Breast

Oriol Planesas, Heribert Roig

10 de abril, 2022

```
library(keras)
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.1.2
## Loading required package: ggplot2
## Loading required package: lattice
```

```
library(pROC)
```

```
## Warning: package 'pROC' was built under R version 4.1.2
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

1. Describe protein abundance and gene expression datasets. How many patients have data of both types available. Are there missing data from some of the datasets? Preprocess them if necessary.

```
protein_abundance <- read.csv(params$file1, sep = "")
gene <- read.csv(params$file2, sep = "")
clinical <- read.csv(params$file3, sep = "\t")
copy <- read.csv(params$file4, sep = "\t")
```

Haremos un pequeño resumen de los primeros genes de las diversas bases de datos para ver en que consisten estos:

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

```
##     Sample              ELMO2              CREB3L1            RPS11
##  Length:526         Min.   :-1.52492   Min.   :-1.4127   Min.   :-1.02588
##  Class :character   1st Qu.:-0.04492   1st Qu.: 0.3743   1st Qu.: 0.08306
##  Mode  :character   Median : 0.32621   Median : 0.9550   Median : 0.38512
##                     Mean   : 0.31785   Mean   : 0.9259   Mean   : 0.40055
##                     3rd Qu.: 0.60642   3rd Qu.: 1.4921   3rd Qu.: 0.70337
##                     Max.   : 2.90117   Max.   : 3.2203   Max.   : 2.52200
##      PNMA1              MMP2              C10orf90           ZHX3
##  Min.   :-1.72625   Min.   :-2.2847   Min.   :-5.027   Min.   :-1.73750
##  1st Qu.:-0.38362   1st Qu.:-0.3141   1st Qu.:-2.233   1st Qu.:-0.48471
##  Median : 0.08175   Median : 0.4322   Median :-1.933   Median :-0.22008
##  Mean   : 0.02557   Mean   : 0.3678   Mean   :-1.735   Mean   :-0.21253
##  3rd Qu.: 0.45344   3rd Qu.: 1.0822   3rd Qu.:-1.542   3rd Qu.: 0.06704
##  Max.   : 1.60050   Max.   : 2.8012   Max.   : 2.572   Max.   : 1.36167
##      ERCC5              GPR98
##  Min.   :-1.51125   Min.   :-2.8070
##  1st Qu.:-0.29169   1st Qu.:-1.4713
##  Median : 0.02825   Median :-0.2177
##  Mean   : 0.03835   Mean   :-0.4008
##  3rd Qu.: 0.38344   3rd Qu.: 0.4988
##  Max.   : 1.94175   Max.   : 2.3814
```

```r
summary(clinical[,1:4])
```

```
##     Sample            Histology           PAM50Call
##  Length:847         Length:847         Length:847
##  Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character
##  ajcc_cancer_metastasis_stage_code
##  Length:847
##  Class :character
##  Mode  :character
```

```r
summary(protein_abundance[,1:10])
```

```
##     Sample          X14.3.3_epsilon       X4E.BP1          X4E.BP1_pS65
##  Length:410         Min.   :-0.46787   Min.   :-1.19402   Min.   :-0.845501
##  Class :character   1st Qu.:-0.09601   1st Qu.:-0.28960   1st Qu.:-0.164652
##  Mode  :character   Median : 0.01371   Median :-0.04515   Median : 0.005191
##                     Mean   : 0.05439   Mean   : 0.09174   Mean   : 0.026804
##                     3rd Qu.: 0.17245   3rd Qu.: 0.36732   3rd Qu.: 0.192625
##                     Max.   : 0.93994   Max.   : 2.56981   Max.   : 1.123657
##   X4E.BP1_pT37      X4E.BP1_pT70          X53BP1           A.Raf_pS299
##  Min.   :-1.2462   Min.   :-0.72685   Min.   :-1.72613   Min.   :-1.0743520
##  1st Qu.:-0.3216   1st Qu.:-0.11489   1st Qu.:-0.30133   1st Qu.:-0.1954479
##  Median : 0.1167   Median : 0.02684   Median : 0.01189   Median :-0.0146859
##  Mean   : 0.1404   Mean   : 0.05617   Mean   : 0.02492   Mean   : 0.0000203
##  3rd Qu.: 0.4857   3rd Qu.: 0.18398   3rd Qu.: 0.37898   3rd Qu.: 0.1523013
##  Max.   : 2.6901   Max.   : 1.25149   Max.   : 2.65049   Max.   : 1.0728245
##       ACC1              ACC_pS79
##  Min.   :-2.45608   Min.   :-1.59836
##  1st Qu.:-0.41713   1st Qu.:-0.35265
##  Median : 0.06032   Median : 0.08966
##  Mean   : 0.11845   Mean   : 0.17207
##  3rd Qu.: 0.60444   3rd Qu.: 0.63231
##  Max.   : 2.46339   Max.   : 2.68644
```

Vemos como las bases de datos de protein y gene consisten en valores numericos con valores bajos alrededor del 0 tanto en positivo como negativo. También observamos como aquellos genes y proteinas con valores similares tienen también un nonmbre parecido, por consecuente, parece que aquellos genes relacionados entre si o que son parecidos tienen un efecto similar.

```r
set1<-intersect(protein_abundance$Sample,gene$Sample)
set1 <- intersect(set1, clinical$Sample)

xgene<-gene[gene$Sample %in% set1,]
xprotein<-protein_abundance[protein_abundance$Sample %in% set1,]
```

```r
xclinical <- clinical[clinical$Sample %in% set1,]
xclinical <- xclinical[,c(1,9)]

dim(xgene)
```

```
## [1]   387 17815
```

387 individuos están presentes en los datasets clinical, gene y protein_abundance.

```r
sum(is.na(xgene)) # 1161 missings en gene
```

```
## [1] 1161
```

```r
sum(is.na(xprotein)) # 0 missings en protein
```

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

```r
sum(is.na(xclinical)) # 0 missings en protein
```

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

Podemos observar como en la base de datos "gene_expression" hay 1161 valores missing, que arreglaremos mas adelante.

With gene expression data:

2. Select the 25% of genes with the most variability

```r
gene_var <- diag(var(gene[,2:ncol(gene)], na.rm = T)) # Calculamos la variabilidad de los genes

gene_topvar <- sort(gene_var, decreasing = T)[1:(length(gene_var)/20)]
# Cogemos solo el 5% con la mayor variabilidad para no tener problemas con los modelos

# Separamos los 5% de genes con mas variabilidad

xgene <- xgene[,c("Sample", names(gene_topvar))]

siNAcol <- apply(is.na(xgene), 2, sum) >= 1

xgene <- xgene[,!siNAcol]

sum(is.na(xgene)) # No hi ha missings
```

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

```r
dim(xgene)
```

```
## [1] 387 890
```

En lugar de utilizar el 25% de genes con más variabilidad, utilizaremos sólo el 5% debido a que el número de variables es relativamente grande para ejecutar el programa en nuestros ordenadores personales.Asimismo, obtenemos un conjunto de datos sin missings.

Los datos de gene expression que utilizaremos tienen una dimension final de 387 observaciones y 890 variables.

