PEC3

Rashid Babiker Sánchez

12 de junio, 2020

Contents

Repositorio Github

Se han probado los algoritmos en distintas condiciones que no se muestran en el informe por agilizar la lectura y la reproducción del código. A lo largo del informe se hace referencia a estos resultados para justificar las decisiones tomadas, se pueden consultar en el siguiente repositorio de github: https://github.com/RashBabiker/PEC3_ML

Librerías usadas

```
knitr::opts_chunk$set(cache = TRUE)
options(scipen=999)
library(tidyverse); library(caret); library(ggfortify); library(doParallel); library(beepr);
library(vegan); library(party); library(knitr)
```

Análisis descriptivo de los datos y control de calidad

```
data <- read.csv(params$datos)
data$id <- NULL # no es una información que necesite
```

Los datos están completos, no falta información de ninguna variable, pero están desapareados, hay más diagnósticos benignos que malignos.

```
sum(is.na(data))
```

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

```
kable(table(data$diagnosis))
```

Var1	Freq
B	357
M	212

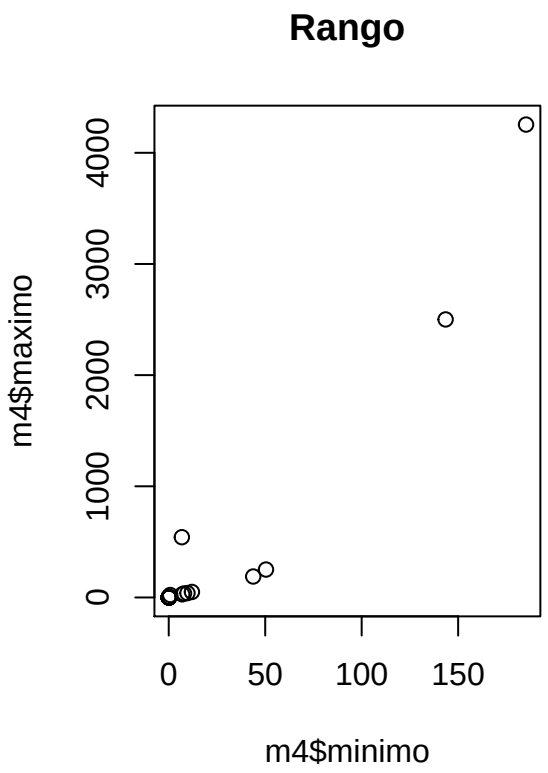
La siguiente tabla muestra el rango de valores de cada variable, así como su media y su mediana:

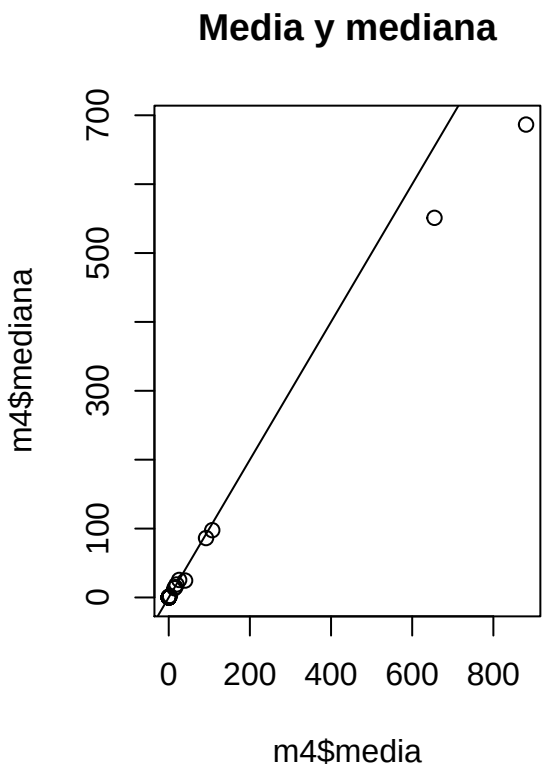
```
m4 <- sapply(data[,1:30], function (x){c(min(x), max(x), mean(x), median(x))}) %>%
  t() %>% as.data.frame()
colnames(m4) <- c("minimo", "maximo", "media", "mediana")
kable(round(m4,3))
```

