Métodos supervisados: Reconocimiento de género basado en voz

Sergio Sáez Bombín

23-01-2023

Índice

1. Introducción

1.1. Dataset seleccionado

Para este ejercicio se ha decidido optar por un dataset de Kaggle conformado por estadísticos frecuenciales de distintas señales de voz, cuyo objetivo es permitir la clasificación del género del emisor (***Gender Recognition by Voice***). El fichero RMarkdown relativo a este trabajo se encuentra en el siguiente enlace de GitHub.

En primer lugar, vamos a cargar el dataset elegido en R, con el objetivo de comenzar la exploración de este y la toma de decisiones en cuanto a su preparación para nuestro modelo de clasificación:

```
library(caret)
library(mlbench)
voice <- read.csv("voice.csv")
names(voice)
```

```
##  [1] "meanfreq" "sd"       "median"   "Q25"      "Q75"      "IQR"
##  [7] "skew"     "kurt"     "sp.ent"   "sfm"      "mode"     "centroid"
## [13] "meanfun"  "minfun"   "maxfun"   "meandom"  "mindom"   "maxdom"
## [19] "dfrange"  "modindx"  "label"
```

Con este sencillo código podemos ver ya de qué variabes (columnas) está compuesto nuesstro dataset, y con el siguiente, vamos a poder ver el tipo de datos que tenemos por cada una de ellas. Además, también se nos indica que disponemos de 3168 observaciones con 21 variables:

```
str(voice)
```

```
## 'data.frame':    3168 obs. of  21 variables:
##  $ meanfreq: num  0.0598 0.066 0.0773 0.1512 0.1351 ...
##  $ sd      : num  0.0642 0.0673 0.0838 0.0721 0.0791 ...
##  $ median  : num  0.032 0.0402 0.0367 0.158 0.1247 ...
##  $ Q25     : num  0.0151 0.0194 0.0087 0.0966 0.0787 ...
##  $ Q75     : num  0.0902 0.0927 0.1319 0.208 0.206 ...
##  $ IQR     : num  0.0751 0.0733 0.1232 0.1114 0.1273 ...
##  $ skew    : num  12.86 22.42 30.76 1.23 1.1 ...
##  $ kurt    : num  274.4 634.61 1024.93 4.18 4.33 ...
##  $ sp.ent  : num  0.893 0.892 0.846 0.963 0.972 ...
##  $ sfm     : num  0.492 0.514 0.479 0.727 0.784 ...
##  $ mode    : num  0 0 0 0.0839 0.1043 ...
##  $ centroid: num  0.0598 0.066 0.0773 0.1512 0.1351 ...
##  $ meanfun : num  0.0843 0.1079 0.0987 0.089 0.1064 ...
##  $ minfun  : num  0.0157 0.0158 0.0157 0.0178 0.0169 ...
##  $ maxfun  : num  0.276 0.25 0.271 0.25 0.267 ...
##  $ meandom : num  0.00781 0.00901 0.00799 0.2015 0.71281 ...
##  $ mindom  : num  0.00781 0.00781 0.00781 0.00781 0.00781 ...
##  $ maxdom  : num  0.00781 0.05469 0.01562 0.5625 5.48438 ...
##  $ dfrange : num  0 0.04688 0.00781 0.55469 5.47656 ...
##  $ modindx : num  0 0.0526 0.0465 0.2471 0.2083 ...
##  $ label   : chr  "male" "male" "male" "male" ...
```

En concreto, vamos a disponer de 20 variables predictoras (todas ellas de tipo *'num'*) y una única variable a predecir o variable objetivo, ***label***, que puede tomar los valores ***female*** o ***male***, y es de tipo *'chr'*. Por este motivo, y con el objetivo de poder alimentar a un algoritmo clasificador, debemos modificar este tipo de variable a predecir a una de tipo *'num'* (pasando por un tipo *factor* para evitar la introducción de valores nulos por coerción), quedando el mapeo como se muestra:

- *"female"* : *1*
- *"male"* : *2*

```
str(voice$label[1])
```

```
##  chr "male"
```

```
str(voice$label[1700])
```

```
##  chr "female"
```

```
voice$label <- as.factor(as.character(voice$label))
str(voice$label[1])
```

```
##  Factor w/ 2 levels "female","male": 2
```

```
str(voice$label[1700])
```

```
##  Factor w/ 2 levels "female","male": 1
```

Por otro lado, tal y como se ha mencionado anteriormente, las 20 variables predictoras consisten en una serie de estadísticos frecuenciales de la voz de las personas grabadas bajo estudio. En concreto, la descripción de estas variables son las siguientes:

Cuadro 1: Variables predictoras

1	meanfreq	Frecuencia media (en kHz)
2	sd	Desviación estándar de la frecuencia
3	median	Mediana de la frecuencia (en kHz)
4	Q25	Primer cuantil (en kHz)
5	Q75	tercer cuantil (en kHz)
6	IQR	Rango intercuantil (en kHz)
7	skew	Sesgo
8	kurt	Curtosis
9	sp.ent	Entropía espectral
10	sfm	Planitud espectral
11	mode	Moda de la frecuencia
12	centroid	Centroide de la frecuencia
13	meanfun	Media de la frecuencia fundamental medida a lo largo de la señal acústica
14	minfun	Frecuencia fundamental mínima medida a lo largo de la señal acústica
15	maxfun	Frecuencia fundamental máxima medida a lo largo de la señal acústica
16	meandom	Media de la frecuencia dominante medida a lo largo de la señal acústica
17	mindom	Frecuencia dominante mínima medida a lo largo de la señal acústica
18	maxdom	Frecuencia dominante máxima medida a lo largo de la señal acústica
19	dfrange	Rango de la frecuencia dominante media a lo largo de la seña acústica
20	modindx	Índice de modulación.

Cabe mencionar que lo primero a realizar antes de la comparación de variables, es analizar:

1. Si es necesario realizar alguna imputación de datos en nuestro dataset.
2. Si se dispone de un desbalanceo entre clases.

Para ello, se va a comprobar tanto si hay algún dato *NA* entre los datos como el número de instancias de cada clase (columna *label*).

```
sum(is.na(voice))
```

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

```
table(voice$label)
```

```
##
## female   male
##   1584   1584
```

Tras esto, podemos ver que nuestros datos están completos y tenemos un balanceo perfecto entre las clases, por lo que no es necesario aplicar ninguna técnica de imputación de datos, así como de balanceo de clases.

A continuación, y antes de ahondar en el análisis que se realiza sobre los datos y cómo entrenar los algoritmos de clasificación, se van a presentar algunas variables predictoras de interés, así como explicar algunos conceptos que puedan no ser conocidos relativos a estas.

Como se ha comentado, el dataset consiste en un conjunto de estadísticos (media, desviación típica, moda, cuantiles, etc.) que se consideran conocidos ya que son comunes en cualquier análisis de datos. Sin embargo, sí se considera de interés explicar sobre qué datos (y su naturaleza) se calculan estos estadísticos, que no son más que las señales de voz de distintas personas, siendo las frecuencias de estas señales (sus estadísticos) nuestras variables predictoras.

La naturaleza de estas variables predictoras son ondas de presión (que una vez muestreadas pueden considerarse electromagnéticas al pasarse por un micrófono), que pueden ser descompuestas en la suma de señales senoidales de las distintas frecuencias que se encuentran en ella (sin entrar en el detalle, esto se debe al trabajo de Fourier, con las series y Transformada de Fourier, que han permitido el desarrollo del análisis de todo tipo de señales). Por ello, a continuación se explican algunos conceptos con respecto a las señales, que se pueden ver en las variables predictoras del dataset:

- **Entropía espectral:** Es una normalización de la entropía de Shannon, que lo que indica es la presencia de información en la señal, es decir, permite distinguir entre el ruido y la señal generada por la voz. Valores pequeños de entropía espectral indican presencia de más información (voz), mientras que valores elevados indican poca información (ruido).
- **Planitud espectral:** Otro tipo de entropía, representa lo mismo que la entropía espectral, ya que valores elevados corresponden a ruido, y valores reducidos a la presencia de voz.
- **Frecuencia fundamental:** A pesar de no ser una variable predictora en sí misma, se encuentra en tres de ellas. Se trata la frecuencia mínima encontrada en la señal.
- **Freucencia dominante:** Al igual que la anterior, no es una variable predictora pero sí se utiliza para calcular cuatro de ellas. Es ña frecuencia con mayor magnitud (amplitud) de la señal.
- **Índice de modulación:** En el dataset es calculado como la diferencia absoluta acumulada entre medidas de frecuencias fundamentales adyacentes, divididas por el rango frecuencial.

Finalmente, se muestran gráficas de visualización de los histogramas de algunas de las variables predictoras. Se ha optado por elegir *sp.ent* y *sfm* por su definición tan parecida, y *meanfun* e *IQR* por ser consideradas clave para la clasificación por los autores del dataset)):

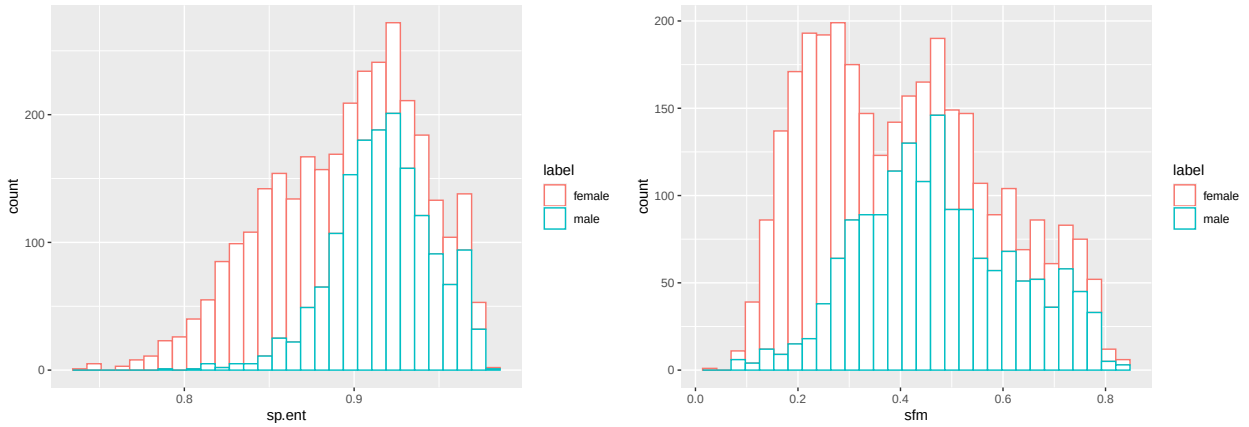

Figura 1: Histogramas de variables sp.ent y sfm

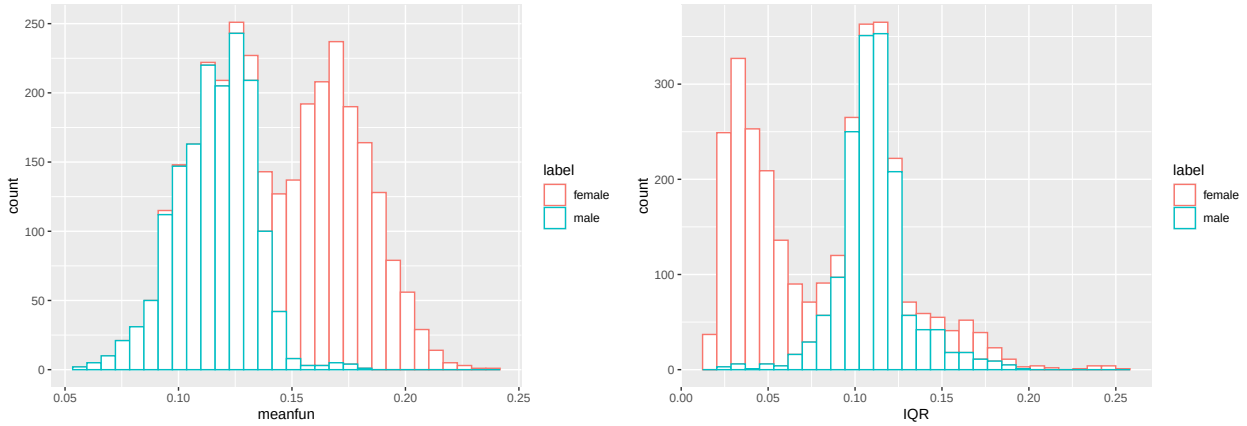

Figura 2: Histogramas de variables meanfun e IQR

Aquí se puede ver que, a pesar de representar la misma o semejante información, las variables *sp.ent* y *sfm* no tienen distribuciones tan parecidas como cabría esperar por su definición, por lo que habrá que esperar a comparar las correlaciones entre ellas para determinar si finalmente debemos descartar alguna de las dos. Por otro lado, efectivamente se puede ver que las variables *meanfun* e IQR podrían ser consideradas clave en la clasificación, ya que hay una clara distinción entre las clases en sus distribuciones: por encima de un valor de $meanfun > 0.15$ aproximadamente, prácticamente solamente se tienen muestras de la clase *female*, mientras que por debajo hay que utilizar la variable IQR para determinar la clase ($IQR \leq 0.6$-$IQR \leq 0.8$ aproximadamente, se consideraría *female*).

1.2. Secciones del notebook

Tras introducir el dataset, a continuación se presenta una explicación a alto nivel de cómo se estructura este notebook, y por tanto qué se va a presentar. La aproximación tomada para este trabajo es la siguiente:

- **Análisis y limpieza del dataset**: En primer lugar se comprobará el estado del dataset, así como cualquier correlación, dependencia, etc. que pueda encontrarse entre las variables predictoras, y que suponga una eliminación de estas.
- **Selección de variables**: Algún método de selección de variables.
- **Extracción de variables**: PCA.
- **Entrenamiento *10-fold cross-validation*** con varios clasificadores:
  - Con los datos tras su limpieza → *trainingBase*
  - Con los datos tras la selección de variables → *trainingFS*
  - Con los datos tras la extracción de variables → *trainingFE*
- **Comparativa de resultados para cada caso anterior.**

2. Preprocesado de datos

2.1. Particionado de dataset para entrenamiento y test

En primer lugar, es necesario separar nuestros datos entre entrenamiento y test con el objetivo de que nuestro algoritmo no vea datos que se usarán en la fase de predicción. La separación que se realiza es de 75% del conjunto completo de datos para entrenamiento, y el 25% restante para test.

```
set.seed(206)

inTrain <- createDataPartition(voice$label, p=.75, list=FALSE)

training <- voice[inTrain,]
testing <- voice[-inTrain,]
```

2.2. Análisis de variables

A continuación, una vez separados los datos de entrenamiento y test, realizamos el análisis de las variables predictoras para extraer cualquier dependencia o correlación entre ellas, así como desechar variables que aporten escasa o ninguna información.

Para ello, comenzamos explorando los datos, haciendo uso del paquete `skimr` (eliminamos del `summary` el histograma de cada variable predictora para evitar errores en la generación del PDF con Latex.)

```
library(skimr)
summary <- skim(training)
summary[,c(1:14)]
```

Cuadro 2: Data summary

Name	training
Number of rows	2376
Number of columns	21
Column type frequency:	
factor	1
numeric	20
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
label	0	1	FALSE	2	fem: 1188, mal: 1188

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
meanfreq	0	1	0.18	0.03	0.04	0.16	0.18	0.20	0.25
sd	0	1	0.06	0.02	0.02	0.04	0.06	0.07	0.11
median	0	1	0.19	0.04	0.01	0.17	0.19	0.21	0.26
Q25	0	1	0.14	0.05	0.00	0.11	0.14	0.18	0.24

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
Q75	0	1	0.22	0.02	0.04	0.21	0.23	0.24	0.27
IQR	0	1	0.08	0.04	0.01	0.04	0.10	0.11	0.25
skew	0	1	3.16	4.29	0.14	1.65	2.20	2.94	34.54
kurt	0	1	37.19	137.02	2.07	5.69	8.35	13.73	1271.35
sp.ent	0	1	0.90	0.05	0.74	0.86	0.90	0.93	0.98
sfm	0	1	0.41	0.18	0.04	0.26	0.40	0.54	0.83
mode	0	1	0.17	0.08	0.00	0.12	0.19	0.22	0.28
centroid	0	1	0.18	0.03	0.04	0.16	0.18	0.20	0.25
meanfun	0	1	0.14	0.03	0.06	0.12	0.14	0.17	0.23
minfun	0	1	0.04	0.02	0.01	0.02	0.05	0.05	0.20
maxfun	0	1	0.26	0.03	0.10	0.25	0.27	0.28	0.28
meandom	0	1	0.83	0.52	0.01	0.42	0.76	1.16	2.96
mindom	0	1	0.05	0.06	0.00	0.01	0.02	0.07	0.46
maxdom	0	1	5.05	3.58	0.01	2.06	4.95	7.00	21.87
dfrange	0	1	5.00	3.57	0.00	2.03	4.92	6.98	21.84
modindx	0	1	0.17	0.12	0.00	0.10	0.14	0.21	0.88

Esta tabla también nos indica lo que habíamos visto anteriormente de manera "manual", y que se mantiene tras la separación de datos entre el conjunto de entrenamiento y el de test: **no hay datos faltantes** (columnas *n_missing* de todas las variables a 0, y columnas *complete_rate* a 1) y nuestras **clases están completamente balanceadas** (la columna *top_counts* de nuestra variable a predecir, *label*, nos indica que tenemos las mismas instancias de ambas clases, 1188).

