Práctica Obligatoria Aprendizaje Automático

Carlos Sanabria Miranda

23/12/2018

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
geneLevel <- read.csv('data.csv',stringsAsFactors = F)
label <- read.csv('labels.csv',stringsAsFactors = F)

tcga <- merge(geneLevel, label, sort = F)
tcga$X <- NULL
```

```
# Se van a usar 100 variables, elegidas en función del UO
set.seed(250707)
tcga.filtered <- tcga[,c(sample(ncol(tcga)-1,100),ncol(tcga))]
tcga.filtered$Class <- as.factor(tcga.filtered$Class)
```

```
head(tcga.filtered)
```

```
##   gene_13715 gene_5803 gene_5816 gene_16023 gene_4190 gene_17644 gene_9893
## 1  11.657801  5.543620  8.011943  0.5918709  11.58106  1.3342822  1.598651
## 2   3.365469  6.684650  6.335094  2.3994172  10.11925  1.4650341  3.095199
## 3  10.586830  0.000000  4.802193  0.0000000  10.32695  0.0000000  5.686041
## 4  11.004003  3.913761  7.011809  0.4348817  10.37460  0.4348817  3.239245
## 5   9.473935  3.294621  5.256999  0.8897072  10.50573  0.0000000  4.620270
## 6  11.498894  6.658826  4.570250  3.3243354  11.23210  3.8389921  3.052990
##   gene_9348 gene_18208 gene_12622 gene_10918 gene_11539 gene_9352 gene_586
## 1 0.5918709   9.071672          0   12.02997   13.03626         0 2.717803
## 2 0.0000000   9.365211          0   13.72926   12.75733         0 1.004394
## 3 2.1098286   9.941100          0   12.33354   12.34483         0 8.215625
## 4 0.0000000  10.797945          0   11.66065   12.59113         0 9.165957
## 5 0.8897072   9.583299          0   12.50329   13.21573         0 4.586531
## 6 0.0000000   9.558637          0   11.39507   14.19647         0 4.114034
##   gene_11046 gene_15982 gene_6641 gene_14575 gene_6572 gene_10991
## 1   9.168943   5.734360  8.884644   10.79939  4.801122   11.39743
## 2   8.756947   9.653970  9.882593   12.63465  8.044913   12.03279
## 3   9.815162   8.475166  7.916632   13.29324  3.352236   11.75734
## 4  10.559492   7.340918  8.990481   12.50391  6.324679   11.38549
## 5   9.624408   7.512938  8.353416   12.73438 10.922280   11.90960
## 6   9.666236   6.040776  9.405386   12.14296  7.732534   11.81101
##   gene_8892 gene_10520 gene_13681 gene_2105 gene_9454 gene_7203 gene_16948
## 1  8.542045   3.105561  1.3342822 10.456673 0.5918709  1.822037  10.337901
## 2  9.424502   2.530820  0.3236583  5.539965 0.0000000  3.763008   8.534272
## 3  8.607807   4.947105  0.7965978  8.795384 0.0000000  3.186960   9.512973
## 4  9.209470   6.362181  1.0394192  9.769982 0.0000000  4.847566   9.306442
## 5  9.958287   4.174438  1.8312692  8.088502 0.0000000  3.116930   8.579293
## 6  8.921540   5.053229  1.1941501  9.207947 0.0000000  4.543941   9.264436
##   gene_15445 gene_11874 gene_8608 gene_13217 gene_10973 gene_18975
## 1  0.0000000   8.353592 5.4123344   3.478079  0.0000000   6.107665
```

```
## 2  0.8111422   9.996742 2.6512239   1.590818  1.1747899   2.961846
## 3  0.0000000   9.095682 0.4525954   2.438799  0.0000000   0.000000
## 4  0.0000000   9.067230 2.1756523   2.058697  1.4640932   2.478532
## 5  0.3609822   8.526887 0.3609822   5.456977  2.1412039   3.927290
## 6  0.0000000   9.297299 3.4988763   1.194150  0.5154097   5.471392
##   gene_17001 gene_11002 gene_8126 gene_19646 gene_12975 gene_4428
## 1   1.010279  11.523077 0.5918709   11.40717   7.865900  6.635004
## 2   1.004394  10.038055 4.1580300   10.03840   9.186800  9.678975
## 3   0.000000   9.927517 3.9421673   10.71666   8.164661  5.324101
## 4   0.000000  10.938278 2.8033097   10.47821   8.491156  5.421661
## 5   3.414853   8.762319 4.0561500   10.44848   8.190294  7.318534
## 6   4.521189   9.925795 2.9763453   10.66344   8.225983  5.161037
##   gene_3899 gene_15876 gene_11713 gene_16771 gene_3748 gene_2178
## 1 10.388997   9.549904   8.229886          0         0 6.6641557
## 2  9.319372   8.598402   6.063924          0         0 3.2932822
## 3  9.456584   9.080050   7.101640          0         0 0.7965978
## 4 10.204388   8.902010   5.681056          0         0 3.2392453
## 5  9.264511   8.333209   7.602179          0         0 5.0620828
## 6 10.128097   9.772753   8.140763          0         0 5.6173367
##   gene_18755 gene_1361 gene_15447 gene_5824 gene_12997 gene_8090 gene_681
## 1   7.850137  9.070312  0.5918709  0.000000   8.441940  4.877779 8.490085
## 2   8.384874  9.796257  1.7065085  0.000000   7.866716  9.072827 9.008756
## 3   9.526012  9.929699  0.4525954  0.000000   7.279833  1.683023 8.674383
## 4   9.310445  9.984062  1.2673560  1.267356   8.628292  2.058697 8.006203
## 5   9.472780 10.016460  3.6288786  0.000000   8.733029  7.583060 9.072200
## 6   8.756610  9.842558  0.0000000  0.000000   7.582255  6.890532 8.870836
##   gene_17105 gene_6068 gene_17140 gene_10842 gene_8605 gene_11108
## 1   4.926711  5.559223   8.023239   8.440400  8.477823   10.72914
## 2   7.892907  5.927491   6.178826   8.505601  9.881438   11.04683
## 3   7.871948  4.195285   5.505316  10.183003 10.716965   10.82084
## 4   6.142708  5.926101   7.390160   9.071953  9.846398   10.52925
## 5   6.275553  5.989593   7.216115   8.104981  9.846910   10.62583
## 6   5.666950  6.345041   7.403549   8.449755  9.285435   10.89511
##   gene_17534 gene_19205 gene_858 gene_16373 gene_10097 gene_5866
## 1          0   9.869102 6.699788   8.750533          0  10.31338
## 2          0   8.306809 7.251889   6.638879          0  10.29272
## 3          0  10.222252 5.030800   8.205754          0  11.13920
## 4          0   9.691749 5.445151   8.093185          0  11.35849
## 5          0   8.219483 4.304657   7.522228          0  10.50403
## 6          0   9.503379 4.344942   7.938545          0   9.84458
##   gene_16335 gene_14355 gene_8504 gene_1042 gene_5404 gene_14877
## 1  10.051155   11.48810  7.986206 10.229324 3.3403914   6.755021
## 2   8.622539   10.65284  5.246948  9.916781 1.1747899   6.860441
## 3   9.315195   11.52851  7.592345 10.361714 1.5071603   4.723892
## 4   9.551349   12.25546  8.657497 10.445470 0.4348817   6.178185
## 5   9.006736   12.51504  8.688694  9.712589 2.3961869   7.244744
## 6   9.493959   11.37772  7.884439  9.606305 1.4422800   6.895739
##   gene_10800 gene_13648 gene_20028 gene_14609 gene_10417 gene_1734
## 1   8.508286   13.42823   7.123646   6.788503   6.277624         0
## 2   7.305633   12.99883   7.415987   8.408559   7.323865         0
## 3   3.186960   13.16007   6.237176  10.675640   4.195285         0
## 4   5.804239   12.48677   7.312466  11.154527   4.982113         0
## 5   7.824723   12.70851   6.856064   9.998534   5.409520         0
## 6   8.801725   12.91064   6.627842   7.559446   6.664938         0
```

