

Machine Learning (M0-163)

Tercera prueba de evaluación continua - Cancer de mama

María Plaza García

9 de Juni, 2020

Índice

1. Introducción	3
2. Objetivo	3
3. Trabajando con los datos	3
3.1. Lectura de datos	3
3.2. Exploración y preparación de los datos	3
3.3. Partición de los datos en training/test	10
4. Aplicación de cada uno de los algoritmos para la clasificación	11
4.1. k-Nearest Neighbour	11
4.1.1. Transformación de los datos	11
4.1.2. Entrenar el modelo	11
4.1.3. Predicción y Evaluación del algoritmo	11
4.2. Naive Bayes	15
4.2.1. Transformación de los datos	15
4.2.2. Entrenar el modelo	15
4.2.3. Predicción y Evaluación del algoritmo	15
4.3. Artificial Neural Network	20
4.3.1. Transformación de los datos	20
4.3.2. Entrenar el modelo	20
4.3.3. Predicción y Evaluación del algoritmo	21
4.3.4. Paquete <i>caret</i> : modelo nnet	24
4.4. Support Vector Machine	27
4.4.1. Transformación de los datos	27
4.4.2. Entrenar el modelo	27
4.4.3. Predicción y Evaluación del algoritmo.	28
4.4.4. Paquete <i>caret</i>	29
4.5. Árbol de Decisión	32
4.5.1. Transformación de los datos	32
4.5.2. Entrenar el modelo	32
4.5.3. Predicción y Evaluación del algoritmo	34
4.5.4. Paquete <i>caret</i>	40
4.6. Random Forest	41
4.6.1. Transformación de los datos	41
4.6.2. Entrenar el modelo	42
4.6.3. Predicción y Evaluación del algoritmo	44
4.6.4. Paquete <i>caret</i>	46
5. Conclusión y discusión	47
5.1. Rendimiento, interpretabilidad, ... de los algoritmos para el problema tratado	47

5.2. Decisión del modelo más apropiado	47
--	----

Bibliografía	48
---------------------	-----------

Repositorio Github:

https://github.com/mariaplaza/PEC3_MachineLearning

1. Introducción

En esta PEC vamos a resolver un análisis para detectar el cáncer en las mediciones de células biopsiadas de mujeres con masas mamarias anormales.

Utilizaremos datos de cáncer de mama que incluyen mediciones de imágenes digitalizadas de aspiración con aguja fina de una masa mamaria. Los valores representan características de los núcleos celulares presentes en la imagen digital.

En el fichero BreastCancer1.csv estan los datos sobre el cáncer de mama de 569 casos de biopsias de cáncer, cada uno con 32 características. La primera característica es un número de identificación, después son 30 mediciones de laboratorio con valores numéricos y por último, esta el diagnóstico. El diagnóstico se codifica como M para indicar maligno o B para indicar benigno.

Los ficheros necesarios para realizar la PEC estan en formato csv. Se encuentran dentro de mi repositorio github (Github 2016), asi como cada uno de los archivos creados para la realización de esta PEC:

https://github.com/mariaplaza/PEC3_Machine_Learning

2. Objetivo

En esta PEC se analizan estos datos mediante la implementación de los diferentes algoritmos estudiados: k-Nearest Neighbour, Naive Bayes, Artificial Neural Network, Support Vector Machine, Arbol de Decisión y Random Forest para diagnosticar el tipo de cáncer de mama.

3. Trabajando con los datos

3.1. Lectura de datos

Para facilitar la reproducibilidad del informe, se han incluido varios parámetros en el encabezado YAML del documento cuyos valores se pueden establecer cuando se procesa el informe. Se ha incluido tanto la semilla que emplearemos más tarde en la creación de los datos de test y de entrenamiento así como los nombres de los archivos y la ruta de acceso, de esta forma podemos leer los datos con el siguiente código:

```
# Ahora ya se importan los datos a formato data.frame
library(readr)
m.file <- ifelse(params$folder.data=="",
                params$file,
                file.path(params$folder.data,params$file))
data <- read.csv(file=m.file)
```

3.2. Exploración y preparación de los datos

Comprobamos que el dataset está completo y se presentan los seis primeros registros:

```
dim(data)
```

```
[1] 569  32
```

```
class(data)
```

```
[1] "data.frame"
```

```
head(data)
```

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
1	842302	17.99	10.38	122.80	1001.0	0.11840
2	842517	20.57	17.77	132.90	1326.0	0.08474
3	84300903	19.69	21.25	130.00	1203.0	0.10960
4	84348301	11.42	20.38	77.58	386.1	0.14250
5	84358402	20.29	14.34	135.10	1297.0	0.10030

6	843786	12.45	15.70	82.57	477.1	0.12780
	compactness_mean	concavity_mean	concave.points_mean	symmetry_mean		
1	0.27760	0.3001	0.14710	0.2419		
2	0.07864	0.0869	0.07017	0.1812		
3	0.15990	0.1974	0.12790	0.2069		
4	0.28390	0.2414	0.10520	0.2597		
5	0.13280	0.1980	0.10430	0.1809		
6	0.17000	0.1578	0.08089	0.2087		
	fractal_dimension_mean	radius_se	texture_se	perimeter_se	area_se	smoothness_se
1	0.07871	1.0950	0.9053	8.589	153.40	0.006399
2	0.05667	0.5435	0.7339	3.398	74.08	0.005225
3	0.05999	0.7456	0.7869	4.585	94.03	0.006150
4	0.09744	0.4956	1.1560	3.445	27.23	0.009110
5	0.05883	0.7572	0.7813	5.438	94.44	0.011490
6	0.07613	0.3345	0.8902	2.217	27.19	0.007510
	compactness_se	concavity_se	concave.points_se	symmetry_se	fractal_dimension_se	
1	0.04904	0.05373	0.01587	0.03003		0.006193
2	0.01308	0.01860	0.01340	0.01389		0.003532
3	0.04006	0.03832	0.02058	0.02250		0.004571
4	0.07458	0.05661	0.01867	0.05963		0.009208
5	0.02461	0.05688	0.01885	0.01756		0.005115
6	0.03345	0.03672	0.01137	0.02165		0.005082
	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	
1	25.38	17.33	184.60	2019.0		0.1622
2	24.99	23.41	158.80	1956.0		0.1238
3	23.57	25.53	152.50	1709.0		0.1444
4	14.91	26.50	98.87	567.7		0.2098
5	22.54	16.67	152.20	1575.0		0.1374
6	15.47	23.75	103.40	741.6		0.1791
	compactness_worst	concavity_worst	concave.points_worst	symmetry_worst		
1	0.6656	0.7119	0.2654	0.4601		
2	0.1866	0.2416	0.1860	0.2750		
3	0.4245	0.4504	0.2430	0.3613		
4	0.8663	0.6869	0.2575	0.6638		
5	0.2050	0.4000	0.1625	0.2364		
6	0.5249	0.5355	0.1741	0.3985		
	fractal_dimension_worst	diagnosis				
1	0.11890	M				
2	0.08902	M				
3	0.08758	M				
4	0.17300	M				
5	0.07678	M				
6	0.12440	M				

Nuestro conjunto de datos, *BreastCancer1.csv*, esta formado por 569 registros, con valores de 32 características.

La primera variable es una variable numerica denominada *id*. Ya que es un simple identificador de cada paciente en los datos, no proporciona informacion útil y necesitamos excluirlo del modelo. Como está localizado en la primera columna, podemos excluirlo con el siguiente codigo, eliminando del *data.frame* la columna número 1.

```
# drop the id feature
data <- data[,-1]
```

La última variable, “diagnosis”, es de particular interés, ya que es el resultado que esperamos predecir. Esta característica indica si la muestra es de tipo benigno “B” o maligno “M”. La salida `table()` indica que las muestras NA son benignas mientras que NA son malignas. La clasificación existente según la diagnosis:

```
# table of diagnosis
table(data$diagnosis)
```

```
B    M
357 212
```

La mayoría de los clasificadores en *Machine Learning* requieren que esta característica objetivo esté codificada como factor, por lo que recodificamos la variable, de tipo caracter, en tipo factor.

```
class(data$diagnosis)
```

```
[1] "character"
```

```
data$diagnosis <- factor(data$diagnosis, levels = c("B", "M"),
                          labels = c("Benigno", "Maligno"))
```

```
class(data$diagnosis)
```

```
[1] "factor"
```

Ahora, cuando miramos la salida `prop.table()`, notamos que los valores han sido etiquetados como 'Benigno' y 'Maligno'.

```
# table or proportions with more informative labels
round(prop.table(table(data$diagnosis)) * 100, digits = 1)
```

```
Benigno Maligno
  62.7      37.3
```

Las características restantes de 0 son todas numéricas y, como era de esperar, consisten en mediciones diferentes de 30 características. Por ejemplo:

```
# summarize three numeric features
kable(summary(data[c("radius_mean", "area_mean", "smoothness_mean"])))
```

	radius_mean	area_mean	smoothness_mean
Min. :	6.981	143.5	0.05263
1st Qu.:	11.700	420.3	0.08637
Median :	13.370	551.1	0.09587
Mean :	14.127	654.9	0.09636
3rd Qu.:	15.780	782.7	0.10530
Max. :	28.110	2501.0	0.16340

La estructura final es:

```
str(data)
```

```
'data.frame':  569 obs. of  31 variables:
 $ radius_mean      : num  18 20.6 19.7 11.4 20.3 ...
 $ texture_mean     : num  10.4 17.8 21.2 20.4 14.3 ...
 $ perimeter_mean   : num  122.8 132.9 130 77.6 135.1 ...
 $ area_mean        : num  1001 1326 1203 386 1297 ...
 $ smoothness_mean  : num  0.1184 0.0847 0.1096 0.1425 0.1003 ...
 $ compactness_mean : num  0.2776 0.0786 0.1599 0.2839 0.1328 ...
 $ concavity_mean   : num  0.3001 0.0869 0.1974 0.2414 0.198 ...
 $ concave.points_mean : num  0.1471 0.0702 0.1279 0.1052 0.1043 ...
 $ symmetry_mean    : num  0.242 0.181 0.207 0.26 0.181 ...
 $ fractal_dimension_mean : num  0.0787 0.0567 0.06 0.0974 0.0588 ...
 $ radius_se        : num  1.095 0.543 0.746 0.496 0.757 ...
 $ texture_se       : num  0.905 0.734 0.787 1.156 0.781 ...
 $ perimeter_se     : num  8.59 3.4 4.58 3.44 5.44 ...
 $ area_se          : num  153.4 74.1 94 27.2 94.4 ...
 $ smoothness_se    : num  0.0064 0.00522 0.00615 0.00911 0.01149 ...
```

```

$ compactness_se      : num  0.049 0.0131 0.0401 0.0746 0.0246 ...
$ concavity_se        : num  0.0537 0.0186 0.0383 0.0566 0.0569 ...
$ concave.points_se   : num  0.0159 0.0134 0.0206 0.0187 0.0188 ...
$ symmetry_se         : num  0.03 0.0139 0.0225 0.0596 0.0176 ...
$ fractal_dimension_se : num  0.00619 0.00353 0.00457 0.00921 0.00511 ...
$ radius_worst        : num  25.4 25 23.6 14.9 22.5 ...
$ texture_worst       : num  17.3 23.4 25.5 26.5 16.7 ...
$ perimeter_worst     : num  184.6 158.8 152.5 98.9 152.2 ...
$ area_worst          : num  2019 1956 1709 568 1575 ...
$ smoothness_worst    : num  0.162 0.124 0.144 0.21 0.137 ...
$ compactness_worst   : num  0.666 0.187 0.424 0.866 0.205 ...
$ concavity_worst     : num  0.712 0.242 0.45 0.687 0.4 ...
$ concave.points_worst : num  0.265 0.186 0.243 0.258 0.163 ...
$ symmetry_worst      : num  0.46 0.275 0.361 0.664 0.236 ...
$ fractal_dimension_worst : num  0.1189 0.089 0.0876 0.173 0.0768 ...
$ diagnosis           : Factor w/ 2 levels "Benigno","Maligno": 2 2 2 2 2 2 2 2 2 2 ...