```r
# Eliminamos las filas donde la respuesta no es ni negativa ni positiva:
sel1<-which(xclinical$breast_carcinoma_estrogen_receptor_status != "Positive")
sel2<-which(xclinical$breast_carcinoma_estrogen_receptor_status != "Negative")
sel<-intersect(sel1,sel2)
```

```
xclinical<-xclinical[-sel,]
data1 <- merge(xclinical, xgene, by.x = "Sample", by.y = "Sample")
```

Generamos los datos de training y test para los modelos que vengan a continuacion de la base de datos gene:

```
set.seed(123)
training<-sample(1:nrow(data1),2*nrow(data1)/3)

escalat1 <- scale(data1[,-c(1,2)])

xtrain<-escalat1[training,]
xtest<-escalat1[-training,]
ytrain<-data1[training,2]
ytest<-data1[-training,2]
ylabels<-vector()
ylabels[ytrain=="Positive"]<-1
ylabels[ytrain=="Negative"]<-0
ylabelstest<-vector()
ylabelstest[ytest=="Positive"]<-1
ylabelstest[ytest=="Negative"]<-0
```

3. Implement an stacked autoencoder (SAE) with three stacked layers of 1000, 100, 50 nodes. Provide in each case evidence of the quality of the coding obtained.

Empezaremos generando el primer autoencoder con un shape de el numero de columnas nunmericas que hay en la base de datos xgene, que es el mismo numero de columnas que tiene el train, en el primer layer_input con los datos de "gene_expression". El primer decode tendra 1000 nodos y asi succesivamente con los valores dados en el enunciado.

```
# Autoencoder 1

# Encoder
input_enc1 <- layer_input(shape = (ncol(xgene) - 1))
```

```
## Loaded Tensorflow version 2.6.0
```

```
output_enc1 <- input_enc1 %>%
  layer_dense(units = 1000, activation = "relu", name='G_Enc1')
encoder1 = keras_model(input_enc1, output_enc1)
summary(encoder1)
```

```
## Model: "model"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                    Param #
## ================================================================================
## input_1 (InputLayer)                [(None, 889)]                   0
## ________________________________________________________________________________
## G_Enc1 (Dense)                      (None, 1000)                    890000
## ================================================================================
## Total params: 890,000
## Trainable params: 890,000
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
# Decoder
input_dec1 = layer_input(shape = 1000)
output_dec1 <- input_dec1 %>%
  layer_dense(units = (ncol(xgene)-1), activation="linear", name='G_Dec1')
decoder1 = keras_model(input_dec1, output_dec1)
summary(decoder1)
```

```
## Model: "model_1"
## ________________________________________________________________________________
## Layer (type)                      Output Shape                      Param #
## ================================================================================
## input_2 (InputLayer)              [(None, 1000)]                    0
## ________________________________________________________________________________
## G_Dec1 (Dense)                    (None, 889)                       889889
## ================================================================================
## Total params: 889,889
## Trainable params: 889,889
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
# Juntar el encoder y el decoder
aen_input1 = layer_input(shape = (ncol(xgene)-1))
aen_output1 = aen_input1 %>%
  encoder1() %>%
  decoder1()
sae1 = keras_model(aen_input1, aen_output1)
summary(sae1)
```

```
## Model: "model_2"
## ________________________________________________________________________________
## Layer (type)                      Output Shape                      Param #
## ================================================================================
## input_3 (InputLayer)              [(None, 889)]                     0
## ________________________________________________________________________________
## model (Functional)                (None, 1000)                      890000
## ________________________________________________________________________________
## model_1 (Functional)              (None, 889)                       889889
## ================================================================================
## Total params: 1,779,889
## Trainable params: 1,779,889
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
sae1 %>% compile(
optimizer = "rmsprop",
loss = "mse")

sae1 %>% fit(
x=as.matrix(xtrain),
y=as.matrix(xtrain),
epochs = 25,
batch_size=64,
validation_split = 0.2)

#Generador con en encoder
encoded_expression1 <- encoder1 %>% predict(as.matrix(xtrain))
```

El primer autoencoder tiene 1779889 parámetros.

```r
# Autoencoder 2
input_enc2 <- layer_input(shape = 1000)
output_enc2 <- input_enc2 %>%
  layer_dense(units = 100, activation = "relu", name='Enc_AE2')
encoder2 = keras_model(input_enc2, output_enc2)
summary(encoder2)
```

```
## Model: "model_3"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                        Param #
## ================================================================================
## input_4 (InputLayer)                [(None, 1000)]                      0
## ________________________________________________________________________________
## Enc_AE2 (Dense)                     (None, 100)                         100100
## ================================================================================
## Total params: 100,100
## Trainable params: 100,100
## Non-trainable params: 0
## ________________________________________________________________________________
```

```r
input_dec2 = layer_input(shape = 100)
output_dec2 <- input_dec2 %>%
  layer_dense(units = 1000, activation="linear", name='Dec_AE1')
decoder2 = keras_model(input_dec2, output_dec2)
summary(decoder2)
```

```
## Model: "model_4"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                        Param #
## ================================================================================
## input_5 (InputLayer)                [(None, 100)]                       0
## ________________________________________________________________________________
## Dec_AE1 (Dense)                     (None, 1000)                        101000
## ================================================================================
## Total params: 101,000
## Trainable params: 101,000
## Non-trainable params: 0
## ________________________________________________________________________________
```

```r
aen_input2 = input_enc2
aen_output2 = aen_input2 %>%
  encoder2() %>%
  decoder2()
sae2 = keras_model(aen_input2, aen_output2)
summary(sae2)
```

```
## Model: "model_5"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                        Param #
## ================================================================================
## input_4 (InputLayer)                [(None, 1000)]                      0
## ________________________________________________________________________________
## model_3 (Functional)                (None, 100)                         100100
## ________________________________________________________________________________
## model_4 (Functional)                (None, 1000)                        101000
## ================================================================================
## Total params: 201,100
## Trainable params: 201,100
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
sae2 %>% compile(
optimizer = "rmsprop",
loss = "mse")

sae2 %>% fit(
x=as.matrix(encoded_expression1),
y=as.matrix(encoded_expression1),
epochs = 25,
batch_size=64,
validation_split = 0.2)

encoded_expression2 <- encoder2 %>% predict(as.matrix(encoded_expression1))
```

Este autoencoder tiene 201100 parámetros

```
# Autoencoder 3

# Encoder
input_enc3 <- layer_input(shape = 100)
output_enc3 <- input_enc3 %>%
  layer_dense(units = 50, activation = "relu", name='Enc_AE3')
encoder3 = keras_model(input_enc3, output_enc3)
summary(encoder3)
```

```
## Model: "model_6"
## ________________________________________________________________________________
## Layer (type)                       Output Shape                     Param #
## ================================================================================
## input_6 (InputLayer)               [(None, 100)]                    0
## ________________________________________________________________________________
## Enc_AE3 (Dense)                    (None, 50)                       5050
## ================================================================================
## Total params: 5,050
## Trainable params: 5,050
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
# Decoder
input_dec3 = layer_input(shape = 50)
output_dec3 <- input_dec3 %>%
  layer_dense(units = 100, activation="linear", name='Dec_AE1')
decoder3 = keras_model(input_dec3, output_dec3)
summary(decoder3)
```

```
## Model: "model_7"
## ________________________________________________________________________________
## Layer (type)                       Output Shape                     Param #
## ================================================================================
## input_7 (InputLayer)               [(None, 50)]                     0
## ________________________________________________________________________________
## Dec_AE1 (Dense)                    (None, 100)                     5100
## ================================================================================
## Total params: 5,100
## Trainable params: 5,100
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
aen_input3 = input_enc3
aen_output3 = aen_input3 %>%
  encoder3() %>%
  decoder3()
sae3 = keras_model(aen_input3, aen_output3)
summary(sae3)
```