	minimo	maximo	media	mediana
radius_mean	6.981	28.110	14.127	13.370
texture_mean	9.710	39.280	19.290	18.840
perimeter_mean	43.790	188.500	91.969	86.240
area_mean	143.500	2501.000	654.889	551.100
smoothness_mean	0.053	0.163	0.096	0.096
compactness_mean	0.019	0.345	0.104	0.093
concavity_mean	0.000	0.427	0.089	0.062
concave.points_mean	0.000	0.201	0.049	0.034
symmetry_mean	0.106	0.304	0.181	0.179
fractal_dimension_mean	0.050	0.097	0.063	0.062
radius_se	0.112	2.873	0.405	0.324
texture_se	0.360	4.885	1.217	1.108
perimeter_se	0.757	21.980	2.866	2.287
area_se	6.802	542.200	40.337	24.530
smoothness_se	0.002	0.031	0.007	0.006
compactness_se	0.002	0.135	0.025	0.020
concavity_se	0.000	0.396	0.032	0.026
concave.points_se	0.000	0.053	0.012	0.011
symmetry_se	0.008	0.079	0.021	0.019
fractal_dimension_se	0.001	0.030	0.004	0.003
radius_worst	7.930	36.040	16.269	14.970
texture_worst	12.020	49.540	25.677	25.410
perimeter_worst	50.410	251.200	107.261	97.660
area_worst	185.200	4254.000	880.583	686.500
smoothness_worst	0.071	0.223	0.132	0.131
compactness_worst	0.027	1.058	0.254	0.212
concavity_worst	0.000	1.252	0.272	0.227
concave.points_worst	0.000	0.291	0.115	0.100
symmetry_worst	0.156	0.664	0.290	0.282
fractal_dimension_worst	0.055	0.208	0.084	0.080

La media y la mediana coinciden en la mayoría de las variables, lo que sugiere que siguen una distribución normal y que no tienen outliers, a excepción de las dos variables con mayor varianza y media:

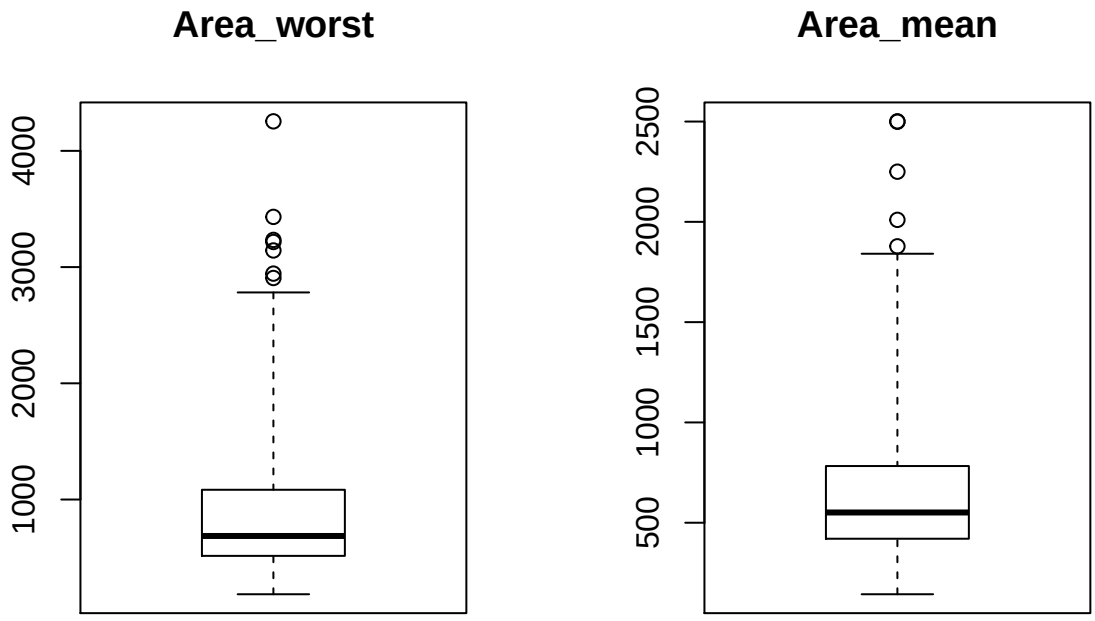
```
par(mfrow=c(1,2))
plot(m4$minimo, m4$maximo, main = "Rango")
plot(m4$media, m4$mediana, main = "Media y mediana")
abline(a=0,b=1)
```





