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

Cristina Lendinez Gonzalez

24 de mayo, 2021

Índice general

```
library(knitr)
library(tinytex)
```

1 Algoritmo Red Neuronal Artificial (ANN)

Las redes neuronales artificiales se asemejan a las redes neuronales que posee el cerebro. Las neuronas son remplazadas por nodos que se encargan de recibir y enviar señales (información). Se crea un red con diferentes capas interconectadas para procesar la información. Cada capa esta formada por un grupo de nodos que transmite la información a los nodos de la capa siguiente.

La caracteristicas de la red neuronal artificial son:

- la topología: Esto corresponde a la cantidad de capas y nodos. Tiene en cuenta la dirreción en la que se transmite la informacion de un nodo al siguiente, bien dentro de las capas o entre capas
- La función de activación: Gracias a esta función se reciben un conjunto de entradas e integras las señales para transmitir la información a otro nodo/capa.
- El algoritmo de entrenamiento: Estable la importancia de cada conexion para decididr si debe transmitir la señal a los nodos correspondientes. El algoritmo mas usado es el "backpropagation" que esta basado en que para corregir los errores de prediccion va hacia atras de la red corrigiendo los pesos de los nodos.

Las fortalezas y debilidades del algoritmo son los siguientes:

Fortalezas	**Debilidades**
- Adaptable a clasificación o problemas de predicción númerica. - Capaz de modelas patronas más complejos que casi cualquie otro algoritmo - No necesita muchas restricciones acerca de las relaciones subyacentes de los datos.	- Propenso a sobreajustar los datos de entrenamiento. - Es un modelo de caja negra complejo que es dificil, si no imposible, de interpretar. - Requiere de gran potencia computacional y en general es de aprendizaje lento, particularmente si la topologia es compleja

1.1 Step 1 - Descarga y lectura de los datos

Descargare los archivos csv, para poder emplezar el análisis, voy a cargar los archivos PCA, así me garantizo que estará bien hecho el ejercicio

```
PCA <- read.csv("../Documentos PEC Cris/pcaComponents7 (5).csv")
clases <- read.csv("../Documentos PEC Cris/class7 (4).csv")
```

Cargo los datos para ver si me da tiempo hacer el pca(hare el pca pero tirare de los datos que nos da el profesor).

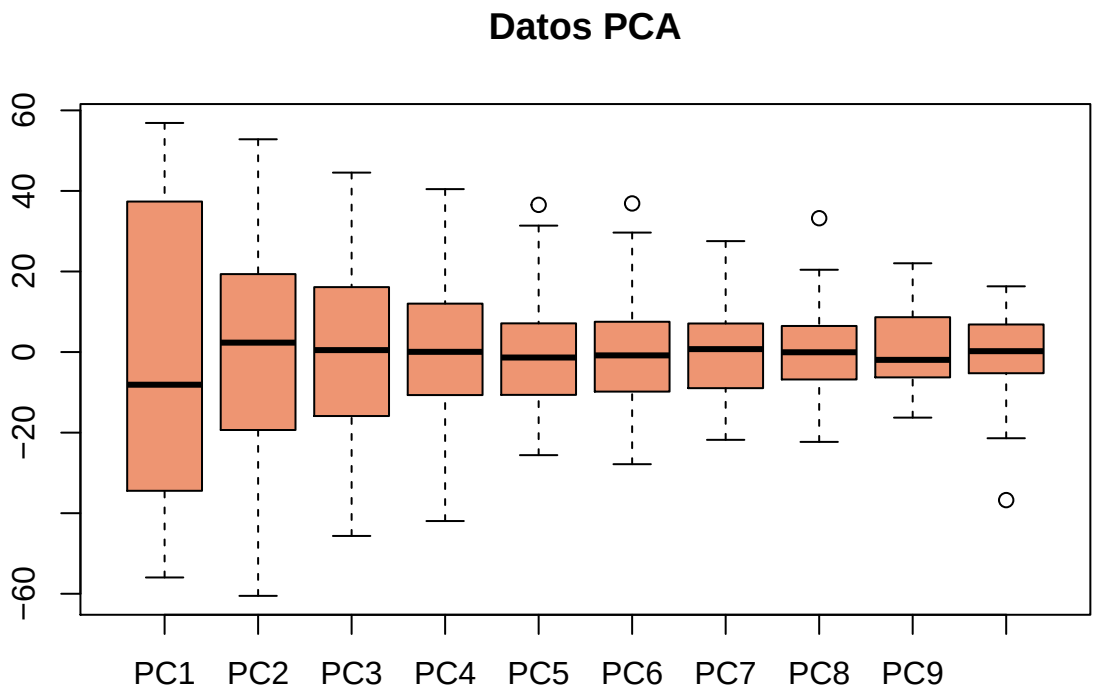
```
datos <- read.csv("./data7 (5).csv")
```

Como solo tengo que coger 10 datos, voy a selecionar las 10 primeras columnas del dataset PCA

```r
PCA_10 <- PCA[,1:10]
```

Voy a hacer una exploracion de los datos que he selecionado del dataframe creado PCA_10

```r
boxplot(PCA_10, main="Datos PCA", col = "lightsalmon2")
```



Voy a ver cuantas observaciones tengo

```r
dim(PCA_10)
```

```
## [1] 60 10
```

```r
str(PCA_10)
```

```
## 'data.frame':    60 obs. of  10 variables:
##  $ PC1 : num  -39.2 -12 -30.1 -24.3 -31.9 ...
##  $ PC2 : num  33.3 4.9 -35.2 -56 -11.8 ...
##  $ PC3 : num  -28.72 -45.65 -20.05 -3.06 -21.11 ...
##  $ PC4 : num  17.27 -41.95 -6.11 -11.52 -4.39 ...
##  $ PC5 : num  -25.59 1.59 -16.19 -9.16 -18.18 ...
##  $ PC6 : num  2.17 15.32 -10.02 -2.84 -2.47 ...
##  $ PC7 : num  -21.78 -20.82 -2.41 -1.01 17.13 ...
##  $ PC8 : num  20.4349 -4.4144 -2.8768 -0.0699 5.9137 ...
##  $ PC9 : num  -16.2 15.12 -2.51 11.83 4.87 ...
##  $ PC10: num  10.27 -7.83 -3.47 -21.39 16.32 ...
```