```

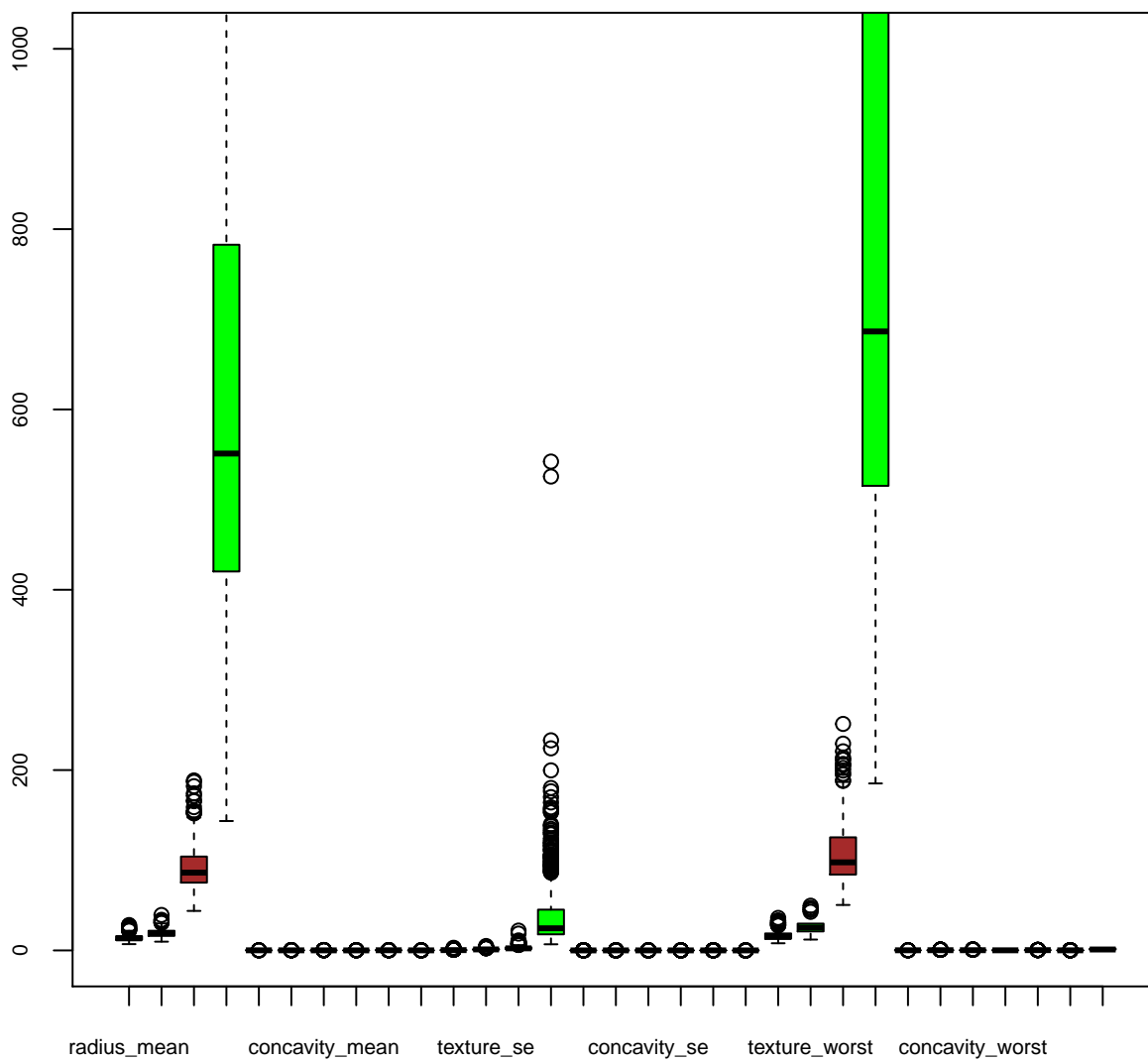
Veamos como se comportan las características estudiadas según la clasificación de nuestra principal variable, diagnóstico. Como veremos, para poder distinguir adecuadamente las variables, debemos cambiar el límite de los ejes, ya que los valores son muy diferentes según las variables:

```

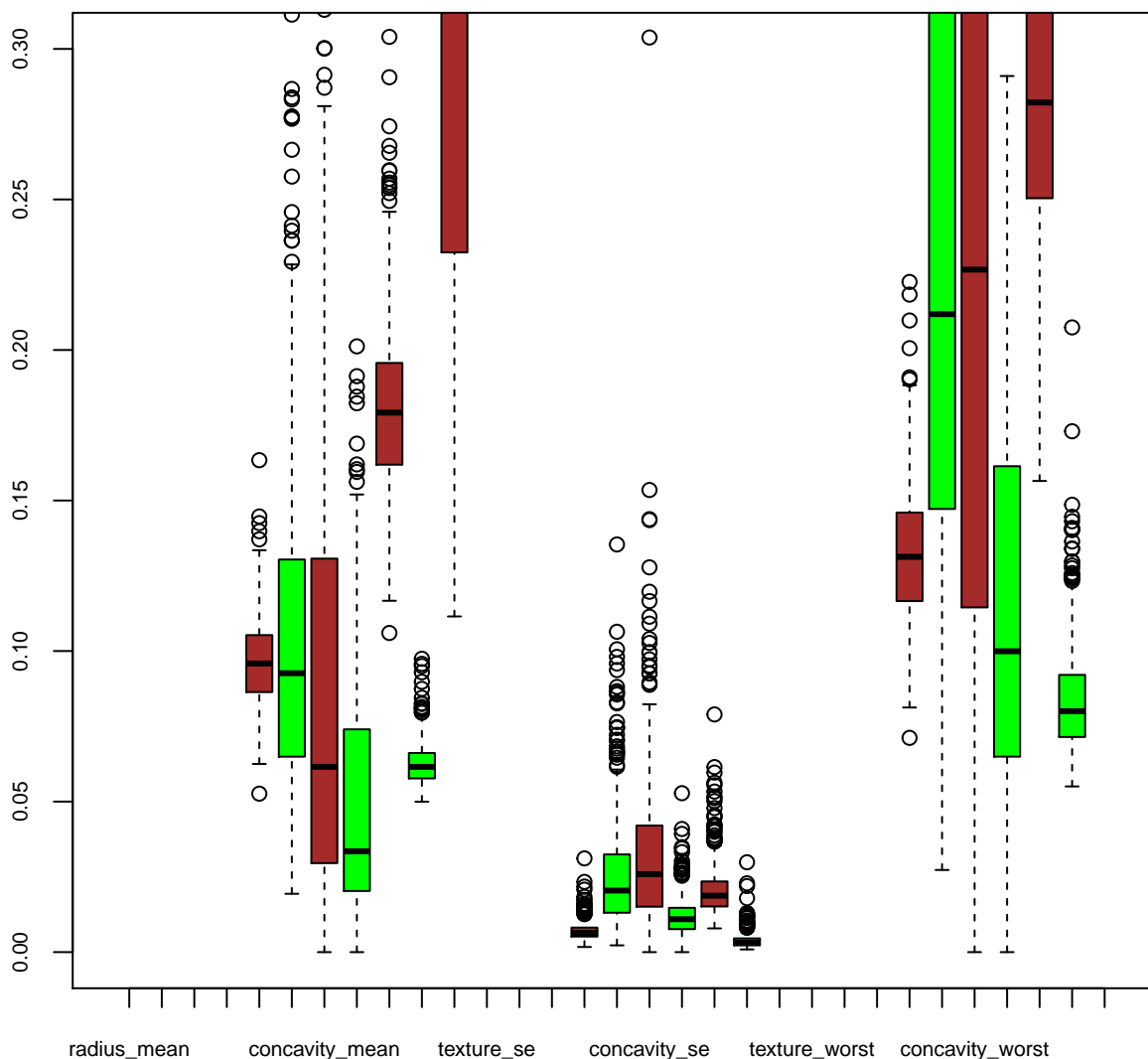
boxplot(data,main='Datos sin normalizar',col=c('brown', 'green'),cex.axis=0.7,subset=data$diagnosis, ylim=
boxplot(data,main='Datos sin normalizar',col=c('brown', 'green'),cex.axis=0.7,subset=data$diagnosis, ylim=

```

Datos sin normalizar



Datos sin normalizar



Cada variable muestra diferencias en los resultados según pertenezcan a tejido sano o tumoral. Sin embargo, aunque normalmente el tejido sano parece mostrar valores menores que el tejido tumoral, depende generalmente de la variable. Mostrando por ejemplo valores muy similares en la variable “fractal_dimension_mean”. Podemos ver cada variable de forma independiente, pero ya que tenemos un gran número de variables, el estudio puede extenderse demasiado.

Como muestra, podemos realizar un análisis aleatorio de todas las muestras. Para ello, se toma una muestra de 15 tejidos sanos y otra muestra de tejidos tumorales, analizando la distribución resultante de 5 variables aleatorias:

```
indicesT <- which(data$diagnosis == 'Maligno')
indicesN <- which(data$diagnosis == 'Benigno')

set.seed(123456)
aleatT <- sample(indicesT, 15)
set.seed(params$seed.train)
```



```

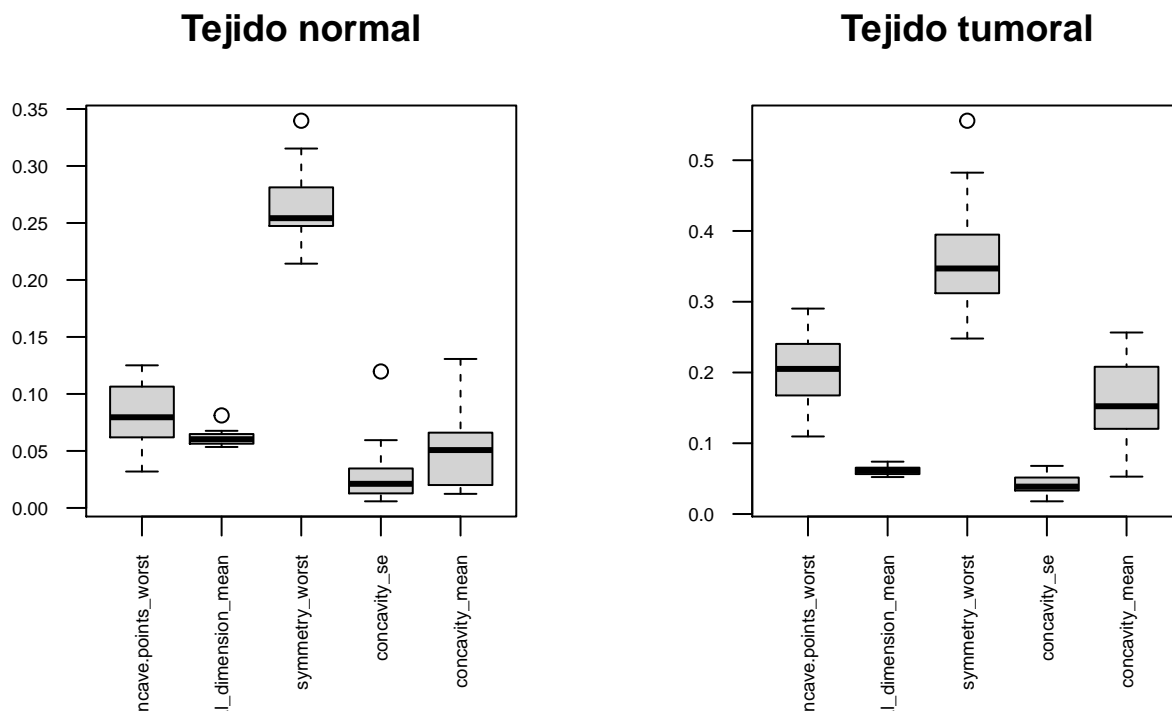
aleatN <- sample(indicesN,15)

#10 muestras aleatorias
set.seed(123456)
aleatG <- sample(1:(ncol(data)-1),5)

#Dataset "reducidos" con las muestras
datasetT <- data[aleatT,aleatG]
datasetN <- data[aleatN,aleatG]

par(mfrow=c(1,2))
boxplot(datasetN,cex.axis=0.6,ylab='',main="Tejido normal",las=2)
abline(h = 500,col='red')
boxplot(datasetT,cex.axis=0.6,ylab='', main="Tejido tumoral", las=2)
abline(h = 500,col='red')

```



Por otro lado, podemos realizar una normalización de los datos, de forma que todos se encuentran entre 0 y 1 y nos permita una visualización más sencilla y fácil de interpretar:

```

# Funcion de normalizacion
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}

```

Ahora podemos aplicar la función `normalize()` a las características numéricas en nuestro marco de datos. En lugar de normalizar cada una de las 30 variables numéricas individualmente, utilizaremos una de las funciones de R para automatizar el proceso.

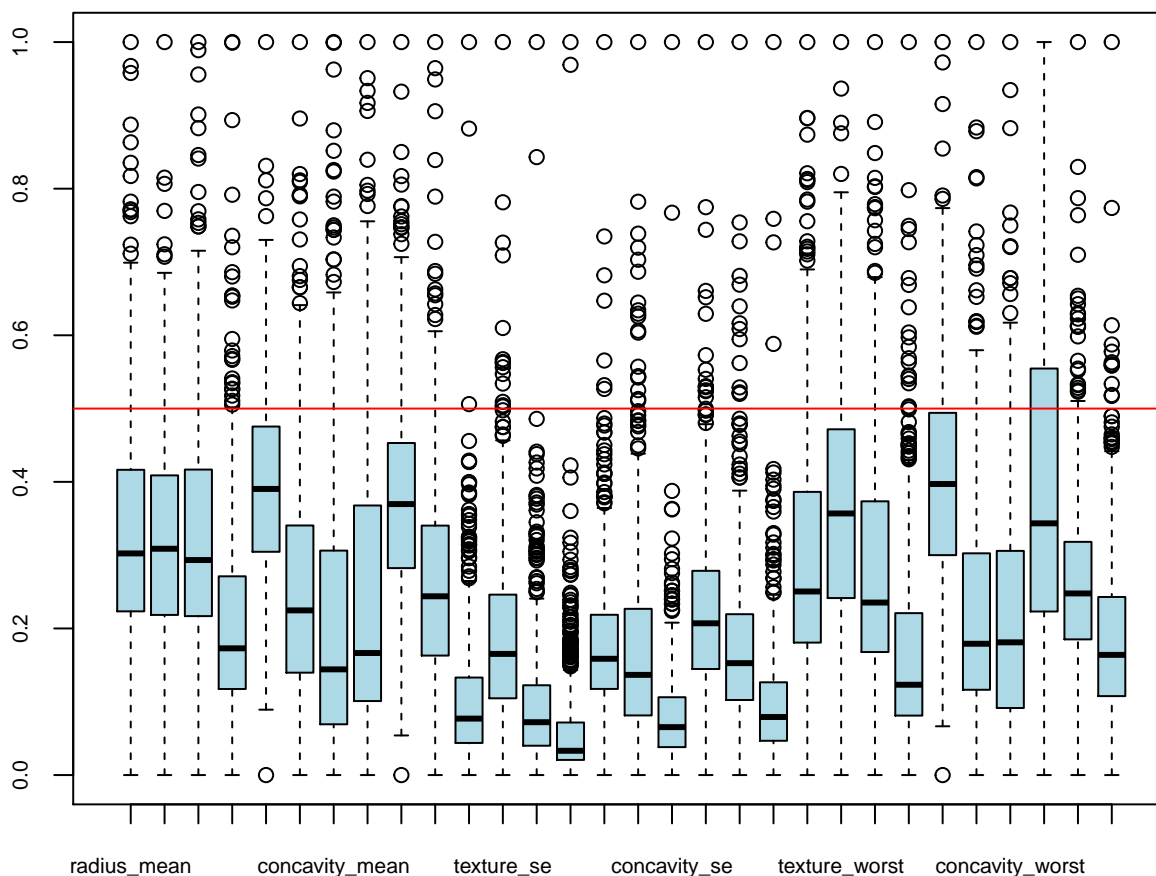
```

# normalizamos los datos
data.norm <- as.data.frame(lapply(data[, -31], normalize))
# Los graficamos
boxplot(data.norm,main='Datos normalizados',col=c('lightblue'),cex.axis=0.7,subset=data$diagnosis)

```

```
abline(h=0.5,lwd=1, col="red")
```

Datos normalizados



3.3. Partición de los datos en training/test

Realizamos *aleatoriamente* una extracción de los datos para entrenar el modelo, en concreto 66.67% de todas las observaciones, que son 379 registros, y el resto, 190 registros, para evaluarlo (test). Usamos los datos sin normalizar y si en algún caso necesitan normalización o alguna otra transformación lo haremos directamente dentro del análisis de cada modelo.

```
#fijamos la semilla para el generador pseudoaleatorio
set.seed(params$seed.train)

# creamos los grupos de entrenamiento (train) y prueba (test)
train<-sample(1:nrow(data),round(nrow(data)*params$p.train,0))
datos.train <- data.frame(data[train,])
datos.test  <- data.frame(data[-train,])

nrow(datos.train)/nrow(datos.test) # debe estar alrededor de 2
```

```
[1] 1.994737
```

El porcentaje de muestras con el diagnóstico de maligno con nuestros datos de entrenamiento es:

```
prop.table(table(datos.train$diagnosis))
```

```
Benigno Maligno  
0.6385224 0.3614776
```

El porcentaje de muestras con el diagnóstico de maligno con nuestros datos de test es:

```
prop.table(table(datos.test$diagnosis))
```

```
Benigno Maligno  
0.6052632 0.3947368
```

Vemos que es muy similar, por lo que los datos están equilibrados.

4. Aplicación de cada uno de los algoritmos para la clasificación

4.1. k-Nearest Neighbour

4.1.1. Transformación de los datos

Para aplicar el algoritmo de k-nearest necesitamos tener los datos normalizados. Este paso lo hemos realizado con el fin de graficar los datos de forma apropiada en el apartado anterior, por lo que usaremos el data frame `data.norm` creado en el apartado *Exploración y preparación de los datos*.

Cuando construimos nuestros datos de entrenamiento y prueba, excluimos la variable objetivo, 'diagnosis'. Para entrenar el modelo kNN, necesitaremos almacenar estas etiquetas de clase en vectores de factores, divididos en los conjuntos de datos de entrenamiento y prueba, además creamos de nuevo los grupos de entrenamiento y prueba con los datos normalizados:

```
#fijamos la semilla para el generador pseudoaleatorio  
set.seed(params$seed.train)  
  
# creamos los grupos de entrenamiento (train) y prueba (test) con los datos normalizados  
train<-sample(1:nrow(data),round(nrow(data)*params$p.train,0))  
datos.train.n <- data.frame(data.norm[train,])  
datos.test.n  <- data.frame(data.norm[-train,])  
  
# Creamos las etiquetas para ambos grupos  
datos.train.label <- data[train, 31]  
datos.test.label <- data[-train, 31]
```

4.1.2. Entrenar el modelo

Para clasificar nuestras muestras, usamos la función `knn()` para clasificar los datos del grupo test:

```
library(class)  
data.test.pred <- knn(train = datos.train.n, test = datos.test.n,  
                      cl = datos.train.label, k = 21)
```

Como sabemos, la función `knn()` devuelve un vector factorial de etiquetas predichas para cada una de las muestras en el conjunto de datos test, que hemos asignado a `data.test.pred`.