Las variables con mayor media y mediana son area_worst y area_mean, podrían contener outliers, a continuación se representa su distribución para localizarlos, los círculos muestran valores que superan 3 veces el rango intercuartílico:

```
par(mfrow=c(1,2))
boxplot(data$area_worst, range = 3, main = "Area_worst")
boxplot(data$area_mean, range = 3, main = "Area_mean")
```



Estos valores pueden distorsionar los análisis posteriores, o pueden ser claros indicadores de cáncer.

```
#preparo los rangos intercuartílicos:
IQR_worst <- as.numeric(quantile(data$area_worst, 0.75)-quantile(data$area_worst, 0.25))
IQR_mean <- as.numeric(quantile(data$area_mean, 0.75)-quantile(data$area_mean, 0.25))
# datos sin los outliers
data_fil <- data %>% filter(area_mean<3*IQR_mean & area_worst<3*IQR_worst)
# outliers
outliers <- data %>% filter(area_mean>3*IQR_mean & area_worst>3*IQR_worst)
kable(table(outliers$diagnosis))
```

Var1	Freq
B	0
M	51

La tabla sin outliers se queda con 489 muestras de las 569 totales. Todos los outliers corresponden a tumores malignos, es decir, un área excesivamente grande es indicador de cáncer. Se han probado todos los algoritmos sin estos outliers y en general la precisión disminuye, por lo que se mantienen todos los datos. Los datos in outliers están en el repositorio de github: https://github.com/RashBabiker/PEC3_ML/blob/master/resultados/precisi%C3%B3n%20datos%20no%20outliers.csv