Revisando la tabla "a vista de pájaro", podemos ver que tenemos tanto variables predictoras cuyos valores tienen alta variabilidad como otras con baja. Por ello, y siguiendo el índice del paquete `caret`, vamos a comprobar las variables predictoras que tengan poca varianza (cercana a 0), así como relaciones de correlación y dependencia lineal entre ellas.

Nota: Para ninguna de estas operaciones se va a normalizar las variables previamente ya que no es necesario:

- *Varianza: Considera cada variable individualmente.*
- *Correlación: La correlación mide la relación lineal entre dos variables, es decir, la medida en que varían juntas. La normalización no afecta a la relación lineal entre las variables, por lo que no es necesario normalizar los datos antes de calcular la correlación.*
- *Dependencias lineales: Justificando de manera semejante que con la correlación, al tratarse de una dependencia/relación lineal, la normalización no afecta.*

2.2.1. Varianza cero o cercana a cero

La presencia de variables predictoras con varianza cero o cercana a cero hace que no aporten ningún tipo de información a nuestro modelo que ayude a discernir o predecir la clase de una instancia de entrenamiento. Por lo tanto, todas las variables predictoras que cumplan con este supuesto, pueden ser eliminadas de nuestro dataset.

Para realizar este cálculo, hacemos uso de la función `nearZeroVar` del paquete `caret`:

```
nzv <- nearZeroVar(training, saveMetrics=TRUE)
nzv
```

```
##          freqRatio percentUnique zeroVar   nzv
## meanfreq  2.000000   99.95791246   FALSE FALSE
## sd        1.000000   99.87373737   FALSE FALSE
## median    1.000000   97.81144781   FALSE FALSE
## Q25       2.000000   98.44276094   FALSE FALSE
## Q75       1.250000   96.88552189   FALSE FALSE
```

```
## IQR       1.500000   97.64309764   FALSE FALSE
## skew      2.000000   99.95791246   FALSE FALSE
## kurt      2.000000   99.95791246   FALSE FALSE
## sp.ent    1.000000   99.78956229   FALSE FALSE
## sfm       2.000000   99.95791246   FALSE FALSE
## mode     25.142857   90.27777778   FALSE FALSE
## centroid  2.000000   99.95791246   FALSE FALSE
## meanfun   1.000000   99.87373737   FALSE FALSE
## minfun    1.066667   32.87037037   FALSE FALSE
## maxfun    1.172012    4.75589226   FALSE FALSE
## meandom  11.250000   95.62289562   FALSE FALSE
## mindom    1.548440    3.11447811   FALSE FALSE
## maxdom    3.000000   39.85690236   FALSE FALSE
## dfrange   3.357143   41.37205387   FALSE FALSE
## modindx  15.666667   97.43265993   FALSE FALSE
## label     1.000000    0.08417508   FALSE FALSE
```

Como se puede observar, no hay **ninguna variable predictora** cuya **varianza sea cero o cercana a cero**, por lo que podemos continuar el análisis con todas las variables.

2.2.2. Variables predictoras correladas

Por otro lado, dos variables correladas entre sí podrían considerarse como que están aportando información semejante o redundante con respecto a la variable clase.

A continuación, se va a comprobar si algunas de las variables predictoras está correlacionada entre sí, de tal manera qe podamos eliminar alguna de ellas. Denominando al coeficiente de correlación r, podemos entender una baja o alta correlación en base a los siguientes valores de r (en valor absolito):

- $r<0.25$ = Baja correlación
- $0.25<r<0.5$ = Correlación débil
- $0.5<r<0.75$ = Correlación moderada
- $r>0.75$ = Alta correlación

Para este ejercicio, solamente vamos a eliminar las variables predictoras que estén **altamente correlacionadas**, es decir, cuyo $r>0.75$.

```
library('plot.matrix')
r <- 0.75
varTraining <- training[,c(1:20)]
correlation <- cor(varTraining)
plot(correlation>r,
     main=paste("Correlation>",r),
     cex.lab=0.5,
     cex.axis=0.5,
     axis.col=list(side=1, las=2), axis.row = list(side=2, las=1))
```

En primer lugar, se muestra una matriz lógica en la que determinamos si la correlación entre las variables es mayor o menor que el valor indicado. Con esta representación se confirma lo que se indicaba cuando se exploraron algunas variables predictoras al inicio del trabajo → la entropía espectral (sp.ent) y la planitud espectral (sfm) están altamente correlacionadas. Además, se puede ver a simple vista que hay varios casos de correlación entre variables, como también se muestra en la siguiente `Figura 4`, acompañada del resumen estadístico de la correlación de variables. Teniendo en cuenta que esta es una matriz simétrica con respecto a la diagonal, solamente debemos considerar la parte triangular superior de esta (incluimos la función `upper.tri` para ello).

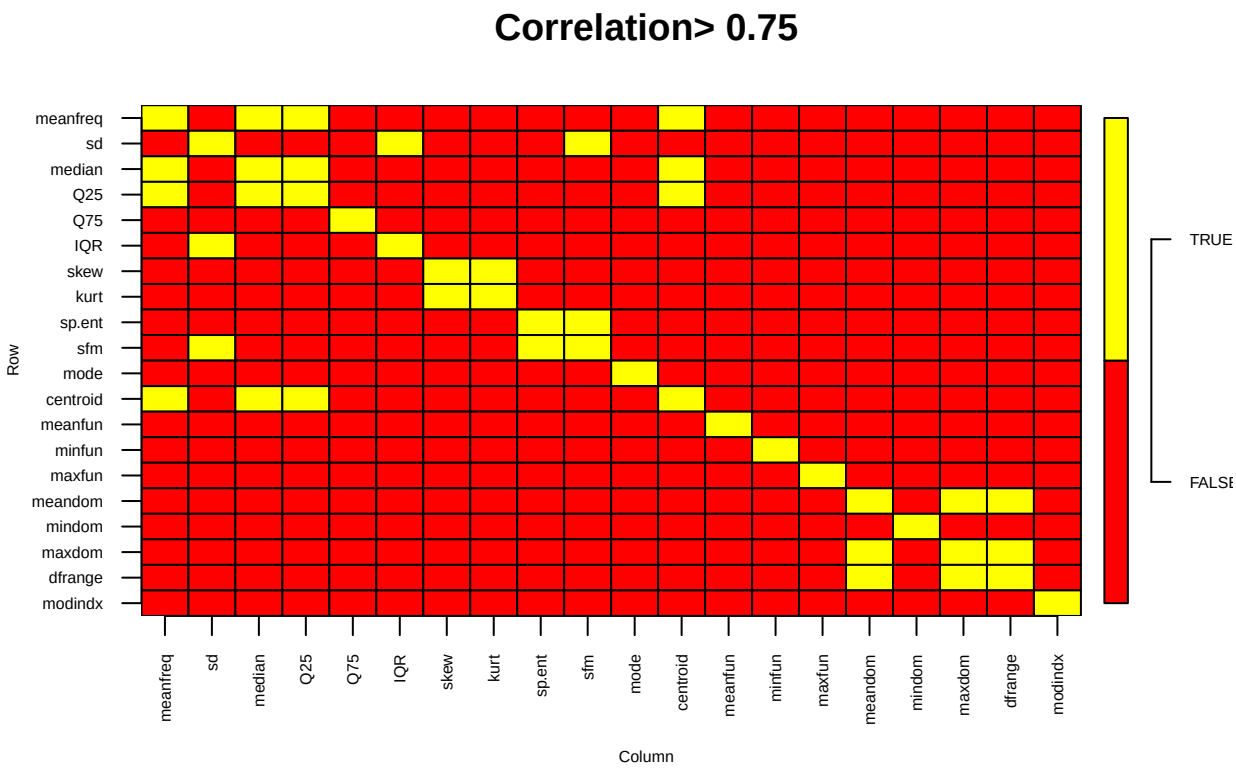

Figura 3: Matriz de correlación entre variables predictoras

```
corrSummary <- summary(correlation[upper.tri(correlation)])
corrSummary
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -0.875302 -0.319348 -0.001754  0.043033  0.374038  1.000000
```

```
plot(correlation[upper.tri(correlation)])
abline(h=r, col="darkgreen")
abline(h=-r, col="darkgreen")
abline(h=corrSummary[2], col="blue", lwd = 4, lty = 4)
abline(h=corrSummary[5], col="blue", lwd = 4, lty = 4)
```

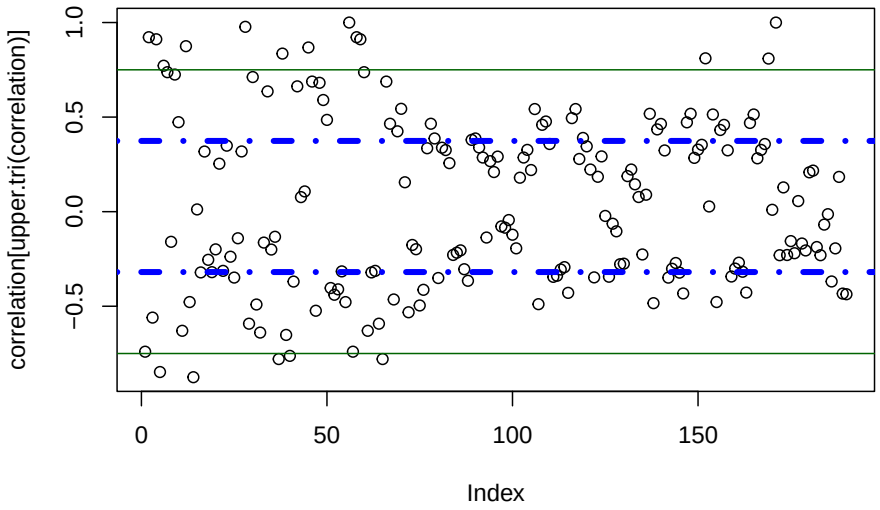

Figura 4: Matriz triangular superior de correlación, marcado en verde los límites de alta correlación y en azul el primer y tercer cuantil

Por lo tanto, tras confirmar que tenemos variables predictoras altamente correlacionadas, debemos identificarlas y posteriormente eliminarlas de nuestros datos para continuar el análisis.

```
highCorr <- findCorrelation(abs(correlation), cutoff=r)
highCorr
```

```
## [1]  1 12  4  2 10 18 19  7
```

Tras eliminarlas, comprobamos que efectivamente, el resumen estadístico ha mejorado: tanto correlación mínima y máxima como el 1er y 3er cuantil son más cercanos a 0, indicando que el 75% de las variables están menos correlacionadas que antes.

```
trainingUncorr <- varTraining[,-highCorr]
correlation2 <- cor(trainingUncorr)
corrSummary2 <- summary(correlation2[upper.tri(correlation2)])
corrSummary2
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.53194 -0.22821 -0.01804  0.02181  0.28800  0.72496
```

```
plot(correlation2[upper.tri(correlation2)])
abline(h=r, col="darkgreen")
abline(h=-r, col="darkgreen")
abline(h=corrSummary2[2], col="blue", lwd = 4, lty = 4)
abline(h=corrSummary2[5], col="blue", lwd = 4, lty = 4)
```

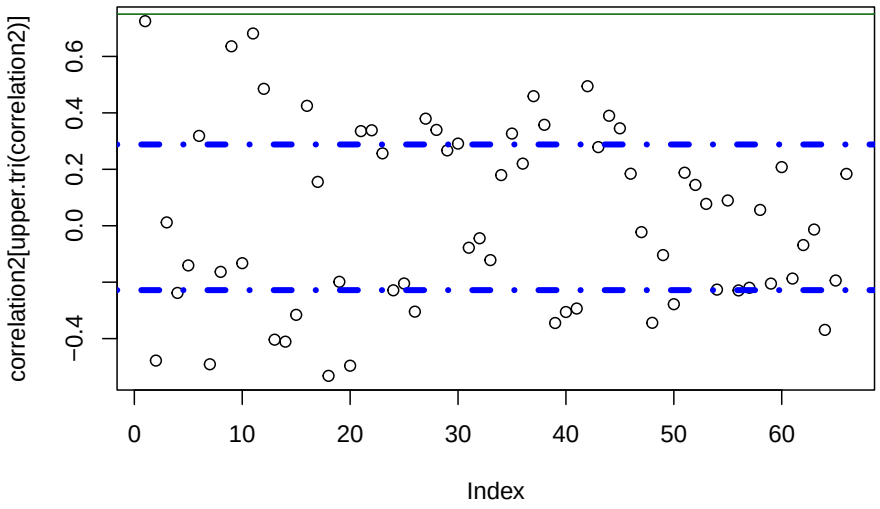

Figura 5: Matriz triangular superior de correlación, marcado en verde los límites de alta correlación y en azul el primer y tercer cuantil

Como ejemplo, se muestran un caso de dos variables no correladas (`Figura 6`) y otro de variables correladas (`Figura 7`) a través de sendos gráficos:

Por lo tanto, debemos eliminar estas variables de nuestro dataset de entrenamiento original, *training*.

```
trainingUncorr <- training[,-highCorr]
```

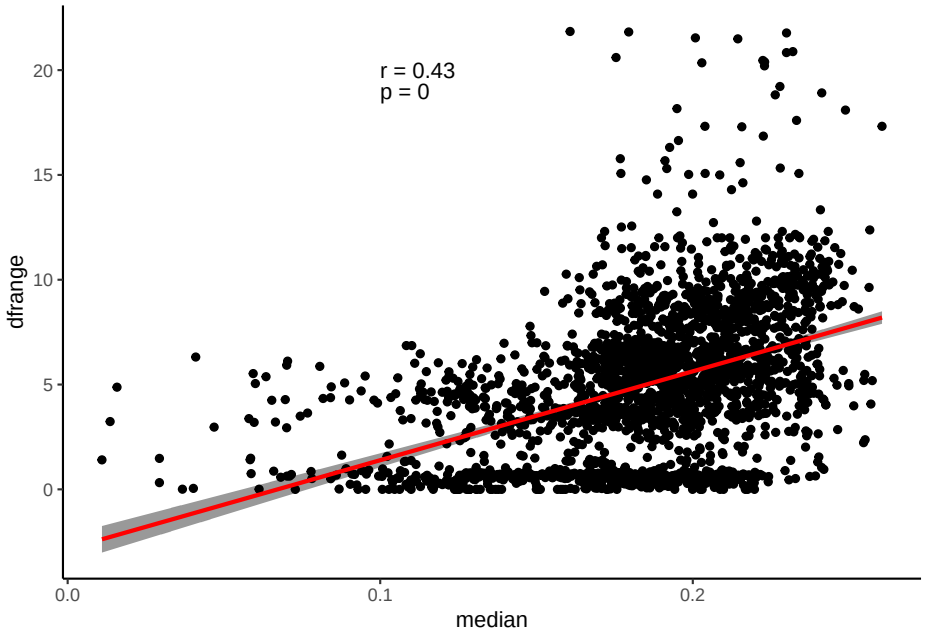

Figura 6: Variables median vs. dfrange

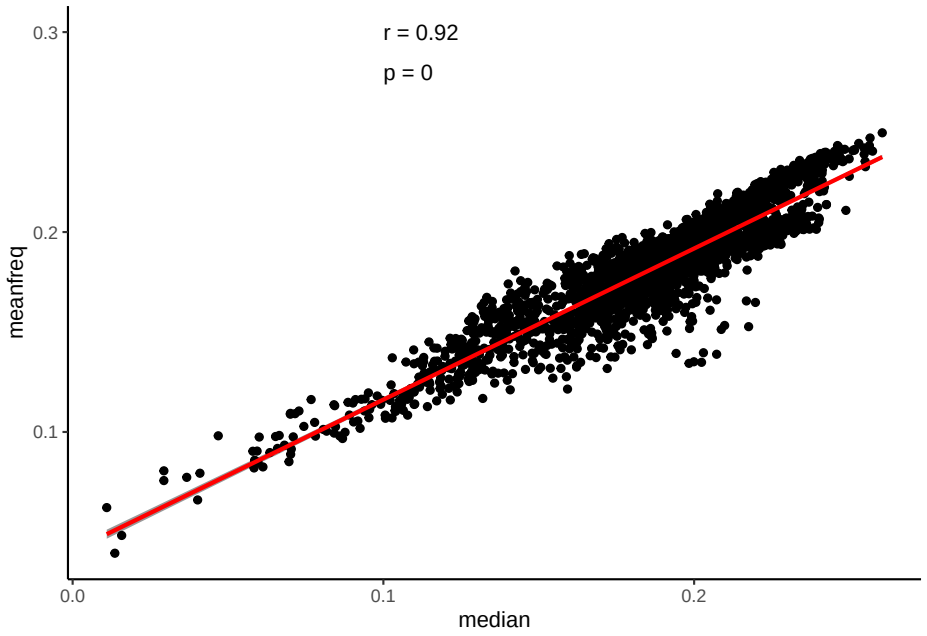

Figura 7: Variables median vs. meanfreq

2.2.3. Dependencias lineales entre variables

De manera similar al anterior apartado, es necesario encontrar variables predictoras que compartan dependencias lineales entre sí. Esto es así porque estas variables, en realidad, lo que aportan es, en realidad, la misma información.