```
##   gene_8847 gene_8094 gene_368 gene_20289 gene_12842 gene_14896 gene_5622
## 1  1.598651  9.064856 4.747656   7.282468  11.269606  10.160892  8.234750
## 2  0.000000  7.357446 8.098853   9.090906   7.803150   8.691747  9.929489
## 3  0.000000  8.222761 4.683084   2.622673  10.594409   9.487434  6.112102
## 4  5.310954  7.257133 5.681056   7.929170  10.346037  10.676177  7.858764
## 5  1.831269  7.013540 5.774194   9.028105   9.932079  10.825166  9.473359
## 6  0.000000  8.758040 6.991340   9.046289  10.902903   9.414926 10.560963
##   gene_6373 gene_9824 gene_13171 gene_6646 gene_12913 gene_10565
## 1  5.939029  2.476226  0.0000000  4.926711   9.537158   6.454895
## 2  8.332927  1.465034  0.0000000  8.356368   5.476181  11.484632
## 3  4.435342  0.000000  0.0000000  4.964237   7.889985   7.865121
## 4  4.737725  5.934425  0.0000000  6.205950   8.936603   9.658222
## 5 11.509458  5.607561  0.6493855  4.822230   6.971670   8.402722
## 6  6.753725  1.838427  0.0000000  5.443192   7.706082  10.522738
##   gene_16470 gene_13840 gene_10724 gene_10133 gene_17270 gene_2599
## 1  10.403108   8.321933   4.063658   3.478079  10.459800  9.598847
## 2   9.865967   9.350833   5.623305   4.365895   7.047626  7.926106
## 3  10.428810  10.544385   5.510573   3.186960   9.274572  7.997908
## 4  10.249279   8.415881   6.030725   3.947049   9.689905  8.623154
## 5  11.046728   8.245391   6.342784   4.892950  10.332719  7.877996
## 6  10.566282   8.750868   4.936704   3.443010  10.267793  8.777933
##   gene_4009 gene_3648 gene_14235 gene_12566 gene_14075 Class
## 1 0.0000000         0          0  0.0000000    9.97685  PRAD
## 2 2.7078561         0          0  0.0000000   10.43582  LUAD
## 3 0.0000000         0          0  0.0000000   11.45410  PRAD
## 4 0.4348817         0          0  0.4348817   10.29886  PRAD
## 5 2.1412039         0          0  0.0000000   10.25464  BRCA
## 6 1.1941501         0          0  0.5154097   10.39515  PRAD
```

Esquema de evaluación

Para evaluar los modelos se hará primeramente una división en entrenamiento/test, de 80%/20% respectivamente. Esto nos permitirá realizar una validación externa, para ver cómo de bueno es realmente el modelo con datos que no ha visto durante el proceso de entrenamiento.

```
set.seed(107)
inTrain <- createDataPartition(
  y = tcga.filtered$Class,
  p = .80, # 80% instancias para entrenamiento
  list = FALSE
)
training <- tcga.filtered[ inTrain,]
testing  <- tcga.filtered[-inTrain,]
# Revisamos que el número de instancias totales es el mismo
nrow(training)+nrow(testing)
```

```
## [1] 801
```

```
# Eliminamos las columnas con todo 0, para que no haya problemas con ciertos algoritmos
training <- training[,apply(tcga.filtered,2,function(x){all(x!=0)})]
testing <- testing[,apply(tcga.filtered,2,function(x){all(x!=0)})]
```

Para la validación interna he seleccionado Bootstrap con ponderación de instancias y 10 repeticiones, ya que considero que el conjunto de datos del que se dispone es pequeño (801 instancias). Bootstrap es muy adecuado

para conjuntos de datos pequeños, y ponderando las instancias se reduce el sesgo. Con 10 repeticiones evitamos que los resultados dependan enteramente de una selección de los datos determinada.

```
ctrl <- trainControl(
  method = "boot632", # bootstrap con ponderación de instancias
  number = 10 # 10 repeticiones
)
```

Métrica utilizada para comparar los modelos

```
cat('Número de datos de cada clase: \n')
```

```
## Número de datos de cada clase:
```

```
summary(tcga.filtered$Class)
```

```
## BRCA COAD KIRC LUAD PRAD
##  300   78  146  141  136
```

```
cat('\nPorcentaje de datos de cada clase: \n')
```

```
##
## Porcentaje de datos de cada clase:
```

```
summary(tcga.filtered$Class)/sum(summary(tcga.filtered$Class))
```

```
##       BRCA       COAD       KIRC       LUAD       PRAD
## 0.37453184 0.09737828 0.18227216 0.17602996 0.16978777
```

Si echamos un vistazo al número de datos que hay por cada clase, podemos ver como están ligeramente desbalanceados. La clase BCRA tiene 300 datos, mientras que la clase COAD 78. Por tanto, para comparar los modelos voy a utilizar la métrica Kappa, que me permite saber cómo de bueno es el clasificador en comparación con una predicción basada en la frecuencia de las clases. Además, dicha métrica tiene en cuenta también el valor del porcentaje de acierto.

Generación de los modelos

```
# Creamos una función para ver los resultados de la validación interna
internalValidation <- function(model){
  print(model)
  print(model$results[rownames(model$bestTune),1:4])
}
```

```
# Creamos una función para ver los resultados de la validación externa
externalValidation <- function(model, testingData = testing){
  print(confusionMatrix(predict(model,testingData),testingData$Class))
}
```

Árboles de decisión (DT)

C4.5

Vamos a probar primero con un árbol de decisión que utiliza el algoritmo C4.5 (usa la métrica Gain Ratio para generar el árbol).

```
set.seed(725)
c4.5Fit <- train(
  Class ~ .,
  data = training,
  method = "J48",
  tuneLength = 7,
  trControl = ctrl,
  metric = "Kappa"
)
```

```
# Resultados validación interna
internalValidation(c4.5Fit)
```

```
## C4.5-like Trees
##
## 642 samples
##  99 predictor
##   5 classes: 'BRCA', 'COAD', 'KIRC', 'LUAD', 'PRAD'
##
## No pre-processing
## Resampling: Bootstrapped (10 reps)
## Summary of sample sizes: 642, 642, 642, 642, 642, 642, ...
## Resampling results across tuning parameters:
##
##   C           M  Accuracy   Kappa
##   0.01000000  1  0.8795382  0.8405941
##   0.01000000  2  0.8785911  0.8393396
##   0.01000000  3  0.8712867  0.8297773
##   0.01000000  4  0.8709141  0.8295222
##   0.01000000  5  0.8690896  0.8267902
##   0.01000000  6  0.8582874  0.8122912
##   0.01000000  7  0.8518056  0.8029806
##   0.09166667  1  0.8845146  0.8474159
##   0.09166667  2  0.8772631  0.8377382
##   0.09166667  3  0.8710177  0.8295158
##   0.09166667  4  0.8717558  0.8307949
##   0.09166667  5  0.8688207  0.8264218
##   0.09166667  6  0.8620295  0.8173734
##   0.09166667  7  0.8525958  0.8041276
##   0.17333333  1  0.8853962  0.8486552
##   0.17333333  2  0.8769964  0.8374099
##   0.17333333  3  0.8728895  0.8319993
##   0.17333333  4  0.8720780  0.8312217
##   0.17333333  5  0.8693937  0.8271734
##   0.17333333  6  0.8622997  0.8178151
##   0.17333333  7  0.8545850  0.8070217
##   0.25500000  1  0.8859330  0.8494376
##   0.25500000  2  0.8772794  0.8378530
```