Normalización

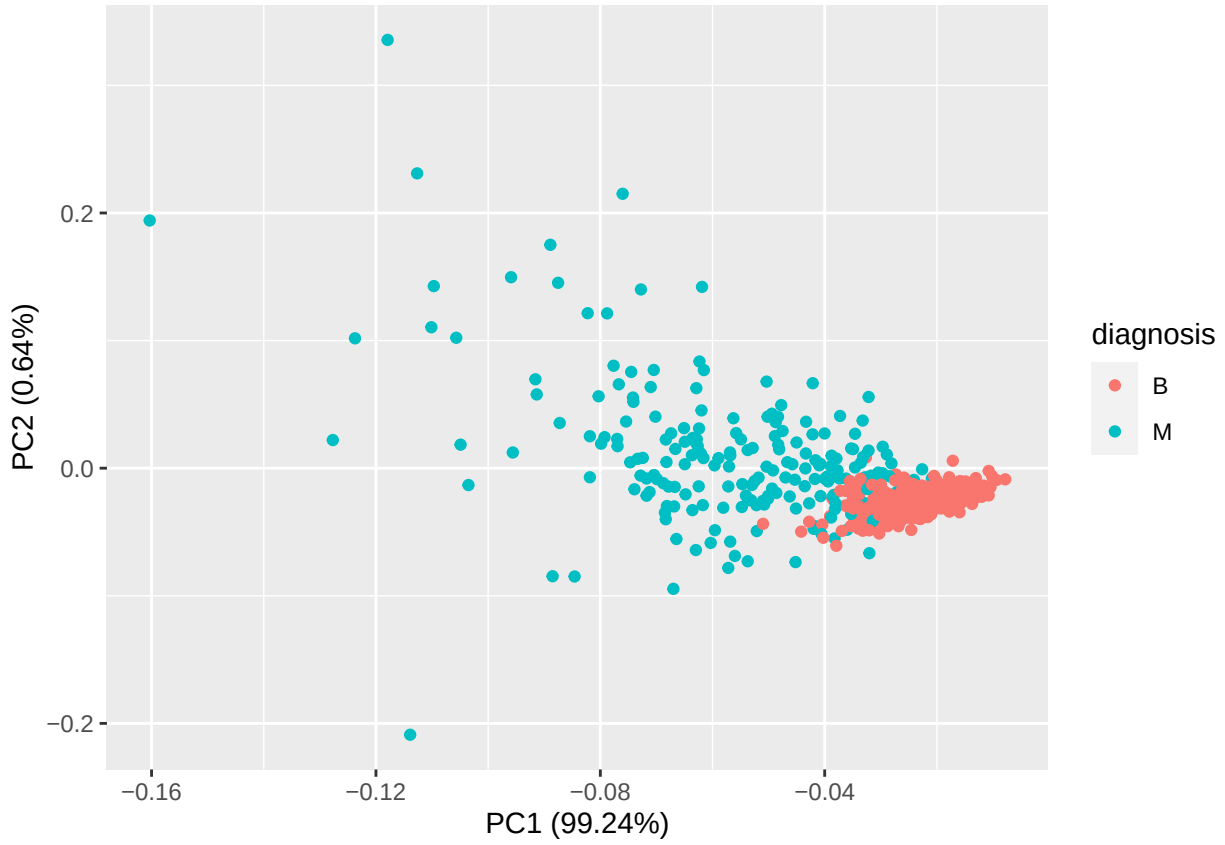
Los datos tienen distintos rangos, como en principio no hay un parámetro más importante que otro, puede ser recomendable normalizar los datos para usarlos con ciertos algoritmos. La normalización solo ha mostrado ser eficaz con el algoritmo knn, por lo que solo se usa con ese método. Los resultados usando datos normalizados están en github: https://github.com/RashBabiker/PEC3_ML/blob/master/resultados/precisi%C3%B3n%20datos%20normalizados.csv

Comparación entre tumores

Para que se puedan clasificar correctamente, las muestras deben ser diferentes, esto se confirma mediante PCA y PERMANOVA. El PCA muestra claras diferencias entre los grupos en general, aunque hay ciertos puntos donde las diferencias no están claras. El PERMANOVA muestra diferencias significativas entre los dos diagnósticos.

```
PCA <- prcomp(data[,1:30], center = F,scale. = F)

autoplot(PCA, data = data, colour = "diagnosis")
```




```
permanova <- adonis(data[,1:30]~data$diagnosis, permutations = 10000, method = "euclidean")
# p-valor, probabilidad de que siendo iguales B y M se viera esta distribución
permanova$aov.tab$`Pr(>F)`[1]
```

```
## [1] 0.00009999
```

Selección de las muestras

Se usa el método holdout para que, una vez elegidos los mejores parámetros de cada algoritmo, se pruebe la eficacia con datos que no ha visto nunca.

```
set.seed(12345)

trainIndex <- createDataPartition(data$diagnosis, p = 2/3,
                                  list = FALSE,
                                  times = 1)
train <- data[trainIndex,]
test <- data[-trainIndex,]

c(nrow(train), nrow(test))
```

```
## [1] 380 189
```

Clasificación

Para entrenar y optimizar los mejores parámetros se usará el 10-fold-cross-validation.

Como los falsos negativos son mucho más peligrosos que los falsos positivos, se ha probado a entrenar y elegir los modelos con mayor sensibilidad en el cross validation, los resultados están en la carpeta resultados del repositorio de github. Sorprendentemente al enfrentar estos modelos al test dataset, que no habían visto, no se ve una mejora respecto a los modelos entrenados basados en la precisión, incluso el modelo de neural network empeora todo lo posible, valorando todos los tumores como benignos (kappa=0). Por ello se descarta la idea.