4.1.3. Predicción y Evaluación del algoritmo

El siguiente paso es evaluar qué tan bien las clases predichas en el vector `data.test.pred` coinciden con los valores conocidos en el vector `datos.test.label`. Para hacer esto, podemos usar la función `CrossTable()` en el paquete `gmodels`:

```
library(gmodels)
# Podemos crear una tabulación cruzada que indique el acuerdo entre los dos vectores. Especificando `prop.`
conf.mat <- CrossTable(x = datos.test.label, y = data.test.pred,
  prop.chisq = FALSE)
```

Cell Contents

```
|-----|
|              N |
|      N / Row Total |
|      N / Col Total |
|      N / Table Total |
|-----|
```

Total Observations in Table: 190

	data.test.pred		
datos.test.label	Benigno	Maligno	Row Total
Benigno	114	1	115
	0.991	0.009	0.605
	0.919	0.015	
	0.600	0.005	
Maligno	10	65	75
	0.133	0.867	0.395
	0.081	0.985	
	0.053	0.342	
Column Total	124	66	190
	0.653	0.347	

Los casos 114 de 190 indican casos en los que la muestra era benigna, y el algoritmo kNN la identificó correctamente. Un total de predicciones 65 de 190 fueron verdaderos positivos, es decir, identificados adecuadamente como “Malignos”.

Con este modelo se clasificó incorrectamente un total de 10 de las muestras 190.

Para ver los resultados podemos emplear la función `confusionMatrix` del paquete `caret`, que nos indica los valores obtenidos de *Sensitivity* y *Specificity*. Es la función que vamos a emplear en todos los clasificadores para poder compararlos al final.

```
require(caret,quietly = TRUE)
cf.knn <- confusionMatrix(data.test.pred,datos.test.label,positive="Maligno")
cf.knn
```

Confusion Matrix and Statistics

```

      Reference
Prediction Benigno Maligno
Benigno    114      10
Maligno     10      65
```

```

Accuracy : 0.9421
95% CI : (0.8988, 0.9707)
No Information Rate : 0.6053
P-Value [Acc > NIR] : < 2e-16

Kappa : 0.8763

Mcnemar's Test P-Value : 0.01586

Sensitivity : 0.8667
Specificity : 0.9913
Pos Pred Value : 0.9848
Neg Pred Value : 0.9194
Prevalence : 0.3947
Detection Rate : 0.3421
Detection Prevalence : 0.3474
Balanced Accuracy : 0.9290

'Positive' Class : Maligno

```

El modelo knn con categoria positiva 'Maligno' obtiene una precision de 0.942 y una sensibilidad y especificidad de 0.867 y 0.991 respectivamente.

Vamos a intentar ahora otra iteración del modelo para ver si podemos mejorar el rendimiento y reducir el número de valores que se han clasificado incorrectamente, especialmente aquellos casos *falsos negativos*, ya que sería grave identificar como benigno una muestra cuando en realidad es maligna. cuando en realidad era benigno. Aunque todos los errores obtenidos deben de tratar evitarse, ya que los falsos positivos pueden ocasionar estres al paciente o gastos innecesarios al hospital.

Para mejorar el modelo vamos a realizar dos variaciones además de implementar el modelo con el paquete caret. Intentaremos dos variaciones simples en el clasificador anterior. Primero, un método que reescala las variables numéricas y en segundo lugar, intentaremos varios valores del parámetro k .

4.1.3.1. Posible mejora 1: Transformación - estandarización de z

Para estandarizar nuestros valores, excepto la variable diagnosis que no es numérica, emplearemos la función `scale()` de R, que por defecto reescala los valores usando the z-score standardization y almacenamos el resultado en la variable `data.z`.

```

# use the scale() function to z-score standardize a data frame
data.z <- as.data.frame(scale(data[-31]))
# comprobamos que se ha aplicado la transformacion
summary(data.z$area_mean)

```

```

  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-1.4532 -0.6666 -0.2949  0.0000  0.3632  5.2459

```

La media de una variable estandarizada z-score siempre debe ser cero (como ocurre en nuestro caso), y el rango debe ser bastante compacto. Un z-score mayor que 3 o menor que -3 indica un valor extremadamente raro. El resumen anterior nos indica que quizá esta transformación no es la más adecuada, ya que el rango es elevado, más de 3.

Sin embargo, veamos los resultados del modelo. Para ello, volvemos a entrenar el modelo con los nuevos datos de entrenamiento transformados usando la función `knn()`. Luego compararemos las etiquetas predichas con las etiquetas reales usando `confusionMatrix()`:

```

# Creamos de nuevo los grupos de entrenamiento y test con los datos normalizados
data.train.z <- data.z[train, ]
data.test.z <- data.z[-train, ]

# re-classify test

```

```
data.test.pred.z <- knn(train = data.train.z, test = data.test.z,
                        cl = datos.train.label, k = 21)
# Creamos la tabla de resultados
cf.knn.z <- confusionMatrix(data.test.pred.z,datos.test.label,positive="Maligno")
cf.knn.z
```

Confusion Matrix and Statistics

```

              Reference
Prediction Benigno Maligno
Benigno      115      12
Maligno       0       63

      Accuracy : 0.9368
      95% CI : (0.8923, 0.9669)
No Information Rate : 0.6053
P-Value [Acc > NIR] : < 2.2e-16

      Kappa : 0.864

McNemar's Test P-Value : 0.001496

      Sensitivity : 0.8400
      Specificity : 1.0000
Pos Pred Value : 1.0000
Neg Pred Value : 0.9055
Prevalence : 0.3947
Detection Rate : 0.3316
Detection Prevalence : 0.3316
Balanced Accuracy : 0.9200

      'Positive' Class : Maligno
```

En la tabla anterior los resultados muestran una ligera disminución en el grado de precisión (Accuracy). Por otro lado, se mejora la *Specificity* (proporción de sanos correctamente identificados) pero empeora la *Sensitivity* (detectar la enfermedad en sujetos enfermos). Por lo que no se mejoran los datos del clasificador anterior.

4.1.3.2. Posible mejora 2: distintos valores de k

Vamos a intentar ahora mejorar el modelo con otros valores de k , empleando para ello los datos normalizados y los datos de prueba, e imprimimos una tabla resumen con los resultados obtenidos

```
k_valor <- params$k_value
resum <- data.frame(k_valor, Accuracy=NA, Kappa=NA, Sensitivity=NA, Specificity=NA)
j <- 0
for (i in k_valor){
  j <- j +1
  data.test.pred.k <- knn(train = datos.train.n, test = datos.test.n, cl = datos.train.label, k=i)
  cf.knn.k <- confusionMatrix(data.test.pred.k,datos.test.label,positive="Maligno")

  resum[j,2:5] <- c(round(cf.knn.k$overall["Accuracy"], 3), round(cf.knn.k$overall["Kappa"], 3), round(cf.knn.k$overall["Sensitivity"], 3), round(cf.knn.k$overall["Specificity"], 3))
}
print(resum)
```

	k_valor	Accuracy	Kappa	Sensitivity	Specificity
1	1	0.937	0.867	0.907	0.957
2	5	0.947	0.889	0.907	0.974

3	11	0.958	0.910	0.893	1.000
4	17	0.942	0.876	0.867	0.991
5	23	0.942	0.876	0.867	0.991
6	27	0.932	0.853	0.840	0.991

Aunque el primer clasificador empleado no ha sido perfecto, el enfoque 1NN pudo evitar algunos de los falsos negativos a expensas de agregar falsos positivos.

4.2. Naive Bayes

4.2.1. Transformación de los datos

Calcula las probabilidades a-posteriores condicionales de una variable de clase categórica dadas otras variables predictoras independientes usando la regla de Bayes, sin necesidad de ningún tipo de transformación en los datos.

4.2.2. Entrenar el modelo

Como ya tenemos los grupos de train y training creados sin los datos transformados en el apartado 3.3, ya podemos entrenar el algoritmo. El valor predeterminado del argumento `laplace` (0) deshabilita el suavizado de Laplace:

```
library(e1071)
data.nb <- naiveBayes(datos.train, datos.train.label, laplace=0)
```

El resultado del entrenamiento contiene las probabilidades condicionadas de cada categoría según el tipo de diagnóstico. Veamos las cuatro primeras variables:

```
data.nb$tables[1:4]

$radius_mean
      radius_mean
datos.train.label  [,1]      [,2]
      Benigno 12.04088 1.744117
      Maligno 17.52124 3.129247

$texture_mean
      texture_mean
datos.train.label  [,1]      [,2]
      Benigno 18.17752 4.172830
      Maligno 21.81869 3.980472

$perimeter_mean
      perimeter_mean
datos.train.label  [,1]      [,2]
      Benigno 77.36731 11.64092
      Maligno 115.76584 21.38945

$area_mean
      area_mean
datos.train.label  [,1]      [,2]
      Benigno 454.3711 129.3648
      Maligno 984.9847 363.7715
```

4.2.3. Predicción y Evaluación del algoritmo

Aplicamos la función `predict` del algoritmo con los datos de test para hacer su predicción:

```
test.pred.nb <- predict(data.nb, datos.test)
```

Ahora miramos los resultados en una *Cross Table* usando la función `confusionMatrix` del package `caret`:

```
require(caret,quietly = TRUE)
nb.matrix <- confusionMatrix(test.pred.nb,datos.test.label,positive="Maligno")
nb.matrix
```

Confusion Matrix and Statistics

```

      Reference
Prediction Benigno Maligno
Benigno    113      4
Maligno      2     71

      Accuracy : 0.9684
      95% CI : (0.9325, 0.9883)
No Information Rate : 0.6053
P-Value [Acc > NIR] : <2e-16

      Kappa : 0.9336

McNemar's Test P-Value : 0.6831

      Sensitivity : 0.9467
      Specificity : 0.9826
      Pos Pred Value : 0.9726
      Neg Pred Value : 0.9658
      Prevalence : 0.3947
      Detection Rate : 0.3737
      Detection Prevalence : 0.3842
      Balanced Accuracy : 0.9646

      'Positive' Class : Maligno
```

El modelo Naive Bayes con categoria positiva ‘Maligno’ obtiene una precisión de 0.968 y una sensibilidad y especificidad de 0.947 y 0.983 respectivamente.

4.2.3.1. Posible mejora: *laplace = 1*

Ahora se prueba a entrenar el modelo aplicando la opción `laplace = 1`:

```
data.nb2 <- naiveBayes(datos.train, datos.train.label, laplace=1)
```

y se hace la predicción:

```
test.pred.nb2 <- predict(data.nb2, datos.test)
```

Ahora se evalua el modelo con la función `confusionMatrix` del package `caret`.

```
nb.matrix.1 <- confusionMatrix(test.pred.nb2,datos.test.label,positive="Maligno")
```

El nuevo modelo daplizando la opción `laplace = 1` obtiene una precision de 0.963 y una sensibilidad y especificidad de 0.933 y 0.983 respectivamente. Vemos que el modelo obtenido con SVM lineal tiene una mayor precision

4.2.3.2. Curvas ROC

Se presentan las curvas ROC para el modelo de Naive Bayes con `laplace=0` y `laplace= 1`.

Caso `laplace=0`

El primer paso es obtener las probabilidades diagnosis (Maligno/Benigno) para cada muestra de los datos de test:


```
test.pred.nb3 <- predict(data.nb, datos.test, type="raw")
tail(test.pred.nb3)
```

```

      Benigno      Maligno
[185,] 1.000000e+00 1.774260e-19
[186,] 1.000000e+00 2.401845e-16
[187,] 1.000000e+00 4.743641e-19
[188,] 1.000000e+00 5.832161e-15
[189,] 6.419662e-45 1.000000e+00
[190,] 2.146303e-167 1.000000e+00

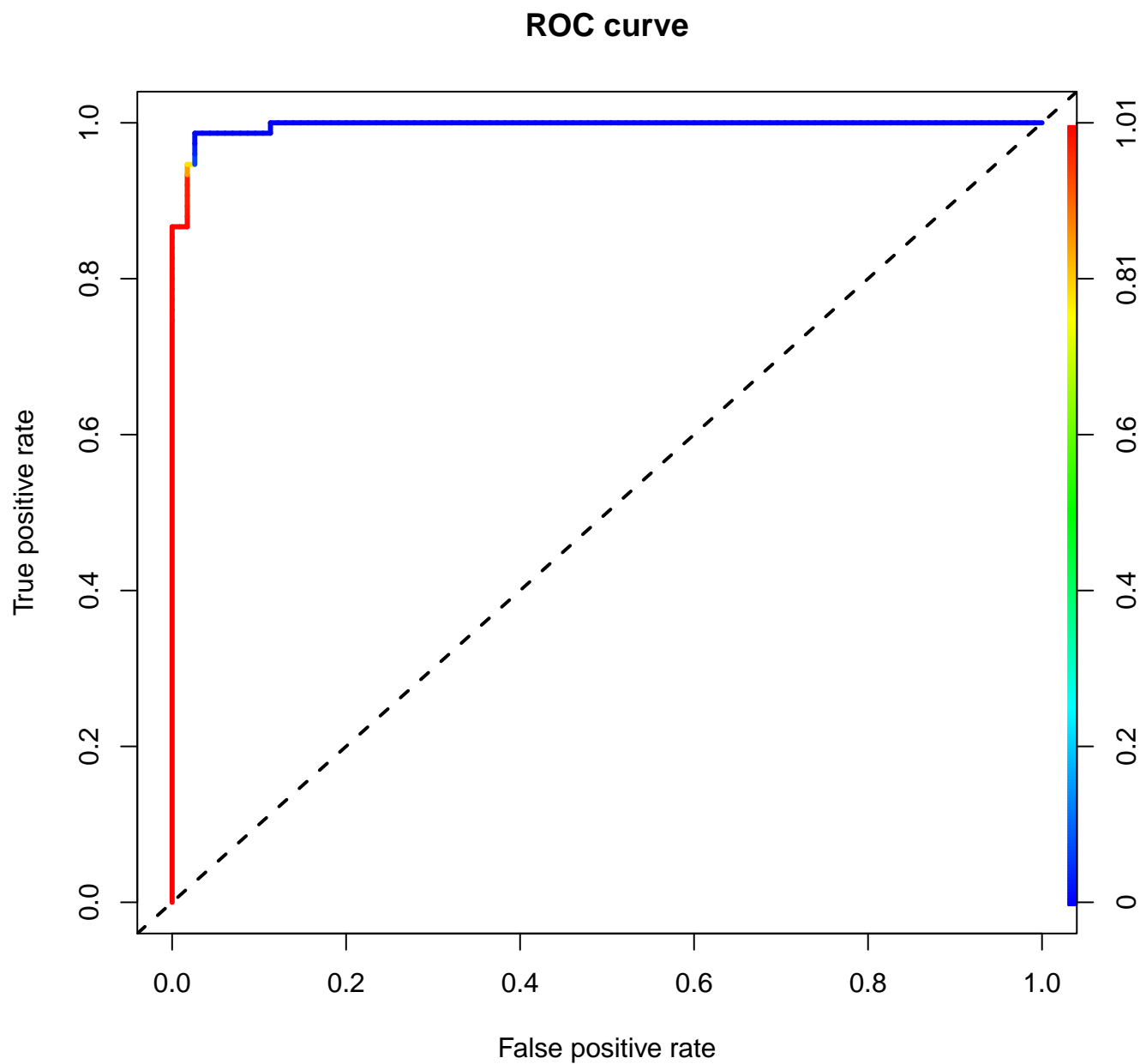
```

Con la información de las probabilidades de la clase positiva (1) se construye la curva ROC.

```
require(ROCR,quietly=TRUE)

pred <- prediction(predictions= test.pred.nb3[,2], labels=datos.test.label)
perf <- performance(pred, measure="tpr", x.measure="fpr")

plot(perf, main= "ROC curve", col= "blue", lwd=3, colorize=TRUE)
abline(a=0, b= 1, lwd= 2, lty = 2)
perf.auc <- performance(pred, measure ="auc")
```



El area bajo la curva es **0.996058**.

