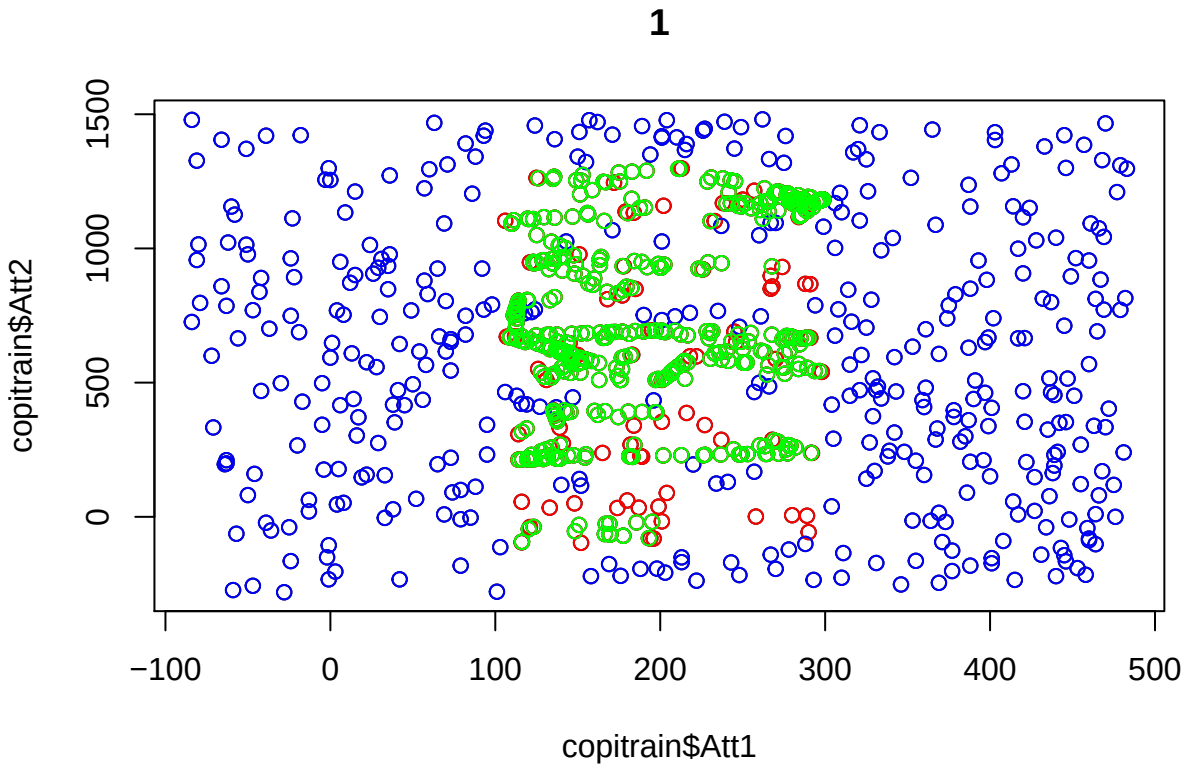
```
#ImplementaciÃ³n Borderline-SMOTE1
fichero <- read.table("subclus.txt", sep=",")
colnames(fichero) <- c("Att1", "Att2", "Class")
#Determinar el radio de imbalanceamiento
nClass0 <- sum(fichero$Class == 0)
nClass1 <- sum(fichero$Class == 1)
IR <- nClass1 / nClass0
IR #Por cada ejemplo positivo hay 5 negativos
```