Para identificar si existen o no dependencias lineales entre estas variables predictoras, se va a utilizar la función *findLinearCombos* del paquete `caret`:

```
linearities <- findLinearCombos(trainingUncorr[1:12])
linearities
```

```
## $linearCombos
## list()
##
## $remove
## NULL
```

Este resultado nos indica que **no hay ninguna dependencia lineal entre variables predictoras**, por lo que no debemos eliminar ninguna.

Finalmente, tras estas comprobaciones, disponemos de las siguientes 12 variables predictoras, que guardaremos como nuestro dataset base (*trainingBase*) junto a la variable clase:

```
##  [1] "median"  "Q75"     "IQR"     "kurt"    "sp.ent"  "mode"    "meanfun"
##  [8] "minfun"  "maxfun"  "meandom" "mindom"  "modindx" "label"
```

```
##  [1] "median"  "Q75"     "IQR"     "kurt"    "sp.ent"  "mode"    "meanfun"
##  [8] "minfun"  "maxfun"  "meandom" "mindom"  "modindx" "label"
```

3. Selección y extracción de variables predictoras

Recordemos que en este trabajo se va a realizar un entrenamiento con validación cruzada *10-fold*. Con esto cabe la pregunta: ¿Se debería realizar la selección y extracción de variables para cada *fold* o previo a estas tareas?

Hablando de forma estricta, las técnicas de selección y extracción deben aplicarse de manera independiente en cada *fold*, de tal manera que no hay forma posible de que durante el entrenamiento algún tipo de información (derivada de estas técnicas) se encuentre tanto en el set de entrenamiento como el de validación.

Para este trabajo, la selección de variables se realizará también con validación cruzada *10-fold*, utilizando la misma semilla que se utilizará durante el entrenamiento. De esta forma, los conjuntos de datos se dividirán exactamente de la misma forma en ambas ocasiones, evitando cualquier tipo de contaminación entre el set de entrenamiento y de validación.

Por otro lado, con respecto a la extracción de variables, en la sección 3.2 se explicará la técnica utilizada y la justificación de ciertas decisiones, utilizando el set de datos completo. Sin embargo, para el entrenamiento, se aplicará esta misma técnica para cada *fold* de manera independiente, buscando así una clasificación honesta.

3.1. Selección de variables

Tal y como se ha indicado al inicio de este notebook, una de las tareas a realizar en cuanto al *pipeline* de análisis de datos aquí llevado a cabo es la selección de variables. Para esta tarea, se va a utilizar lo que se conoce como ***Recursive Feature Elimination***, siguiendo el índice del paquete `caret` (enlace).

Este método ejecuta una búsqueda hacia atrás de qué variables son más importantes para el rendimiento del modelo. Es un proceso iterativo en el que, partiendo de todas las variables predicotras de las que dispone el modelo, va reentrenándolo eliminando aquellas que se queden fuera del grupo de variables más importantes.

Para este ejercicio se realizará este algoritmo con una validación cruzada *10-fold*, con tamaños de set de variables de 1 a 6, y con funciones de Naive Bayes (aunque no se define ningún modelo en específico):

```
library(klaR)
```

```
## Loading required package: MASS
```

```
set.seed(206)

normalization <- preProcess(trainingBase[1:12])
x <- predict(normalization, trainingBase[1:12])
x <- as.data.frame(x)

ctrl <- rfeControl(functions = nbFuncs, method = "cv", verbose = TRUE)
rfeProfile <- rfe(x=x, y=trainingBase$label,
                  sizes = c(1:6),
                  rfeControl = ctrl)
rfeProfile
```

```
##
## Recursive feature selection
##
## Outer resampling method: Cross-Validated (10 fold)
##
## Resampling performance over subset size:
##
##  Variables Accuracy  Kappa AccuracySD KappaSD Selected
##          1   0.9541 0.9083    0.01056 0.02112
##          2   0.9659 0.9318    0.01038 0.02076        *
```

```
##          3   0.9604 0.9209    0.01089 0.02179
##          4   0.9604 0.9209    0.01302 0.02604
##          5   0.9562 0.9125    0.01317 0.02634
##          6   0.9575 0.9150    0.01364 0.02728
##         12   0.9520 0.9041    0.01429 0.02858
##
## The top 2 variables (out of 2):
##    meanfun, IQR
```

Este resultado corrobora lo que se introducía al inicio del trabajo, las variables *meanfun* e *IQR* son las principales predictoras de la clase. Por lo tanto, vamos a reducir nuestro conjunto de variables predictoras a solamente dos (*trainingFS*):

```
trainingFS <- trainingBase[predictors(rfeProfile)]
trainingFS$label <- trainingBase$label
```

3.2. Extracción de variables

Con respecto a la extracción de variables predictoras, se va a ejecutar, tal y como se solicita en el enunciado del trabajo, un ***Principal Component Anlysisis* (PCA)**. Este método de extracción de variables lo que busca es reducir la dimensionalidad de nuestros datos a partir de combinaciones lineales de las variables predictoras. Es decir, intenta reducir las variables predictoras a utilizar, generando unas nuevas que contengan toda la información, de manera conjunta, que estaba presente en nuestras variables originales.

Para ello, vamos a utilizar la función `prcomp`, centrando y escalando previa ejecución, con el objetivo de que todas las variables tengan una distribución normal:

```
pca <- prcomp(trainingBase[,1:12], center=TRUE, scale=TRUE)
pca
```

```
## Standard deviations (1, .., p=12):
##  [1] 2.0317062 1.3283224 1.1846202 1.0299016 0.9007987 0.8399513 0.7771491
##  [8] 0.7118852 0.6029382 0.5743250 0.4563142 0.3382349
##
## Rotation (n x k) = (12 x 12):
##                PC1           PC2         PC3        PC4         PC5
## median   0.4177783 -0.0814136658  0.05823912 -0.3339445  0.02453449
## Q75      0.2830979 -0.3476275299  0.23745226 -0.4406684 -0.16928161
## IQR     -0.3262841 -0.3753267077  0.21098041 -0.1527749 -0.31769398
## kurt    -0.1834230 -0.1807885156 -0.59225816 -0.3073064 -0.36487423
## sp.ent  -0.3135638 -0.1891926361  0.49759498  0.1065401  0.12904145
## mode     0.3800822 -0.0151021143  0.24622049 -0.1796087  0.14433588
## meanfun  0.3178615  0.1387576565 -0.22026287  0.3674749 -0.05968270
## minfun   0.2749328  0.0007528243  0.10902649  0.2489161 -0.73422069
## maxfun   0.2029431 -0.4348259478 -0.12301353  0.4239016  0.04355805
## meandom  0.3338909 -0.1064791249  0.14780033  0.1399385  0.04566048
## mindom   0.1238045  0.4980567051 -0.02899916 -0.3532172 -0.02106279
## modindx -0.1334400  0.4418418481  0.37215070  0.1089504 -0.39003424
##                 PC6          PC7          PC8          PC9         PC10
## median  -0.264341946  0.04372983  0.07936478 -0.155635370  0.04552470
## Q75     -0.303758903 -0.11619312  0.10876327 -0.045954964  0.30983346
## IQR      0.012105132 -0.19098149 -0.12147170  0.359730209 -0.04009128
## kurt     0.051191405 -0.07338734  0.17038755  0.190668694 -0.32777717
## sp.ent   0.007170093 -0.18638832 -0.23490038  0.192422631  0.02203201
## mode     0.082461460  0.21546348 -0.25228753  0.228124288 -0.73564691
## meanfun -0.473644731 -0.12165279 -0.15937483  0.636323632  0.14840376
```

```
## minfun   0.295347504  0.22373484 -0.33736585 -0.168467752  0.11904696
## maxfun  -0.084916861 -0.56070792 -0.03377418 -0.395812723 -0.30100628
## meandom  0.561123420 -0.14598903  0.60236291  0.345608943  0.10504547
## mindom   0.266735224 -0.66195296 -0.30971182  0.000267515  0.07076279
## modindx -0.351351594 -0.15105475  0.46918873 -0.097137877 -0.32921301
##                 PC11          PC12
## median   0.367152463  0.6812759432
## Q75     -0.110005533 -0.5403448480
## IQR     -0.458838425  0.4369948906
## kurt     0.394194939 -0.1382429688
## sp.ent   0.672438673 -0.1030764539
## mode    -0.101262175 -0.1457680281
## meanfun  0.029848415 -0.0152750168
## minfun   0.133277949 -0.0081025054
## maxfun  -0.054415143  0.0063140170
## meandom  0.052776382  0.0372513018
## mindom  -0.017505415  0.0005290455
## modindx  0.006400878 -0.0146637464
```

```
summary(pca)
```

```
## Importance of components:
##                           PC1   PC2    PC3     PC4     PC5     PC6     PC7
## Standard deviation     2.032 1.328 1.1846 1.02990 0.90080 0.83995 0.77715
## Proportion of Variance 0.344 0.147 0.1169 0.08839 0.06762 0.05879 0.05033
## Cumulative Proportion  0.344 0.491 0.6080 0.69636 0.76398 0.82277 0.87310
##                           PC8     PC9    PC10    PC11    PC12
## Standard deviation     0.71189 0.60294 0.57432 0.45631 0.33823
## Proportion of Variance 0.04223 0.03029 0.02749 0.01735 0.00953
## Cumulative Proportion  0.91533 0.94563 0.97311 0.99047 1.00000
```

Como se puede extraer del resultado, un tercio de la varianza de las componentes se concentra en la primera componente, quedando repartida la varianza de datos completa entre el resto de componentes. La cantidad de componentes principales (o vectores propios) a elegir en un (PCA) depende del objetivo de la investigación y de la cantidad de datos disponibles. En general, se suele elegir el número de componentes que explican una cantidad significativa de la varianza en los datos. Sin embargo, la elección final debe basarse en un equilibrio entre la explicación de la varianza y la interpretabilidad de los componentes.

```
ggplot(as.data.frame(pca$sdev^2/sum(pca$sdev^2)),
       aes(x = 1:length(pca$sdev), y = pca$sdev^2/sum(pca$sdev^2))) +
  geom_line() +
  xlab("Número de componentes principales") +
  ylab("Proporción de varianza aportada")
```

```
ggplot(as.data.frame(pca$sdev^2/sum(pca$sdev^2)),
       aes(x = 1:length(pca$sdev), y = cumsum(pca$sdev^2/sum(pca$sdev^2)))) +
  geom_line() +
  xlab("Número de componentes principales") +
  ylab("Varianza acumulada")
```

Estas gráficas (`Figura 8` y `Figura 9`) nos indican cuánto aporta cada componente a la varianza de los datos. Analizando estas, podemos ver que con 6 componentes principales (reduciendo a la mitad el número de variables predictoras) podemos mantener alrededor del 82% de la varianza de los datos, aportando, a partir de este punto menos de un 5% por cada nueva componente que se añade. Por lo tanto, para el ejercicio se priorizará reducir a la mitad el número de variables predictoras, considerándose solamente las primeras 6 componentes principales.

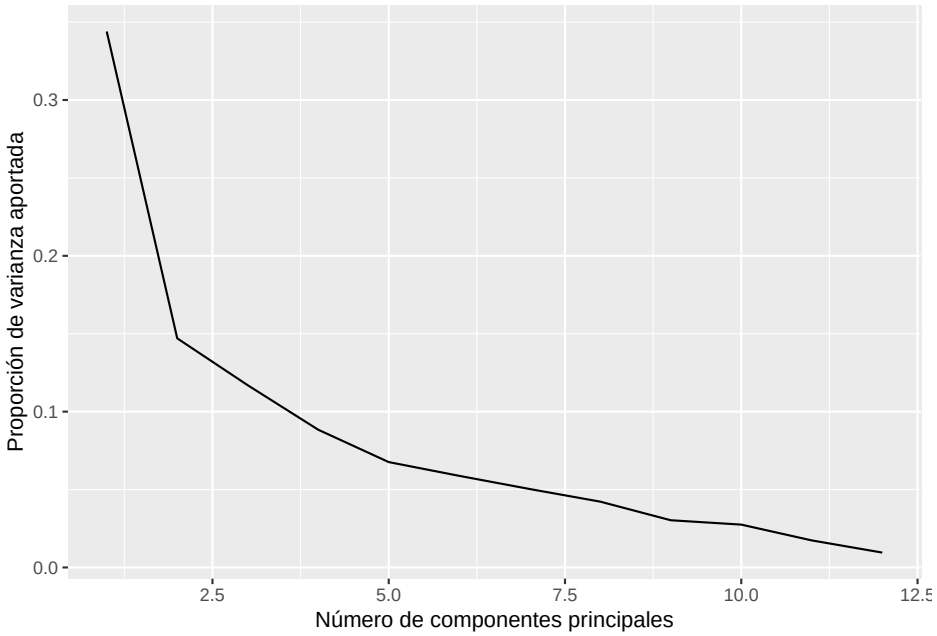

Figura 8: Proporción de varianza que aporta cada una de las componentes principales

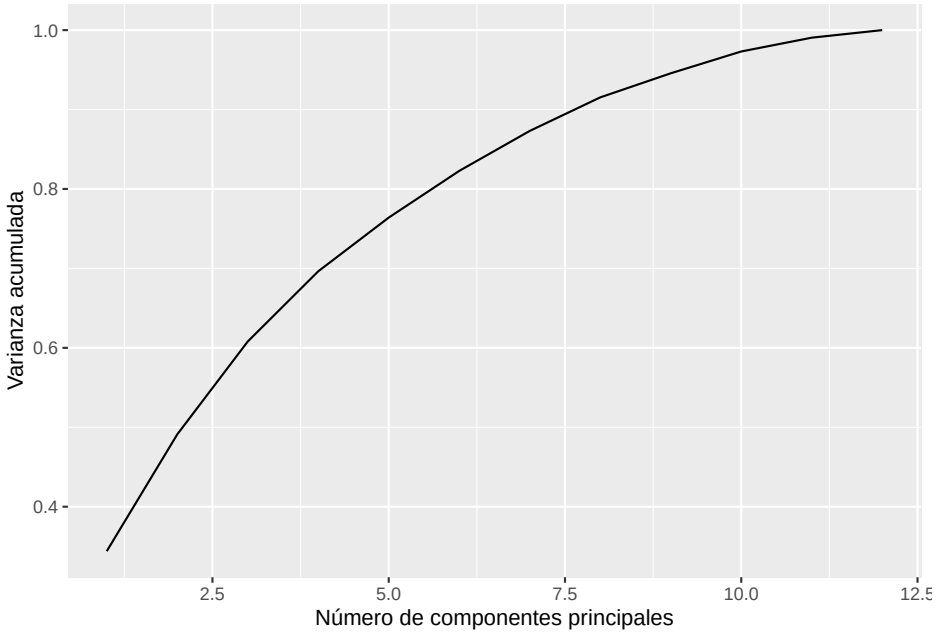

Figura 9: Varianza total acumulada por cada componente principale adicional considerada

En el enunciado del ejercicio se solicita que se visualicen las dos primeras componentes principales resultantes, para poder intuir una separabilidad entre clases. Para ello se hace uso del paquete ***ggbiplot*** (enlace).

```
library(ggbiplot)
ggbiplot(pca, obs.scale=1, var.scale=1,
         groups=trainingBase$label, ellipse=TRUE, circle=TRUE,
         alpha=0.25) +
         scale_color_discrete(name='') +
         theme(legend.direction='horizontal', legend.position='top')
```