Caso laplace=1

El primer paso es obtener las probabilidades diagnosis (Maligno/Benigno) para cada muestra de los datos de test:

```
test.pred.nb4 <- predict(data.nb2, datos.test, type="raw")
tail(test.pred.nb4)
```

	Benigno	Maligno
[185,]	1.000000e+00	1.281699e-18
[186,]	1.000000e+00	1.735057e-15
[187,]	1.000000e+00	3.426735e-18
[188,]	1.000000e+00	4.213066e-14
[189,]	2.650074e-44	1.000000e+00
[190,]	8.860065e-167	1.000000e+00

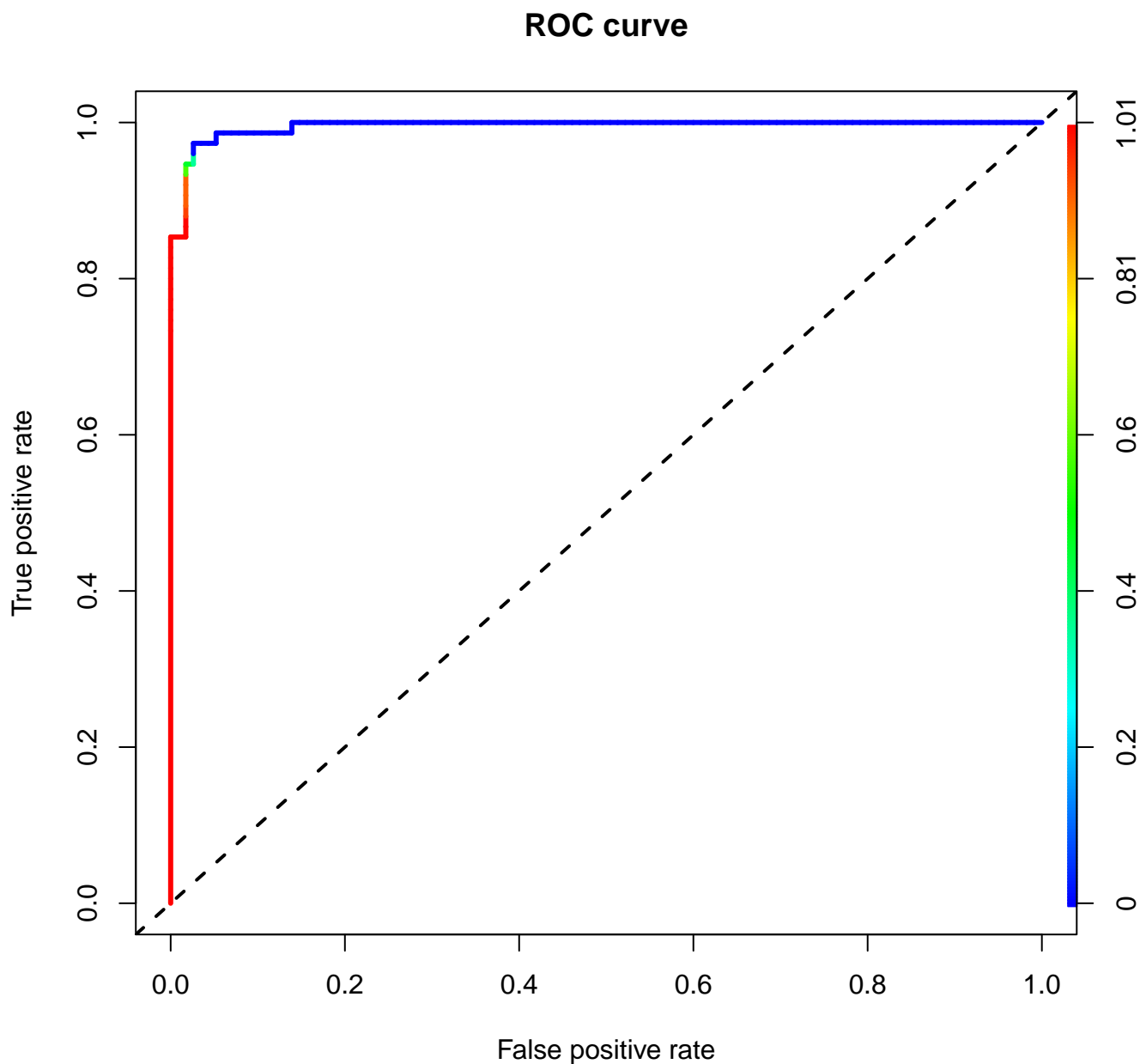
Con la información de las probabilidades de la clase positiva (1) se construye la curva ROC.

```
pred2 <- prediction(predictions= test.pred.nb4[,2], labels=datos.test.label)
perf2 <- performance(pred2, measure="tpr", x.measure="fpr")

plot(perf2, main= "ROC curve", col= "blue", lwd=3, colorize=TRUE)
abline(a=0, b= 1, lwd= 2, lty = 2)
perf2.auc <- performance(pred2, measure = "auc")

unlist(perf2.auc@y.values)
```

```
[1] 0.9951304
```



El area bajo la curva es **0.9951304** y con laplace = 0 es `unlist(perf.auc@y.values)`. Como vemos no mejora

la predicción, por lo que no tiene sentido cambiar el argumento de 0 a 1.

4.3. Artificial Neural Network

4.3.1. Transformacion de los datos

Hay que normalizar las variables para que tomen valores entre 0 y 1, tal y como hemos realizado en la clasificación de k-nn. Los datos normalizados se encuentran dentro de `data.norm`

```
#data.norm <- as.data.frame(lapply(data[, -31], normalize))
```

Se confirma que el rango de valores esta entre 0 y 1.

```
summary(data.norm)[,1:5]
```

radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
Min. :0.0000	Min. :0.0000	Min. :0.0000	Min. :0.0000	Min. :0.0000
1st Qu.:0.2233	1st Qu.:0.2185	1st Qu.:0.2168	1st Qu.:0.1174	1st Qu.:0.3046
Median :0.3024	Median :0.3088	Median :0.2933	Median :0.1729	Median :0.3904
Mean :0.3382	Mean :0.3240	Mean :0.3329	Mean :0.2169	Mean :0.3948
3rd Qu.:0.4164	3rd Qu.:0.4089	3rd Qu.:0.4168	3rd Qu.:0.2711	3rd Qu.:0.4755
Max. :1.0000	Max. :1.0000	Max. :1.0000	Max. :1.0000	Max. :1.0000

Ahora se crean tantas variables binarias como categorías tiene la variable `diagnosis`. Por lo que ahora nuestra matriz de datos tiene 32 variables, las 30 variables numéricas, y las dos variables que creamos que son de tipo binario, con valores de TRUE y/o FALSE.

```
# Creación de variables binarias en lugar de usar la variable factor
data.norm$M <- data$diagnosis=="Maligno"
data.norm$B <- data$diagnosis=="Benigno"
```

Del mismo modo emplearemos los conjuntos para entrenamiento y prueba creados en el apartado anterior con datos normalizados, pero incluyendo ahora las dos nuevas variables:

```
# create labels for training and test data

datos.train.nn <- data.frame(data.norm[train,])
datos.test.nn <- data.frame(data.norm[-train,])

datos.train.label <- data[train, 31]
datos.test.label <- data[-train, 31]
```

4.3.2. Entrenar el modelo

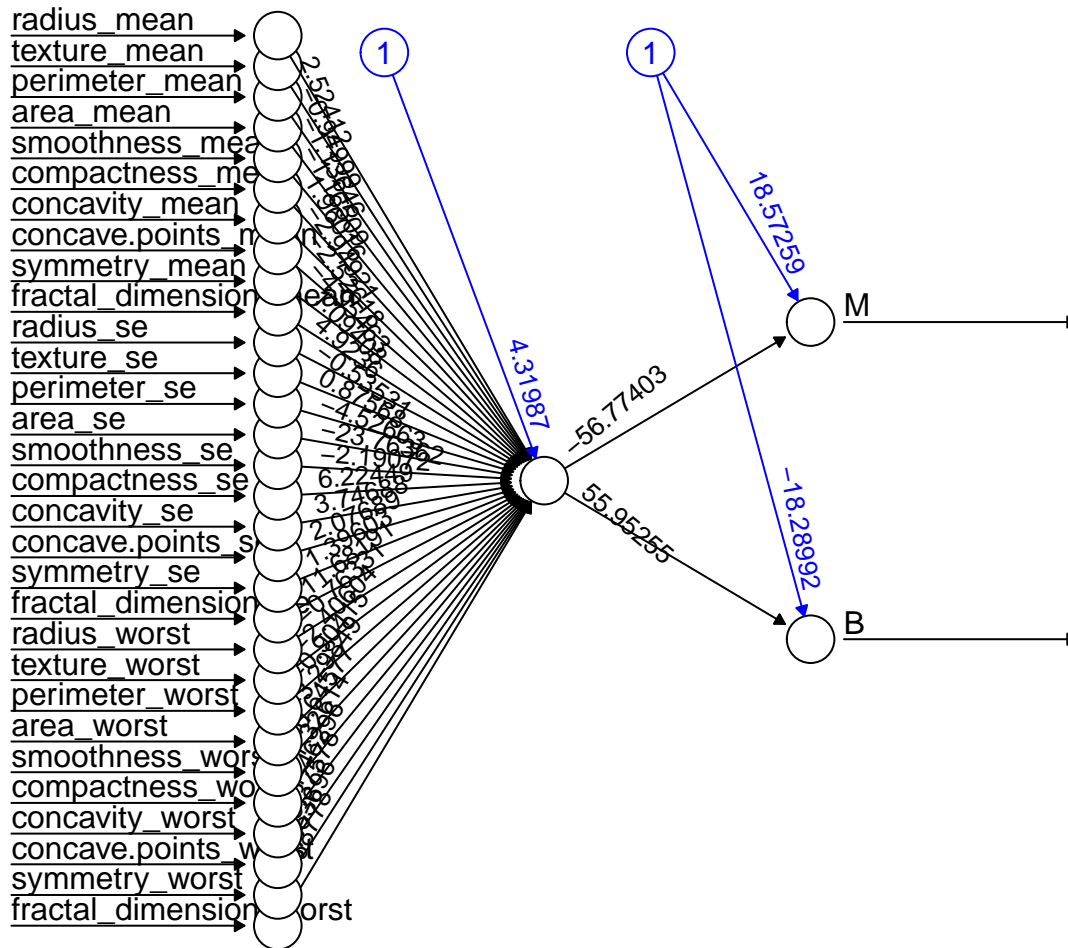
Para la construcción de la red neuronal artificial se usa la función `neuralnet()` del paquete *neuralnet*. La fórmula del modelo tiene 32 nodos de entrada y 2 (niveles de `data$diagnosis`) nodos de salida:

```
# Creamos una formula para desarrollar el modelo
# con todas y cada una de las 32 variables:
xnam <- names(data[1:30])
(fmla <- as.formula(paste("M + B ~ ", paste(xnam, collapse= "+"))))
```

```
M + B ~ radius_mean + texture_mean + perimeter_mean + area_mean +
smoothness_mean + compactness_mean + concavity_mean + concave.points_mean +
symmetry_mean + fractal_dimension_mean + radius_se + texture_se +
perimeter_se + area_se + smoothness_se + compactness_se +
concavity_se + concave.points_se + symmetry_se + fractal_dimension_se +
radius_worst + texture_worst + perimeter_worst + area_worst +
smoothness_worst + compactness_worst + concavity_worst +
concave.points_worst + symmetry_worst + fractal_dimension_worst
```

El modelo aplicado es de un nodo en la capa oculta, esto se consigue con el argumento `hidden=1`. El modelo se construye con el argumento `linear.output=FALSE` ya que se trata de un problema de clasificación.

```
# simple ANN con una única neurona en la capa oculta
set.seed(params$seed.clsfier) # semilla que nos garantiza resultados reproducibles
library(neuralnet)
nn.model.1 <- neuralnet(fmla, data = datos.train.nn, hidden = 1, linear.output = FALSE)
# Visualizamos la topología de la red neuronal:
plot(nn.model.1, rep = "best")
```



4.3.3. Predicción y Evaluación del algoritmo

Una vez obtenido el primer modelo, se evalúa su rendimiento con los datos de test. Se debe de clasificar las muestras test con la función `predict` al igual que en los algoritmos anteriores. El resultado de la matriz de confusión con los datos de test en este modelo podemos obtenerla con el siguiente código:

```
nn.model.1.matrix <- predict(nn.model.1, datos.test.nn[, 1:32])

# Creamos una salida binaria múltiple a nuestra salida categórica
max.idx <- function(xxy) {
  return(which(xxy == max(xxy)))
}
idx1 <- apply(nn.model.1.matrix, 1, max.idx)
prediction <- c("Maligno", "Benigno")[idx1]
result <- table(prediction, data$diagnosis[-train])
```

```
# Resultados
library(caret)
(cmatrix1 <- confusionMatrix(result, positive = "Maligno"))
```

Confusion Matrix and Statistics

```
prediction Benigno Maligno
Benigno      107      4
Maligno       8      71

      Accuracy : 0.9368
      95% CI : (0.8923, 0.9669)
No Information Rate : 0.6053
P-Value [Acc > NIR] : <2e-16

      Kappa : 0.869

McNemar's Test P-Value : 0.3865

      Sensitivity : 0.9467
      Specificity : 0.9304
      Pos Pred Value : 0.8987
      Neg Pred Value : 0.9640
      Prevalence : 0.3947
      Detection Rate : 0.3737
      Detection Prevalence : 0.4158
      Balanced Accuracy : 0.9386

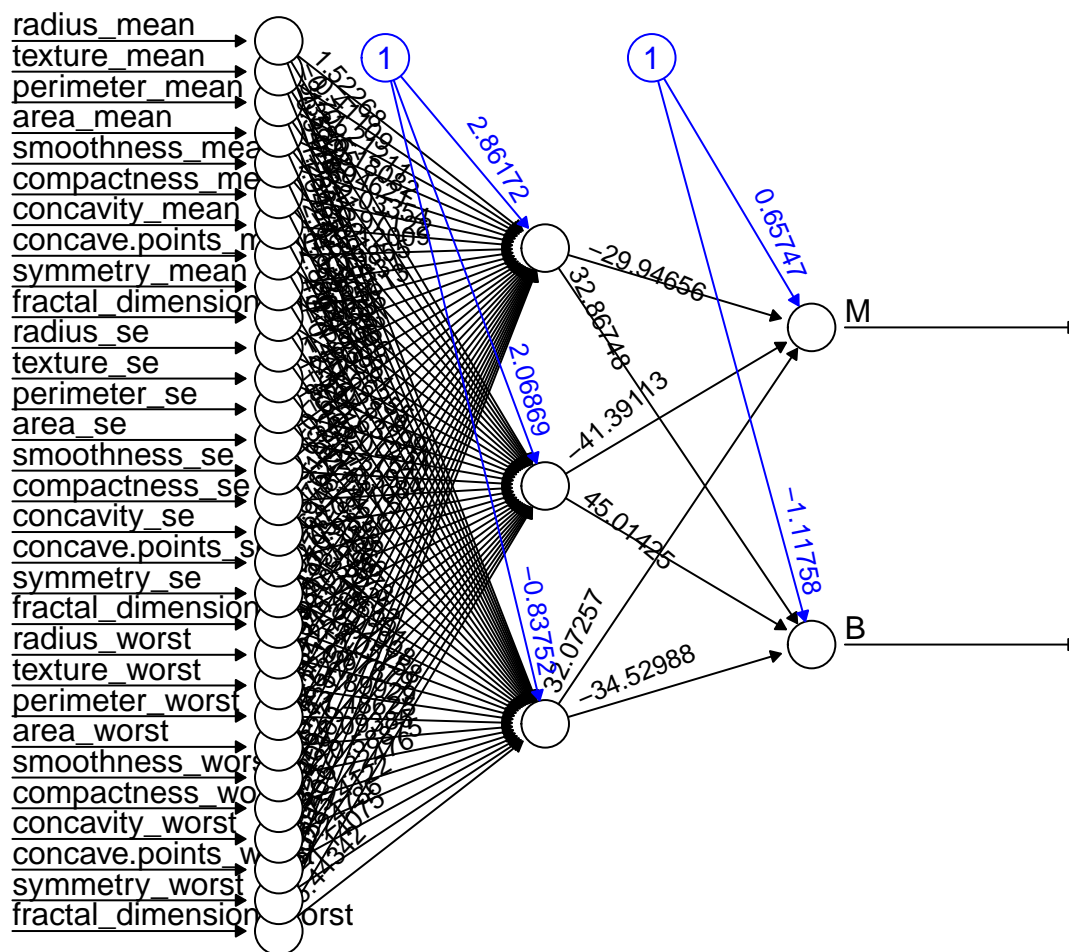
      'Positive' Class : Maligno
```

El modelo de una capa con categoría positiva ‘Maligno’ obtiene una precisión de 0.937 y una sensibilidad y especificidad de 0.947 y 0.93 respectivamente.