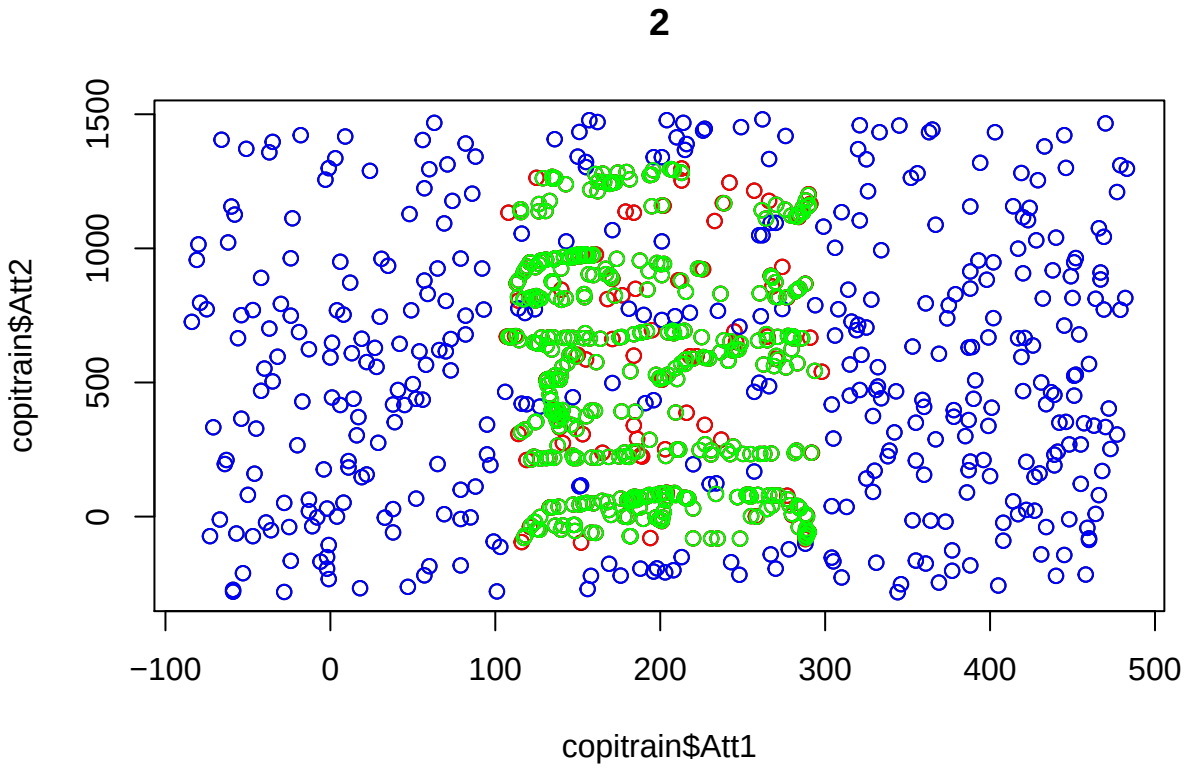
```
## [1] 5
```

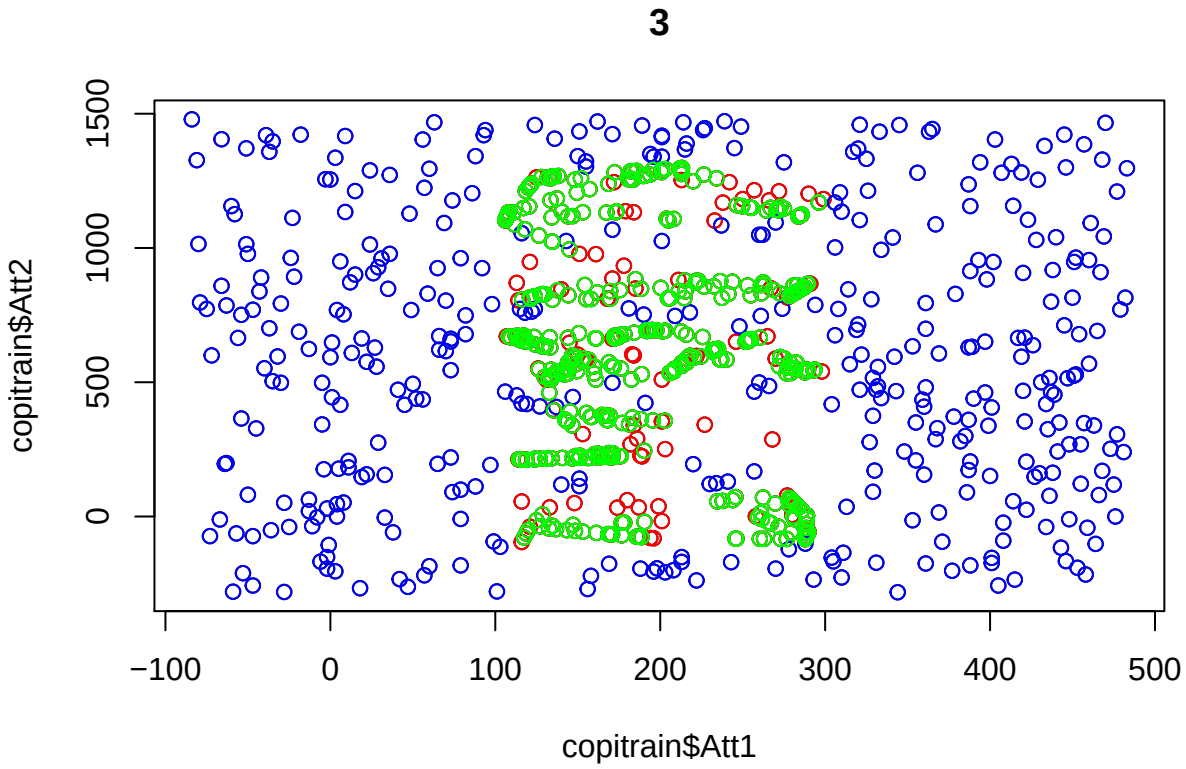
```r
#Dividimos el dataset en 5 partes para la validaciÃ³n cruzada
set.seed(1234)
pos <- (1:dim(fichero)[1])[fichero$Class==0]
neg <- (1:dim(fichero)[1])[fichero$Class==1]
#Hacemos las divisiones en los 5 conjuntos de cada clase
CVperm_pos <- matrix(sample(pos,length(pos)), ncol=5, byrow=T)
CVperm_neg <- matrix(sample(neg,length(neg)), ncol=5, byrow=T)
#Unimos las dos clases
CVperm <- rbind(CVperm_pos, CVperm_neg)

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)
}
#FunciÃ³n GetNeighbors
getNeighbors <- function(x, todos.los.indices, train){
  #1 Por cada x, recorro todas las instancias y busco sus 5 elementos mÃ¡s cercanos
  #1.1 Creo la funciÃ³n que devuelva la distancia de x a cada elemento de la clase minoritaria
  cercano = function(y){
    distance(x,y,fichero)
  }
  #1.2 Aplico para encontrar la distancia
  #points(fichero[x,],col="black")
  distancias <- sapply(todos.los.indices,cercano )
  #1.3 Busco los 5 elementos con menor distancia salvo el mismo
  matriz.elemento.distancia <- cbind(todos.los.indices,distancias)
  menores = function(matriz){
    #Ordeno las distancias
    a = matriz[,2]
    a <- a[order(a)]
    #Si la primera distancia es 0, es que es el mismo y lo elimino
    if (a[1]==0){
      a <- c(a[2:length(a)])
    }
    #Me quedo con las 5 mÃ¡s pequeÃ±as
    a <- c(a[1:5])
    return(a)
  }
  los.menores <- menores(matriz.elemento.distancia)
  #matriz.elemento.distancia[los.5.cercanos,1]
  los.5.cercanos <- matriz.elemento.distancia[,2] %in% los.menores
  valores.los.5.cercanos <- todos.los.indices[los.5.cercanos]
  #points(fichero[valores.los.5.cercanos,],col="blue")
```

