# PEC2: Gene expression patterns of phenotypes subclasses

Artificial neural networks & support vector machines

Escribir vuestro nombre y apellidos

28 de noviembre, 2020

## Índice

**Classification of phenotypes using gene expression profiling.** **2**

**Algoritmo Red Neuronal Artificial (ANN)** **2**
Step 1 - Recoger los datos . . . 2
Step 2 - Exploración y preparación de los datos . . . 3
Step 3 - Entrenar el modelo con los datos . . . 6
Step 4 - Evaluación del rendimiento del algoritmo . . . 8
Step 5 - Mejora del rendimiento del algoritmo . . . 9
3-fold crossvalidation . . . 11

**Algoritmo Support Vector Machine (SVM)** **15**
Step 1 - Recoger los datos . . . 15
Step 2 - Exploración y preparación de los datos . . . 16
Step 3 - Entrenar el modelo con los datos . . . 17
Step 4 - Evaluación del rendimiento del algoritmo . . . 17
Step 5 - Mejora del rendimiento del algoritmo . . . 18
3-fold crossvalidation . . . 19

**Discusión** **20**

**Referencias** **21**

# Classification of phenotypes using gene expression profiling.

En esta PEC vamos a realizar un informe que analiza un experimento relacionado con la clasificación de 4 tipos de fenotipos:

*1: FNT1 2: FNT2 3: FNT2 4: FNT4*

El objetivo es implementar una red neuronal artificial y un "support vector machine" (SVM) para predecir los cuatro tipos de fenotipos.

Como el algoritmo de red neuronal artificial es muy costoso si el número de variables es alto, se opta por realizar un análisis de componentes principales para reducir la dimensión de las variables iniciales y usar solo las 8 primeras en el algoritmo.

El análisis de componentes principales (PCA, en ingles) es una técnica básica y común en análisis multivariante para reducir el número de variables. Se basa en crear nuevas variables, denominadas componentes principales, como combinación lineal de las originales buscando máximizar la varianza explicada. Como no sé si sabeis realizar PCA en R, tambíen se dispone del fichero "pcaComponents6.csv" resultado del PCA donde las observaciones son representadas con las componentes principales.

En cambio, el algoritmo "support vector machine" admite un número muy elevado de variables sin un incremento sustancial en su coste computacional. Por tanto, se puede utilizar los datos originales.

# 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 información 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.

```
mydata0 <- read.csv(file=file.path(params$fold,params$file1_ANN))
clase <- read.csv(file=file.path(params$fold,params$file2))
nvar <- params$nvar
```

El primer conjunto de datos denominado *pcaComponents6.csv* esta formado por 102 muestras. Es el resultado de haber realizado el análisis de componentes principales (PCA) sobre los datos originales.

El segundo conjunto de datos denominado *class6.csv* corresponde a la clase de fenotipos de los anteriores datos.

Como variables solo se van a escoger las *8* primeras variables del PCA o componentes principales.

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

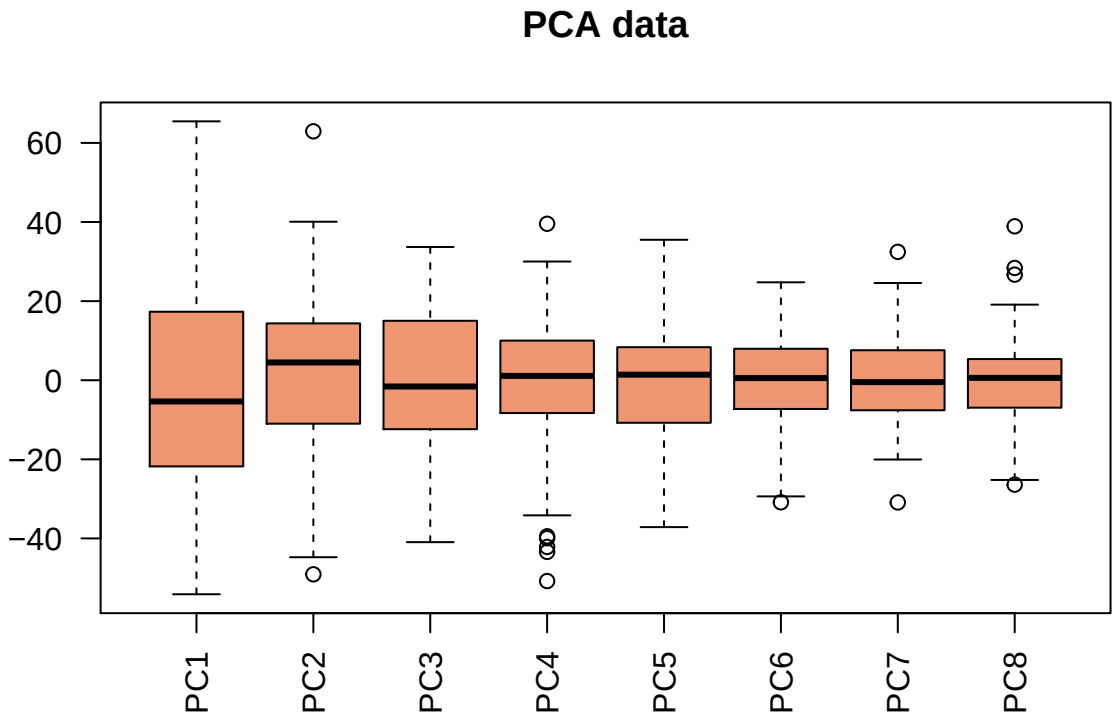
## Step 2 - Exploración y preparación de los datos

