Análisis de Supervivencia Alta Dimensionalidad

VII Jornadas de Usuarios de R
Salamanca

Jesús Herranz Valera
jesus.herranz@imdea.org
Instituto IMDEA Alimentación

6 de noviembre de 2015

Análisis de Supervivencia. Alta Dimensionalidad

Análisis Supervivencia

Modelos Predictivos

Machine Learning

Selección de Variables

Alta Dimensionalidad

Regresión Penalizada. Lasso

Random Survival Forest

Índice

- ✓ **Bibliografía**
- ✓ **Análisis de Supervivencia. Regresión de Cox**
- ✓ **Regresión de Cox Penalizada. Lasso**
- ✓ **Paquete *glmnet***
- ✓ **Random Survival Forest**
- ✓ **Paquete *randomForestSRC***

Bibliografía

- Verweij, P. J. M., van Houwelingen H.C*. "Penalized likelihood in Cox regression",* Statistics in Medicine. 1994
- Simon, N., Friedman, J., Hastie, T., Tibshirani, R. "*Regularization Paths for Cox's Proportional Hazards Model via Coordinate Descent*", Journal of Statistical Software 2011

- H. Ishwaran and U. B. Kogalur, E. H. Blackstone and M. S. Lauer. "*Random survival forests*". Annual Applied Statistics, 2008
- H. Ishwaran and U. B. Kogalur. "*Random survival forest for R*". Rnews, 2007
- H. Ishwaran, U. B., Kogalur, X. Chen, A. J. Minn. "*Random Survival Forests for High-Dimensional Data".* Stat Analysis and Data Mining, 2010
- H. Ishwaran, U. B., Kogalur, E. Gorodesky, M.S. Lauer. *"High-Dimensional Variable Selection for Survival Data".* J Am Stat Assoc, 2010
- Svetnik, V., Liaw, A., Tong, C. and Wang, T., "*Application of Breiman's Random Forest to Modeling Structure-Activity Relationships of Pharmaceutical Molecules*", MCS 2004
- Ehrlinger J., Rajeswaran J. and Blackstone E.H. *"ggRandomForests: Exploring Random Forest Survival"* (Vignettes del paquete)

Sobre este taller ……

- Se va a trabajar con **2 ficheros de datos:**
  - “BreastCancerGenes” para la presentación (n=295, p=4948)
  - “pwbc” es el fichero de datos para los ejercicios (n=194, p=34)

- Hay varios **Scripts**
  - “1_1 Taller Supervivencia Alta Dimensionalidad LASSO.r”
  - “1_2 Taller Supervivencia Alta Dimensionalidad RSF.r”

  - “2_1 Taller Sup Alta Dim Ejercicios Para Resolver.r”
  - “2_2 Taller Sup Alta Dim Ejercicios Resueltos.r”

Fichero de datos: BreastCancerGenes

- **Objetivo del estudio:** analizar los **genes** que pueden influir en **supervivencia** de **295 pacientes** de cáncer de mama. (Dutch Cancer Institute NKI de Amsterdam)

- El fichero contiene **4948 variables**:
  - **Variable respuesta: Estado vital** (event_death) y **tiempo hasta la muerte** o último seguimiento (survival.detah.)
  - **Variables clínicas categóricas:** Chemo, Hormonal, Mastectomy, Grade_3_classes (grado histlógico), pN_3_classes (invasión muscular), ER (nivel de estrógeno)
  - **Variable clínicas continuas**: Age.years. , Lymph_node_number_postive , diameter.mm.
  - **Variables genéticas**: datos de expresión de **4919 genes**

- Ejemplo extraído de *"Dynamic Prediction in Clinical Survival Analysis". Hans C. van Houwelingen, Hein Putter*

Fichero de datos: pwc

- **Objetivo del estudio:** analizar la influencia de las medidas de los núcleos de células de tumor, obtenidas por análisis digital de imagen en la **recaída** de **198 pacientes** de cáncer de mama invasivo

- El fichero contiene **34 variables**:
  - **Variable respuesta: recaída** (status) con códigos “R” y “N” y **tiempo hasta la recaída** o último seguimiento (time)
  - **30 variables** de las medidas de las células
  - **2 variable clínicas**: tamaño del tumor y número de nodos

- Ejemplo extraído del paquete de R “*TH.data*”

Análisis de Supervivencia

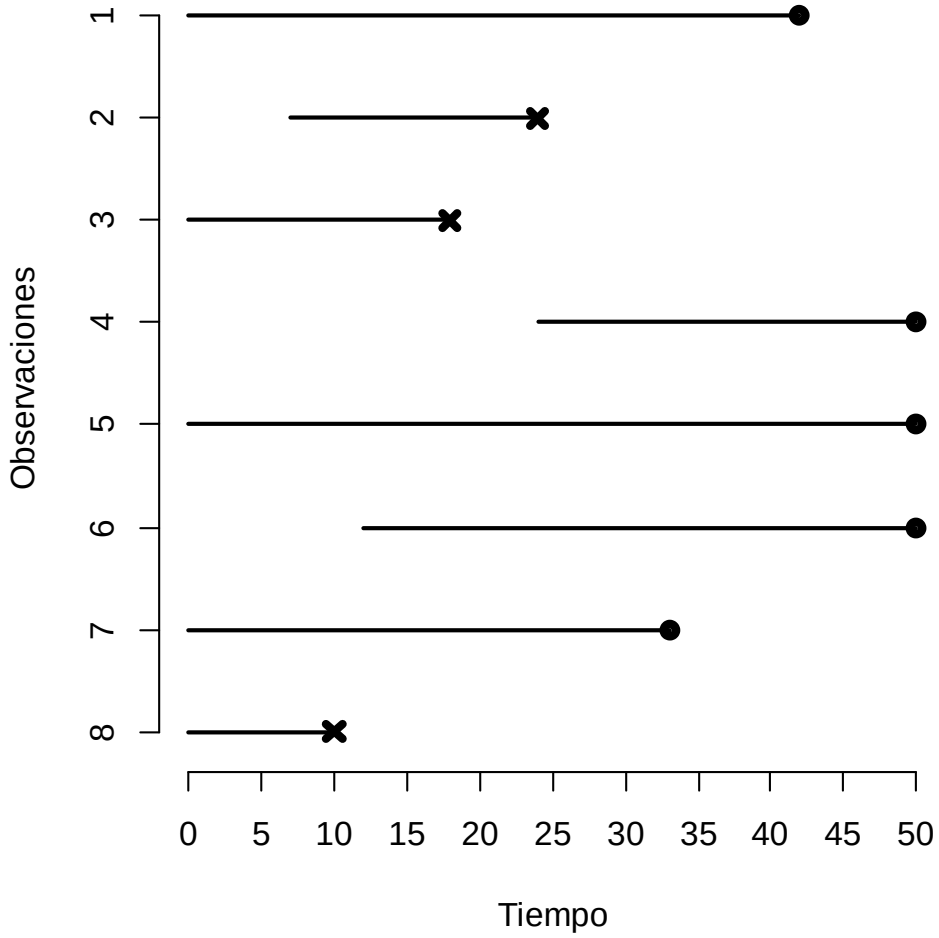
- En el análisis de supervivencia la **variable respuesta** a analizar es el **tiempo hasta que ocurre un evento de interés**
  - **Tiempo de supervivencia**: tiempo entre la incorporación al estudio y la fecha en la que ha ocurrido el evento
  - **Observaciones censuradas**: individuos para los que no ha ocurrido el evento

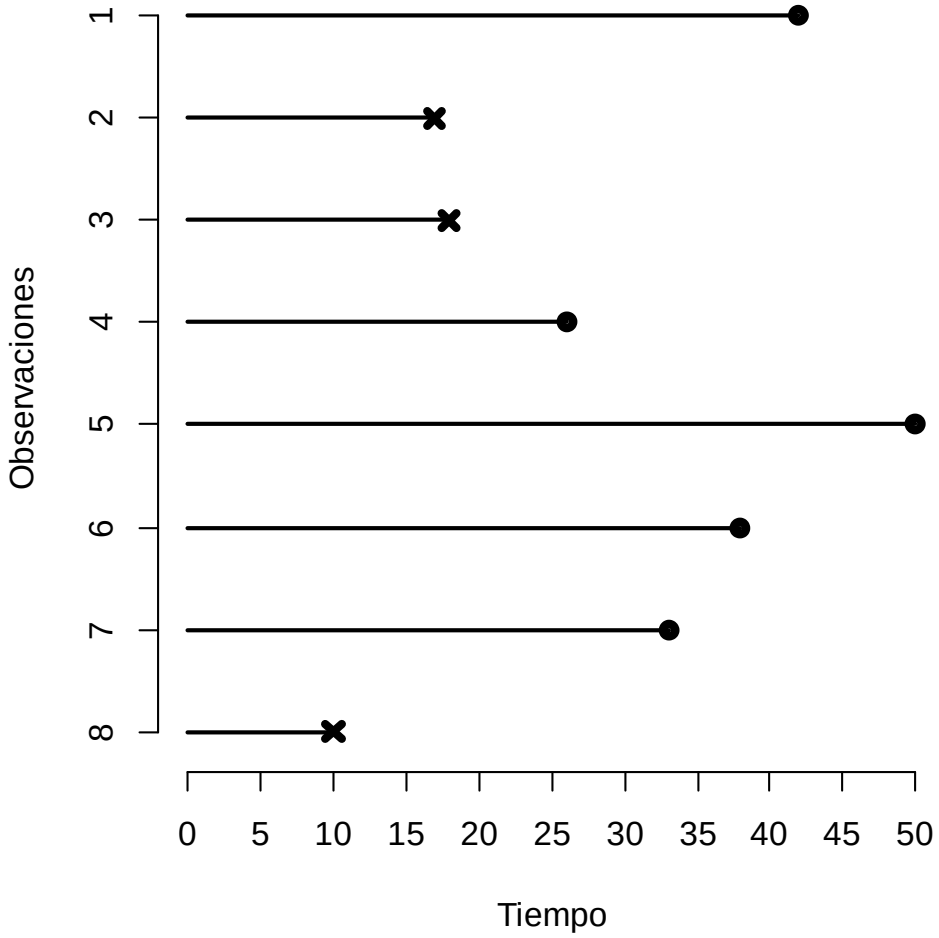
- La **variable respuesta** tiene dos componentes:
  - variable **binaria** llamada **estado (status)**
    - 1 si ha ocurrido el evento, 0 si es una observación censurada
  - **tiempo de supervivencia** que coincide con **el tiempo de seguimiento** para las observaciones censuradas

- **Función de supervivencia** es la probabilidad de que un individuo sobreviva durante un tiempo superior a t

$$S(t) = \Pr ob(T > t)$$

Análisis de Supervivencia



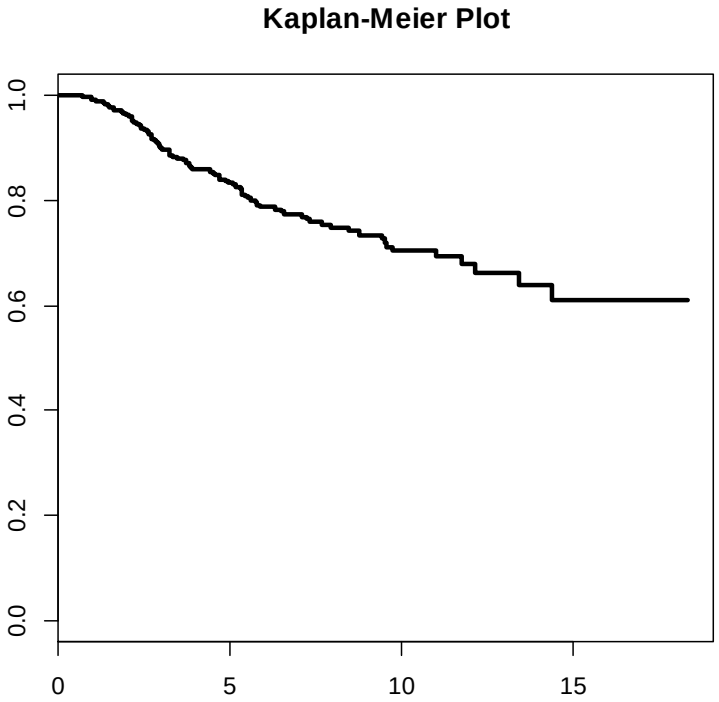


x – evento
o – censurado

tiempo de supervivencia
tiempo de seguimiento

Análisis de Supervivencia

- El **estimador de Kaplan-Meier** estima la función de supervivencia
  - Es una función **decreciente**
  - Es **escalonada** porque estima la supervivencia en los instantes en los que ha ocurrido el evento



- También es de interés la **función de riesgo acumulado** (CHF), que es creciente

$$H(t) = -\ln S(t)$$

- El **test log-rank** se utiliza para analizar si hay diferencias significativas entre dos o más grupos
  - Compara el **número de eventos observados** en cada uno de los grupos con el **número de eventos esperados**, en los tiempos en los que ha ocurrido algún evento

Regresión de Cox

- El **modelo de regresión de Cox** o modelo de **riesgos proporcionales** modela la función de riesgo en función de unas variables predictoras

$$h(t;X) = h_0(t) \cdot e^{\beta_1 X_1 + ... + \beta_p X_p}$$

- La **función de riesgo** es una medida del riesgo de fallecer en cada instante

- El modelo de regresión de Cox es **semiparamétrico**
  - Se estiman los parámetros β_j del modelo, pero la forma de **$h_0(t)$ no se especifica**
  - Se usa el método de verosimiltud parcial para estimar los coeficientes

- El **Hazard Ratio** (HR), exp(β) compara el riesgo entre 2 grupos
- **Principio de riesgos proporcionales**: el HR entre dos individuos con perfiles distintos es constante durante todo el tiempo de seguimiento

Regresión de Cox. Predicciones

- El **modelo de regresión de Cox** permite predecir la función de supervivencia para un individuo que presente unos determinados valores en las variables predictoras

- Se suele llamar **Risk Score** al predictor lineal del modelo
  - Permite obtener una predicción numérica por individuo
  - **Valores altos del Risk Score** significan mayor riesgo de que se produzca el evento, y por tanto, **peor supervivencia**

$$\text{Risk Score} = \text{Pred. Lin.} = \beta_1 X_1 + ... + \beta_p X_p$$

Capacidad Predictiva. C-index

- El **C-index** es una medida de concordancia que se usa para evaluar la **capacidad predictiva** de un **modelo de supervivencia**
  - Es una **generalización del AUC** para datos de supervivencia

- El **C-index** se define como la proporción de pares de observaciones para los cuales el orden de los tiempos de **supervivencia observados** y las **predicciones** del modelo son **concordantes**
  - Se omiten aquellos pares en los que el tiempo más corto es censurado

- **Observación:** las funciones que calculan el **c-index** comparan supervivencias observadas con supervivencias predichas. En los casos en los que se tiene un **risk score**, se puede utilizar con **signo negativo**

- Hay extensiones de las **curvas ROC** a análisis de supervivencia, que analizan la capacidad predictiva de un modelo en **tiempos fijos**
  - Paquetes de R: *SurvivalROC* y *risksetROC*