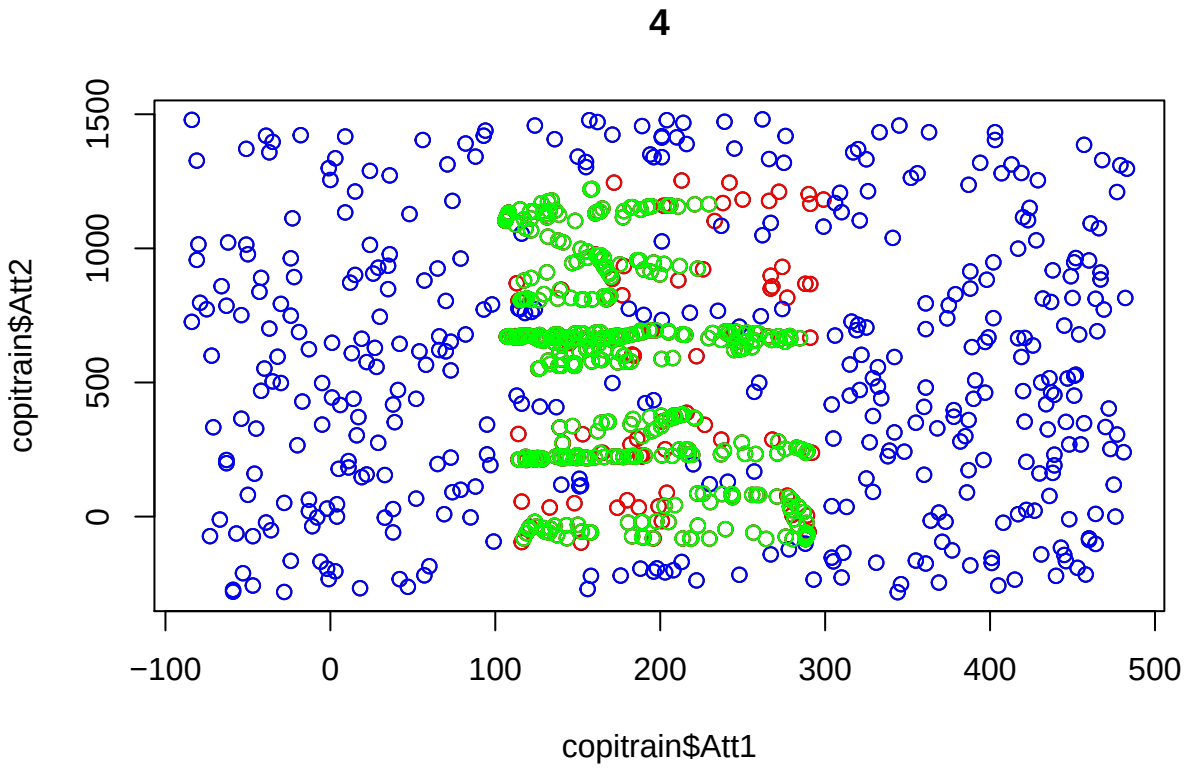
```
  return(valores.los.5.cercanos)
}
#FunciÃ³n SyntheticInstance
syntheticInstance2 <- function(elemento, cercano){
  #Cojo un elemento de los cercanos aleatoriamente
  aleatorio.cercanos <- 1
  valor.aleatorio.cercanos <- cercano
  #Saco los valores att1 y att2 de mi elemento y del aleatorio cercano
  att1.x <- fichero[elemento,1]
  att1.y <- fichero[valor.aleatorio.cercanos,1]
  att1.z = att1.y - att1.x
  att2.x <- fichero[elemento,2]
  att2.y <- fichero[valor.aleatorio.cercanos,2]
  att2.z = att2.y - att2.x
  multiplicador = runif(1,0,1)
  #print("elemento,multiplciador,att1.x,att1.y,att1.z,att2.x,att2.y,att2.z,aleatoriodecercanos:")
  #print(elemento);print(multiplicador);print(att1.x);print(att1.y);print(att1.z);print(att2.x);print(a
  att1.z = att1.x + (multiplicador*att1.z)
  att2.z = att2.x + (multiplicador*att2.z)
  #print("Final z: ");print(att1.z);print(att2.z)
  salida = NULL
  salida$Att1 = att1.z
  salida$Att2 = att2.z
  return(list(salida,aleatorio.cercanos))
  #return(salida)
}
BorderlineSMOTE = function(slot,visualizacion=FALSE){
  #Preparo los conjuntos de train y test
  train <- fichero[-CVperm[,slot],-3]
  classes.train <- fichero[-CVperm[,slot], 3]
  test  <- fichero[CVperm[,slot], -3]
  #Busco los Ã?ndices de la clase minoritaria
  indices <- CVperm_pos[,slot]
  indices.neg <- CVperm_neg[,slot]
  minority.indices <- (1:length(pos))[-indices]
  union.indices <- c(indices,indices.neg)
  todos.los.indices <- (1:(dim(fichero)[1]))[-union.indices]
  mayority.indices <- todos.los.indices[!(todos.los.indices %in% minority.indices)]
  #Creo el nuevo train
  copitrain <- train
  instancias.nuevas <- NULL
  total.a.crear = dim(train)[1] - length(minority.indices)
  #Para cada Ã?ndice de la clase minoritaria creare un nuevo elemento
  j=0
  i=0
  while(i!=total.a.crear){
    j=j+1
    if(j==(length(minority.indices)))
      j=1
    #Busco los elementos cercanos
    cercanos <- getNeighbors(minority.indices[j],todos.los.indices,fichero)
    if ((sum(mayority.indices %in% cercanos)) > (length(cercanos)/2) ){ #Se considera DANGER
      cercanos.danger <- getNeighbors(minority.indices[j],minority.indices,fichero)
```