En primer lugar veremos los seis primeros registros:

```
         PC1      PC2        PC3       PC4        PC5         PC6        PC7        PC8
1 -40.210185 19.23744  13.885727 10.110266   6.682348 -13.5694975   1.016854  -4.264229
2 -36.921936 14.37209  -4.903639 26.651474  30.377865 -30.8737713  10.202086 -21.975164
3 -27.590151 40.08005  -1.768691 19.150609   3.200707   1.8964900  -9.377989 -13.317018
4 -19.419953 32.17562  16.248369 17.559142 -11.279857   0.1597929 -17.438327  -7.291898
5   1.390606 11.47379  21.910047  9.583781 -16.907643   7.3933244 -30.899032   1.526249
6 -46.621655 62.94365 -11.829080 -1.732755  -8.525425  -4.0475086   9.122549  10.486402
```

Un exploración gráfica mediante boxplot da:

```
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.

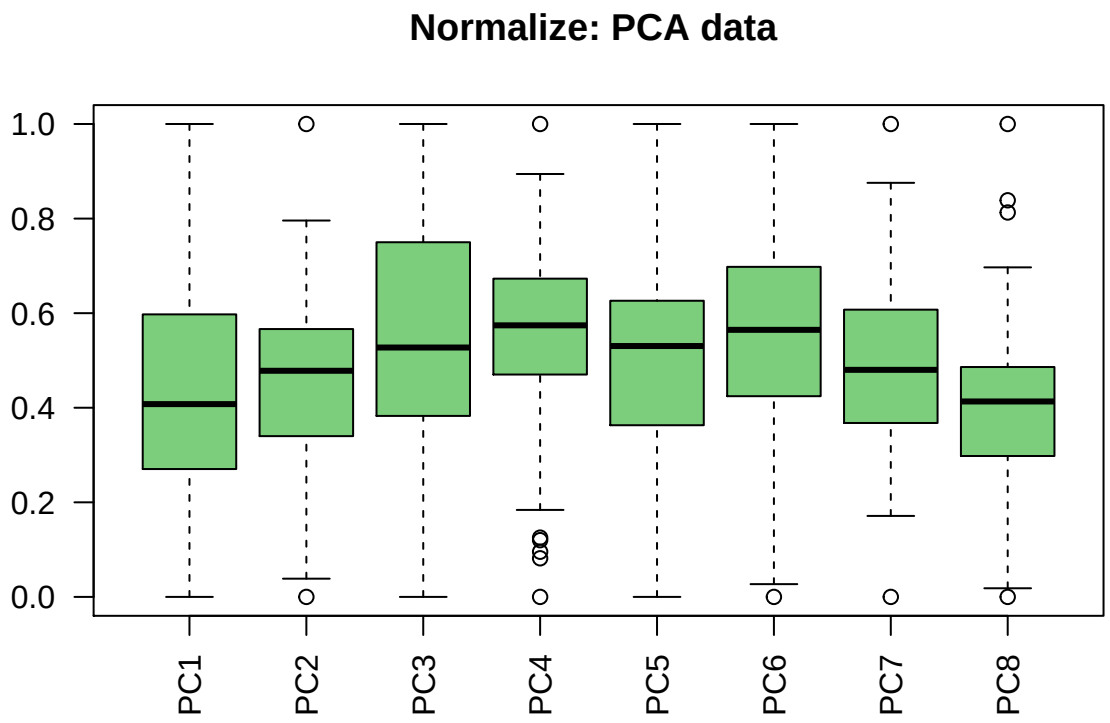
```
# custom normalization function
normalize <- function(x) {
return((x - min(x)) / (max(x) - min(x)))
```

```
}

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.:0.2745   1st Qu.:0.3405   1st Qu.:0.3828   1st Qu.:0.4750   1st Qu.:0.3651  
 Median :0.4077   Median :0.4782   Median :0.5274   Median :0.5743   Median :0.5305  
 Mean   :0.4526   Mean   :0.4381   Mean   :0.5486   Mean   :0.5621   Mean   :0.5112  
 3rd Qu.:0.5830   3rd Qu.:0.5655   3rd Qu.:0.7498   3rd Qu.:0.6723   3rd Qu.:0.6260  
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000  
      PC6               PC7               PC8         
 Min.   :0.0000   Min.   :0.0000   Min.   :0.0000  
 1st Qu.:0.4249   1st Qu.:0.3683   1st Qu.:0.3022  
 Median :0.5647   Median :0.4802   Median :0.4133  
 Mean   :0.5550   Mean   :0.4877   Mean   :0.4042  
 3rd Qu.:0.6979   3rd Qu.:0.6028   3rd Qu.:0.4852  
 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 fenotipos registrada en cada muestra. Mediante una tabla se muestra el número de clases de cada tipo:

```
table(clase)
```

```
clase
 1  2  3  4 
```

```
25 26 28 23
```

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

```
lab.group <- c("FNT1","FNT2","FNT3","FNT4")
clase.f <- factor(clase$x,labels=lab.group)
```