```
##   0.25500000  3  0.8728895  0.8319993
##   0.25500000  4  0.8712676  0.8302178
##   0.25500000  5  0.8688451  0.8265534
##   0.25500000  6  0.8622997  0.8178151
##   0.25500000  7  0.8554048  0.8083442
##   0.33666667  1  0.8859330  0.8494376
##   0.33666667  2  0.8770104  0.8375415
##   0.33666667  3  0.8728895  0.8319993
##   0.33666667  4  0.8712676  0.8302178
##   0.33666667  5  0.8688451  0.8265534
##   0.33666667  6  0.8633532  0.8193671
##   0.33666667  7  0.8554048  0.8084041
##   0.41833333  1  0.8856641  0.8491020
##   0.41833333  2  0.8770104  0.8375415
##   0.41833333  3  0.8728895  0.8319993
##   0.41833333  4  0.8712676  0.8302178
##   0.41833333  5  0.8698986  0.8280532
##   0.41833333  6  0.8633532  0.8193671
##   0.41833333  7  0.8554048  0.8084041
##   0.50000000  1  0.8854102  0.8487839
##   0.50000000  2  0.8770104  0.8375415
##   0.50000000  3  0.8728895  0.8319993
##   0.50000000  4  0.8712676  0.8302178
##   0.50000000  5  0.8698986  0.8280532
##   0.50000000  6  0.8633532  0.8193671
##   0.50000000  7  0.8554048  0.8084041
##
## Kappa was used to select the optimal model using the largest value.
## The final values used for the model were C = 0.255 and M = 1.
##        C M Accuracy     Kappa
## 22 0.255 1 0.885933 0.8494376
```

El máximo valor de C es 0.5, y a mayor valor de M, peores resultados. Por tanto, los valores C = 0.255 and M = 1 son de los mejores posibles.

```
# Resultados validación externa
externalValidation(c4.5Fit)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction BRCA COAD KIRC LUAD PRAD
##       BRCA   47    1    3    2    1
##       COAD    3   12    0    1    0
##       KIRC    2    0   26    0    0
##       LUAD    8    2    0   25    4
##       PRAD    0    0    0    0   22
##
## Overall Statistics
##
##                Accuracy : 0.8302
##                  95% CI : (0.7626, 0.885)
##     No Information Rate : 0.3774
##     P-Value [Acc > NIR] : < 2.2e-16
##
```

```
##                   Kappa : 0.7776
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: BRCA Class: COAD Class: KIRC Class: LUAD
## Sensitivity               0.7833     0.80000      0.8966      0.8929
## Specificity               0.9293     0.97222      0.9846      0.8931
## Pos Pred Value            0.8704     0.75000      0.9286      0.6410
## Neg Pred Value            0.8762     0.97902      0.9771      0.9750
## Prevalence                0.3774     0.09434      0.1824      0.1761
## Detection Rate            0.2956     0.07547      0.1635      0.1572
## Detection Prevalence      0.3396     0.10063      0.1761      0.2453
## Balanced Accuracy         0.8563     0.88611      0.9406      0.8930
##                      Class: PRAD
## Sensitivity               0.8148
## Specificity               1.0000
## Pos Pred Value            1.0000
## Neg Pred Value            0.9635
## Prevalence                0.1698
## Detection Rate            0.1384
## Detection Prevalence      0.1384
## Balanced Accuracy         0.9074
```

Resultados: Validación interna=[Accuracy=0.8859330 y Kappa=0.8494376]. Validación externa=[Accuracy=0.8302 y Kappa=0.7776].

CART

Vamos a probar ahora con un árbol de decisión que utiliza el algoritmo CART (usa la métrica Gini para generar el árbol).

Rpart

Primero probaremos con la versión rpart, que utiliza el parámetro cp (solo se introducirá un nodo en el árbol cuando permita que se reduzca el error en al menos el valor de cp).

```r
set.seed(725)
rpartFit <- train(
  Class ~ .,
  data = training,
  method = "rpart",
  tuneLength = 10,
  trControl = ctrl,
  metric = "Kappa"
)
```

```r
# Resultados validación interna
internalValidation(rpartFit)
```

```
## CART
##
## 642 samples
##  99 predictor
##   5 classes: 'BRCA', 'COAD', 'KIRC', 'LUAD', 'PRAD'
```

```
##
## No pre-processing
## Resampling: Bootstrapped (10 reps)
## Summary of sample sizes: 642, 642, 642, 642, 642, 642, ...
## Resampling results across tuning parameters:
##
##   cp           Accuracy   Kappa
##   0.000000000  0.8566588  0.8107228
##   0.004975124  0.8551795  0.8088394
##   0.012437811  0.8489619  0.8005896
##   0.014925373  0.8450065  0.7953938
##   0.019900498  0.8326427  0.7792314
##   0.022388060  0.8248036  0.7683383
##   0.024875622  0.8176883  0.7585090
##   0.081260365  0.7192557  0.6130929
##   0.241293532  0.5667617  0.3503639
##   0.261194030  0.4287630  0.1032443
##
## Kappa was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.
##   cp  Accuracy     Kappa AccuracySD
## 1  0 0.8566588 0.8107228 0.03642714
```

A medida que se aumenta el cp, los resultados empeoran. El mejor cp es 0.

```
# Resultados validación externa
externalValidation(rpartFit)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction BRCA COAD KIRC LUAD PRAD
##       BRCA   52    0    1    1    1
##       COAD    1   11    0    0    0
##       KIRC    1    0   27    1    0
##       LUAD    5    4    1   23    6
##       PRAD    1    0    0    3   20
##
## Overall Statistics
##
##                Accuracy : 0.8365
##                  95% CI : (0.7697, 0.8903)
##     No Information Rate : 0.3774
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7849
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: BRCA Class: COAD Class: KIRC Class: LUAD
## Sensitivity               0.8667     0.73333      0.9310      0.8214
## Specificity               0.9697     0.99306      0.9846      0.8779
## Pos Pred Value            0.9455     0.91667      0.9310      0.5897
## Neg Pred Value            0.9231     0.97279      0.9846      0.9583
```

```
## Prevalence                0.3774     0.09434      0.1824      0.1761
## Detection Rate            0.3270     0.06918      0.1698      0.1447
## Detection Prevalence      0.3459     0.07547      0.1824      0.2453
## Balanced Accuracy         0.9182     0.86319      0.9578      0.8496
##                       Class: PRAD
## Sensitivity               0.7407
## Specificity               0.9697
## Pos Pred Value            0.8333
## Neg Pred Value            0.9481
## Prevalence                0.1698
## Detection Rate            0.1258
## Detection Prevalence      0.1509
## Balanced Accuracy         0.8552
```

Resultados: Validación interna=[Accuracy=0.8566588 y Kappa=0.8107228]. Validación externa=[Accuracy=0.8365 y Kappa=0.7849].
Los resultados en validación interna son peores que los del árbol generado con el algoritmo C4.5, pero en validación externa son mejores.

Rpart2

Ahora probamos con la versión rpart2, que utiliza el parámetro maxdepth (controla la profundidad máxima del árbol).

```
set.seed(725)
rpart2Fit <- train(
  Class ~ .,
  data = training,
  method = "rpart2",
  tuneLength = 10,
  trControl = ctrl,
  metric = "Kappa"
)
```

```
## note: only 9 possible values of the max tree depth from the initial fit.
##  Truncating the grid to 9 .
```

```
# Resultados validación interna
internalValidation(rpart2Fit)
```

```
## CART
##
## 642 samples
##  99 predictor
##   5 classes: 'BRCA', 'COAD', 'KIRC', 'LUAD', 'PRAD'
##
## No pre-processing
## Resampling: Bootstrapped (10 reps)
## Summary of sample sizes: 642, 642, 642, 642, 642, 642, ...
## Resampling results across tuning parameters:
##
##   maxdepth  Accuracy   Kappa
##    1        0.5683361  0.3546728
##    2        0.6847854  0.5542354
##    5        0.7952405  0.7271738
##    6        0.8134707  0.7532282
```

```
##    7        0.8258818  0.7701368
##    9        0.8442343  0.7946148
##   10        0.8506128  0.8030511
##   12        0.8555245  0.8091635
##   14        0.8555245  0.8091635
##
## Kappa was used to select the optimal model using the largest value.
## The final value used for the model was maxdepth = 12.
##   maxdepth  Accuracy     Kappa AccuracySD
## 8       12 0.8555245 0.8091635 0.03395524
```

A medida que se aumenta la profundidad, los resultados mejoran, pero la máxima profundidad permitida para este árbol es 14, que tiene los mismos resultados que el árbol con profundidad 12 (que es el valor final seleccionado para el modelo).