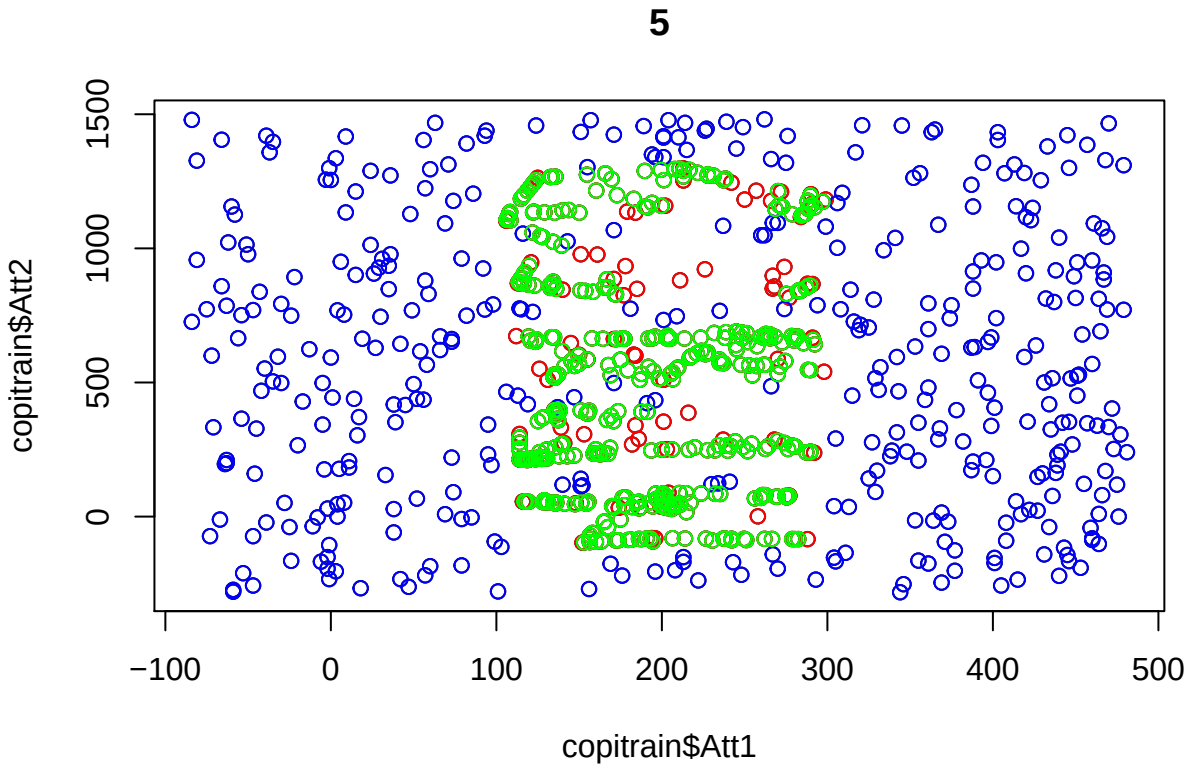
```
      aleatorio.danger <- sample(1:5,1)
      if ((i+aleatorio.danger) > total.a.crear)
        aleatorio.danger = total.a.crear - i
      i = i + aleatorio.danger
      for (h in 1:aleatorio.danger){
        salida.ynthetic <- syntheticInstance2(minority.indices[j],cercanos.danger[h])
        instancia.nueva = salida.ynthetic[[1]]
        #AÃ±ado la nueva instancia al nuevo train
        copitrain <- rbind(copitrain, instancia.nueva)
        instancias.nuevas <- rbind(instancias.nuevas,instancia.nueva)
        #AÃ±ado la clase del nuevo elemento
        classes.train <- c(classes.train,0)
      }
    }
  }
  #Realizo knn para la prediccIÃ³n
  predictions <-  knn(copitrain, test, classes.train, k = 3)
  knn.pred <- c(knn.pred, predictions)
  if (visualizacion == TRUE){
    plot(copitrain$Att1,copitrain$Att2,title(slot))
    points(copitrain[classes.train==0,1],copitrain[classes.train==0,2],col="red")
    points(copitrain[classes.train==1,1],copitrain[classes.train==1,2],col="blue")
    points(instancias.nuevas,col="green")
  }
  return(knn.pred)
}


#Borderline-SMOTE1
set.seed(1234)
knn.pred = NULL
#Aplico Borderline-SMOTE1 y visualizo los resultados
for (j in 1:5){
  knn.pred <- BorderlineSMOTE(j,TRUE)
}
```