4.3.3.1. Posible mejora: 3 nodos

Se plantean 3 nodos en la capa oculta para tratar de mejorar el rendimiento.

```
# ANN simple con tres neuronas ocultas
set.seed(params$seed.clsfier) # garantiza los resultados reproducibles
nn.model.3 <- neuralnet(fmla, data = datos.train.nn, hidden = 3, linear.output = FALSE)
# visualizamos la topología de la red
plot(nn.model.3, rep = "best")
```



El resultado de la matriz de confusión con los datos de test es:

```
nn.model.3.matrix <- predict(nn.model.3, datos.test.nn[, 1:32])

idx3 <- apply(nn.model.3.matrix, 1, max.idx)
prediction3 <- c("Maligno", "Benigno")[idx3]
result.3 <- table(prediction3, data$diagnosis[-train])

# Resultados
(cmatrix3 <- confusionMatrix(result.3, positive = "Maligno"))
```

Confusion Matrix and Statistics

```
prediction3 Benigno Maligno
Benigno    106      5
Maligno      9     70
```

```
Accuracy : 0.9263
95% CI : (0.8795, 0.9591)
No Information Rate : 0.6053
P-Value [Acc > NIR] : <2e-16
```

```
Kappa : 0.8472
```

```
McNemar's Test P-Value : 0.4227
```

```

      Sensitivity : 0.9333
      Specificity : 0.9217
      Pos Pred Value : 0.8861
      Neg Pred Value : 0.9550
      Prevalence : 0.3947
      Detection Rate : 0.3684
      Detection Prevalence : 0.4158
      Balanced Accuracy : 0.9275

      'Positive' Class : Maligno

```

El nuevo modelo con 3 nodos ocultos obtiene una precisión de 0.926 y una sensibilidad y especificidad de 0.933 y 0.922 respectivamente. Vemos que el modelo obtenido con un solo nodo tiene una mayor precisión.

Si se parecen debemos emplear el modelo más sencillo para evitar overfitting.

4.3.4. Paquete *caret*: modelo *nnet*

La función `nnet` admite datos de tipo factor, así que no hay que transformar la variable `diagnosis` en variables binarias. Empleamos los datos de entrenamiento y test creados en el apartado 3.3. Veamos como responde el modelo con los datos de test y train:

```

# modelo Train/test without repetition
library(caret)
library(NeuralNetTools)
model.nnet <- train(diagnosis ~ ., datos.train, method='nnet',
  trControl= trainControl(method='none'),
  # preProcess = "range",
  tuneGrid= NULL, tuneLength=1 ,trace = FALSE) #

plotnet(model.nnet)
#summary(model.nnet)
prediction.nnet <- predict(model.nnet, datos.test) # predict
result.caret1 <- table(prediction.nnet, datos.test$diagnosis) # compare
result.caret1

```

```

prediction.nnet Benigno Maligno
      Benigno      115      75
      Maligno       0       0

```

```

# predict también puede devolver la probabilidad para cada clase:
prediction.nnet1 <- predict(model.nnet, datos.test, type="prob")
head(prediction.nnet1)

```

```

      Benigno      Maligno
2  0.6385224 0.3614776
4  0.6385224 0.3614776
5  0.6385224 0.3614776
9  0.6385224 0.3614776
11 0.6385224 0.3614776
19 0.6385224 0.3614776

```

```

# Resultados
(cmatrix.nnet <- confusionMatrix(result.caret1,positive = "Maligno"))

```

Confusion Matrix and Statistics


```

prediction.nnet Benigno Maligno
  Benigno      115      75
  Maligno       0       0

      Accuracy : 0.6053
      95% CI   : (0.5319, 0.6753)
No Information Rate : 0.6053
P-Value [Acc > NIR] : 0.5316

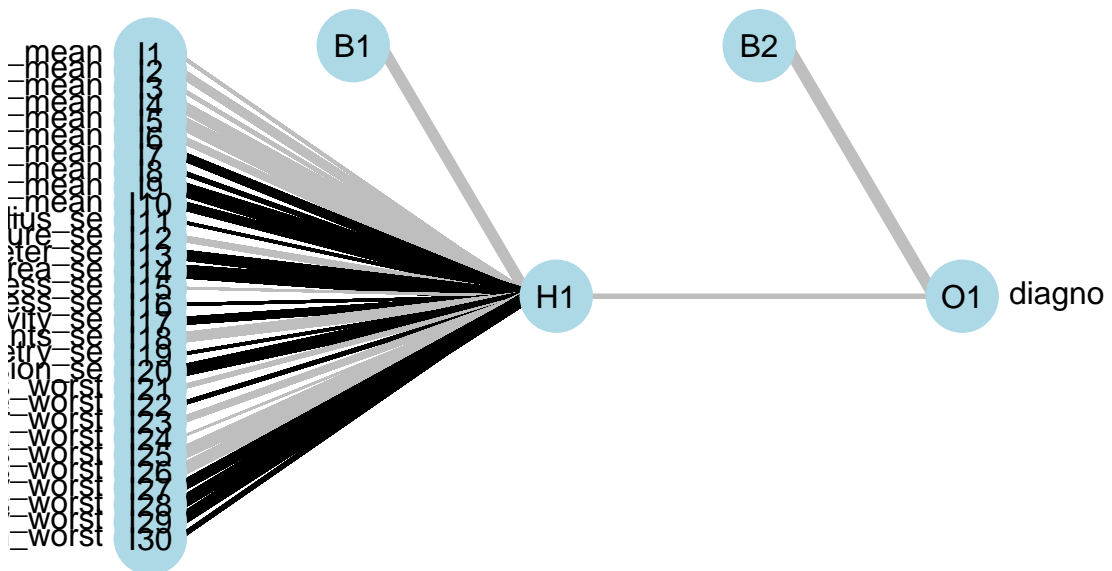
      Kappa : 0

Mcnemar's Test P-Value : <2e-16

      Sensitivity : 0.0000
      Specificity : 1.0000
Pos Pred Value :    NaN
Neg Pred Value : 0.6053
Prevalence : 0.3947
Detection Rate : 0.0000
Detection Prevalence : 0.0000
Balanced Accuracy : 0.5000

'Positive' Class : Maligno

```



El modelo empleando caret con categoria positiva 'Maligno' obtiene una precision de 0.605 y una sensibilidad y

especificidad de 0 y 1 respectivamente.

Empleamos ahora **5-fold crossvalidation** para tratar de mejorar la clasificación:

```
# modelo 5-crossvalidation
model.nnet.2 <- train(diagnosis ~ ., datos.train, method='nnet',
                     trControl= trainControl(method='cv', number=5),
                     tuneGrid= NULL, tuneLength=10 ,trace = FALSE)

plotnet(model.nnet.2, alpha=0.6)
#summary(model.nnet.2)
prediction.nnet2 <- predict(model.nnet.2, datos.test)           # predict
result.caret2 <- table(prediction.nnet2, datos.test$diagnosis)  # compare
result.caret2
```

```
prediction.nnet2 Benigno Maligno
               Benigno      107         7
               Maligno       8         68
```

```
# la probabilidad para cada clase:
prediction.nnet2 <- predict(model.nnet.2, datos.test, type="prob")
head(prediction.nnet2)
```

```
               Benigno      Maligno
2  0.006467878  0.993532122
4  0.996815731  0.003184269
5  0.006467931  0.993532069
9  0.118938318  0.881061682
11 0.006673474  0.993326526
19 0.006467878  0.993532122
```

```
(cmatrix.nnet2 <- confusionMatrix(result.caret2,positive = "Maligno"))
```

Confusion Matrix and Statistics

```
prediction.nnet2 Benigno Maligno
               Benigno      107         7
               Maligno       8         68
```

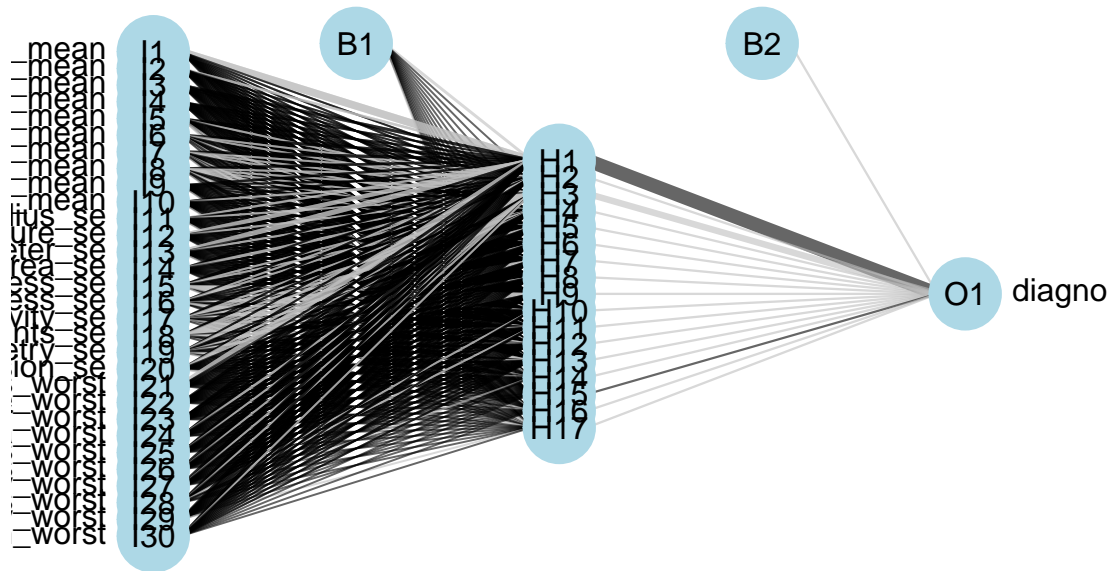
```
               Accuracy : 0.9211
               95% CI : (0.8731, 0.9551)
               No Information Rate : 0.6053
               P-Value [Acc > NIR] : <2e-16
```

```
               Kappa : 0.8352
```

```
McNemar's Test P-Value : 1
```

```
               Sensitivity : 0.9067
               Specificity : 0.9304
               Pos Pred Value : 0.8947
               Neg Pred Value : 0.9386
               Prevalence : 0.3947
               Detection Rate : 0.3579
               Detection Prevalence : 0.4000
               Balanced Accuracy : 0.9186
```

'Positive' Class : Maligno



El modelo empleando caret con 5-fold crossvalidation obtiene una precision de 0.921 y una sensibilidad y especificidad de 0.907 y 0.93 respectivamente.

Como vemos con el paquete caret obtenemos los mejores valores si no realizamos la 5-fold crossvalidation.

4.4. Support Vector Machine

4.4.1. Transformacion de los datos

El SVM no requiere realizar transformaciones normalizantes de los datos.

El porcentaje de muestras tumorales en los datos de training es de 36.15 %, y en los de test de 39.47 %. Así que, como el tamaño de la muestra es más bien bajo, se podría escoger otros metodos como k-fold cross validation o bootstrap, que emplearemos después.

4.4.2. Entrenar el modelo

El algoritmo de SVM que se usa es la funcion `ksvm()` del paquete *kernelab*. Se construye el modelo más sencillo, el lineal, usando como kernel el valor `vanilladot`:

```
library(kernelab)
set.seed(params$seed.clsfier)
modeloLineal <- ksvm(diagnosis~.,data=datos.train, kernel='vanilladot')
```

Setting default kernel parameters

```
# Veamos la información básica sobre el modelo
modeloLineal
```

Support Vector Machine object of class "ksvm"

SV type: C-svc (classification)
parameter : cost C = 1

Linear (vanilla) kernel function.

Number of Support Vectors : 24

Objective Function Value : -11.728
Training error : 0.010554

Se puede observar que la función lineal no tiene parámetros adicionales ('hiperparámetros') al de coste.

4.4.3. Predicción y Evaluación del algoritmo.

Para evaluar el modelo y analizar su rendimiento se realiza la predicción con nuevos datos: los datos de test.

```
modLineal.pred <- predict(modeloLineal, datos.test)
```

Se obtiene la matriz de confusión, usando la función `confusionMatrix`, al igual que en los otros algoritmos:

```
#Modelo lineal
res.svm <- table(modLineal.pred, datos.test$diagnosis)
(cmatrixSVM <- confusionMatrix(res.svm, positive="Maligno"))
```

Confusion Matrix and Statistics

modLineal.pred	Benigno	Maligno
Benigno	112	9
Maligno	3	66

Accuracy : 0.9368
95% CI : (0.8923, 0.9669)
No Information Rate : 0.6053
P-Value [Acc > NIR] : <2e-16

Kappa : 0.866

Mcnemar's Test P-Value : 0.1489

Sensitivity : 0.8800
Specificity : 0.9739
Pos Pred Value : 0.9565
Neg Pred Value : 0.9256
Prevalence : 0.3947
Detection Rate : 0.3474
Detection Prevalence : 0.3632
Balanced Accuracy : 0.9270

'Positive' Class : Maligno

El modelo de SVM lineal con categoría positiva 'tumor' obtiene una precisión de 0.937 y una sensibilidad y especificidad de 0.88 y 0.974 respectivamente.