```
# Resultados validación externa
externalValidation(rpart2Fit)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction BRCA COAD KIRC LUAD PRAD
##       BRCA   53    0    1    4    1
##       COAD    2   12    0    2    0
##       KIRC    1    0   27    1    0
##       LUAD    3    3    1   18    6
##       PRAD    1    0    0    3   20
##
## Overall Statistics
##
##                Accuracy : 0.8176
##                  95% CI : (0.7487, 0.8743)
##     No Information Rate : 0.3774
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7591
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: BRCA Class: COAD Class: KIRC Class: LUAD
## Sensitivity               0.8833     0.80000      0.9310      0.6429
## Specificity               0.9394     0.97222      0.9846      0.9008
## Pos Pred Value            0.8983     0.75000      0.9310      0.5806
## Neg Pred Value            0.9300     0.97902      0.9846      0.9219
## Prevalence                0.3774     0.09434      0.1824      0.1761
## Detection Rate            0.3333     0.07547      0.1698      0.1132
## Detection Prevalence      0.3711     0.10063      0.1824      0.1950
## Balanced Accuracy         0.9114     0.88611      0.9578      0.7718
##                      Class: PRAD
## Sensitivity               0.7407
## Specificity               0.9697
## Pos Pred Value            0.8333
## Neg Pred Value            0.9481
## Prevalence                0.1698
```

```
## Detection Rate            0.1258
## Detection Prevalence      0.1509
## Balanced Accuracy         0.8552
```

Resultados: Validación interna=[Accuracy=0.8555245 y Kappa=0.8091635]. Validación externa=[Accuracy=0.8176 y Kappa=0.7591].
Los resultados, tanto en validación interna como externa, son peores que los de los otros 2 árboles.

Conclusiones árboles de decisión

Se puede ver como el mejor árbol de decisión en la validación externa es el que utilizar el algoritmo CART con la versión rpart. Este será por tanto el modelo seleccionado de árboles de decisión.

Vecinos más cercanos (kNN)

```
set.seed(725)
knnFit <- train(
  Class ~ .,
  data = training,
  method = "knn",
  trControl = ctrl,
  tuneLength = 8, # valor de k
  metric = "Kappa"
)
```

```
# Resultados validación interna
internalValidation(knnFit)
```

```
## k-Nearest Neighbors
##
## 642 samples
##  99 predictor
##   5 classes: 'BRCA', 'COAD', 'KIRC', 'LUAD', 'PRAD'
##
## No pre-processing
## Resampling: Bootstrapped (10 reps)
## Summary of sample sizes: 642, 642, 642, 642, 642, 642, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.9770035  0.9696066
##    7  0.9779876  0.9708902
##    9  0.9752496  0.9672700
##   11  0.9780502  0.9709278
##   13  0.9804938  0.9741859
##   15  0.9818124  0.9759346
##   17  0.9788371  0.9720047
##   19  0.9770715  0.9696570
##
## Kappa was used to select the optimal model using the largest value.
## The final value used for the model was k = 15.
##    k  Accuracy     Kappa  AccuracySD
## 6 15 0.9818124 0.9759346 0.006912416
```

Los resultados van en aumento a medida que se aumenta k, salvo a partir de k=15, que van en descenso. k=15 es el mejor valor para dicho parámetro.

```
# Resultados validación externa
externalValidation(knnFit)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction BRCA COAD KIRC LUAD PRAD
##       BRCA   57    0    2    0    0
##       COAD    0   15    0    1    0
##       KIRC    0    0   27    0    0
##       LUAD    3    0    0   27    1
##       PRAD    0    0    0    0   26
##
## Overall Statistics
##
##                Accuracy : 0.956
##                  95% CI : (0.9114, 0.9821)
##     No Information Rate : 0.3774
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9419
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: BRCA Class: COAD Class: KIRC Class: LUAD
## Sensitivity               0.9500     1.00000      0.9310      0.9643
## Specificity               0.9798     0.99306      1.0000      0.9695
## Pos Pred Value            0.9661     0.93750      1.0000      0.8710
## Neg Pred Value            0.9700     1.00000      0.9848      0.9922
## Prevalence                0.3774     0.09434      0.1824      0.1761
## Detection Rate            0.3585     0.09434      0.1698      0.1698
## Detection Prevalence      0.3711     0.10063      0.1698      0.1950
## Balanced Accuracy         0.9649     0.99653      0.9655      0.9669
##                      Class: PRAD
## Sensitivity               0.9630
## Specificity               1.0000
## Pos Pred Value            1.0000
## Neg Pred Value            0.9925
## Prevalence                0.1698
## Detection Rate            0.1635
## Detection Prevalence      0.1635
## Balanced Accuracy         0.9815
```

Resultados: Validación interna=[Accuracy=0.9818124 y Kappa=0.9759346]. Validación externa=[Accuracy=0.956 y Kappa=0.9419].

Redes neuronales (NN)

```
plotNNErrorEvolution <- function(nnModel){
  ggplot() + geom_line(aes(x=1:length(nnModel$finalModel$IterativeFitError),
```

```
                          y=nnModel$finalModel$IterativeFitError)) +
  xlab("Iteraciones") + ylab("Error")
}
```

1 capa oculta

Primeramente, vamos a probar con un red neuronal de una sola capa oculta.

```
set.seed(472)
nn1LFit <- train(
  Class ~ .,
  data = training,
  method = "mlp",
  trControl = ctrl,
  tuneGrid = data.frame(size=seq(9,25,4)),
  maxit = 300,
  metric = "Kappa"
)
```

```
# Resultados validación interna
internalValidation(nn1LFit)
```

```
## Multi-Layer Perceptron
##
## 642 samples
##  99 predictor
##   5 classes: 'BRCA', 'COAD', 'KIRC', 'LUAD', 'PRAD'
##
## No pre-processing
## Resampling: Bootstrapped (10 reps)
## Summary of sample sizes: 642, 642, 642, 642, 642, 642, ...
## Resampling results across tuning parameters:
##
##   size  Accuracy   Kappa
##    9    0.2967567  0.000000000
##   13    0.3695304  0.000000000
##   17    0.3791251  0.000000000
##   21    0.2676741  0.000000000
##   25    0.3283512  0.002098648
##
## Kappa was used to select the optimal model using the largest value.
## The final value used for the model was size = 25.
##   size  Accuracy       Kappa AccuracySD
## 5   25 0.3283512 0.002098648  0.1020108
```

En la validación interna los resultados son desastrosos. Aunque se aumente el número de neuronas de la capa oculta los resultados son los mismos. El valor de Kappa es 0 y el mayor Accuracy ronda el 0.37, lo que quiere decir que la red lo que hace es decir siempre que la clase es la clase mayoritaria en los datos de entrenamiento (BCRA, que es la clase del 37% de las instancias de entrenamiento).