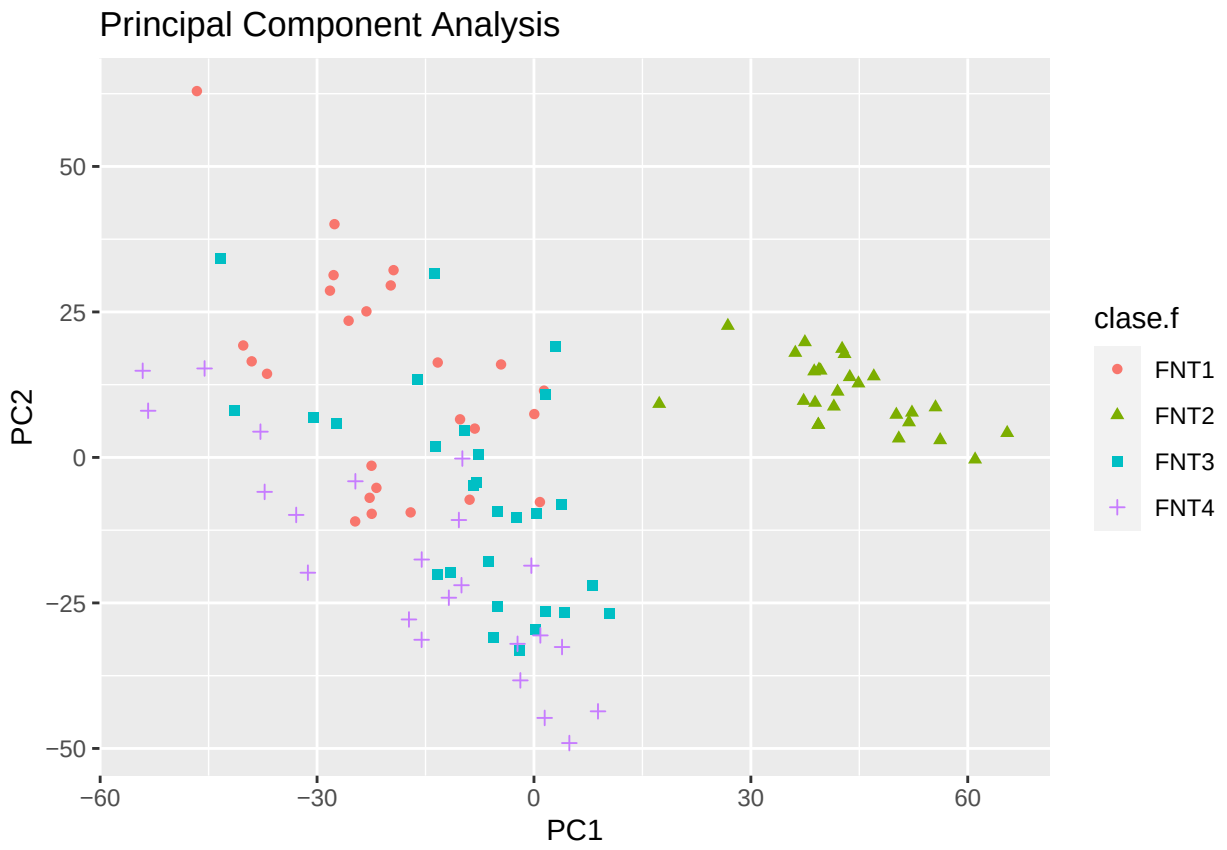
Ahora la tabla queda como:

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

```
clase.f
FNT1 FNT2 FNT3 FNT4
  25   26   28   23
```

Para ver la disposición de las observaciones según la clase se representa en un diagrama de puntos las dos primeras componentes principales

```
ggplot(mydata, aes(x= PC1, y= PC2, shape=clase.f, color=clase.f, )) +
  labs(title="Principal Component Analysis")+
  geom_point()
```




Se observa como el fenotipo FNT2 queda claramente diferenciado respecto del resto de fenotipos en la primera componente principal.

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 fenotipo.

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

for (i in 1:length(lab.group)){
  mydata_ann[,nvar+i]<- clase.f==lab.group[i]
```

```
}
names(mydata_ann)[(nvar+1):(nvar+length(lab.group))] <- lab.group
names(mydata_ann)
```

```
 [1] "PC1"  "PC2"  "PC3"  "PC4"  "PC5"  "PC6"  "PC7"  "PC8"  "FNT1" "FNT2" "FNT3" "FNT4"
```

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.

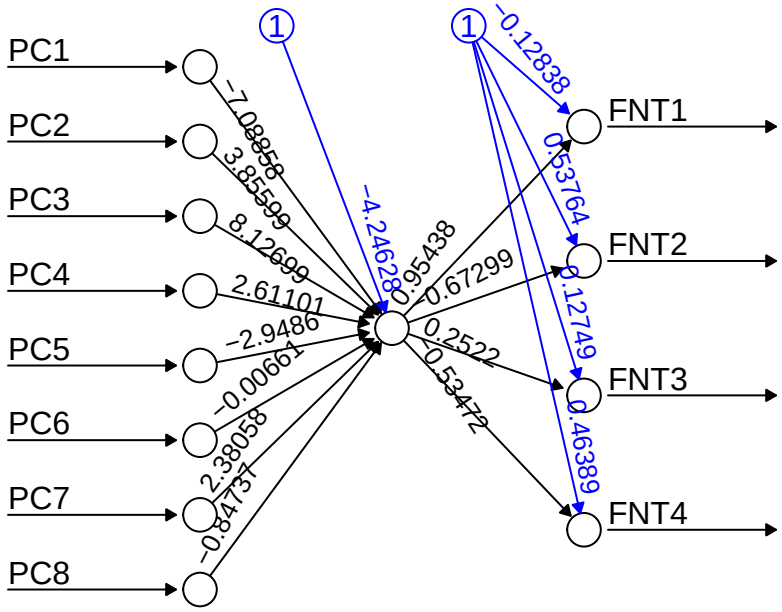
```
############### libraries loading
#require(neuralnet)