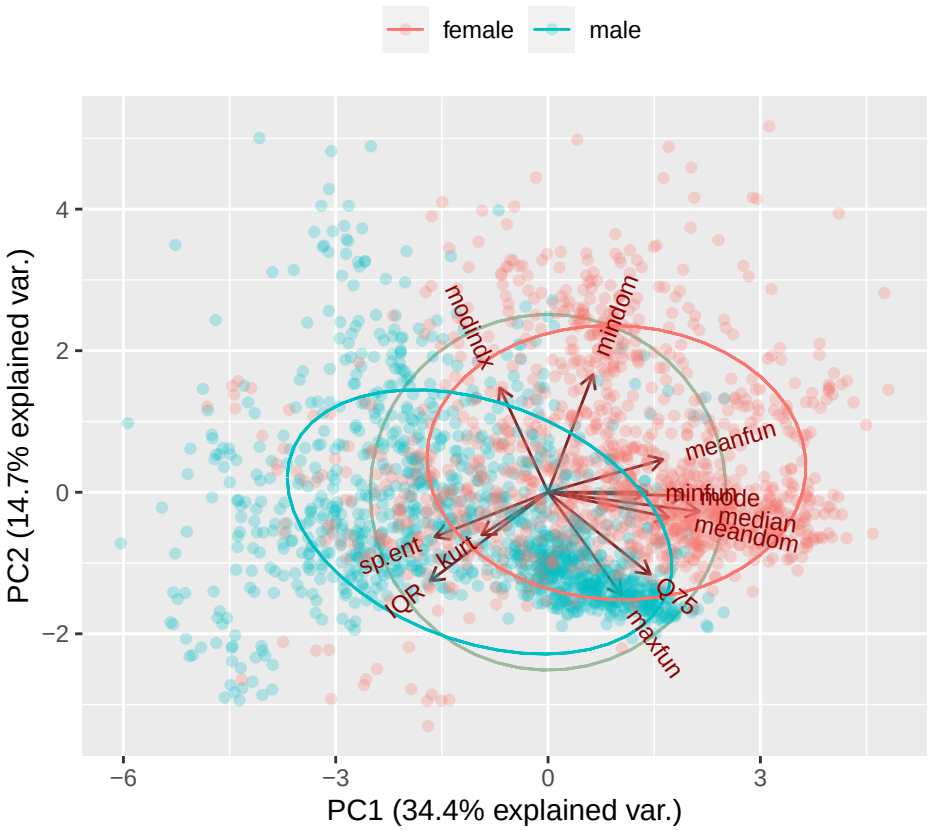

Figura 10: PC1 vs. PC2 con cada clase

De esta gráfica (`Figura 10`) se puede extraer que a pesar de no tener una separación plenamente clara entre las dos clases, sí se puede ver que cuando la PC1 y PC2 toman valores positivos (cuadrante superior derecho), la clase es *'female'*, mientras que cuando PC1 es negativo (independientemente del valor de PC2), generalmente la clase es *'male'*.

Recordemos que las componentes principales son combinaciones lineales de las variables originales. Los pesos de cada variable original en estas nuevas componentes se pueden ver en la matriz 12x12 que muestra la sección *Rotation*:

$$PC_j = \sum_{i=1}^{12} w_i \times X_i \qquad 1 \leq j \leq 12$$

```
pca$rotation
```

```
##                PC1           PC2         PC3        PC4         PC5
## median   0.4177783 -0.0814136658  0.05823912 -0.3339445  0.02453449
## Q75      0.2830979 -0.3476275299  0.23745226 -0.4406684 -0.16928161
## IQR     -0.3262841 -0.3753267077  0.21098041 -0.1527749 -0.31769398
## kurt    -0.1834230 -0.1807885156 -0.59225816 -0.3073064 -0.36487423
## sp.ent  -0.3135638 -0.1891926361  0.49759498  0.1065401  0.12904145
## mode     0.3800822 -0.0151021143  0.24622049 -0.1796087  0.14433588
## meanfun  0.3178615  0.1387576565 -0.22026287  0.3674749 -0.05968270
## minfun   0.2749328  0.0007528243  0.10902649  0.2489161 -0.73422069
## maxfun   0.2029431 -0.4348259478 -0.12301353  0.4239016  0.04355805
## meandom  0.3338909 -0.1064791249  0.14780033  0.1399385  0.04566048
## mindom   0.1238045  0.4980567051 -0.02899916 -0.3532172 -0.02106279
## modindx -0.1334400  0.4418418481  0.37215070  0.1089504 -0.39003424
##                 PC6          PC7          PC8          PC9         PC10
## median  -0.264341946  0.04372983  0.07936478 -0.155635370  0.04552470
## Q75     -0.303758903 -0.11619312  0.10876327 -0.045954964  0.30983346
## IQR      0.012105132 -0.19098149 -0.12147170  0.359730209 -0.04009128
## kurt     0.051191405 -0.07338734  0.17038755  0.190668694 -0.32777717
## sp.ent   0.007170093 -0.18638832 -0.23490038  0.192422631  0.02203201
## mode     0.082461460  0.21546348 -0.25228753  0.228124288 -0.73564691
## meanfun -0.473644731 -0.12165279 -0.15937483  0.636323632  0.14840376
## minfun   0.295347504  0.22373484 -0.33736585 -0.168467752  0.11904696
## maxfun  -0.084916861 -0.56070792 -0.03377418 -0.395812723 -0.30100628
## meandom  0.561123420 -0.14598903  0.60236291  0.345608943  0.10504547
## mindom   0.266735224 -0.66195296 -0.30971182  0.000267515  0.07076279
## modindx -0.351351594 -0.15105475  0.46918873 -0.097137877 -0.32921301
##                PC11          PC12
## median   0.367152463  0.6812759432
## Q75     -0.110005533 -0.5403448480
## IQR     -0.458838425  0.4369948906
## kurt     0.394194939 -0.1382429688
## sp.ent   0.672438673 -0.1030764539
## mode    -0.101262175 -0.1457680281
## meanfun  0.029848415 -0.0152750168
## minfun   0.133277949 -0.0081025054
## maxfun  -0.054415143  0.0063140170
## meandom  0.052776382  0.0372513018
## mindom  -0.017505415  0.0005290455
## modindx  0.006400878 -0.0146637464
```

Por ejemplo, las dos componentes principales en este ejercicio serían:

$$PC_1 = (0.4199184 \times median) + (0.2848441 \times Q75) - (0.3330073 \times IQR) -$$
$$(0.185731 \times kurt) - (0.327417 \times sp.ent) + (0.3825853 \times mode) +$$
$$(0.3114205 \times meanfun) + (0.2668212 \times minfun) + (0.1844461 \times maxfun) +$$
$$(0.3284787 \times meandom) + (0.1428324 \times mindom) - (0.1143365 \times modindx)$$

$$PC_2 = (-0.0950030147 \times median) - (0.3628411461 \times Q75) - (0.3676992877 \times IQR) -$$
$$(-0.1354436250 \times kurt) - (0.1991776067 \times sp.ent) - (0.0399646470 \times mode) +$$
$$(0.1557273784 \times meanfun) + (0.0003266332 \times minfun) - (0.4316993088 \times maxfun) -$$
$$(0.1345968005 \times meandom) + (0.4957604409 \times mindom) + (0.4358829733 \times modindx)$$

Si se deseara aplicar este PCA a nuestro conjunto de datos, bastaría con utilizar la función `predict` de `caret`, quedándonos solamente con las 6 primeras componentes y añadiendo la variable clase:

```
trainingFE <- predict(pca,newdata=trainingBase)[,1:6]
trainingFE <- as.data.frame(trainingFE)
trainingFE$label <- trainingBase$label
colnames(trainingFE)
```

```
## [1] "PC1"   "PC2"   "PC3"   "PC4"   "PC5"   "PC6"   "label"
```

Sin embargo, no se va a utilizar realmente este dataset *trainingFE* durante el entrenamiento, ya que no nos daría un resultado honesto (esto se explicará en el apartado 4.1.3).

4. Entrenamiento

Como se indicó al inicio de este *notebook*, se va a realizar un entrenamiento con 3 conjuntos de datos procesados de manera diferente (con procesamiento base, con selección de variables, y con extracción de variables por PCA), utilizando dos clasificadores distintos, para posteriormente comparar en qué caso ha rendido mejor. La propia página de presentación del dataset ya indica muy buenos resultados con regresión logística, CART, XGBoost, etc., por lo que se intenta probar otros clasificadores.

Al tratarse de una clasificación binaria, se van a utilizar dos algoritmos de clasificación ampliamente conocidos como son el **Support Vector Machine** (aunque con una pequeña variación para diferenciarlo del trabajo de los autores del dataset) y **k-Nearest Neighbors**.

Pero antes de adentrarnos en el en entrenamiento, se va a configurar la variable \texttt{trainControl], con el objetivo de poder controlar ciertos aspectos del entrenamiento de manera sencilla. En este caso, se opta por una **validación cruzada *10-fold* sin repetición**:

```
trainControl <- trainControl(
  method='cv',
  number=10,
  allowParallel=TRUE)
```

Cabe mencionar que, al tratarse de un dataset con un balanceo perfecto de clases, se opta por utilizar como métrica de decisión sobre el mejor modelo la **Accuracy**:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Siendo TP los positivos verdaderos, TN los negativos verdaderos, FP los falsos positivos y FN los falsos negativos. Por lo tanto, se puede considerar la *accuracy* como la fracción entre los bien clasificados y todos los clasificados. Esta naturaleza hace que en un escenario como el de este ejercicio, de un conjunto de datos perfectamente balanceado, esta sea una métrica válida, mientras que en un caso de desbalanceo de clases, sería engañosa.

A continuación se muestra un ejemplo a modo de justificación:

- **Caso desbalanceado:** Tenemos 10 muestras (9 son de la clase A y 1 de la B) y nuestro algoritmo clasifica perfectamente la clase A pero no la clase B. El resultado es una **ccuracy* del 90%, cuando en realidad solo es capaz de detectar una de las clases.
- **Caso balanceado:** Por otro lado, si de las 10 muestras tenemos 5 de cada clase y nuestro algoritmo clasifica bien solamente una de las clases, tendremos una *accuracy* del 50%, por lo que no estaremos "auto-engañándonos" con la métrica (lo mismo aplicaría para otros porcentajes del resultado).

4.1. Clasificador Support Vector Machine (SVM) con función kernel de base radial

Conocido en inglés como ***Support Vector Machine with Radial Basis Function Kernel*** **(SVM-RBF)**, este es un clasificador que, al igual que un **SVM** convencional, trata de encontrar una frontera lineal (un hiperplano) que permita diferenciar las clases bajo estudio, maximizando la distancia de esta con los puntos de cada clase.

El aspecto clave de este algoritmo es que, generalmente, los datasets con los que se trabaja no son linealmente separables, por lo que es necesario transformar el espacio de características (variables predictoras) a otra dimensión donde sí lo sean. Es aquí donde entra en juego la **función kernel de base radial**, que, como otras funciones kernel, convierte el espacio de entrada de dimensión reducida en otro espacio de dimensión mayor. El aspecto diferenciador de la base radial es que es más potente y eficiente que otros kernels lineales o polinómicos, al combinar diferentes kernels de este tipo.

En concreto, la función de base radial utilizada por este kernel es la siguiente:

$$K(X_1, X_2) = exp(-\frac{||X_1 - X_2||^2}{2\sigma^2})$$

Siendo el numerador la *distancia euclídea al cuadrado* y σ un parámetro libre. Si se introduce el parámetro $\gamma = \frac{1}{2\sigma^2}$, la función queda como sigue:

$$K(X_1, X_2) = exp(-\gamma||X_1 - X_2||^2)$$

Esto permite tomar ventaja del "*truco del kernel*", al evitarnos transformar los datos directamente, encontrando la transformación de los productos internos para realizar el mapeo a dimensiones superiores.

4.1.1. SVM-RBF con datos base

En primer lugar, se va a realizar el entrenamiento con los datos "base", es decir, tras eliminar las variables sin varianza, las correladas y aquellas que pudieran tener dependencias lineales (que en el caso de este dataset, no hay ninguna de este último caso). Por lo tanto, el conjunto de datos utilizado en este entrenamiento es anterior a la selección y extracción de variables (*trainingBase*).

Además, es en este ejemplo donde se va a profundizar más en las diferencias entre las opciones de entrenamiento ***tuneLength*** y ***tuneGrid***, tal y como se pide en el enunciado del trabajo.

Entrenamiento con tuneLength
Esta opción nos va a permitir delegar en `caret` el *tuning* de los hiperparámetros de nuestro modelo, que, en este son el **coste (C)** y la σ **en nuestra fórmula**:

- **C:** Por lo que se ha podido encontrar en implementaciones de este algoritmo en otros paquetes como scikit-learn, este coste es un parámetro de regularización, que actúa sobre la maximización de la función de decisión. A mayor valor tome, más estricto es el umbral dedecisión, y a menor valor, menos estricto es este umbral.
- **σ:** Como se puede ver en la fórmula es un parámetro que actúa sobre γ. Este γ se puede ver como el inverso del radio de influencia de las muestras seleccionadas por el modelo como vectores soporte. Esto es, cuando toma un valor reducido (σ alto) estos tienen poca influencia, mientras que si toma un valor elevado (σ bajo) su influencia es mayor.

En este caso, vamos a pedir a `caret` que haga el entrenamiento probando con 10 valores diferentes de estos hiperparámetros (tal y como se puede ver en la documentación, sección Support Vector Machines With Radial Basis Function Kernel, cuando se utiliza *tuneLength* se tomarán 10 valores de C y solamente un máximo de 6 para σ):

```
set.seed(206)
svmBase = train(label ~ ., data=trainingBase, method='svmRadialSigma',
                tuneLength=10, metric='Accuracy',
                trControl=trainControl, preProc=c("center","scale"))
svmBase
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 2376 samples
##   12 predictor
##    2 classes: 'female', 'male'
##
## Pre-processing: centered (12), scaled (12)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2138, 2139, 2138, 2139, 2138, 2138, ...
## Resampling results across tuning parameters:
##
```

```
##    sigma       C       Accuracy   Kappa
##    0.02052912    0.25  0.9718025  0.9436035
##    0.02052912    0.50  0.9743253  0.9486494
##    0.02052912    1.00  0.9764262  0.9528510
##    0.02052912    2.00  0.9781175  0.9562342
##    0.02052912    4.00  0.9776920  0.9553830
##    0.02052912    8.00  0.9776938  0.9553868
##    0.02052912   16.00  0.9785341  0.9570670
##    0.02052912   32.00  0.9768499  0.9536982
##    0.02052912   64.00  0.9764315  0.9528618
##    0.02052912  128.00  0.9751674  0.9503340
##    0.04898649    0.25  0.9743253  0.9486494
##    0.04898649    0.50  0.9768552  0.9537095
##    0.04898649    1.00  0.9785377  0.9570746
##    0.04898649    2.00  0.9789561  0.9579114
##    0.04898649    4.00  0.9772683  0.9545353
##    0.04898649    8.00  0.9772701  0.9545388
##    0.04898649   16.00  0.9789543  0.9579080
##    0.04898649   32.00  0.9755787  0.9511567
##    0.04898649   64.00  0.9768392  0.9536777
##    0.04898649  128.00  0.9755823  0.9511633
##    0.07744385    0.25  0.9739069  0.9478125
##    0.07744385    0.50  0.9764333  0.9528655
##    0.07744385    1.00  0.9764333  0.9528655
##    0.07744385    2.00  0.9760060  0.9520105
##    0.07744385    4.00  0.9772701  0.9545388
##    0.07744385    8.00  0.9776938  0.9553867
##    0.07744385   16.00  0.9764226  0.9528442
##    0.07744385   32.00  0.9785252  0.9570496
##    0.07744385   64.00  0.9764191  0.9528373
##    0.07744385  128.00  0.9734797  0.9469584
##    0.10590122    0.25  0.9734885  0.9469758
##    0.10590122    0.50  0.9772754  0.9545497
##    0.10590122    1.00  0.9760113  0.9520214
##    0.10590122    2.00  0.9760060  0.9520105
##    0.10590122    4.00  0.9781122  0.9562232
##    0.10590122    8.00  0.9781104  0.9562198
##    0.10590122   16.00  0.9776867  0.9553724
##    0.10590122   32.00  0.9760025  0.9520032
##    0.10590122   64.00  0.9743165  0.9486320
##    0.10590122  128.00  0.9743218  0.9486439
##    0.13435858    0.25  0.9747491  0.9494968
##    0.13435858    0.50  0.9764333  0.9528652
##    0.13435858    1.00  0.9755894  0.9511773
##    0.13435858    2.00  0.9768481  0.9536946
##    0.13435858    4.00  0.9768517  0.9537022
##    0.13435858    8.00  0.9797964  0.9595917
##    0.13435858   16.00  0.9772630  0.9545248
##    0.13435858   32.00  0.9743147  0.9486284
##    0.13435858   64.00  0.9764262  0.9528519
##    0.13435858  128.00  0.9751639  0.9503273
##    0.16281595    0.25  0.9751710  0.9503409
##    0.16281595    0.50  0.9760113  0.9520211
##    0.16281595    1.00  0.9772736  0.9545459
```

```
##   0.16281595    2.00  0.9772683  0.9545349
##   0.16281595    4.00  0.9772718  0.9545424
##   0.16281595    8.00  0.9789543  0.9579072
##   0.16281595   16.00  0.9751603  0.9503197
##   0.16281595   32.00  0.9747420  0.9494835
##   0.16281595   64.00  0.9755841  0.9511680
##   0.16281595  128.00  0.9755841  0.9511680
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.1343586 and C = 8.
```

```
ggplot(svmBase)
```

