

Machine Learning con Datos Censurados usando *tidymodels*

**III Congreso de Usuarios de R
Sevilla**

7 de noviembre de 2024

Jesús Herranz Valera
jherranzvalera@gmail.com
Bioestadístico GEICAM

Material del taller



- <https://github.com/jesusherranz/DatosCensuradosTidymodels>
- Instalar las librerías: “tidymodels”, “tidyverse”, “censored”, “smoothHR”, “survminer”
- Reinstalar también “survival”

Índice

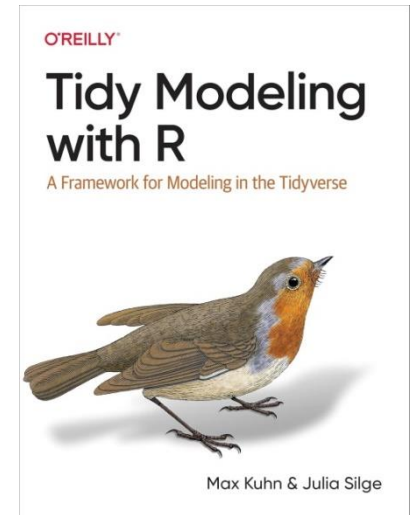
- ✓ **Análisis de Supervivencia. Datos Censurados**
- ✓ **Introducción a *tidymodels***

- ✓ **Modelos de Regresión Penalizados. Elastic Net**
- ✓ **Árboles de Supervivencia**
- ✓ **Random Survival Forest**

- ✓ **Anexo 1: Regresión de Cox con *tidymodels***
- ✓ **Anexo 2: Cross-Validated Predictions**

Links - *tidymodels*

- Libro “Tidy Modeling with R” Max Kuhn y Julia Silge
 - <https://www.tmwr.org/>
- *tidymodels*: <https://www.tidymodels.org/>
- Datos Censurados: <https://censored.tidymodels.org/>
- Datos Censurados: <https://www.tidymodels.org/learn/statistics/survival-case-study/>
- Datos Censurados: <https://www.tidyverse.org/blog/2021/11/survival-analysis-parsnip-adjacent/>
- Medidas dinámicas de capacidad predictiva:
 - <https://www.tidymodels.org/learn/statistics/survival-metrics/>
 - <https://www.tidymodels.org/learn/statistics/survival-metrics-details/>
- Regresión Penalizada con *glmnet* <https://parsnip.tidymodels.org/reference/glmnet-details.html>
- Random Forest con *partykit*
https://parsnip.tidymodels.org/reference/details_rand_forest_partykit.html



Fichero de datos: whas500 (1)

- Objetivo del estudio: explorar factores asociados a la supervivencia, después de sufrir un infarto de miocardio (MI) (n=500, p=13)

| Nombre | Descripción | Categorías |
|--------|-------------------------|---|
| id | Identificador | |
| lenfol | Tiempo de Seguimiento | Tiempo entre la admisión en el hospital y la última fecha de seguimiento o fecha de fallecimiento |
| fstat | Estado vital | 0 = vivo, 1 = fallecido |
| age | Edad | Continua |
| gender | Sexo | 0 = hombre, 1 = mujer |
| hr | Ritmo cardiaco | Continua |
| sysbp | Presión sistólica | Continua |
| diasbp | Presión diastólica | Continua |
| bmi | Índice de masa corporal | Continua |
| | | |

- “*Applied Survival Analysis*”. D. Hosmer, S. Lemeshow, S. May

Fichero de datos: whas500 (2)