1.2 Step 2. Normalizar las variables.

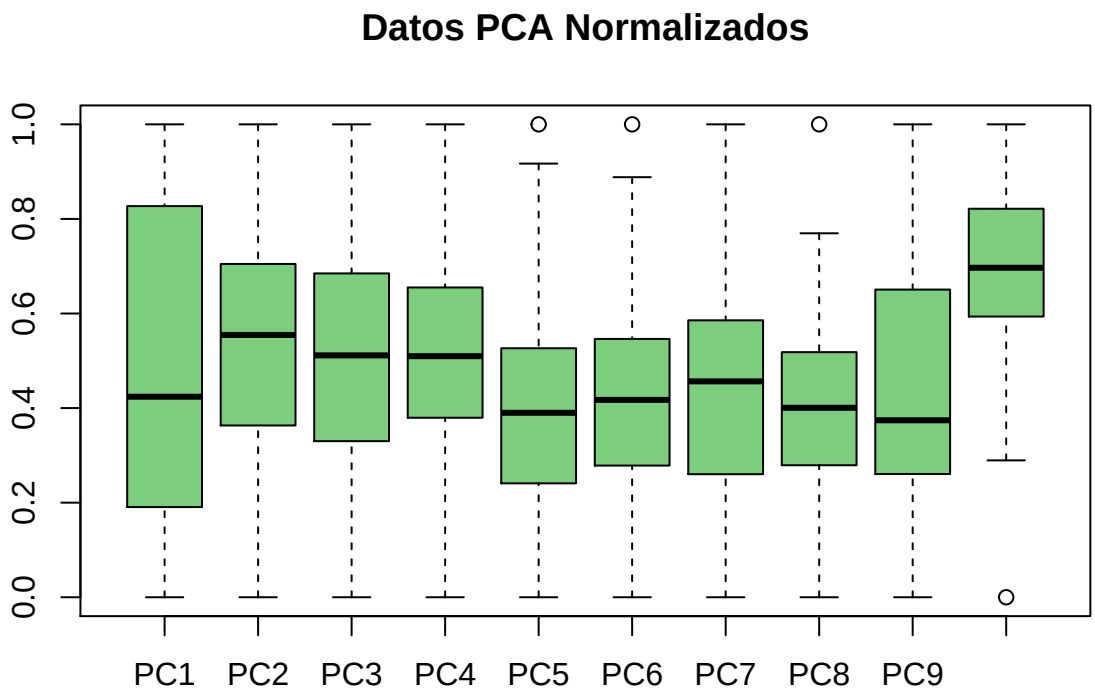
Voy a normalizar las variables para que los valores esten entre 0 y 1. Para ello generaremos una función con el nombre de `normalizar` y despues se realizara otro boxplot para observar la diferencia.

```
normal = function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}
```

```
PCA_Normalizados = as.data.frame(lapply(PCA_10, normal))
summary(PCA_Normalizados)
```

```
##       PC1              PC2              PC3              PC4
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.1990   1st Qu.:0.3651   1st Qu.:0.3379   1st Qu.:0.3821
##  Median :0.4241   Median :0.5546   Median :0.5114   Median :0.5098
##  Mean   :0.4959   Mean   :0.5340   Mean   :0.5061   Mean   :0.5091
##  3rd Qu.:0.8227   3rd Qu.:0.6878   3rd Qu.:0.6821   3rd Qu.:0.6527
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##       PC5              PC6              PC7              PC8
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.2413   1st Qu.:0.2800   1st Qu.:0.2623   1st Qu.:0.2800
##  Median :0.3901   Median :0.4173   Median :0.4565   Median :0.4007
##  Mean   :0.4119   Mean   :0.4300   Mean   :0.4417   Mean   :0.4016
##  3rd Qu.:0.5224   3rd Qu.:0.5438   3rd Qu.:0.5754   3rd Qu.:0.5162
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##       PC9              PC10
##  Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.2621   1st Qu.:0.5957
##  Median :0.3742   Median :0.6967
##  Mean   :0.4248   Mean   :0.6925
##  3rd Qu.:0.6414   3rd Qu.:0.8139
##  Max.   :1.0000   Max.   :1.0000
```

```
boxplot(PCA_Normalizados, main="Datos PCA Normalizados", col = "palegreen3")
```



1.3 Poner las etiquetas

Los distintos fenotipos están registrados de manera numérica, así que creamos las etiquetas que se indican en el enunciado y se las ponemos a cada una de las distintas clases:

```
labels = c("ALL","AML","CLL","CML", "NoL")
clases.label<-factor(clases$x,labels=labels)
```

Y la tabla finalmente queda asi:

```
table(clases.label)
```

```
## clases.label
## ALL AML CLL CML NoL
##  12  12  12  12  12
```

Creamos un dataframe que contiene las etiquetas, aqui podemos ver que tenemos los PCA con sus fenotipos correspondientes:

```
ANNdatos = PCA_Normalizados
ANNdatos$ALL = clases.label=="ALL"
ANNdatos$AML = clases.label=="AML"
ANNdatos$CLL = clases.label=="CLL"
ANNdatos$CML = clases.label=="CML"
```

```
ANNdatos$NoL = clases.label=="NoL"

# Verificamos que se han añadido correctamente
names(ANNdatos)
```

```
##  [1] "PC1"  "PC2"  "PC3"  "PC4"  "PC5"  "PC6"  "PC7"  "PC8"  "PC9"  "PC10"
## [11] "ALL"  "AML"  "CLL"  "CML"  "NoL"
```