4.4.3.1. Posible mejora: kernel Gaussiano

Ahora se plantea un SVM con un el kernel Gaussiano, `rbfdot` para tratar de mejorar el rendimiento:

```
set.seed(params$seed.clsfier)
modeloGauss <- ksvm(diagnosis ~ ., data = datos.train, kernel = "rbfdot")
# Prediccion
modeloGauss.pred <- predict(modeloGauss, datos.test)
```

El resultado de la matriz de confusión con los datos de test es:

```
res.svm.gaus <- table(modeloGauss.pred, datos.test$diagnosis)
# Results
library(caret)
(cmatrixSVM2 <- confusionMatrix(res.svm.gaus, positive = "Maligno"))
```

Confusion Matrix and Statistics

```
modeloGauss.pred Benigno Maligno
      Benigno      112         5
      Maligno       3         70

      Accuracy : 0.9579
      95% CI : (0.9187, 0.9816)
      No Information Rate : 0.6053
      P-Value [Acc > NIR] : <2e-16

      Kappa : 0.9115

      Mcnemar's Test P-Value : 0.7237

      Sensitivity : 0.9333
      Specificity : 0.9739
      Pos Pred Value : 0.9589
      Neg Pred Value : 0.9573
      Prevalence : 0.3947
      Detection Rate : 0.3684
      Detection Prevalence : 0.3842
      Balanced Accuracy : 0.9536

      'Positive' Class : Maligno
```

El nuevo modelo de SVM con kernel gaussiano obtiene una precision de 0.958 y una sensibilidad y especificidad de 0.933 y 0.974 respectivamente. Vemos que el modelo obtenido con SVM gaussiano tiene una mayor precision.

4.4.4. Paquete caret

Se construye un nuevo modelo con el objetivo de comprobar si la implementación del paquete **caret** produce diferencias. Se evalúa el rendimiento con la partición train/test y 5-fold cross validation en el modelo lineal y bootstrap con el modelo radial.

Para crear la partición de los datos en training/test podemos emplear la función `createDataPartition` del paquete **caret**, aunque también podemos emplear los datos creados en el apartado 3.3.

```
library(caret)
#Particion de datos
set.seed(params$seed.train)
# We wish 75% for the trainset
```

```
inTrain <- createDataPartition(y=data$diagnosis, p=params$p.train, list=FALSE)

train.set <- data[inTrain,]
test.set  <- data[-inTrain,]

nrow(train.set)/nrow(test.set) # debe estar alrededor de 2
```

```
[1] 2.010582
```

Entrenamos el modelo con los datos de train y test creados por el paquete caret:

```
# modelo solo de Train sin repeticion
set.seed(params$seed.otro)
model.svm.caret <- train(diagnosis ~ ., train.set, method='svmLinear',
  trControl= trainControl(method='none'),
  tuneGrid= NULL, trace = FALSE) #

prediction.svm.caret <- predict(model.svm.caret, test.set) # predict
res.svm.caret <- table(prediction.svm.caret, test.set$diagnosis) # compare

confusionMatrix(res.svm.caret, positive="Maligno")
```

Confusion Matrix and Statistics

```
prediction.svm.caret Benigno Maligno
      Benigno      113         2
      Maligno       6       68

      Accuracy : 0.9577
      95% CI : (0.9183, 0.9816)
      No Information Rate : 0.6296
      P-Value [Acc > NIR] : <2e-16

      Kappa : 0.9103

      Mcnemar's Test P-Value : 0.2888

      Sensitivity : 0.9714
      Specificity : 0.9496
      Pos Pred Value : 0.9189
      Neg Pred Value : 0.9826
      Prevalence : 0.3704
      Detection Rate : 0.3598
      Detection Prevalence : 0.3915
      Balanced Accuracy : 0.9605

      'Positive' Class : Maligno
```

Ahora se repite el modelo pero con los mismos datos de training/test usados en la funcion `ksvm` con kernel `vanilladot` del paquete `kernlab`, para comprobar las variaciones:

```
# modelo solo de Train sin repeticion
set.seed(params$seed.otro)
model.svm.caret2 <- train(diagnosis ~ ., datos.train, method='svmLinear',
  trControl= trainControl(method='none'),
  tuneGrid= NULL, trace = FALSE)
```

```
prediction.svm.caret2 <- predict(model.svm.caret2, datos.test)      # predict
res.svm.caret2 <- table(prediction.svm.caret2, datos.test$diagnosis) # compare

svm.caret.matrix <- confusionMatrix(res.svm.caret2, positive="Maligno")
```

El modelo de SVM lineal con categoría positiva 'Maligno' obtiene una precisión de 0.937 y una sensibilidad y especificidad de 0.88 y 0.974 respectivamente.

Como vemos los datos empleados en el análisis son importantes ya que pueden cambiar el resultado de los datos. En este caso la selección de datos empleada por el paquete `caret` mejora tanto la precisión como la sensibilidad y la especificidad.

Empleamos ahora **5-fold crossvalidation** para tratar de mejorar la clasificación:

```
# modelo 5-crossvalidation
set.seed(params$seed.clsfier)
model.svm.fold <- train(diagnosis ~ ., train.set, method='svmLinear',
  trControl= trainControl(method='cv', number=5),
  tuneGrid= NULL, trace = FALSE)

prediction.fold <- predict(model.svm.fold, test.set)      # predict
res.fold <- table(prediction.fold, test.set$diagnosis)    # compare

svm.caret.matrix5 <- confusionMatrix(res.fold, positive="Maligno")
```

El modelo de SVM lineal con categoría positiva 'Maligno' obtiene una precisión de 0.958 y una sensibilidad y especificidad de 0.971 y 0.95 respectivamente.

Obtenemos datos similares empleando el paquete `caret` pero sin 5-fold crossvalidation.

Por último empleamos bootstrap con el modelo SVM `svmRadial`:

```
# Por defecto es Bootstrap, con 25 repeticiones para 3 posibles decay
# y 3 posibles sizes
set.seed(params$seed.clsfier)
model.svm.caret3 <- train(diagnosis ~ ., datos.train, method='svmRadial', trace = FALSE) # train
# we also add parameter 'preProc = c("center", "scale")' at train() for centering and scaling the data

prediction.boots <- predict(model.svm.caret3, datos.test)      # predict
res.svm.boots <- table(prediction.boots, datos.test$diagnosis)  # compare

res.svm.boots.matrix <- confusionMatrix(res.svm.boots, positive="Maligno") # compare
res.svm.boots.matrix
```

Confusion Matrix and Statistics

```
prediction.boots Benigno Maligno
      Benigno      112         5
      Maligno         3      70
```

```
      Accuracy : 0.9579
      95% CI : (0.9187, 0.9816)
No Information Rate : 0.6053
P-Value [Acc > NIR] : <2e-16
```

```
      Kappa : 0.9115
```

```
Mcnemar's Test P-Value : 0.7237
```

```

      Sensitivity : 0.9333
      Specificity : 0.9739
      Pos Pred Value : 0.9589
      Neg Pred Value : 0.9573
      Prevalence : 0.3947
      Detection Rate : 0.3684
      Detection Prevalence : 0.3842
      Balanced Accuracy : 0.9536

      'Positive' Class : Maligno

```

El modelo de SVM radial con validación bootstrap obtiene una precisión de 0.958 y una sensibilidad y especificidad de 0.933 y 0.974 respectivamente.

4.5. Árbol de Decisión

4.5.1. Transformación de los datos

El algoritmo de árboles de decisión no requiere realizar transformaciones normalizantes de los datos, empleamos los datos de entrenamiento y test creados en el apartado 3.3.

4.5.2. Entrenar el modelo

```

#fijar la semilla para el clasificador
library(C50)
set.seed(params$seed.clsfier)
data.model.ab<- C5.0(datos.train[-31], datos.train$diagnosis)
data.model.ab

```

```

Call:
C5.0.default(x = datos.train[-31], y = datos.train$diagnosis)

```

```

Classification Tree
Number of samples: 379
Number of predictors: 30

```

```

Tree size: 8

```

```

Non-standard options: attempt to group attributes

```

Para ver todos los detalles del árbol de decisiones creado podemos usar **summary**:

```

res.ab<-summary(data.model.ab)
res.ab

```

```

Call:
C5.0.default(x = datos.train[-31], y = datos.train$diagnosis)

```

```

C5.0 [Release 2.07 GPL Edition]      Tue Jun 09 01:36:11 2020
-----

```

```

Class specified by attribute `outcome'

```

```

Read 379 cases (31 attributes) from undefined.data

```


Decision tree:

```
perimeter_worst > 115.9: Maligno (111)
perimeter_worst <= 115.9:
...concave.points_worst > 0.1466:
  ...texture_worst <= 23.73: Benigno (6/1)
  : texture_worst > 23.73: Maligno (18/1)
concave.points_worst <= 0.1466:
...perimeter_worst <= 102.3: Benigno (212/1)
  perimeter_worst > 102.3:
  ...texture_mean <= 21.01: Benigno (21/1)
    texture_mean > 21.01:
    ...radius_worst > 16.67: Maligno (4)
      radius_worst <= 16.67:
      ...texture_mean <= 22.61: Maligno (2)
        texture_mean > 22.61: Benigno (5)
```

Evaluation on training data (379 cases):

```
      Decision Tree
-----
Size      Errors

      8      4( 1.1%)  <<

(a)  (b)    <-classified as
----  ----
241    1    (a): class Benigno
  3   134   (b): class Maligno
```

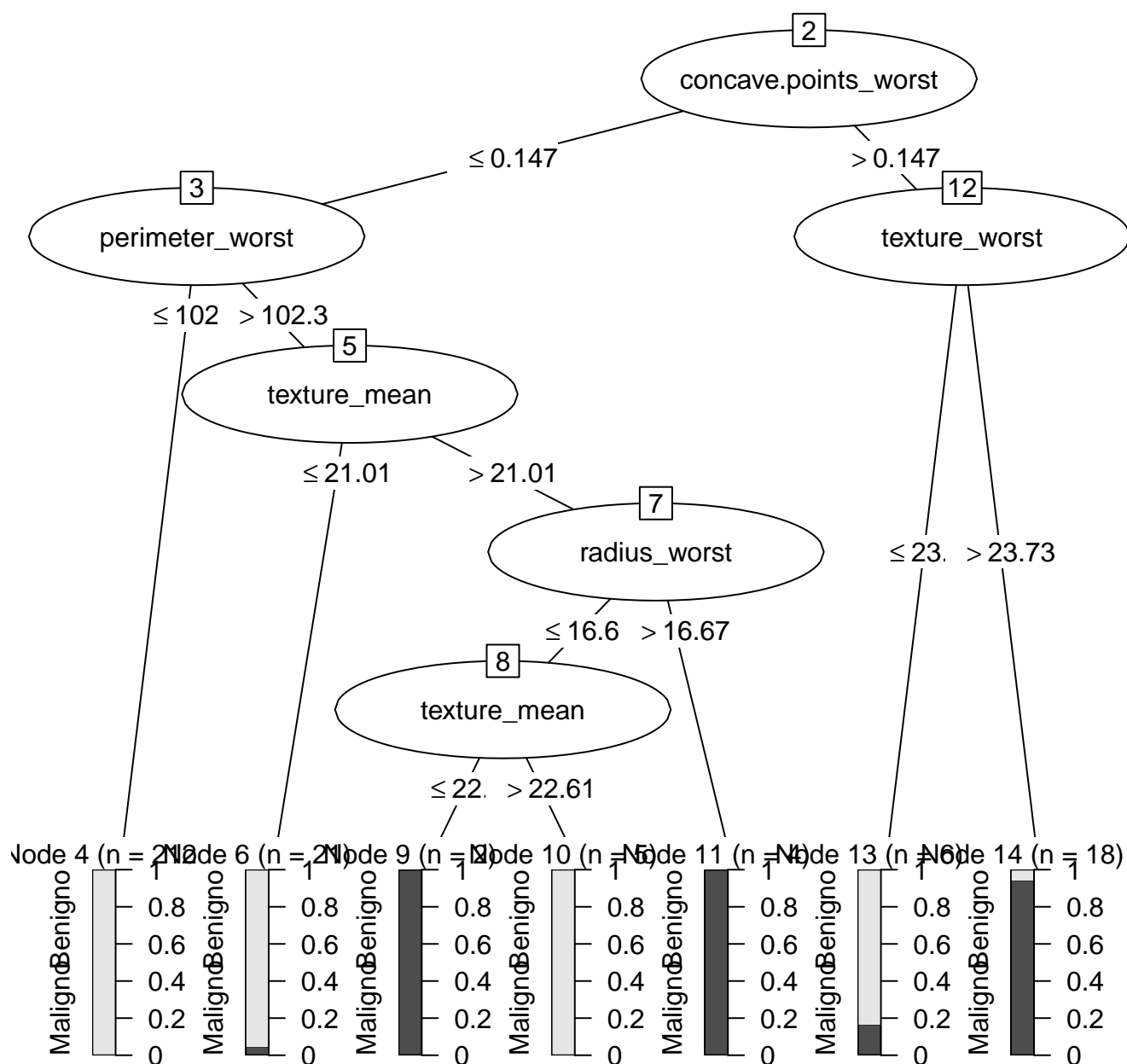
Attribute usage:

```
100.00% perimeter_worst
 70.71% concave.points_worst
  8.44% texture_mean
  6.33% texture_worst
  2.90% radius_worst
```

Time: 0.0 secs

El resultado anterior muestra las ramas en el árbol de decisión. Después del árbol, el resultado del resumen muestra una **matriz de confusión**, que nos indica los registros incorrectamente clasificados por el modelo. Se observa que este modelo clasifica adecuadamente el diagnóstico de todas las muestras excepto 1 de las 379 muestras, con una tasa de error del 1,1 %. Un total de 134 valores reales no se clasificaron incorrectamente como (falsos positivos), mientras que 3 valores se clasificaron erróneamente como Malignos (falsos negativos).

```
plot(data.model.ab, subtree=2)
```



4.5.3. Predicción y Evaluación del algoritmo

La salida anterior debemos validarla con los datos de test:

```
data.predict.ab<- predict(data.model.ab, datos.test)
```

Creamos la tabla de resultados con la función `confusionMatrix()` de `caret`:

```
cf.arbol<-confusionMatrix(data.predict.ab, datos.test$diagnosis, positive = "Maligno")
cf.arbol
```

Confusion Matrix and Statistics

	Reference	
Prediction	Benigno	Maligno
Benigno	110	8

Cuadro 1: ANÁLISIS USANDO ÁRBOLES DE CLASIFICACIÓN

modelo	Accuracy	Kappa	Sensitivity
C5.0	0.932	0.856	0.89

Maligno 5 67

Accuracy : 0.9316

95% CI : (0.8858, 0.9631)

No Information Rate : 0.6053

P-Value [Acc > NIR] : <2e-16

Kappa : 0.8558

Mcnemar's Test P-Value : 0.5791

Sensitivity : 0.8933

Specificity : 0.9565

Pos Pred Value : 0.9306

Neg Pred Value : 0.9322

Prevalence : 0.3947

Detection Rate : 0.3526

Detection Prevalence : 0.3789

Balanced Accuracy : 0.9249

'Positive' Class : Maligno

De las 0 muestras incluidas en los datos de test, nuestro modelo predijo correctamente que 110 como **Maligno** y 67 como **Benigno**, lo que resulta una precisión del % y una tasa de error del %.

Este resultado es peor que su rendimiento en los datos de entrenamiento, pero no es inesperado, dado que el rendimiento de un modelo a menudo es peor en datos no vistos.

Si nos fijamos en **Sensitivity**, sensibilidad del modelo, vemos que es 0.89, y si nos fijamos en **Specificity**, ratio de verdaderos negativos, que nos mide la proporción de negativos que han sido correctamente clasificados, vemos que es 0.96.

En resumen:

4.5.3.1. Posible mejora: algoritmo C 4.5

Comprobamos si empleando el algoritmo C 4.5, mediante la adición de refuerzo adaptativo, mejora el modelo. Este es un proceso en el que se construyen muchos árboles de decisión y los árboles votan en la mejor clase para cada ejemplo.