## Create a formula for a model with a large number of variables:
xnam <- names(mydata_ann[1:nvar])
dep_var <-paste(lab.group, collapse=" + ")
(fmla <- as.formula(paste(dep_var, " ~ ",  paste(xnam, collapse= "+"))))
```

```
FNT1 + FNT2 + FNT3 + FNT4 ~ PC1 + PC2 + PC3 + PC4 + PC5 + PC6 +
    PC7 + PC8
```

```
# 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)
```

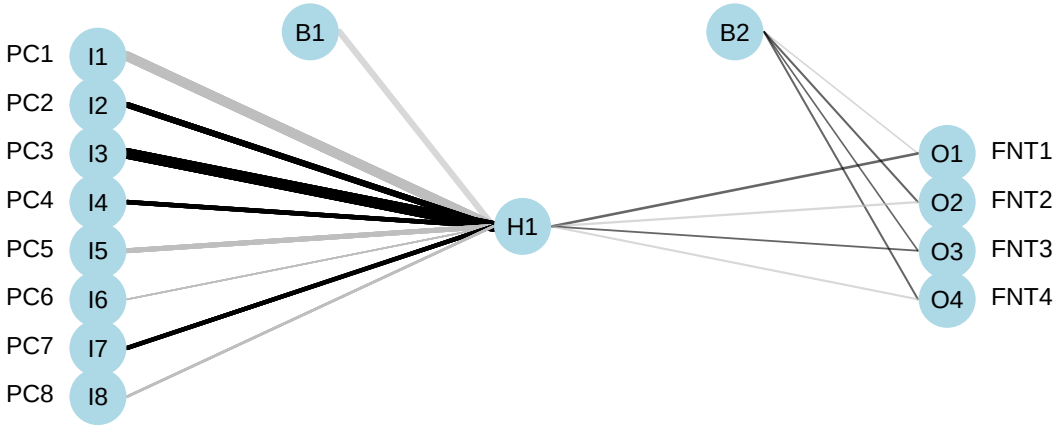
La representación de la red neuronal artificial es:

```
#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:nvar])$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=1:length(lab.group),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 FNT1 FNT2 FNT3 FNT4
      FNT1    6    0   10    0
      FNT2    0    9    2    7
      FNT3    0    0    0    0
      FNT4    0    0    0    0

Overall Statistics

               Accuracy : 0.4412
                 95% CI : (0.2719, 0.6211)
    No Information Rate : 0.3529
    P-Value [Acc > NIR] : 0.1839

                  Kappa : 0.2806

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: FNT1 Class: FNT2 Class: FNT3 Class: FNT4
Sensitivity               1.0000      1.0000      0.0000      0.0000
Specificity               0.6429      0.6400      1.0000      1.0000
Pos Pred Value            0.3750      0.5000         NaN         NaN
Neg Pred Value            1.0000      1.0000      0.6471      0.7941
Prevalence                0.1765      0.2647      0.3529      0.2059
Detection Rate            0.1765      0.2647      0.0000      0.0000
Detection Prevalence      0.4706      0.5294      0.0000      0.0000
Balanced Accuracy         0.8214      0.8200      0.5000      0.5000
```

El modelo de ANN de una capa oculta con un nodo consigue una precisión de 0.44 y un estadístico kappa ($\kappa$) de 0.28. Los valores de sensibilidad y especificidad varían según el tipo de fenotipo, obteniendo como valor medio 0.5 y 0.82 respectivamente.

Observar que un ANN de una capa oculta con un nodo solo puede asignar valores a dos clases. Por tanto, solo es

adecuado para clasificación binaria.

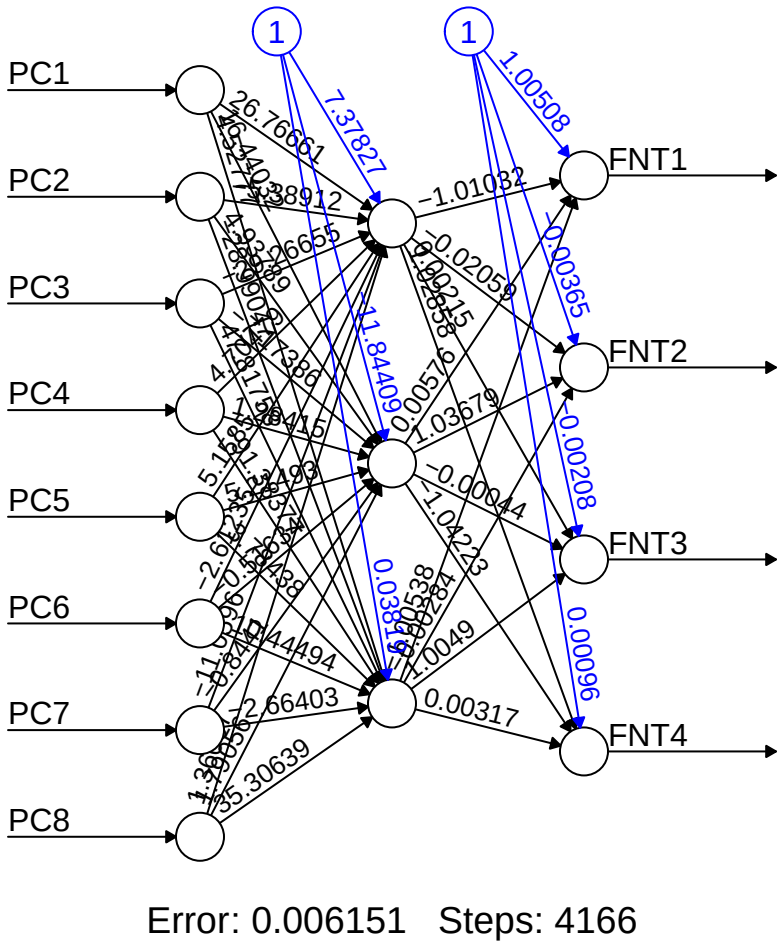
## Step 5 - Mejora del rendimiento del algoritmo

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 3 hidden neurons
set.seed(params$seed.clsfier) # to guarantee repeatable results
mydata_model2 <- neuralnet(fmla,
                           data = mydata_ann.train,
                           linear.output = TRUE,
                           hidden=3)
```