```
## Model: "model_8"
## ________________________________________________________________________________
## Layer (type)                       Output Shape                    Param #
## ================================================================================
## input_6 (InputLayer)               [(None, 100)]                   0
## ________________________________________________________________________________
## model_6 (Functional)               (None, 50)                      5050
## ________________________________________________________________________________
## model_7 (Functional)               (None, 100)                     5100
## ================================================================================
## Total params: 10,150
## Trainable params: 10,150
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
sae3 %>% compile(
optimizer = "rmsprop",
loss = "mse")

sae3 %>% fit(
x=as.matrix(encoded_expression2),
y=as.matrix(encoded_expression2),
epochs = 40,
batch_size=64,
validation_split = 0.2)

encoded_expression3 <- encoder3 %>% predict(as.matrix(encoded_expression2))
```

El tercer autoencoder tiene 10150 parámetros.

4.Using the SAE as pre-training model, couple it with a two-layer DNN to predict the state of the estrogen receptor. The DNN must have 10 nodes in the first layer followed by the output layer.

Generamos el modelo juntando los 3 encoders anteriores

```
sae_input = layer_input(shape = (ncol(xgene)-1), name = "input_gene")
sae_output = sae_input %>%
  encoder1() %>%
  encoder2() %>%
  encoder3() %>%
  layer_dense(10,activation = "relu", name='L1_SAE1')%>%
  layer_dense(1,activation = "sigmoid", name='L2_SAE1')
```

```
sae = keras_model(sae_input, sae_output)
summary(sae)
```

```
## Model: "model_9"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                       Param #
## ================================================================================
## input_gene (InputLayer)             [(None, 889)]                      0
## ________________________________________________________________________________
## model (Functional)                  (None, 1000)                       890000
## ________________________________________________________________________________
## model_3 (Functional)                (None, 100)                        100100
## ________________________________________________________________________________
## model_6 (Functional)                (None, 50)                         5050
## ________________________________________________________________________________
## L1_SAE1 (Dense)                     (None, 10)                         510
## ________________________________________________________________________________
## L2_SAE1 (Dense)                     (None, 1)                          11
## ================================================================================
## Total params: 995,671
## Trainable params: 995,671
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
freeze_weights(sae,from=1,to=3)
```

Al juntar todos los encoder en un mismo modelo tenemos que este acaba teniendo 995671 parametros.

```
sae %>% compile(
optimizer = "rmsprop",
loss = 'binary_crossentropy',
metric = "acc"
)
```

```
sae %>% fit(
x=xtrain,
y=ylabels,
epochs = 15,
batch_size = 64,
validation_split = 0.2
)
```

El valor de accuracy que obtenemos es cercano a 0.85 y la pérdida es aproximadamente 0.4

```
sae %>% evaluate(as.matrix(xtest), ylabelstest)
```

```
##      loss       acc
## 0.3277466 0.8897638
```

Cuando evaluamos el modelo con los datos de test, conseguimos unos valores de las métricas de loss y accuracy muy parecidos a los de entrenamiento, por lo que podemos decir que tenemos un buen rendimiento en el modelo.

```
yhat <- predict(sae,as.matrix(xtest))
```

```
yhatclass<-as.factor(ifelse(yhat<0.5,0,1))
table(yhatclass, ylabelstest)
```

```
##          ylabelstest
## yhatclass  0  1
##         0 20  6
##         1  8 93
```

```
confusionMatrix(yhatclass,as.factor(ylabelstest))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 20  6
##          1  8 93
##
##                Accuracy : 0.8898
##                  95% CI : (0.822, 0.9384)
##     No Information Rate : 0.7795
##     P-Value [Acc > NIR] : 0.001011
##
##                   Kappa : 0.6709
##
##  Mcnemar's Test P-Value : 0.789268
##
##             Sensitivity : 0.7143
##             Specificity : 0.9394
##          Pos Pred Value : 0.7692
##          Neg Pred Value : 0.9208
##              Prevalence : 0.2205
##          Detection Rate : 0.1575
##    Detection Prevalence : 0.2047
##       Balanced Accuracy : 0.8268
##
##        'Positive' Class : 0
##
```

Vemos que al predecir valores con la prediccion observamos como el modelo tiene mayor error a la hora de predecir los casos negativos (0). Puede ser debido a que hay un número mayor de muestras con respuesta positiva, por lo que el modelo está más entrenado para este caso.

5.On the test set, provide the ROC curve and AUC and other performance metrics.

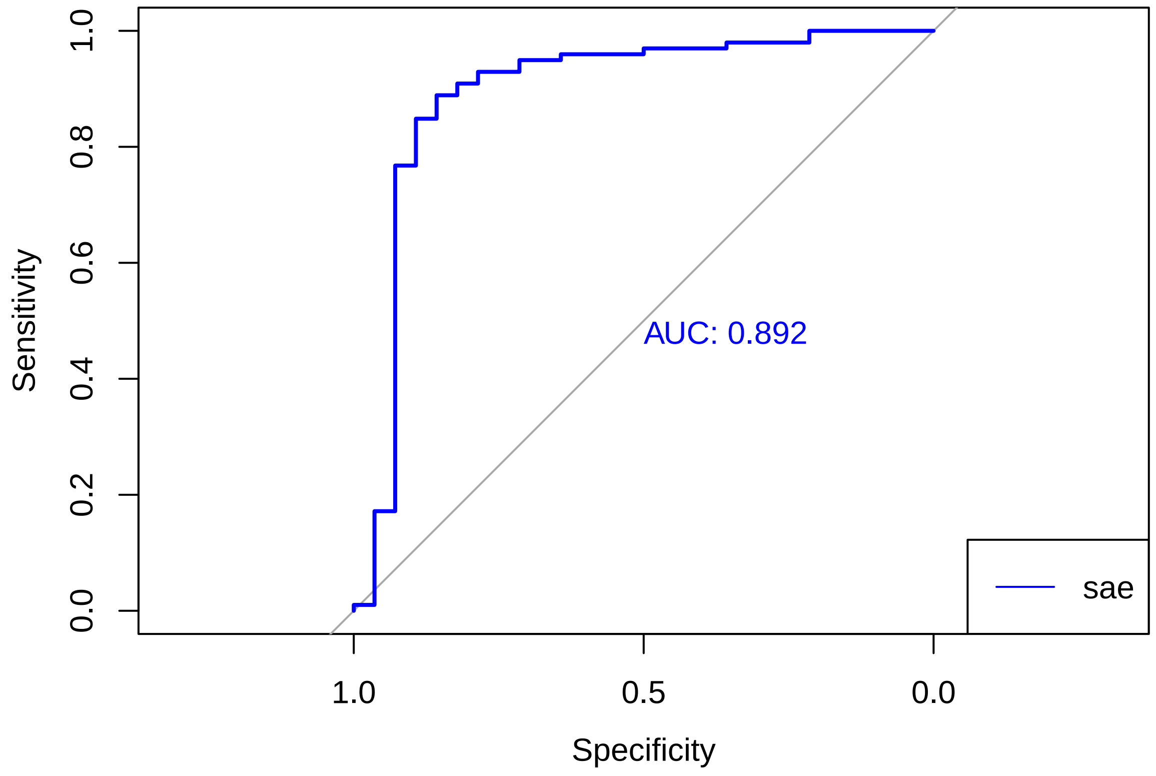
```
roc_sae_test <- roc(response = ylabelstest, predictor =yhat)
```

