Proyecto Final. Cardiotocography

Anabel Gómez Ríos y Gustavo Rivas Gervilla

16 de junio de 2016

1. Definición del problema a resolver y enfoque elegido.

En este proyecto vamos a trabajar con una base da datos algo mayor que las que hemos venido usando en las prácticas (2126 instancias con 23 atributos cada una) con el objetivo de poner en práctica los conocimientos adquiridos en la asignatura para resolver un problema de clasificación del mundo real.

La base de datos elegida es Cardiotocography del respositorio de bases de datos UCI la cual la podemos descargar **aquí**. En esta base de datos se recogen distintas características de cardiotocografías en las cuales se mide la frencuencia cardiaca fetal (FHR), los movimientos fetales (FM) y las contracciones uterinas (UC), obteniendo las siguientes características a partir de estos datos:

1. LB: punto de referencia del FHR en pulsaciones por minuto.
2. AC: aceleraciones del pulso por segundo.
3. FM: movimientos fetales por segundo.
4. UC: contracciones uterinas por segundo.
5. DL: deceleraciones suaves por segundo.
6. DS: deceleraciones fuertes por segundo.
7. DP: deceleraciones prolongadas por segundo.
8. ASTV: porcentaje de tiempo con variaciones anormales cortas del pulso.
9. MSTV: media de las variaciones anormales cortas del pulso.
10. ALTV: porcentaje de tiempo con variaciones anormales largas del pulso.
11. MLTV: media de las variaciones anormales largas del pulso.
12. Width: amplitud del histograma FHR.
13. Min: mínimo del histograma FHR.
14. Max: máximo del hisotograma FHR.
15. Nmax: número de picos en el histograma.
16. Nzeros: número de ceros en el histograma.
17. Mode: moda del histograma.
18. Mean: media del histograma.
19. Median: mediana del histograma.
20. Variance: varianza del histograma.
21. Tendency: tendencia del histograma.
22. CLASS: código del tipo de patrón del histograma FHR [1-10].

23. NSP: código del estado fetal. [1: Normal, 2: Sospechoso y 3: Patológico]

Lo que queremos es emplear estos datos para poder predecir ante una nueva cardiotocografía si el estado del feto es normal, sospecho o patológico, es decir, vamos a predecir la variable NSP con el resto. Además, vamos a hacer la clasificación también según la variable CLASS, puesto que también es uno de los problemas de la base de datos.

El enfoque elegido por tanto es hacer clasificación multiclase para clasificar nuevos datos según dos variables (por separado), una que tiene 3 clases y otra que tiene 10.

```
# Leemos los datos
datos <- read.csv("datos.csv")
```

2. Codificación de los datos de entrada para hacerlos útiles a los algoritmos.

Nuestra base de datos estaba contenida en una hoja de cálculo. Para poder usarla dentro de R lo que hemos hecho es generar un CSV con los datos previamente formateados puesto que hemos tenido que cambiar el formato decimal de algunas columnas para que fuese el que emplea R. Además en el fichero original aparecían más variables como la fecha y el tiempo de inicio y fin de la cardiotocografía, las cuales no hemos considerado relevantes para el estudio por lo que no están presententes en el CSV.

3. Valoración del interés de las variables para el problema y selección de un subconjunto en su caso.

En primer lugar tenemos que `Width` se calcula como la diferencia entre `Max` y `Min` con lo cual suponemos que una de las tres no tendrá relevancia ya que la información aportada por ella se puede deducir de las otras dos.

Para el resto de variables dado el poco conocimiento que tenemos en la materia no podemos saber qué factores son los que más influyen en determinar el estado del feto por tanto hemos decidido realizar un análisis de componentes principales para ver si podemos reducir el número de variables a considerar, haciendo que los algoritmos sean más eficientes en tiempo. La técnica que hemos usado en clase para tal propósito ha sido emplear el Lasso para obtener aquellas variables que sus coeficientes estuviesen por encima de un cierto umbral determinado por nosotros. Esto precisamente es lo que nos ha llevado a decantarnos por el PCA ya que con él podemos saber el conjunto de variables que son capaces de explicar al menos 95% de la variabilidad de los datos (aunque podemos cambiar este 95% y aumentarlo para que sea más estricto). Para saber cómo emplear PCA en R hemos consultado el enlace [2] de la bibliografía.

Lo primero que vamos a hacer es separar los datos en las muestras de entrenamiento y test (80-20) que emplearemos a lo largo de todo el estudio. En esta ocasión como la variable a predecir no depende de la media de otras variables entonces vamos a poder realizar un particionado homogéneo de los datos para tener una distribución de las clases de cada muestra lo más uniforme posible (no corremos el riesgo de contaminar la variable con datos de test como un ocurría en prácticas).

```
set.seed(1)
# Cogemos los índices del 80% de los datos para cada clase
train_idx <- c(sample(which(datos$NSP == 1), size =
                        ceiling(0.8*sum(datos$NSP==1))),
               sample(which(datos$NSP == 2), size =
                        ceiling(0.8*sum(datos$NSP==2))),
               sample(which(datos$NSP == 3), size =
```

```
                        ceiling(0.8*sum(datos$NSP==3))))

# Hacemos el conjunto de train con estos índices
train <- datos[train_idx,]
# Hacemos el conjunto de test con todas aquellas variables que no tengan estos
# índices
test <- datos[-train_idx,]
```

```
set.seed(1)
# Cogemos los índices del 80% de los datos para cada clase

train_idx <- c(sample(which(datos$CLASS == 1), size =
                        ceiling(0.8*sum(datos$CLASS==1))),
               sample(which(datos$CLASS == 2), size =
                        ceiling(0.8*sum(datos$CLASS==2))),
               sample(which(datos$CLASS == 3), size =
                        ceiling(0.8*sum(datos$CLASS==3))),
               sample(which(datos$CLASS == 4), size =
                        ceiling(0.8*sum(datos$CLASS==4))),
               sample(which(datos$CLASS == 5), size =
                        ceiling(0.8*sum(datos$CLASS==5))),
               sample(which(datos$CLASS == 6), size =
                        ceiling(0.8*sum(datos$CLASS==6))),
               sample(which(datos$CLASS == 7), size =
                        ceiling(0.8*sum(datos$CLASS==7))),
               sample(which(datos$CLASS == 8), size =
                        ceiling(0.8*sum(datos$CLASS==8))),
               sample(which(datos$CLASS == 9), size =
                        ceiling(0.8*sum(datos$CLASS==9))),
               sample(which(datos$CLASS == 10), size =
                        ceiling(0.8*sum(datos$CLASS==10))))

#train_idx <- sample(seq(1, nrow(datos)), ceiling(0.8*nrow(datos)))
# Hacemos el conjunto de train con estos índices
train10 <- datos[train_idx,]
# Hacemos el conjunto de test con todas aquellas variables que no tengan estos
# índices
test10 <- datos[-train_idx,]
```