Regresión de Cox. Variables Clínicas

```
> library(survival)
> library(caret)
> ## Fichero Datos: Breast Cancer Genes
> yy=read.delim("C://Taller Supervivencia Alta Dimensionalidad/BreastCancerGenes.txt",
sep="\t")
> dim(yy)
[1]  295 4948
> ## Regresión de Cox. Modelo Multivariante Variables Clínicas
> cox.clin <- coxph( Surv(survival.death. , event_death) ~ Chemo + Hormonal +
+                    Mastectomy + Grade_3_classes + pN_3_classes + diameter.mm. +
+                    Lymph_node_number_postive + Age.years. + ER , data=yy )
> cox.clin
                             coef exp(coef) se(coef)      z      p
ChemoYes                  -0.4628     0.630   0.3634 -1.273 0.2000
HormonalYes               -0.4461     0.640   0.4588 -0.972 0.3300
MastectomyYes              0.0865     1.090   0.2458  0.352 0.7200
Grade_3_classesPoorly diff 0.2891     1.335   0.2793  1.035 0.3000
Grade_3_classesWell diff  -1.5359     0.215   0.5425 -2.831 0.0046
pN_3_classes1-3           -0.6910     0.501   0.5477 -1.261 0.2100
pN_3_classespN0           -0.9655     0.381   0.7460 -1.294 0.2000
diameter.mm.               0.0191     1.019   0.0131  1.457 0.1500
Lymph_node_number_postive -0.0174     0.983   0.0970 -0.179 0.8600
Age.years.                -0.0391     0.962   0.0203 -1.928 0.0540
ERPositive                -0.8022     0.448   0.2536 -3.164 0.0016
```

- Se ajusta un modelo de **Regresión de Cox** con todas las variables clínicas
- Las variables “ER”, “Grade_3_classes” y “Age” son las más importantes

Métodos de regresión penalizados

- Los **métodos de regresión penalizados** introducen una **penalización** en la función de pérdida a ser optimizada (loss function)
  - Los estimadores “cásicos” en regresión producen **estimadores muy inestables** (valores muy altos y con mucha varianza) en problemas de **alta dimensionalidad** (p>>n) y con **variables correlacionadas**

- Los **estimadores** proporcionados por los métodos de regresión penalizada son **sesgados** pero tienen **menor varianza**
  - Eso dificulta la obtención de errores estándares de los coeficientes, de intervalos de confianza y de test específicos sobre cada coeficiente

- Son **modelos lineales**, pero utilizan un método diferente en la estimación de los coeficientes de regresión

Métodos de regresión penalizados. Lasso

- **LASSO** (“least absolute shrinkage and selection operator”) usa la restricción **L1-penalty**: la suma de los valores absolutos de los coeficientes de regresión debe ser menor que una constante C

- En un problema de regresión de Cox, **Lasso** maximiza la función

$$\mathrm{pl}_{\mathrm{lasso}}(\beta) = \mathrm{pl}(\beta) - \lambda \cdot \sum_{j=1}^{p} \left| \beta_j \right|$$

pl es la función de verosimilitud parcial

  - **Lambda** es un parámetro que determina el peso de la penalización
  - Cuánto mayor es lambda los coeficientes son más pequeños
  - Si λ=0 se obtiene el modelo regresión de Cox

Métodos de regresión penalizados. Lasso

- **Lambda** se considera una **parámetro de tuning** del método y se suele optimizar con **validación cruzada** (CV)
  - Se usa una medida que se llama la **cross-validated likelihood** (CVL) que se basa en la verosimilitud evaluada en las particiones de testing en un proceso de CV

- Muchos de los **coeficientes** estimados por Lasso alcanzan el valor 0, y solo **unos pocos** quedan con valores **distintos de 0**

- **Lasso** puede ser usado como un método de **selección de variables** y es adecuado en problemas de **alta dimensionalidad**

- Las **variables** se deben utilizar **estandarizadas**, ya que la penalización afecta al conjunto de los coeficientes

Métodos de regresión penalizados

- **Ridge Regression** usa la restricción **L2-penalty**

$$pl_{ridge}(\beta) = pl(\beta) - \lambda \cdot \sum_{j=1}^{p} \beta_j^2$$

pl es la función de verosimilitud parcial

  - El modelo se compone de todas las variables

- **Elastic Net** combina las 2 penalizaciones

$$pl_{elastic-net}(\beta) = pl(\beta) - \lambda \cdot \left[(1-\alpha) \cdot \sum_{j=1}^{p} \beta_j^2 + \alpha \cdot \sum_{j=1}^{p} \left| \beta_j \right| \right]$$

  - **alpha** es un parámetro de compromiso entre Ridge (α=0) y Lasso (α=1)
  - Muchos coeficientes son 0 como en Lasso

Paquete *glmnet* de R

- La función ***glmnet()*** permite ajustar **modelos de regresión penalizados** de regresión **lineal, logística, de Cox y de Poisson**

- **Parámetros**
  - **x**: matriz con el conjunto de predictores
  - **y** : variable respuesta
  - **penalty.factor** : controla si se quiere incluir predictores sin penalizar
  - **alpha** : **Lasso** con alpha=1 (por defecto) , **Ridge** con alpha=0 y **Elastic Net** con alphas intermedios
  - **lambda:** Un valor de lambda o una secuencia de lambdas
  - **nlambda:** Número de valores de lambda (por defecto, 100)
  - **family :** "gaussian", "binomial", **"cox",** "poisson"
  - **standardize = T** : se usan variables estandarizadas (por defecto TRUE)

  - Comentario: el argumento "*data=*" a veces da problemas

Paquete *glmnet* de R. Optimización de parámetros

- La función ***cv.glmnet()*** permite evaluar los **modelos de regresión penalizados** con **validación cruzada** para optimizar el parámetro **lambda**

- **Parámetros**
  - **x**: matriz con el conjunto de predictores
  - **y** : variable respuesta
  - **lambda, nlambda:** Secuencia o número de lambdas
  - **alpha, family, standardize, … :** Parámetros de la función *glmnet()*

  - **nfolds** : Número de particiones (folds) en la CV (por defecto = 10)
  - **foldid :** Vector con valores de las folds si se han asignado
  - **type.measure :** Tipo de medida a evaluar con CV, según la variable respuesta. Valores: “mse”, “deviance”, “auc”, “class”

    Para **regresión de Cox** solo está **“deviance”** disponible
  - **parallel :** Para ejecución en paralelo (por defecto, FALSE)

Lasso. Preparación de los datos

```
> yy$Chemo      = as.numeric ( yy$Chemo == "Yes" )
> yy$Hormonal   = as.numeric ( yy$Hormonal == "Yes" )
> yy$Mastectomy = as.numeric ( yy$Mastectomy == "Yes" )
> yy$ER         = as.numeric ( yy$ER == "Positive" )
>
> ## Variables Dummy para "pN_3_classes"
> table(yy$pN_3_classes)
>=4 1-3 pN0
 38 106 151
> yy$pN_3_classes.1 = as.numeric ( yy$pN_3_classes == "pN0" )
> yy$pN_3_classes.2 = as.numeric ( yy$pN_3_classes == "1-3" )
> yy$pN_3_classes.3 = as.numeric ( yy$pN_3_classes == ">=4" )
>
> ## Variables Dummy para "Grade_3_classes"
> table(yy$Grade_3_classes)
Intermediate  Poorly diff    Well diff
         101          119           75
> yy$Grade_3_classes.1 = as.numeric ( yy$Grade_3_classes == "Well diff" )
> yy$Grade_3_classes.2 = as.numeric ( yy$Grade_3_classes == "Intermediate" )
> yy$Grade_3_classes.3 = as.numeric ( yy$Grade_3_classes == "Poorly diff" )
```

- La función ***glmnet()*** solo trabaja con variables numéricas. Las variables categóricas definidas como factores deben ser transformadas en **variables dummy**
- Las variables categóricas binarias se convierten en numéricas y se definen 3 variables dummy para las variables “pN_3_classes” y “Grade_3_classes”

Lasso. Preparación de los datos

```
> ## Data Frame
> xx = yy [ , c ( "ID", "event_death", "survival.death.",
+                 "diameter.mm.", "Lymph_node_number_postive", "Age.years.",
+                 "Chemo", "Hormonal", "Mastectomy", "ER",
+                 "Grade_3_classes.1", "Grade_3_classes.2", "Grade_3_classes.3",
+                 "pN_3_classes.1", "pN_3_classes.2", "pN_3_classes.3" ) ]
>
> ## Se añaden las variables genéticas
> xx = merge ( xx, yy[ , c( 1, 30:4948) ] , by="ID" )
> xx = xx [ , - 1 ]     ## se quita el ID
> dim(xx)
[1]  295 4934
> ## Vectores que contienen los número de las columnas de cada tipo de predictor
> col.cli = 3:15
> col.gen = 16:4934
> col.all = 3:4934
```

- Se forma un **data frame** que contiene exclusivamente la **variable respuesta** (estado y tiempo) y las **variables predictoras** (para poder usar formulas del tipo **~ .**)
- Se quita el ID para que no se utilice como un predictor
- Primero están las variables clínicas y luego las genéticas
- Se crean vectores con los números de las **columnas** de cada tipo de variables predictoras: clínicas, genéticas y todas

Lasso. Muestras de training y testing

```
> ## Muestra de Training y Muestra de Testing
> set.seed (444)
> ind.train = createDataPartition ( factor(xx$event_death) , p = 2/3 )$Resample1
> set.seed (NULL)
> xx.train = xx [ ind.train , ]
> xx.test  = xx [ - ind.train  , ]
> rm(xx)
> n.all = nrow ( xx.train )
> dim(xx.train)
[1]  197 4934
> table ( xx$event_death )
  0   1
216  79
> table ( xx.train$event_death )
  0   1
144  53
> table ( xx.test$event_death )
 0  1
72 26
```

- Se crean las **muestras de training y test**, reservando 1/3 para ésta última
- Se fija una semilla para que se puedan reproducir los resultados
- Se utiliza la función ***createDataPartition()*** del paquete ***caret*** para que la proporción de eventos y observaciones censuradas en las 2 muestras sea proporcional al conjunto de datos (partición aleatoria estratificada). Hay que introducir la variable como factor

Regresión de Cox. Alta Dimensionalidad

```
> coxph( Surv(survival.death. , event_death) ~ . , data = xx.train [ , 1:100 ] )
                              coef exp(coef) se(coef)         z   p
diameter.mm.                 0.657  1.93e+00    12517  5.25e-05   1
Lymph_node_number_postive    0.562  1.75e+00   120384  4.67e-06   1
Age.years.                   0.546  1.73e+00    24939  2.19e-05   1
Chemo                       -0.187  8.29e-01   476368 -3.93e-07   1
Hormonal                     3.484  3.26e+01   621560  5.61e-06   1
Mastectomy                  -6.438  1.60e-03   263941 -2.44e-05   1
ER                         -20.111  1.85e-09   589267 -3.41e-05   1
Grade_3_classes.1          -21.999  2.79e-10   470703 -4.67e-05   1
Grade_3_classes.2          -32.585  7.05e-15   369866 -8.81e-05   1
. . .
. . .
J00129                      11.303  8.10e+04   464310  2.43e-05   1
Contig29982_RC              -1.720  1.79e-01   804179 -2.14e-06   1
Contig42854                 -4.306  1.35e-02  1255020 -3.43e-06   1
Contig42014_RC              53.695  2.09e+23  1148528  4.68e-05   1
. . .
. . .
NM_003064                   77.888  6.71e+33  1005375  7.75e-05   1
NM_002336                   -9.873  5.16e-05   780172 -1.27e-05   1
NM_002337                  -71.999  5.38e-32        0      -Inf   0
NM_003066                 -113.703  4.16e-50        0      -Inf   0
```

- El modelo de Cox con 197 observaciones y 100 variables predictoras obtiene **estimaciones muy altas** para los coeficientes de regresión y sus errores estándares

Lasso. Lambda óptimo con CV. Modelo genético

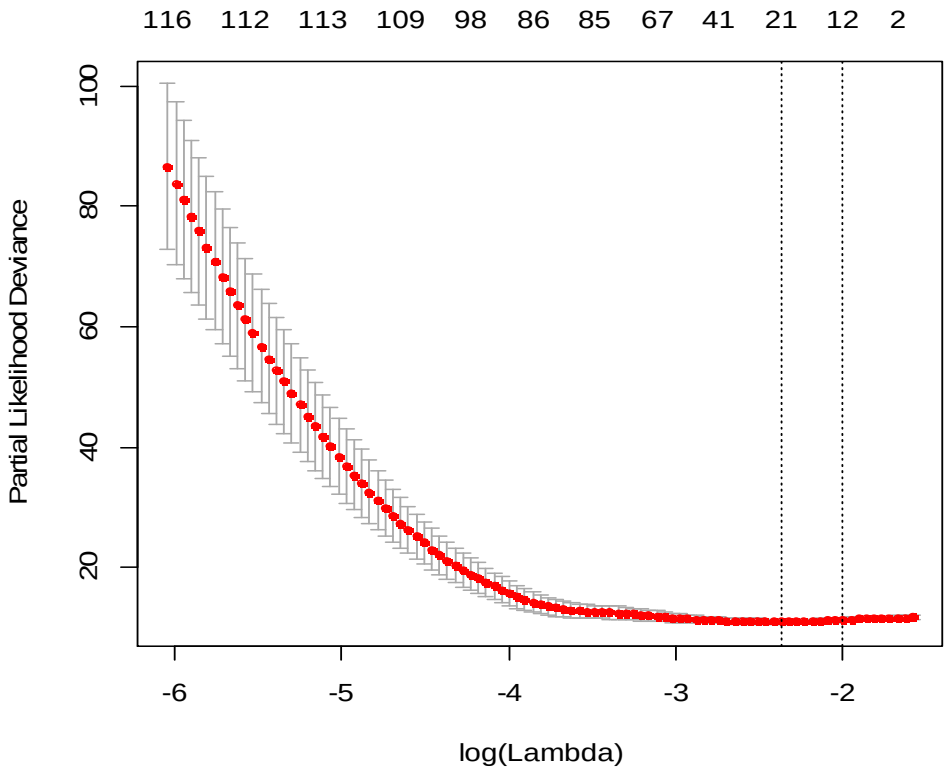
```
> library(glmnet)
> library(rms)
> library(doParallel)                      ## library ( doMC ) ## Linux
> registerDoParallel( detectCores()-1 )  ## registerDoMC( detectCores()-1 ) ## Linux
> ## Lasso - CV para seleccionar el lambda óptimo
> lasso.cv = cv.glmnet( as.matrix ( xx.train [ , col.gen ] ),
+                       Surv ( xx.train$survival.death. , xx.train$event_death ),
+                       family="cox", nfolds = 10, alpha=1, standardize=T, parallel=T )
> ## Lambdas analizados
> head(lasso.cv$lambda)
[1] 0.2064230 0.1970407 0.1880849 0.1795362 0.1713760 0.1635867
> length(lasso.cv$lambda)
[1] 97
```

- Se cargan las librerías necesarias y se declaran los cores para **la paralelización** en Windows (en Linux se usa la librería ***doMC*** y la función ***registerDoMC()***)
- Se busca el **lambda óptimo** con **validación cruzada**, 10-fold CV (*nfolds=10*), usando la función ***cv.glmnet()*** con paralelización (*parallel=T*)
- Se construye un modelo **de regresión de Cox penalizado** (*family="cox"*) en el conjunto de training con **Lasso** (*alpha=1*) con las variables **estandarizadas**, usando solo las **variables genéticas,** que se introducen como **una matriz**, ya que no admite un data frame (se usa la función *as.matrix()*)
- La función ha probado 97 lambdas, seleccionados de forma automática