1.4 Step 3 - Entrenamiento del modelo con los datos

Ahora se va a entrenar el primer modelo con los datos de entrenamiento. Para ello, se utilizara la función `neuralnet` que esta en el paquete `neuralnet`. Este modelo tendra un nodo oculto.

```
set.seed(12345)
prueba_training=floor(0.67*nrow(ANNdatos))
train=sample(seq_len(nrow(ANNdatos)),size=prueba_training)
ANNtraining=ANNdatos[train,]
ANNtest=ANNdatos[-train,]
dim(ANNtraining)
```

```
## [1] 40 15
```

```
dim(ANNtest)
```

```
## [1] 20 15
```

```
require(neuralnet)
```

```
## Loading required package: neuralnet
```

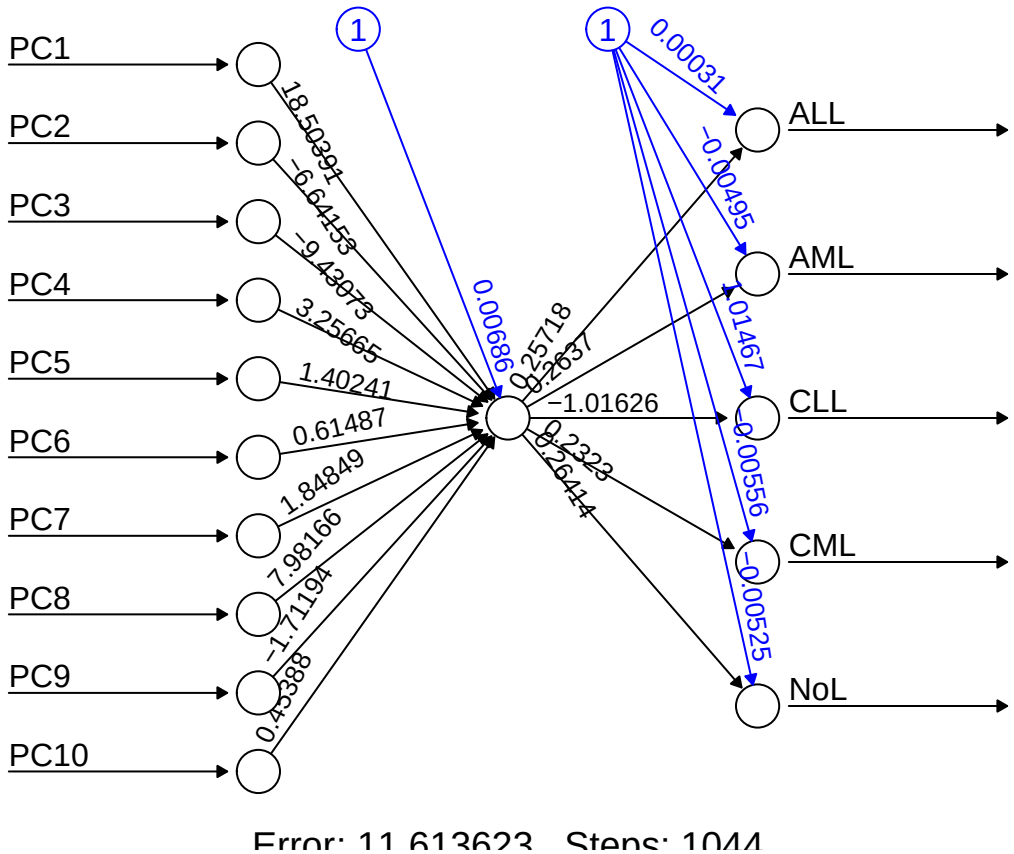
```
xnam = names(ANNdatos[1:10])
(fmla = as.formula(paste("ALL+AML+CLL+CML+NoL ~ ", paste(xnam,collapse = "+"))))
```

```
## ALL + AML + CLL + CML + NoL ~ PC1 + PC2 + PC3 + PC4 + PC5 + PC6 +
##     PC7 + PC8 + PC9 + PC10
```

```
set.seed(1234567)
ANNmodelo = neuralnet(fmla,data=ANNtraining,hidden = 1)
```

Ahora se representara el modelo con la función `plot`, queremos graficar para ver como se comporta el modelos

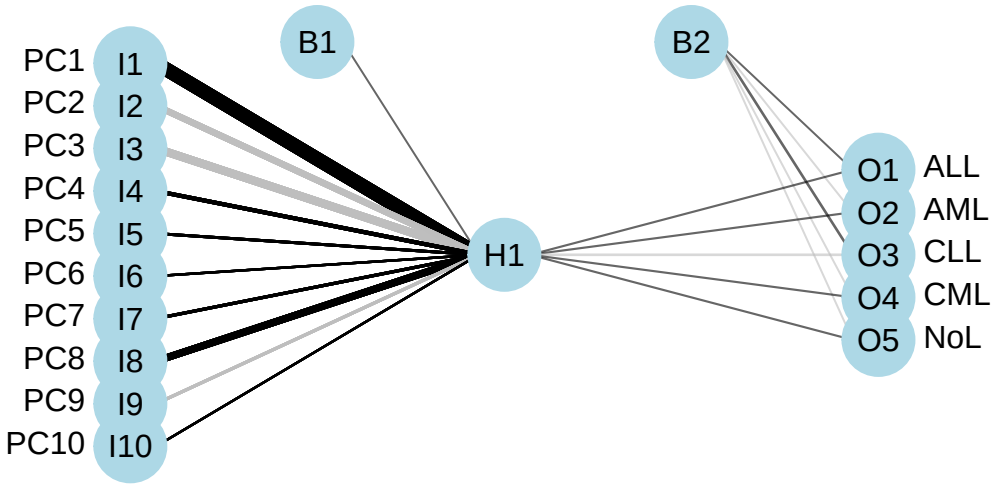
```
plot(ANNmodelo,rep="best")
```




```
# ANN representation
require(NeuralNetTools)
```

```
## Loading required package: NeuralNetTools
```

```
plotnet (ANNmodelo, alpha=0.6)
```



