# PEC2: Classification and diagnostic prediction of cancers using gene expression profiling

Artificial neural networks & support vector machines

*Escribir vuestro nombre y apellidos*

*27 de diciembre, 2017*

## Índice

# Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks

En esta PEC vamos a realizar un informe que analiza un caso basado en los datos del artículo:

**Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks. Khan et al. Nature Medicine, 2001, 6, 673-679**

Los datos se pueden obtener directamente de la revista Nature Medicine o directamente de la PEC.

En dicho artículo se investiga la predición del diagnóstico de un tipo de cancer, "small, round blue cell tumors (SRBCTs)" en la infancia usando información del perfil de expresión génica obtenica mediante técnicas de microarrays.

Se estudian 4 tipos de canceres:

*The small, round blue cell tumors (SRBCTs) of childhood, which include neuroblastoma (NB), rhabdomyosarcoma (RMS), non- Hodgkin lymphoma (NHL) and the Ewing family of tumors (EWS), are so named because of their similar appearance on routine histology1. However, accurate diagnosis of SRBCTs is essential because the treatment options, responses to therapy and prognoses vary widely depending on the diagnosis. As their name implies, these cancers are difficult to distinguish by light microscopy, and currently no single test can precisely distinguish these cancers.*

El proceso de clasificación (ver Fig. 1) es mucho más elaborado que los presentados en las unidades. No reproduciremos este esquema, aunque es importante que se entienda. Se basa en una red neuronal artificial usando 3-fold crossvalidación y repitiendo el proceso 1250 veces mediante particiones aleatorias (proceso similar al bootstrap) para poder estudiar la robustez del modelo. Observar que el número de variables es muy grande (2308) así que se ha optado por realizar un análisis de componentes principales para reducir la dimensión de las variables iniciales y usar solo las 10 primeras en el algoritmo.

El análisis de componentes principales (PCA, en ingles) es una técnica básica y muy utilizada en análisis multivariante para reducir el número de variables creando nuevas variables como combinación lineal de las originales buscando máximizar la varianza explicada. Como no sé si sabeis realizar en R un PCA he optado por crear un fichero con el resultado del PCA llamado "pcaComponents.csv".

En esta PEC se usará los datos del artículo para implementar el algoritmo de red neuronal artificial y "support vector machine" (SVM) para predecir los cuatro tipos de canceres.

# Algoritmo Red Neuronal Artificial (ANN)

Las redes neuronales artificiales se inspira en las redes neuronales como las que se tiene en el cerebro. Las neuronas se sustituyen por nodos que reciben y envian señales (información). Se crea una red con diferentes capas interconectadas para procesar la información. Cada capa esta formada por un grupo de nodos que transmite la información a los otros nodos de las capas siguientes.

Una red neuronal artificial se caracteriza por:

- La topología: Esto corresponde al número de capas y de nodos. Además de la dirección en que se la información pasa de un nodo al siguiente, dentro de capas o entre capas.
- La función de activación: Función que recibe un conjunto de entradas e integra la señales para transmitir la nformación a otro nodo/capa.
- El algoritmo de entrenamiento: Establece la importancia de cada conexión para transmitir o no la señal a los nodos correspondientes. El más usado es el algoritmo "backpropagation". El nombre indica que para corregir los errores de predicción va hacia atras de la red corrigiendo los pesos de los nodos.

Las fortalezas y debilidades de este algoritmo son:

| Fortalezas | Debilidades |
|---|---|
| - Adaptable a clasificación o problemas de predicción numérica | - Requiere de gran potencia computacional y en general es de aprendizaje lento, particularmente si la topología es compleja |
| - Capaz de modelar patrones más complejos que casi cualquier otro algoritmo | - Propenso a sobreajustar los datos de entrenamiento |
| - No necesita muchas restricciones acerca de las relaciones subyacentes de los datos | - Es un modelo de caja negra complejo que es difícil, si no imposible, de interpretar |

## Step 1 - Recoger los datos

Se usará los archivos depositados en la PEC ya que se tiene el resultado del PCA.

```
fold <- "dataset"

file1_ANN <- "pcaComponents.csv"
file2 <- "class.csv"
```

```
mydata0 <- read.csv(file=file.path(params$fold,params$file1_ANN))
clase <- read.csv(file=file.path(params$fold,params$file2))
#gene.names <- read.csv(file=file.path(fold,"names.csv"))
```

El primer conjunto de datos denominado *pcaComponents.csv* esta formado por 83 muestras, entre biopsias de tumores y líneas celulares. Es el resultado de haber realizado el análisis de componentes principales (PCA) sobre los datos originales.

El segundo conjunto de datos denominado *class.csv* corresponde a la clase de tumor de los anteriores datos.

Como variables solo se van a escoger las *10* primeras variables del PCA, es decir, las *10* primeras componentes principales.

```
# Se selecciona las 10 primeras componentes
mydata <- mydata0[,1:10]
```

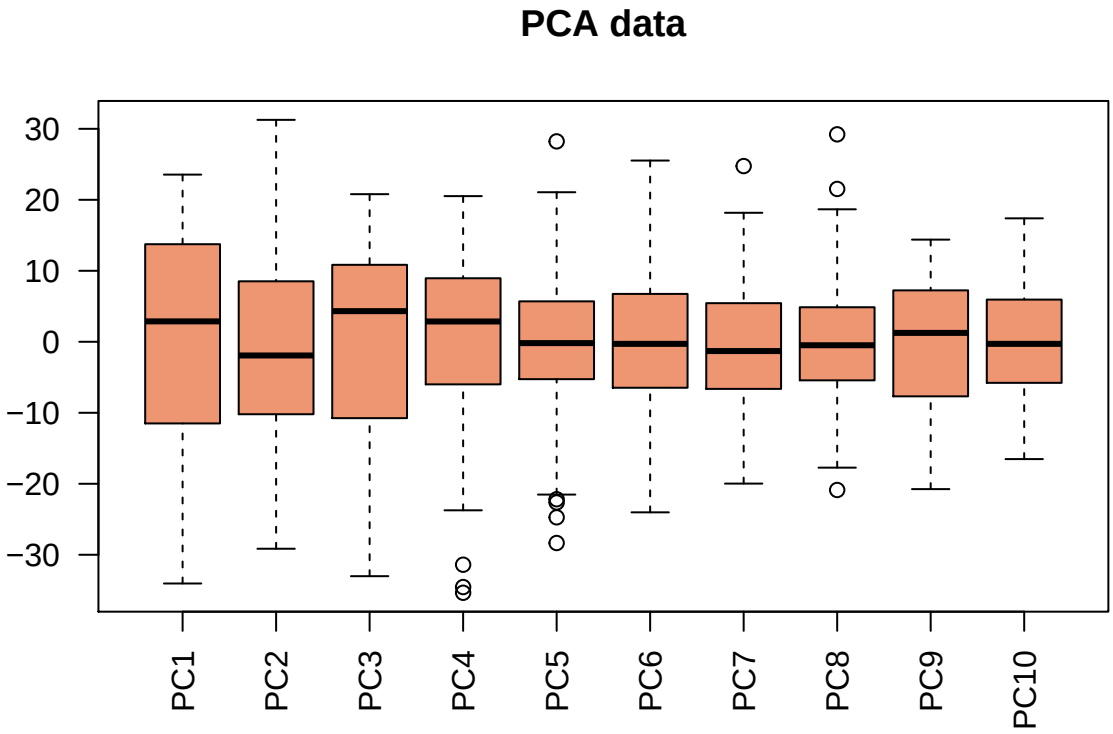
## Step 2 - Exploración y preparación de los datos