Ahora vamos a quitar las variables `NSP` y `CLASS` de `train` y `test`, ya que son las salidas, y las vamos a guardar en dos vectores aparte.

```
NSP.train <- train$NSP
# Eliminamos las columnas correspondientes a las variables NSP y CLASS
train <- train[,-c(22,23)]
NSP.test <- test$NSP
test <- test[,-c(22,23)]

CLASS.train <- train10$CLASS
train10 <- train10[,-c(22,23)]
CLASS.test <- test10$CLASS
test10 <- test10[,-c(22,23)]
```

Vamos a hacer un `summary` sobre los datos de `train` para ver si podemos descartar alguna variable que a simple vista se vea que no va a aportar nada.

```
summary(train)
```

```
##        LB              AC                  FM                  UC          
##  Min.   :106.0   Min.   :0.000000   Min.   :0.000000   Min.   :0.000000  
##  1st Qu.:126.0   1st Qu.:0.000000   1st Qu.:0.000000   1st Qu.:0.000000  
##  Median :133.0   Median :0.000000   Median :0.000000   Median :0.000000  
##  Mean   :133.4   Mean   :0.002904   Mean   :0.008724   Mean   :0.004292  
##  3rd Qu.:140.0   3rd Qu.:0.010000   3rd Qu.:0.000000   3rd Qu.:0.010000  
##  Max.   :160.0   Max.   :0.020000   Max.   :0.480000   Max.   :0.010000  
##        DL                  DS          DP                   ASTV       
##  Min.   :0.000000   Min.   :0   Min.   :0.000e+00   Min.   :12.00  
##  1st Qu.:0.000000   1st Qu.:0   1st Qu.:0.000e+00   1st Qu.:32.00  
##  Median :0.000000   Median :0   Median :0.000e+00   Median :48.00  
##  Mean   :0.001593   Mean   :0   Mean   :5.879e-06   Mean   :46.73  
##  3rd Qu.:0.000000   3rd Qu.:0   3rd Qu.:0.000e+00   3rd Qu.:61.00  
##  Max.   :0.020000   Max.   :0   Max.   :1.000e-02   Max.   :86.00  
##       MSTV             ALTV              MLTV            Width       
##  Min.   :0.200   Min.   : 0.000   Min.   : 0.000   Min.   :  3.00  
##  1st Qu.:0.700   1st Qu.: 0.000   1st Qu.: 4.500   1st Qu.: 37.00  
##  Median :1.200   Median : 0.000   Median : 7.400   Median : 67.00  
##  Mean   :1.332   Mean   : 9.731   Mean   : 8.138   Mean   : 70.31  
##  3rd Qu.:1.700   3rd Qu.:11.000   3rd Qu.:10.600   3rd Qu.:100.00  
##  Max.   :7.000   Max.   :91.000   Max.   :50.700   Max.   :176.00  
##       Min              Max              Nmax            Nzeros       
##  Min.   : 50.00   Min.   :122.0   Min.   : 0.000   Min.   :0.0000  
##  1st Qu.: 67.00   1st Qu.:152.0   1st Qu.: 2.000   1st Qu.:0.0000  
##  Median : 93.00   Median :163.0   Median : 3.000   Median :0.0000  
##  Mean   : 93.75   Mean   :164.1   Mean   : 4.054   Mean   :0.3192  
##  3rd Qu.:120.00   3rd Qu.:174.0   3rd Qu.: 6.000   3rd Qu.:0.0000  
##  Max.   :158.00   Max.   :238.0   Max.   :18.000   Max.   :8.0000  
##       Mode             Mean            Median          Variance      
##  Min.   : 60.0   Min.   : 73.0   Min.   : 77.0   Min.   :  0.00  
##  1st Qu.:129.0   1st Qu.:125.0   1st Qu.:129.0   1st Qu.:  2.00  
##  Median :139.0   Median :136.0   Median :139.0   Median :  7.00  
##  Mean   :137.5   Mean   :134.7   Mean   :138.1   Mean   : 18.85  
##  3rd Qu.:148.0   3rd Qu.:145.0   3rd Qu.:148.0   3rd Qu.: 24.00  
##  Max.   :187.0   Max.   :182.0   Max.   :186.0   Max.   :269.00  
##     Tendency       
##  Min.   :-1.0000  
##  1st Qu.: 0.0000  
##  Median : 0.0000  
##  Mean   : 0.3139  
##  3rd Qu.: 1.0000  
##  Max.   : 1.0000
```

```
summary(train10)
```

```
##        LB              AC                  FM                  UC          
##  Min.   :106.0   Min.   :0.000000   Min.   :0.000000   Min.   :0.000000  
##  1st Qu.:126.0   1st Qu.:0.000000   1st Qu.:0.000000   1st Qu.:0.000000
```

```
##  Median :133.0   Median :0.000000   Median :0.000000   Median :0.000000
##  Mean   :133.1   Mean   :0.002896   Mean   :0.009683   Mean   :0.004215
##  3rd Qu.:140.0   3rd Qu.:0.010000   3rd Qu.:0.000000   3rd Qu.:0.010000
##  Max.   :159.0   Max.   :0.020000   Max.   :0.480000   Max.   :0.010000
##        DL                 DS          DP                 ASTV
##  Min.   :0.000000   Min.   :0   Min.   :0.000e+00   Min.   :12.00
##  1st Qu.:0.000000   1st Qu.:0   1st Qu.:0.000e+00   1st Qu.:32.00
##  Median :0.000000   Median :0   Median :0.000e+00   Median :48.00
##  Mean   :0.001559   Mean   :0   Mean   :5.862e-06   Mean   :46.92
##  3rd Qu.:0.000000   3rd Qu.:0   3rd Qu.:0.000e+00   3rd Qu.:61.00
##  Max.   :0.020000   Max.   :0   Max.   :1.000e-02   Max.   :87.00
##       MSTV             ALTV              MLTV            Width
##  Min.   :0.200   Min.   : 0.000   Min.   : 0.000   Min.   :  3.00
##  1st Qu.:0.700   1st Qu.: 0.000   1st Qu.: 4.600   1st Qu.: 36.00
##  Median :1.200   Median : 0.000   Median : 7.500   Median : 67.00
##  Mean   :1.337   Mean   : 9.907   Mean   : 8.302   Mean   : 70.47
##  3rd Qu.:1.700   3rd Qu.:11.000   3rd Qu.:11.000   3rd Qu.:100.00
##  Max.   :7.000   Max.   :91.000   Max.   :50.700   Max.   :180.00
##       Min              Max            Nmax            Nzeros
##  Min.   : 50.00   Min.   :122   Min.   : 0.000   Min.   :0.0000
##  1st Qu.: 66.00   1st Qu.:152   1st Qu.: 2.000   1st Qu.:0.0000
##  Median : 94.00   Median :162   Median : 4.000   Median :0.0000
##  Mean   : 93.56   Mean   :164   Mean   : 4.062   Mean   :0.3183
##  3rd Qu.:120.00   3rd Qu.:174   3rd Qu.: 6.000   3rd Qu.:0.0000
##  Max.   :159.00   Max.   :238   Max.   :18.000   Max.   :8.0000
##       Mode             Mean           Median         Variance
##  Min.   : 60.0   Min.   : 73.0   Min.   : 77.0   Min.   :  0.00
##  1st Qu.:129.0   1st Qu.:125.0   1st Qu.:128.0   1st Qu.:  2.00
##  Median :139.0   Median :136.0   Median :139.0   Median :  7.00
##  Mean   :137.2   Mean   :134.5   Mean   :137.9   Mean   : 18.91
##  3rd Qu.:148.0   3rd Qu.:145.0   3rd Qu.:148.0   3rd Qu.: 24.00
##  Max.   :187.0   Max.   :182.0   Max.   :186.0   Max.   :269.00
##     Tendency
##  Min.   :-1.0000
##  1st Qu.: 0.0000
##  Median : 0.0000
##  Mean   : 0.3095
##  3rd Qu.: 1.0000
##  Max.   : 1.0000
```