1.5 Step 4 - Evaluación de la ejecución del modelo

Una vez obtenido el modelo, se evalua su rendimiento con los datos de test. Para ello se utilizara la función `compute`.

```
ANNresultado= compute(ANNmodelo,ANNtest[1:10])$net.result
maxidx=function(arr) {
  return(which(arr == max(arr)))}

idx= apply(ANNresultado,1,maxidx)
prediction=factor(idx,levels = c(1,2,3,4,5), labels = labels)
res= table(prediction, clases.label[-train])
```

Despues, se obtiene la matriz de confusion con las predicciones y las clases reales. para ello se utiliza la funcion `confusionMatrix` del paquete `caret`.

```
require(caret)
```

```
## Loading required package: caret
```

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

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

```
(confusion_matrix<-confusionMatrix(res))
```

```
## Confusion Matrix and Statistics
##
##
## prediction ALL AML CLL CML NoL
##        ALL   0   0   0   0   0
##        AML   0   0   0   0   0
##        CLL   0   0   3   0   0
##        CML   0   0   0   0   0
##        NoL   4   4   0   5   4
##
## Overall Statistics
##
##                Accuracy : 0.35
##                  95% CI : (0.1539, 0.5922)
##     No Information Rate : 0.25
##     P-Value [Acc > NIR] : 0.2142
##
##                   Kappa : 0.195
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: ALL Class: AML Class: CLL Class: CML Class: NoL
## Sensitivity                 0.0        0.0       1.00       0.00     1.0000
## Specificity                 1.0        1.0       1.00       1.00     0.1875
## Pos Pred Value              NaN        NaN       1.00        NaN     0.2353
## Neg Pred Value              0.8        0.8       1.00       0.75     1.0000
## Prevalence                  0.2        0.2       0.15       0.25     0.2000
## Detection Rate              0.0        0.0       0.15       0.00     0.2000
## Detection Prevalence        0.0        0.0       0.15       0.00     0.8500
## Balanced Accuracy           0.5        0.5       1.00       0.50     0.5938
```

Obtengo un Accuracy de 0.35 y un KAPPA de 0.195, son valores bajos es un valor bajo, y estos dayos son debidos ANN se utiliza para problemas de clasificacion binaria.

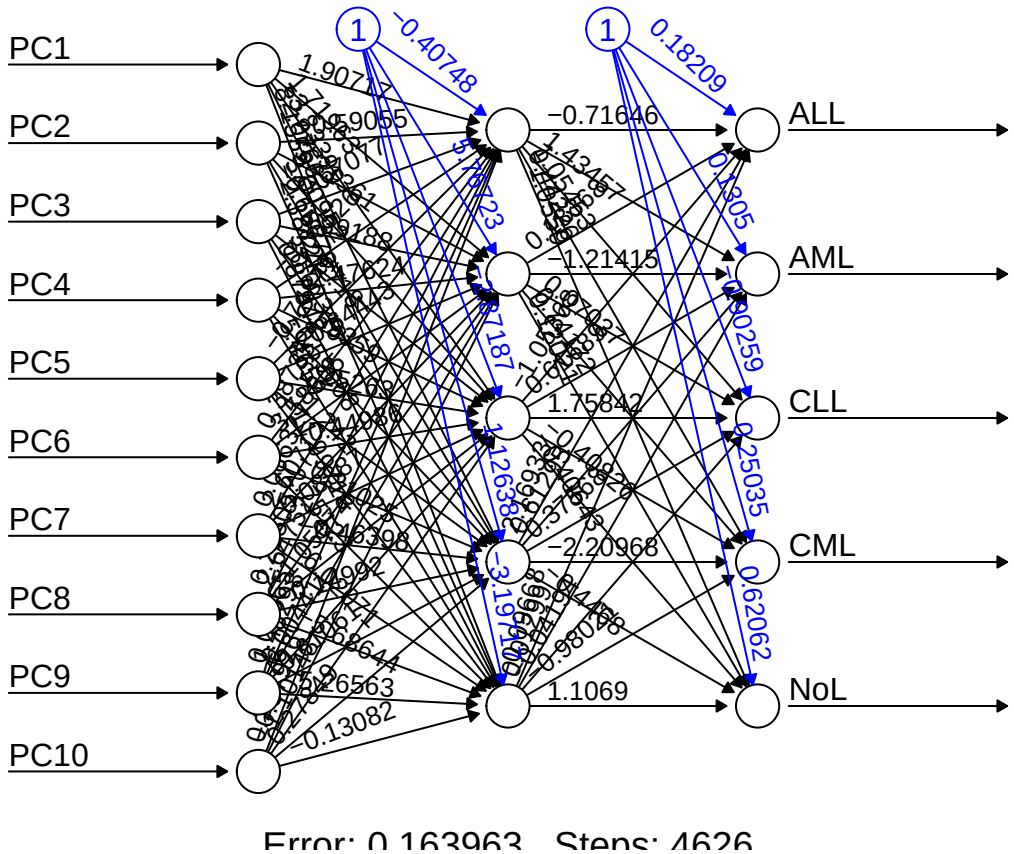
1.6 Step 5 - Mejora de la ejecución del modelo