En primer lugar veremos los seis primeros registros:

```
        PC1       PC2       PC3      PC4         PC5       PC6        PC7        PC8
1  -2.060636 11.447834 -3.259748 9.006774  -5.0305477  3.810002  -6.279456  3.8873006
2  -2.142176 21.413399 -1.184447 9.026926 -10.2230153 -3.155479  -8.389044 -0.4873145
3 -17.035724 15.583160 -2.693816 9.562166  -9.5895712 -1.007284 -10.920550  1.6646891
4  -6.047469 26.085785  1.683572 7.914051  -2.0586879 17.630749  24.749087 13.3739683
5 -24.171911  7.635202 13.849113 4.861939  -1.4898417 11.462415  -2.045261 -6.1078435
6 -20.643544  1.650100 13.859697 7.213407  -0.3532495  6.323860 -14.271809 -7.4320632
        PC9       PC10
1 -10.893210  1.0351450
2  -7.720870  0.6601277
3  -6.759050  1.4410271
4 -15.156545 -3.8204845
5  -2.863407  2.1654871
6  -8.654020  2.4740657
```

Un exploración gráfica mediante boxplot da:

```r
boxplot(mydata, las=2, col="lightsalmon2", main="PCA data")
```



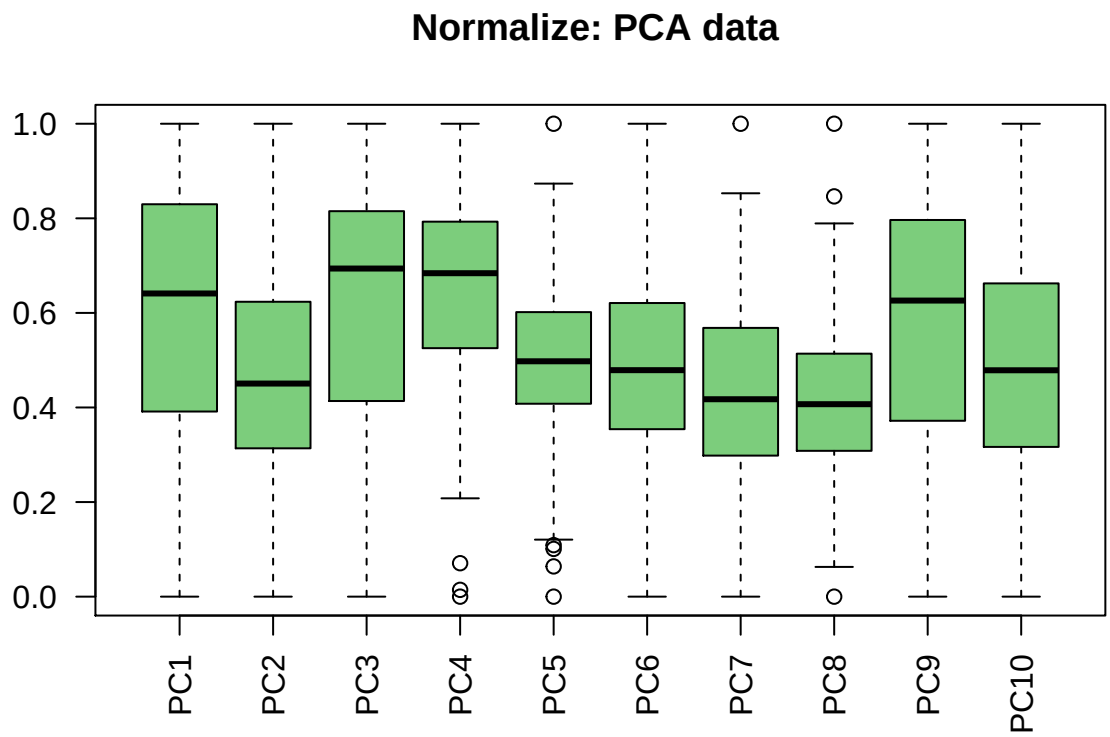

Hay que normalizar las variables para que tomen valores entre 0 y 1. Se define la función `normalize` para realizar está operación.

```r
# custom normalization function
normalize <- function(x) {
return((x - min(x)) / (max(x) - min(x)))
}

mydata_nrm <- as.data.frame(lapply(mydata, normalize))
summary(mydata_nrm)
```

```
      PC1                PC2                PC3                PC4                PC5          
 Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000  
 1st Qu.:0.3914   1st Qu.:0.3136   1st Qu.:0.4137   1st Qu.:0.5254   1st Qu.:0.4079  
 Median :0.6411   Median :0.4506   Median :0.6938   Median :0.6840   Median :0.4977  
 Mean   :0.5911   Mean   :0.4825   Mean   :0.6136   Mean   :0.6328   Mean   :0.5010  
 3rd Qu.:0.8298   3rd Qu.:0.6236   3rd Qu.:0.8151   3rd Qu.:0.7929   3rd Qu.:0.6017  
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000  
      PC6                PC7                PC8                PC9                PC10         
 Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000  
 1st Qu.:0.3541   1st Qu.:0.2983   1st Qu.:0.3083   1st Qu.:0.3719   1st Qu.:0.3166  
 Median :0.4789   Median :0.4175   Median :0.4070   Median :0.6261   Median :0.4787  
 Mean   :0.4848   Mean   :0.4466   Mean   :0.4167   Mean   :0.5904   Mean   :0.4872  
 3rd Qu.:0.6209   3rd Qu.:0.5683   3rd Qu.:0.5138   3rd Qu.:0.7965   3rd Qu.:0.6624  
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000  
```