Ejemplo: Lambda óptimo. Lasso

```
> ## Gráfico de los Errores frente a los lambdas
> dev.new() ; plot(lasso.cv)
> ## Media, SD, cvm-cvsd, cvm+cvsd de los Errores de la CV
> head(lasso.cv$cvm)
[1] 11.74306 11.72384 11.69067 11.65508 11.61774 11.57958
> head(lasso.cv$cvsd)
[1] 0.3061598 0.3047625 0.3040338 0.3037648 0.3040257 0.3059663
> head(lasso.cv$cvlo)
[1] 11.43690 11.41908 11.38664 11.35131 11.31372 11.27362
> head(lasso.cv$cvup)
[1] 12.04922 12.02861 11.99471 11.95884 11.92177 11.88555
```

- El gráfico muestra **la desvianza** de la verosimilitud parcial, obtenidas **en la CV** (objeto *$cvm*) frente a los lambdas analizados (escala log)
- Las barras de error representan los valores de **Mean-SD** y y **Mean+SD** (objetos *$cvlo* y *$cvup)*
- Se marcan 2 lambdas óptimos, el que tiene la **mínima desvianza** y el que cumple la **regla 1-SE,** "one-standard error"



Ejemplo: Lambda óptimo. Lasso

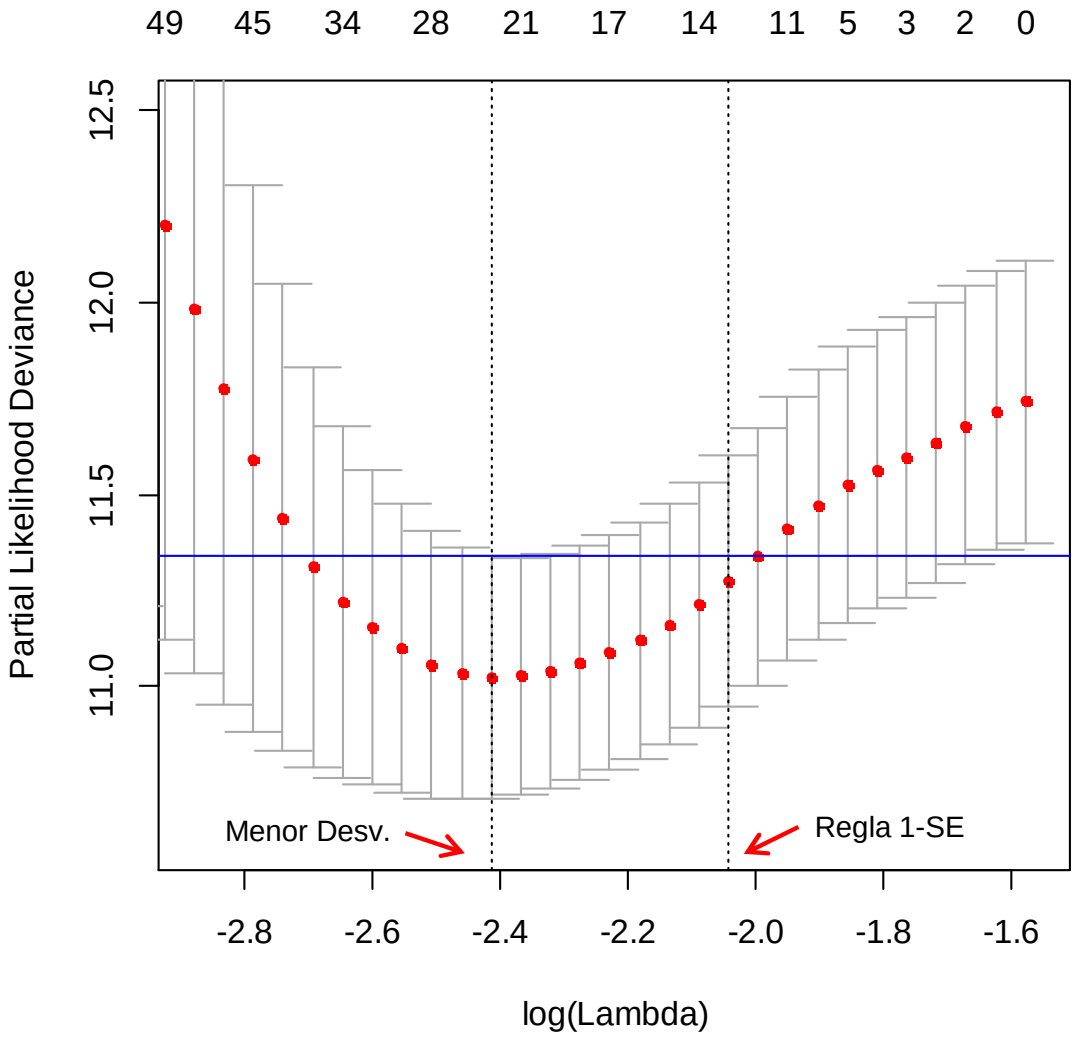
```
> ## Lambda óptimo con Desvianza Mínima
> lasso.cv$lambda.min
[1] 0.09361032
> min ( lasso.cv$cvm )
[1] 11.05085
> ind.min = which.min ( lasso.cv$cvm )          ## Posición del mínimo en el vector
> lasso.cv$lambda [ ind.min ]                   ## Comprobación
[1] 0.09361032
> ## Lambda óptimo con la regla 1-SE
> lasso.cv$lambda.1se
[1] 0.1358125
> lasso.cv$cvup [ ind.min ]                     ## Mean+SE del Mínimo Error
[1] 11.42031
> ind1se = which ( lasso.cv$cvm < lasso.cv$cvup [ ind.min ] ) [1] ## Modelo más simple
[1] 10
> lasso.cv$lambda [ ind1se ]                     ## Comprobación
> [1] 0.1358125
> lasso.cv$cvm [ ind1se ]
[1] 11.35538
```

- La función *cv.glmnet()* devuelve 2 **lambdas óptimos**, el que tiene **mínima desvianza** (*$lambda.min*) y el que cumple la **regla 1-SE** (*$lambda.1se*) , que es el modelo más simple (con menos variables, mayor penalización lambda) con un error menor que “Error+SD” del Mínimo
- La desvianza mínima es 11.05 y la del modelo con lambda correspondiente a la regla 1-SE aumenta muy poco, 11.36

Ejemplo: Lambda óptimo. Lasso

```
> ## Gráfico de las Desvianzas frente a los lambdas en los intervalos de interés
> dev.new()
> plot(lasso.cv, xlim = c( log(0.056) , log(0.21) ) , ylim = c( 10.6 , 12.5 ) )
> abline ( h = lasso.cv$cvup [ind.min] , col="blue" )
```

- Se muestra el mismo gráfico de la desvianza frente a lambda en el intervalo de interés
- Se ha añadido una línea azul con el “Error+SD” del modelo con Mínimo MSE
- En la parte superior del gráfico, se indica el número de variables con coeficientes distintos de cero
- Se observa que el modelo que cumple la **regla 1-SE** es el más simple (menos variables) con error menor que “Error+SD” del Mínimo



Ejemplo: Lambda óptimo. Modelo

```
> ## Modelo Lasso con el lambda óptimo
> coef.opt = coef ( lasso.cv, s = "lambda.min" )
> length(coef.opt)
[1] 4919
> head(coef.opt)
J00129         .
Contig29982_RC .
Contig42854    .
Contig42014_RC .
Contig27915_RC .
> coef.opt = as.matrix(coef.opt) ## Se convierte a matriz para extraer bien los nombres
> row.names( coef.opt )[ coef.opt != 0 ]
 [1] "Contig67229_RC" "Contig55111_RC" "Contig46084_RC" "NM_013290"
 [5] "NM_013332"      "NM_006054"      "NM_006096"      "Contig64940_RC"
. . .
[21] "NM_001124"      "NM_001355"
> round ( coef.opt [ coef.opt != 0 ] , 3 )
 [1] -0.177  0.332  0.095  0.398  0.251  0.906  0.683  0.154  0.131 -0.010  0.065
[12]  0.371  0.554  0.535  0.828  1.342 -0.221 -0.096  0.209  0.770  0.151  0.145
```

- La función ***coef()*** devuelve los coeficientes de todas las variables del modelo en su escala original, incluidas las que Lasso no ha seleccionado (coef=0). Se eligen los coeficientes del modelo con el **lambda óptimo** (parámetro *s="lambda.min"*)
- Se seleccionan los coeficientes distintos de 0 y los nombres de esas variables. El modelo con el **lambda óptimo** incluye **22 variables**. No hay intercept porque es Regresión de Cox

Ejemplo: Lasso. Predicciones

```
> head( predict( lasso.cv, s ="lambda.min", newx = as.matrix( xx.train [ , col.gen ])))
             1
4  -0.37616449
5  -0.57559228
 . .
> pred.train = predict.cv.glmnet( lasso.cv, s = "lambda.min" ,
+                         newx = as.matrix ( xx.train [ , col.gen ] ) )  [ ,1 ]
> pred.test  = predict.cv.glmnet( lasso.cv, s = "lambda.min" ,
+                         newx = as.matrix ( xx.test [ , col.gen ] ) )   [ ,1 ]
> ## C-index
> rcorr.cens( - pred.train,
+             Surv ( xx.train$survival.death. , xx.train$event_death ))["C Index"]
0.8923293
> rcorr.cens( - pred.test,
+             Surv ( xx.test$survival.death. , xx.test$event_death ))["C Index"]
0.7299671
```

- Se usa la función ***predict()*** para obtener las **predicciones**. Devuelve una matriz con una columna, que es la que se selecciona. Los valores de las observaciones donde se van a realizar las predicciones se introducen como una matriz
- Por defecto, devuelve las predicciones para *"lambda.1se"*, pero se puede especificar *"lambda.min"* o cualquier otro lambda que se desee (parámetro *s*=)
- El **C-index** está sobreajustado en la muestra de **training 0.892** (error aparente). El de la muestra de **testing, 0.73** es una estimación "honesta" del error

Ejemplo: Modelo combinado

```
> lasso.cv.all = cv.glmnet( as.matrix ( xx.train [ , col.all ] ),
+                           Surv ( xx.train$survival.death. , xx.train$event_death ),
+                           penalty.factor = c ( rep( 0, length(col.cli) ),
+                                                rep( 1, length(col.gen) ) ) ,
                          family="cox", nfolds = 10, alpha=1, standardize=T, parallel=T)
>
> coef.opt.all = coef ( lasso.cv.all, s = "lambda.min" )
> coef.opt.all = as.matrix(coef.opt.all)
> row.names( coef.opt.all )[ coef.opt.all != 0 ]
 [1] "diameter.mm."               "Lymph_node_number_postive" "Age.years."
 [4] "Chemo"                      "Hormonal"                  "Mastectomy"
 [7] "ER"                         "Grade_3_classes.1"         "Grade_3_classes.2"
[10] "Grade_3_classes.3"          "pN_3_classes.1"            "pN_3_classes.2"
[13] "pN_3_classes.3"             "Contig55111_RC"            "NM_021025"
[16] "NM_006054"                  "NM_004950"                 "NM_016109"
[19] "AL137347"                   "U18919"                    "U79277"
[22] "Contig58368_RC"             "NM_017709"                 "L27560"
[25] "M24895"
```

- Se puede imponer la **penalización de los coeficientes** solo en las variables que se desee, con el parámetro *penalty.factor*=. Se carga un vector de 0 y 1 donde el valor 0 significa que no se va a penalizar, y es el que se asigna a las **13 variables clínicas**
- Por tanto, construimos un **modelo combinado** de variables genéticas que incluye obligatoriamente las 13 variables clínicas
- Lasso selecciona además **12 variables genéticas**

Ejemplo: Modelo combinado

```
> round ( coef.opt.all [ coef.opt.all != 0 ] , 3 )
 [1]  0.023 -0.046 -0.034 -0.249 -0.195  0.113 -0.704 -2.190 -0.356  0.218 -0.241
[12] -0.147  0.246  0.196 -0.167  0.082 -0.098  0.179  1.698  0.616  0.272  0.613
[23] -0.125  0.149 -0.024
>
> ## Predicciones
> pred.train.all = predict.cv.glmnet( lasso.cv.all, s = "lambda.min" ,
+                       newx = as.matrix ( xx.train [ , col.all ] ) )  [ ,1 ]
> pred.test.all  = predict.cv.glmnet( lasso.cv.all, s = "lambda.min" ,
+                       newx = as.matrix ( xx.test [ , col.all ] ) )   [ ,1 ]
>
> ## C-index
> rcorr.cens( - pred.test.all,
+             Surv ( xx.test$survival.death. , xx.test$event_death ))["C Index"]
  C Index
0.7755214
```

- Se obtienen los coeficientes y las predicciones de forma habitual
- El **C-index** en la muestra de testing con el **modelo combinado** que incluye también las variables clínicas es **0.776** superior a 0.73 del modelo que solo incluía variables genéticas

Ejemplo: Elastic Net

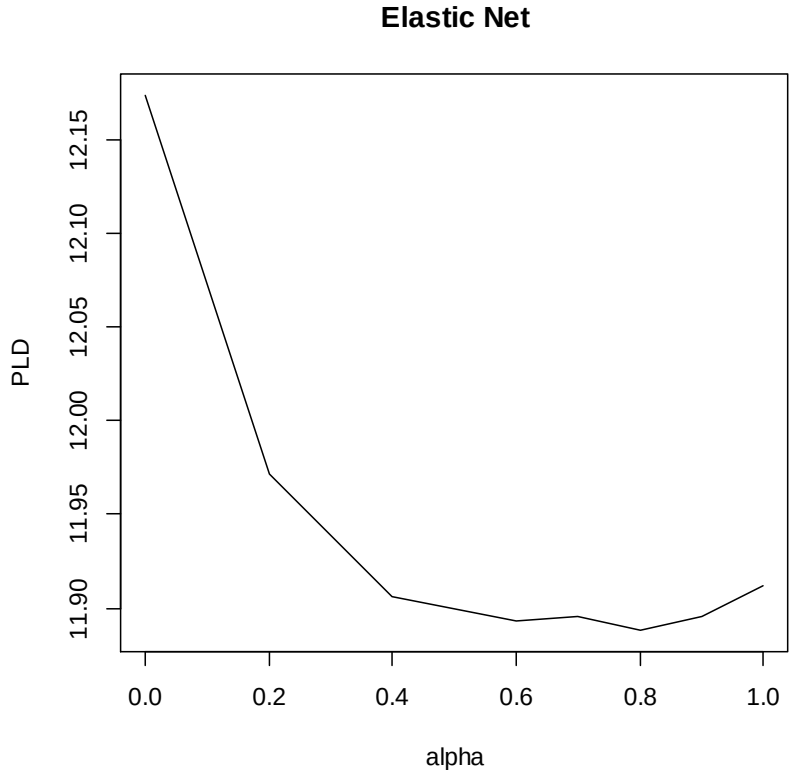
```
> ## Elastic Net
> set.alpha = c( 0, 0.2, 0.4, 0.6, 0.7, 0.8, 0.9, 1 ) ## Conjunto de alphas a explorar
> res.lambda     = rep ( NA,  length(set.alpha) )
> res.min.pld    = rep ( NA,  length(set.alpha) )
> res.coef.not.0 = rep ( NA,  length(set.alpha) )
> ## La 10-fold CV se realiza con las mismas particiones de la muestra
> w.foldid = sample ( 1:10 , size=nrow(xx.train) , replace = T )
> for ( i in 1:length(set.alpha) )
+ {
+   ## CV para encontrar el lambda óptimo, para el alpha específicado
+   enet.cv = cv.glmnet(as.matrix ( xx.train [ , col.all ] ),
                       Surv ( xx.train$survival.death. , xx.train$event_death ),
                       penalty.factor = c ( rep( 0, length(col.cli) ),
                                            rep( 1, length(col.gen) ) ) ,
                       alpha = set.alpha[i] ,
                       family="cox", foldid = w.foldid, standardize=T  parallel=T )
+   ## Lambda óptimo y mínima PLD obtenida
+   res.lambda[i]  = enet.cv$lambda.min
+   res.min.pld[i] = min ( enet.cv$cvm )
+   ## Número de coeficientes distintos de 0 del modelo con lambda óptimo
+   coef.opt = as.matrix ( coef ( enet.cv, s = "lambda.min" ) )
+   res.coef.not.0[i] = length ( coef.opt [ coef.opt != 0 ]  )
+ }
```