También se han probado los modelos entrenados eligiendo los parámetros que mayor precisión mediante, y por otro lado con mayor área bajo la curva ROC (AUC), los resultados son similares, algunos algoritmos se benefician del entrenamiento seleccionando precisión y otros del entrenamiento seleccionando el mejor AUC, pero el mejor algoritmo (random forest) mantiene la misma eficacia. Se ha elegido usar el área bajo la curva porque mejora la precisión y, más importante, la sensibilidad de los árboles de decisión, que es un algoritmo fácilmente interpretable y puede ser útil; pero entrenando buscando la mayor precisión se obtienen buenos resultados también.

- Resultados entrenando sensibilidad: https://github.com/RashBabiker/PEC3_ML/blob/master/resultados/precisi%C3%B3n%20entrenando%20con%20sensibilidad.csv
- Resultados entrenando precisión: https://github.com/RashBabiker/PEC3_ML/blob/master/resultados/precisi%C3%B3n%20entrenando%20seg%C3%BAn%20precisi%C3%B3n.csv
- Los resultados entrenando el area bajo la curva ROC se encuentran en la conclusión de este informe y en este enlace: https://github.com/RashBabiker/PEC3_ML/blob/master/resultados/precisi%C3%B3n%20entrenando%20seg%C3%BAn%20ROC.csv

```
set.seed(12345)
Control <- trainControl(method = "repeatedcv", number = 10, repeats = 10, allowParallel = T,
                        summaryFunction = twoClassSummary,
                        classProbs = TRUE)
```

k-Nearest Neighbour

```
time1=Sys.time()
set.seed(12345)
model_knn <- train(diagnosis ~ ., train, method='knn',
                   trControl= Control,
# a diferencia de los otros algoritmos, knn si se beneficia de la normalización
                   preProc = c("center", "scale"),
# pruebo 10 condiciones diferentes, el programa elige la mejor.
                   tuneGrid= NULL, tuneLength=params$ntuning,
                   metric = "ROC")


prediction <- predict(model_knn, test[,-31])
res <- table(prediction, test$diagnosis)
cmatrix <- confusionMatrix(res,positive="M")

time2=Sys.time()

resumen_knn <- data.frame(precisión =  cmatrix$overall[1],
                          kappa = cmatrix$overall[2],
                          sensibilidad =  cmatrix$byClass[1],
                          especificidad = cmatrix$byClass[2],
                          tiempo = difftime(time2,time1, units = "mins"))
rownames(resumen_knn) <- "k-NN"
kable(resumen_knn)
```

	precisión	kappa	sensibilidad	especificidad	tiempo
k-NN	0.962963	0.9208211	0.9571429	0.9663866	0.2275298 mins

Naive Bayes

```
cl <- makePSOCKcluster(4)
registerDoParallel(cl)

time1=Sys.time()
set.seed(12345)
model_naive_bayes <- train(diagnosis ~ ., train, method='naive_bayes',
                   trControl= Control,
                   tuneGrid= NULL, tuneLength=params$ntuning,trace = FALSE,
                   metric = "ROC")

prediction <- predict(model_naive_bayes, test[,-31])
res <- table(prediction, test$diagnosis)
cmatrix <- confusionMatrix(res,positive="M")

time2=Sys.time()

resumen_naive_bayes <- data.frame(precisión =  cmatrix$overall[1],
                                  kappa = cmatrix$overall[2],
                                  sensibilidad =  cmatrix$byClass[1],
                                  especificidad = cmatrix$byClass[2],
                                  tiempo = difftime(time2,time1, units = "mins"))
rownames(resumen_naive_bayes) <- "Naive Bayes"
kable(resumen_naive_bayes)
```

	precisión	kappa	sensibilidad	especificidad	tiempo
Naive Bayes	0.9153439	0.820598	0.9142857	0.9159664	0.1332478 mins

```
stopCluster(cl)
```