El boxplot de los datos transformados queda:

```
boxplot(mydata_nrm, las=2, col="palegreen3", main="Normalize: PCA data ")
```




Por otro parte, se tiene la información de la clase de tumors registrada en cada muestra. Mediante una tabla se muestra el número de clases de cada tipo:

```
table(clase)
```

```
clase
 1  2  3  4
29 11 18 25
```

La notación de cada clase es numerica. Seria más claro hacer una notación con etiquetas.

```
lab.group <- c("EWS","BL","NB","RMS")
clase.f <- factor(clase$x,labels=lab.group)
```

Ahora la tabla queda como:

```
table(clase.f)
```

```
clase.f
EWS  BL  NB RMS
 29  11  18  25
```

Para acabar, se crea un dataset, `mydata_ann` que contiene las variables explicativas y 4 nuevas variables dummies que servirán para indicar el tipo de tumor.

```
# Create 4 news dummies variables
mydata_ann <- mydata_nrm
```

```
mydata_ann$EWS <- clase.f=="EWS"
mydata_ann$BL <- clase.f=="BL"
mydata_ann$NB <- clase.f=="NB"
mydata_ann$RMS <- clase.f=="RMS"
```

Primero se divide el dataset en una parte de entrenamiento y en otra de test.

```
################## data spliting
set.seed(params$seed.train) #fijar la semilla para el generador pseudoaleatorio
n_train <- params$p.train
n <- nrow(mydata_ann)
train <- sample(n,floor(n*n_train))
mydata_ann.train <- mydata_ann[train,]
mydata_ann.test  <- mydata_ann[-train,]
```

## Step 3 - Entrenar el modelo con los datos

Ahora vamos a entrenar el primer modelo con los datos de train. Se usa la función `neuralnet` del paquete que tiene el mismo nombre.

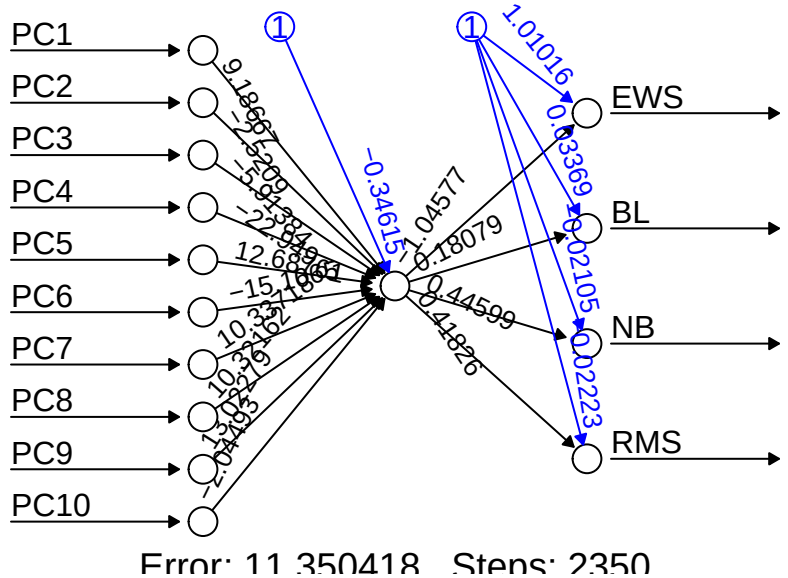
```
################ libraries loading
#require(neuralnet)

## Create a formula for a model with a large number of variables:
xnam <- names(mydata_ann[1:10])
(fmla <- as.formula(paste("EWS+BL+NB+RMS ~ ",  paste(xnam, collapse= "+"))))
```

```
EWS + BL + NB + RMS ~ PC1 + PC2 + PC3 + PC4 + PC5 + PC6 + PC7 +
    PC8 + PC9 + PC10
```

```
# simple ANN with only a single hidden neuron
set.seed(params$seed.clsfier) # to guarantee repeatable results
mydata_model <- neuralnet(fmla,
                          data = mydata_ann.train,
                          hidden=1)
```

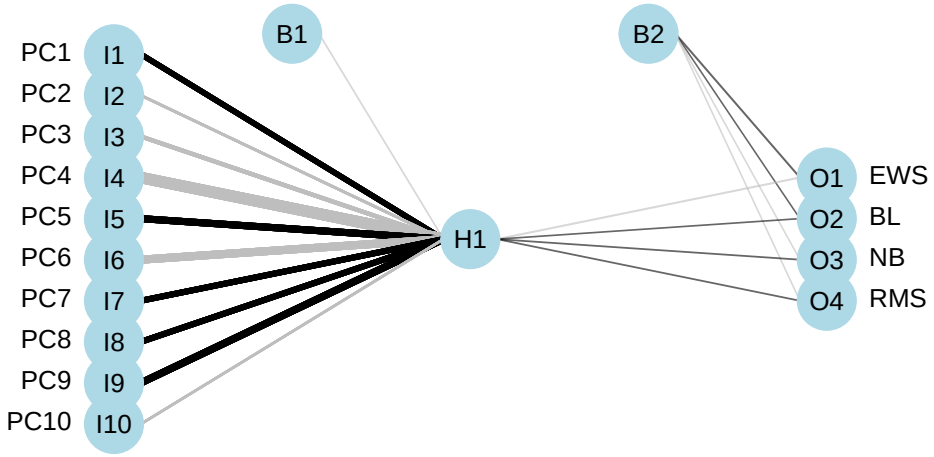
La representación de la red neuronal artificial es:

```
#Representamos gráficamente el modelo generado
plot(mydata_model, rep="best")
```




También se puede hacer la representación con otro package

```
# ANN representation
#require(NeuralNetTools)

plotnet(mydata_model, alpha=0.6)
```



## Step 4 - Evaluación del rendimiento del algoritmo

Una vez obtenido el primer modelo, se evalua su rendimiento con los datos de test. Se debe de clasificar las muestras de los datos de `test` con la función `compute`.

```
# obtain model results
model_results <- compute(mydata_model, mydata_ann.test[1:10])$net.result

# Put multiple binary output to categorical output
maxidx <- function(arr) {
  return(which(arr == max(arr)))
}
idx <- apply(model_results, 1, maxidx)
prediction <- factor(idx,levels=c(1,2,3,4),labels=lab.group )
res <- table(prediction, clase.f[-train])
```