El primer modelo tiene un nodo en la capa oculta. Para mejorar el modelo y el rendimiento utilizare un modelo con 3 nodos ocultos en la capa oculta

```
set.seed(1234567)
ANNmodelo_5=neuralnet(fmla,data=ANNtraining,linear.output = TRUE, hidden=5)
```

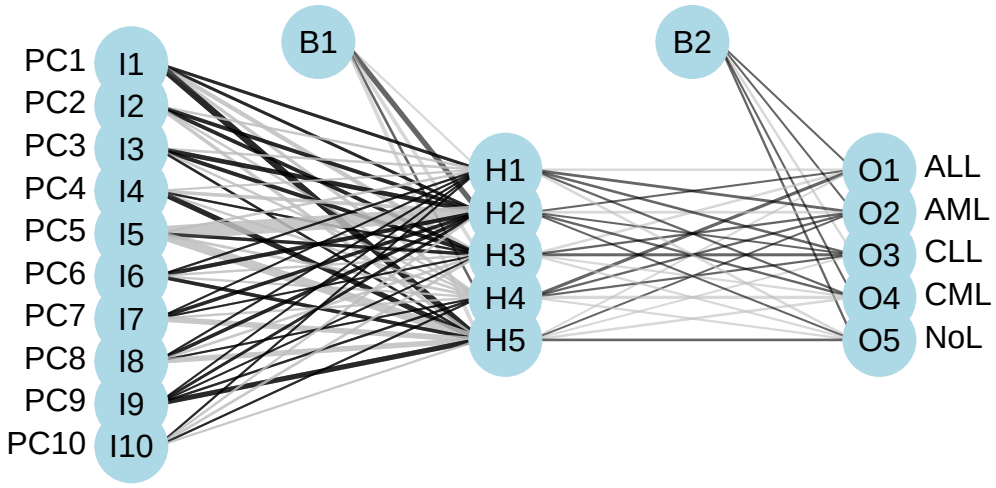
Represento el modelo con 3 nodos ocultos

```
plot(ANNmodelo_5,rep = "best")
```



Mediante la función `plotnet` represento el grafico

```
plotnet(ANNmodelo_5,alpha=0.6)
```



Hago la matriz de confusión es:

```
ANNresultado_5=compute(ANNmodelo_5, ANNtest[1:10])$net.result
idx=apply(ANNresultado_5,1,maxidx)
prediction_5=factor(idx,levels=c(1,2,3,4,5),labels = labels)
res5=table(prediction_5,clases.label[-train])
(confusion_matrix<-confusionMatrix(res5))
```

```
## Confusion Matrix and Statistics
##
##
## prediction_5 ALL AML CLL CML NoL
##          ALL   4   0   0   0   0
##          AML   0   3   0   0   0
##          CLL   0   0   3   0   0
##          CML   0   0   0   5   0
##          NoL   0   1   0   0   4
##
## Overall Statistics
##
##                Accuracy : 0.95
##                  95% CI : (0.7513, 0.9987)
##     No Information Rate : 0.25
##     P-Value [Acc > NIR] : 5.548e-11
##
##                   Kappa : 0.9371
```

```
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: ALL Class: AML Class: CLL Class: CML Class: NoL
## Sensitivity                 1.0     0.7500       1.00       1.00     1.0000
## Specificity                 1.0     1.0000       1.00       1.00     0.9375
## Pos Pred Value              1.0     1.0000       1.00       1.00     0.8000
## Neg Pred Value              1.0     0.9412       1.00       1.00     1.0000
## Prevalence                  0.2     0.2000       0.15       0.25     0.2000
## Detection Rate              0.2     0.1500       0.15       0.25     0.2000
## Detection Prevalence        0.2     0.1500       0.15       0.25     0.2500
## Balanced Accuracy           1.0     0.8750       1.00       1.00     0.9688
```

Al hacer la matriz de confusion con 3 nodos he obtenido un Accuracy de 0.95 y un valor Kappa 0.9371, la sensibilidad y mejora del modelo ha mejorado bastante.

1.6.1 3-fold crossvalidation

Voy a usar el paquete aret, paa poder hacer el modelo con los 5 nodos.en la capa oculta usare el 3-fold crossvalidation.

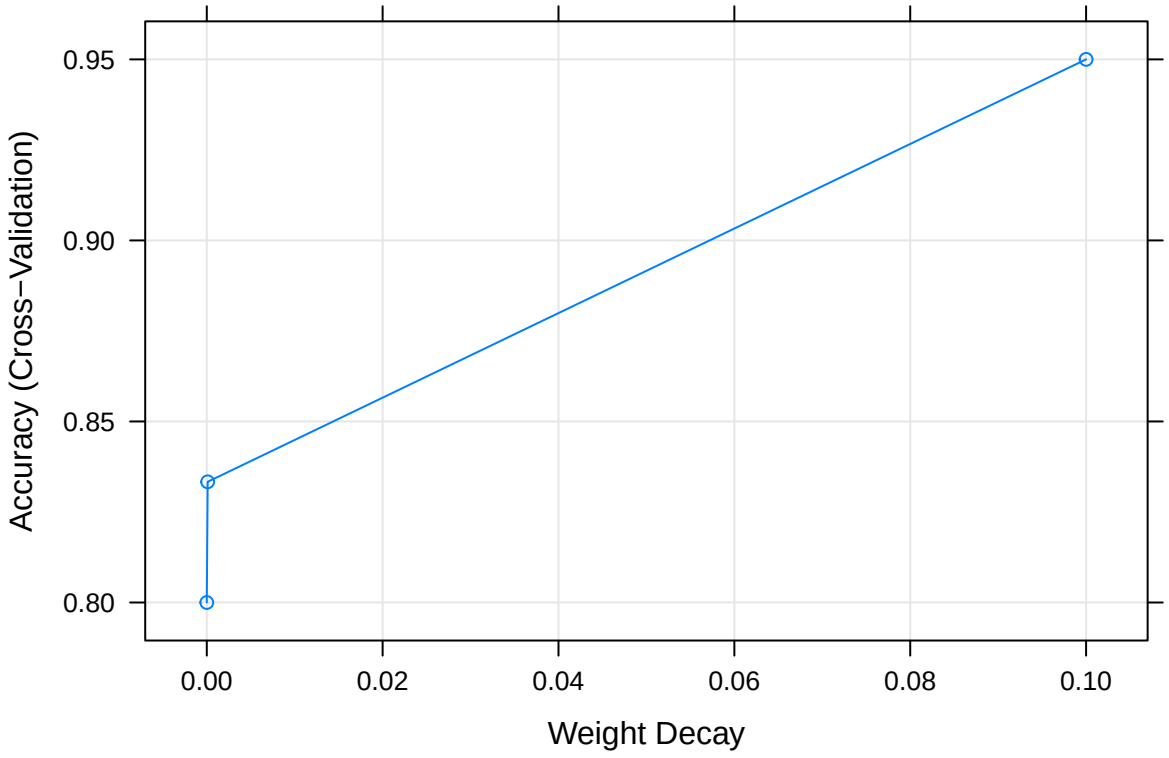
```
caretData<-PCA_10
caretData$clase<-clases.label
```