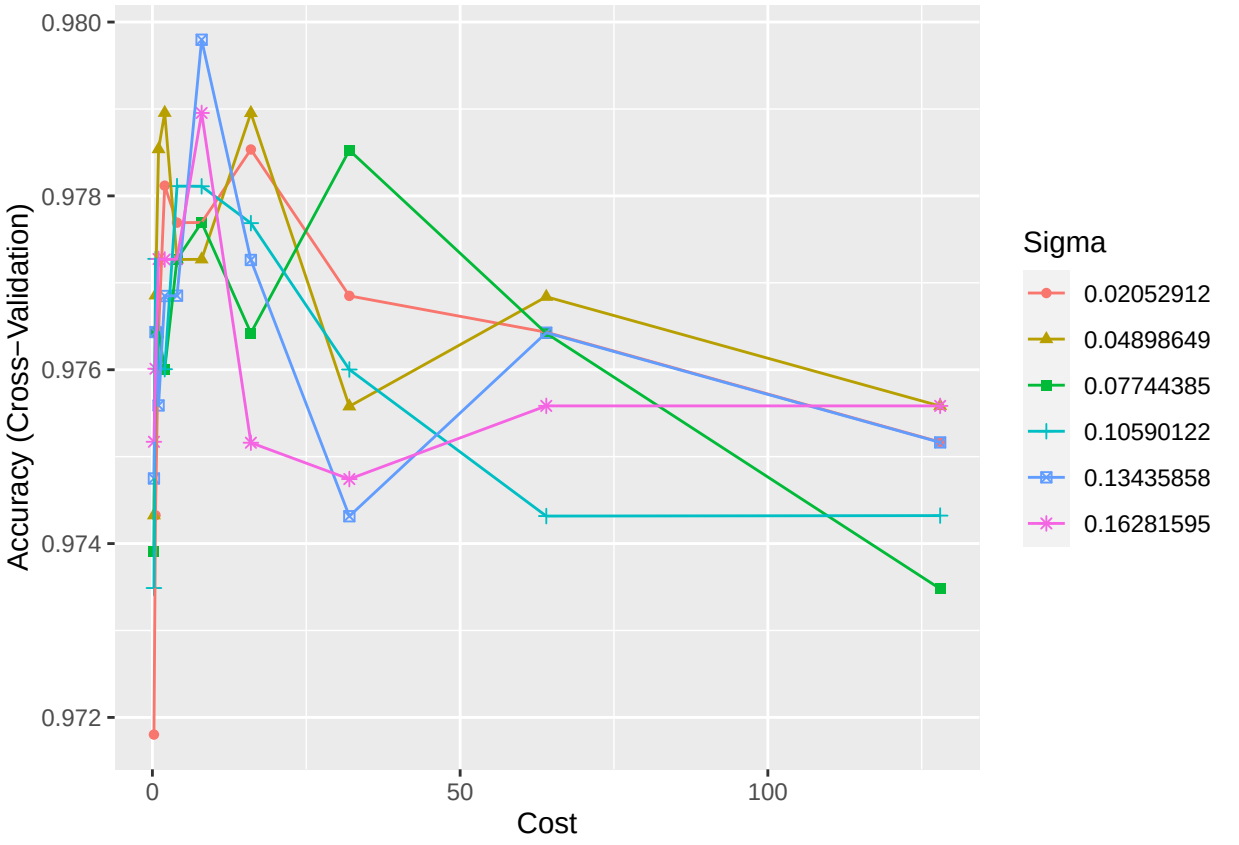

Figura 11: Evolución de accuracy para cada hiperparámetro durante el entrenamiento

Tras el entrenamiento `Figura 11`, se puede ver cómo `caret` ha probado 10 valores diferentes del hiperparámetro **C** y 6 de σ llevando a tener, como valores óptimos, $C = 8$ y $\sigma = 0.1343586$, lo que nos da una *accuracy* = 0.9797964. Con la gráfica parece que se puede intuir que con mayor coste, la *accuracy* se ve altamente afectada, lo que tiene sentido, ya que este hiperparámetro se utiliza como regularizador.

Entrenamiento con tuneGrid

Ahora, al utilizar *tuneGrid*, también vamos a actuar sobre el *tuning* de los hiperparámetros de nuestro modelo. Sin embargo, con esta variable, somos nosotros los que seleccionamos qué valores dar a cada hiperparámetro.

En este caso, vamos a pedir a `caret` que haga el entrenamiento probando con 10 valores diferentes de estos hiperparámetros, siguiendo el comportamiento con *tuneLength*, es decir, 10 valores para C y 6 para σ:

```
set.seed(206)
svmBaseGrid =
  train(label ~ ., data=trainingBase, method='svmRadialSigma',
        tuneGrid=expand.grid(sigma=seq(0.01,0.2,length.out=6),
                             C=c(1:8,50,100)),
        metric='Accuracy',trControl=trainControl,
        preProc=c("center","scale"))

svmBaseGrid
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 2376 samples
##   12 predictor
##    2 classes: 'female', 'male'
##
## Pre-processing: centered (12), scaled (12)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2138, 2139, 2138, 2139, 2138, 2138, ...
## Resampling results across tuning parameters:
##
##   sigma  C    Accuracy   Kappa
##   0.010    1  0.9747419  0.9494825
##   0.010    2  0.9751639  0.9503266
##   0.010    3  0.9764244  0.9528476
##   0.010    4  0.9768481  0.9536952
##   0.010    5  0.9768517  0.9537023
##   0.010    6  0.9768517  0.9537023
##   0.010    7  0.9772718  0.9545427
##   0.010    8  0.9772718  0.9545427
##   0.010   50  0.9772701  0.9545388
##   0.010  100  0.9760078  0.9520140
##   0.048    1  0.9785377  0.9570746
##   0.048    2  0.9789561  0.9579114
##   0.048    3  0.9768463  0.9536911
##   0.048    4  0.9776885  0.9553756
##   0.048    5  0.9776920  0.9553829
##   0.048    6  0.9776902  0.9553791
##   0.048    7  0.9768481  0.9536950
##   0.048    8  0.9768481  0.9536950
##   0.048   50  0.9755769  0.9511528
##   0.048  100  0.9768428  0.9536843
##   0.086    1  0.9764333  0.9528655
##   0.086    2  0.9755858  0.9511701
##   0.086    3  0.9776902  0.9553791
##   0.086    4  0.9776920  0.9553829
##   0.086    5  0.9781122  0.9562233
##   0.086    6  0.9781122  0.9562232
##   0.086    7  0.9772718  0.9545426
##   0.086    8  0.9781122  0.9562232
##   0.086   50  0.9768393  0.9536776
##   0.086  100  0.9738998  0.9477987
##   0.124    1  0.9760113  0.9520214
##   0.124    2  0.9764279  0.9528543
```

```
##   0.124    3  0.9768481  0.9536946
##   0.124    4  0.9768517  0.9537022
##   0.124    5  0.9776902  0.9553794
##   0.124    6  0.9772683  0.9545353
##   0.124    7  0.9781104  0.9562198
##   0.124    8  0.9789525  0.9579040
##   0.124   50  0.9751639  0.9503276
##   0.124  100  0.9755823  0.9511645
##   0.162    1  0.9772736  0.9545459
##   0.162    2  0.9772683  0.9545349
##   0.162    3  0.9764297  0.9528581
##   0.162    4  0.9772718  0.9545424
##   0.162    5  0.9781104  0.9562198
##   0.162    6  0.9793762  0.9587514
##   0.162    7  0.9789561  0.9579111
##   0.162    8  0.9793762  0.9587514
##   0.162   50  0.9760042  0.9520084
##   0.162  100  0.9755841  0.9511680
##   0.200    1  0.9772736  0.9545459
##   0.200    2  0.9768481  0.9536946
##   0.200    3  0.9764297  0.9528579
##   0.200    4  0.9776956  0.9553899
##   0.200    5  0.9781140  0.9562266
##   0.200    6  0.9785341  0.9570669
##   0.200    7  0.9793780  0.9587547
##   0.200    8  0.9797982  0.9595950
##   0.200   50  0.9755876  0.9511752
##   0.200  100  0.9755876  0.9511752
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.2 and C = 8.
```

```
ggplot(svmBaseGrid)
```

Tras el entrenamiento `Figura 12`, se puede ver cómo `caret` ha probado 10 valores diferentes del hiperparámetro **C** y 6 de σ llevando a tener, como valores óptimos, $C = 8$ y $\sigma = 0.2$, lo que nos da una *accuracy* $= 0.9797982$. De nuevo, y como era de esperar, obtenemos un gráfica semejante a la que obteníamos con *tuneLength*.

El resultado obtenido es ligeramente superior al anterior, para un valor de σ más elevado. Como conclusión de esta comparativa se puede sacar que *tuneLength* lo podemos utilizar como una primera aproximación al entrenamiento, probando, aleatoriamente con ciertos valores de los hiperparámetros, mientras que *tuneGrid* puede utilizarse para refinar más el modelo, o si se tiene muy claro qué efectos pueden provocar los valores de los hiperparámetros, ya que da mucho más control sobre el entrenamiento.

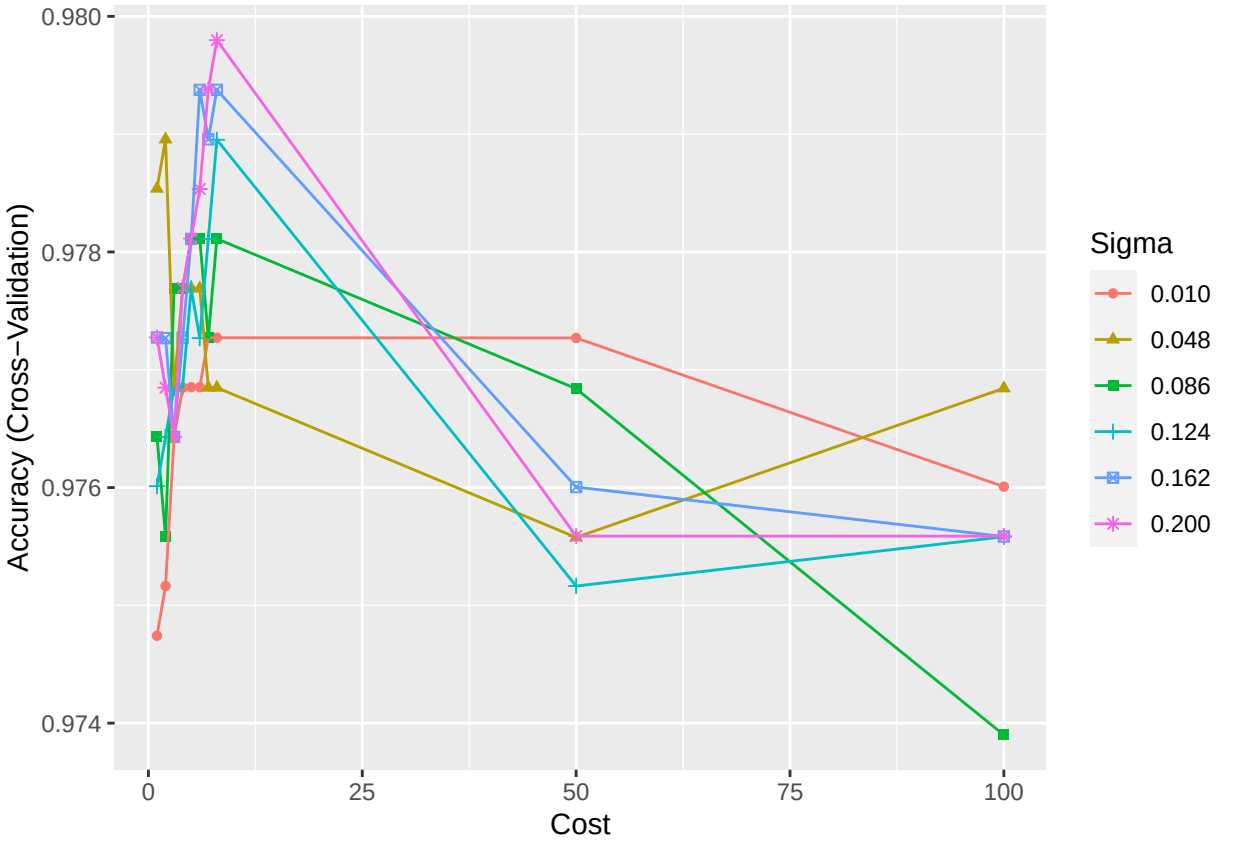

Figura 12: Evolución de accuracy para cada hiperparámetro durante el entrenamiento

4.1.2. SVM-RBF con selección de variables

A continuación, se va a utilizar el conjunto de datos sobre el que se ejecutó la selección de variables, tras la que nos quedamos solamente con *meanfun* e *IQR*.

El entrenamiento se realizará con *tunelength* por simplicidad, almacenándolo en una variable distinta a la del anterior modelo:

```
set.seed(206)
svmFS = train(label ~ ., data=trainingFS, method='svmRadialSigma',
              tuneLength=10, metric='Accuracy',
              trControl=trainControl, preProc=c("center","scale"))

svmFS
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 2376 samples
##    2 predictor
##    2 classes: 'female', 'male'
##
## Pre-processing: centered (2), scaled (2)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2138, 2139, 2138, 2139, 2138, 2138, ...
## Resampling results across tuning parameters:
##
##   sigma      C       Accuracy   Kappa
##   0.0995225    0.25  0.9646508  0.9293001
##   0.0995225    0.50  0.9650710  0.9301404
##   0.0995225    1.00  0.9663315  0.9326614
##   0.0995225    2.00  0.9663315  0.9326614
##   0.0995225    4.00  0.9650656  0.9301296
##   0.0995225    8.00  0.9646472  0.9292932
##   0.0995225   16.00  0.9650674  0.9301335
##   0.0995225   32.00  0.9671753  0.9343501
##   0.0995225   64.00  0.9671753  0.9343501
##   0.0995225  128.00  0.9667552  0.9335097
##   1.1231224    0.25  0.9684412  0.9368819
##   1.1231224    0.50  0.9697035  0.9394063
##   1.1231224    1.00  0.9709640  0.9419267
##   1.1231224    2.00  0.9701219  0.9402424
##   1.1231224    4.00  0.9709569  0.9419120
##   1.1231224    8.00  0.9701165  0.9402313
##   1.1231224   16.00  0.9701165  0.9402313
##   1.1231224   32.00  0.9705385  0.9410762
##   1.1231224   64.00  0.9692726  0.9385435
##   1.1231224  128.00  0.9692726  0.9385435
##   2.1467222    0.25  0.9701236  0.9402462
##   2.1467222    0.50  0.9692798  0.9385577
##   2.1467222    1.00  0.9692762  0.9385506
##   2.1467222    2.00  0.9684305  0.9368593
##   2.1467222    4.00  0.9696946  0.9393878
##   2.1467222    8.00  0.9696946  0.9393878
##   2.1467222   16.00  0.9684305  0.9368593
##   2.1467222   32.00  0.9663226  0.9326420
##   2.1467222   64.00  0.9659024  0.9318017
```

```
##   2.1467222  128.00  0.9663261  0.9326502
##   3.1703220    0.25  0.9688596  0.9377181
##   3.1703220    0.50  0.9671718  0.9343418
##   3.1703220    1.00  0.9680104  0.9360190
##   3.1703220    2.00  0.9684323  0.9368628
##   3.1703220    4.00  0.9701147  0.9402281
##   3.1703220    8.00  0.9667445  0.9334858
##   3.1703220   16.00  0.9663244  0.9326462
##   3.1703220   32.00  0.9667463  0.9334906
##   3.1703220   64.00  0.9675884  0.9351747
##   3.1703220  128.00  0.9654876  0.9309726
##   4.1939219    0.25  0.9688596  0.9377181
##   4.1939219    0.50  0.9680121  0.9360225
##   4.1939219    1.00  0.9680104  0.9360190
##   4.1939219    2.00  0.9675884  0.9351747
##   4.1939219    4.00  0.9667463  0.9334899
##   4.1939219    8.00  0.9671665  0.9343303
##   4.1939219   16.00  0.9667463  0.9334899
##   4.1939219   32.00  0.9663297  0.9326567
##   4.1939219   64.00  0.9654894  0.9309756
##   4.1939219  128.00  0.9680121  0.9360211
##   5.2175217    0.25  0.9680139  0.9360269
##   5.2175217    0.50  0.9671700  0.9343383
##   5.2175217    1.00  0.9675902  0.9351787
##   5.2175217    2.00  0.9680121  0.9360230
##   5.2175217    4.00  0.9667463  0.9334899
##   5.2175217    8.00  0.9667481  0.9334940
##   5.2175217   16.00  0.9659095  0.9318164
##   5.2175217   32.00  0.9663297  0.9326563
##   5.2175217   64.00  0.9675902  0.9351773
##   5.2175217  128.00  0.9680104  0.9360176
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 1.123122 and C = 1.
```

```
ggplot(svmFS)
```

En este caso se tiene un valor óptimo de *accuracy* = 0.9709640 para $C = 1$ y $\sigma = 0.123122$, algo inferior al obtenido con el anterior modelo, pero con muchas menos variables (recordemos que pasamos de 12 a solamente 2).