Al final, se obtiene la matriz de confusión con las predicciones y las clases reales. La función `confusionMatrix` del paquete `caret` genera esta matriz y calcula diferentes del rendimiento del algoritmo.

```
#require(caret, quietly = TRUE)
(conf_matrix<- confusionMatrix(res))
```

```
Confusion Matrix and Statistics


prediction EWS BL NB RMS
       EWS   9  1  0   0
       BL    0  0  0   0
       NB    0  2  4  12
       RMS   0  0  0   0

Overall Statistics

               Accuracy : 0.4642857
                 95% CI : (0.2751086, 0.6613009)
    No Information Rate : 0.4285714
    P-Value [Acc > NIR] : 0.4210306

                  Kappa : 0.3247588
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: EWS Class: BL Class: NB Class: RMS
Sensitivity           1.0000000 0.0000000 1.0000000  0.0000000
Specificity           0.9473684 1.0000000 0.4166667  1.0000000
Pos Pred Value        0.9000000       NaN 0.2222222        NaN
Neg Pred Value        1.0000000 0.8928571 1.0000000  0.5714286
Prevalence            0.3214286 0.1071429 0.1428571  0.4285714
Detection Rate        0.3214286 0.0000000 0.1428571  0.0000000
Detection Prevalence  0.3571429 0.0000000 0.6428571  0.0000000
Balanced Accuracy     0.9736842 0.5000000 0.7083333  0.5000000
```

El modelo de ANN de un nodo oculto obtiene un valor de precisión de 0.46 y un estadístico $\kappa = 0.32$. Los valores de sensibilidad y especificidad varían según el tipo de tumor, obteniendo como valor medio 0.5 y 0.84 respectivamente.

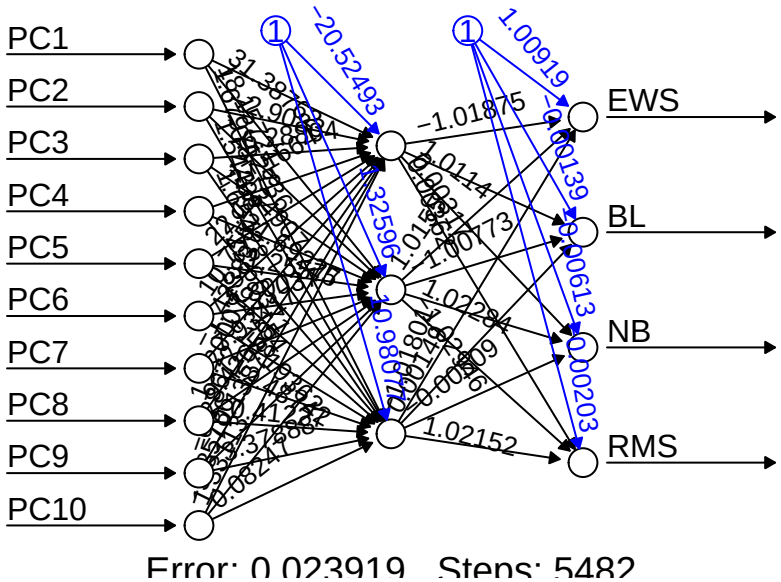
## Step 5 - Mejora del rendimiento del algoritmo

El primer modelo fue con *un nodo* en la capa oculta. Ahora se plantea *3 nodos* en la capa oculta para tratar de mejorar el rendimiento.

```
# a more complex neural network topology with 5 hidden neurons
set.seed(params$seed.clsfier) # to guarantee repeatable results
mydata_model2 <- neuralnet(fmla,
                           data = mydata_ann.train,
                           linear.output = TRUE,
                           hidden=3)
```

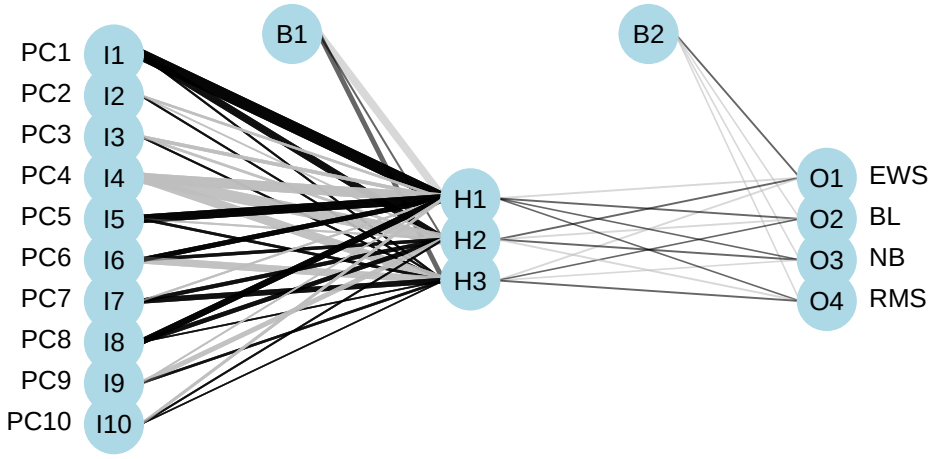
La representación de la red neuronal artificial es:

```
# visualize the network topology
plot(mydata_model2, rep="best")
```




También se puede hacer la representación con otro package

```
# ANN representation
plotnet(mydata_model2, alpha=0.6)
```




El resultado de la matriz de confusión es:

```
# evaluate the results as we did before
model_results2 <- compute(mydata_model2, mydata_ann.test[1:10])$net.result
idx <- apply(model_results2, 1, maxidx)
prediction2 <- factor(idx,levels=c(1,2,3,4),labels=lab.group )
#prediction2 <- c('EWS', 'BL',"NB","RMS")[idx]
res <- table(prediction2, clase.f[-train])

(conf_matrix3<- confusionMatrix(res))
```

```
Confusion Matrix and Statistics


prediction2 EWS BL NB RMS
        EWS   8  0  0   1
        BL    0  3  0   0
        NB    1  0  4   1
        RMS   0  0  0  10

Overall Statistics

               Accuracy : 0.8928571
                 95% CI : (0.7177356, 0.9773349)
    No Information Rate : 0.4285714
    P-Value [Acc > NIR] : 0.0000004215255

                  Kappa : 0.8472727
 Mcnemar's Test P-Value : NA
```

```
Statistics by Class:

                     Class: EWS Class: BL Class: NB Class: RMS
Sensitivity           0.8888889 1.0000000 1.0000000  0.8333333
Specificity           0.9473684 1.0000000 0.9166667  1.0000000
Pos Pred Value        0.8888889 1.0000000 0.6666667  1.0000000
Neg Pred Value        0.9473684 1.0000000 1.0000000  0.8888889
Prevalence            0.3214286 0.1071429 0.1428571  0.4285714
Detection Rate        0.2857143 0.1071429 0.1428571  0.3571429
Detection Prevalence  0.3214286 0.1071429 0.2142857  0.3571429
Balanced Accuracy     0.9181287 1.0000000 0.9583333  0.9166667
```

El nuevo modelo de ANN con tres nodos en la capa oculta obtiene un valor de precisión de 0.89 y un estadístico $\kappa = 0.85$. Los valores de sensibilidad y especificidad varían según el tipo de tumor, obteniendo como valor medio 0.93 y 0.97 respectivamente.