```
## Setting levels: control = 0, case = 1
```

```
## Warning in roc.default(response = ylabelstest, predictor = yhat): Deprecated use
## a matrix as predictor. Unexpected results may be produced, please pass a numeric
## vector.
```

```
## Setting direction: controls < cases
```

```
plot(roc_sae_test, col = "blue", print.auc=TRUE)
legend("bottomright", legend = c("sae"), lty = c(1), col = c("blue"))
```



En este grafico para ver el valor de auc obtenemos que este valor es de 0.915, teniendo asi que este modelo tiene un buen valor de diagnostico.

6. With tfruns() repeat points 4 and 5, exploring the configurations of the first layer of the DNN based on 5, 10 and 20 nodes. Determine which configuration is the best.

Para realizar el tfruns, generaremos el código en otro archivo .R y entonces cargaremos aqui los diferentes modelos con el codigo que hay a continuacion.

```
library(tfruns)
```

```
## Warning: package 'tfruns' was built under R version 4.1.2
```

```
nodes <- c(5, 10, 20)

for (i in 1:length(nodes)){
  print(i)
  training_run("Breast_tfruns.R",
               flags = c(units = nodes[i]))
}
```

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

```
## Using run directory runs/2022-04-10T19-30-26Z
```

```
##
## > FLAGS <- flags(flag_integer("units", 10))
##
## > sae_input = layer_input(shape = (ncol(xgene) - 1))
##
```

```
## > sae_output = sae_input %>% encoder1() %>% encoder2() %>%
## +     encoder3() %>% layer_dense(FLAGS$units, activation = "relu") %>%
## +     layer_dense( .... [TRUNCATED]
##
## > sae = keras_model(sae_input, sae_output)
##
## > summary(sae)
## Model: "model_10"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                    Param #
## ================================================================================
## input_8 (InputLayer)                [(None, 889)]                   0
## ________________________________________________________________________________
## model (Functional)                  (None, 1000)                    890000
## ________________________________________________________________________________
## model_3 (Functional)                (None, 100)                     100100
## ________________________________________________________________________________
## model_6 (Functional)                (None, 50)                      5050
## ________________________________________________________________________________
## dense_1 (Dense)                     (None, 5)                       255
## ________________________________________________________________________________
## dense (Dense)                       (None, 1)                       6
## ================================================================================
## Total params: 995,411
## Trainable params: 5,311
## Non-trainable params: 990,100
## ________________________________________________________________________________
##
## > sae %>% compile(optimizer = "rmsprop", loss = "binary_crossentropy",
## +     metric = "acc")
##
## > sae %>% fit(x = xtrain, y = ylabels, epochs = 15,
## +     batch_size = 64, validation_split = 0.2)
##
## > yhat <- predict(sae, as.matrix(xtest))
##
## > yhatclass <- as.factor(ifelse(yhat < 0.5, 0, 1))
##
## > table(yhatclass, ylabelstest)
##          ylabelstest
## yhatclass  0  1
##         0 22  7
##         1  6 92
##
## > confusionMatrix(yhatclass, as.factor(ylabelstest))
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 22  7
##          1  6 92
##
##                Accuracy : 0.8976
##                  95% CI : (0.8313, 0.9444)
##     No Information Rate : 0.7795
##     P-Value [Acc > NIR] : 0.0004164
##
##                   Kappa : 0.706
##
##  Mcnemar's Test P-Value : 1.0000000
##
##             Sensitivity : 0.7857
##             Specificity : 0.9293
##          Pos Pred Value : 0.7586
##          Neg Pred Value : 0.9388
##              Prevalence : 0.2205
##          Detection Rate : 0.1732
##    Detection Prevalence : 0.2283
##       Balanced Accuracy : 0.8575
##
##        'Positive' Class : 0
##
##
## > roc_sae_test <- roc(response = ylabelstest, predictor = yhat)

## Setting levels: control = 0, case = 1

## Warning in roc.default(response = ylabelstest, predictor = yhat): Deprecated use
## a matrix as predictor. Unexpected results may be produced, please pass a numeric
## vector.

## Setting direction: controls < cases

##
## > plot(roc_sae_test, col = "blue", print.auc = TRUE)

##
## > legend("bottomright", legend = c("sae"), lty = c(1),
## +     col = c("blue"))

##
## Run completed: runs/2022-04-10T19-30-26Z

## [1] 2

## Using run directory runs/2022-04-10T19-30-30Z

##
## > FLAGS <- flags(flag_integer("units", 10))
##
## > sae_input = layer_input(shape = (ncol(xgene) - 1))
##
## > sae_output = sae_input %>% encoder1() %>% encoder2() %>%
## +     encoder3() %>% layer_dense(FLAGS$units, activation = "relu") %>%
## +     layer_dense( .... [TRUNCATED]
##
## > sae = keras_model(sae_input, sae_output)
##
## > summary(sae)
## Model: "model"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                     Param #
## ================================================================================
## input_1 (InputLayer)             [(None, 889)]                   0
## ________________________________________________________________________________
## model (Functional)                (None, 1000)                    890000
## ________________________________________________________________________________
## model_3 (Functional)              (None, 100)                     100100
## ________________________________________________________________________________
## model_6 (Functional)              (None, 50)                      5050
## ________________________________________________________________________________
## dense_1 (Dense)                   (None, 10)                      510
## ________________________________________________________________________________
## dense (Dense)                     (None, 1)                       11
## ================================================================================
## Total params: 995,671
## Trainable params: 5,571
## Non-trainable params: 990,100
## ________________________________________________________________________________
##
## > sae %>% compile(optimizer = "rmsprop", loss = "binary_crossentropy",
## +     metric = "acc")
##
## > sae %>% fit(x = xtrain, y = ylabels, epochs = 15,
## +     batch_size = 64, validation_split = 0.2)
##
## > yhat <- predict(sae, as.matrix(xtest))
##
## > yhatclass <- as.factor(ifelse(yhat < 0.5, 0, 1))
##
## > table(yhatclass, ylabelstest)
##          ylabelstest
## yhatclass  0  1
##         0 21  6
##         1  7 93
##
## > confusionMatrix(yhatclass, as.factor(ylabelstest))
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 21  6
##          1  7 93
##
##                Accuracy : 0.8976
##                  95% CI : (0.8313, 0.9444)
##     No Information Rate : 0.7795
##     P-Value [Acc > NIR] : 0.0004164
##
##                   Kappa : 0.6983
##
##  Mcnemar's Test P-Value : 1.0000000
##
##             Sensitivity : 0.7500
##             Specificity : 0.9394
##          Pos Pred Value : 0.7778
##          Neg Pred Value : 0.9300
##              Prevalence : 0.2205
##          Detection Rate : 0.1654
##    Detection Prevalence : 0.2126
##       Balanced Accuracy : 0.8447
##
##        'Positive' Class : 0
##
##
## > roc_sae_test <- roc(response = ylabelstest, predictor = yhat)

## Setting levels: control = 0, case = 1

## Warning in roc.default(response = ylabelstest, predictor = yhat): Deprecated use
## a matrix as predictor. Unexpected results may be produced, please pass a numeric
## vector.