1
copitrain$Att2
1500
1000
500
0
−100
0
100
200
300
400
500
copitrain$Att1


2
copitrain$Att2
1500
1000
500
0
−100
0
100
200
300
400
500
copitrain$Att1


3
copitrain$Att2
1500
1000
500
0
−100
0
100
200
300
400
500
copitrain$Att1


4
copitrain$Att2
1500
1000
500
0
−100
0
100
200
300
400
500
copitrain$Att1




```
tpr.BorSMOTE <- sum(fichero$Class[as.vector(CVperm)] == 0 & knn.pred == 1) / nClass0
tpr.BorSMOTE
```

```
## [1] 0.87
```

```
#Obtenemos un 0.86 en la clase positiva en la clase positiva
tnr.BorSMOTE <- sum(fichero$Class[as.vector(CVperm)] == 1 & knn.pred == 2) / nClass1
tnr.BorSMOTE
```

```
## [1] 0.934
```

```
#Obtenemos un 0.934 en la clase negativa
gmean.BorSMOTE <- sqrt(tpr.BorSMOTE * tnr.BorSMOTE)
gmean.BorSMOTE
```

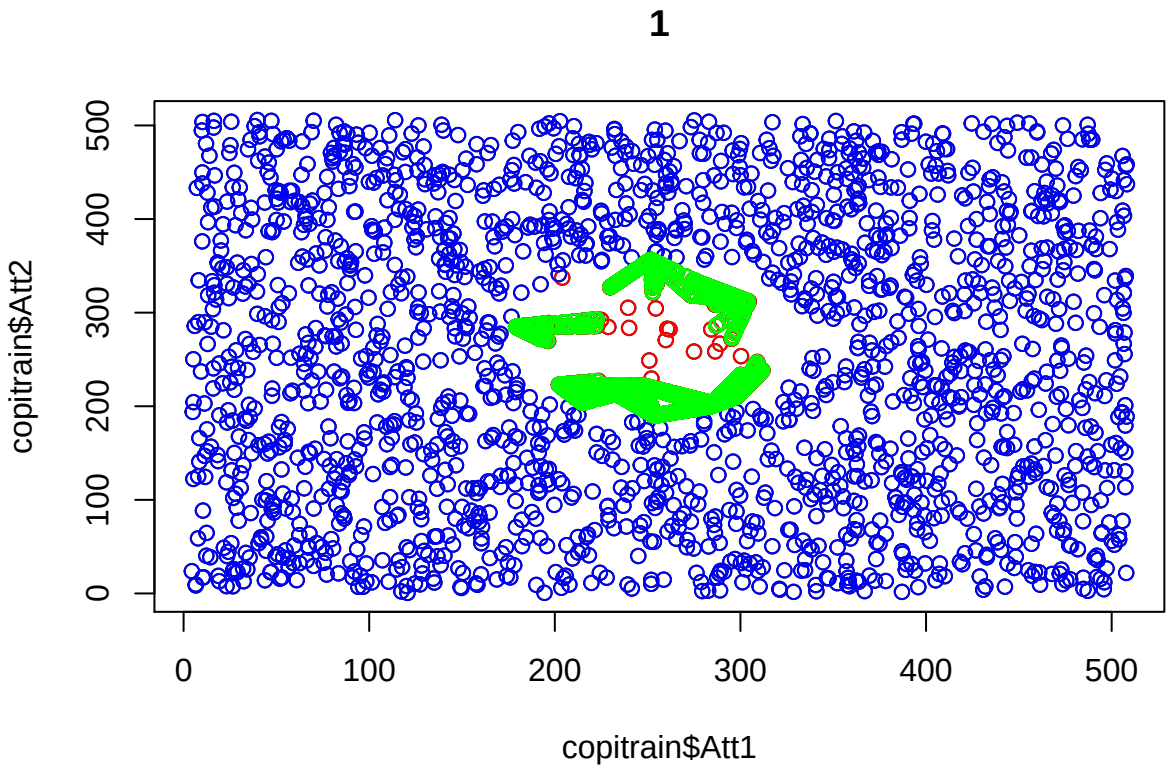
```
## [1] 0.9014322
```

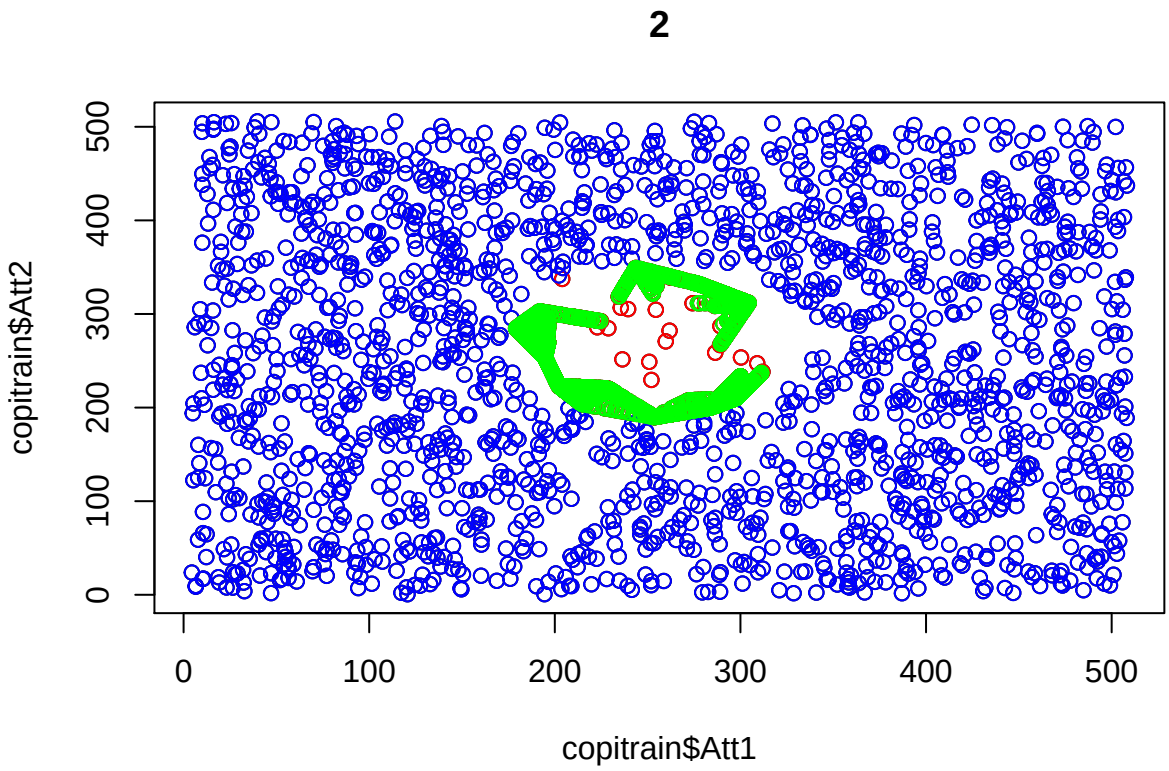
```
#La media es de un 0.896
```

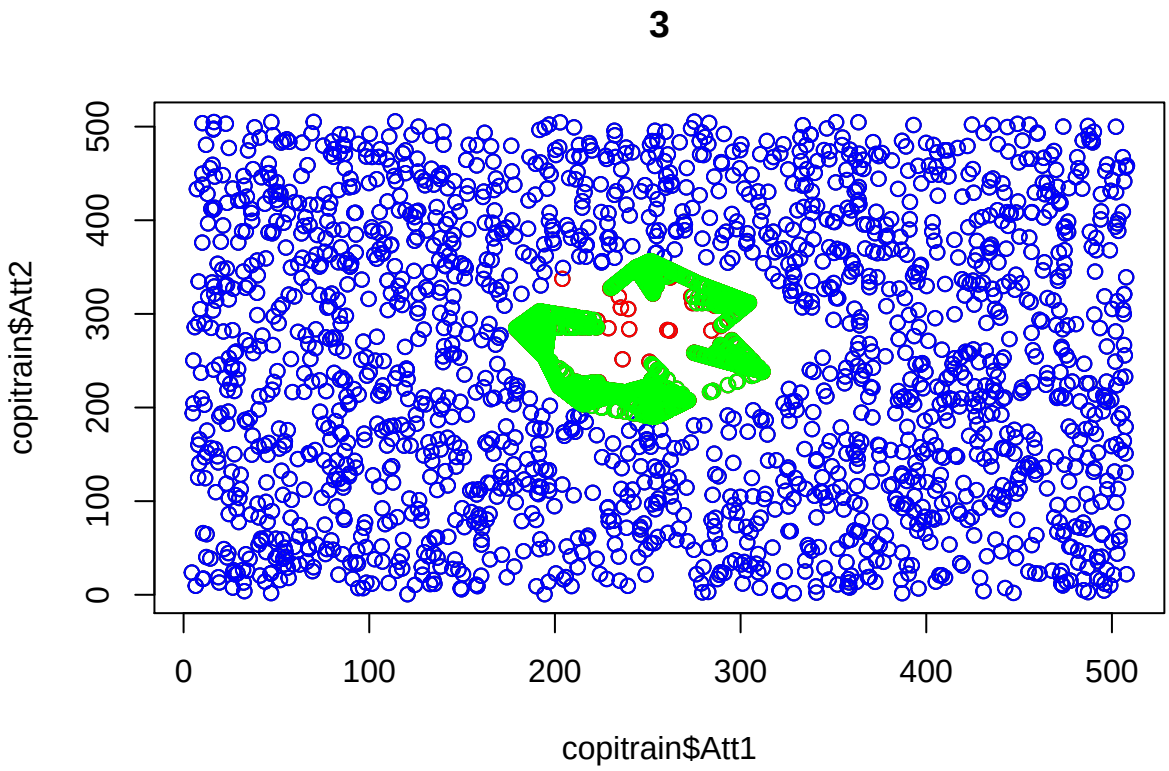
```
#Circle
fichero <- read.table("circle.txt", sep=",")
colnames(fichero) <- c("Att1", "Att2", "Class")
#Determinar el radio de imbalanceamiento
nClass0 <- sum(fichero$Class == 0)
nClass1 <- sum(fichero$Class == 1)
IR <- nClass1 / nClass0
IR #Por cada ejemplo positivo hay 5 negativos
```