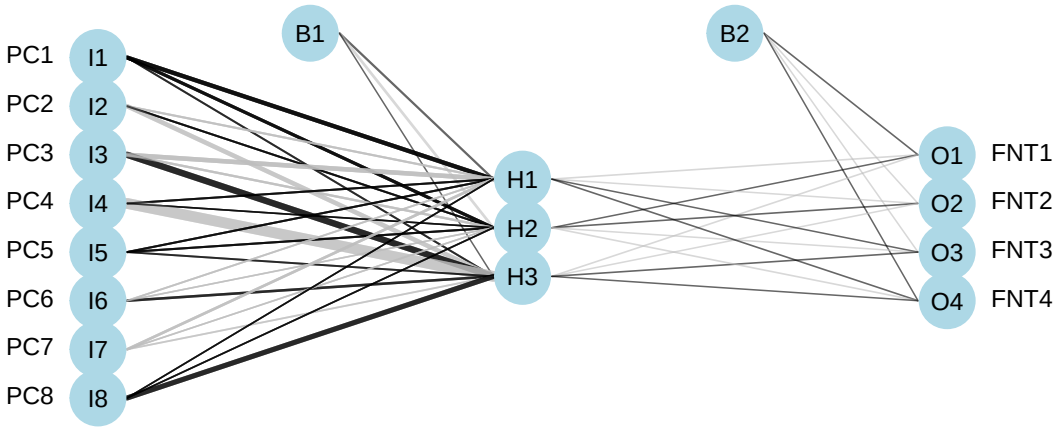
La representación de la red neuronal artificial es:

```
# 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:nvar])$net.result

idx <- apply(model_results2, 1, maxidx)
prediction2 <- factor(idx,levels=1:length(lab.group),labels=lab.group )

res <- table(prediction2, clase.f[-train])

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

```
Confusion Matrix and Statistics


prediction2 FNT1 FNT2 FNT3 FNT4
       FNT1    6    0    1    0
       FNT2    0    9    0    0
       FNT3    0    0   11    0
       FNT4    0    0    0    7

Overall Statistics

               Accuracy : 0.9706
                 95% CI : (0.8467, 0.9993)
    No Information Rate : 0.3529
    P-Value [Acc > NIR] : 2.652e-14

                  Kappa : 0.9601

 Mcnemar's Test P-Value : NA

Statistics by Class:
```

```
                     Class: FNT1 Class: FNT2 Class: FNT3 Class: FNT4
Sensitivity               1.0000      1.0000      0.9167      1.0000
Specificity               0.9643      1.0000      1.0000      1.0000
Pos Pred Value            0.8571      1.0000      1.0000      1.0000
Neg Pred Value            1.0000      1.0000      0.9565      1.0000
Prevalence                0.1765      0.2647      0.3529      0.2059
Detection Rate            0.1765      0.2647      0.3235      0.2059
Detection Prevalence      0.2059      0.2647      0.3235      0.2059
Balanced Accuracy         0.9821      1.0000      0.9583      1.0000
```

El nuevo modelo de ANN con tres nodos en la capa oculta obtiene un valor de precisión de 0.97 y un estadístico $\kappa = 0.96$. Los valores de sensibilidad y especificidad varían según el tipo de fenotipo, obteniendo como valor medio 0.98 y 0.99 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
```

En el modelo `nnet` de caret tiene dos hiper-parámetros a ajustar: `size` número de nodos en la capa oculta y `decay` es el parámetro de regularización para evitar el sobreajuste, por defecto 0.