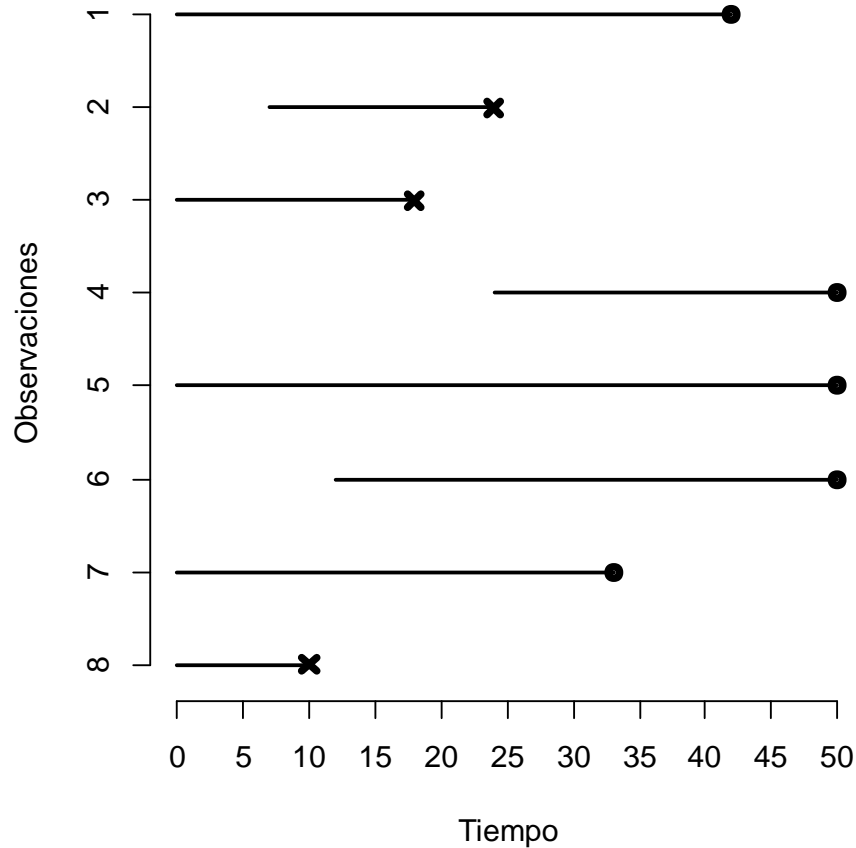
| Nombre | Descripción | Categorías |
|---------------|---------------------------------|-----------------------------|
| | | |
| cvd | Historia enferm. Cardiovascular | 0 = no, 1 = si |
| afb | Fibrilación arterial | 0 = no, 1 = si |
| sho | Shock cardiaco | 0 = no, 1 = si |
| chf | Complicaciones cardiacas | 0 = no, 1 = si |
| av3 | Bloqueo cardiaco | 0 = no, 1 = si |
| miord | Orden del infarto | 0 = primero, 1 = recurrente |
| mitype | Tipo del infarto | 0 = no Q-wave, 1 = Q-wave |

- Variables binarias codificadas como 0 / 1 (variables dummy)
- Tiempo en días, que se transformará a meses

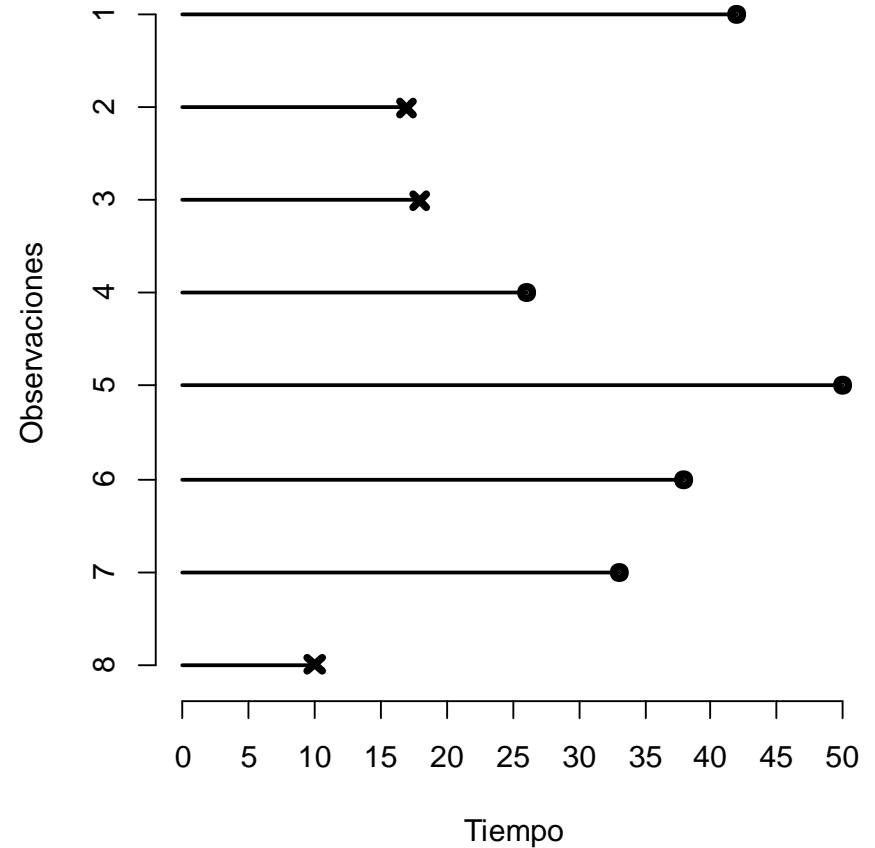
Análisis de Supervivencia. Datos censurados