Como vemos, la variable DS tiene máximo y mínimo 0, con lo que es igual a 0 para todas las variables y por tanto no va a infuir para nuestro análisis en ninguno de los conjuntos de train. Lo que hacemos por tanto es quitarla tanto de train como de test.

```
# Quitamos la variable DS, que ocupa la sexta columna
train <- train[,-6]
test <- test[,-6]
train10 <- train10[,-6]
test10 <- test10[,-6]
```

Como hemos podido ver hay muchas que tienen valores cercanos a cero, pero sobre estos no podemos afirmar nada en claro, así que vamos a pasar a utilizar el algoritmo PCA. Para ello vamos a utilizar la función `prcomp` del paquete `stats` instalado por defecto en `R`.

El análisis de componente principales (PCA) sigue la siguiente idea: nosotros podemos tener nuestras muestras con gran multitud de características, es decir, muestras con una elevada dimensión. Ahora bien, puede darse el caso de que no todas estas características tengan la misma relevancia, es decir, que no aporten la misma información. Por motivos de eficiencia computacional y obtener un modelo más sencillo es claro que es bueno reducir este número de variables, aunque sea a costa de perder algo de información, y esto es lo que hace el PCA. Buscamos una representación de los datos de dimensión más baja que capture toda (o una cantidad considerable) de la información. Un dimensión es interesante en términos de cómo las observaciones varían a lo largo de dicha dimensión.

Si nuestras muestras tienen el siguiente conjunto de características $X_1, X_2, ..., X_p$ entonces la primera componente principal será $Z_1 = \phi_1 X_1 + ... + \phi_p X_p$ y estamos buscando por tanto una combinación lineal de las características anteriores que maximize la varianza, cumpliendo que el cuadrado de los coeficientes (*loadings*) sumen uno, es decir, que maximice $\frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{p}\phi_{j1}x_{ij}\right)^2$ (estamos sumoniendo que las variables tienen media cero para estos cálculos). Señalar que la condición de normalización que le imponemos a los *loadings* es para que no ganemos máxima varianza simplemente haciéndolos crecer mucho. Para calcular el resto de componentes principales se hace de la misma manera restringiendo además a que no estén correlados con los anteriores: que sean ortogonales (si vemos los *loadings* como vectores) a los anteriores.

Ya hemos asumido antes que la media de cada variable es cero ya que sólo estamos interesados en la varianza y esto facilita los cálculos. Por otro lado, si no tenemos ninguna restricción que lo impida, es conveniente escalar las variables de modo que no tenga una más influencia que otra simplemente por la escala en la que está medida, por ejemplo si una variable da saltos de 1000 en 1000 y otra de 10 en 10, aunque los datos de la primera presenten menos varianza debido a la magnitud de los datos ésta tendrá más influencia.

Por tanto, vamos a utilizar la función `prcomp()` diciéndole que queremos que la media de las variables sea cero con el parámetro `center = TRUE` y que escale las variables con el parámetro `scale = TRUE`.

```
pca.out <- prcomp(train, center = TRUE, scale = TRUE)
pca.out10 <- prcomp(train10, center = TRUE, scale = TRUE)
```

Veamos qué nos ha devuelto el PCA sobre 3 variables

```
summary(pca.out)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4     PC5     PC6
## Standard deviation     2.3844 1.8157 1.28738 1.20024 1.12886 1.00605
## Proportion of Variance 0.2843 0.1648 0.08287 0.07203 0.06372 0.05061
## Cumulative Proportion  0.2843 0.4491 0.53197 0.60400 0.66772 0.71832
##                           PC7     PC8     PC9    PC10    PC11    PC12
## Standard deviation     0.99106 0.91783 0.86660 0.83583 0.70591 0.69738
## Proportion of Variance 0.04911 0.04212 0.03755 0.03493 0.02492 0.02432
## Cumulative Proportion  0.76743 0.80955 0.84710 0.88204 0.90695 0.93127
##                           PC13    PC14    PC15    PC16    PC17    PC18
## Standard deviation     0.61533 0.57288 0.52038 0.41818 0.35897 0.25862
## Proportion of Variance 0.01893 0.01641 0.01354 0.00874 0.00644 0.00334
## Cumulative Proportion  0.95020 0.96661 0.98015 0.98889 0.99534 0.99868
##                           PC19      PC20
## Standard deviation     0.16245 1.284e-15
## Proportion of Variance 0.00132 0.000e+00
## Cumulative Proportion  1.00000 1.000e+00
```

Como podemos ver con `summary()`, con las 14 primeras componentes principales estamos explicando un 96% de los datos, y son con las que nos vamos a quedar para hacer el estudio reducido y ver si hay mejora al utilizar PCA. Vamos a ver también el PCA para 10 clases:

```r
summary(pca.out10)
```

```
## Importance of components:
##                            PC1    PC2     PC3     PC4     PC5     PC6
## Standard deviation     2.3798 1.8335 1.28944 1.18724 1.12364 1.00316
## Proportion of Variance 0.2832 0.1681 0.08313 0.07048 0.06313 0.05032
## Cumulative Proportion  0.2832 0.4512 0.53439 0.60486 0.66799 0.71831
##                            PC7     PC8     PC9    PC10    PC11    PC12
## Standard deviation     0.99050 0.91916 0.86618 0.84038 0.70866 0.70131
## Proportion of Variance 0.04905 0.04224 0.03751 0.03531 0.02511 0.02459
## Cumulative Proportion  0.76736 0.80960 0.84712 0.88243 0.90754 0.93213
##                           PC13    PC14    PC15   PC16    PC17   PC18
## Standard deviation     0.62263 0.55602 0.50954 0.4219 0.35681 0.2607
## Proportion of Variance 0.01938 0.01546 0.01298 0.0089 0.00637 0.0034
## Cumulative Proportion  0.95151 0.96697 0.97995 0.9889 0.99522 0.9986
##                           PC19      PC20
## Standard deviation     0.16630 7.091e-16
## Proportion of Variance 0.00138 0.000e+00
## Cumulative Proportion  1.00000 1.000e+00
```

De nuevo con las 14 primeras componentes principales conseguimos explicar más del 96% de los datos, con lo que vamos a quedarnos también con las 14 primeras. Vamos a hacer entonces la combinación lineal que nos da PCA para ambos conjuntos para obtener los nuevos conjuntos de train.

```r
# Recorremos las combinaciones lineales devueltas por prcomp y las aplicamos
# al conjunto de train.
trainPCA <- apply(pca.out$rotation[, 1:14], 2, function(x) {
  apply(train, 1, function(y) {
    x%*%y
  })
})

trainPCA10 <- apply(pca.out10$rotation[ ,1:14], 2, function(x) {
  apply(train10, 1, function(y) {
    x%*%y
  })
})
```

Vamos a hacerle la combinación lineal al conjunto de test también con las componentes principales de train:

```r
testPCA <- apply(pca.out$rotation[ ,1:14], 2, function(x) {
  apply(test, 1, function(y) {
    x%*%y
  })
})


testPCA10 <- apply(pca.out10$rotation[ ,1:14], 2, function(x) {
  apply(test10, 1, function(y) {
    x%*%y
  })
})
```

Lo que vamos a hacer es realizar el estudio a partir de aquí tanto con PCA como sin PCA, ya que no tenemos una dimensión excesivamente grande como para no poder trabajar con ella, de forma que vamos a ver si en cada caso merece la pena utilizar PCA o no en términos de la cantidad de información que se pierda (que se traducirá en una peor predicción con los datos).