Artificial Neural Network

```
cl <- makePSOCKcluster(4)
registerDoParallel(cl)

time1=Sys.time()
set.seed(12345)
model_nnet <- train(diagnosis ~ ., train, method='nnet',
                    trControl= Control,
                    tuneGrid= NULL, tuneLength=params$ntuning,trace = FALSE,
                    metric = "ROC")

prediction <- predict(model_nnet, test[,-31])
res <- table(prediction, test$diagnosis)
cmatrix <- confusionMatrix(res,positive="M")

time2=Sys.time()

resumen_nnet <- data.frame(precisión =  cmatrix$overall[1],
                                kappa = cmatrix$overall[2],
                                sensibilidad =  cmatrix$byClass[1],
                                especificidad = cmatrix$byClass[2],
                                tiempo = difftime(time2,time1, units = "mins"))
rownames(resumen_nnet) <- "Neural network"
kable(resumen_nnet)
```

	precisión	kappa	sensibilidad	especificidad	tiempo
Neural network	0.9470899	0.887218	0.9428571	0.9495798	8.38658 mins

```
stopCluster(cl)
```

Support Vector Machine

SVM lineal

```
cl <- makePSOCKcluster(4)
registerDoParallel(cl)

time1=Sys.time()
set.seed(12345)
model_svm_lineal <- train(diagnosis ~ ., train, method='svmLinear',
                    trControl= Control,
                    tuneGrid= NULL, tuneLength=params$ntuning,trace = FALSE,
                    metric = "ROC")

prediction <- predict(model_svm_lineal, test[,-31])
res <- table(prediction, test$diagnosis)
cmatrix <- confusionMatrix(res,positive="M")

time2=Sys.time()

resumen_svm_lineal <- data.frame(precisión =  cmatrix$overall[1],
                                 kappa = cmatrix$overall[2],
                                 sensibilidad =  cmatrix$byClass[1],
                                 especificidad = cmatrix$byClass[2],
                                 tiempo = difftime(time2,time1, units = "mins"))
rownames(resumen_svm_lineal) <- "SVM lineal"
kable(resumen_svm_lineal)
```

	precisión	kappa	sensibilidad	especificidad	tiempo
SVM lineal	0.973545	0.9434437	0.9714286	0.9747899	0.152876 mins

```
stopCluster(cl)
```

SVM radial (RBF)

```
cl <- makePSOCKcluster(4)
registerDoParallel(cl)

time1=Sys.time()
set.seed(12345)
model_svm_RBF <- train(diagnosis ~ ., train, method='svmRadial',
                   trControl= Control,
                   tuneGrid= NULL, tuneLength=params$ntuning,trace = FALSE,
                   metric = "ROC")

prediction <- predict(model_svm_RBF, test[,-31])
res <- table(prediction, test$diagnosis)
cmatrix <- confusionMatrix(res,positive="M")

time2=Sys.time()

resumen_svm_RBF <- data.frame(precisión =  cmatrix$overall[1],
                              kappa = cmatrix$overall[2],
                              sensibilidad =  cmatrix$byClass[1],
                              especificidad = cmatrix$byClass[2],
                              tiempo = difftime(time2,time1, units = "mins"))
rownames(resumen_svm_RBF) <- "SVM RBF"
kable(resumen_svm_RBF)
```

	precisión	kappa	sensibilidad	especificidad	tiempo
SVM RBF	0.978836	0.9548872	0.9857143	0.9747899	0.3535486 mins

```
stopCluster(cl)
```