## Setting direction: controls < cases

##
## > plot(roc_sae_test, col = "blue", print.auc = TRUE)

##
## > legend("bottomright", legend = c("sae"), lty = c(1),
## +     col = c("blue"))

##
## Run completed: runs/2022-04-10T19-30-30Z

## [1] 3

## Using run directory runs/2022-04-10T19-30-34Z

##
## > FLAGS <- flags(flag_integer("units", 10))
##
## > sae_input = layer_input(shape = (ncol(xgene) - 1))
##
## > sae_output = sae_input %>% encoder1() %>% encoder2() %>%
## +     encoder3() %>% layer_dense(FLAGS$units, activation = "relu") %>%
## +     layer_dense( .... [TRUNCATED]
##
## > sae = keras_model(sae_input, sae_output)
##
## > summary(sae)
## Model: "model"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                    Param #
## ================================================================================
## input_1 (InputLayer)                [(None, 889)]                   0
## ________________________________________________________________________________
## model (Functional)                  (None, 1000)                    890000
## ________________________________________________________________________________
## model_3 (Functional)                (None, 100)                     100100
## ________________________________________________________________________________
## model_6 (Functional)                (None, 50)                      5050
## ________________________________________________________________________________
## dense_1 (Dense)                     (None, 20)                      1020
## ________________________________________________________________________________
## dense (Dense)                       (None, 1)                           21
## ================================================================================
## Total params: 996,191
## Trainable params: 6,091
## Non-trainable params: 990,100
## ________________________________________________________________________________
##
## > sae %>% compile(optimizer = "rmsprop", loss = "binary_crossentropy",
## +     metric = "acc")
##
## > sae %>% fit(x = xtrain, y = ylabels, epochs = 15,
## +     batch_size = 64, validation_split = 0.2)
##
## > yhat <- predict(sae, as.matrix(xtest))
##
## > yhatclass <- as.factor(ifelse(yhat < 0.5, 0, 1))
##
## > table(yhatclass, ylabelstest)
##          ylabelstest
## yhatclass  0  1
##         0 22  4
##         1  6 95
##
## > confusionMatrix(yhatclass, as.factor(ylabelstest))
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 22  4
##          1  6 95
##
##                Accuracy : 0.9213
##                  95% CI : (0.86, 0.9616)
##     No Information Rate : 0.7795
##     P-Value [Acc > NIR] : 1.759e-05
##
##                   Kappa : 0.7649
##
##  Mcnemar's Test P-Value : 0.7518
##
##             Sensitivity : 0.7857
##             Specificity : 0.9596
##          Pos Pred Value : 0.8462
##          Neg Pred Value : 0.9406
##              Prevalence : 0.2205
##          Detection Rate : 0.1732
##    Detection Prevalence : 0.2047
##       Balanced Accuracy : 0.8727
##
##        'Positive' Class : 0
##
##
## > roc_sae_test <- roc(response = ylabelstest, predictor = yhat)
## Setting levels: control = 0, case = 1

## Warning in roc.default(response = ylabelstest, predictor = yhat): Deprecated use
## a matrix as predictor. Unexpected results may be produced, please pass a numeric
## vector.

## Setting direction: controls < cases

##
## > plot(roc_sae_test, col = "blue", print.auc = TRUE)

##
## > legend("bottomright", legend = c("sae"), lty = c(1),
## +     col = c("blue"))

##
## Run completed: runs/2022-04-10T19-30-34Z
```

```r
runs <- ls_runs(latest_n = 3)
runs <- runs[, c("flag_units", "metric_val_acc", "metric_val_loss")]
(runs <- runs[order(runs$metric_val_acc, decreasing = T),])
```

```
## Data frame: 3 x 3
##   flag_units metric_val_acc metric_val_loss
## 1         20         0.8824          0.4612
## 3          5         0.8824          0.4392
## 2         10         0.8627          0.5518
```

Vemos que los tres modelos ejecutados tienen un accuracy parecido, por lo que en los siguientes ejercicios utilizaremos la configuración inicial.

So far, we have two SAEs. One for the abundance of proteins (see class examples) and the other for gene expression we just built.

7. Split the set of patients with complete data (gene expression and protein abundance) in train and test sets.

```r
xprotein<-protein_abundance[protein_abundance$Sample %in% set1,]
```

```r
data2 <- merge(xclinical, xprotein, by.x = "Sample", by.y = "Sample")
data2 <- merge(data2, xgene, by.x = "Sample", by.y = "Sample")
```

```r
escalat2 <- scale(data2[,-c(1,2)])



xtrain2<-escalat2[training,-c(1,2)]
xtest2<-escalat2[-training,-c(1,2)]
xtrain2<-scale(xtrain2)
xtest2<-scale(xtest2)
ytrain2<-escalat2[training,2]
ytest2<-escalat2[-training,2]
ylabels2<-vector()
ylabels2[ytrain2=="Positive"]<-1
ylabels2[ytrain2=="Negative"]<-0
ylabelstest2<-vector()
ylabelstest2[ytest2=="Positive"]<-1
ylabelstest2[ytest2=="Negative"]<-0
```

8. Concatenate the two SAEs to fit, on the trainset, a DNN that integrates both data sources to predict estrogen receptor status. The DNN must have a dense layer (with the better number

of nodes according with point 6) and the output layer.

Modelo de la proteina

```
data3<-merge(xclinical,xprotein,by.x="Sample",by.y="Sample")
```

```
escalat3 <- scale(data3[,-c(1,2)])

xtrain3<-escalat3[training,]
xtest3<-escalat3[-training,]

ytrain3<-data3[training,2]
ytest3<-data3[-training,2]

ylabels3<-vector()
ylabels3[ytrain3=="Positive"]<-1
ylabels3[ytrain3=="Negative"]<-0


ytestlabels3<-vector()
ytestlabels3[ytest3=="Positive"]<-1
ytestlabels3[ytest3=="Negative"]<-0
```

```
# AE1
input_enc1_prot<-layer_input(shape = 142)
output_enc1_prot<-input_enc1_prot %>%
  layer_dense(units=50,activation="relu")
encoder1_prot = keras_model(input_enc1_prot, output_enc1_prot, name = "AE1")
summary(encoder1_prot)
```

```
## Model: "AE1"
## ________________________________________________________________________________
## Layer (type)                       Output Shape                     Param #
## ================================================================================
## input_1 (InputLayer)               [(None, 142)]                    0
## ________________________________________________________________________________
## dense (Dense)                      (None, 50)                       7150
## ================================================================================
## Total params: 7,150
## Trainable params: 7,150
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
input_dec1_prot = layer_input(shape = 50)
output_dec1_prot<-input_dec1_prot %>%
  layer_dense(units=142,activation="linear")

decoder1_prot = keras_model(input_dec1_prot, output_dec1_prot)

summary(decoder1_prot)
```

```
## Model: "model"
## ________________________________________________________________________________
## Layer (type)                       Output Shape                     Param #
## ================================================================================
## input_2 (InputLayer)               [(None, 50)]                     0
## ________________________________________________________________________________
## dense_1 (Dense)                    (None, 142)                     7242
## ================================================================================
## Total params: 7,242
## Trainable params: 7,242
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
aen_input1_prot = layer_input(shape = 142)
aen_output1_prot = aen_input1_prot %>%
  encoder1_prot() %>%
  decoder1_prot()

sae1_prot = keras_model(aen_input1_prot, aen_output1_prot)
summary(sae1_prot)
```