4. Normalización de las variables (en su caso).

En el caso de modelos de aprendizaje que trabajan por similaridad, es decir, por distancia entre las muestras para asignar una clase a un dato (como son el KNN y las funciones de base radial) es conveniente normalizar las características de modo que evitemos que unas tengan más peso que otras en las decisiones del modelo, debido a las diferencias de magnitud.

Entonces vamos a normalizar las variables antes de aplicar ningún modelo de aprendizaje para así trabajar con los mismos datos. Vamos a proceder a normalizar los datos, para lo que, igual que hemos hecho en prácticas anteriores, vamos a normalizar los datos de train y empleando los factores de normalización de dicho proceso normalizaremos los de test. De este modo no vamos a contaminar los datos de entrenamiento con información sobre los de test, asegurando que el proceso de aprendizaje sea adecuado.

```
# Escalamos el conjunto de train
train <- scale(train)
medias <- attr(train, "scaled:center")
escalados <- attr(train, "scaled:scale")
# Escalamos el conjunto de test con los centros y las escalas del de train
test <- scale(test, medias, escalados)

trainPCA <- scale(trainPCA)
medias <- attr(trainPCA, "scaled:center")
escalados <- attr(trainPCA, "scaled:scale")
testPCA <- scale(testPCA, medias, escalados)
```

```
train10 <- scale(train10)
medias <- attr(train10, "scaled:center")
escalados <- attr(train10, "scaled:scale")
test10 <- scale(test10, medias, escalados)

trainPCA10 <- scale(trainPCA10)
medias <- attr(trainPCA10, "scaled:center")
escalados <- attr(trainPCA10, "scaled:scale")
testPCA10 <- scale(testPCA10, medias, escalados)
```

5. Selección de las técnicas y valoración de la idoneidad de las mismas frente a otras alternativas.

5.1. Selección de técnicas paramétricas

En cuanto a los modelos paramétricos, es claro que no podemos plantearnos usar el perceptron ya que de inicio estamos ante un problema de clasifación no binaria, con lo que no vamos a poder realizar una buena clasificación con él.

Como podemos leer en el libro ISLR, regresión logística y SVM tienen funciones de pérdida muy parecidas, en consecuencia los rendimientos que ofrecen son muy parecidos. Ahora bien, lo que distingue a ambos en su

comportamiento es el solapamiento que haya entre las distintas clases de la muestra. Así cuando tenemos un solamiento mayor la regresión logística muestra, experimentalmente, un mejor rendimiento que el SVM. En cambio cuando los datos son separables es el SVM el que se comporta mejor.

Podríamos pensar en que SVM cuenta con la potencia que le aportan los núcleos, no obstante si el solapamiento entre las clases es considerable ningún núcleo será el adecuado, es decir, no tendríamos seguridad de que SVM funcione mejor.

Por tanto para decidirnos vamos a medir el solapamiento entre las distintas clases. Esto lo haremos mediante la métrica CSM que se basa en realizar un cociente entre las distancias de los puntos de cada clase al punto medio de su clase con las distancias entre los puntos medios de las clases al punto medio de la muestra total. Así si las primeras son muy grandes el índice, J (que es este cociente), será pequeño indicando que hay solapamiento entre las clases, ya que por decirlo de alguna manera las clases están más dispersas que la muestra, indicando que efectivamente han de estar entremezcladas. Haciendo varios experimentos nos hemos dado cuenta de que si las clases están separadas pero juntas este índice el 0.5. Si las clases están separadas y además hay separación entre ellas J será mayor que 0.5 (y mayor cuanta más separación haya) y si las clases están solapadas J será menor que 0.5. Este estudio lo hemos sacado gracias a los enlaces [4] y [5].

A continuación mostramos el código de la función que nos medirá este índice de solapamiento para una muestra dada:

```
classSepMeasure <- function(trainp, clases, numClases, n) {
  train <- as.matrix(trainp)
  m <- apply(train, 2, sum)/nrow(train)
  mis <- sapply(1:numClases, function(i) {
    datosi <- train[clases==i, ]
    apply(datosi, 2, sum)/nrow(datosi)
  })

  Sw <- matrix(0, n, n)
  for (i in 1:numClases) {
    matriz <- matrix(0, n, n)
    datosi <- train[clases==i, ]
    for(j in 1:nrow(datosi)) {
      matriz_j <- (datosi[j,] - mis[,i])%*%t(datosi[j,] - mis[,i])
      matriz <- matriz + matriz_j
    }
    Sw <- Sw + matriz
  }

  Sb <- matrix(0, n, n)
  for (i in 1:numClases) {
    ni <- sum(clases == i)
    Mi <- ni*((mis[,i] - m)%*%t(mis[,i] - m))
    Sb <- Sb + Mi
  }

  J <- sum(diag(Sb))/sum(diag(Sw))

  return(J)
}
```

```
classSepMeasure(train10, CLASS.train, 10, 20)
```

```
## [1] 0.446033
```

```
classSepMeasure(trainPCA10, CLASS.train, 10, 14)
```

```
## [1] 0.5091602
```

```
classSepMeasure(train, NSP.train, 3, 20)
```

```
## [1] 0.1026909
```

```
classSepMeasure(trainPCA, NSP.train, 3, 14)
```

```
## [1] 0.1307058
```

Como vemos, este índice es menor que 0.5 en todos los casos, pero es que además cuando tenemos en cuenta 3 clases en lugar de 10 (`NSP`) este índice es un poco mayor que 0.1 sin PCA y 0.05 con PCA, con lo que las clases están muy solapadas y por tanto nos vamos a decantar por la regresión logística en ambos casos, tanto con 10 clases como con 3.

Por otro lado las funciones de base radial paramétricas no nos parecen una buena opción ya que dependen fuertemente de cómo estén distribuidos los datos, es decir, les ocurre como al KNN; si tenemos puntos próximos al punto a etiquetar de clase distinta a la real del punto entonces la clasificación no será buena. En cambio pensamos que dado que la regresión logística nos da una visión probabilística del etiquetado será más robusta a estas situaciones.