- En el análisis de supervivencia o de datos censurados, 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, donde se mide el **tiempo de seguimiento**
- 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

Análisis de Supervivencia. Datos censurados



x – evento
o – censurado



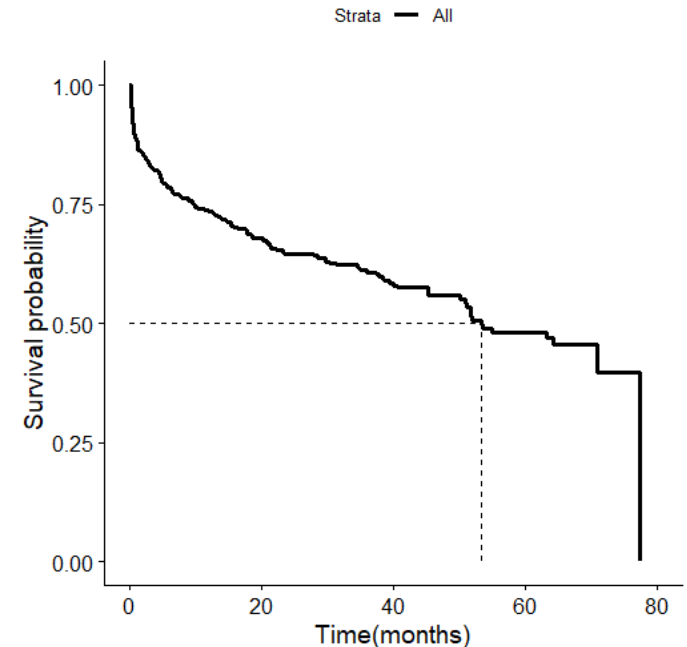
tiempo de supervivencia
tiempo de seguimiento

Análisis de Supervivencia. Datos censurados

- El objetivo es estimar la **función de supervivencia**, que es la probabilidad de que un individuo sobreviva durante un tiempo superior a t

$$S(t) = \text{Prob}(T > t)$$

- La forma más común de **describir** la función de supervivencia es con el **estimador de Kaplan-Meier**
 - Es una función **decreciente y escalonada**
 - Se estima en los **tiempos** donde se han producido **eventos**
- El **test log-rank** se utiliza para analizar si hay **diferencias** significativas en la función de supervivencia entre **dos o más grupos**



Regresión de Cox

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

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

- 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 verosimilitud parcial** para estimar los coeficientes
- El **Hazard Ratio** (HR), $\exp(\beta)$ 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

Predicciones y Medidas Capacidad Predictiva

| | Clasificación | Regresión | Datos Censurados |
|-------------------------------------|--|------------------------------------|---|
| Variable Respuesta | Categórica | Continua | Tiempo hasta que ocurre un evento |
| Predicciones | Clases Probabilidades | Valor numérico | Función de Supervivencia Predictor lineal (modelos de regresión) |
| Medidas Capacidad Predictiva | Precisión (Tasa de error) Kappa AUC Brier Score | MSE (mean squared error) R2 | C-index Medidas dinámicas de rendimiento predictivo: AUC(t) Brier Score(t) |