```
## Model: "model_1"
## ________________________________________________________________________________
## Layer (type)                       Output Shape                    Param #
## ================================================================================
## input_3 (InputLayer)               [(None, 142)]                   0
## ________________________________________________________________________________
## AE1 (Functional)                   (None, 50)                      7150
## ________________________________________________________________________________
## model (Functional)                 (None, 142)                     7242
## ================================================================================
## Total params: 14,392
## Trainable params: 14,392
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
sae1_prot %>% compile(
  optimizer = "rmsprop",
  loss = "mse"
)

sae1_prot %>% fit(
  x=as.matrix(xtrain3),
  y=as.matrix(xtrain3),
  epochs = 50,
  batch_size=64,
  validation_split = 0.2
  )

#Generating with Autoencoder
encoded_expression1_prot <- encoder1_prot %>% predict(as.matrix(xtrain3))
```

El primer autoencoder tiene 14392 parámetros

```
# AE2
input_enc2_prot<-layer_input(shape = 50)
output_enc2_prot<-input_enc2_prot %>%
  layer_dense(units=20,activation="relu")
encoder2_prot = keras_model(input_enc2_prot, output_enc2_prot)
summary(encoder2_prot)
```

```
## Model: "model_2"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                        Param #
## ================================================================================
## input_4 (InputLayer)                [(None, 50)]                        0
## ________________________________________________________________________________
## dense_2 (Dense)                     (None, 20)                          1020
## ================================================================================
## Total params: 1,020
## Trainable params: 1,020
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
input_dec2_prot = layer_input(shape = 20)
output_dec2_prot<-input_dec2_prot %>%
  layer_dense(units=50,activation="linear")

decoder2_prot = keras_model(input_dec2_prot, output_dec2_prot)

summary(decoder2_prot)
```

```
## Model: "model_3"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                        Param #
## ================================================================================
## input_5 (InputLayer)                [(None, 20)]                        0
## ________________________________________________________________________________
## dense_3 (Dense)                     (None, 50)                          1050
## ================================================================================
## Total params: 1,050
## Trainable params: 1,050
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
aen_input2_prot = layer_input(shape = 50)
aen_output2_prot = aen_input2_prot %>%
  encoder2_prot() %>%
  decoder2_prot()

sae2_prot = keras_model(aen_input2_prot, aen_output2_prot)
summary(sae2_prot)
```

```
## Model: "model_4"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                        Param #
## ================================================================================
## input_6 (InputLayer)                [(None, 50)]                        0
## ________________________________________________________________________________
## model_2 (Functional)                (None, 20)                          1020
## ________________________________________________________________________________
## model_3 (Functional)                (None, 50)                          1050
## ================================================================================
## Total params: 2,070
## Trainable params: 2,070
## Non-trainable params: 0
## ___________________________________________________________________________
```

```
sae2_prot %>% compile(
  optimizer = "rmsprop",
  loss = "mse"
)

sae2_prot %>% fit(
  x=as.matrix(encoded_expression1_prot),
  y=as.matrix(encoded_expression1_prot),
  epochs = 50,
  batch_size=64,
  validation_split = 0.2
  )

#Generating with Autoencoder
encoded_expression2_prot <- encoder2_prot %>% predict(as.matrix(encoded_expression1_prot))
```

El segundo autoencoder tiene 2070 parámetros.

```
# AE3
input_enc3_prot<-layer_input(shape = 20)
output_enc3_prot<-input_enc3_prot %>%
  layer_dense(units=10,activation="relu")
encoder3_prot = keras_model(input_enc3_prot, output_enc3_prot)
summary(encoder3_prot)
```

```
## Model: "model_5"
## ___________________________________________________________________________
## Layer (type)                      Output Shape                   Param #
## ===========================================================================
## input_7 (InputLayer)              [(None, 20)]                   0
## ___________________________________________________________________________
## dense_4 (Dense)                   (None, 10)                     210
## ===========================================================================
## Total params: 210
## Trainable params: 210
## Non-trainable params: 0
## ___________________________________________________________________________
```

```
input_dec3_prot = layer_input(shape = 10)
output_dec3_prot<-input_dec3_prot %>%
  layer_dense(units=20,activation="linear")

decoder3_prot = keras_model(input_dec3_prot, output_dec3_prot)

summary(decoder3_prot)
```

```
## Model: "model_6"
## ___________________________________________________________________________
## Layer (type)                      Output Shape                   Param #
## ===========================================================================
## input_8 (InputLayer)              [(None, 10)]                   0
## ___________________________________________________________________________
## dense_5 (Dense)                   (None, 20)                     220
## ===========================================================================
## Total params: 220
## Trainable params: 220
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
aen_input3_prot = layer_input(shape = 20)
aen_output3_prot = aen_input3_prot %>%
  encoder3_prot() %>%
  decoder3_prot()

sae3_prot = keras_model(aen_input3_prot, aen_output3_prot)
summary(sae3_prot)
```

```
## Model: "model_7"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                    Param #
## ================================================================================
## input_9 (InputLayer)                [(None, 20)]                    0
## ________________________________________________________________________________
## model_5 (Functional)                (None, 10)                      210
## ________________________________________________________________________________
## model_6 (Functional)                (None, 20)                      220
## ================================================================================
## Total params: 430
## Trainable params: 430
## Non-trainable params: 0
## ________________________________________________________________________________
```

```
sae3_prot %>% compile(
  optimizer = "rmsprop",
  loss = "mse"
)

sae3_prot %>% fit(
  x=as.matrix(encoded_expression2_prot),
  y=as.matrix(encoded_expression2_prot),
  epochs = 50,
  batch_size=64,
  validation_split = 0.2
  )


#Generating with Autoencoder
encoded_expression3_prot <- encoder3_prot %>% predict(as.matrix(encoded_expression2_prot))
```

El tercer autoencoder para el conjunto de datos proteicos tiene 430 parámetros.

```
### Final model

sae_input_prot = layer_input(shape = 142, name = "input_prot")
sae_output_prot = sae_input_prot %>%
  encoder1_prot() %>%
  encoder2_prot()  %>%
  encoder3_prot() %>%
  layer_dense(5,activation = "relu")%>%
  layer_dense(1,activation = "sigmoid")
```

```
sae_prot = keras_model(sae_input_prot, sae_output_prot)
summary(sae_prot)
```

```
## Model: "model_8"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                    Param #
## ================================================================================
## input_prot (InputLayer)             [(None, 142)]                   0
## ________________________________________________________________________________
## AE1 (Functional)                    (None, 50)                      7150
## ________________________________________________________________________________
## model_2 (Functional)                (None, 20)                      1020
## ________________________________________________________________________________
## model_5 (Functional)                (None, 10)                      210
## ________________________________________________________________________________
## dense_7 (Dense)                     (None, 5)                       55
## ________________________________________________________________________________
## dense_6 (Dense)                     (None, 1)                       6
## ================================================================================
## Total params: 8,441
## Trainable params: 8,441
## Non-trainable params: 0
## ________________________________________________________________________________
```

El total de parámetros para el modelo Stacked autoencoder para el conjunto de datos de protein_abundance es de 8170.