A continuación vamos a explicar cómo funciona la regresion logística multinomial. Lo que hace es basarse en la regresión logística binaria utilizando el *uno contra todos* o *one versus all*, de forma que para cada clase se divide el conjunto entre las instancias que pertenecen a mi clase y todas las demás y en estas dos clases artificiales se utiliza la regresión logística binaria. De esta forma cuando nos llega un nuevo dato se calcula la probabilidad de que pertenezca a cada clase (es decir, para cada clase se calcula la probabilidad de que pertenezca a esa clase en lugar de a todas las demás) y nos quedamos con el máximo de estas probabilidades, obteniendo así la clase de dato con una cierta probabilidad. En cuanto a la regresión binomial, para obtener la probabilidad de que un dato pertenezca a una determinada clase se utiliza la función logística, que devuelve un número entre 0 y 1 y es la siguiente: $p(X) = \frac{e^{\beta_0+\beta_1 X_1+\ldots+\beta_n X_n}}{1+e^{\beta_0+\beta_1 X_1+\ldots+\beta_n X_n}}$ suponiendo que haya n variables con las que predecir.

Ahora, tenemos que ajustar los coeficientes $\beta_0, \beta_1, ..., \beta_n$ y la forma en que lo hace el paquete `glmnet`, que es el que vamos a utilizar para la regresión binomial, es utilizando el método de máxima verosimilitud. La idea básica de este método es buscar estimaciones para cada β_i de forma que la probabilidad predicha para cada individuo sea tan parecida como sea posible a su etiqueta real, teniendo en cuenta que toma las etiquetas o 0 o 1, de forma que se le da una probabilidad lo más cerca a 0 si la etiqueta es 0 y lo más cercana posible a 1 si la etiqueta es 1, es decir, se intenta maximizar la función de verosimilitud: $\prod_{i:y_i=1} p(x_i) \prod_{i':y'_i=0}(1-p(x'_i))$.

Aunque el paquete `glmnet` permite hacer regresión multinomial con el parámetro `family`, hemos tenido problemas a la hora de clasificar con 10 clases, ya que nos devolvía siempre la misma clase para todos los datos a predecir. Tras hacer un pequeño estudio para descubrir si el problema era que las etiquetas no tenían correlación realmente con los datos o si era problema de la función considerada, que desarrollamos a continuación, nos dimos cuenta de que no era problema de los datos y lo que hemos hecho es programar nosotros la regresión multinomial del modo que se ha explicado anteriormente, y esta ha sido la que hemos utilizado ya tanto para 3 clases como para 10.

El estudio que hemos hecho ha sido el siguiente: para cada clase, hacer un problema binario cogiendo todas las instancias de train de esa clase y una muestra del mismo tamaño de las demás, y pasarle después el resto de muestras de train como datos de test, esperando que para todas ellas devolviera que no pertenecen a la clase considerada:

```r
for (i in 1:10) {
  set.seed(1)
  clase = i
  # Tomamos los datos de la clase i
  train_una_clase = train10[CLASS.train == clase,]
  n_datos_clase = nrow(train_una_clase)
  # Tomamos una muestra del resto del mismo tamaño
  train_idx <- sample(which(CLASS.train != clase), n_datos_clase)
  train.resto = train10[train_idx,]
  # Juntamos la clase considerada con la muestra de las demás
  train.OVA = rbind(train_una_clase, train.resto)
  # Asignamos como etiquetas 1 si pertenece a la clase considerada y 0 si no.
  train.CLASS.OVA = c(rep(1, n_datos_clase), rep(0, n_datos_clase))

  # Consideramos como test el resto de datos de train
  test.OVA <- train10[-c(train_idx, which(CLASS.train == clase)), ]

  # Hacemos cross validation para el problema binomial
  cv.fit <- cv.glmnet(as.matrix(train.OVA), train.CLASS.OVA, alpha = 0,
                      family = "binomial", type.measure = "mse")
  lambda <- cv.fit$lambda.min
  # Entrenamos un modelo binomial
  modelo <- glmnet(as.matrix(train.OVA), train.CLASS.OVA, alpha = 0,
                   lambda = lambda, family="binomial")
  predicciones <- predict(modelo, s=lambda, newx = test.OVA, type = "response")
  data <- as.matrix(data.frame(predicciones))

  clases.predichas = rep(0, nrow(test.OVA))
  clases.predichas[predicciones > .5] = 1

  # Obtenemos el porcentaje de acierto
  cat("Porcentaje de acierto para la clase", i, ":",
      100*sum(clases.predichas == 0)/nrow(test.OVA), "\n")
}
```

```
## Porcentaje de acierto para la clase 1 : 69.72477
## Porcentaje de acierto para la clase 2 : 76.86375
## Porcentaje de acierto para la clase 3 : 81.17284
## Porcentaje de acierto para la clase 4 : 94.86041
## Porcentaje de acierto para la clase 5 : 82.26415
## Porcentaje de acierto para la clase 6 : 74.1908
## Porcentaje de acierto para la clase 7 : 85.63748
## Porcentaje de acierto para la clase 8 : 96.08866
## Porcentaje de acierto para la clase 9 : 89.58595
## Porcentaje de acierto para la clase 10 : 83.30935
```

Como vemos devuelve cosas lógicas en tanto que los porcentajes de acierto son relativamente altos, con lo que el problema no es que las etiquetas no tengan relación con los datos.

5.2. Selección de técnicas no paramétricas

Para elegir entre KNN y RBF (funciones de base radial) vamos a volver a utilizar el índice J que habíamos calculado previamente para elegir entre regresión logística y SVM, ya que aquí también nos influye cómo

de solapadas estén las clases, ya que si no hay solapamiento convendría más utilizar KNN ya que ahí los K vecinos más cercanos influyen lo mismo para elegir la clase, mientras que si hay solapamiento conviene más utilizar funciones de base radial para que influyan menos los valores que están más lejanos, ya que al tenerlos mezclados en la frontera, a un punto lo suficientemente cercano a la frontera le viene mejor que estos influyan menos a que los coja por igual el KNN. Además, al estar las clases solapadas usando KNN corremos el riesgo de que los K vecinos más cercanos no sean (en mayoría) de la clase del punto a clasificar. Este problema lo evitamos empleando base radial puesto que en ella todos los puntos tienen influencía, así por ejemplo pueden vencer puntos que estén algo más alejados del punto en cuestión pero que sean de la clase correcta.