Regresión de Cox. Predicciones

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

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

- A partir de esta función, se puede deducir **la función de supervivencia**, en todo el rango del tiempo **$S(t; X)$**
- El **predictor lineal** del modelo permite obtener una **predicción numérica única**, para cada observación
 - Si el valor del **predictor lineal es alto**, significa que tiene mayor riesgo de que se produzca el evento, y por tanto, **peor supervivencia**
 - También se le llama **Risk Score** o Pronostic Score, según el contexto

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

tidymodels

- ***tidymodels*** es un conjunto de **paquetes de R**, diseñados especialmente para la **construcción de modelos predictivos**
- El objetivo principal es producir **modelos estadísticos** y de **machine learning** de **alta calidad**
- Sigue la filosofía y el dialecto de ***tidyverse***
- Uso del operador **pipe %>%**, **tubo**, para **encadenar funciones** de R: la **salida** de cada función, lo que retorna, es la **entrada** de la siguiente función
- Al igual que ***tidyverse***, recoge el objetivo general de estar “**diseñado para humanos**”, en el sentido de estar diseñado para usuarios, no para los desarrolladores de los paquetes
- Usa **nombres de funciones** conocidos y reconocibles
- Trabaja con cualquier **estructuras de datos**: matrices, fórmulas, ...

tidymodels



Paquetes de *tidymodels*

- Cada paquete recoge **funciones específicas** de **cada paso** de la construcción de los modelos
- Facilita el **mantenimiento** de los paquetes

| Paquete | Descripción |
|-------------------------|---|
| <i>rsample</i> | División de los datos en submuestras, y control de las técnicas de remuestreo |
| <i>parsnip</i> | Construcción de los modelos |
| <i>recipes</i> | Preprocesamiento de los datos |
| <i>workflows</i> | Integra el pre-procesamiento, el modelado y el post-procesamiento |
| <i>yardstick</i> | Medidas del rendimiento predictivo de los modelos |
| <i>tune</i> | Optimización de los parámetros del modelo |
| <i>dials</i> | Tratamiento de los parámetros de tuning en rejillas |
| <i>broom</i> | Convierte la información en formatos más manejables |

Análisis de Datos Censurados con *tidymodels*

- La librería ***censored*** es la extensión del paquete ***parsnip***, que contiene los **modelos predictivos para datos censurados**, y por tanto, hay que instalarla, y cargarla al inicio
- Se recomienda instalar la **última versión** de la librería ***survival***, para sustituir la que viene en la instalación de R
- Sin embargo, el paquete ***yardstick***, que es el que contiene las **medidas de capacidad predictiva**, ya contiene las correspondientes medidas para datos censurados



Preprocesamiento. Paquete *recipes* de *tidymodels*

- El paquete ***recipes*** puede combinar en un **único objeto** distintas técnicas de **tratamiento** de variables y **tareas de pre-procesamiento**
- Un ***recipe***, “receta”, es un objeto que incluye la definición de los **pasos del pre-procesamiento de datos**
 - Es **una especificación** de los pasos a seguir, pero **no se ejecutan** en el momento de definirlos
 - Este objeto puede ser aplicado después a diferentes **conjuntos de datos**
- Los pasos del pre-procesamiento se definen con **funciones** de tipo ***step_****()
- A las variables implicadas se les llama “**ingredientes**” de la receta, y pueden ser la variable respuesta, “outcome”, o las variables predictoras, variables que tienen **diferentes roles**
 - Las variables pueden ser referenciados en conjunto, con funciones del tipo: ***all_predictors()***, ***all_numeric_predictors()***, ***all_nominal_predictors()***, ***all_nominal()***, ***all_numeric()***, ***all_outcomes()***,