La función `C5.0()` facilita agregar aumentos a nuestro árbol de decisión C5.0. Simplemente necesitamos agregar un parámetro adicional **trials** que indique el **número de árboles de decisión separados** para usar en el modelo. Este parámetro establece un límite superior; el algoritmo dejará de agregar árboles si reconoce que las pruebas adicionales no parecen mejorar la precisión. Comenzaremos con **trials=10**, un número qestándar, ya que las investigaciones sugieren que esto reduce las tasas de error en los datos de prueba en aproximadamente un 25 por ciento:

```
#fijar la semilla para el clasificador
set.seed(params$seed.clsfier)
model.boost10 <- C5.0(datos.train[-31], datos.train$diagnosis, trials = 10)
summary(model.boost10)
```

```
Call:
C5.0.default(x = datos.train[-31], y = datos.train$diagnosis, trials = 10)
```

```
C5.0 [Release 2.07 GPL Edition]      Tue Jun 09 01:36:11 2020
-----
```

Class specified by attribute `outcome'

Read 379 cases (31 attributes) from undefined.data

----- Trial 0: -----

Decision tree:

```
perimeter_worst > 115.9: Maligno (111)
perimeter_worst <= 115.9:
...concave.points_worst > 0.1466:
  ...texture_worst <= 23.73: Benigno (6/1)
  : texture_worst > 23.73: Maligno (18/1)
concave.points_worst <= 0.1466:
...perimeter_worst <= 102.3: Benigno (212/1)
  perimeter_worst > 102.3:
  ...texture_mean <= 21.01: Benigno (21/1)
    texture_mean > 21.01:
    ...radius_worst > 16.67: Maligno (4)
      radius_worst <= 16.67:
      ...texture_mean <= 22.61: Maligno (2)
        texture_mean > 22.61: Benigno (5)
```

----- Trial 1: -----

Decision tree:

```
area_worst <= 645.8: Benigno (161.2/0.8)
area_worst > 645.8:
...concavity_worst <= 0.206: Benigno (28.6/0.8)
  concavity_worst > 0.206:
  ...texture_mean <= 15.51: Benigno (9.8/1.5)
    texture_mean > 15.51: Maligno (179.4/9)
```

----- Trial 2: -----

Decision tree:

```
concave.points_worst > 0.1599: Maligno (92.6/0.6)
concave.points_worst <= 0.1599:
...radius_worst <= 16.51: Benigno (230.3/21.8)
  radius_worst > 16.51: Maligno (56.1/15.4)
```

----- Trial 3: -----

Decision tree:

```
concave.points_worst > 0.1708: Maligno (66.2)
concave.points_worst <= 0.1708:
```

```

:...texture_worst <= 29.51: Benigno (223/28.8)
  texture_worst > 29.51:
:...radius_worst <= 14.42: Benigno (12)
  radius_worst > 14.42:
:...compactness_se > 0.0431: Benigno (4.9)
  compactness_se <= 0.0431:
:...smoothness_worst <= 0.1086: Benigno (6.2)
  smoothness_worst > 0.1086: Maligno (66.6/1.3)

```

----- Trial 4: -----

Decision tree:

```

concave.points_mean > 0.04908:
:...area_se <= 18.04: Benigno (18.5)
:  area_se > 18.04: Maligno (173.1/7.5)
concave.points_mean <= 0.04908:
:...symmetry_worst <= 0.2827: Benigno (105.4/0.7)
  symmetry_worst > 0.2827:
:...fractal_dimension_mean <= 0.06065: Maligno (41.3/7.5)
  fractal_dimension_mean > 0.06065: Benigno (40.7/0.3)

```

----- Trial 5: -----

Decision tree:

```

perimeter_worst > 115.9: Maligno (79.7)
perimeter_worst <= 115.9:
:...smoothness_worst <= 0.1408: Benigno (216/24.8)
  smoothness_worst > 0.1408:
:...area_worst <= 645.8: Benigno (21.5)
  area_worst > 645.8: Maligno (61.8/6.4)

```

----- Trial 6: -----

Decision tree:

```

area_se <= 18.51: Benigno (74.4)
area_se > 18.51:
:...concavity_worst <= 0.1546: Benigno (33.5)
  concavity_worst > 0.1546:
:...texture_worst > 25.82:
  ...symmetry_mean <= 0.1516: Benigno (12/2.9)
  :  symmetry_mean > 0.1516: Maligno (150.5/6.7)
  texture_worst <= 25.82:
  ...perimeter_worst > 116.6: Maligno (23)
    perimeter_worst <= 116.6:
    ...fractal_dimension_worst <= 0.09782: Benigno (59)
      fractal_dimension_worst > 0.09782: Maligno (26.6/9.6)

```

----- Trial 7: -----

Decision tree:

```

area_worst > 862.1:
:...radius_se <= 0.2387: Benigno (6/0.2)

```

```

:   radius_se > 0.2387: Maligno (94.5)
area_worst <= 862.1:
...perimeter_worst <= 91.88: Benigno (106.2)
   perimeter_worst > 91.88:
     ...perimeter_worst <= 92.04: Maligno (16.1/0.3)
       perimeter_worst > 92.04:
         ...concave.points_worst <= 0.1108: Benigno (48.8)
           concave.points_worst > 0.1108:
             ...compactness_se > 0.03889: Benigno (32/0.5)
               compactness_se <= 0.03889:
                 ...texture_worst <= 20.35: Benigno (16.7)
                   texture_worst > 20.35: Maligno (58.7/9.4)

```

----- Trial 8: -----

Decision tree:

```

perimeter_worst > 115.9: Maligno (65.3)
perimeter_worst <= 115.9:
...concave.points_worst > 0.1712: Maligno (31)
   concave.points_worst <= 0.1712:
     ...texture_mean <= 21.57: Benigno (201.2/11.1)
       texture_mean > 21.57:
         ...area_worst <= 639.1: Benigno (21.2)
           area_worst > 639.1: Maligno (60.2/13.7)

```

----- Trial 9: -----

Decision tree:

```

concave.points_mean <= 0.04908:
...radius_worst <= 16.82: Benigno (206.4/9.4)
:   radius_worst > 16.82: Maligno (16.1/3.4)
concave.points_mean > 0.04908:
...area_se <= 18.04: Benigno (15)
   area_se > 18.04:
     ...compactness_se > 0.07217: Benigno (9.1/0.3)
       compactness_se <= 0.07217:
         ...texture_worst <= 20.35: Benigno (8.6/2.1)
           texture_worst > 20.35: Maligno (123.9/1.8)

```

Evaluation on training data (379 cases):

Trial	Decision Tree	
	Size	Errors
0	8	4(1.1%)
1	4	16(4.2%)
2	3	15(4.0%)
3	6	25(6.6%)
4	5	37(9.8%)
5	4	11(2.9%)
6	7	25(6.6%)
7	8	11(2.9%)

```

      8      5    16( 4.2%)
      9      6    14( 3.7%)
boost      0( 0.0%)  <<

```

```

      (a)  (b)  <-classified as
-----  -----
      242      (a): class Benigno
           137  (b): class Maligno

```

Attribute usage:

```

100.00% concave.points_mean
100.00% area_se
100.00% texture_worst
100.00% perimeter_worst
100.00% area_worst
100.00% concave.points_worst
99.74% texture_mean
75.46% radius_worst
75.20% smoothness_worst
74.93% concavity_worst
61.48% symmetry_worst
44.06% compactness_se
33.25% symmetry_mean
31.66% radius_se
21.64% fractal_dimension_mean
11.08% fractal_dimension_worst

```

Time: 0.0 secs

El clasificador cometió un error de 4 en 379 registros de datos de entrenamiento para una tasa de error de 1.06 por ciento. Representa una gran mejora con respecto a la tasa de error del modelo anterior antes de agregar potenciación. Veamos queda por ver si hay una mejora en los datos de prueba:

```

model.boost.pred10 <- predict(model.boost10, datos.test)

cf.arbol.10<-confusionMatrix(model.boost.pred10, datos.test$diagnosis, positive = "Maligno")
cf.arbol.10

```

Confusion Matrix and Statistics

```

      Reference
Prediction Benigno Maligno
Benigno      109      7
Maligno       6      68

      Accuracy : 0.9316
      95% CI : (0.8858, 0.9631)
No Information Rate : 0.6053
P-Value [Acc > NIR] : <2e-16

      Kappa : 0.8565

```

McNemar's Test P-Value : 1

```

      Sensitivity : 0.9067
      Specificity : 0.9478
      Pos Pred Value : 0.9189
      Neg Pred Value : 0.9397
      Prevalence : 0.3947
      Detection Rate : 0.3579
      Detection Prevalence : 0.3895
      Balanced Accuracy : 0.9272

      'Positive' Class : Maligno

```

La precisión se ha mantenido respecto al modelo C5.0 sin boosting. Sin embargo la sensibilidad mejora levemente.

Accuracy es de 0.932

Kappa es de 0.856

4.5.4. Paquete caret

Empleamos los mismos grupos de train y test que hemos empleado en el análisis anterior. Para ajustar el modelo se emplea la función `train` y empleamos 5-fold crossvalidation para entrenarlo:

```

## método : cv K-fold cross validation
## número : K folds
ctrl <- trainControl( method="cv",
                      number=5,
                      selectionFunction= "oneSE")

## Parámetros de Grid para algoritmo de C50
## trials -> número de iteraciones boosting
grid_C50 <- expand.grid(model="tree", trials=c(5,10,20,30),winnow="FALSE")

# fijar la semilla para el clasificador
set.seed(params$seed.clsfier)
## trace <- FALSE para suprimir las iteraciones de entrenamiento
modeloC5.0 <- train (diagnosis ~ .,
                    data = datos.train,
                    method ="C5.0",
                    trControl=ctrl,
                    tuneGrid = grid_C50,
                    metric="Accuracy",
                    preProc = c("center", "scale"),
                    verbose =FALSE,
                    trace = FALSE)

modeloC5.0

```

C5.0

```

379 samples
30 predictor
2 classes: 'Benigno', 'Maligno'

```

No pre-processing

Resampling: Cross-Validated (5 fold)

Summary of sample sizes: 302, 303, 304, 304, 303

Resampling results across tuning parameters:

trials	Accuracy	Kappa
5	0.9683140	0.9305421


```

10      0.9709114  0.9362221
20      0.9710148  0.9370581
30      0.9683140  0.9309235

```

Tuning parameter 'model' was held constant at a value of tree

Tuning parameter

'winnow' was held constant at a value of FALSE

Accuracy was used to select the optimal model using the one SE rule.

The final values used for the model were trials = 5, model = tree and winnow = FALSE.

para la evaluación del rendimiento creamos la matriz de resultados:

```

# Predicción de Clases
prd.c50 <- predict ( modeloC5.0, newdata = datos.test)
# Matriz de Confusión
(cf.c50 <- confusionMatrix( data=prd.c50, datos.test$diagnosis))

```

Confusion Matrix and Statistics

	Reference	
Prediction	Benigno	Maligno
Benigno	113	7
Maligno	2	68

Accuracy : 0.9526

95% CI : (0.912, 0.9781)

No Information Rate : 0.6053

P-Value [Acc > NIR] : <2e-16

Kappa : 0.8997

Mcnemar's Test P-Value : 0.1824

Sensitivity : 0.9826

Specificity : 0.9067

Pos Pred Value : 0.9417

Neg Pred Value : 0.9714

Prevalence : 0.6053

Detection Rate : 0.5947

Detection Prevalence : 0.6316

Balanced Accuracy : 0.9446

'Positive' Class : Benigno

La precisión ha aumentado respecto al modelo anterior sin caret.

Accuracy es de 0.953

Kappa es de 0.9.

4.6. Random Forest

4.6.1. Transformacion de los datos

El algoritmo de *Random Forest* no requiere realizar transformaciones normalizantes de los datos, empleamos los datos de entrenamiento y test creados en el apartado 3.3 y empleados en la mayoría de los algoritmos.

4.6.2. Entrenar el modelo

La función `randomForest()` crea por defecto un conjunto de 500 árboles, donde cada uno de ellos elige \sqrt{p} variables de forma aleatoria para cada árbol, donde p es el número de variables en el conjunto de datos de entrenamiento.

El objetivo de utilizar una gran cantidad de árboles es entrenar lo suficiente para que cada variable tenga la oportunidad de aparecer en varios modelos, base del parámetro `mtry`. El uso de este valor limita las variables lo suficiente como para que ocurra una variación aleatoria sustancial de árbol a árbol.

Empleamos como en los casos anteriores la variable factor “diagnosis” para que el método ejecutado sea **classification**, de lo contrario se ejecuta **regresion**:

```
library(randomForest)
set.seed(params$seed.clsfier)
randomf <- randomForest(diagnosis ~ ., data = datos.train)
randomf
```

Call:

```
randomForest(formula = diagnosis ~ ., data = datos.train)
      Type of random forest: classification
      Number of trees: 500
```

No. of variables tried at each split: 5

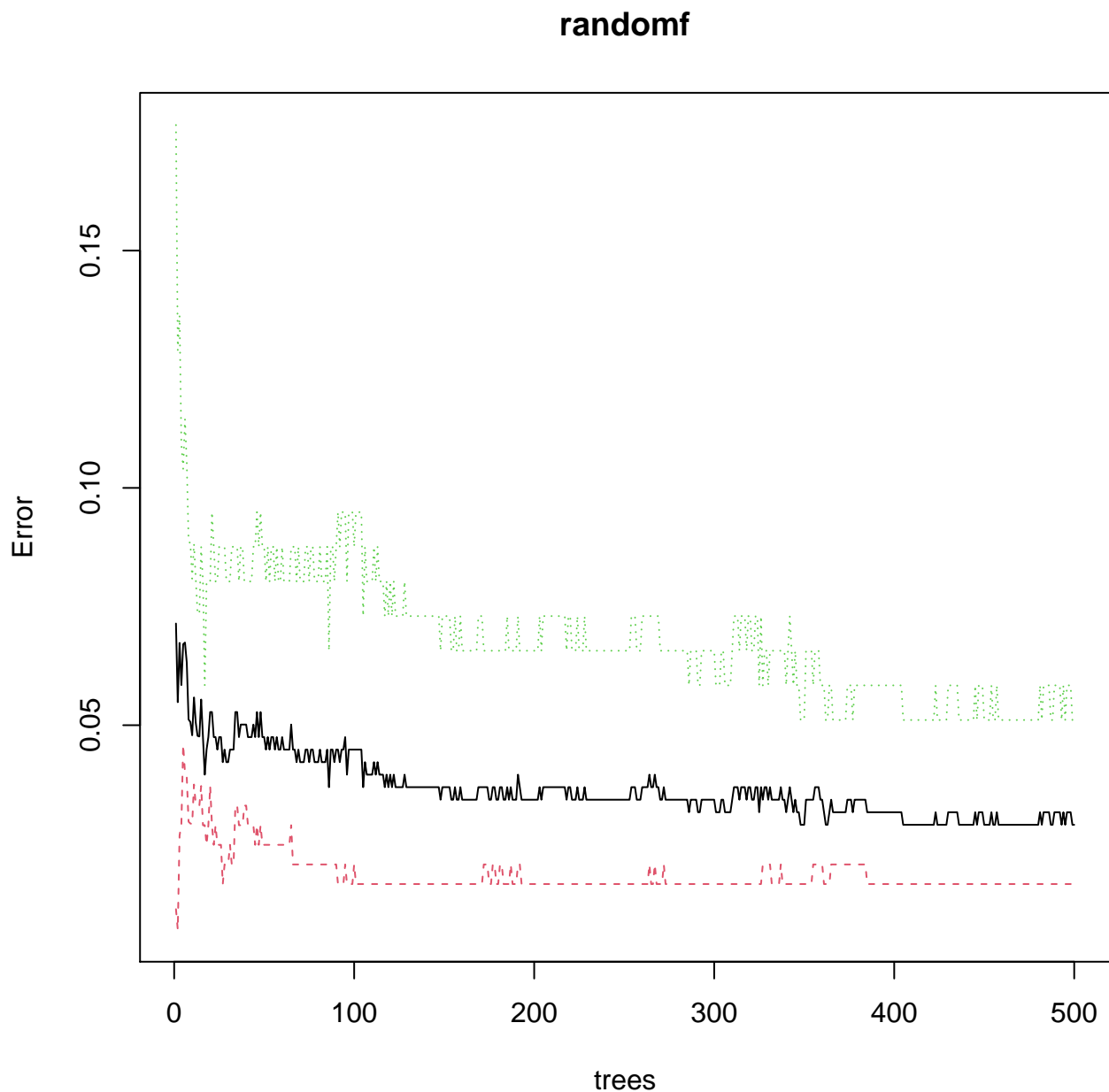
OOB estimate of error rate: 2.9%

Confusion matrix:

	Benigno	Maligno	class.error
Benigno	238	4	0.01652893
Maligno	7	130	0.05109489

El resultado indica que *Random Forest* incluyó 500 árboles y probó 5 variables en cada división. Hagamos un gráfico del modelo:

```
plot(randomf)
```



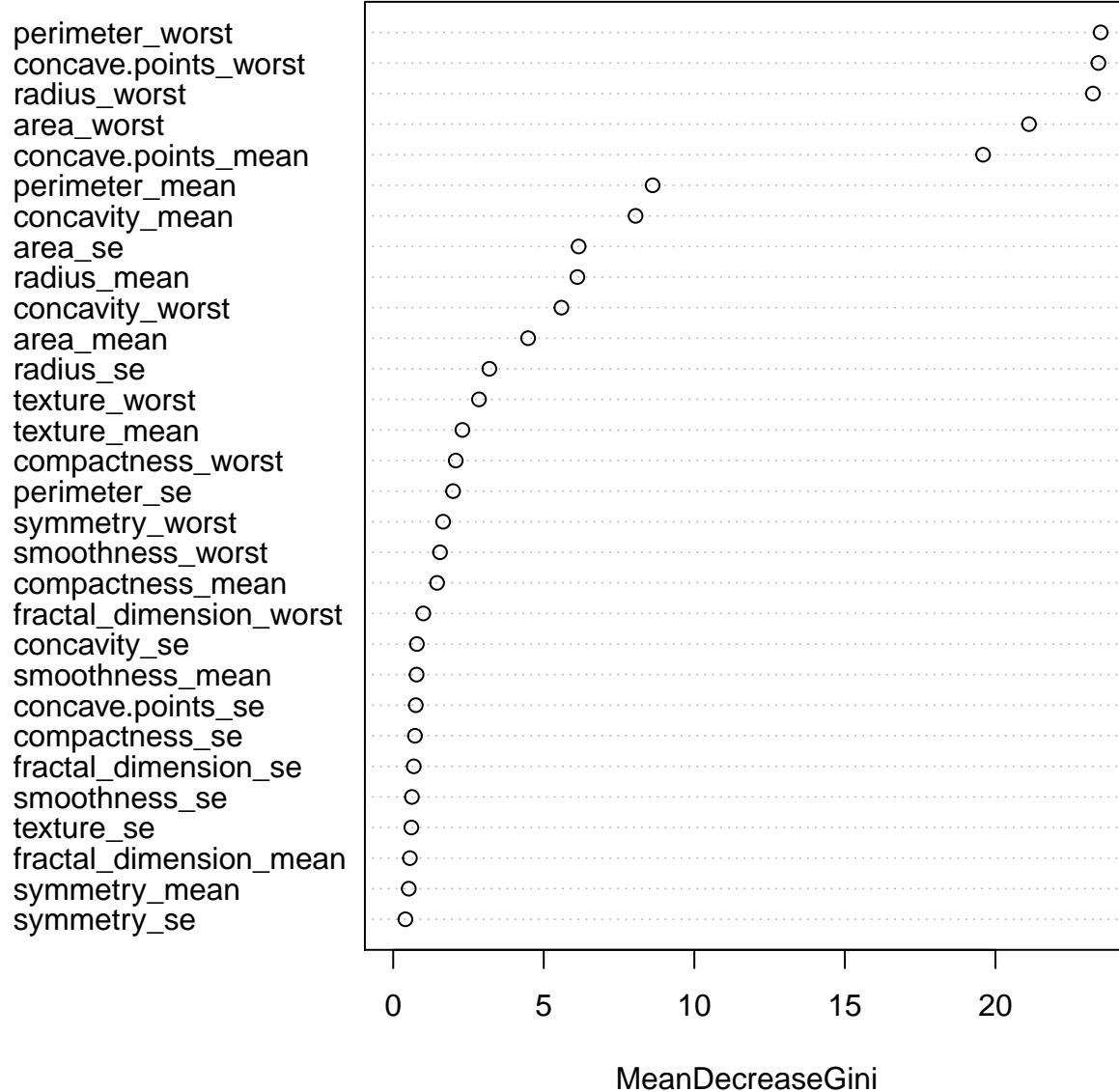
Donde la línea roja representa el error cometido al intentar predecir el diagnóstico, y la línea verde es el error en la predicción **Benigno**. Podemos ver la importancia de cada una de las variables en la clasificación del modelo:

```
head(importance(randomf))
```

	MeanDecreaseGini
radius_mean	6.1171750
texture_mean	2.2982140
perimeter_mean	8.6155845
area_mean	4.4807408
smoothness_mean	0.7773583
compactness_mean	1.4580878

```
varImpPlot(randomf)
```

randomf



4.6.3. Predicción y Evaluación del algoritmo

Creamos para ello la matriz de los resultados obtenidos en la predicción del diagnóstico:

```
library(caret)
predict.rf<- predict(randomf, datos.test)
rf.matrix <- confusionMatrix(datos.test$diagnosis, predict.rf, positive = "Maligno")
rf.matrix
```

Confusion Matrix and Statistics

	Reference	
Prediction	Benigno	Maligno
Benigno	112	3
Maligno	8	67

```

Accuracy : 0.9421
95% CI : (0.8988, 0.9707)
No Information Rate : 0.6316
P-Value [Acc > NIR] : <2e-16

```

```
Kappa : 0.8774
```

```
McNemar's Test P-Value : 0.2278
```

```

Sensitivity : 0.9571
Specificity : 0.9333
Pos Pred Value : 0.8933
Neg Pred Value : 0.9739
Prevalence : 0.3684
Detection Rate : 0.3526
Detection Prevalence : 0.3947
Balanced Accuracy : 0.9452

```

```
'Positive' Class : Maligno
```

Como vemos la precisión (Accuracy) es de 0.942 y Kappa es de 0.877.

4.6.3.1. Posible mejora: aumento del número de árboles

Veamos si incrementando el número de árboles el modelo mejora. Entrenamos de nuevo el modelo:

```

set.seed(params$seed.clsfier)
rf.1000 <- randomForest(diagnosis ~ ., data = datos.train, ntree=1000)
rf.1000

```

Call:

```

randomForest(formula = diagnosis ~ ., data = datos.train, ntree = 1000)
Type of random forest: classification
Number of trees: 1000

```

No. of variables tried at each split: 5

```
OOB estimate of error rate: 2.9%
```

Confusion matrix:

```

Benigno Maligno class.error
Benigno    238      4 0.01652893
Maligno      7    130 0.05109489

```

```

predict.rf<- predict(rf.1000, datos.test)
rf.matrix1000 <- confusionMatrix(datos.test$diagnosis, predict.rf)
rf.matrix1000

```

Confusion Matrix and Statistics

```

Reference
Prediction Benigno Maligno
Benigno    112      3
Maligno      7    68

```

```

Accuracy : 0.9474
95% CI : (0.9053, 0.9745)
No Information Rate : 0.6263

```

```

P-Value [Acc > NIR] : <2e-16

      Kappa : 0.8888

McNemar's Test P-Value : 0.3428

      Sensitivity : 0.9412
      Specificity : 0.9577
      Pos Pred Value : 0.9739
      Neg Pred Value : 0.9067
      Prevalence : 0.6263
      Detection Rate : 0.5895
      Detection Prevalence : 0.6053
      Balanced Accuracy : 0.9495

      'Positive' Class : Benigno

```

El nuevo modelo obtiene una precisión de 0.947 y una sensibilidad y especificidad de 0.941 y 0.958 respectivamente. Vemos que el modelo obtenido con tres nodo tiene una mayor precision.

4.6.4. Paquete caret

Se vuelve a analizar el mismo dataset pero ahora usando el modelo **rf** del paquete **caret**. Y lo validadmos con 5-fold cross validation. Como sabemos, el fichero de train contiene 379 observaciones y el de test 190.

```

## método : repeatedcv K-fold cross validation
## number : K folds
## repeats : número de repeticiones
set.seed(params$semilla)
ctrl2 <- trainControl( method="repeatedcv",
                      number=5,
                      summaryFunction = defaultSummary,
                      verboseIter =FALSE,
                      repeats=3)

## Tunegrid para Random Forest
# mtry define cuantas variables se seleccionan al azar en cada split. Por
# defecto sqrt(n.variables)
grid_rf <- expand.grid(mtry = c(2,4,8,16))

## trace <- FALSE to suppress train iterations
modelo.rf.caret <- train (diagnosis ~ .,
                        data = datos.train,
                        method = "rf",
                        trControl=ctrl2,
                        tuneGrid = grid_rf,
                        metric="Accuracy",
                        prePoc = c("center", "scale"),
                        verbose =FALSE,
                        trace = FALSE)

```

para la evaluación del rendimiento creamos la matriz de resultados:

```

# Predicción del diagnóstico
predict.rf2 <- predict(modelo.rf.caret, newdata = datos.test)
# Matriz de Confusión
(cfrf <- confusionMatrix(data=predict.rf2, datos.test$diagnosis))

```

Confusion Matrix and Statistics

```

      Reference
Prediction Benigno Maligno
Benigno    112      7
Maligno     3     68

Accuracy : 0.9474
95% CI : (0.9053, 0.9745)
No Information Rate : 0.6053
P-Value [Acc > NIR] : <2e-16

Kappa : 0.8888

Mcnemar's Test P-Value : 0.3428

Sensitivity : 0.9739
Specificity : 0.9067
Pos Pred Value : 0.9412
Neg Pred Value : 0.9577
Prevalence : 0.6053
Detection Rate : 0.5895
Detection Prevalence : 0.6263
Balanced Accuracy : 0.9403

'Positive' Class : Benigno
```

La precisión es similar al modelo anterior sin el paquete caret

Accuracy es de 0.947

Kappa es de 0.889

5. Conclusión y discusión

5.1. Rendimiento, interpretabilidad, ... de los algoritmos para el problema tratado

5.2. Decisión del modelo más apropiado

Para poder tomar una decisión sobre cuál o cuáles podrían ser los mejores modelos a aplicar en este caso, realizo una tabla resumen con los principales parámetros obtenidos de la función `confusionMatrix` del paquete `caret` en cada uno de ellos y posteriormente los unimos en una única tabla. Muestro aquí sólo el código empleado en los tres primeros casos, en el resto empleamos el mismo código:

```
resum1 <- data.frame(modelo=NA, Accuracy=NA, Kappa=NA, Sensitivity=NA, Specificity=NA)
resum1[1,1:5] <- c("knn",round(cf.knn$overall[1],3),round(cf.knn$overall[2],3), round(cf.knn$byClass[1],2),
tabla1 <- as.data.frame(resum1[,1:5])

resum2 <- data.frame(modelo=NA, Accuracy=NA, Kappa=NA, Sensitivity=NA, Specificity=NA)
resum2[1,1:5] <- c("knn z transf",round(cf.knn.z$overall[1],3),round(cf.knn.z$overall[2],3), round(cf.knn.z$byClass[1],2),
tabla2 <- as.data.frame(resum2[,1:5])

resum3 <- data.frame(modelo=NA, Accuracy=NA, Kappa=NA, Sensitivity=NA, Specificity=NA)
resum3[1,1:5] <- c("Naive-Bayes L0",round(nb.matrix$overall[1],3),round(nb.matrix$overall[2],3), round(nb.matrix$byClass[1],2),
tabla3 <- as.data.frame(resum3[,1:5])
```

Orneamos los datos según los valores de precisión y especificidad. Como vemos esta lista cambia según ordenemos los valores teniendo en cuenta ambos parámetros al mismo tiempo, o poniendo uno de ellos como principal factor a tener en cuenta y tras ordenar la lista por uno de ellos (precisión), la ordenamos por sensibilidad.

```

resultado.final <- data.frame(rbind(tabla1, tabla2, resum, tabla3, tabla4, tabla5, tabla6, tabla7, tabla8,

library(dplyr)
resultado.final %>%
arrange(Accuracy, Sensitivity) %>%
tail()

```

	modelo	Accuracy	Kappa	Sensitivity	Specificity
20	11	0.958	0.91	0.893	1
21	SVM Gauss	0.958	0.911	0.93	0.974
22	SVM Caret boots	0.958	0.911	0.93	0.974
23	SVM Caret 5fold	0.958	0.91	0.97	0.95
24	Naive-Bayes L1	0.963	0.922	0.93	0.983
25	Naive-Bayes L0	0.968	0.934	0.95	0.983

```

resultado.final %>%
arrange(Accuracy) %>%
arrange(Sensitivity) %>%
tail()

```

	modelo	Accuracy	Kappa	Sensitivity	Specificity
20	ANN 1 nodo	0.937	0.869	0.95	0.93
21	Naive-Bayes L0	0.968	0.934	0.95	0.983
22	Random forest	0.942	0.877	0.96	0.933
23	Random forest caret	0.947	0.889	0.97	0.907
24	SVM Caret 5fold	0.958	0.91	0.97	0.95
25	Arbol decision caret	0.953	0.9	0.98	0.907

Por ello, el mejor modelo a aplicar dependerá de la importancia que le demos tanto a la precisión como a la sensibilidad y especificidad en la detección del cáncer de mama según nuestros resultados. La sensibilidad caracteriza la capacidad de la prueba para detectar la enfermedad en sujetos enfermos, lo que tras la precisión quizá debería de ser el factor a tener en cuenta y ante igualdad, un mejor valor de especificidad. El modelo con mejor precisión es *Naive-Bayes L0* sin embargo hay otros modelos con mejor sensibilidad y valores parecidos de precisión. El modelo *SVM Caret 5fold* tiene una sensibilidad muy elevada, aunque pierde levemente en precisión. *Random forest* tiene una elevada precisión y sensibilidad pero un valor muy bajo de **kappa**.

Por ello, según estos resultados, y los objetivos que buscamos con nuestros análisis tanto el modelo *SVM Caret 5fold* como *Naive-Bayes L0* serían buenos modelos y con una precisión cercana al 95 % tanto en precisión como en sensibilidad y con valores de especificidad aceptables. Aunque el modelo que tiene mejores valores en cada uno de los factores es el **árbol de decisión creado con el paquete caret y 5-cross validation**.

Bibliografía

Core Team, RCTR, and others. 2013. “R: A Language and Environment for Statistical Computing.” *R Foundation for Statistical Computing, Vienna*.

Github, Inc. 2016. “GitHub.”

Lantz, Brett. 2015. *Machine Learning with R*. Packt Publishing Ltd. <http://www.packtpub.com/books/content/machine-learning-r>.