Además de utilizar este índice, vamos a calcular la distancia media entre todos los puntos de una clase y la distancia media entre todos los puntos de la muestra, de forma que si la distancia media de una clase con respecto a la distancia media de la muestra total es igual o mayor, esto podría decirnos que hay solapamiento, mientras que si es menor, que no lo hay. Lo vamos a hacer tanto para NSP, donde tenemos 3 clases, como para CLASS, donde tenemos 10.

```
getDistanciasMedias <- function(clases, train, numClases) {
  distanciasMedias <- sapply(1:numClases, function(i) {
    if (is.null(clases)) {
      data <- as.matrix(train)
    }
    else {
      data <- which(clases == i)
      data <- as.matrix(train[data,])
    }
    distancia <- rdist(data)
    distMedia <- sum(distancia)/
      (nrow(distancia)*ncol(distancia)- nrow(distancia))
    distMedia
  })
  return(distanciasMedias)
}
```

```
cat("Distancias medias de las 3 clases:",
      getDistanciasMedias(NSP.train, train, 3))
```

```
## Distancias medias de las 3 clases: 5.442215 4.649694 7.938131
```

```
cat("Distancia media de la muestra completa para 3 clases:",
    getDistanciasMedias(NULL, train, 1))
```

```
## Distancia media de la muestra completa para 3 clases: 5.872131
```

```
cat("Distancias medias de las 10 clases: ",
    getDistanciasMedias(CLASS.train, train10, 10))
```

```
## Distancias medias de las 10 clases:  4.153753 4.748245 3.820817 4.942507 3.948518 5.467317 5.564578 (
```

```
cat("Distancia media de la muestra completa para 10 clases:",
    getDistanciasMedias(NULL, train10, 1))
```

```
## Distancia media de la muestra completa para 10 clases: 5.868203
```

```r
cat("Distancias medias de las 3 clases con PCA:",
      getDistanciasMedias(NSP.train, trainPCA, 3))
```

```
## Distancias medias de las 3 clases con PCA: 4.31293 4.450205 6.662785
```

```r
cat("Distancia media de la muestra completa para 3 clases con PCA:",
    getDistanciasMedias(NULL, trainPCA, 1))
```

```
## Distancia media de la muestra completa para 3 clases con PCA: 4.852614
```

```r
cat("Distancias medias de las 10 clases con PCA: ",
    getDistanciasMedias(CLASS.train, trainPCA10, 10))
```

```
## Distancias medias de las 10 clases con PCA:  3.563906 3.783283 3.099307 4.447984 3.770173 3.720392 4
```

```r
cat("Distancia media de la muestra completa para 10 clases con PCA:",
    getDistanciasMedias(NULL, trainPCA10, 1))
```

```
## Distancia media de la muestra completa para 10 clases con PCA: 4.844126
```

Efectivamente nos sale algo parecido a lo que deducíamos del índice J, ya que para las 3 clases (tanto con PCA como sin él), las medias de las clases están muy cerca o por encima de la media de la muestra total, mientras que para 10 clases (tanto con PCA como sin él), las medias están cerca de la media total de la muestra, con lo que tenemos más solapamiento con 3 clases que con 10 pero en ningún caso están completamente separadas.

Vamos a utilizar por tanto funciones de base radial. A continuación vemos la implementación de las mismas, que hemos llevado a cabo nosotros.

Vamos a hacer primero una función para hacer la clasificación con las funciones de base radial, donde en lugar de hacer la media de las distancias con las etiquetas, lo que vamos a hacer es sumar pesos que irán en función de r y el punto de test en cuestión. Donde r es la anchura que le vamos a dar a las gaussianas, que va a ser un valor que vamos a obtener por validación cruzada más adelante.

6. Aplicación de las técnicas especificando claramente qué algoritmos se usan en la estimación de los parámetros, los hiperparámetros y el error de generalización.

Empezamos haciendo el análisis para las 3 clases, es decir, según la variable `NSP`, y además empezamos haciéndolo con la transformación que nos ha dado PCA.

Para ello, vamos a utilizar regresión logística con regularización weigth-decay con el paquete `glmnet`. Como sabemos, hay un coeficiente, λ, que regula la influencia que tiene la condición de regularización (de "contracción") en el ajuste. Para elegir dicho λ vamos a hacer validación cruzada con el mismo paquete haciendo uso de la función `cv.glmnet`, que nos devuelve justo el mejor λ encontrado entre una rejilla. Esta validación cruzada la vamos a hacer con cinco particiones especificándolo con la variable `nfolds = 5` indicando que la regularización que queremos hacer es weight-decay (que se especifica con el parámetro `alpha = 0`). El porqué hemos elegido esta regularización está explicado en la sección dedicada para ello, la número 7. Vamos a hacer que elija como λ aquel que minimice la media de los errores cuadráticos, para lo que tenemos que poner como argumento a la función `type.measure = "mse"`.

Como hemos comentado, vamos a programar la regresión logística multinomial haciendo uso de `glmnet` para la binomial:

```
generarOVA <- function(train, clase, clases) {
  # Asignamos como etiquetas 1 si pertenece a la clase considerada y 0 si no.
  train.CLASS.OVA <- clases
  train.CLASS.OVA[clases == clase] <- 1
  train.CLASS.OVA[clases != clase] <- 0

  # Hacemos cross validation para el problema binomial
  cv.fit <- cv.glmnet(as.matrix(train), train.CLASS.OVA, alpha = 0,
                      family = "binomial", type.measure = "mse")
  lambda <- cv.fit$lambda.min
  # Entrenamos un modelo binomial
  modelo <- glmnet(as.matrix(train), train.CLASS.OVA, alpha = 0,
                   lambda = lambda, family="binomial")
  return(list(modelo, lambda))
}

predictOVA <- function(train, clases, numClases, test) {
  prediccionesOVA <- matrix(0, nrow(test), numClases)
  for (i in 1:numClases) {
    Mi <- generarOVA(train, i, clases)
    modeloi <- Mi[[1]]
    lambdai <- Mi[[2]]
    prediccionesOVA[,i] <- predict(modeloi, s=lambdai, newx = test,
                                   type = "response")
  }

  clases.predichas <- apply(prediccionesOVA, 1, which.max)
  return(clases.predichas)
}
```

Lo utilizamos con 3 clases y PCA:

```
# Fijamos una semilla
prediccionesPCA3RL <- predictOVA(trainPCA, NSP.train, 3, testPCA)
cat("Porcentaje de acierto con 3 clases y PCA:",
    100*sum(prediccionesPCA3RL == NSP.test)/length(NSP.test))
```

```
## Porcentaje de acierto con 3 clases y PCA: 85.88235
```

Obtenemos un porcentaje de acierto de 85.88, lo que no está mal teniendo en cuenta que las clases están solapadas y es un modelo paramétrico.

Vamos a hacerlo ahora sin PCA a ver la diferencia que hay con PCA, a ver si merece la pena bajar las dimensiones porque no se pierde mucha información o por el contrario no merece la pena porque sale una predicción mucho mejor con todas las variables (en nuestro caso que tampoco tenemos muchas variables y podemos permitirnos hacer este estudio).

```
predicciones3RL <- predictOVA(train, NSP.train, 3, test)
cat("Porcentaje de acierto con 3 clases sin PCA:",
    100*sum(predicciones3RL == NSP.test)/length(NSP.test))
```

```
## Porcentaje de acierto con 3 clases sin PCA: 88.70588
```

Como vemos el porcentaje de acierto es similar: es un 4% mayor. Sin embargo, en el problema que nos ocupa, quizás sí sea una diferencia significativa. Aunque en realidad teniendo en cuenta el porcentaje de acierto, que tampoco es muy alto, quizás habría que centrarse primero en mejorarlo cambiando de técnica (mejorando con técnicas no paramétricas como vamos a hacer a continuación) que preocuparse por ese 4%.