- La función ***cv.glmnet()*** optimiza lambda para **un alpha fijo**. Para optimizar el parámetro de **Elastic Net** hay que programarlo directamente
- Para cada alpha a explorar, se almacena su lambda óptimo y la PLD mínima. Se deben usar las **mismas particiones en la CV** para todos los **valores de alpha**

Ejemplo: Elastic Net

```
> ## Resultados de la CV
> res.lambda
 [1] 26.59028071  0.29321361  0.14660680  0.10239173  0.10090767  0.08829421
 [7]  0.07848374  0.07063537
> res.min.pld
 [1] 12.17321 11.97191 11.90615 11.89272 11.89589 11.88852 11.89542 11.91193
> res.coef.not.0
 [1] 4932  110   75   62   44   46   45   44
> dev.new()
> plot( set.alpha, res.min.pld, main="Elastic Net", xlab="alpha", ylab="PLD", type="l")
```

- La mínima PLD se ha obtenido con **alpha=0.8** pero distintos valores de alpha/lambda pueden proporcionar resultados semejantes
- PLD parece que disminuye para valores de alpha mayores. Con Ridge Regression (alpha=0) se obtienen los peores resultados
- El número de coeficientes distintos de 0 va disminuyendo conforme aumenta alpha, pero no siempre es una relación monótona



Ejemplo: Elastic Net

```
> ## Modelo Elastic Net con parámetros óptimos
> ind.min.enet = which.min ( res.min.pld )      ## Posición la mínima PLD
> alpha.opt.enet  = set.alpha [ ind.min.enet ]
> lambda.opt.enet = res.lambda [ ind.min.enet ]
> ## Modelo
> enet.out = glmnet ( as.matrix ( xx.train [ , col.all ] ),
+                     Surv ( xx.train$survival.death. , xx.train$event_death ),
+                     penalty.factor = c ( rep( 0, length(col.cli) ),
+                                          rep( 1, length(col.gen) ) ) ,
+                     lambda = lambda.opt.enet, alpha = alpha.opt.enet,
+                     family="cox", standardize=T )
> ## Coeficientes del modelo
> enet.coef =  as.matrix ( coef ( enet.out ) )
> row.names( enet.coef )[ enet.coef != 0 ]
 [1] "diameter.mm."              "Lymph_node_number_postive" "Age.years."
. . .
[43] "L27560"                   "D14678"                    "NM_001355"
[46] "M24895"
> ## Predicciones y C-index
> pred.test.enet  = predict( enet.out, s = "lambda.min" ,
+                        newx = as.matrix ( xx.test [ , col.all ] ) )   [ ,1 ]
> rcorr.cens( - pred.test.enet,
+             Surv ( xx.test$survival.death. , xx.test$event_death ))["C Index"]
0.7683864
```

- Se ajusta el modelo de **Elastic Net** con la función ***glmnet()*** y los parámetros óptimos
- El C-index es **0.768**, muy parecido al obtenido con Lasso

Ejercicio

- **Fichero de datos: “pwbc”,** con muestras de training y testing
- **Variable respuesta: “time”** tiempo de supervivencia y **“status”** variable que indica si ha recaído (con código = “R”)

- Construir un modelo de **regresión de Cox penalizado** con **Lasso** en la muestra de training, optimizando lambda (función *cv.glmnet()*)
- Obtener el gráfico del **perfil de la desvianza** respecto a los lambdas
- Obtener el modelo y **los coeficientes**, e interpretarlos
- Obtener las **predicciones** en las muestras de training y testing
- Calcular los **C-index** y compararlos

- Optimizar alpha y lambda para construir un modelo de **regresión de Cox penalizado** con **Elastic Net** en la muestra de training
  - Probar con *set.alpha = seq (0, 1 , by=0.1)*

Random Forest

- **Random Forest (RF)** es una técnica de machine learning que consiste en la **construcción de una gran número de árboles** con las siguientes características:
  - Está basado en **muestras bootstrap**. Cada árbol está basado en una muestra aleatoria con reemplazamiento de las observaciones
  - Cada división del árbol está basada en una **muestra aleatoria de los predictores**
  - Los **árboles no se cortan**, son tan largos como sea posible. No hay prunning

- En la construcción de cada árbol de RF una parte de las observaciones no se utiliza (37% aprox.). Se llama **muestra out-of-bag (OOB)** y es usada para obtener una **estimación honesta** de la capacidad predictiva del modelo

Random Forest

- **Random Forest** se puede usar en problemas de **clasificación y regresión**

- **Random Survival Forest (RSF)** es la extensión de RF para datos de **supervivencia**

- **Propiedades de Random Forest**
  - Analiza eficientemente **un gran número de variables**, sin tener que hacer selección previa
  - Es un **método no paramétrico**, ya que no hace supuestos sobre el modelo. Puede incorporar **relaciones no lineales** e **interacciones**
  - En supervivencia, **no asume el principio de riesgos proporcionales**

- Random Forest tiene dos **parámetros de tuning**: el número de árboles y el número de predictores que son evaluados en cada división

Random Forest

	Clasificación	**Regresión**	**Supervivencia**
Variable Respuesta	Categórica	Continua	Tiempo hasta que ocurre un evento
Criterios de partición	Indice de Gini Entropía	Reducción MSE	Test Log-rank
Predicciones	Clases Probabilidades	Valor Predicho	Funciones de Supervivencia y Hazard
Capacidad Predictiva **OOB Error**	Tasa de Error	Mean Squared Error (MSE)	C-index

Random Survival Forest. Criterio de Partición

- La construcción de cada árbol de RSF se basa en el **particionamiento recursivo**, donde cada nodo es dividido en dos nodos hijos, seleccionando el predictor que **maximiza la diferencia en supervivencia** en los nodos hijos

- **Test log-rank**
  - El test log-rank es el método estándar para evaluar si hay **diferencias significativas** en la función de supervivencia entre **dos o más grupos**
  - En cada nodo, se calcula el **estadístico del test log-rank** para **todos los predictores** evaluados en esa partición
    - Todos los puntos de corte posibles de los predictores continuos
    - Todas las combinaciones de categorías en los predictores categóricos
  - Se elige como **mejor partición** la que generan el predictor y el punto de corte con el **máximo estadístico** del test

- Se suele establecer un número **mínimo de eventos** en los nodos terminales

Random Survival Forest. Predicciones

- En RSF, se estiman la **función de supervivencia** y **la función de riesgo acumulado** (“cumulative hazard”) para cada observación (Kaplan-Meir)
  - En primer lugar, se estiman en los **nodos terminales** de cada **árbol**
  - Después, se **promedian** los valores de estas funciones en todos los árboles

- Para establecer en RSF una **predicción numérica** por observación se **suma la función de riesgo acumulada** en todos los tiempos
  - Es equivalente a un **risk score**, cuyos valores más altos corresponden a las observaciones con mayor riesgo, con peor supervivencia

- Las predicciones para **cada observación**, se pueden basar en **todos los árboles** o solo en los que no participó en su construcción (**OOB sample**)
  - OOB ensemble cumulative hazard estimator

- En RSF, se define la **Tasa de Error = 1 – C-index**

Importancia de las Variables

- **Importancia de las Variables**
  - La estimación del error está basada en la **muestra OOB**
  - Para evaluar **la importancia de la variable X**, se **permutan aleatoriamente** los valores de esa variable en la muestra OOB, y se vuelve a calcular el error
  - La **Importancia de la variable (VIMP)** se define como la diferencia entre esos dos errores promediada sobre todos los árboles
    - Si el error aumenta al permutar los valores de la variable significa que esa variable es importante, ya que se obtenía un error menor con los valores observados

- El **coste computacional** es alto
  - Se permutan los valores de todas las variables en todos los árboles
  - Para establecer adecuadamente la importancia de las variables en el modelo, hay que valorar el uso de un número grande de árboles, para que todas las variables tenga la oportunidad de participar en la construcción de bastantes árboles

Random Forest en R

- El paquete ***randomForest*** es el paquete habitual que se ha usado en R para construir modelos de Random Forest para problemas de **clasificación** y **regresión**
- El paquete ***randomSurvivalForest*** es el que se usaba para problemas de **supervivencia**, pero ya no se mantiene

- El paquete ***randomForestSRC*** es el nuevo paquete que integra los 2 anteriores, y por tanto permite construir modelos de **clasificación, regresión y supervivencia**
  - Permite **paralelización**

- El paquete ***ggRandomForests*** es un paquete que permite explorar los modelos construidos
  - Extrae los objetos generados por *randomForestSRC* y genera gráficos con el paquete ***ggplot2***, pudiéndose usar los comandos de este paquete para modificar los gráficos

Paquete *randomForestSRC* de R. Paralelización

- El paquete ***randomForestSRC*** permite **paralelización**

- Las **instrucciones** son distintas para Windows y Linux, y están en la documentación del paquete y en la página: http://www.ccs.miami.edu/~hishwaran/rfsrc.html

- Para **Windows**, hay que descargar **una versión especial** del paquete desde esa página web, e instalarla como un **ZIP**
  - Este ZIP está en la documentación del curso

- No hace falta ninguna librería. Solo hay que **declarar** al principio del script **los núcleos** que se van a usar

```
options ( rf.cores=detectCores() - 1, mc.cores=detectCores() – 1 )
```

Paquete *randomForestSRC* de R

- La función ***rfsrc()*** se usa para construir un modelo de **Random Forest**

- **Parámetros**
  - **formula** : formula con la variable respuesta y los predictores
  - **data :** data frame que contiene los datos
  - **ntree** : número de árboles
  - **mtry** : número de variables que entran como candidatas en cada división
    Por defecto: sqrt(p) en clasificación y supervivencia y p/3 en regresión
  - **nodesize :** número mínimo de observaciones en un nodo terminal (survival, 3)
  - **nsplit :** número de puntos aleatorios a explorar en los predictores continuos
  - **importance = T** : método para calcular la importancia de las variables
    Poned importance="none" si no se desea usar
  - **proximity = T** : para calcular la matriz de proximidades entre observaciones

Random Forest. Preparación de los datos

```
> library(survival)
> library(randomForestSRC)
> library(ggRandomForests)
> library(rms)
> library(caret)
> library(risksetROC)
> detectCores()
[1] 4
> options(rf.cores=detectCores()-1, mc.cores=detectCores()-1)  ## Cores paralelización
> ## Fichero Datos: Breast Cancer Genes
> yy=read.delim("C://Taller Supervivencia Alta Dimensionalidad/BreastCancerGenes.txt",
sep="\t")
> xx = yy [ , c ( "event_death", "survival.death.", "diameter.mm.",
+                 "Lymph_node_number_postive", "pN_3_classes",
+                 "Mastectomy", "ER", "Grade_3_classes", "Age.years.",
+                 "Chemo", "Hormonal", names(yy)[30:4948] ) ]
> ## Muestra de Trining y Muestra de Testing
> set.seed (444)
> ind.train = createDataPartition ( factor(xx$event_death) , p = 2/3 )$Resample1
> set.seed (NULL)
> xx.train = xx [ ind.train , ]
> xx.test  = xx [ - ind.train  , ]
> n.all = nrow ( xx.train )
> dim(xx.train)
[1]  197 4930
```

- Se cargan las **librerías** necesarias para el análisis, los cores para la **paralelización**. Se lee el **fichero de datos**, y se crean las mismas **muestras de training y testing**

Random Forest. Parámetro *nsplit*=

```
> out.rsf.1 <- rfsrc ( Surv ( survival.death. , event_death ) ~ . , data=xx.train,
+                      ntree=1000, nsplit=10 )
> out.rsf.1

                         Sample size: 197
                  Number of deaths: 53
                   Number of trees: 1000
        Minimum terminal node size: 3
     Average no. of terminal nodes: 25.748
No. of variables tried at each split: 71
            Total no. of variables: 4928
                        Analysis: RSF
                          Family: surv
                  Splitting rule: logrank *random*
     Number of random split points: 10
                        Error rate: 27.53%
```

- El objeto devuelto por Random Forest proporciona información sobre los parámetros de ejecución: tamaño de la muestra, número de predictores, número de árboles, …
- El número **mínimo de eventos** en un nodo terminal es 3 y el **número de variables**, elegidas aleatoriamente, para analizar cada división es 71 (= sqrt(4928))
- Random Forest prioriza la inclusión de **las variables continuas**, porque analiza todas las variables binarias formadas con todos los puntos de corte posibles
- El **Error Rate es 27.53%,** que está estimado en **las muestras OOB**

Random Forest. Parámetro *nsplit*=

```
> head(out.rsf.1$importance)
             diameter.mm. Lymph_node_number_postive              pN_3_classes
             2.721257e-05              8.612140e-05             -2.541378e-05
               Mastectomy                        ER           Grade_3_classes
            -2.324263e-05              3.330448e-05              4.505191e-05
>
> imp.rsf.1 <- sort(out.rsf.1$importance, decreasing=T)
> head(imp.rsf.1)
 NM_001168    NM_013277       U96131    NM_003981    NM_001897    NM_013290
0.0013235642 0.0011202738 0.0010830726 0.0010494821 0.0009860618 0.0009555169
>
> which( names(imp.rsf.1) == "ER" )
[1] 1108
> which( names(imp.rsf.1) == "Grade_3_classes" )
[1] 852
```

- El objeto ***$importance*** contiene la **importancia de las variables**, en el mismo orden en que aparecen en el dataframe. Utilizamos la función *sort()* para ordenarlas
- Con *nsplit=10* se prueban solo 10 puntos de corte aleatoriamente elegidos, pero aún así es muy difícil que las variables binarias o con pocas categorías se sitúen entre las más importantes
- Las dos **variables clínicas** más importantes, ER y Grade, no aparecen entre las más importantes (posiciones 1108 y 852)