Voy a hacer el crossvalidation con el método `nnet`, considerando que `size` indica el número de nodos en la capa oculta y `decay` para poder controlar el ajunte usare este parametro de regularizacion:

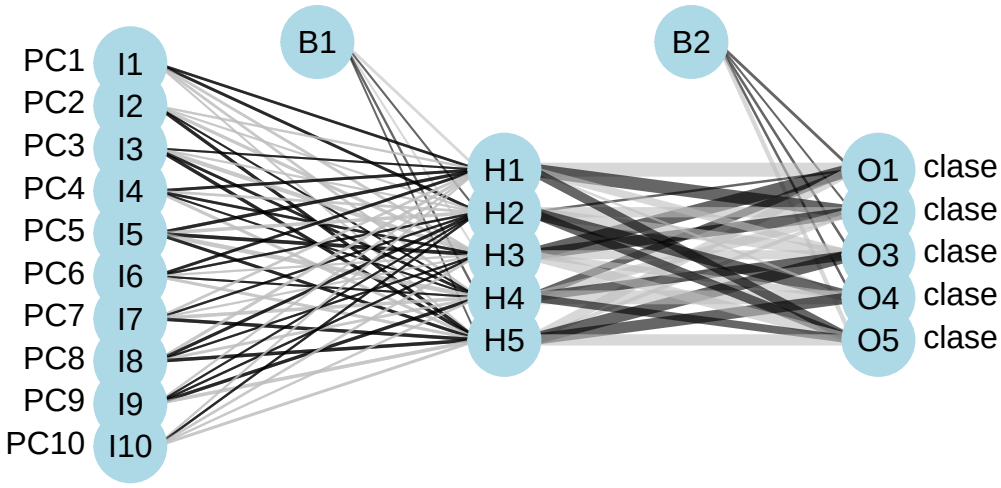
```
set.seed(1234567)
model_cv<-train(clase ~ ., caretData, method="nnet",
                trControl= trainControl(method="cv",number=3),
                tuneGrid=data.frame(size=5,decay=c(0,0.0001,0.1)), trace= FALSE)
```

Hago varios graficos

```
plot(model_cv, rep=best)
```




```
require(NeuralNetTools)
plotnet(model_cv,alpha=0.6)
```




```
model_cv
```

```
## Neural Network
##
## 60 samples
## 10 predictors
##  5 classes: 'ALL', 'AML', 'CLL', 'CML', 'NoL'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 40, 40, 40
## Resampling results across tuning parameters:
##
##   decay  Accuracy   Kappa
##   0e+00  0.8000000  0.7500000
##   1e-04  0.8333333  0.7916667
##   1e-01  0.9500000  0.9375000
##
## Tuning parameter 'size' was held constant at a value of 5
## 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.
```

```
summary(model_cv)
```

```
## a 10-5-5 network with 85 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
##   -0.18    0.14   -0.07    0.02    0.19    0.26    0.19   -0.13   -0.23   -0.06
## i10->h1
##   -0.04
##   b->h2  i1->h2  i2->h2  i3->h2  i4->h2  i5->h2  i6->h2  i7->h2  i8->h2  i9->h2
##    0.03    0.18   -0.02   -0.03   -0.13   -0.29    0.08    0.07    0.23    0.04
## i10->h2
##    0.08
##   b->h3  i1->h3  i2->h3  i3->h3  i4->h3  i5->h3  i6->h3  i7->h3  i8->h3  i9->h3
##   -0.03   -0.15   -0.25   -0.44    0.16    0.33   -0.04   -0.04    0.06    0.08
## i10->h3
##   -0.12
##   b->h4  i1->h4  i2->h4  i3->h4  i4->h4  i5->h4  i6->h4  i7->h4  i8->h4  i9->h4
##    0.08   -0.14    0.01    0.18    0.02   -0.47    0.03   -0.29   -0.16    0.27
## i10->h4
##   -0.06
##   b->h5  i1->h5  i2->h5  i3->h5  i4->h5  i5->h5  i6->h5  i7->h5  i8->h5  i9->h5
##    0.00   -0.01    0.25    0.29   -0.22    0.22   -0.20    0.29    0.29   -0.30
## i10->h5
##   -0.18
##  b->o1 h1->o1 h2->o1 h3->o1 h4->o1 h5->o1
##   0.19  -2.40   0.10   2.24   1.17  -1.57
##  b->o2 h1->o2 h2->o2 h3->o2 h4->o2 h5->o2
##   0.03   2.20  -1.65   1.83  -1.38  -0.38
##  b->o3 h1->o3 h2->o3 h3->o3 h4->o3 h5->o3
##   0.13  -1.14  -1.77  -1.42   1.44   1.97
##  b->o4 h1->o4 h2->o4 h3->o4 h4->o4 h5->o4
##   0.12  -0.10   1.91  -0.93  -2.38   1.82
##  b->o5 h1->o5 h2->o5 h3->o5 h4->o5 h5->o5
##  -0.47   1.44   1.41  -1.72   1.15  -1.85
```