Vamos ahora a enfocar el problema desde las 10 clases. Lo hacemos primero con PCA y después sin PCA:

```
prediccionesPCA10RL <- predictOVA(trainPCA10, CLASS.train, 10, testPCA10)
cat("Porcentaje de acierto con 10 clases y PCA:",
    100*sum(prediccionesPCA10RL == CLASS.test)/length(CLASS.test))
```

```
## Porcentaje de acierto con 10 clases y PCA: 54.28571
```

```
prediccones10RL <- predictOVA(train10, CLASS.train, 10, test10)
cat("Porcentaje de acierto con 10 clases sin PCA:",
    100*sum(prediccones10RL == CLASS.test)/length(CLASS.test))
```

```
## Porcentaje de acierto con 10 clases sin PCA: 65.2381
```

En este caso los porcentajes de acierto son más bajos y sí hay más de un 10% de diferencia entre utilizar PCA y no utilizarlo, con lo que descartamos para 10 clases usar PCA en el caso de paramétricos.

6.2 Modelo no paramétrico:

Como hemos dicho en la sección 5, nos decantamos por utilizar las funciones de base radial. En este caso no hemos encontrado un paquete de `R` que las tuviera implementadas, y como además las tenemos que modificar para adaptarlas a clasificación, hemos optado por programarlas nosotros.

Lo que hacemos es construir primero una función que nos devuelva una gaussiana normalizada para d dimensiones y utilizarla para *colocar* una en cada dato de train. Cada gaussiana tendrá una anchura `r` que elegiremos luego por validación cruzada. Lo que hacemos es calcular para cada dato de test la gaussiana de la distancia entre dicho dato de test y todos los de train, de forma que obtenemos unos pesos para cada dato de train. Posteriormente sumamos los pesos por etiquetas (todos los pesos para los datos de train que tengan clase 1, para los de clase 2, etc.) y nos quedamos con la etiqueta que tenga mayor suma (normalizando por el total de pesos), de forma que estamos haciendo clasificación. Si se da el caso de que las sumas de los pesos es 0, no clasificamos dicho dato, asignándole la etiqueta 0.

```
#FUNCIONES DE BASE RADIAL

#Función que devuelve un núcleo Gaussiano normalizado para R^d
fiD <- function(d){
  function(z) {exp(-0.5 * z^2)/((2*pi)^(-d/2))}
}

# Función para calcular la distancia entre dos puntos
distancia <- function(x, y) {
  sqrt(sum((x-y)^2))
}

RBF <- function(x, datos.train, et.train, numClases, r) {
  # Calculamos la gaussiana
  fi <- fiD(ncol(datos.train))
```

```
  # Calculamos los pesos
  alfas <- apply(datos.train, 1, function(y) { fi(distancia(x,y)/r) } )
  suma_total  <- sum(alfas)
  # Sumamos por clases
  sumas_parciales <- sapply(1:numClases, function(i) {
    sum(alfas[et.train == i])
  })
  # Nos quedamos con el máximo en caso de que lo haya
  if(suma_total != 0) {
    sumas_parciales <- sumas_parciales/suma_total
    max <- which.max(sumas_parciales)
  }
  else {
    max <- 0
  }

  return(max)
}
```

A continuación creamos una función para predecir las clases para un conjunto completo de test, llamando a la función anterior una vez por cada dato de test:

```
predictEt <- function(datos.train, et.train, datos.test, numClases, r) {
  etiquetas <- sapply(1:nrow(datos.test), function(i) {
    RBF(datos.test[i,], datos.train, et.train, numClases, r)
  })
  return(etiquetas)
}
```

Como hemos comentado antes, vamos a hacer validación cruzada para elegir el mejor r de entre un rango de valores que se le va a pasar a la función por parámetros, que vamos a programar nosostros también. Lo que hace esta función es dividir el conjunto de train en 5 subconjuntos con los que se hará la validación cruzada (de forma equilibrada) y, para cada r, hacer la media del porcentaje de acierto obtenido para cada partición. Después, devuelve la r con la que se ha obtenido la mejor media.

```
cv.RBF <- function(datos.train, et.train, numClases, rango_r) {
  # Realizamos las particiones de forma equilibrada
  k = 5
  folds = rep(1, nrow(datos.train))
  for (i in 1:numClases) {
    folds[et.train == i] <- c(sample(rep(seq(k), floor(sum(et.train == i)/k))),
                              rep(1, sum(et.train == i) -
                                    k*floor(sum(et.train == i)/k)))
  }

  media_aciertos <- vector("numeric", length(rango_r))

  for(j in 1:length(rango_r)) {
    aciertos <- vector("numeric", k)
    for(i in 1:k) {
      etiq <- predictEt(datos.train[folds != i, ], et.train[folds != i],
                datos.train[folds == i, ], numClases, rango_r[j])
      aciertos[i] <- sum(etiq == et.train[folds == i])/length(etiq)
```

```
    }
    media_aciertos[j] <- mean(aciertos)
  }

  return(rango_r[which.max(media_aciertos)])

}
```

Hacemos ahora, para 3 clases y para 10, con PCA y sin PCA, las funciones de base radial con previa validación cruzada para r.

```
# Fijamos primero la semilla
set.seed(1)
best_r <- cv.RBF(train, NSP.train, 3, seq(0.1, 1.5, 0.1))
predicciones3RBF <- predictEt(train, NSP.train, test, 3, best_r)
acierto <- sum(predicciones3RBF== NSP.test)/
  length(predicciones3RBF[predicciones3RBF != 0])
no_clasificados <- length(predicciones3RBF[predicciones3RBF == 0])/
  length(predicciones3RBF)
cat("El porcentaje de acierto para 3 clases sin PCA ha sido:", 100*acierto)
```

```
## El porcentaje de acierto para 3 clases sin PCA ha sido: 91.76471
```

```
cat("El porcentaje de no clasificados ha sido:", 100*no_clasificados)
```

```
## El porcentaje de no clasificados ha sido: 0
```

```
best_r <- cv.RBF(trainPCA, NSP.train, 3, seq(0.1, 1.5, 0.1))
prediccionesPCA3RBF <- predictEt(trainPCA, NSP.train, testPCA, 3, best_r)
acierto <- sum(prediccionesPCA3RBF == NSP.test)/
  length(prediccionesPCA3RBF[prediccionesPCA3RBF != 0])
no_clasificados <- length(prediccionesPCA3RBF[prediccionesPCA3RBF == 0])/
  length(prediccionesPCA3RBF)
cat("El porcentaje de acierto para 3 clases con PCA ha sido:", 100*acierto)
```

```
## El porcentaje de acierto para 3 clases con PCA ha sido: 90.35294
```

```
cat("El porcentaje de no clasificados ha sido:", 100*no_clasificados)
```

```
## El porcentaje de no clasificados ha sido: 0
```

```
best_r <- cv.RBF(trainPCA10, CLASS.train, 10, seq(0.1, 1.5, 0.1))
prediccionesPCA10RBF <- predictEt(trainPCA10, CLASS.train, testPCA10, 10, best_r)
acierto <- sum(prediccionesPCA10RBF == CLASS.test)/
  length(prediccionesPCA10RBF[prediccionesPCA10RBF != 0])
no_clasificados <- length(prediccionesPCA10RBF[prediccionesPCA10RBF == 0])/
  length(prediccionesPCA10RBF)
cat("El porcentaje de acierto para 10 clases con PCA ha sido:", 100*acierto)
```

```
## El porcentaje de acierto para 10 clases con PCA ha sido: 67.61905
```

```r
cat("El porcentaje de no clasificados ha sido:", 100*no_clasificados)
```

```
## El porcentaje de no clasificados ha sido: 0
```

```r
best_r <- cv.RBF(train10, CLASS.train, 10, seq(0.1, 1.5, 0.1))
predicciones10RBF <- predictEt(train10, CLASS.train, test10, 10, best_r)
acierto <- sum(predicciones10RBF == CLASS.test)/
  length(predicciones10RBF[predicciones10RBF != 0])
no_clasificados <- length(predicciones10RBF[predicciones10RBF == 0])/
  length(predicciones10RBF)
cat("El porcentaje de acierto para 10 clases sin PCA ha sido:", 100*acierto)
```

```
## El porcentaje de acierto para 10 clases sin PCA ha sido: 69.28571
```

```r
cat("El porcentaje de no clasificados ha sido:", 100*no_clasificados)
```

```
## El porcentaje de no clasificados ha sido: 0
```