Random Forest. Parámetro *nsplit*=

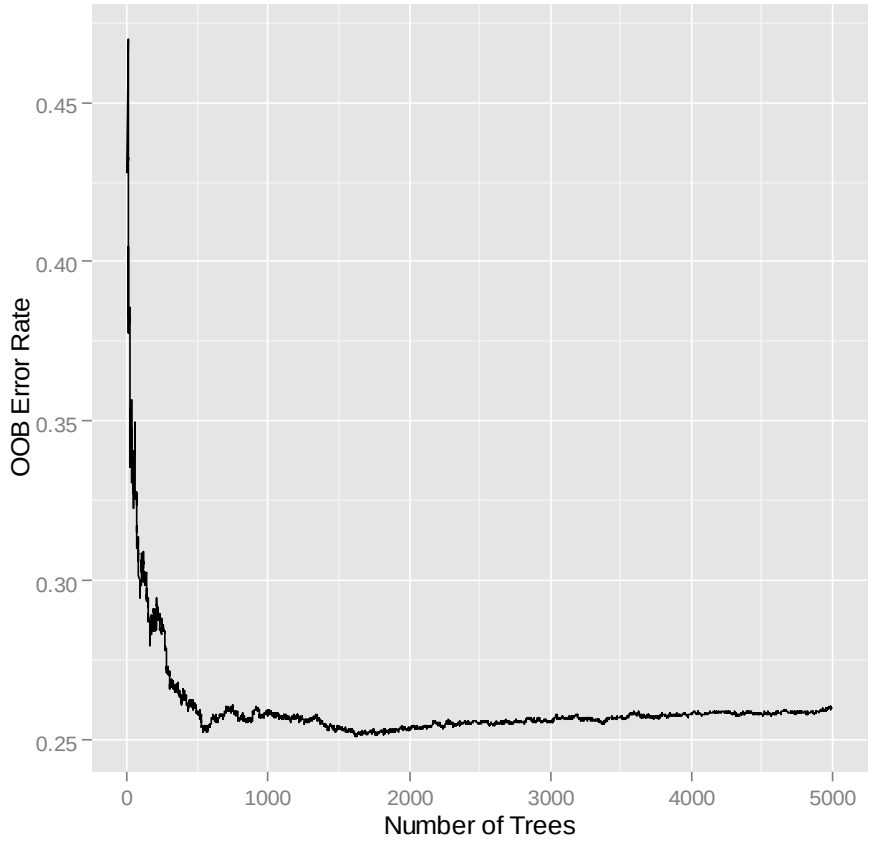
```
> out.rsf.2 <- rfsrc ( Surv ( survival.death. , event_death ) ~ . , data=xx.train,
+                      ntree=1000, nsplit=1 )
> out.rsf.2
                         Sample size: 197
                    Number of deaths: 53
                     Number of trees: 1000
          Minimum terminal node size: 3
       Average no. of terminal nodes: 27.976
No. of variables tried at each split: 71
              Total no. of variables: 4928
                            Analysis: RSF
                              Family: surv
                     Splitting rule: logrank *random*
       Number of random split points: 1
                          Error rate: 26.36%
> imp.rsf.2 <- sort(out.rsf.2$importance, decreasing=T)
> which( names(imp.rsf.2) == "ER" )
[1] 20
> which( names(imp.rsf.2) == "Grade_3_classes" )
[1] 52
```

- Con *nsplit=1* las **variables clínicas** ER y Grade se sitúan entre las más importantes
- El **Error Rate** es muy parecido, y por tanto la elección no está asociada a la capacidad predictiva del modelo; pero se debe valorar la elección del parámetro *nsplit*= teniendo en cuenta la **interpretación** del modelo

Random Forest. Parámetro *ntree*=

```
> out.rsf.3 <- rfsrc ( Surv ( survival.death. , event_death ) ~ . , data=xx.train,
+                      ntree=5000 , importance="none", nsplit=1 )
> out.rsf.3
. . . .

                         Error rate: 26.02%
>
> ## Gráfico del OOB Error frente al número de árboles
> dev.new(); plot ( gg_error ( out.rsf.3 ) )
```

- Para establecer el **número de árboles óptimo**, utilizamos la opción *importance="none"* para que no estime la importancia de las variables y el coste computacional sea menor
- Usamos la función ***gg_error()*** del paquete *ggRandomForests*
- El OOB Error Rate converge a partir de un **número de árboles** entre **1500 y 2000**



Random Forest. Modelo completo

```
> ## Construcción de RF con todas las variables y número óptimo de árboles (9 min)
> out.rsf.all <- rfsrc ( Surv ( survival.death. , event_death ) ~ . , data=xx.train,
+                        ntree=2000, nsplit=1 )
> out.rsf.all
. . . .

                         Error rate: 26.61%
>
> ## OOB Error Rate
> length(out.rsf.all$err.rate)
[1] 2000
> head(out.rsf.all$err.rate)
[1] 0.4483776 0.4115580 0.3888350 0.3584939 0.3389871 0.3553349
> tail(out.rsf.all$err.rate)
[1] 0.2660550 0.2659276 0.2661825 0.2660550 0.2661825 0.2660550
> err.rate.rsf = out.rsf.all$err.rate [ out.rsf.all$ntree ]
> err.rate.rsf
[1] 0.266055
```

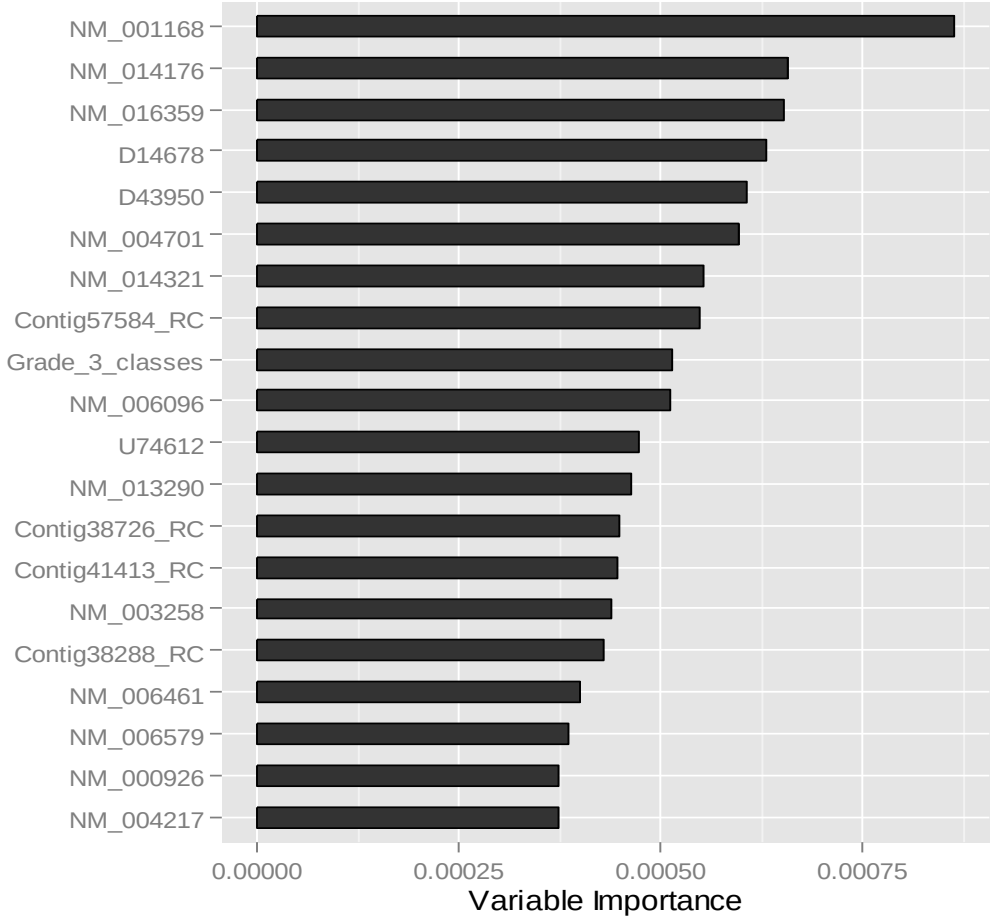
- Construimos el **modelo completo** de **Random Survival Forest** con todas las variables predictoras y el número de árboles óptimo (*ntree=2000*)
- El objeto ***$err.rate*** contiene un vector con los valores del **OOB error rate** obtenidos en función del número de árboles construidos. Son los valores del gráfico anterior
- El último elemento contiene **el error del modelo** de Random Forest

Random Forest. Modelo completo

```
> ## Importancia de las Variables
> imp.rsf.all <- sort(out.rsf.all$importance, decreasing=T)
> imp.rsf.all[1:4]
   NM_001168    NM_014176    NM_016359       D14678
0.0008644244 0.0006570748 0.0006517638 0.0006307825
> ## Gráfico de la Importancia de las variables
> dev.new(); plot ( gg_vimp ( out.rsf.all , n_var= 20 ) )
```

- En el objeto ***$importance*** se almacenan la **importancia de las variables**

- Usamos la función ***gg_vimp()*** del paquete *ggRandomForests* para obtener un gráfico de las variables más importantes



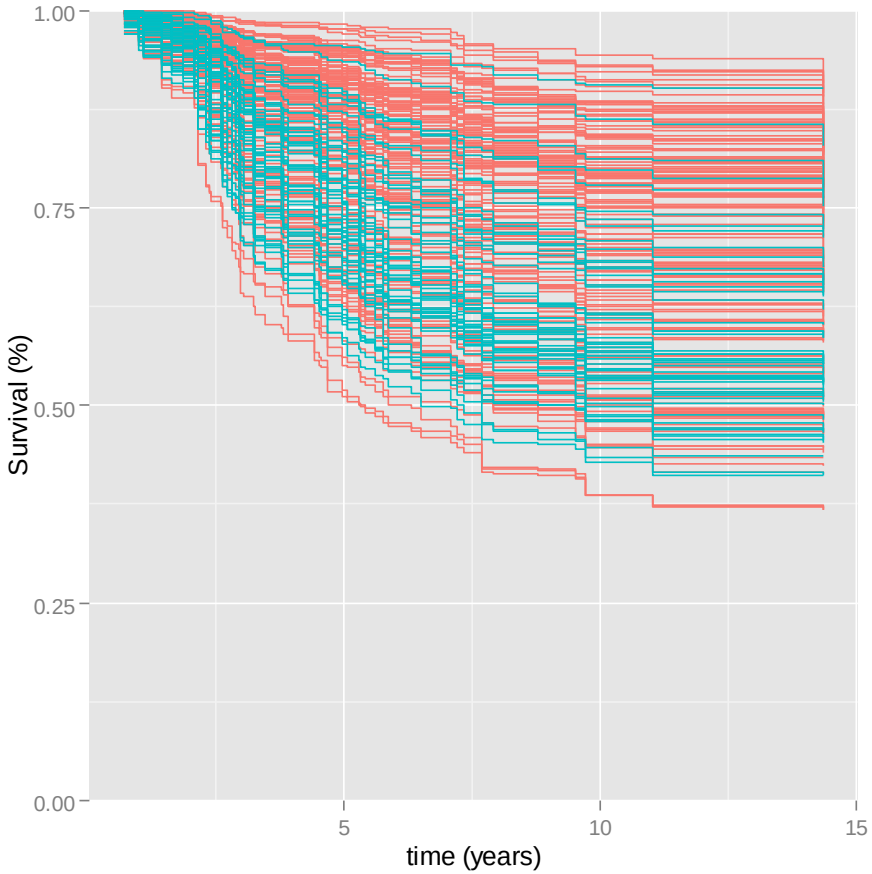
Random Survival Forest. Predicciones

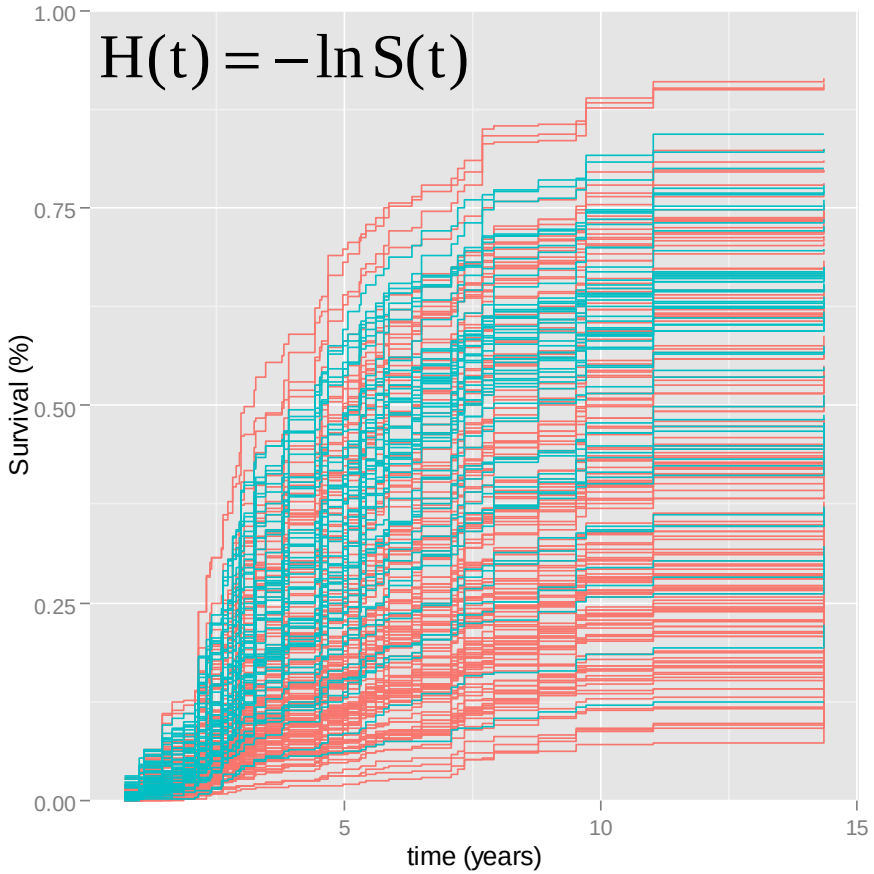
```
> ## Tiempos de supervivencia, donde se han producido eventos
> out.rsf.all$ndead
[1] 53
> length ( out.rsf.all$time.interest )
[1] 53
> head ( out.rsf.all$time.interest )
[1] 0.7118412 0.9746749 1.0732375 1.4346338 1.6344969 1.8480493
> head ( sort ( xx.train$survival.death. [ xx.train$event_death == 1 ] ) )
[1] 0.7118412 0.9746749 1.0732375 1.4346338 1.6344969 1.8480493
> ## Predicciones. Función de Supervivencia y Función de Riesgo Acumulado (CHF)
> dim ( out.rsf.all$survival.oob )
[1] 197  53
> dim ( out.rsf.all$chf.oob )
[1] 197  53
> dim ( out.rsf.all$survival )
[1] 197  53
> dim ( out.rsf.all$chf )
[1] 197  53
> ## Observación 1
> out.rsf.all$survival.oob [ 1 , ]
 [1] 0.9927807 0.9863414 0.9833111 0.9768494 0.9735740 0.9670009 0.9449198 0.9344029
. . .
[49] 0.5708668 0.5648507 0.5388258 0.5021947 0.4986297
```

- El objeto ***$time.interest*** contiene los tiempos de supervivencia en los que han ocurrido eventos. Los objetos ***$survival.oob*** y ***$chf.oob*** contiene las estimaciones para cada observación de la función de supervivencia y de riesgo acumulado (muestras OOB)