Por tanto, el modelo de entrenamiento queda como:

```
###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= data.frame(size=3,decay=0),  trace = FALSE)
```

El modelo obtenido es:

```
model
```

```
Neural Network

102 samples
  8 predictor
  4 classes: 'FNT1', 'FNT2', 'FNT3', 'FNT4'

No pre-processing
Resampling: Cross-Validated (3 fold)
Summary of sample sizes: 66, 69, 69
Resampling results:

  Accuracy   Kappa
  0.8914141  0.8551081

Tuning parameter 'size' was held constant at a value of 3
Tuning parameter 'decay'
 was held constant at a value of 0
```

Sabiendo que con caret es muy facil entrenar y validar el modelo sobre una variedad de valores de los hiper-parámetros se plantea un nuevo código que explora para tres valores diferentes de `size` y `decay`.

```
###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)
```

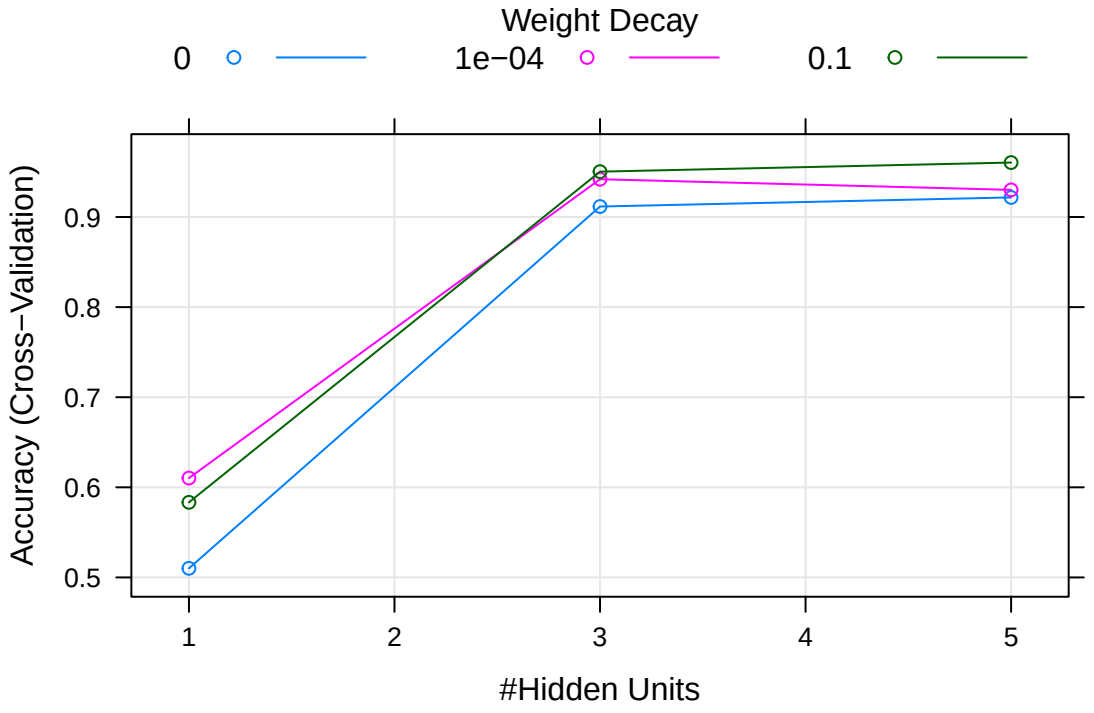
El modelo obtenido es:

```
model
```

```
Neural Network

102 samples
  8 predictor
  4 classes: 'FNT1', 'FNT2', 'FNT3', 'FNT4'

No pre-processing
Resampling: Cross-Validated (3 fold)
Summary of sample sizes: 66, 69, 69
Resampling results across tuning parameters:

  size  decay  Accuracy   Kappa
  1     0e+00  0.5101010  0.3386484
  1     1e-04  0.6102694  0.4727328
  1     1e-01  0.5833333  0.4396434
  3     0e+00  0.9116162  0.8818271
  3     1e-04  0.9419192  0.9225151
  3     1e-01  0.9503367  0.9336207
  5     0e+00  0.9217172  0.8951308
  5     1e-04  0.9301347  0.9066343
  5     1e-01  0.9604377  0.9471508

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.
```

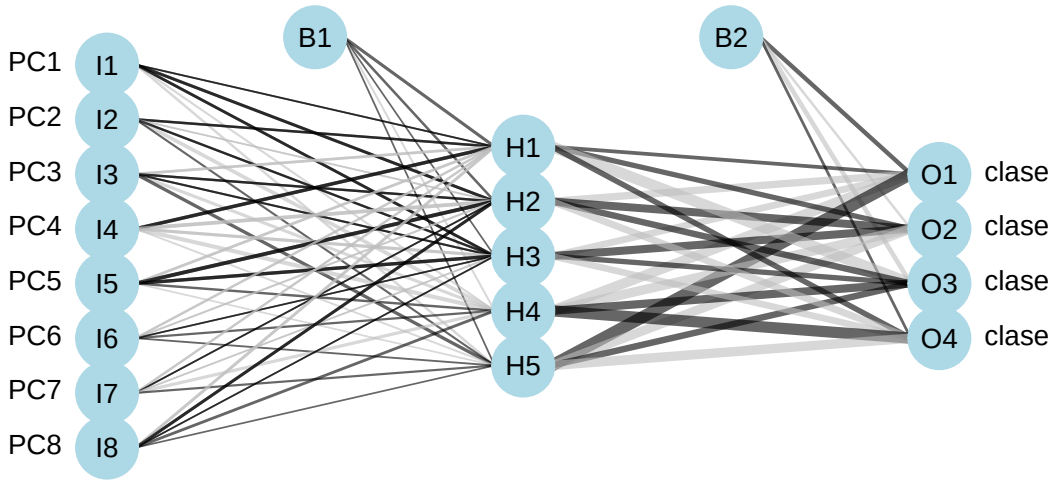
El gráfico del rendimiento según diferente parámetros es

```
# 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 8-5-4 network with 69 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
  0.36   0.02   0.19  -0.23   0.44  -0.24  -0.16  -0.14  -0.29
 b->h2 i1->h2 i2->h2 i3->h2 i4->h2 i5->h2 i6->h2 i7->h2 i8->h2
  0.21   0.30   0.00   0.17  -0.48   0.51  -0.05   0.02   0.32
 b->h3 i1->h3 i2->h3 i3->h3 i4->h3 i5->h3 i6->h3 i7->h3 i8->h3
  0.00   0.27   0.16   0.09  -0.33   0.41   0.00  -0.01   0.00
 b->h4 i1->h4 i2->h4 i3->h4 i4->h4 i5->h4 i6->h4 i7->h4 i8->h4
 -0.09  -0.11  -0.50  -0.35  -0.60   0.18   0.13  -0.37   0.29
 b->h5 i1->h5 i2->h5 i3->h5 i4->h5 i5->h5 i6->h5 i7->h5 i8->h5
  0.02  -0.19   0.11   0.41  -0.05  -0.01   0.03   0.12   0.00
 b->o1 h1->o1 h2->o1 h3->o1 h4->o1 h5->o1
  0.67   0.51  -1.40  -1.26  -2.09   2.60
 b->o2 h1->o2 h2->o2 h3->o2 h4->o2 h5->o2
 -0.20   0.88   1.56   1.59  -1.91  -1.86
 b->o3 h1->o3 h2->o3 h3->o3 h4->o3 h5->o3
 -0.74  -2.34   1.17   0.89   1.62   1.18
 b->o4 h1->o4 h2->o4 h3->o4 h4->o4 h5->o4
  0.26   0.95  -1.32  -1.22   2.38  -1.92
```