Árbol de decisión

```
cl <- makePSOCKcluster(4)
registerDoParallel(cl)

time1=Sys.time()
set.seed(12345)
model_C5.0 <- train(diagnosis ~ ., train, method='C5.0',
                    trControl= Control,
                    tuneGrid= NULL, tuneLength=params$ntuning,trace = FALSE,
                    metric = "ROC")

prediction <- predict(model_C5.0, test[,-31])
res <- table(prediction, test$diagnosis)
cmatrix <- confusionMatrix(res,positive="M")

time2=Sys.time()
time2-time1
```

```
## Time difference of 1.267871 mins
```

```
resumen_c5 <- data.frame(precisión =  cmatrix$overall[1],
                                kappa = cmatrix$overall[2],
                                sensibilidad =  cmatrix$byClass[1],
                                especificidad = cmatrix$byClass[2],
                                tiempo = difftime(time2,time1, units = "mins"))
rownames(resumen_c5) <- "Decisión Trees"
kable(resumen_c5)
```

	precisión	kappa	sensibilidad	especificidad	tiempo
Decisión Trees	0.973545	0.9437734	0.9857143	0.9663866	1.267871 mins

```
stopCluster(cl)
```

Random Forest

```
cl <- makePSOCKcluster(4)
registerDoParallel(cl)

time1=Sys.time()
set.seed(12345)
model_RF <- train(diagnosis ~ ., train, method='rf',
                  trControl= Control,
                  tuneGrid= NULL, tuneLength=params$ntuning,trace = FALSE,
                  metric = "ROC")

prediction <- predict(model_RF, test[,-31])
res <- table(prediction, test$diagnosis)
cmatrix <- confusionMatrix(res,positive="M")

time2=Sys.time()

resumen_rf <- data.frame(precisión =  cmatrix$overall[1],
                                kappa = cmatrix$overall[2],
                                sensibilidad =  cmatrix$byClass[1],
                                especificidad = cmatrix$byClass[2],
                                tiempo = difftime(time2,time1, units = "mins"))
rownames(resumen_rf) <- "Random Forest"
kable(resumen_rf)
```

	precisión	kappa	sensibilidad	especificidad	tiempo
Random Forest	0.989418	0.9773109	0.9857143	0.9915966	1.268046 mins

```
beep(3)
stopCluster(cl)
```

Conclusión

```
resumen_algoritmos <- rbind.data.frame(resumen_knn, resumen_naive_bayes, resumen_nnet,
                                       resumen_svm_lineal,  resumen_svm_RBF, resumen_c5, resumen_rf)
kable(resumen_algoritmos)
```

	precisión	kappa	sensibilidad	especificidad	tiempo
k-NN	0.9629630	0.9208211	0.9571429	0.9663866	0.2275298 mins
Naive Bayes	0.9153439	0.8205980	0.9142857	0.9159664	0.1332478 mins
Neural network	0.9470899	0.8872180	0.9428571	0.9495798	8.3865799 mins
SVM lineal	0.9735450	0.9434437	0.9714286	0.9747899	0.1528760 mins
SVM RBF	0.9788360	0.9548872	0.9857143	0.9747899	0.3535486 mins
Decisión Trees	0.9735450	0.9437734	0.9857143	0.9663866	1.2678712 mins
Random Forest	0.9894180	0.9773109	0.9857143	0.9915966	1.2680456 mins

```
write.csv(resumen_algoritmos, "resultados/precisión entrenando según ROC.csv")
```

El mejor algoritmo para clasificar los tumores es el método random forest, es un algoritmo difícil de interpretar, pero eficaz. El algoritmo de toma de decisiones es un poco menos preciso, pero mantiene la sensibilidad, por lo que podría usarse para hacer una guía que permitiera a los médicos que analizan las células al microscopio hacerse una idea de la gravedad del tumor; personalmente no lo recomiendo, porque la toma de datos implica hacer una biopsia del tejido mamario, por tanto, ya se sospecha que puede ser maligno, la guía podría ser útil en otras circunstancias, pero no en esta. Además, una vez tomadas las muestras el random forest podría clasificarlas en menos de dos minutos, seguramente menos de lo que tardaría el médico en seguir la clave dicotómica, y más preciso.

Por todo esto, y a modo de conclusión, para esta tarea se recomiendo el algoritmo random forest, porque, aunque no podamos seguir fácilmente el proceso por el que toma las decisiones, es el que menores fallos comete prediciendo datos con los que no ha entrenado.