Como vemos, tenemos resultados mejores que con la técnica paramétrica tanto para 3 clases como para 10, y la diferencia entre PCA y sin PCA es mucho menos notable. Concretamente, para 3 clases logramos pasar el 90% de acierto, lo que consideramos que es un muy buen porcentaje.

7. Argumentar sobre la idoneidad de la función regularización usada (en su caso).

La regularización que hemos empleado es la ridge regression. Nosotros hemos visto dos formas principales de regresión: LASSO y ridge regression. La diferencia entre ambas es su condición de regularización. La primer minimiza la suma de los valores absolutos de los coeficientes de regresión mientras que la segunda minimiza la suma de sus cuadrados. Esta diferencia deriva en que mientras que LASSO puede dar lugar a emplear coeficientes nulos, descartando variables (se realiza una selección automática de características) la ridge regression puede dar coeficientes muy próximos a cero pero no nulos, es decir, no realiza selección de características.

Nos hemos decantado por ridge regression dado que ya hacemos una especie de selección de características con PCA quedándonos con aquellos componentes principales que recogen algo más del 95% de la información y no queremos sacrificar más información que esta al usar el LASSO, además hemos hecho una comparativa con los resultados obtenidos usando PCA y sin usarlo con lo que para ambos casos empleamos el mismo tipo de regularizacion.

8. Valoración de los resultados (gráficas, métricas de error, análisis de residuos, etc.)

```r
getMatrizConfusion <- function(et.predichas, et.reales, numClases) {
  M <- matrix(0, numClases, numClases)
  for (i in seq(numClases)) {
    for (j in seq(numClases)) {
```

```
      M[i,j] = sum(et.predichas == i & et.reales == j)
    }
  }

  M
}
```

Vamos a generar las matrices de confusión para los distintos experimentos que hemos realizado:

```
M3RL <- getMatrizConfusion(predicciones3RL, NSP.test, 3)
MPCA3RL <- getMatrizConfusion(prediccionesPCA3RL, NSP.test, 3)
M10RL <- getMatrizConfusion(predicciones10RL, CLASS.test, 10)
MPCA10RL <- getMatrizConfusion(prediccionesPCA10RL, CLASS.test, 10)
```

9. Justificar que se ha obtenido la mejor de las posibles soluciones con la técnica elegida y la muestra dada. Argumentar en términos de la dimensión VC del modelo, el error de generalización y las curvas de aprendizaje.

10. Comparativa con otros resultados sobre la misma base de datos

Hemos encontrado diversos documentos donde se muestran experimentos con distintas técnicas y enfoques sobre nuestra base de datos así que a continuación vamos a mostrar tales resultados que compararemos con los resultados.

En un primer documento se realiza una simplificación del problema puesto que pasa de los tres valores que puede tomar NSP a sólo dos pasando a un problema de clasificación binaria que son los que hemos tratado en esta asignatura. Así agrupa en la clase *No-normal* a aquellas muestras etiquetadas como sospechosas o anormal. Así tras esta agrupación y empleando kNN obtiene la siguiente matriz de confusión (referencia [9]):

Matriz de confusión		Clase predicha	
		Normal	No-normal
Clase	Normal	12413	4137
verdadera	No-normal	9202	37898

En el siguiente documento que hemos consultado (referencia [10]) se muestra un estudio de cómo se comportan distintos algoritmos empleando técnicas de AdaBoost y sin él, a continuacion mostramos las dos tablas, primero sin AdaBoost y después con él:

Algoritmo	MAE	Kappa	Acc. (%)
Redes de funciones de base radial (param.)	0.123	0.642	85.983
SVM	0.250	0.674	88.758
Redes neuronales	0.058	0.784	92.098
Árboles de decisión	0.059	0.793	92.457

Algoritmo	MAE	Kappa	Acc. (%)
Redes de funciones de base radial (param.)	0.103	0.668	87.676
SVM	0.098	0.673	88.664
Redes neuronales	0.069	0.783	92.051
Árboles de decisión	0.034	0.861	95.014

Tenemos también la matriz de confusión para los árboles de decisión:

	Predichos		
Clase Real	Normal	Sospechoso	Patológico
Normal	1622	26	7
Sospechoso	55	236	4
Patológico	7	7	162

las metricas que se emplean en estas tables son:

MAE = $\frac{sum_{i=1}^{N}|y_i - p_i|}{n}$ donde p_i es el valor predicho, esta medida como vemos penaliza más etiquetar uno sano como patológico y viceversa que el resto de errores que pueden cometerse, puesto que como hemos mencionado anteriormente 3 es patológico, 2 es sospechoso y 1 es normal. En nuestra opinión esta penalización mayor tiene sentido, puesto que el error es más grave.

Kappa = $\frac{pr_{clasificacion} - pr_{casualidad}}{1 - pr_{casualidad}}$ donde $pr_clasificacion$ es el porcentaje de muestras bien clasificadas t $pr_{casualidad}$ es la proporción esperada simplemente por azar. Así si Kappa es 0 significa que los resultados que obtenemos son los mismos que obtendríamos al azar y en cambio si obtenemos un 1 indica que es perfecto.

Accuracy: proporción de muestras bien etiquetadas.

Estos resultados fueron obtenidos calculando los resultados medios en 10 folds.

Bibliografía

1. La base de datos: https://archive.ics.uci.edu/ml/datasets/Cardiotocography#
2. PCA con 'R': http://www.r-bloggers.com/computing-and-visualizing-pca-in-r/
3. Partición de los datos: http://stackoverflow.com/questions/...
4. http://www.prasa.org/proceedings/2004/prasa04-12.pdf
5. http://ro.uow.edu.au/cgi/viewcontent.cgi?article=1791&context=eispapers
6. http://discuss.analyticsvidhya.com/t/difference-between-ridge-regression-and-lasso-and-its-effect/3000/2
7. https://www.quora.com/Why-is-it-that...
8. http://es.slideshare.net/Sh...
9. http://goo.gl/OlocGY
10. http://file.scirp.org/pdf/JCC_2014071111175480.pdf