En resumen, el mejor modelo de ANN con 3-fold crossvalidation de los parámetros explorados se obtiene con 5 nodos ocultos y un valor de *decay* de 0.1, con el se obtiene una precisión media de 0.96 y un estadístico $\kappa$ igual a 0.95.

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

## 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 8 primeras componentes principales como variables explicativas.

```
#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]  102 5506
```

El primer conjunto de datos denominado *data6.csv* esta formado por 102 muestras y tiene la expresión génica de 5506 genes.

El segundo conjunto de datos denominado *class6.csv* corresponde a la clase de fenotipo de los anteriores datos.

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

```
# Add class variable
lab.group <- c("FNT1","FNT2","FNT3","FNT4")
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:

```
       V5499     V5500       V5501       V5502       V5503      V5504      V5505
1  0.4711272 0.3121510  0.29548946 -0.01193568  0.69130083  0.6002394 -1.7187721
2 -0.3639832 0.1138226  1.05042187  0.45779508 -0.60871617  0.5458863 -1.7187721
3  0.9188849 0.3851066  1.03214296 -1.79633480 -0.57590954  0.4658861 -1.7187721
4  0.7732686 0.2901426  0.82106903  0.45955169  0.07131779  0.3982556 -0.0148425
5  0.3933221 0.2795571 -0.07527715  0.08712988  0.03036379 -0.3731539  1.1924094
6  0.5536640 0.6899715  1.07252685  0.10908812 -0.33546815  0.7789699 -1.7187721
         V5506 clase
1 -0.503736032  FNT1
2 -0.217283216  FNT1
3 -0.094941968  FNT1
4  0.001528351  FNT1
5 -0.912673289  FNT1
6  0.083572768  FNT1
```

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[1:6, (ncol(mydata)-8):ncol(mydata)])
```

```
'data.frame':   6 obs. of  9 variables:
 $ V5499: num  0.471 -0.364 0.919 0.773 0.393 ...
 $ V5500: num  0.312 0.114 0.385 0.29 0.28 ...
 $ V5501: num  0.2955 1.0504 1.0321 0.8211 -0.0753 ...
 $ V5502: num  -0.0119 0.4578 -1.7963 0.4596 0.0871 ...
 $ V5503: num  0.6913 -0.6087 -0.5759 0.0713 0.0304 ...
 $ V5504: num  0.6 0.546 0.466 0.398 -0.373 ...
 $ V5505: num  -1.7188 -1.7188 -1.7188 -0.0148 1.1924 ...
 $ V5506: num  -0.50374 -0.21728 -0.09494 0.00153 -0.91267 ...
 $ clase: Factor w/ 4 levels "FNT1","FNT2",..: 1 1 1 1 1 1
```

Una breve estadistica descriptiva de las 9 anteriores variables es:

```
summary(mydata[,(ncol(mydata)-8):ncol(mydata)])
```

```
     V5499               V5500               V5501              V5502         
 Min.   :-1.383241   Min.   :-0.662318   Min.   :-1.36616   Min.   :-1.79633  
 1st Qu.:-0.367210   1st Qu.:-0.169697   1st Qu.:-0.31477   1st Qu.:-0.33078  
 Median : 0.027569   Median : 0.007803   Median :-0.01284   Median : 0.06259  
 Mean   :-0.003661   Mean   :-0.008391   Mean   :-0.01081   Mean   :-0.01367  
 3rd Qu.: 0.357335   3rd Qu.: 0.180352   3rd Qu.: 0.34892   3rd Qu.: 0.43174  
 Max.   : 1.162727   Max.   : 0.913376   Max.   : 1.07253   Max.   : 1.24004  
     V5503               V5504               V5505              V5506          
 Min.   :-1.26052   Min.   :-1.20489   Min.   :-1.71877   Min.   :-0.9126733  
 1st Qu.:-0.34054   1st Qu.:-0.23695   1st Qu.:-1.57920   1st Qu.:-0.1932040  
 Median : 0.03495   Median :-0.03242   Median : 0.47271   Median : 0.0318996  
 Mean   : 0.01236   Mean   :-0.01481   Mean   : 0.01685   Mean   :-0.0005242  
 3rd Qu.: 0.41836   3rd Qu.: 0.26056   3rd Qu.: 1.06217   3rd Qu.: 0.2141462  
 Max.   : 1.23903   Max.   : 0.77897   Max.   : 2.04460   Max.   : 0.5264223  
      clase    
 FNT1:25  
 FNT2:26  
 FNT3:28  
 FNT4:23  
```