```
# Porcentaje de datos de cada clase en los datos de entrenamiento
summary(training$Class)/sum(summary(training$Class))
```

```
##       BRCA       COAD       KIRC       LUAD       PRAD
## 0.37383178 0.09813084 0.18224299 0.17601246 0.16978193
```

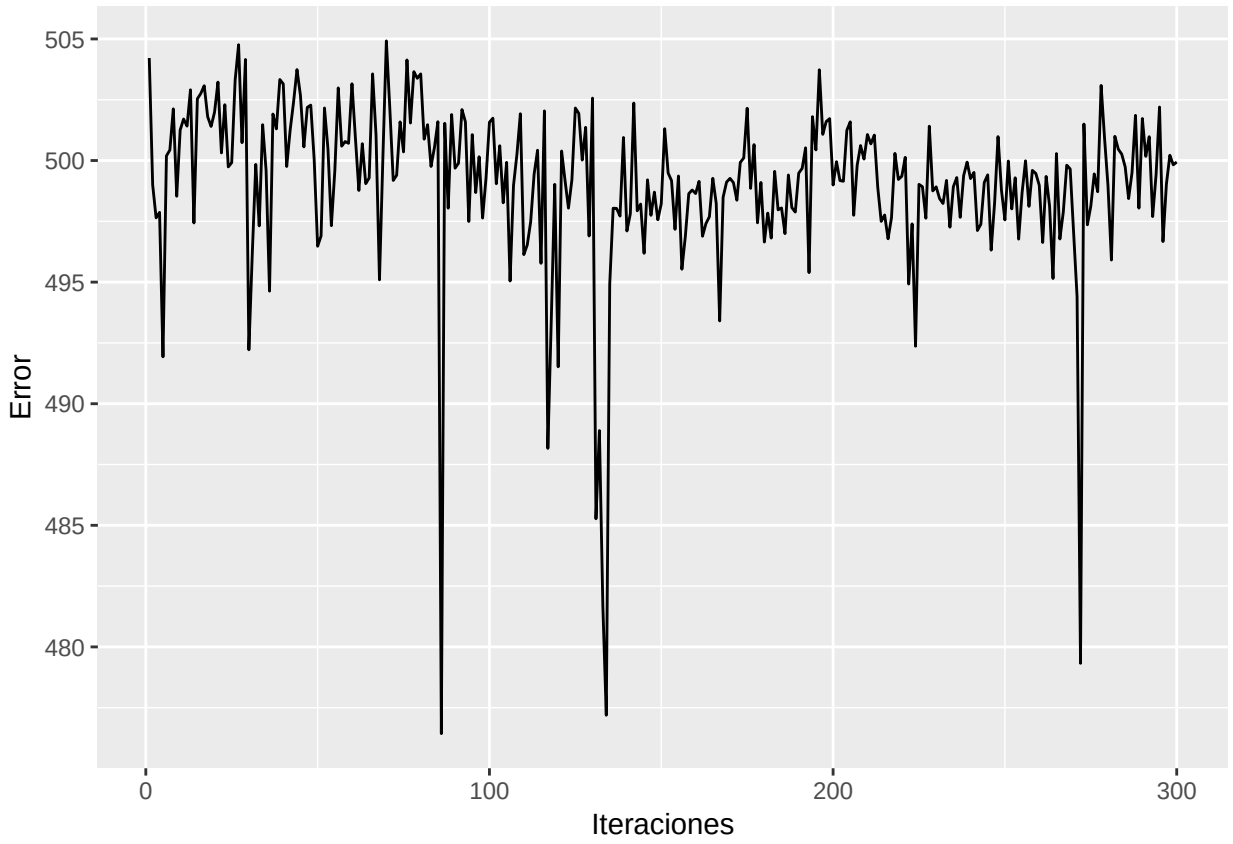
Podemos apreciar cómo siempre predice que la clase es BCRA mirando la matriz de confusión con los datos de entrenamiento.

```
confusionMatrix(predict(nn1LFit,training),training$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction BRCA COAD KIRC LUAD PRAD
##       BRCA  240   63  117  113  109
##       COAD    0    0    0    0    0
##       KIRC    0    0    0    0    0
##       LUAD    0    0    0    0    0
##       PRAD    0    0    0    0    0
##
## Overall Statistics
##
##                Accuracy : 0.3738
##                  95% CI : (0.3363, 0.4125)
##     No Information Rate : 0.3738
##     P-Value [Acc > NIR] : 0.5149
##
##                   Kappa : 0
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: BRCA Class: COAD Class: KIRC Class: LUAD
## Sensitivity               1.0000     0.00000      0.0000       0.000
## Specificity               0.0000     1.00000      1.0000       1.000
## Pos Pred Value            0.3738         NaN         NaN         NaN
## Neg Pred Value               NaN     0.90187      0.8178       0.824
## Prevalence                0.3738     0.09813      0.1822       0.176
## Detection Rate            0.3738     0.00000      0.0000       0.000
## Detection Prevalence      1.0000     0.00000      0.0000       0.000
## Balanced Accuracy         0.5000     0.50000      0.5000       0.500
##                      Class: PRAD
## Sensitivity               0.0000
## Specificity               1.0000
## Pos Pred Value               NaN
## Neg Pred Value            0.8302
## Prevalence                0.1698
## Detection Rate            0.0000
## Detection Prevalence      0.0000
## Balanced Accuracy         0.5000
```

Vamos a observar cómo se ha ido modificando el error a lo largo de los epochs/iteraciones de la red, para ver qué puede causar tan malos resultados.

```
# Dibujamos la evolución del error a lo largo de las iteraciones de la red
plotNNErrorEvolution(nn1LFit)
```



Se puede ver como hay muchos picos en el error. Esto lo podemos intentar solucionar disminuyendo el valor de la tasa de aprendizaje, para que así la red no realice variaciones muy grandes de los pesos. Vamos a probar primero con una tasa de aprendizaje de 0.03.

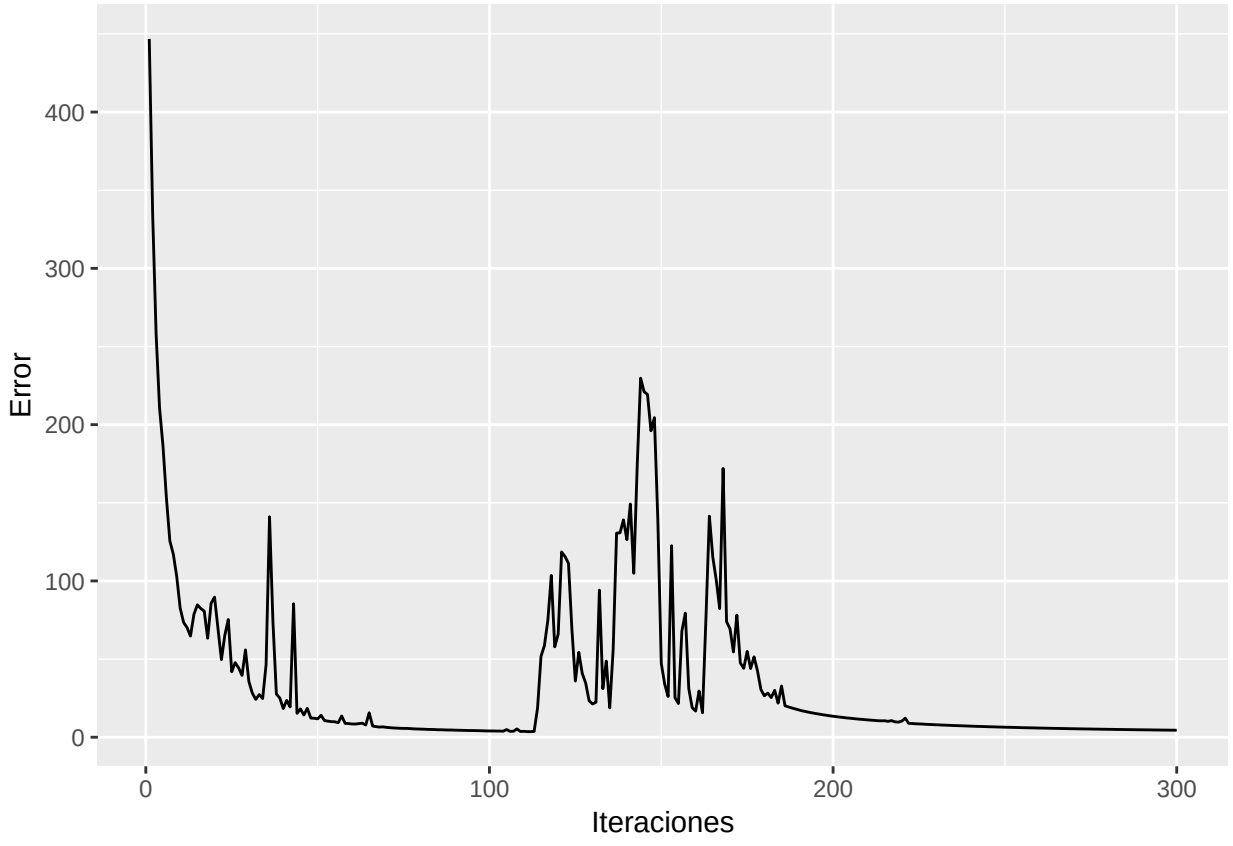
```
set.seed(472)
nn1L_lr003_Fit <- train(
  Class ~ .,
  data = training,
  method = "mlp",
  trControl = ctrl,
  tuneGrid = data.frame(size=seq(9,29,4)),
  maxit = 300,
  metric = "Kappa",
  learnFuncParams = c(0.03,0) # tasa aprendizaje = 0.03
)
```