En resumen, el nuevo modelo obtenido con tres nodos tiene mayor precisión que el modelo más simple de un solo nodo. Además, el nuevo modelo obtenido con tres nodos tiene mayor valor de kappa que el modelo más simple de un solo nodo.

### 3-fold crossvalidation

Por ultimo, se plantea realizar el modelo de tres nodos con 3-fold crossvalidation usando el paquete `caret`.

En primer lugar preparo el dataset, con las variables explicativas y la variable respuesta tipo `factor` con 4 clases.

```
# Create new dataset
mydata_caret <- mydata
mydata_caret$clase <- clase.f
```

El modelo de entrenamiento es:

```
###3-fold crossvalidation
set.seed(params$seed.clsfier) # to guarantee repeatable results
model <- train(clase ~ ., mydata_caret, method='nnet',
               trControl= trainControl(method='cv', number=3),
               tuneGrid= NULL, tuneLength=3 ,trace = FALSE)
```

El modelo obtenido es:

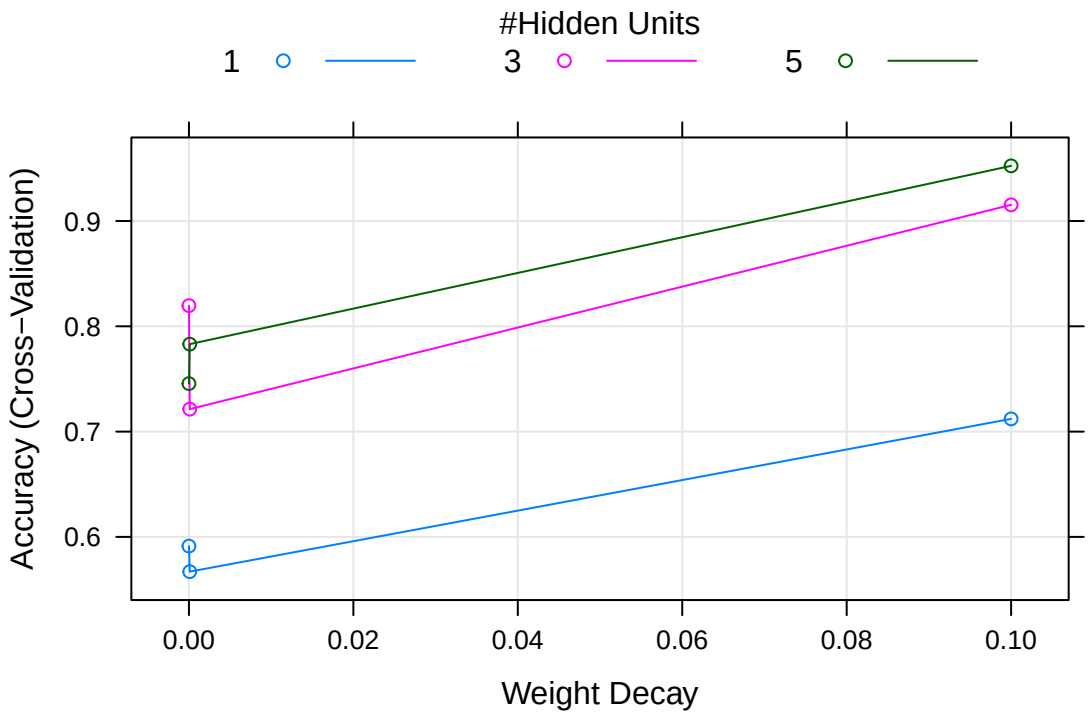
```
model
```

```
Neural Network

83 samples
10 predictors
 4 classes: 'EWS', 'BL', 'NB', 'RMS'

No pre-processing
Resampling: Cross-Validated (3 fold)
Summary of sample sizes: 55, 55, 56
Resampling results across tuning parameters:

  size  decay   Accuracy      Kappa
  1     0.0000  0.5912698413  0.4236051678
  1     0.0001  0.5670194004  0.3642854122
```

```
1     0.1000  0.7120811287  0.5773233217
3     0.0000  0.8196649030  0.7465262014
3     0.0001  0.7213403880  0.6160302800
3     0.1000  0.9153439153  0.8847033677
5     0.0000  0.7455908289  0.6406940862
5     0.0001  0.7830687831  0.6978256410
5     0.1000  0.9523809524  0.9354094579

Accuracy was used to select the optimal model using the largest value.
The final values used for the model were size = 5 and decay = 0.1.
```

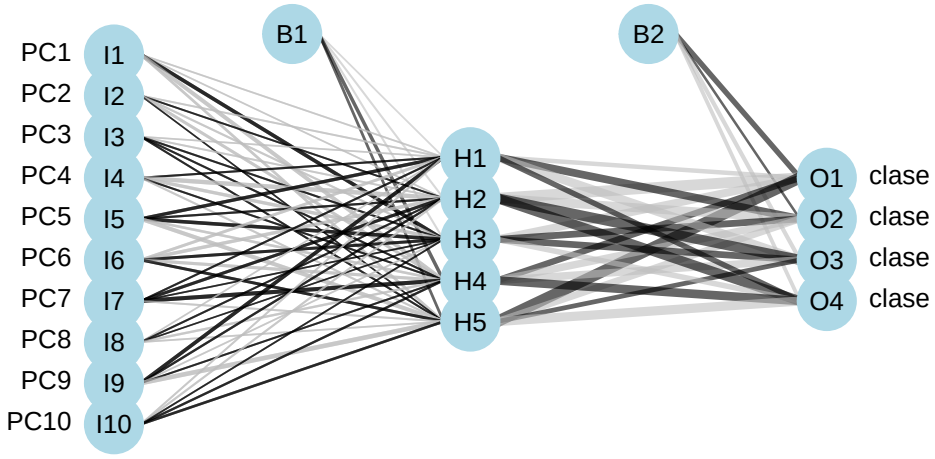
El gráfico del rendimiento según diferente parámetros es

```
# ANN representation
plot(model,rep=best)
```




La representación gráfica es:

```
# ANN representation
plotnet(model, alpha=0.6)
```