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 : 58

Objective Function Value : -6e-04 -0.0014 -0.001 -6e-04 -5e-04 -0.001
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 FNT1 FNT2 FNT3 FNT4
           FNT1    6    0    0    0
           FNT2    0    9    0    0
           FNT3    0    0   11    0
           FNT4    0    0    1    7

Overall Statistics

               Accuracy : 0.9706
                 95% CI : (0.8467, 0.9993)
    No Information Rate : 0.3529
    P-Value [Acc > NIR] : 2.652e-14
```

```
                 Kappa : 0.96

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: FNT1 Class: FNT2 Class: FNT3 Class: FNT4
Sensitivity               1.0000      1.0000      0.9167      1.0000
Specificity               1.0000      1.0000      1.0000      0.9630
Pos Pred Value            1.0000      1.0000      1.0000      0.8750
Neg Pred Value            1.0000      1.0000      0.9565      1.0000
Prevalence                0.1765      0.2647      0.3529      0.2059
Detection Rate            0.1765      0.2647      0.3235      0.2059
Detection Prevalence      0.1765      0.2647      0.3235      0.2353
Balanced Accuracy         1.0000      1.0000      0.9583      0.9815
```

El algoritmo de SVM lineal tiene un valor de precisión de 0.97 y un estadístico $\kappa = 0.96$. Vemos que los valores de sensibilidad y especificidad varían según el factor, obteniendo como valor medio 0.98 y 0.99 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 =  8.67067839624197e-05

Number of Support Vectors : 62

Objective Function Value : -6.1195 -13.8653 -10.6871 -6.4468 -5.5085 -10.9498
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 FNT1 FNT2 FNT3 FNT4
           FNT1    6    0    1    0
           FNT2    0    9    0    0
           FNT3    0    0   11    0
           FNT4    0    0    0    7

Overall Statistics

               Accuracy : 0.9706
                 95% CI : (0.8467, 0.9993)
    No Information Rate : 0.3529
    P-Value [Acc > NIR] : 2.652e-14

                  Kappa : 0.9601

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: FNT1 Class: FNT2 Class: FNT3 Class: FNT4
Sensitivity               1.0000      1.0000      0.9167      1.0000
Specificity               0.9643      1.0000      1.0000      1.0000
Pos Pred Value            0.8571      1.0000      1.0000      1.0000
Neg Pred Value            1.0000      1.0000      0.9565      1.0000
Prevalence                0.1765      0.2647      0.3529      0.2059
Detection Rate            0.1765      0.2647      0.3235      0.2059
Detection Prevalence      0.2059      0.2647      0.3235      0.2059
Balanced Accuracy         0.9821      1.0000      0.9583      1.0000
```

El algoritmo de SVM de función RBF tiene un valor de precisión de 0.97 y un estadístico $\kappa = 0.96$. Como se puede esperar, los valores de sensibilidad y especificidad varían según la clase, obteniendo como valor medio 0.98 y 0.99 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 tiene mayor valor de kappa que el modelo más sencillo de SVM con 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
```

```
 102 samples
5506 predictors
   4 classes: 'FNT1', 'FNT2', 'FNT3', 'FNT4'

No pre-processing
Resampling: Cross-Validated (3 fold)
Summary of sample sizes: 66, 69, 69
Resampling results:

  Accuracy   Kappa
  0.9806397  0.9741574

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 0.98 y un valor $\kappa$ de 0.97.

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 fenotipos 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 Crossvalidation** | **Precisión** | **Kappa** |
|---|---|---|---|---|---|---|
| ANN | `normalize` | - | hidden = 1 | NO | 0.44 | 0.28 |
| ANN | `normalize` | - | hidden = 3 | NO | 0.97 | 0.96 |
| ANN | `normalize` | - | hidden = 5 | SI | 0.96 | 0.95 |
| SVM | - | lineal | C = 1 | NO | 0.97 | 0.96 |
| SVM | - | gaussiano | C = 1 | NO | 0.97 | 0.96 |
| SVM | - | lineal | C = 1 | SI | 0.98 | 0.97 |

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 0.96 y 0.98, y un estadístico kappa de 0.95 y 0.97, 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 5507 genes. En cambio, el algoritmo de ANN se entrenó con las 8 primeras componentes principales. Este hecho puede influir en un rendimiento diferente de los modelos de ANN respecto a los de SVM. En este caso, va un poco mejor el modelo SVM.

En conclusión, los resultados en ambos algoritmos son similares, aunque el algoritmo un poco mejor es el modelo entrenado por ***SVM*** lineal y 3-fold crossvalidation.

# Referencias

Kuhn, Max. 2017. *A Short Introduction to the caret Package*. https://cran.r-project.org/web/packages/caret/vignettes/caret.pdf.

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