```
# Resultados validación interna
internalValidation(nn1L_lr003_Fit)
```

```
## Multi-Layer Perceptron
##
## 642 samples
##  99 predictor
##   5 classes: 'BRCA', 'COAD', 'KIRC', 'LUAD', 'PRAD'
##
## No pre-processing
## Resampling: Bootstrapped (10 reps)
## Summary of sample sizes: 642, 642, 642, 642, 642, 642, ...
```

```
## Resampling results across tuning parameters:
##
##   size  Accuracy   Kappa
##    9    0.9515294  0.9369832
##   13    0.9822204  0.9764447
##   17    0.9916690  0.9889309
##   21    0.9929765  0.9906875
##   25    0.9946250  0.9928714
##   29    0.9941164  0.9921674
##
## Kappa was used to select the optimal model using the largest value.
## The final value used for the model was size = 25.
##   size Accuracy     Kappa  AccuracySD
## 5   25 0.994625 0.9928714 0.004529861
```

```
# Dibujamos la evolución del error a lo largo de las iteraciones de la red
plotNNErrorEvolution(nn1L_lr003_Fit)
```




```
# Resultados validación externa
externalValidation(nn1L_lr003_Fit)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction BRCA COAD KIRC LUAD PRAD
##       BRCA   60    0    0    0    0
##       COAD    0   15    0    1    0
##       KIRC    0    0   28    0    0
```

```
##       LUAD    0    0    1   27    1
##       PRAD    0    0    0    0   26
##
## Overall Statistics
##
##                Accuracy : 0.9811
##                  95% CI : (0.9459, 0.9961)
##     No Information Rate : 0.3774
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.975
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: BRCA Class: COAD Class: KIRC Class: LUAD
## Sensitivity               1.0000     1.00000      0.9655      0.9643
## Specificity               1.0000     0.99306      1.0000      0.9847
## Pos Pred Value            1.0000     0.93750      1.0000      0.9310
## Neg Pred Value            1.0000     1.00000      0.9924      0.9923
## Prevalence                0.3774     0.09434      0.1824      0.1761
## Detection Rate            0.3774     0.09434      0.1761      0.1698
## Detection Prevalence      0.3774     0.10063      0.1761      0.1824
## Balanced Accuracy         1.0000     0.99653      0.9828      0.9745
##                      Class: PRAD
## Sensitivity               0.9630
## Specificity               1.0000
## Pos Pred Value            1.0000
## Neg Pred Value            0.9925
## Prevalence                0.1698
## Detection Rate            0.1635
## Detection Prevalence      0.1635
## Balanced Accuracy         0.9815
```

Como en la red anterior el error seguía teniendo ciertos picos, aunque da unos resultados muy buenos, vamos a probar con una tasa de aprendizaje menor.

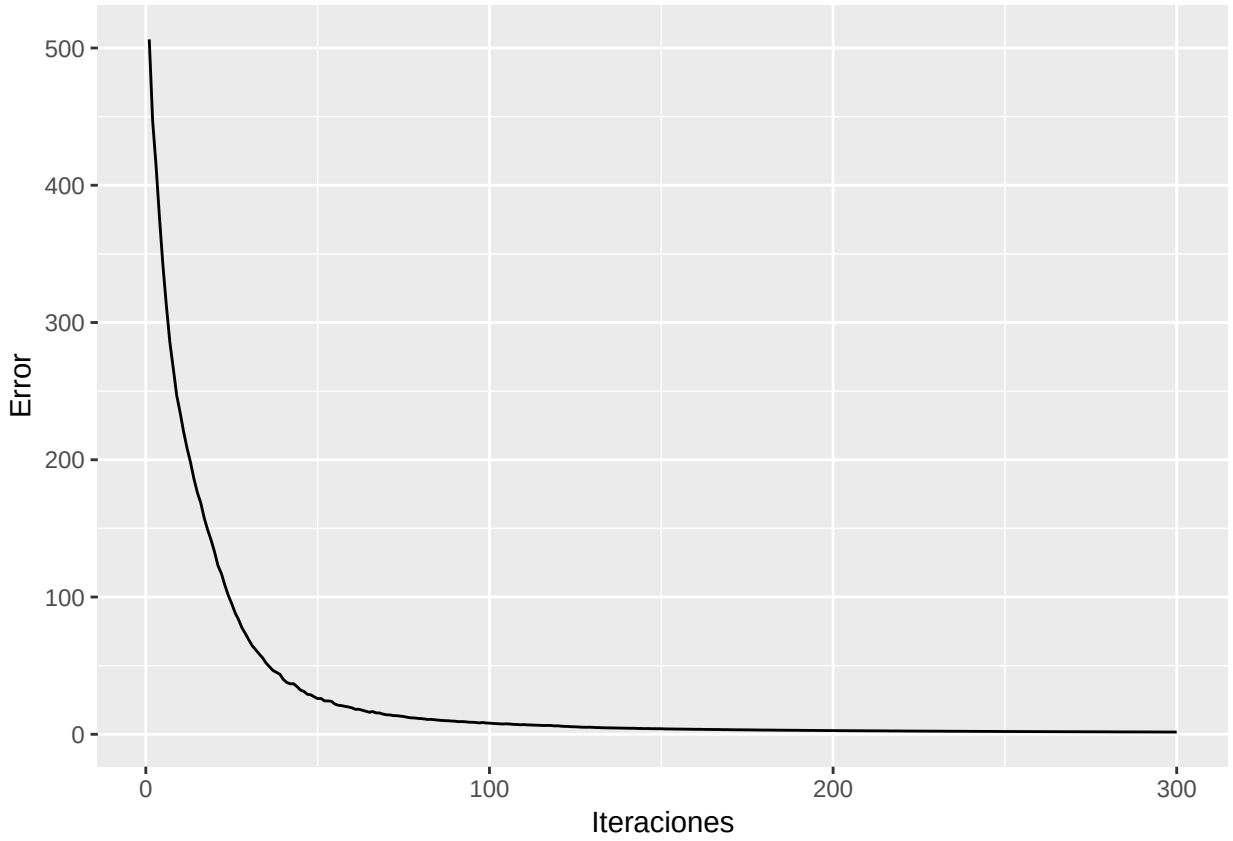
```
set.seed(472)
nn1L_lr001_Fit <- train(
  Class ~ .,
  data = training,
  method = "mlp",
  trControl = ctrl,
  tuneGrid = data.frame(size=seq(9,29,4)),
  maxit = 300,
  metric = "Kappa",
  learnFuncParams = c(0.01,0) # tasa aprendizaje = 0.01
)
```