Random Survival Forest. Predicciones

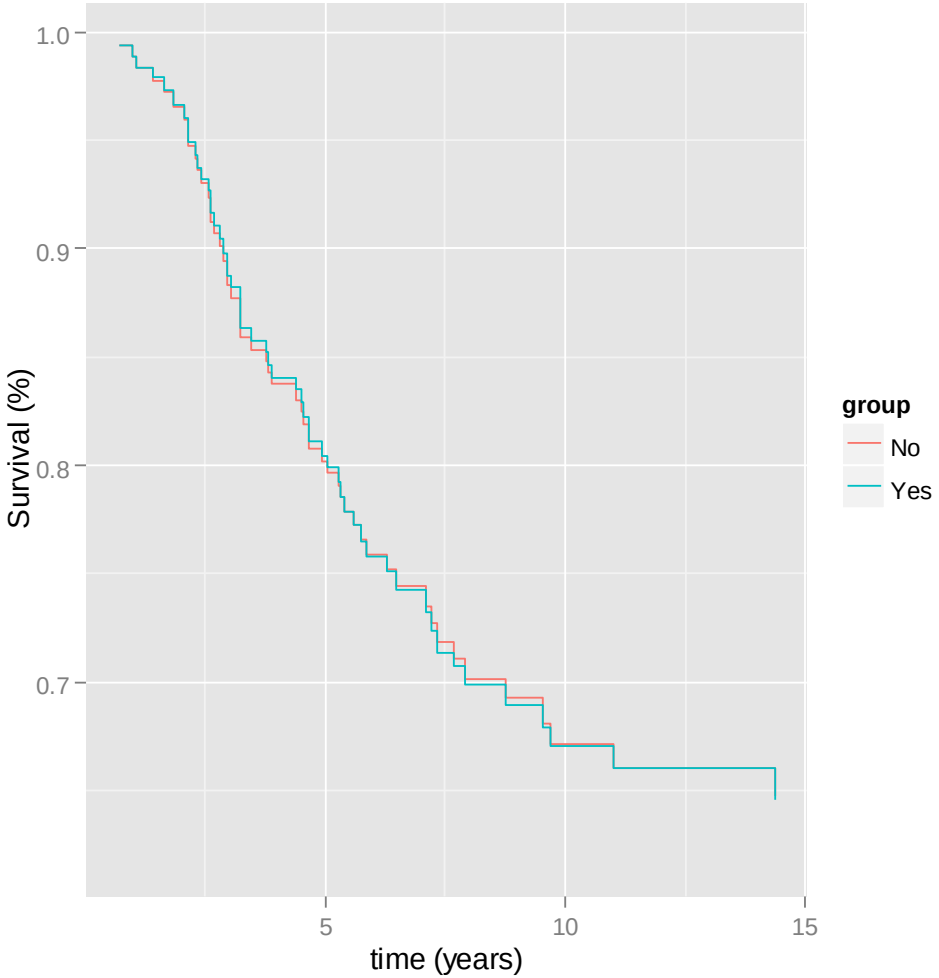
```
> ## Gráfico de las Curvas de Supervivencia Predichas (muestras OOB)
> dev.new()
> plot ( gg_rfsrc(out.rsf.all) ) +
+ theme ( legend.position="none" ) + coord_cartesian ( y = c ( 0, 1 ) )
> ## Gráfico de las Curvas de Riesgo Acumulada Predichas (muestras OOB)
> dev.new()
> plot ( gg_rfsrc(out.rsf.all, surv_type = "chf"  ) )  +
+ theme ( legend.position="none" ) + coord_cartesian ( y = c ( 0, 1 ) )
```

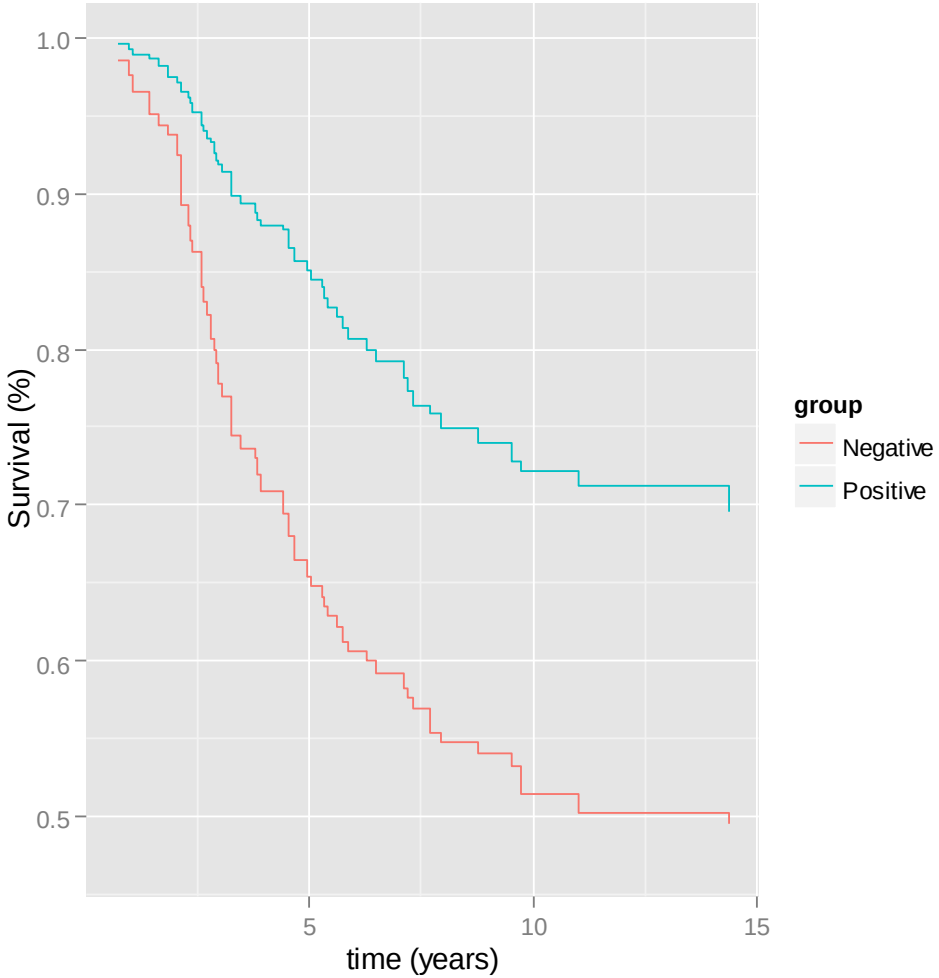




Random Survival Forest. Predicciones

```
> ## Gráficas de las Curvas de Supervivencia por Grupos
> dev.new(); plot ( gg_rfsrc(out.rsf.all , by="Chemo" ) )
> dev.new(); plot ( gg_rfsrc(out.rsf.all , by="ER" ) )
```





Random Survival Forest. Risk Score

```
> ## Predicción. Suma de la función de Riesgo
> length ( out.rsf.all$predicted.oob )
[1] 197
> head ( out.rsf.all$predicted.oob )
[1] 17.406395  8.735400 15.230128 17.607549  5.399075  5.044161
> sum.chf.oob = apply ( out.rsf.all$chf.oob , 1, sum )
> head ( sum.chf.oob )
[1] 17.406395  8.735400 15.230128 17.607549  5.399075  5.044161
> ## OOB Error = 1 – C-index
> rcorr.cens( out.rsf.all$predicted.oob,
+             Surv ( xx.train$survival.death. , xx.train$event_death ))["C Index"]
0.266055
> err.rate.rsf
[1] 0.266055
> ## C-index ( Mayor supervivencia se relaciona con menor riesgo )
> rcorr.cens( - out.rsf.all$predicted.oob,
+             Surv ( xx.train$survival.death. , xx.train$event_death ))["C Index"]
0.733945
```

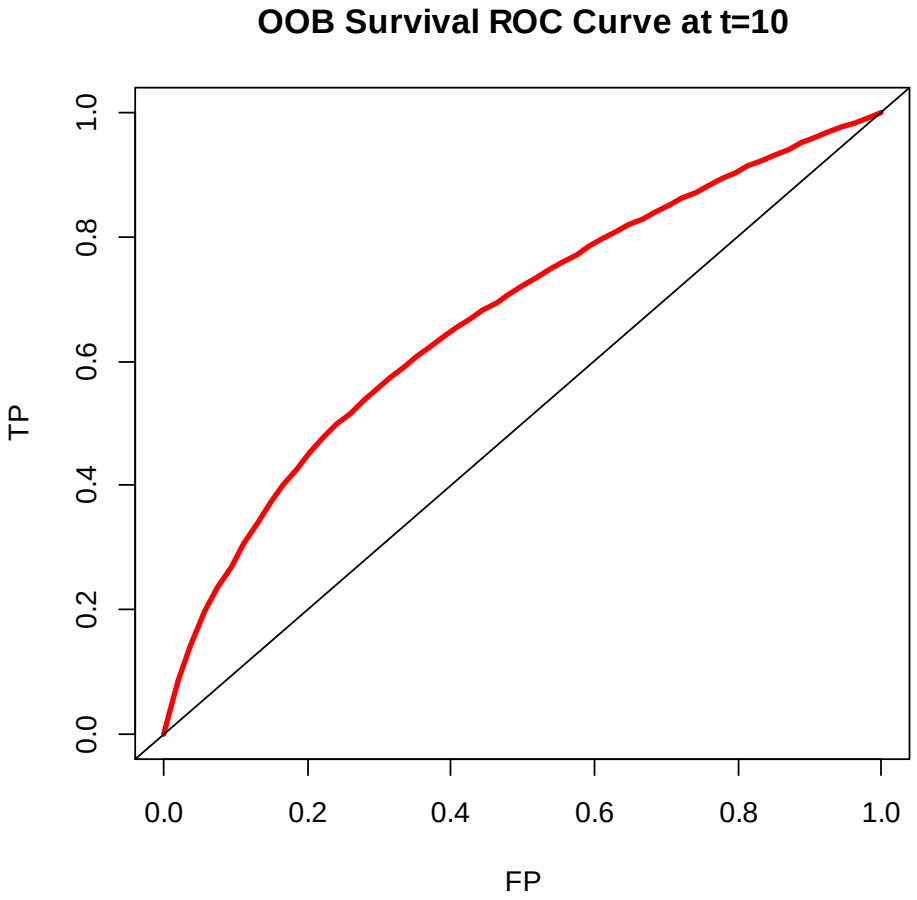
- El objeto ***$predicted.oob*** contiene las predicciones obtenidas en las muestras OOB, que son la **suma de la función de riesgo acumulada**. Es semejante a un **risk score** obtenido por validación cruzada (“*CV predictive index*”)
- La función ***rcorr.cens()*** del paquete ***rms*** se puede usar para calcular el **C-index**. Calcula la concordancia con la supervivencia observada. Si se introduce una función de riesgo, se obtiene **1 – C-index**, que comprobamos que coincide con el **OOB Error Rate**. Para obtener el C-index, se introduce el risk score con signo negativo

Random Survival Forest. Risk Score

```
> ## Survival ROC at t=10
> w.ROC = risksetROC( Stime=xx.train$survival.death., status=xx.train$event_death,
+                     marker=out.rsf.all$predicted.oob, predict.time=10, method="Cox",
+                     main="OOB Survival ROC Curve at t=10", lwd=3, col="red" )
>
> w.ROC$AUC
[1] 0.6696648
```

- El paquete ***risksetROC*** permite obtener curvas ROC para datos de supervivencia, especificando un tiempo de interés (t=10)

- El **risk score** obtenido con las **muestras OOB** puede ser usado para mostrar una **curva ROC sin sobreajustar**



Random Survival Forest. Sobreajuste

```
> ## C-index sobreajustado
> rcorr.cens( - out.rsf.all$predicted,
+             Surv ( xx.train$survival.death. , xx.train$event_death ))["C Index"]
  C Index
0.9831804
> ## Survival ROC at t=10 sobreajustada
> risksetROC( Stime=xx.train$survival.death., status=xx.train$event_death,
+             marker=out.rsf.all$predicted, predict.time=10, method="Cox", plot=F)$AUC
[1] 0.8263632
```

- El objeto ***$predicted*** contiene las predicciones obtenidas para las observaciones basadas en **todos los árboles** de Random Forest, incluidos los árboles en los que esas observaciones han participado en su construcción
- **No deberían utilizarse** para presentar resultados del modelo, porque proporciona resultados **sobreajustados**
- Comprobamos los valores tan altos que se obtienen al calcular el **C-index** o el **AUC** si se utilizan estas predicciones

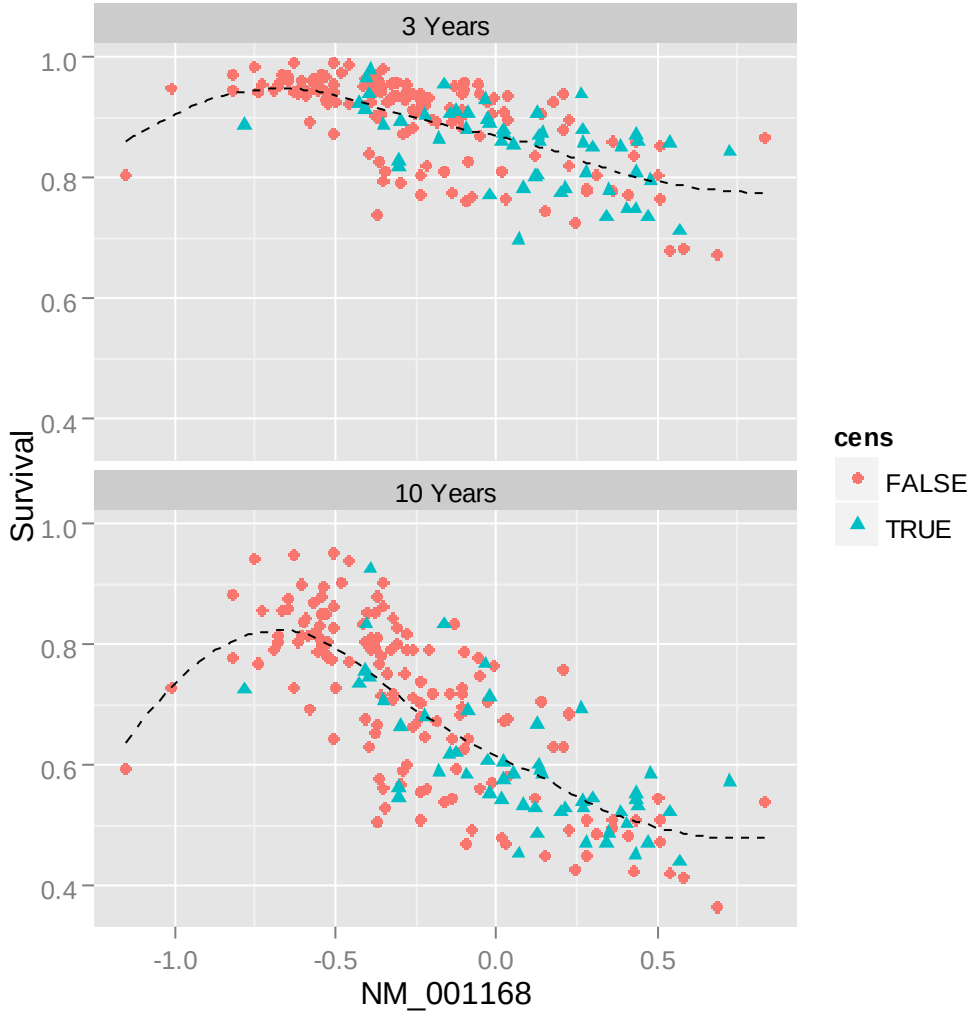
RSF. Dependencia de las Variables

```
> gg_v = gg_variable ( out.rsf.all, time = c( 3, 10 ),
+                      time.labels = c("3 Years", "10 Years") )
> dev.new()
> plot ( gg_v, xvar="NM_001168", se=0.95 )
```