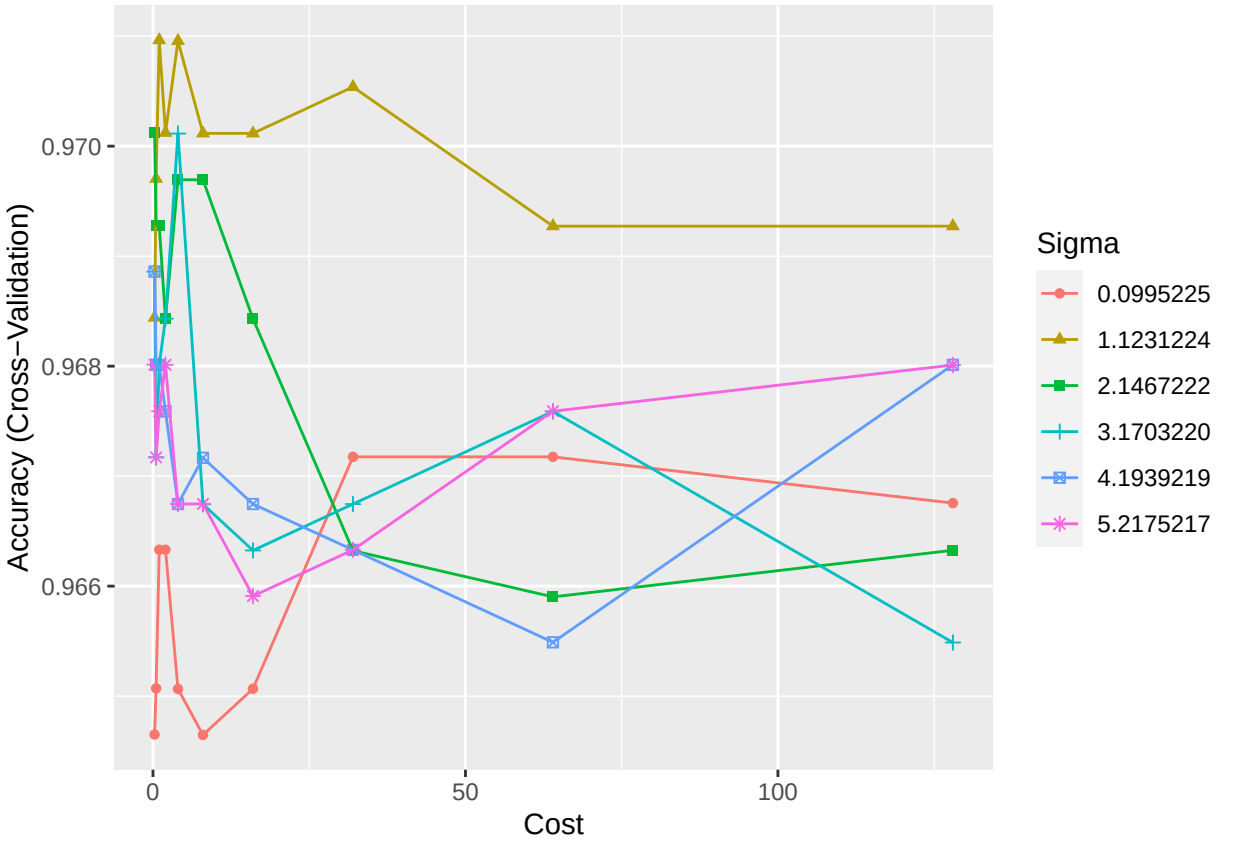

Figura 13: Evolución de accuracy para cada hiperparámetro durante el entrenamiento

4.1.3. SVM-RBF con PCA

Finalmente, se realiza el entrenamiento del modelo con PCA. En este caso, en lugar de utilizar el dataset resultante del apartado 3.2, y por obtener un resultado más honesto, se va a realizar el PCA por cada *fold*. Por defecto, `caret` se queda con las componentes que mantienen, al menos, el 95% de la varianza de los datos, por lo que solicitamos a `caret` que se quede con las primeras 6 componentes prioncipales, que es el resultado que obtuvimos en el apartado 3.2.

```
trainControlPCA <- trainControl(
  method='cv',
  number=10,
  allowParallel=TRUE,
  preProcOptions=list(pcaComp=6)
)
```

```
set.seed(206)
svmPCA = train(label ~ ., data=trainingBase, method='svmRadialSigma',
               tuneLength=10, metric='Accuracy',
               trControl=trainControlPCA,
               preProc=c("center","scale","pca"))

svmPCA
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 2376 samples
##   12 predictor
##    2 classes: 'female', 'male'
##
## Pre-processing: centered (12), scaled (12), principal component
##  signal extraction (12)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2138, 2139, 2138, 2139, 2138, 2138, ...
## Resampling results across tuning parameters:
##
##   sigma       C       Accuracy   Kappa
##   0.02601254    0.25  0.9469735  0.8939449
##   0.02601254    0.50  0.9507639  0.9015260
##   0.02601254    1.00  0.9520262  0.9040512
##   0.02601254    2.00  0.9579210  0.9158417
##   0.02601254    4.00  0.9612912  0.9225820
##   0.02601254    8.00  0.9612894  0.9225778
##   0.02601254   16.00  0.9612752  0.9225495
##   0.02601254   32.00  0.9625410  0.9250810
##   0.02601254   64.00  0.9633832  0.9267651
##   0.02601254  128.00  0.9642271  0.9284520
##   0.05683847    0.25  0.9528630  0.9057241
##   0.05683847    0.50  0.9558184  0.9116354
##   0.05683847    1.00  0.9600254  0.9200496
##   0.05683847    2.00  0.9617096  0.9234188
##   0.05683847    4.00  0.9612823  0.9225632
##   0.05683847    8.00  0.9654858  0.9309705
##   0.05683847   16.00  0.9642217  0.9284418
##   0.05683847   32.00  0.9646455  0.9292886
##   0.05683847   64.00  0.9629648  0.9259273
##   0.05683847  128.00  0.9638016  0.9276005
```

```
##   0.08766441    0.25  0.9553947  0.9107886
##   0.08766441    0.50  0.9583429  0.9166846
##   0.08766441    1.00  0.9617096  0.9234182
##   0.08766441    2.00  0.9625428  0.9250848
##   0.08766441    4.00  0.9637998  0.9275981
##   0.08766441    8.00  0.9633778  0.9267538
##   0.08766441   16.00  0.9646472  0.9292926
##   0.08766441   32.00  0.9629612  0.9259198
##   0.08766441   64.00  0.9637998  0.9275964
##   0.08766441  128.00  0.9595963  0.9191888
##   0.11849035    0.25  0.9583376  0.9166746
##   0.11849035    0.50  0.9587631  0.9175249
##   0.11849035    1.00  0.9608639  0.9217268
##   0.11849035    2.00  0.9638033  0.9276058
##   0.11849035    4.00  0.9633814  0.9267609
##   0.11849035    8.00  0.9625357  0.9250682
##   0.11849035   16.00  0.9629595  0.9259157
##   0.11849035   32.00  0.9654822  0.9309618
##   0.11849035   64.00  0.9625429  0.9250825
##   0.11849035  128.00  0.9650639  0.9301249
##   0.14931628    0.25  0.9587578  0.9175145
##   0.14931628    0.50  0.9591815  0.9183615
##   0.14931628    1.00  0.9621227  0.9242444
##   0.14931628    2.00  0.9629612  0.9259212
##   0.14931628    4.00  0.9633796  0.9267568
##   0.14931628    8.00  0.9629630  0.9259236
##   0.14931628   16.00  0.9654840  0.9309659
##   0.14931628   32.00  0.9654894  0.9309761
##   0.14931628   64.00  0.9633814  0.9267594
##   0.14931628  128.00  0.9612735  0.9225426
##   0.18014222    0.25  0.9587578  0.9175145
##   0.18014222    0.50  0.9600200  0.9200390
##   0.18014222    1.00  0.9638033  0.9276058
##   0.18014222    2.00  0.9625393  0.9250761
##   0.18014222    4.00  0.9629595  0.9259165
##   0.18014222    8.00  0.9646401  0.9292788
##   0.18014222   16.00  0.9650621  0.9301215
##   0.18014222   32.00  0.9638069  0.9276110
##   0.18014222   64.00  0.9621174  0.9242313
##   0.18014222  128.00  0.9595893  0.9191744
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.1493163 and C = 32.
```

```
ggplot(svmPCA)
```

Para el PCA obtenemos un valor óptimo de *accuracy* $= 0.9654894$ para $C = 32$ y $\sigma = 0.1493163$, inferior a los dos anteriores, pero ahora con 6 variables, por lo que parece que el PCA no está aportando ninguna ayuda al clasificador.

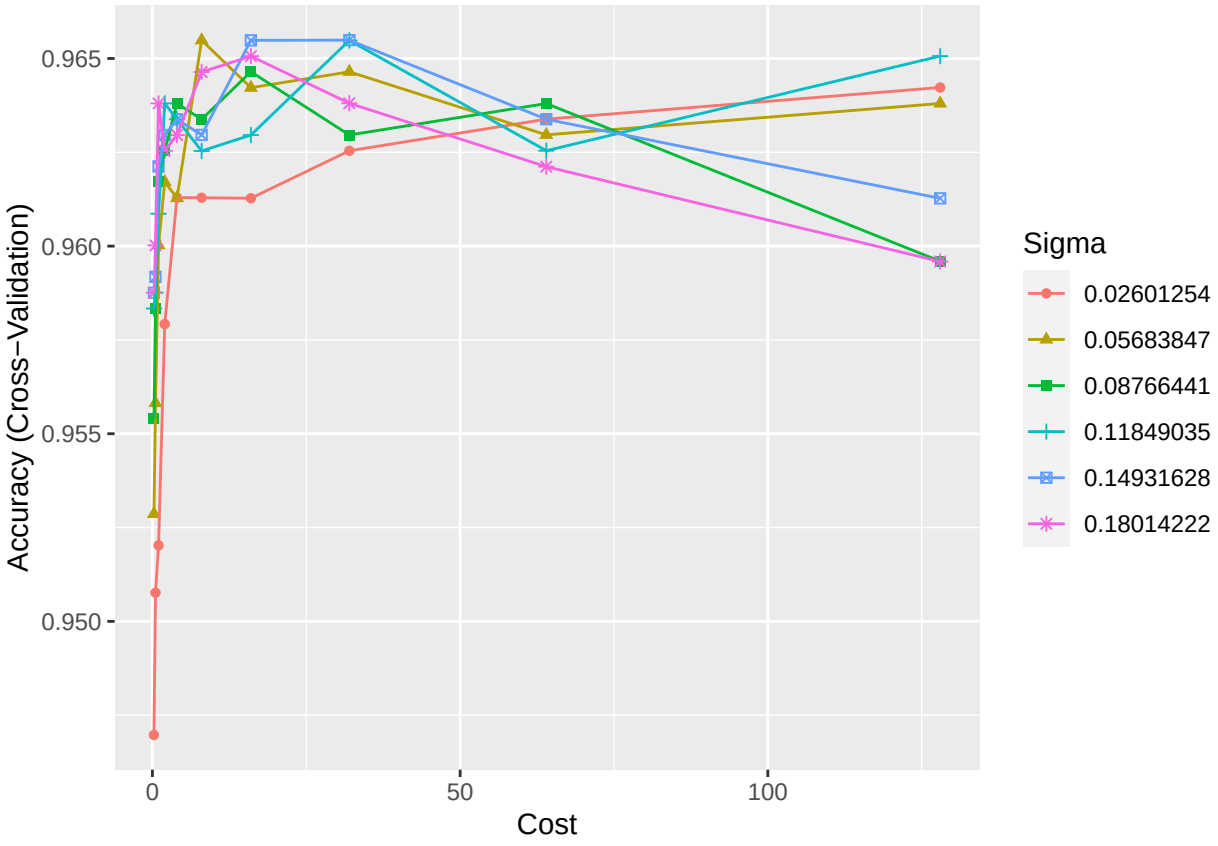

Figura 14: Evolución de accuracy para cada hiperparámetro durante el entrenamiento

4.2. Clasificador k-Nearest Neighbors

El algoritmo ***k-Nearest Neighbors (k-NN)*** es uno de los algoritmos más conocidos en Minería de Datos, que en este caso se va a utilizar para la clasificación de nuestro interés.

La forma en la que este algoritmo decide la etiqueta de una muestra (la clasifica) se basa en la clasificación de sus vecinos más cercanos. El método que utiliza es un "voto por mayoría" ya que asigna la etiqueta que más se repite entre los vecinos, aunque este término no es preciso al no requerir que la mayoría de los vecinos compartan la etiqueta, simplemente asigna la que más se repite entre ellos.

En este algoritmo entran en juego, por tanto, dos parámetros principalmente:

- **k:** Es el número de vecinos con los que comparar. Valores elevados de este parámetro aumenta el sesgo y reduce la varianza, mientras que lo contrario ocurre con valores bajos de este.
- **Distancia:** La forma de calcular la distancia entre el nuevo valor y sus vecinos, como por ejemplo la distancia euclídea, de Manhattan, o de Hamming, entre otras.

El paquete `caret` implementa, por defecto, la distancia euclídea (línea recta entre dos puntos) para este algoritmo, por lo que utilizaremos el parámetro *tuneLength* para que realice el entrenamiento con 10 valores distintos de **k** (o k_{max}).

$$d(x, y) = \sqrt{\sum_{i=1}^{k} (y_i - x_i)^2}$$

4.2.1. kNN con datos base

Ahora, se va a entrenar el algoritmo kNN con los datos tras el procesamiento pero sin la selección ni extracción de variables (*trainingBase*). El entrenamiento se realiza con las mismas condiciones que en la anterior sección 4.1, pero cambiando solamente el algoritmo.

```
set.seed(206)
knnBase = train(label ~ ., data=trainingBase, method='knn',
                tuneGrid=expand.grid(k=c(1:15)), metric='Accuracy',
                trControl=trainControl, preProc=c("center","scale"))
knnBase
```

```
## k-Nearest Neighbors
##
## 2376 samples
##   12 predictor
##    2 classes: 'female', 'male'
##
## Pre-processing: centered (12), scaled (12)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2138, 2139, 2138, 2139, 2138, 2138, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.9688649  0.9377286
##    2  0.9654912  0.9309807
##    3  0.9684483  0.9368952
##    4  0.9692922  0.9385833
##    5  0.9697053  0.9394088
##    6  0.9680246  0.9360481
##    7  0.9697106  0.9394203
##    8  0.9684483  0.9368955
```

```
##    9  0.9684501  0.9368993
##   10  0.9688667  0.9377325
##   11  0.9688702  0.9377397
##   12  0.9671842  0.9343674
##   13  0.9667623  0.9335236
##   14  0.9663421  0.9326835
##   15  0.9659255  0.9318501
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 7.
```

```
ggplot(knnBase)
```

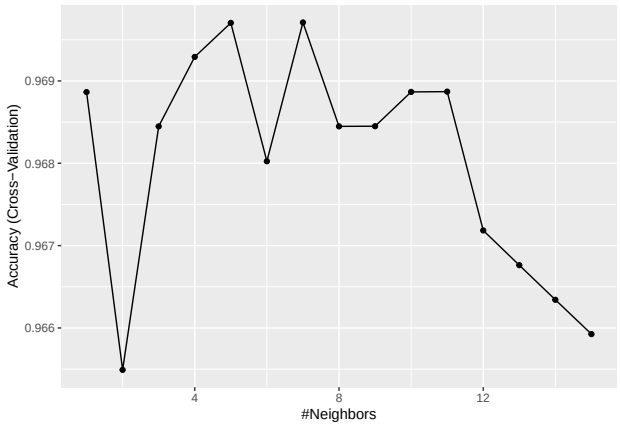

Figura 15: Evolución de accuracy para cada hiperparámetro durante el entrenamiento

El resultado obtenido es de *accuracy* = 0.9684501 para una $k = 7$, semejante al peor de los casos con SVM-RBF.