Los pesos para cada variable o nodo son:

```
summary(model)
```

```
a 10-5-4 network with 79 weights
options were - softmax modelling  decay=0.1
  b->h1  i1->h1  i2->h1  i3->h1  i4->h1  i5->h1  i6->h1  i7->h1  i8->h1  i9->h1 i10->h1
  -0.05   -0.04   -0.07   -0.02    0.12    0.43   -0.32    0.12    0.00    0.51   -0.08
  b->h2  i1->h2  i2->h2  i3->h2  i4->h2  i5->h2  i6->h2  i7->h2  i8->h2  i9->h2 i10->h2
  -0.11   -0.01    0.07   -0.07   -0.66    0.15   -0.44    0.37    0.11    0.18   -0.11
  b->h3  i1->h3  i2->h3  i3->h3  i4->h3  i5->h3  i6->h3  i7->h3  i8->h3  i9->h3 i10->h3
  -0.12    0.52   -0.12    0.11   -0.43    0.48    0.34    0.03    0.01   -0.03    0.05
  b->h4  i1->h4  i2->h4  i3->h4  i4->h4  i5->h4  i6->h4  i7->h4  i8->h4  i9->h4 i10->h4
   0.48   -0.58   -0.34    0.12    0.11   -0.23   -0.37    0.61   -0.18    0.09    0.18
  b->h5  i1->h5  i2->h5  i3->h5  i4->h5  i5->h5  i6->h5  i7->h5  i8->h5  i9->h5 i10->h5
   0.29   -0.26    0.01    0.11   -0.10   -0.59    0.33   -0.08   -0.04   -0.72    0.25
 b->o1 h1->o1 h2->o1 h3->o1 h4->o1 h5->o1
  0.96  -0.71  -2.32  -1.60   1.18   2.45
 b->o2 h1->o2 h2->o2 h3->o2 h4->o2 h5->o2
  0.24   1.52  -1.83   1.30  -1.45  -1.33
 b->o3 h1->o3 h2->o3 h3->o3 h4->o3 h5->o3
 -0.65  -1.78   2.14   1.27  -1.50   0.78
 b->o4 h1->o4 h2->o4 h3->o4 h4->o4 h5->o4
 -0.55   0.96   2.01  -0.98   1.77  -1.90
```

En resumen, el mejor modelo de ANN con 3-fold crossvalidation de los parametros explorados se obtiene con 5 nodos ocultos y un valor de *decay* de 0.1, con el se obtiene una precisión media de 0.95 y un estadístico $\kappa$ igual a 0.94.

Finalmente, podemos decir que el modelo obtenido por 3-fold crossvalidation con la función nnet es el mejor

modelo obtenido con una red neuronal.

# Algoritmo Support Vector Machine (SVM)

Las máquinas de vectores de soporte (Support Vector Machines, SVM) son un conjunto de algoritmos de aprendizaje supervisado, dirigido tanto a la resolución de problemas de clasificación como de regresión, indistintamente.

Los algoritmos de SVM se basa en buscar el hiperplano que tenga mayor margen posible y de forma homogénea entre las clases. Formalmente, una SVM construye un hiperplano o conjunto de hiperplanos en un espacio de dimensionalidad muy alta (o incluso infinita) para crear particiones bastante homogéneas a cada lado.

Algunas de las aplicaciones son:

- Clasificar genes diferencialmente expresados a partir de datos de microarrays.
- Clasificación de texto en distintas categorías temáticas.
- Detección de eventos críticos de escasa frecuencia, como terremotos.

Cuando los datos no son separables de forma lineal, es necesario el uso de kernels, o funciones de similitud y especificar un parámetro C para minimizar la función de coste. La elección de este parámetro es a base de ensayo/error, pero se buscan valores que no sean extremos en la búsqueda del equilibrio sesgo/varianza.

Los kernels más populares son el lineal y el gausiano, aunque hay otros como el polinomial, string kernel, chi-square kernel, etc.

| **Fortalezas** | **Debilidades** |
|---|---|
| - Se puede usar para problemas de clasificación o predicción numérica | - Encontrar el mejor modelo requiere probar diferentes kernels y parámetros del modelo (prueba y error) |
| - Funciona bastante bien con datos ruidosos y no es muy propenso al overfitting | - Lento de entrenar, sobre todo a medida que aumenta el número de características |
| - Puede ser más fácil de usar que las redes neuronales, en particular debido a la existencia de varios algoritmos SVM bien soportados | - Los resultados del modelo son difícil, si no imposible, de interpretar (caja negra) |
| - Gana popularidad debido a su alta precisión y ganancias de alto perfil en competiciones de minería de datos | |

## Step 1 - Recoger los datos

Con el algoritmo *Support Vector Machine*, se usa los datos de expresión génica (microarrays) originales, a diferencia del caso del modelo ANN que se realizo una PCA y limita a las 10 primeras componentes principales como variables explicativas.

```
fold <- "dataset"

file1_SVM <- "data.csv"
file2 <- "class.csv"
```

```
#Leemos los datos
mydata <- read.csv(file=file.path(params$fold,params$file1_SVM))
clase <- read.csv(file=file.path(params$fold,params$file2))

dim(mydata)
```

```
[1]   83 2308
```

El primer conjunto de datos denominado *data.csv* esta formado por 83 muestras, entre biopsias de tumores y líneas celulares y tiene la expresión génica de 2308 genes.

El segundo conjunto de datos denominado *class.csv* corresponde a la clase de tumor de los anteriores datos.

Se añade la variable clase como factor al conjunto de datos explicativos.

```
#Añadir  clase
lab.group <- c("EWS","BL","NB","RMS")
clase.f <- factor(clase$x,labels=lab.group)

mydata$clase <- clase.f
```

## Step 2 - Exploración y preparación de los datos

En primer lugar veremos una muestra del dataset: los seis primeros registros y las ultimas 9 variables:

```
   V2300  V2301  V2302  V2303  V2304  V2305  V2306  V2307  V2308 clase
1 1.6679 0.1493 0.6918 1.4151 0.2756 0.1521 0.3175 0.7240 0.2044   EWS
2 3.6014 0.3048 1.7957 1.0701 0.2688 0.1932 0.4140 1.2708 0.2990   EWS
3 1.5152 0.2382 0.8720 0.6819 0.3221 0.2156 0.3227 1.2142 0.2230   EWS
4 1.0282 0.1049 0.5632 1.2264 0.8123 0.2758 0.3016 0.7235 0.0871   EWS
5 0.5961 0.0707 0.4001 1.5271 0.4084 0.6412 0.3552 1.3928 0.2157   EWS
6 1.3635 0.0580 0.7737 2.8123 0.3279 0.3388 0.1654 0.8991 0.2525   EWS
```