- Para analizar la relación entre una variable predictora y la variable supervivencia se puede obtener **los gráficos de dependencia** con la función ***gg_variable()***

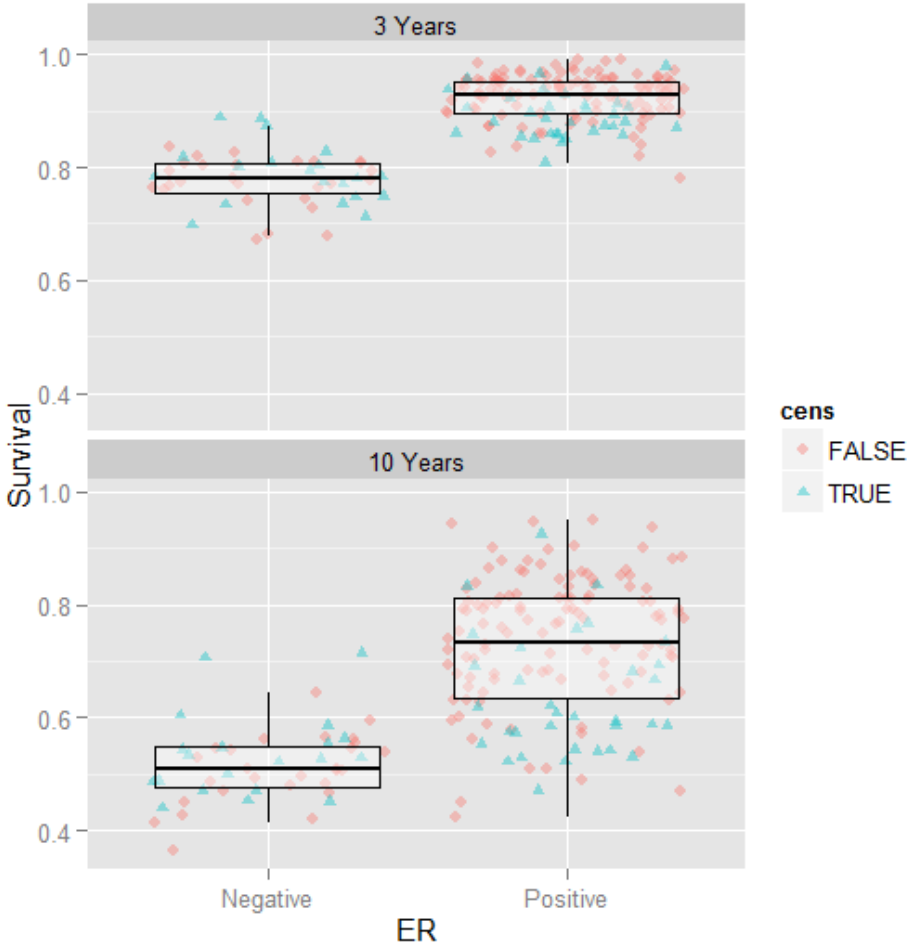
- Se fijan 2 tiempos de interés: 3 y 10 años
- En la función ***plot()*** se especifica la variable que se desea mostrar

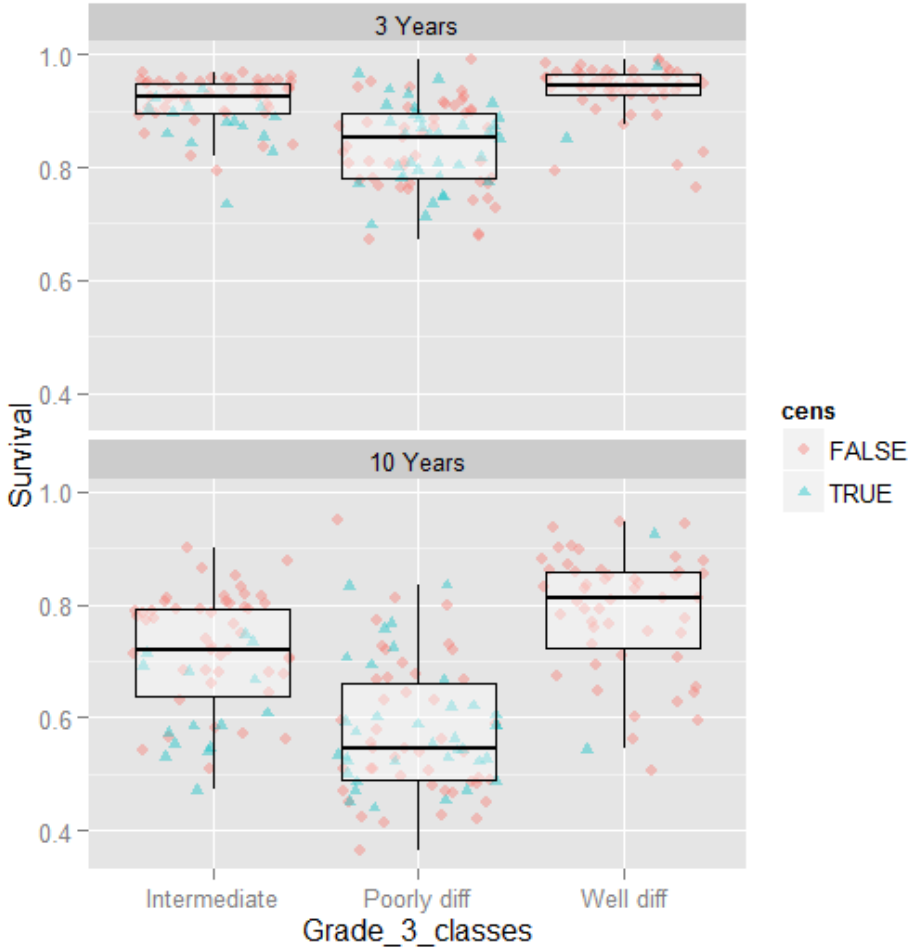
- En la leyenda *cens=T* (triángulos azules) significa *status=1*



RSF. Dependencia de las Variables

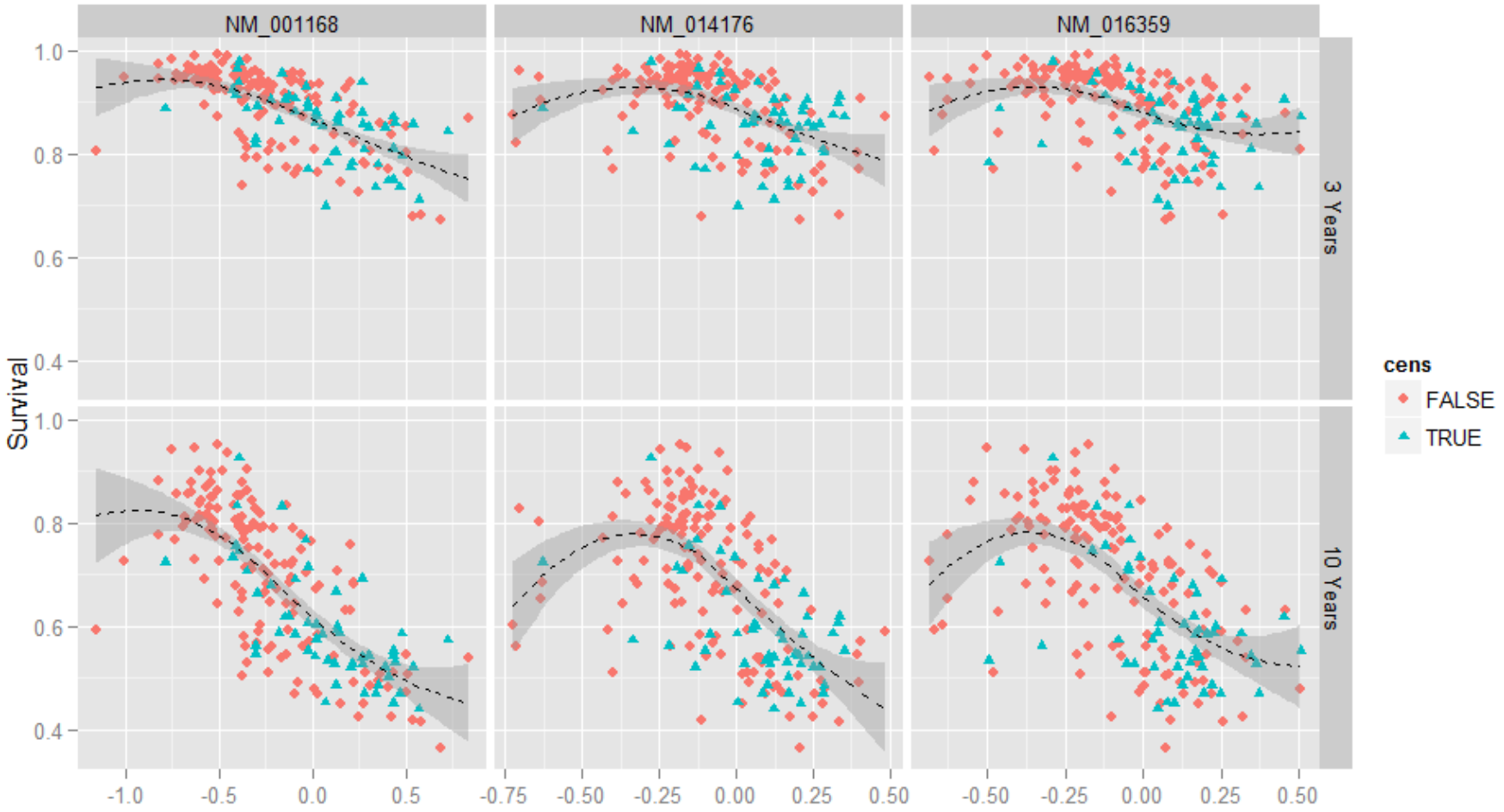
```
> dev.new()
> plot ( gg_v, xvar="ER", se=0.95, alpha=0.4 )
> dev.new()
> plot ( gg_v, xvar="Grade_3_classes", se=0.95, alpha=0.4 )
```





RSF. Dependencia de las Variables

```
> dev.new()
> plot ( gg_v, xvar=c("NM_001168","NM_014176","NM_016359"),
+        panel=T, se=0.95, span=1 )
```



Optimización del tamaño del modelo

- **Random Forest** genera un modelo con todas las variables, que tiene una **alta capacidad predictiva**

- Se puede plantear si es posible obtener un subconjunto de variables que proporcione un **modelo reducido**, sin perder poder predictivo
  - Se suelen fijar **los tamaños del modelo** a explorar
  - Se suelen tomar valores: p/2, p/4, p/8, …. hasta modelos muy pequeños

- Se puede aplicar la **metodología** habitual para optimizar un parámetro de una técnica predictiva, basado en **validación cruzada**
  - El **proceso completo** de selección de variables debe estar dentro de la **validación cruzada**

- El modelo reducido podría usarse para **predecir nuevas observaciones**

RF. Optimización del tamaño del modelo

```
> set.size = c ( 2, 5, 10, 25, 50, 100, 250, 500, 1000, 4928 )
> n.cv    = 5     # n.cv = 10
> n.times = 2     # n.times = 5
> test.er = matrix ( NA, n.times*n.cv, length(set.size) )
> ind.1 = 0
> for ( ind.times in 1:n.times )     ## Bucle para cada ejecución de la CV
+ {
+   ## Asignación de particiones a las observaciones de la muestra
+   fold.cv = sample ( rep ( 1:n.cv, length=n.all ) )
+   ## Bucle para cada partición de la CV
+   for ( ind.cv in 1:n.cv )
+   {
+     ind.1 = ind.1 + 1   ## Indice que controla la fila de test.er
+     ## Muestras de training y testing de la CV
+     cv.train = xx.train [ fold.cv != ind.cv , ]
+     cv.test  = xx.train [ fold.cv == ind.cv , ]
+     ## RSF con todas las variables en la muestra de training de la CV
+     rsf.cv <- rfsrc ( Surv ( survival.death. , event_death ) ~ . ,
+                       data=cv.train, ntree=500, nsplit=1 )
+     ## La última columna está reservada para el modelo completo (4928 variables)
+     pred.test = predict ( rsf.cv, newdata=cv.test, importance="none" )
+     test.er [ ind.1, length(set.size) ] = pred.test$err.rate [pred.test$ntree]
```

- Se va a estimar el error con **validación cruzada repetida** (2 times 5-fold CV). Para cada ejecución de la CV, se asignan nuevas particiones
- Para cada submuestra de la CV, se construye **RSF** sobre la **muestra de training** y se predice en la muestra de testing. Se obtiene el error del **modelo completo**

RF. Optimización del tamaño del modelo

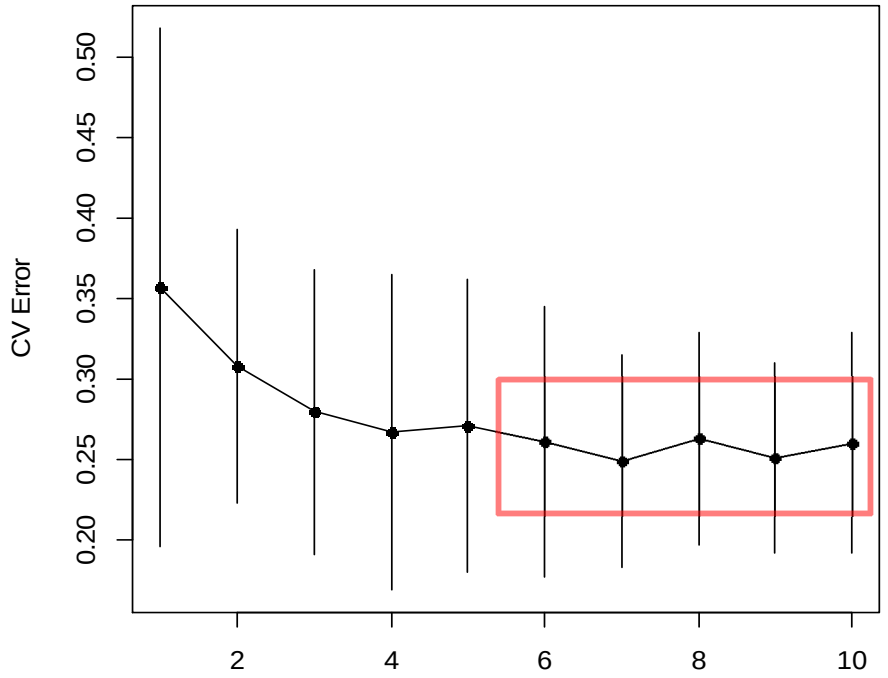
```
+     ## Importancia de las variables
+     imp.rsf.cv =  sort ( rsf.cv$importance, decreasing=T )
+     ##############################
+     ## Bucle para cada tamaño del modelo a analizar
+     ind.size = 0
+     for ( w.size in set.size[1:(length(set.size)-1)] )
+     {
+       ## RF con el conjunto de variables más importantes del tamaño seleccionado
+       cv.sub = cv.train [  , c ( "survival.death." , "event_death" ,
+                                  names(imp.rsf.cv)[1:w.size] ) ]
+
+       rsf.sub <- rfsrc ( Surv ( survival.death. , event_death ) ~ . ,
+                          data=cv.sub, ntree=500, nsplit=1, importance="none" )
+
+       ## Predicciones en la muestra de testing de la CV de RF
+       ind.size = ind.size + 1
+       pred.test = predict ( rsf.sub, newdata=cv.test, importance="none" )
+       test.er [ ind.1, ind.size ] = pred.test$err.rate[pred.test$ntree]
+     }
+   }
+ }
```