4.2.2. kNN con selección de variables

De igual manera, se mantienen mismas condiciones que en el apartado 4.1.2 para realizar el entrenamiento con el conjunto de datos tras la selección de variables (*trainingFS*).

```
set.seed(206)
knnFS = train(label ~ ., data=trainingFS, method='knn',
              tuneGrid=expand.grid(k=c(1:15)), metric='Accuracy',
              trControl=trainControl, preProc=c("center","scale"))

knnFS
```

```
## k-Nearest Neighbors
##
## 2376 samples
##    2 predictor
##    2 classes: 'female', 'male'
##
## Pre-processing: centered (2), scaled (2)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2138, 2139, 2138, 2139, 2138, 2138, ...
## Resampling results across tuning parameters:
##
```

```
##   k   Accuracy   Kappa
##    1  0.9570718  0.9141390
##    2  0.9570736  0.9141427
##    3  0.9633814  0.9267597
##    4  0.9629701  0.9259370
##    5  0.9646437  0.9292849
##    6  0.9659166  0.9318314
##    7  0.9654982  0.9309942
##    8  0.9638069  0.9276116
##    9  0.9659166  0.9318310
##   10  0.9642342  0.9284666
##   11  0.9659166  0.9318310
##   12  0.9663350  0.9326683
##   13  0.9671789  0.9343560
##   14  0.9676008  0.9352008
##   15  0.9659149  0.9318276
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 14.
```

```
ggplot(knnFS)
```

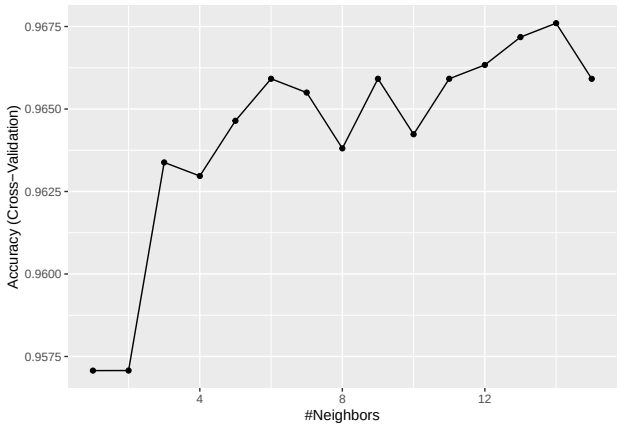

Figura 16: Evolución de accuracy para cada hiperparámetro durante el entrenamiento

En este caso se obtiene un resultado de *accuracy* = 0.9676008 para una $k = 14$, que indica la relación que se veía con el anterior modelo (la diferencia entre usar todas las variables predictoras y solamente las dos extraídas es reducida desde un punto de vista de la validación del entrenamiento).

4.2.3. kNN con PCA

Y finalmente, se realiza el entrenamiento con la extracción de variables mediante PCA para cada uno de los *folds*.

```
set.seed(206)
knnPCA = train(label ~ ., data=trainingBase, method='knn',
               tuneGrid=expand.grid(k=c(1:15)), metric='Accuracy',
               trControl=trainControlPCA,
               preProc=c("center","scale","pca"))

knnPCA
```

```
## k-Nearest Neighbors
##
## 2376 samples
##   12 predictor
##    2 classes: 'female', 'male'
##
## Pre-processing: centered (12), scaled (12), principal component
##  signal extraction (12)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2138, 2139, 2138, 2139, 2138, 2138, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.9545437  0.9090834
##    2  0.9503349  0.9006653
##    3  0.9532921  0.9065802
##    4  0.9537069  0.9074091
##    5  0.9570771  0.9141504
##    6  0.9579175  0.9158306
##    7  0.9579228  0.9158434
##    8  0.9566587  0.9133159
##    9  0.9558184  0.9116340
##   10  0.9558148  0.9116269
##   11  0.9549745  0.9099462
##   12  0.9558166  0.9116311
##   13  0.9562368  0.9124714
##   14  0.9553929  0.9107829
##   15  0.9553965  0.9107898
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 7.
```

```
ggplot(knnPCA)
```

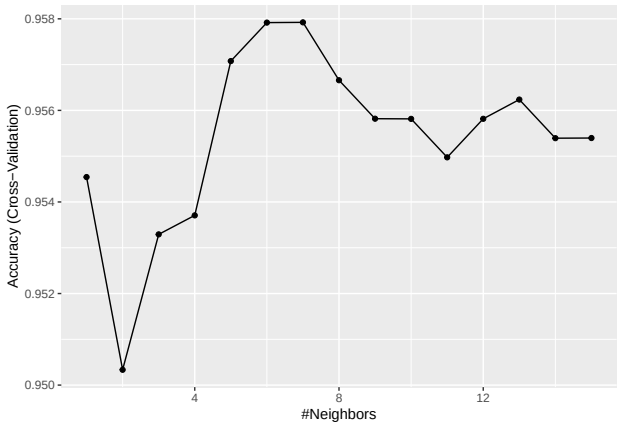

Figura 17: Evolución de accuracy para cada hiperparámetro durante el entrenamiento

5. Comparativa de resultados

En primer lugar, y por completitud del análisis, se va a revisar qué variables han tenido la mayor importancia durante el entrenamiento para cada modelo. Para ello, hacemos uso de la función `varImp` y mostramos gráficamente el porcentaje de importancia en cada caso:

```
ggplotVarImportance <- function(m,mytitle){
  p <-    ggplot(data = varImp(object=m)) + theme(legend.position = "none") +
                ggtitle(mytitle)
  return(p)
}
```

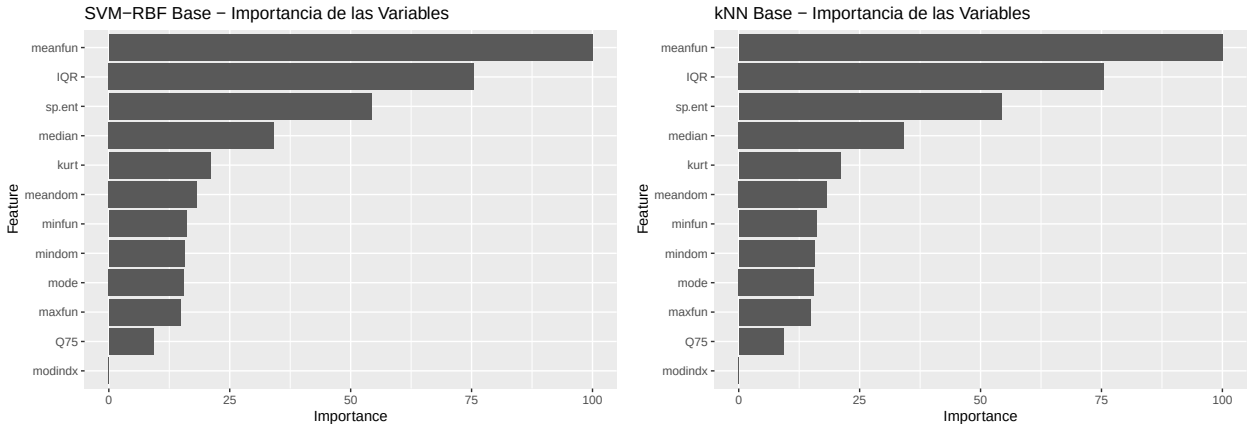

Figura 18: Importancia de cada variables para modelos base

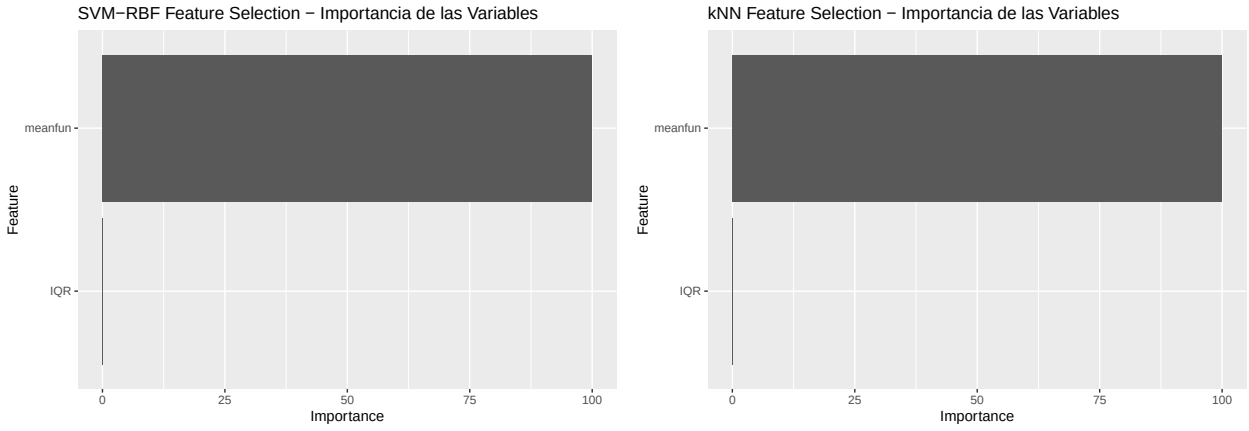

Figura 19: Importancia de cada variables para modelos con selección de variables

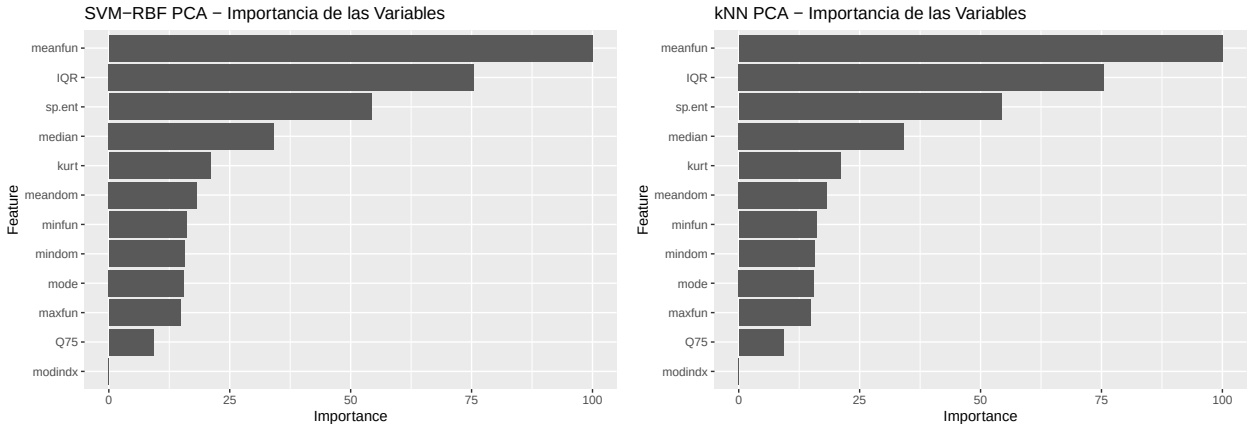

Figura 20: Importancia de cada variables para con PCA

Estas gráficas (`Figura 18-20`) sustentan lo que se decía al inicio del trabajo (y que también se presenta como conclusión por los autores del dataset): las cariables *meanfun* e *IQR* son las más determinantes a la hora de la clasificación, y, en el caso de extracción de vairables *meanfun* adquiere el 100% de importancia, ya que es la primera caracterísitica a considerar de cara a decidir la clase.

Ahora, una vez visto qué variables afectan en mayor medida al entrenamiento, pasamos a comparar los modelos a través de una comparativa directa de los resultados de evaluación interna (de igual manera que en el enunciado del ejercicio). Esto se puede realizar porque, al igual que se ha comentado durante este trabajo, la semilla de aleatorización es siempre la misma, por lo que las particiones (*folds*) siempre van a realizarse de igual manera.

Para ello, vamos a utilizar la función `resamples`:

```
resamps=resamples(list(svmBase=svmBase,svmFS=svmFS,svmPCA=svmPCA,
                       knnBase=knnBase,knnFS=knnFS,knnPCA=knnPCA),
                  metric="Accuracy")
summary(resamps)
```

```
##
## Call:
## summary.resamples(object = resamps)
##
## Models: svmBase, svmFS, svmPCA, knnBase, knnFS, knnPCA
## Number of resamples: 10
##
## Accuracy
##               Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## svmBase 0.9705882 0.9757958 0.9789030 0.9797964 0.9831933 0.9873950    0
## svmFS   0.9579832 0.9673747 0.9705262 0.9709640 0.9747899 0.9831224    0
## svmPCA  0.9327731 0.9621849 0.9683450 0.9654894 0.9736597 0.9831933    0
## knnBase 0.9453782 0.9663866 0.9705882 0.9697106 0.9778213 0.9831224    0
## knnFS   0.9495798 0.9621849 0.9683450 0.9676008 0.9747633 0.9831224    0
## knnPCA  0.9453782 0.9537815 0.9578946 0.9579228 0.9608941 0.9746835    0
##
## Kappa
##               Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## svmBase 0.9411765 0.9515917 0.9578006 0.9595917 0.9663866 0.9747899    0
## svmFS   0.9159664 0.9347493 0.9410466 0.9419267 0.9495798 0.9662441    0
```

```
## svmPCA  0.8655462 0.9243697 0.9366899 0.9309761 0.9473188 0.9663866    0
## knnBase 0.8907563 0.9327731 0.9411765 0.9394203 0.9556386 0.9662417    0
## knnFS   0.8991597 0.9243697 0.9366899 0.9352008 0.9495246 0.9662441    0
## knnPCA  0.8907563 0.9075630 0.9157733 0.9158434 0.9217882 0.9493770    0
```

La gráfica nos indica que durante la validación, el modelo `svmBase` supera al resto, pero es necesario comprobar su rendimiento con datos de test, por lo que,finalmente, se va a realizar una comparación desde el punto de vista de la predicción que realizan estos clasificadores para el set de test. Para las predicciones se utiliza la función `predict`, que solamente nos va a permitir trabajar con la opción `type="raw"` o `type="prob"` por los clasificadores que se han utilizado y por ser una clasificación binaria.

Cabe destacar que el modelo de `caret` almacena tanto el preprocesamiento con el que lo hemos entrenado (variable `caret`) como las variales a utilizar. Por lo tanto, durante la ejecución de `predict` se aplicará lo mismo sobre nuestro conjunto de datos de test (lo normalizará en todos los casos, se quedará con las variables que corresponda en cada caso y aplicará el PCA en el modelo entrenado para ello). Por lo tanto, podemos pasarle a esta función el conjunto de datos de test inicial.