La estructura de los datos es una característica que siempre hay que revisar. En nuestro caso tenemos muchas. A titulo de ejemplo se muestra la estructura de las anteriores 9 variables mostradas.

```
str(mydata[1:5, 2300:2309])
```

```
'data.frame':   5 obs. of  10 variables:
 $ V2300: num  1.668 3.601 1.515 1.028 0.596
 $ V2301: num  0.1493 0.3048 0.2382 0.1049 0.0707
 $ V2302: num  0.692 1.796 0.872 0.563 0.4
 $ V2303: num  1.415 1.07 0.682 1.226 1.527
 $ V2304: num  0.276 0.269 0.322 0.812 0.408
 $ V2305: num  0.152 0.193 0.216 0.276 0.641
 $ V2306: num  0.318 0.414 0.323 0.302 0.355
 $ V2307: num  0.724 1.271 1.214 0.724 1.393
 $ V2308: num  0.2044 0.299 0.223 0.0871 0.2157
 $ clase: Factor w/ 4 levels "EWS","BL","NB",..: 1 1 1 1 1
```

Una breve estadistica descriptiva de las 9 anteriores variables es:

```
summary(mydata[, 2300:2309])
```

```
     V2300              V2301              V2302              V2303        
 Min.   :0.1079000   Min.   :0.0580000   Min.   :0.1029000   Min.   :0.0371000  
 1st Qu.:0.3629000   1st Qu.:0.1567000   1st Qu.:0.3538500   1st Qu.:0.2392000  
 Median :0.5806000   Median :0.2856000   Median :0.4811000   Median :0.6345000  
 Mean   :0.6929904   Mean   :0.3350422   Mean   :0.5519518   Mean   :0.8318386  
 3rd Qu.:0.8761000   3rd Qu.:0.4432500   3rd Qu.:0.6620000   3rd Qu.:1.3320500  
 Max.   :3.6014000   Max.   :1.2339000   Max.   :1.8008000   Max.   :2.8297000  
     V2304              V2305              V2306              V2307        
 Min.   :0.0082000   Min.   :0.0041000   Min.   :0.0430000   Min.   :0.0678000  
 1st Qu.:0.0971500   1st Qu.:0.1191000   1st Qu.:0.2553000   1st Qu.:0.4343000  
 Median :0.2046000   Median :0.1863000   Median :0.3450000   Median :0.6512000  
```

```
 Mean   :0.2412446   Mean   :0.2594229   Mean   :0.4590096   Mean   :0.6977241
 3rd Qu.:0.3043000   3rd Qu.:0.3457000   3rd Qu.:0.5363000   3rd Qu.:0.8633500
 Max.   :1.0609000   Max.   :1.4709000   Max.   :1.6150000   Max.   :1.7691000
     V2308           clase
 Min.   :0.0446000   EWS:29
 1st Qu.:0.1501500   BL :11
 Median :0.2481000   NB :18
 Mean   :0.2806181   RMS:25
 3rd Qu.:0.3721500
 Max.   :1.0420000
```

Entramos en la fase de separar la muestra en train y test. Como en el algoritmo de ANN ya se han separados los individuos solo falta asignar cada dataset al grupo adecuado.

```
mydata.train <- mydata[train,]
mydata.test  <- mydata[-train,]
```

## Step 3 - Entrenar el modelo con los datos

Ahora se entrena modelo SVM lineal con los datos de train. Se usa la función `ksvm` del paquete `kernlab`.

```
# begin by training a simple linear SVM
#library(kernlab)
set.seed(params$seed.clsfier) # to guarantee repeatable results
mydata_model1 <- ksvm(clase ~ ., data = mydata.train,
                      kernel = "vanilladot")
```

```
 Setting default kernel parameters
```

El modelo queda como

```
# look at basic information about the model
mydata_model1
```

```
Support Vector Machine object of class "ksvm"

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

Linear (vanilla) kernel function.

Number of Support Vectors : 51

Objective Function Value : -0.0018 -0.0028 -0.0033 -0.0017 -0.0013 -0.0027
Training error : 0
```

## Step 4 - Evaluación del rendimiento del algoritmo

Una vez obtenido el modelo de SVM lineal, se evalua su rendimiento con los datos de test. Se debe de clasificar las muestras de los datos de `test` con la función `predict`.

```
# predictions on testing dataset
mydata_predict1 <- predict(mydata_model1, mydata.test)

res <- table(mydata_predict1, mydata.test$clase)
```

Al final, se obtiene la matriz de confusión con las predicciones y las clases reales. La función `confusionMatrix` del paquete `caret` genera esta matriz y calcula diferentes del rendimiento del algoritmo.

```
#require(caret)

(conf_mat.s1 <- confusionMatrix(res))
```

```
Confusion Matrix and Statistics


mydata_predict1 EWS BL NB RMS
            EWS   9  0  0   1
            BL    0  3  0   0
            NB    0  0  4   1
            RMS   0  0  0  10

Overall Statistics

               Accuracy : 0.9285714
                 95% CI : (0.7649652, 0.9912295)
    No Information Rate : 0.4285714
    P-Value [Acc > NIR] : 0.00000003532744

                  Kappa : 0.8972477
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: EWS Class: BL Class: NB Class: RMS
Sensitivity           1.0000000 1.0000000 1.0000000  0.8333333
Specificity           0.9473684 1.0000000 0.9583333  1.0000000
Pos Pred Value        0.9000000 1.0000000 0.8000000  1.0000000
Neg Pred Value        1.0000000 1.0000000 1.0000000  0.8888889
Prevalence            0.3214286 0.1071429 0.1428571  0.4285714
Detection Rate        0.3214286 0.1071429 0.1428571  0.3571429
Detection Prevalence  0.3571429 0.1071429 0.1785714  0.3571429
Balanced Accuracy     0.9736842 1.0000000 0.9791667  0.9166667
```

El algoritmo de SVM lineal tiene un valor de precisión de 0.93 y un estadístico $\kappa = 0.9$. Vemos que los valores de sensibilidad y especificidad varían según el factor, obteniendo como valor medio 0.96 y 0.98 respectivamente.