```
freeze_weights(sae_prot,from=1,to=3)
```

```
sae_prot %>% compile(
  optimizer = "rmsprop",
  loss = 'binary_crossentropy',
  metric = "acc"
  )
```

```
sae_prot %>% fit(
  x=xtrain3,
  y=ylabels3,
  epochs = 30,
  batch_size=64,
  validation_split = 0.2
  )
```

```
sae_prot %>%
  evaluate(as.matrix(xtest3), ytestlabels3)
```

```
##      loss       acc
## 0.3472343 0.7795275
```

Para este modelo, la precisión en la evaluación del conjunto de test es cercana a 0.90.

```
yhat_prot <- predict(sae_prot,as.matrix(xtest3))
```

```
yhatclass_prot<-as.factor(ifelse(yhat_prot<0.5,0,1))
table(yhatclass_prot,  ytestlabels3)
```

```
##               ytestlabels3
## yhatclass_prot  0  1
##              1 28 99
```

Vemos que el porcentaje de error es parecido en ambos casos.

```
confusionMatrix(yhatclass_prot,as.factor(ytestlabels3))
```

```
## Warning in confusionMatrix.default(yhatclass_prot, as.factor(ytestlabels3)):
## Levels are not in the same order for reference and data. Refactoring data to
## match.
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0  0  0
##          1 28 99
##
##                Accuracy : 0.7795
##                  95% CI : (0.6974, 0.8482)
##     No Information Rate : 0.7795
##     P-Value [Acc > NIR] : 0.5504
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 3.352e-07
##
##             Sensitivity : 0.0000
##             Specificity : 1.0000
##          Pos Pred Value :    NaN
##          Neg Pred Value : 0.7795
##              Prevalence : 0.2205
##          Detection Rate : 0.0000
##    Detection Prevalence : 0.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##
```

Concatenate the 2 models:

```
sae_input_prova1 = layer_input(shape = (ncol(xgene)-1), name = "input_gene_prova")
sae_output_prova1 = sae_input_prova1 %>%
  encoder1() %>%
  encoder2() %>%
  encoder3() %>%
  layer_dense(10,activation = "relu", name='L1_SAE1')%>%
  layer_dense(1,activation = "sigmoid", name='L2_SAE1')
sae_prova1 = keras_model(sae_input_prova1, sae_output_prova1)

sae_input_prova2 = layer_input(shape = 142, name = "input_prot_prova")
sae_output_prova2 = sae_input_prova2 %>%
  encoder1_prot() %>%
  encoder2_prot() %>%
  encoder3_prot() %>%
  layer_dense(10,activation = "relu", name='L1_SAE2')%>%
  layer_dense(1,activation = "sigmoid", name='L2_SAE2')
```

```
sae_prova2 = keras_model(sae_input_prova2, sae_output_prova2)

concatenated<-layer_concatenate(list(sae_output_prova1,sae_output_prova2))

model_output_con<-concatenated %>%
  layer_dense(units = 20,"relu") %>%
  layer_dense(units = 1,activation = "sigmoid")

model_final<-keras_model(list(sae_input_prova1, sae_input_prova2), model_output_con)
summary(model_final)
```

```
## Model: "model_11"
## ________________________________________________________________________________
## Layer (type)                Output Shape      Param #  Connected to
## ================================================================================
## input_gene_prova (InputLa [(None, 889)]     0
## ________________________________________________________________________________
## input_prot_prova (InputLa [(None, 142)]     0
## ________________________________________________________________________________
## model (Functional)         (None, 1000)      890000   input_gene_prova[0][0]
## ________________________________________________________________________________
## AE1 (Functional)           (None, 50)        7150     input_prot_prova[0][0]
## ________________________________________________________________________________
## model_3 (Functional)       (None, 100)       100100   model[5][0]
## ________________________________________________________________________________
## model_2 (Functional)       (None, 20)        1020     AE1[2][0]
## ________________________________________________________________________________
## model_6 (Functional)       (None, 50)        5050     model_3[5][0]
## ________________________________________________________________________________
## model_5 (Functional)       (None, 10)        210      model_2[2][0]
## ________________________________________________________________________________
## L1_SAE1 (Dense)            (None, 10)        510      model_6[5][0]
## ________________________________________________________________________________
## L1_SAE2 (Dense)            (None, 10)        110      model_5[2][0]
## ________________________________________________________________________________
## L2_SAE1 (Dense)            (None, 1)         11       L1_SAE1[0][0]
## ________________________________________________________________________________
## L2_SAE2 (Dense)            (None, 1)         11       L1_SAE2[0][0]
## ________________________________________________________________________________
## concatenate (Concatenate) (None, 2)          0        L2_SAE1[0][0]
##                                                        L2_SAE2[0][0]
## ________________________________________________________________________________
## dense_9 (Dense)            (None, 20)        60       concatenate[0][0]
## ________________________________________________________________________________
## dense_8 (Dense)            (None, 1)         21       dense_9[0][0]
## ================================================================================
## Total params: 1,004,253
## Trainable params: 5,983
## Non-trainable params: 998,270
## ________________________________________________________________________________
```

En el modelo concatenado tenemos un total de 1004253 parámetros.

```
model_final %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = "acc"
)


# training

model_final %>% fit(
  x = list(input_gene_prova = as.matrix(xtrain), input_prot_prova = as.matrix(xtrain3)),
  y = array(ylabels),  epochs = 30, batch_size = 64, validation_split = 0.2
)
```

```
model_final %>%
  evaluate(list(as.matrix(xtest), as.matrix(xtest3)), ylabelstest)
```

```
##      loss       acc
## 0.4204175 0.9448819
```

En el modelo concatenado, el valor de pérdida es aproximadamente 0.50 y el de accuracy superior a 0.85.

```
yhat_final <- predict(model_final,list(as.matrix(xtest), as.matrix(xtest3)))
```

```
yhatclass_final<-as.factor(ifelse(yhat_final<0.5,0,1))
table(yhatclass_final,  ylabelstest)
```

```
##                ylabelstest
## yhatclass_final  0  1
##               0 25  4
##               1  3 95
```