A continuación (`Figura 21-26`), por cada modelo, se presenta la matriz de confusión obtenida que nos indica, por cada clase, el número de muestras bien y mal clasificadas para todo el set de datos de test, además del *accuracy* asociado a dicha predicción.

```
ggplotConfusionMatrix <- function(m){
  mytitle <- paste("Accuracy", percent_format()(m$overall[1]),
                   "Kappa", percent_format()(m$overall[2]))
  p <-
    ggplot(data = as.data.frame(m$table) ,
           aes(x = Reference, y = Prediction)) +
    geom_tile(aes(fill = log(Freq)), colour = "white") +
    scale_fill_gradient(low = "bisque", high = "bisque3") +
    geom_text(aes(x = Reference, y = Prediction, label = Freq)) +
    theme(legend.position = "none") +
    ggtitle(mytitle)
  return(p)
}
```

```
svmBase_predictions <- predict(svmBase, newdata=testing, type="raw")
cMsvmBase <- confusionMatrix(reference = testing$label,
                             data = svmBase_predictions, mode='everything',
                             positive='female')
ggplotConfusionMatrix(cMsvmBase)
```

```
svmFS_predictions <- predict(svmFS, newdata=testing, type="raw")
cMsvmFS <- confusionMatrix(reference = testing$label,
                           data = svmFS_predictions, mode='everything',
                           positive='female')
ggplotConfusionMatrix(cMsvmFS)
```

```
svmPCA_predictions <- predict(svmPCA, newdata=testing, type="raw")
cMsvmPCA <- confusionMatrix(reference = testing$label,
                            data = svmPCA_predictions, mode='everything',
                            positive='female')
ggplotConfusionMatrix(cMsvmPCA)
```

```
knnBase_predictions <- predict(knnBase, newdata=testing, type="raw")
cMknnBase <- confusionMatrix(reference = testing$label,
                             data = knnBase_predictions, mode='everything',
                             positive='female')
```

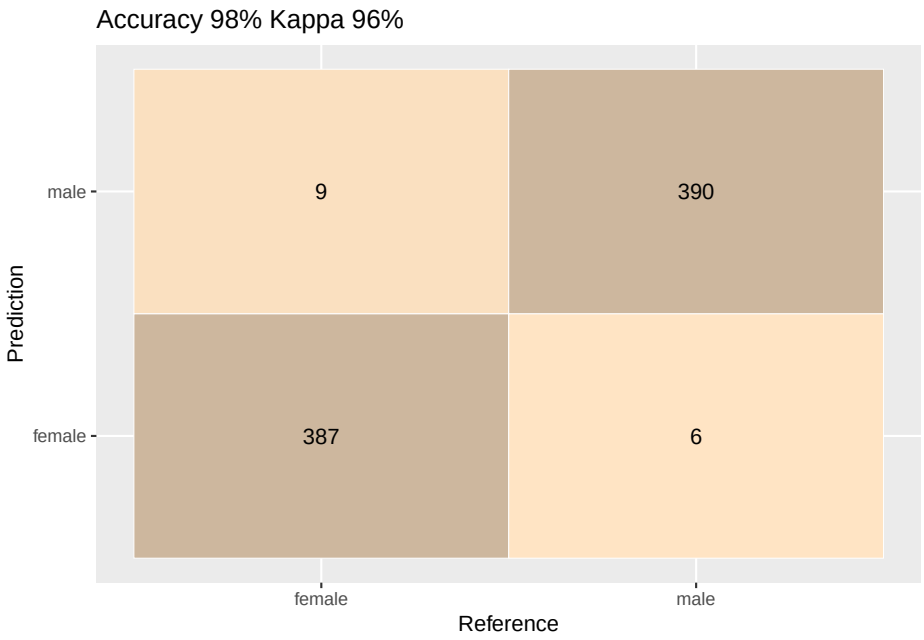

Figura 21: Matriz de confusión de predicciones para svmBase

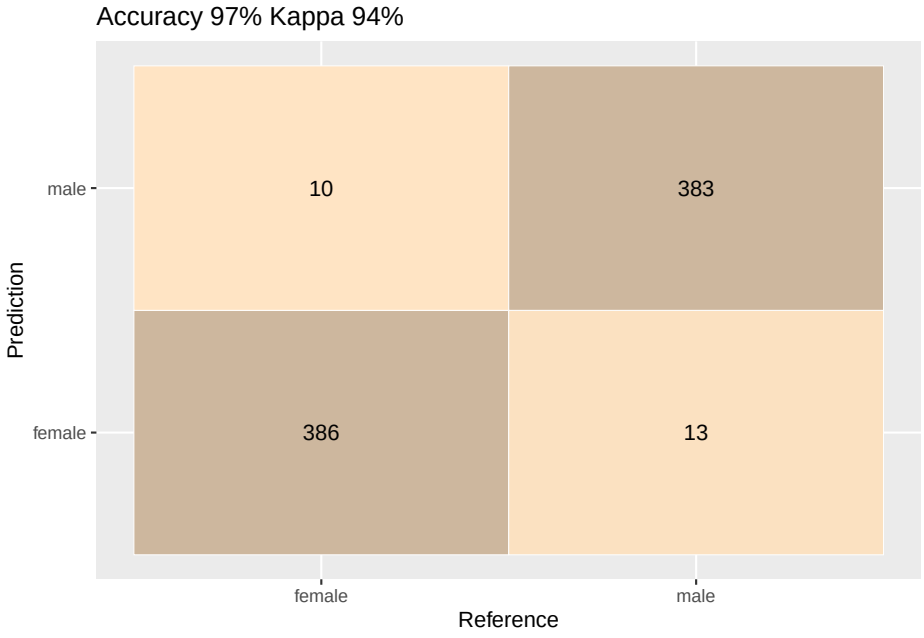

Figura 22: Matriz de confusión de predicciones para svmFS

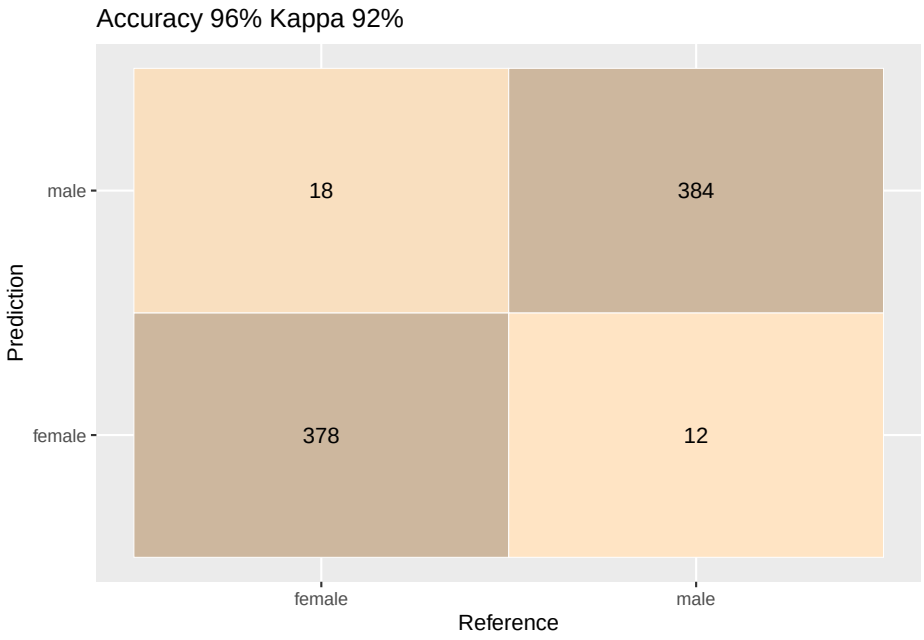

Figura 23: Matriz de confusión de predicciones para svmPCA

```
ggplotConfusionMatrix(cMknnBase)
```

```
knnFS_predictions <- predict(knnFS, newdata=testing, type="raw")
cMknnFS <- confusionMatrix(reference = testing$label,
                           data = knnFS_predictions, mode='everything',
                           positive='female')
ggplotConfusionMatrix(cMknnFS)
```

```
knnPCA_predictions <- predict(knnPCA, newdata=testing, type="raw")
cMknnPCA <- confusionMatrix(reference = testing$label,
                            data = knnPCA_predictions, mode='everything',
                            positive='female')
ggplotConfusionMatrix(cMknnPCA)
```

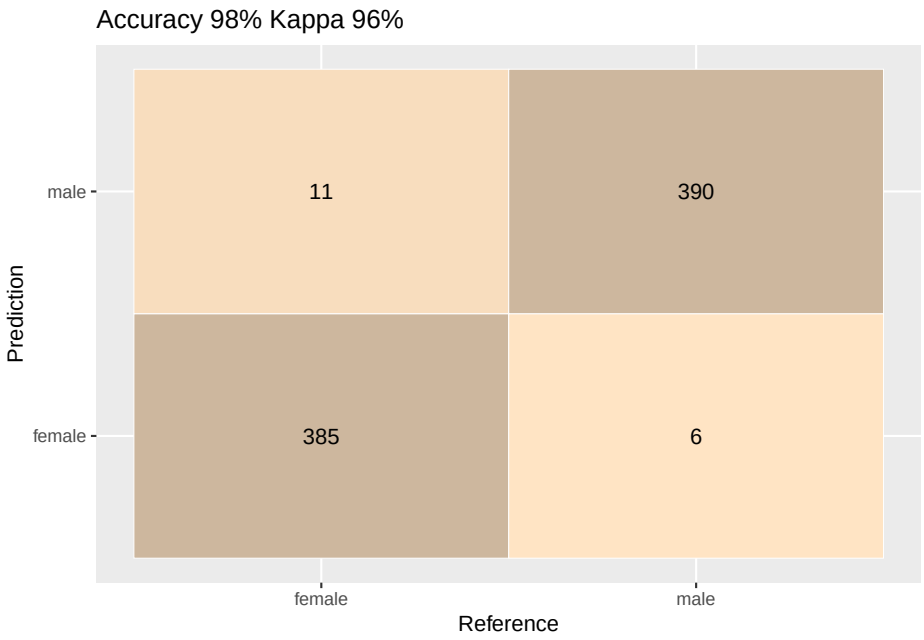

Figura 24: Matriz de confusión de predicciones para knnBase

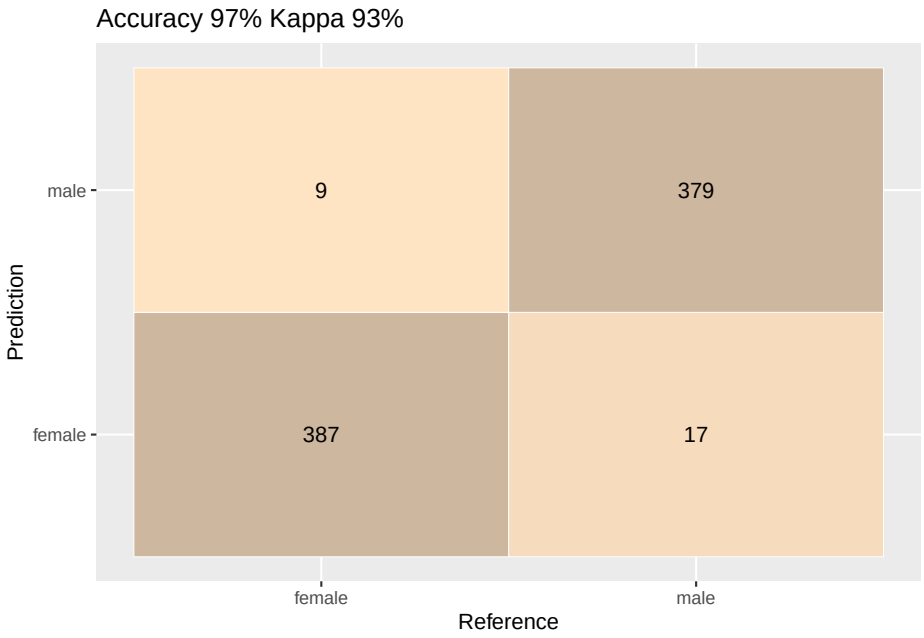

Figura 25: Matriz de confusión de predicciones para knnFS

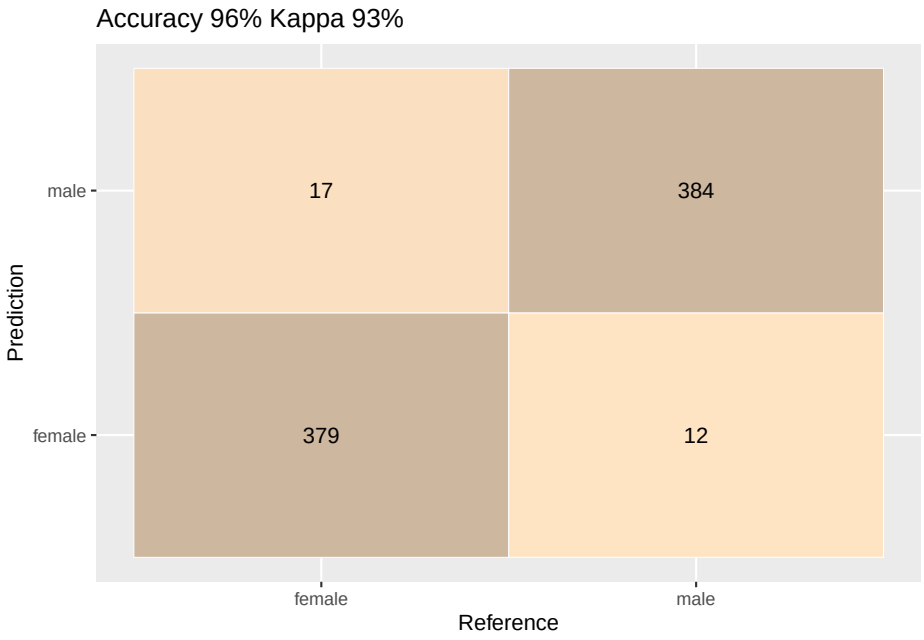

Figura 26: Matriz de confusión de predicciones para knnPCA

Tras revisar las matrices de confusión, cabe destacar que todos los algoritmos obtienen una *accuracy* elevada, quedando claro que los modelos entrenados con el dataset "base" se comportan mejor que el resto. De esta manera, tiene sentido hacer una última comparación, esta vez entre los modelos `svmBase` y `knnBase`, que son los que mejor se han comportado durante el test.

Para ello, se va a utilizar de nuevo la función `resamples` y se va a mostrar la gráfica Bland-Altman (`Figura 27`) como en le enunciado del ejercicio. Posteriormente se calcula la diferencia de las medias de rendmiento de cada clasificador, a través de la función `diff`:

```
resamps2=resamples(list(svmBase=svmBase, knnBase=knnBase), metric="Accuracy")
xyplot(resamps2,what="BlandAltman")
```

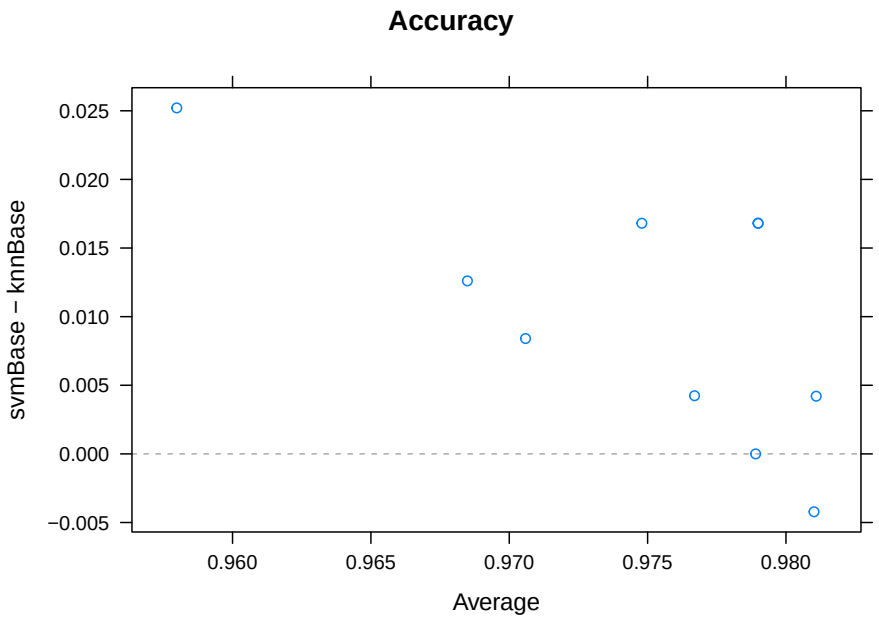

Figura 27: Blan-Altman para los dos mejores modelos clasificadores

```
diffs<-diff(resamps2)
summary(diffs)
```

```
##
## Call:
## summary.diff.resamples(object = diffs)
##
## p-value adjustment: bonferroni
## Upper diagonal: estimates of the difference
## Lower diagonal: p-value for H0: difference = 0
##
## Accuracy
##         svmBase  knnBase
## svmBase          0.01009
## knnBase 0.006755
##
## Kappa
##         svmBase  knnBase
## svmBase          0.02017
## knnBase 0.006756
```

Como se puede observar, el $p - value = 0.006755$, un orden de magnitud menor que 0.05, por lo que se puede considerar que ambos clasificadores son significativamente diferentes desde el punto de vista estadístico.