## Step 5 - Mejora del rendimiento del algoritmo

El modelo que se presenta es un SVM con función gaussiana o rbf.

```
# begin by training a Gaussian SVM
#library(kernlab)
set.seed(params$seed.clsfier) # to guarantee repeatable results
mydata_model2 <- ksvm(clase ~ ., data = mydata.train,
                      kernel = "rbfdot")
```

El modelo queda como

```
# look at basic information about the model
mydata_model2
```

```
Support Vector Machine object of class "ksvm"

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

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.000209572403335019

Number of Support Vectors : 55

Objective Function Value : -8.0423 -10.9106 -13.1993 -7.4708 -6.4026 -10.3091
Training error : 0
```

Una vez obtenido el modelo de SVM lineal, se evalua su rendimiento con los datos de test. Se debe de clasificar las muestras de los datos de `test` con la función `predict`.

```
# predictions on testing dataset
mydata_predict2 <- predict(mydata_model2, mydata.test)

res <- table(mydata_predict2, mydata.test$clase)
```

Al final, se obtiene la matriz de confusión con las predicciones y las clases reales. La función `confusionMatrix` del paquete `caret` genera esta matriz y calcula diferentes del rendimiento del algoritmo.

```
#require(caret)

(conf_mat.s2 <- confusionMatrix(res))
```

```
Confusion Matrix and Statistics


mydata_predict2 EWS BL NB RMS
            EWS   9  0  0   5
            BL    0  3  0   0
            NB    0  0  4   3
            RMS   0  0  0   4

Overall Statistics

               Accuracy : 0.7142857
                 95% CI : (0.5133317, 0.8677635)
    No Information Rate : 0.4285714
    P-Value [Acc > NIR] : 0.002114693

                  Kappa : 0.609075
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: EWS Class: BL Class: NB Class: RMS
Sensitivity           1.0000000 1.0000000 1.0000000  0.3333333
Specificity           0.7368421 1.0000000 0.8750000  1.0000000
Pos Pred Value        0.6428571 1.0000000 0.5714286  1.0000000
Neg Pred Value        1.0000000 1.0000000 1.0000000  0.6666667
Prevalence            0.3214286 0.1071429 0.1428571  0.4285714
Detection Rate        0.3214286 0.1071429 0.1428571  0.1428571
```

```
Detection Prevalence  0.5000000 0.1071429 0.2500000  0.1428571
Balanced Accuracy     0.8684211 1.0000000 0.9375000  0.6666667
```

El algoritmo de SVM de función RBF tiene un valor de precisión de 0.71 y un estadístico $\kappa = 0.61$. Como se puede esperar, los valores de sensibilidad y especificidad varían según la clase, obteniendo como valor medio 0.83 y 0.9 respectivamente.

En resumen, se puede decir que el modelo obtenido con la función RBF no mejora el modelo lineal en cuanto a precisión. Además, el nuevo modelo obtenido de SVM con la función RBF no mejora el valor de kappa del modelo más sencillo de SVM con la función lineal.

### 3-fold crossvalidation

Por ultimo, se plantea realizar el algoritmo de SVM con la función lineal con 3-fold crossvalidation usando el paquete `caret`.

El modelo de entrenamiento es:

```
###3-fold crossvalidation
set.seed(params$seed.clsfier) # to guarantee repeatable results
model_sc <- train(clase ~ ., mydata, method='svmLinear',
                  trControl= trainControl(method='cv', number=3),
                  tuneGrid= NULL, trace = FALSE)
```

El modelo obtenido es:

```
model_sc
```

```
Support Vector Machines with Linear Kernel

  83 samples
2308 predictors
   4 classes: 'EWS', 'BL', 'NB', 'RMS'

No pre-processing
Resampling: Cross-Validated (3 fold)
Summary of sample sizes: 55, 55, 56
Resampling results:

  Accuracy      Kappa
  0.9638447972  0.9500100827

Tuning parameter 'C' was held constant at a value of 1
```

El modelo obtenido con el algoritmo de SVM lineal con 3-fold crossvalidation tiene una precisión de 0.96 y un valor $\kappa$ de 0.95.

Finalmente, podemos decir que el modelo obtenido con la función 'svmLinear' y 3-fold crossvalidation obtiene mayor precisión que el modelo SVM de función lineal con partición train/test .

Además, el modelo obtenido con la función 'svmLinear' y 3-fold crossvalidation tiene mayor valor de kappa que el modelo SVM de función lineal con partición train/test .

# Discusión

Para el problema de clasificación de 4 tipos de tumores usando valores de expresión génica se han usando dos de los algoritmos más comunes de *Machine Learning* : las redes neuronales artificiales (ANN) y las máquinas de vectores de soporte (SVM). Ambos tienen un gran poder de clasificación pero son cajas negras para poder realizar una interpretación del clasificador obtenido.

En la siguiente tabla se resumen los diferentes modelos obtenidos con su valor de precisión y kappa como medidas del rendimiento del algoritmo para los datos usados.

| **Algoritmo** | **Normalización** | **kernel** | **Parámetro** | **3-fold Cross-validation** | **Precisión** | **Kappa** |
|---|---|---|---|---|---|---|
| ANN | `normalize` | - | hidden = 1 | NO | 0.46 | 0.32 |
| ANN | `normalize` | - | hidden = 3 | NO | 0.89 | 0.85 |
| ANN | `normalize` | - | hidden = 5 | SI | 0.95 | 0.94 |
| SVM | - | lineal | C = 1 | NO | 0.93 | 0.9 |
| SVM | - | gaussiano | C = 1 | NO | 0.71 | 0.61 |
| SVM | - | lineal | C = 1 | SI | 0.96 | 0.95 |

Como vemos en la tabla, los dos modelos (ANN y SVM) con 3-fold crossvalidation obtienen los mejores resultados, con unos valores de precisión de 0.95 y 0.96, y un estadístico Kappa de 0.94 y 0.95 respectivamente.

Un punto importante a considerar es que los dos algoritmos se han entrenado con dos data sets diferentes. El algoritmo SVM se entrenó con los datos de expresión génica obtenida del análisis de microarrays usando 2308 genes. En cambio, el algoritmo de ANN se entrenó con las 10 primeras componentes principales que explicaban un 63 % de la varianza original. Este hecho puede influir en el rendimiento menor de los modelos de ANN respecto a los de SVM.

En conclusión, el algoritmo que mejor clasifica los diferentes tumores SRBCTs de los modelos estudiados, con una precisión de 0.96 y un coeficiente $\kappa$ de 0.95, es el modelo entrenado por ***Support Vector Machine*** con *kernel* lineal, parámetro C = 1 y 3-fold Crossvalidation.

# Referencias


Khan, Javed, Jun S Wei, Markus Ringner, Lao H Saal, Marc Ladanyi, Frank Westermann, Frank Berthold, et al. 2001. "Classification and Diagnostic Prediction of Cancers Using Gene Expression Profiling and Artificial Neural Networks." *Nature Medicine* 7 (6). Nature Publishing Group: 673–79.

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