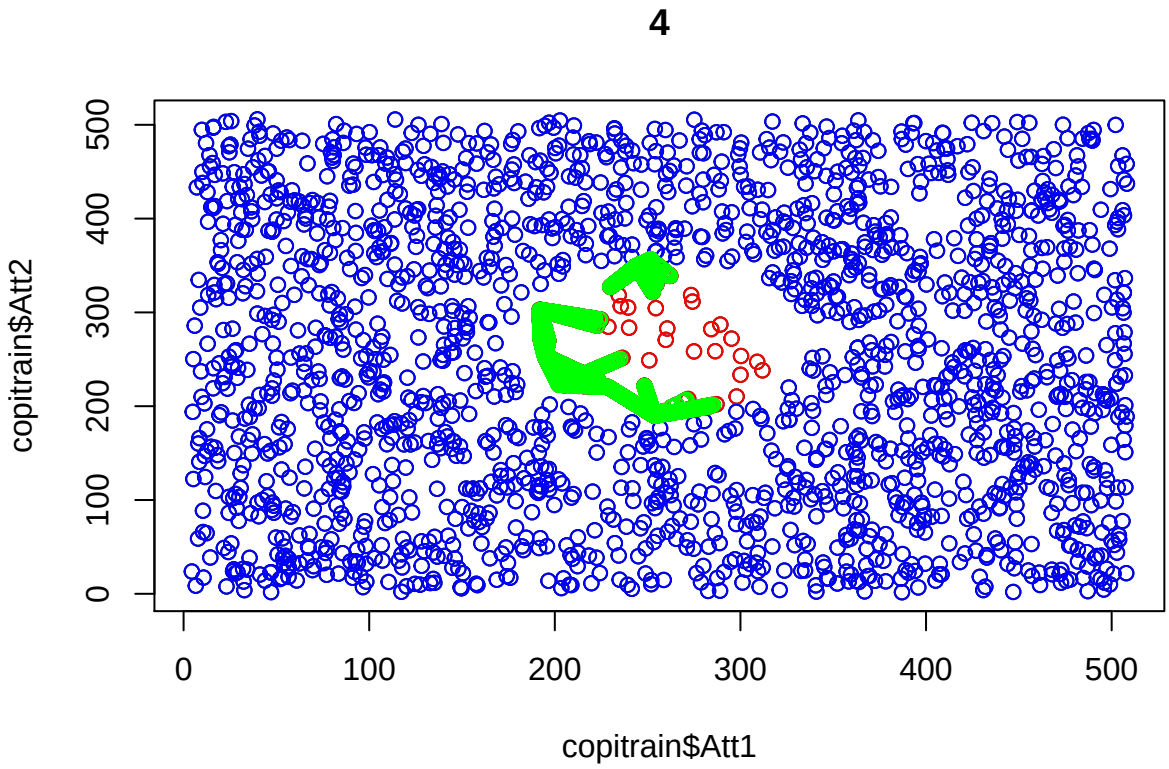
```
## [1] 42.45455
```

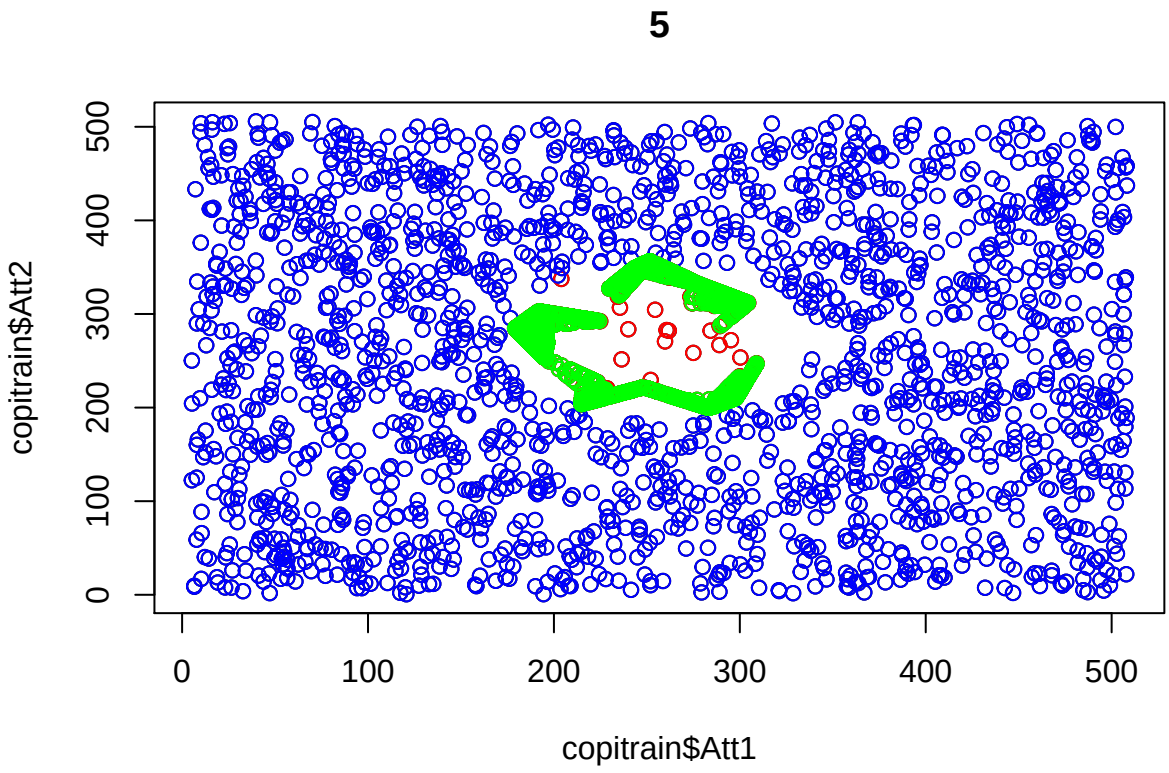
```r
#Dividimos el dataset en 5 partes para la validaciÃ³n cruzada
set.seed(1234)
pos <- (1:dim(fichero)[1])[fichero$Class==0]
neg <- (1:dim(fichero)[1])[fichero$Class==1]
#Hacemos las divisiones en los 5 conjuntos de cada clase
CVperm_pos <- matrix(sample(pos,length(pos)), ncol=5, byrow=T)
CVperm_neg <- matrix(sample(neg,length(neg)), ncol=5, byrow=T)
#Unimos las dos clases
CVperm <- rbind(CVperm_pos, CVperm_neg)

#Borderline-SMOTE1
set.seed(1234)
knn.pred = NULL
#Aplico Borderline-SMOTE1 y visualizo los resultados
for (j in 1:5){
  knn.pred <- BorderlineSMOTE(j,TRUE)
}
```




2
copitrain$Att2
0
100
200
300
400
500
copitrain$Att1
0
100
200
300
400
500


3
copitrain$Att2
0
100
200
300
400
500
copitrain$Att1


4
copitrain$Att2
0
100
200
300
400
500
copitrain$Att1




```
tpr.BorSMOTE <- sum(fichero$Class[as.vector(CVperm)] == 0 & knn.pred == 1) / nClass0
tpr.BorSMOTE
```

```
## [1] 0.9272727
```

```
#Obtenemos un 0.927 en la clase positiva en la clase positiva
tnr.BorSMOTE <- sum(fichero$Class[as.vector(CVperm)] == 1 & knn.pred == 2) / nClass1
tnr.BorSMOTE
```

```
## [1] 0.9978587
```

```
#Obtenemos un 0.997 en la clase negativa
gmean.BorSMOTE <- sqrt(tpr.BorSMOTE * tnr.BorSMOTE)
gmean.BorSMOTE
```

```
## [1] 0.9619185
```

```
#La media es de un 0.961
```