Al obtener los resultados el mejor valor de `decay`de 0.1, obteniendo una precisión = 0.95 y una índice κ = 0.9375. Los resultados que tengo con este modelo son casi iguales a los obtenidos con el algoritmo ANN y 5 nodos en la capa oculta.

2 Algoritmo Support Vector Machine (SVM)

Las maquinas 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.

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

Las aplicaciones mas utilizadas por el algoritmo son:

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

Cuando los datos no se pueden separar de forma lineal el uso de kernels es necesario.Los kernels más populares son el lineal y el gausiano, aunque existen otros como el polinomial, string kernel, chi-square kernel, etc.

Fortalezas	**Debilidades**
- Se puede usar para problemas de clasificación o predicción numérica - Funciona bastante bien con datos ruidosos y no es muy propenso al overfitting - Puede llegar a ser mas facil de usar que las redes neuronales (ANN) Debido a la existencia de varios algoritmos SVM bien soportados - 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 a base de prueba y error - A medida que aumenta el numero de caracteristicas, es lento de entrenar - Los resultados del modelo son difícil, si no es imposible, de interpretar (caja negra)

2.1 Step 1 - Descarga y lectura de los datos

En el ejercicio del algoritmo SVM usare todos los datos de expresión génica originales, no como en el caso del ANN que se limito a los 10 primeros componentes principales , y en mi caso use los datos del PCA que nos dio el profesor.

```
datasvm <- read.csv("./data7 (5).csv")
clases <- read.csv("./class7 (4).csv")
dim(datasvm)
```

```
## [1]   60 5043
```

2.2 Step 2 - Exploración y preparación de los datos

Voy a poner las etiquetasm, que puse en el segundo ejercicio

```
datasvm$clases <- clases.label
```

Voy a ver las primeras observaciones hare un summary de las primeras 10 observaciones, ya que si hago un summary de todo puede ser una autentica locura, ya que son 5043 variables

```
summary(datasvm[,1:10])
```

```
##  ENSG00000000457 ENSG00000000460 ENSG00000000938  ENSG00000001036
##  Min.   :4.290   Min.   :2.973   Min.   : 4.644   Min.   :3.833
##  1st Qu.:5.100   1st Qu.:3.632   1st Qu.: 8.145   1st Qu.:5.836
##  Median :5.469   Median :4.030   Median :10.313   Median :7.241
##  Mean   :5.574   Mean   :4.106   Mean   : 9.527   Mean   :6.914
##  3rd Qu.:6.027   3rd Qu.:4.446   3rd Qu.:11.068   3rd Qu.:8.097
##  Max.   :6.982   Max.   :5.724   Max.   :11.629   Max.   :9.163
##  ENSG00000001084 ENSG00000001561 ENSG00000001629 ENSG00000001630
##  Min.   :4.917   Min.   :3.565   Min.   :4.808   Min.   :5.241
##  1st Qu.:6.794   1st Qu.:5.946   1st Qu.:6.108   1st Qu.:6.548
##  Median :7.701   Median :6.761   Median :6.932   Median :7.334
##  Mean   :7.595   Mean   :6.665   Mean   :6.741   Mean   :7.235
##  3rd Qu.:8.440   3rd Qu.:7.794   3rd Qu.:7.405   3rd Qu.:7.854
##  Max.   :9.599   Max.   :9.204   Max.   :8.454   Max.   :9.403
##  ENSG00000001631 ENSG00000002549
##  Min.   :3.700   Min.   :6.208
##  1st Qu.:4.226   1st Qu.:8.133
##  Median :4.777   Median :8.515
##  Mean   :4.853   Mean   :8.559
##  3rd Qu.:5.457   3rd Qu.:9.147
##  Max.   :6.741   Max.   :9.822
```

Voy a separar las muestras en entrenamiento y test. antes lo hice en el ejercicio 2 pero con los datos del PCA, esta separacion la hare con los datos totales del data7.

```
datasvm.training <- datasvm[train,]
datasvm.test <- datasvm[-train,]
```

2.3 Step 3 - Entrenamiento del modelo con los datos

Se entrenara el modelo SVM lineal con los datos de entrenamiento. Para ello se utiliza la funcion `ksvm`del paquete `kernlab`.

```
library(kernlab)
```

```
##
## Attaching package: 'kernlab'

## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
set.seed(1234567)
svmmodel<-ksvm(clases ~ ., data=datasvm.training,kernel="vanilladot")
```

```
##  Setting default kernel parameters
```

```
svmmodel
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Linear (vanilla) kernel function.
##
## Number of Support Vectors : 36
##
## Objective Function Value : -8e-04 -5e-04 -3e-04 -4e-04 -4e-04 -8e-04 -0.001 -2e-04 -3e-04 -0.0015
## Training error : 0
```