```
# Resultados validación interna
internalValidation(nn1L_lr001_Fit)
```

```
## Multi-Layer Perceptron
##
## 642 samples
##  99 predictor
```

```
##   5 classes: 'BRCA', 'COAD', 'KIRC', 'LUAD', 'PRAD'
##
## No pre-processing
## Resampling: Bootstrapped (10 reps)
## Summary of sample sizes: 642, 642, 642, 642, 642, 642, ...
## Resampling results across tuning parameters:
##
##   size  Accuracy   Kappa
##    9    0.9629372  0.9511986
##   13    0.9905654  0.9875236
##   17    0.9938070  0.9917928
##   21    0.9919156  0.9892731
##   25    0.9946416  0.9928818
##   29    0.9948937  0.9932118
##
## Kappa was used to select the optimal model using the largest value.
## The final value used for the model was size = 29.
##   size  Accuracy     Kappa AccuracySD
## 6   29 0.9948937 0.9932118 0.00652682
```

```r
# Dibujamos la evolución del error a lo largo de las iteraciones de la red
plotNNErrorEvolution(nn1L_lr001_Fit)
```



Aunque le demos más de 300 iteraciones, el error practicamente no va a disminuir más.

```r
# Resultados validación externa
externalValidation(nn1L_lr001_Fit)
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction BRCA COAD KIRC LUAD PRAD
##       BRCA   60    0    0    0    0
##       COAD    0   15    0    1    0
##       KIRC    0    0   29    0    0
##       LUAD    0    0    0   27    0
##       PRAD    0    0    0    0   27
##
## Overall Statistics
##
##                Accuracy : 0.9937
##                  95% CI : (0.9655, 0.9998)
##     No Information Rate : 0.3774
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9917
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: BRCA Class: COAD Class: KIRC Class: LUAD
## Sensitivity               1.0000     1.00000      1.0000      0.9643
## Specificity               1.0000     0.99306      1.0000      1.0000
## Pos Pred Value            1.0000     0.93750      1.0000      1.0000
## Neg Pred Value            1.0000     1.00000      1.0000      0.9924
## Prevalence                0.3774     0.09434      0.1824      0.1761
## Detection Rate            0.3774     0.09434      0.1824      0.1698
## Detection Prevalence      0.3774     0.10063      0.1824      0.1698
## Balanced Accuracy         1.0000     0.99653      1.0000      0.9821
##                      Class: PRAD
## Sensitivity               1.0000
## Specificity               1.0000
## Pos Pred Value            1.0000
## Neg Pred Value            1.0000
## Prevalence                0.1698
## Detection Rate            0.1698
## Detection Prevalence      0.1698
## Balanced Accuracy         1.0000
```

La red con una capa oculta de 29 neuronas y tasa de aprendizaje 0.01 es la mejor de todas.
Resultados: Validación interna=[Accuracy=0.9948937 y Kappa=0.9932118]. Validación externa=[Accuracy=0.9937 y Kappa=0.9917].

No merece la pena probar con redes neuronales de más capas, dado que con una red neuronal de una capa obtenemos muy buenos resultados (dificilmente mejorables), y añadir más capas aumenta tanto la complejidad del entrenamiento como su duración.

Máquinas de vector soporte (SVM)

Vamos a probar primero con un SVM lineal, a ver que resultados nos da.

```
set.seed(627)
svmLinealFit <- train(
```

```
  Class ~ .,
  data = training,
  method = "svmLinear",
  trControl = ctrl,
  tuneGrid = data.frame(C=c(0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5)),
  metric = "Kappa"
)
```

```
# Resultados validación interna
internalValidation(svmLinealFit)
```

```
## Support Vector Machines with Linear Kernel
##
## 642 samples
##  99 predictor
##   5 classes: 'BRCA', 'COAD', 'KIRC', 'LUAD', 'PRAD'
##
## No pre-processing
## Resampling: Bootstrapped (10 reps)
## Summary of sample sizes: 642, 642, 642, 642, 642, 642, ...
## Resampling results across tuning parameters:
##
##   C      Accuracy   Kappa
##   0.001  0.9884006  0.9846238
##   0.005  0.9949460  0.9933097
##   0.010  0.9963049  0.9950986
##   0.050  0.9947480  0.9930320
##   0.100  0.9950060  0.9933676
##   0.500  0.9950060  0.9933676
##   1.000  0.9950060  0.9933676
##   5.000  0.9950060  0.9933676
##
## Kappa was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.01.
##      C  Accuracy     Kappa  AccuracySD
## 3 0.01 0.9963049 0.9950986 0.004129687
```

Los valores de C mayores o iguales a 0.1 dan los mismos resultados, ya que el valor de α de los vectores soporte nunca toma valores mayores de 0.1.

```
# Resultados validación externa
externalValidation(svmLinealFit)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction BRCA COAD KIRC LUAD PRAD
##       BRCA   60    0    0    0    0
##       COAD    0   15    0    0    0
##       KIRC    0    0   27    0    0
##       LUAD    0    0    2   28    0
##       PRAD    0    0    0    0   27
##
## Overall Statistics
##
##                Accuracy : 0.9874
```

```
##                  95% CI : (0.9553, 0.9985)
##     No Information Rate : 0.3774
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9834
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: BRCA Class: COAD Class: KIRC Class: LUAD
## Sensitivity               1.0000     1.00000      0.9310      1.0000
## Specificity               1.0000     1.00000      1.0000      0.9847
## Pos Pred Value            1.0000     1.00000      1.0000      0.9333
## Neg Pred Value            1.0000     1.00000      0.9848      1.0000
## Prevalence                0.3774     0.09434      0.1824      0.1761
## Detection Rate            0.3774     0.09434      0.1698      0.1761
## Detection Prevalence      0.3774     0.09434      0.1698      0.1887
## Balanced Accuracy         1.0000     1.00000      0.9655      0.9924
##                      Class: PRAD
## Sensitivity               1.0000
## Specificity               1.0000
## Pos Pred Value            1.0000
## Neg Pred Value            1.0000
## Prevalence                0.1698
## Detection Rate            0.1698
## Detection Prevalence      0.1698
## Balanced Accuracy         1.0000
```

Resultados: Validación interna=[Accuracy=0.9963049 y Kappa=0.9950986]. Validación externa=[Accuracy=0.9874 y Kappa=0.9834].
Son muy buenos resultados, por lo que no merece la pena probar con un SVM no lineal.

Random Forest (RF)

Este paradigma construye multitud de árboles de decisión, y a la hora de realizar una predicción devuelve la clase que es la moda (para problemas de clasificación, como el que estamos resolviendo) o la media (para problemas de regresión) de los valores devueltos por todos los árboles de decisión. Los árboles no se entrenan con todas las variables ni con todos los datos del conjunto de entrenamiento, sino que se les da un subconjunto aleatorio de las variables y de los datos. Esto genera diversidad en los árboles, que generalmente resulta en un mejor modelo.

Ventajas:

- Al utilizar muchos árboles de decisión, reduce el riesgo de sobreajuste que tienen éstos.
- Puede ser utilizado tanto para problemas de clasificación como de regresión.
- Pocos parámetros a determinar, y fáciles de entender.
- Son rápidos de entrenar.
- Pueden tratar con variables discretas y numéricas.

Desventajas:

- Los resultados no son tan interpretables como en los árboles de decisión.
- Las predicciones son lentas si hay un gran número de árboles.

El parámetro mtry representa el número de variables con las que se entrena cada árbol.