- Para cada submuestra de la CV, el modelo completo RSF determina la **importancia de las variables**, para probar con **modelos reducidos** de los tamaños especificados
- Es decir, la **evaluación de los modelos reducidos** se hace con un modelo distinto en cada submuestra de la CV

RF. Optimización del tamaño del modelo

```
> ## Medias y SDs de la CV por tamaños del modelo (columnas)
> mean.er = apply ( test.er, 2 , mean , na.rm=T )
> sd.er   = apply ( test.er, 2 , sd , na.rm=T )
> mean.er
 [1] 0.3573675 0.3084635 0.2801848 0.2675555 0.2712427 0.2613594 0.2491188 0.2637811
 [9] 0.2515746 0.2608475
> ## Gráfico de OOB Error Rate frente al Número de Variables
> dev.new()
> plot ( 1:length(set.size) , mean.er , xlab="size", ylab="CV Error" , type="l" ,
+        ylim = c( min(mean.er - sd.er) , max(mean.er + sd.er) ) )
> points ( 1:length(set.size), mean.er , pch=16 )
> segments ( 1:length(set.size), mean.er-sd.er, 1:length(set.size), mean.er+sd.er)
```

- Se calculan la **media** y la SD de los **errores** para cada **tamaño del modelo**
- Con la CV se obtienen valores muy parecido al OOB Error: 0.261 y 0.266
- A partir de **100 variables** se obtienen tasas de error semejantes al modelo completo
- Incluso con modelos con 50 variables, el aumento del error puede ser asumible



Random Survival Forest. Modelo Final Reducido

```
> num.var.opt = 100
>
> xx.train.fin = xx.train [ , c ( "survival.death." , "event_death" ,
+                                 names(imp.rsf.all)[1:num.var.opt] ) ]
>
> out.rsf.fin <- rfsrc ( Surv ( survival.death. , event_death ) ~ . ,
+                        data=xx.train.fin, ntree=2000, nsplit=1 )
> out.rsf.fin
                         Sample size: 197
                    Number of deaths: 53
                     Number of trees: 2000
          Minimum terminal node size: 3
       Average no. of terminal nodes: 31.561
No. of variables tried at each split: 10
              Total no. of variables: 100
                            Analysis: RSF
                              Family: surv
                      Splitting rule: logrank *random*
       Number of random split points: 1
                         Error rate: 19.72%
```

- Una vez seleccionado el número óptimo de variables, 100, se eligen las **variables más importantes del modelo completo**, el que se construyó con todas las variables
- El **OOB Error Rate** que devuelve este modelo, 19.72% está **sobreajustado**, debido a que las 100 variables seleccionadas están “contaminadas” por el ranking establecido por el modelo completo. La **tasa de error** obtenido por CV es **26.14%**

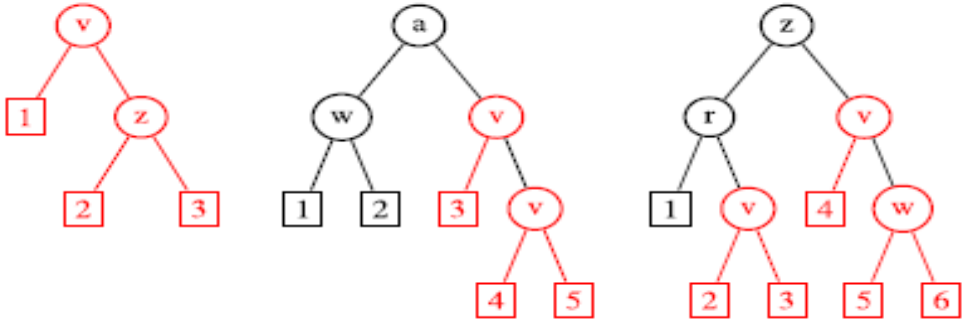
Random Survival Forest. Modelo Final Reducido

```
> ## Predicciones en la muestra de testing
>
> ## Con el Modelo Final Reducido
> pred.test.fin = predict ( out.rsf.fin, newdata=xx.test, importance="none" )
>
> rcorr.cens ( pred.test.fin$predicted ,
+              Surv ( xx.test$survival.death. , xx.test$event_death ) )["C Index"]
  C Index
0.2167947
>
> ## Con el Modelo Completo
> pred.test.all = predict ( out.rsf.all, newdata=xx.test, importance="none" )
>
> rcorr.cens ( pred.test.all$predicted ,
+              Surv ( xx.test$survival.death. , xx.test$event_death ) )["C Index"]
  C Index
0.2102086
```

- El **modelo reducido** nos permite tener un modelo con menos variables y el mismo poder predictivo. Podría ser usado para **predecir en nuevas observaciones**
- Si probamos en la **muestra de testing**, que se reservó al principio del proceso, y que no ha intervenido en la construcción de RSF, las **tasas de error** (1 – C-index) obtenidos por el modelo reducido y el modelo completo son **semejantes**
- Con el modelo construido con Lasso, se obtuvo 0.224 (1 - 0.776)

Selección de Variables. Profundidad Mínima

- El **subárbol máximo (maximal subtree)** para una variable X es el árbol más grande cuyo nodo raíz está dividido por X
- La **profundidad mínima (minimal depth)** para un subárbol máximo es la distancia más corta del nodo raíz al nodo padre del subárbol máximo
  - Medida de **la importancia** que tiene una variable en la predicción. Una variable es importante si participa en las primeras divisiones de árbol



En rojo el subárbol máximo de la variable **v**

Profundidad Mínima de la variable **v** es 0,1 y 1

- La **Profundidad Mínima** es una alternativa al uso de la importancia de las variables para **seleccionar variables**, ya que establece **un ranking**
  - No se vincula a ninguna medida de capacidad predictiva

Selección de Variables. Profundidad Mínima

- Hay técnicas para establecer un **threshold** en la profundidad mínima, y seleccionar las variables que queden por debajo de ese threshold
  - Se basa en la media de la distribución de la profundidad mínima
  - No suele ser muy eficaz si el número de variables es muy alto (p >> n)

- Otro método es llamado **“Variable Hunting”** (“Cazando/Buscando variables”)
  - Se basa en tomar muestras aleatorias de los datos, y subconjuntos aleatorios de las variables
  - Se construye un modelo de RF y se establece un **modelo inicial** a partir de un **threshold** en la **profundidad mínima**
  - Se añaden variables a ese modelo basado en la profundidad mínima hasta que la “**importancia conjunta**” de las variables se estabiliza, definiendo así un **modelo final**
  - Se repite el **proceso varias veces**, y se **seleccionan las variables** que aparecen con **más frecuencia**

RF. Selección de Variables. Profundidad Mínima

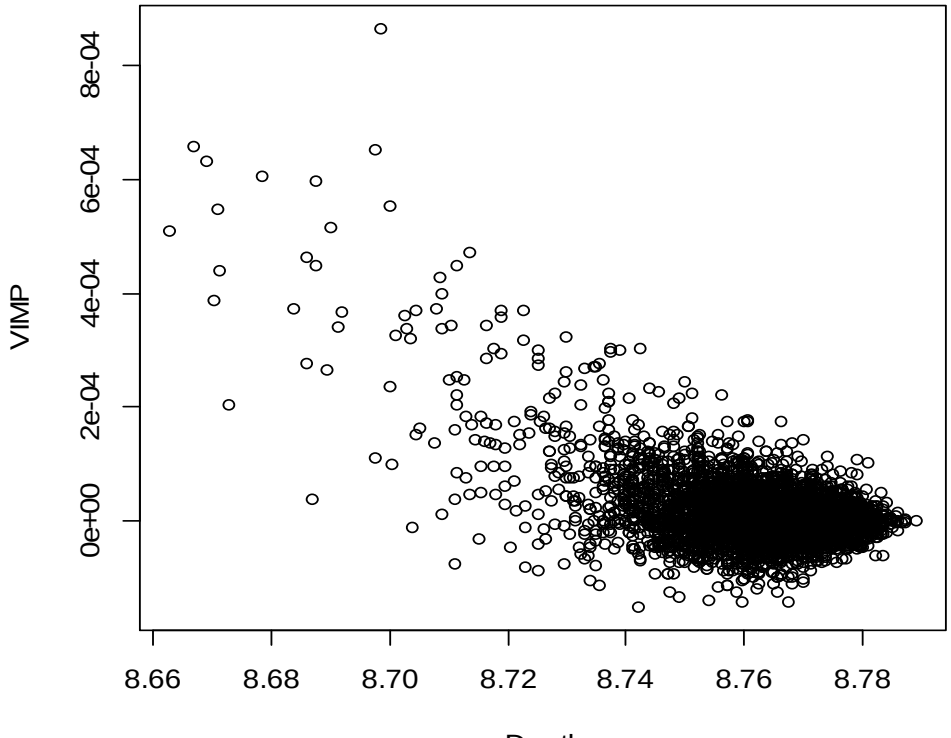
```
> ## Selección de Variables. Profundidad Mínima. Modelo RSF con todas las variables
> out.vs <- var.select ( out.rsf.all )
family              : surv
var. selection      : Minimal Depth
conservativeness    : medium
x-weighting used?   : TRUE
dimension           : 4928
sample size         : 197
ntree               : 2000
nsplit              : 1
mtry                : 71
nodesize            : 3
refitted forest     : FALSE
model size          : 0
depth threshold     : 8.2625
PE (true OOB)       : 26.6055

> out.vs <- var.select ( out.rsf.all , conservative="low" ) # Más conservador
. . .
model size          : 0
depth threshold     : 8.2898
```

- Utilizamos la función ***var.select()*** con la opción por defecto que establece un **threshold** para la profundidad mínima de 8.26 y por debajo de este valor no hay **ninguna variable**
- Se puede calcular otro threshold más conservador, pero tampoco selecciona ninguna variable

RF. Selección de Variables. Profundidad Mínima

```
> head(out.vs$varselect)
 depth         vimp
NM_006096      8.6630 0.0005104958
NM_014176      8.6670 0.0006570748
D14678         8.6690 0.0006307825
NM_006579      8.6705 0.0003864679
Contig57584_RC 8.6710 0.0005476152
NM_003258      8.6715 0.0004392997
> ## Relación entre Depth y Importancia de las Variables
> cor ( out.vs$varselect$depth, out.vs$varselect$vimp )
[1] -0.5184869
> dev.new();plot(out.vs$varselect$depth,out.vs$varselect$vimp,xlab="Depth",ylab="VIMP")
```

- El objeto ***$varselect*** contiene la **profundidad mínima** de todas las variables ordenadas
- También contiene la importancia de las variables, que se había calculado previamente
- Se puede examinar **la relación** que existe entre ambas medidas



RF. Selección de Variables. Hunting

```
## Método "Variable Hunting" (35-40 minutos)
> out.vs <- var.select ( out.rsf.all, method='vh', nsplit = 1 )
>
> class(out.vs$rfsrc.refit.obj)
[1] "rfsrc" "grow"  "surv"
> out.vs$rfsrc.refit.obj
 Sample size: 197
                     Number of deaths: 53
                      Number of trees: 500
           Minimum terminal node size: 2
        Average no. of terminal nodes: 38.776
No. of variables tried at each split: 16
               Total no. of variables: 227
                             Analysis: RSF
                               Family: surv
                     Splitting rule: logrank *random*
        Number of random split points: 1
                           Error rate: 19.67%
> length(out.vs$topvars)
[1] 124
> head(out.vs$topvars)
[1] "Contig55771_RC"            "AB037836"                  "Lymph_node_number_postive"
[4] "AK000004"                  "Contig56390_RC"            "D43950"
```

- Si se usa **Hunting** como método de selección de variables *(method='vh')*, se obtiene un **modelo de RSF con 227 variables**, que se devuelve en el objeto ***$rfsrc.refit.obj***
- El ER reportado, 19.67%, está sobreajustado

RF. Selección de Variables. Hunting

```
> dim(out.vs$varselect)
[1] 4928    2
> head(out.vs$varselect)
                  depth rel.freq
NM_013290      9.332714       28
NM_006115      9.689684       28
NM_001168      9.515467       28
Contig16786_RC 9.608133       26
NM_001897      9.703692       24
NM_020672      9.916889       24
> w.varselect =  out.vs$varselect [ rownames(out.vs$varselect) %in% out.vs$topvars ,  ]
> dim(w.varselect)
[1] 227   2
>
> ## Predicciones en la muestra de testing
> pred.test.hunt = predict( out.vs$rfsrc.refit.obj, newdata=xx.test, importance="none")
> rcorr.cens ( pred.test.hunt$predicted ,
+              Surv ( xx.test$survival.death. , xx.test$event_death ) )["C Index"]
  C Index
0.2129528
```

- En el objeto ***$varselect*** se almacena información sobre la profundidad de todas las variables, y con ***$topvars*** se pueden seleccionar las 227 del modelo
- Se pueden obtener **predicciones** con este modelo de la **muestra de testing**, y evaluar la **tasa de error** (1 - C-index), que es **0.213**, prácticamente igual a la del modelo completo (0.217) y del modelo reducido de 100 variables (0.210)

Curso de Formación Continua

Estadística Aplicada con 

Módulos	Fechas 2015
1. Introducción a R	24, 25 Septiembre
2. Métodos de Regresión con R	15, 16 Octubre
3. Métodos de Regresión Avanzados para la Investigación en Ciencias Naturales con R	19, 20, 21 Octubre
4. Estadística Aplicada a la Investigación Biomédica con R	11, 12, 13 Noviembre
5. Modelos Mixtos / Jerárquicos / Multinivel con R	18, 19, 20 Noviembre
6. Estadística Multivariante con R	26, 27 Noviembre
7. Técnicas Estadísticas de Data Mining con R	14, 15, 16, 17 Diciembre

Información: http://goo.gl/whB1MM y en http://www.alimentacion.imdea.org/unidad-de-formacion

Ejercicio

- **Fichero de datos: “pwbc”,** con muestras de training y testing
- **Variable respuesta: “time”** tiempo de supervivencia y **“status”** variable que indica si ha recaído (con código = “R”)

- Optimización del **número de árboles** para construir un modelo de **Random Survival Forest**. Utilizar el parámetro *nsplit* por defecto
- Construir el **modelo de Random Survival Forest** con el número de árboles óptimo, y evaluar la importancia de las variables
- Obtener las **predicciones** en las muestras de testing y calcular el **C-index**
- Mostrar los gráficos de dependencia de algunas de las variables más importantes (“pnodes”, “mean_radius”, …)

- Aplicar el método de la **profundidad mínima** para encontrar un número adecuado de variables para Random Survival Forest (usad *var.select()* con el parámetro *refit=T*)