2.4 Step 4 - Evaluacion de la ejecución del modelo

Una vez obtenido el modelo de SVM lineal, se evalua su rendimiento con los datos de test. Las muestras de los datos de test se clasificaran mediante la función `predict`.

```
require(caret)
svm_prediction<-predict(svmmodel,datasvm.test)
reslinear<-table(svm_prediction,datasvm.test$clase)
(confusion_matrix.svm<-confusionMatrix(reslinear))
```

```
## Confusion Matrix and Statistics
##
##
## svm_prediction ALL AML CLL CML NoL
##            ALL   4   0   0   0   0
##            AML   0   3   0   0   0
##            CLL   0   1   3   0   0
##            CML   0   0   0   4   0
##            NoL   0   0   0   1   4
##
## Overall Statistics
##
##                Accuracy : 0.9
##                  95% CI : (0.683, 0.9877)
##     No Information Rate : 0.25
##     P-Value [Acc > NIR] : 1.611e-09
##
##                   Kappa : 0.875
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: ALL Class: AML Class: CLL Class: CML Class: NoL
## Sensitivity                 1.0     0.7500     1.0000     0.8000     1.0000
## Specificity                 1.0     1.0000     0.9412     1.0000     0.9375
## Pos Pred Value              1.0     1.0000     0.7500     1.0000     0.8000
## Neg Pred Value              1.0     0.9412     1.0000     0.9375     1.0000
```

```
## Prevalence                0.2     0.2000     0.1500     0.2500     0.2000
## Detection Rate            0.2     0.1500     0.1500     0.2000     0.2000
## Detection Prevalence      0.2     0.1500     0.2000     0.2000     0.2500
## Balanced Accuracy         1.0     0.8750     0.9706     0.9000     0.9688
```

Obtengo un valor de precisión de 0.90 y un índice kappa de 0.875. Los valores de especificidad y sensibilidad son muy buenos.

2.5 Step 5 - Mejora de la ejecución del modelo

Voy a realizar un modelo SVM con función `gaussiana` o `rbf`.

```
set.seed(1234567)
modelo.rbf<-ksvm(clases ~., data= datasvm.training, kernel="rbfdot")
```

```
modelo.rbf
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
##  parameter : cost C = 1
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.000125817162266269
##
## Number of Support Vectors : 39
##
## Objective Function Value : -6.4625 -4.372 -3.0996 -3.2315 -3.8921 -4.913 -5.6159 -2.5418 -2.7277 -6.9
## Training error : 0
```

Con la función `compute` se evalua el rendimiento con los datos de test.

```
rbf.prediction<-predict(modelo.rbf, datasvm.test)
res.rbf<-table(rbf.prediction,datasvm.test$clase)
```

vuelvo a realizar la matriz de confusión para poder calcular el rendimiento del algortmo `rbf` con las predicciones y las clases reales.

```
(confirmar_matrix.rbf<- confusionMatrix(res.rbf))
```

```
## Confusion Matrix and Statistics
##
##
## rbf.prediction ALL AML CLL CML NoL
##            ALL   4   0   0   0   0
##            AML   0   3   0   0   0
##            CLL   0   1   3   0   0
##            CML   0   0   0   3   0
##            NoL   0   0   0   2   4
##
## Overall Statistics
```

```
##
##                Accuracy : 0.85
##                  95% CI : (0.6211, 0.9679)
##     No Information Rate : 0.25
##     P-Value [Acc > NIR] : 2.96e-08
##
##                   Kappa : 0.8131
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: ALL Class: AML Class: CLL Class: CML Class: NoL
## Sensitivity                 1.0     0.7500     1.0000     0.6000     1.0000
## Specificity                 1.0     1.0000     0.9412     1.0000     0.8750
## Pos Pred Value              1.0     1.0000     0.7500     1.0000     0.6667
## Neg Pred Value              1.0     0.9412     1.0000     0.8824     1.0000
## Prevalence                  0.2     0.2000     0.1500     0.2500     0.2000
## Detection Rate              0.2     0.1500     0.1500     0.1500     0.2000
## Detection Prevalence        0.2     0.1500     0.2000     0.1500     0.3000
## Balanced Accuracy           1.0     0.8750     0.9706     0.8000     0.9375
```

Acabo de obtener el algoritmo de SVM gaussiano tiene un valor de precisión de 0.85 y un índice kappa de 0.8131. he obtenido unos valores que son buenos, pero he visto que el valor es mejor el modelo anterior

2.5.1 3-fold crossvalidation

El ultimo apartado es el algoritmo SVM con la funcion lineal con 3-fold crossvalidation usare el paquete `caret`.

El modelo de entrenamiento es:

```
set.seed(1234567)
model.svm<-train(clases ~ ., datasvm, method="svmLinear",
                 trControl=trainControl(method="cv",number=3),
                 tuneGrid= NULL, trace= FALSE)
```

```
model.svm
```

```
## Support Vector Machines with Linear Kernel
##
##   60 samples
## 5043 predictors
##    5 classes: 'ALL', 'AML', 'CLL', 'CML', 'NoL'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 40, 40, 40
## Resampling results:
##
##   Accuracy   Kappa
##   0.9833333  0.9791667
##
## Tuning parameter 'C' was held constant at a value of 1
```

el resultado que he tenido con el algoritmo de SVM con 3-fold crossvalidation tiene una precisión = 0.98333 y un valor kappa = 0.9791667. Este ultimo modelo es, el que mejor rendimiento y mejor funciona de todos l los SVM testados.

3 Discusión final

Los datos que he obtenido con el algoritmo ANN, tanto el modelo de 5 nodos tanto en la capa oculta como el he realizado con la crossvalidation y utilizado un valor `decay`de 0.1 ha sido el que mejor resultado nos ha dado. Sin embargo, los datos que he obtenido con el algoritmo SVM, el mejor modelo es el que emplea la función lineal y 3-fold crossvalidation.

4 Referencias

He tenido problemas al importar las referencias que es el libro de machine learnig de <@lantz2015machine>