```
confusionMatrix(yhatclass_final,as.factor(ylabelstest))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 25  4
##          1  3 95
##
##                Accuracy : 0.9449
##                  95% CI : (0.8897, 0.9776)
##     No Information Rate : 0.7795
##     P-Value [Acc > NIR] : 2.95e-07
##
##                   Kappa : 0.8417
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8929
##             Specificity : 0.9596
##          Pos Pred Value : 0.8621
##          Neg Pred Value : 0.9694
##              Prevalence : 0.2205
##          Detection Rate : 0.1969
##    Detection Prevalence : 0.2283
##       Balanced Accuracy : 0.9262
##
##        'Positive' Class : 0
##
```

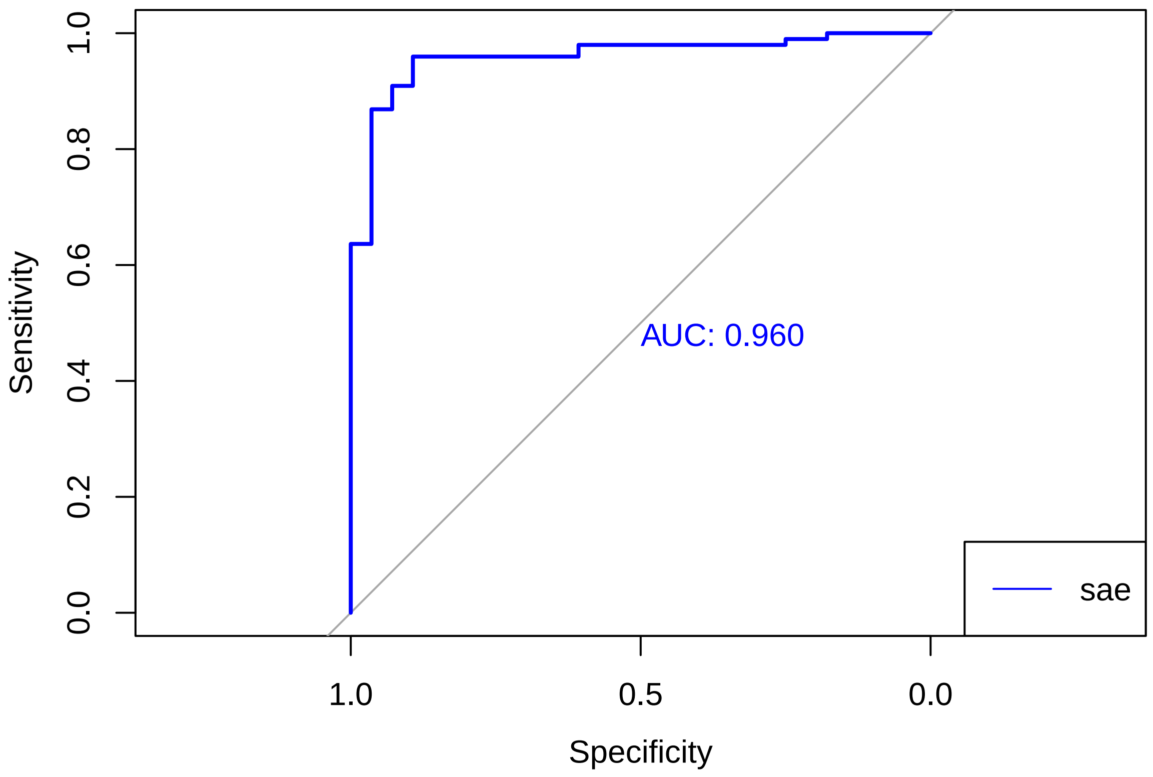
9. On the testset, provide the ROC curve and AUC, and compare it with the model found in point 5.

```
roc_sae_test2 <- roc(response = ylabelstest, predictor =as.vector(yhat_final))
```

```
## Setting levels: control = 0, case = 1
```

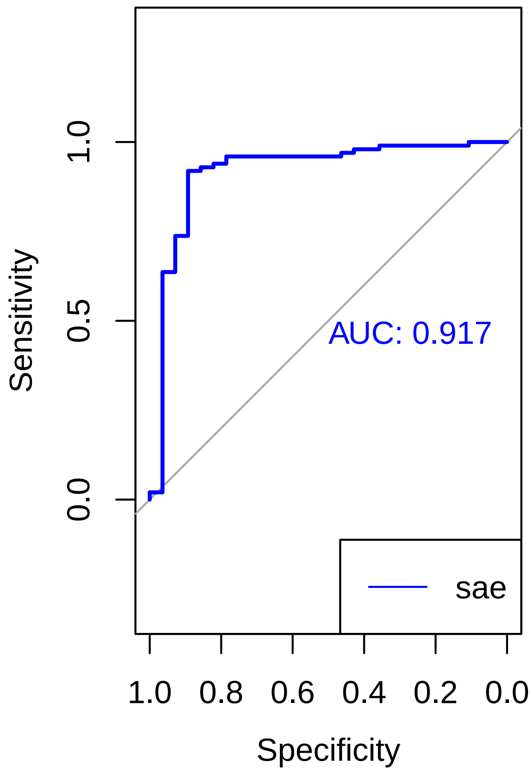
```
## Setting direction: controls < cases
```

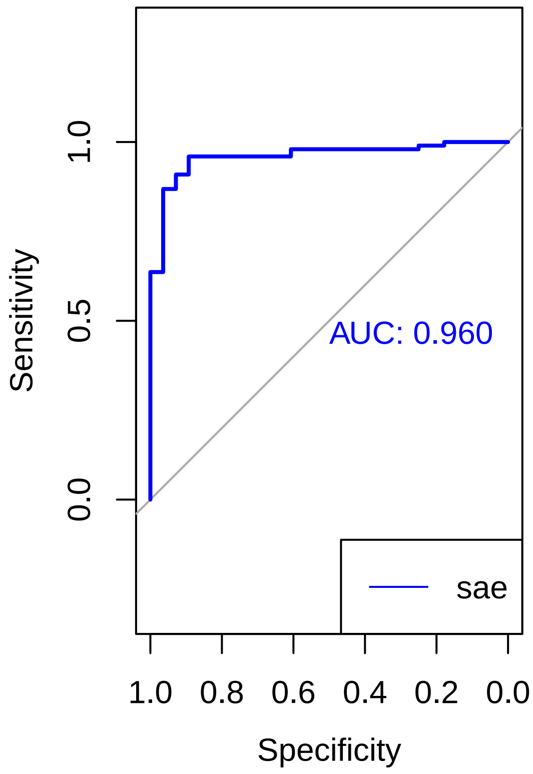
```
plot(roc_sae_test2, col = "blue", print.auc=TRUE)
legend("bottomright", legend = c("sae"), lty = c(1), col = c("blue"))
```



Segun este modelo y el valor obtenido de auc: 0.918, obtenemos que este modelo tiene una precisión alta para nuevos valores.

```
par(mfrow = c(1,2))
plot(roc_sae_test, col = "blue", print.auc=TRUE)
legend("bottomright", legend = c("sae"), lty = c(1), col = c("blue"))
plot(roc_sae_test2, col = "blue", print.auc=TRUE)
legend("bottomright", legend = c("sae"), lty = c(1), col = c("blue"))
```





```
par(mfrow = c(1,1))
```

Vemos que en los dos modelos obtenemos valores similares de AUC.

10. Discuss the results of the analysis.

Primero de todo tenemos el modelo del apartado 4, donde despues de generar los 3 autoencoders tenemos 995671 parametros. En este modelo, al compilarlo y entrenarlo con 15 "epochs" y un "batch size" de 64, obtenemos que al evaluarlo, el valor de la perdida es de 0.40 y la precision de 0.85. Con esto, podríamos decir que este modelo predice bastante bien.

En el apartado 6 comparamos diferentes capas y vemos como la diferencia entre los modelos es muy baja.

Finalmente, tenemos el modelo combinando el modelo generado con el gene y el modelo generado con el protein, extraido de un ejemplo de clase. En este modelo, la precisión aumenta, aunque la pérdida también lo hace. De todos modos, visto desde un punto de vista estadístico, no vemos una diferencia significativa ya que la precisión que hemos obtenido quedaría dentro del intervalo de confianza de la precisión del primero.

Como reflexión final, hemos visto que los autoencoders nos permiten reducir muchísimo el número de parámetros con los que la red neuronal densa va a trabajar. Al ejecutar el autoencoder, calculamos los pesos y después los congelamos, por lo que a partir de ahí, podemos conseguir trabajar en una dimensionalidad muchísimo más baja y, como hemos visto, obteniendo valores de precisión relativamente altos (cercanos al 90%).