```r
set.seed(627)
rfFit <- train(
  Class ~ .,
  data = training,
  method = "rf",
  trControl = ctrl,
  tuneGrid = data.frame(mtry=seq(2,30,4)), # número de variables para cada árbol
  metric = "Kappa"
)
```

```r
# Resultados validación interna
internalValidation(rfFit)
```

```
## Random Forest
##
## 642 samples
##  99 predictor
##   5 classes: 'BRCA', 'COAD', 'KIRC', 'LUAD', 'PRAD'
##
## No pre-processing
## Resampling: Bootstrapped (10 reps)
## Summary of sample sizes: 642, 642, 642, 642, 642, 642, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9827505  0.9769982
##    6    0.9886809  0.9849578
##   10    0.9881488  0.9842496
##   14    0.9889284  0.9852872
##   18    0.9868133  0.9824857
##   22    0.9876207  0.9835485
##   26    0.9851942  0.9803143
##   30    0.9822861  0.9764542
##
## Kappa was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 14.
##   mtry  Accuracy     Kappa  AccuracySD
## 4   14 0.9889284 0.9852872 0.006208084
```

```r
rfFit$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 14
##
##         OOB estimate of  error rate: 1.71%
## Confusion matrix:
##      BRCA COAD KIRC LUAD PRAD class.error
## BRCA  239    0    0    1    0 0.004166667
## COAD    2   61    0    0    0 0.031746032
## KIRC    0    0  117    0    0 0.000000000
## LUAD    5    0    0  107    1 0.053097345
```

```
## PRAD    2    0    0    0  107 0.018348624
```

El bosque generado consta de 500 árboles de decisión, cada uno de los cuales fue entrenado con 14 variables, en lugar de las 100 que tenemos.

```
# Resultados validación externa
externalValidation(rfFit)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction BRCA COAD KIRC LUAD PRAD
##       BRCA   60    0    1    0    1
##       COAD    0   15    0    0    0
##       KIRC    0    0   27    0    0
##       LUAD    0    0    1   28    1
##       PRAD    0    0    0    0   25
##
## Overall Statistics
##
##                Accuracy : 0.9748
##                  95% CI : (0.9368, 0.9931)
##     No Information Rate : 0.3774
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9666
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: BRCA Class: COAD Class: KIRC Class: LUAD
## Sensitivity               1.0000     1.00000      0.9310      1.0000
## Specificity               0.9798     1.00000      1.0000      0.9847
## Pos Pred Value            0.9677     1.00000      1.0000      0.9333
## Neg Pred Value            1.0000     1.00000      0.9848      1.0000
## Prevalence                0.3774     0.09434      0.1824      0.1761
## Detection Rate            0.3774     0.09434      0.1698      0.1761
## Detection Prevalence      0.3899     0.09434      0.1698      0.1887
## Balanced Accuracy         0.9899     1.00000      0.9655      0.9924
##                      Class: PRAD
## Sensitivity               0.9259
## Specificity               1.0000
## Pos Pred Value            1.0000
## Neg Pred Value            0.9851
## Prevalence                0.1698
## Detection Rate            0.1572
## Detection Prevalence      0.1572
## Balanced Accuracy         0.9630
```

Resultados: Validación interna=[Accuracy=0.9889284 y Kappa=0.9852872]. Validación externa=[Accuracy=0.9748 y Kappa=0.9666].

Comparación de los modelos

Para comparar los modelos vamos a utilizar la métrica Kappa obtenida en la validación externa.

Modelos ordenados por el valor de Kappa:

Modelo	Kappa
NN (1 capa)	0.9917
SVM	0.9834
Random forest	0.9666
kNN	0.9419
Árbol de decisión (CART rpart)	0.7849

```
kappaNN <- 0.9917
kappaSVM <- 0.9834
kappaRF <- 0.9666
kappaKNN <- 0.9419
kappaDT <- 0.7849
```

Los modelos con mayor valor Kappa son NN y SVM. Vamos a realizar un test binomial para ver si hay diferencias significaticas entre ellos.

```
binomialTest <- function(percentageSuccess1, percentageSuccess2, testingData = testing){
  binom.test(
    round(c(percentageSuccess1, 1-percentageSuccess1)* # porcentaje de acierto y error
            nrow(testingData)), # multiplicado por el número de instancias
                                # para obtener el número total de aciertos y fallos
    p = percentageSuccess2 # el porcentaje de acierto del modelo con el que comparar
  )
}
```

```
binomialTest(kappaNN, kappaSVM) # Comparación NN y SVM
```

```
##
##  Exact binomial test
##
## data:  round(c(percentageSuccess1, 1 - percentageSuccess1) * nrow(testingData))
## number of successes = 158, number of trials = 159, p-value =
## 0.5292
## alternative hypothesis: true probability of success is not equal to 0.9834
## 95 percent confidence interval:
##  0.9654579 0.9998408
## sample estimates:
## probability of success
##              0.9937107
```

El p-valor es ≥ 0.05, por lo que no hay una diferencia significativa entre NN y SVM.

```
binomialTest(kappaNN, kappaRF) # Comparación NN y RF
```

```
##
##  Exact binomial test
##
## data:  round(c(percentageSuccess1, 1 - percentageSuccess1) * nrow(testingData))
## number of successes = 158, number of trials = 159, p-value =
```

```
## 0.07099
## alternative hypothesis: true probability of success is not equal to 0.9666
## 95 percent confidence interval:
##  0.9654579 0.9998408
## sample estimates:
## probability of success
##              0.9937107
```

El p-valor es ≥ 0.05, por lo que no hay una diferencia significativa entre NN y RF.

```
binomialTest(kappaNN, kappaKNN) # Comparación NN y kNN
```

```
##
##  Exact binomial test
##
## data:  round(c(percentageSuccess1, 1 - percentageSuccess1) * nrow(testingData))
## number of successes = 158, number of trials = 159, p-value =
## 0.00177
## alternative hypothesis: true probability of success is not equal to 0.9419
## 95 percent confidence interval:
##  0.9654579 0.9998408
## sample estimates:
## probability of success
##              0.9937107
```

El p-valor es < 0.05, por lo que hay una diferencia significativa entre NN y kNN.

Interpretación de la comparación y elección del modelo

Por tanto, entre los modelos NN, SVM y RF podemos observar como no hay diferencias significativas en el valor de Kappa, mientras que entre NN y kNN sí las hay. Como el árbol de decisión tiene menor valor Kappa que kNN, ni siquiera lo tenemos en cuenta para la comparación.

La decisión de qué modelo escoger estará entonces entre NN, SVM y RF. El principio de la navaja de Ockham nos dice que, en condiciones similares, escojamos el modelo más simple. RF es el modelo más sencillo y fácil de interpretar, sin embargo, aunque los 3 modelos tengan un valor de Kappa similar, hay otras diferencias importantes entre ellos.

El modelo con menor tiempo de entrenamiento es, con gran diferencia, SVM, seguido de RF y de NN (que tiene un tiempo de entrenamiento muy alto en comparación con los otros). Además, RF es ligeramente más lento en las predicciones, dado que tiene que realizar la predicción en los 500 árboles. Otra ventaja más que tiene SVM es que escala bien para problemas de alta dimensionalidad, como es el caso de este problema. En estas pruebas hemos utilizado una versión reducida del problema, con apenas 100 variables de las 20500 que tiene realmente. Si consideramos que las 20500 variables son relevantes y deben usarse para entrenar el modelo, SVM sería un buen paradigma a utilizar, por lo explicado previamente. Tampoco hemos tenido la necesidad de utilizar una función de kernel, por lo que el SVM es, dentro de lo que cabe, simple. Además, en las NN hay muchos parámetros que ajustar, mientras que en RF y SVM (lineal) tenemos un único parámetro principal: en SVM (lineal) tenemos C, y en RF tenemos mtry (al menos en esta versión del algoritmo, ya que hay versiones más complejas con más parámetros).

Por tanto, el modelo final seleccionado será el SVM. Es con gran diferencia el que tiene menor tiempo de entrenamiento, tarda menos que RF en realizar las predicciones, tiene pocos parámetros que ajustar, y tendrá menos problemas si se incrementa el número de variables.