Funciones *step_**()

| Función | Descripción |
|--|--|
| <i>step_normalize()</i> | Normaliza las variables numéricas a desviación estándar 1 y media 0 |
| <i>step_BoxCox()</i> | Transformaciones de Box y Cox para conseguir simetría |
| <i>step_corr()</i> | Borra variables con fuertes correlaciones con las demás |
| <i>step_pca()</i> <i>step_pls()</i> | Sustituye las variables por las componentes principales (feature extraction) o componentes PLS |
| <i>step_interact()</i> | Crea términos de interacción entre variables predictoras |
| <i>step_ns()</i> | Crea términos del tipo spline, para relaciones no lineales |

Funciones *step_**()

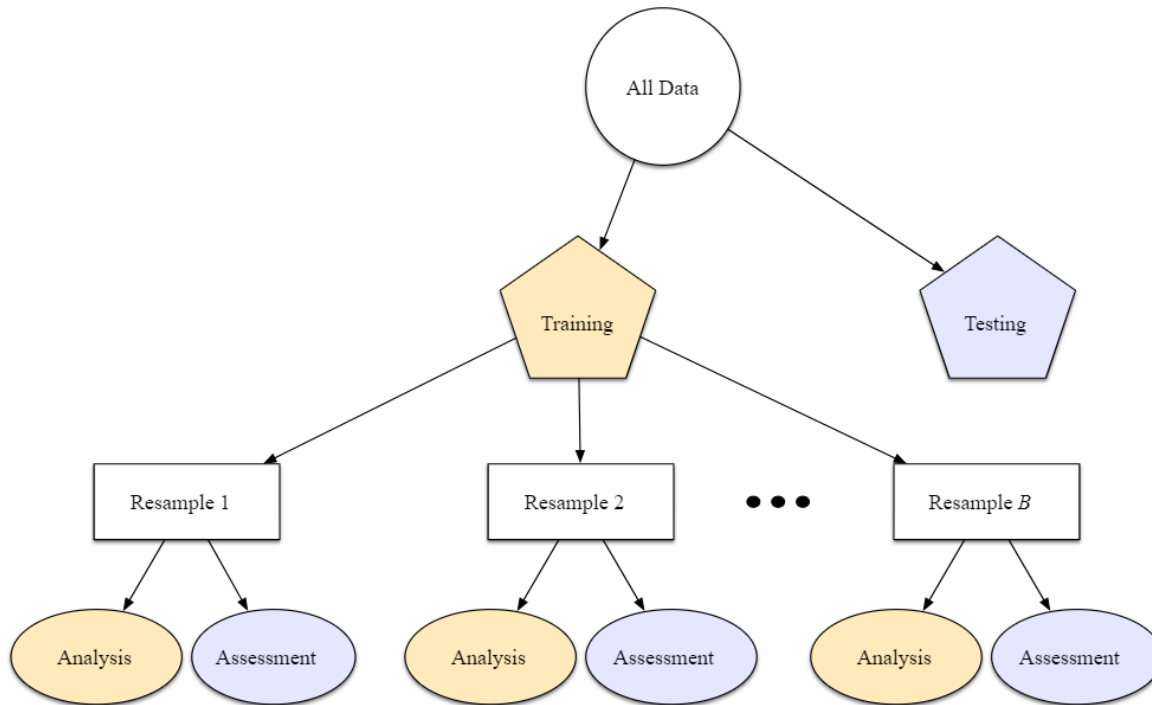
| Función | Descripción |
|---|---|
| <i>step_zv()</i> <i>step_nzv()</i> | Elimina las variables con varianza cero, o varianza “casi cero” |
| <i>step_dummy()</i> | Crea variables dummy para las variables categóricas |
| <i>step_unknow()</i> | Crea un nivel para los NAs en una variable categórica |
| <i>step_other()</i> | Crea una categoría “other” uniendo las categorías que tienen poca frecuencia |
| | |
| <i>step_impute_XXX()</i> | Diferentes métodos de imputación de missings (knn, mean, median, mode, linear, ...) |

Paquete *recipes* de *tidymodels*

- Las **operaciones básicas** que hay que hacer con un objeto **recipe** son las siguientes:
 - La función ***recipe()*** crea el objeto, **especificando los pasos** que se desean aplicar, y es dónde se asignan los roles de las variables, respuesta o predictores
 - La función ***prep()*** prepara el objeto para poder aplicarlo. **Calcula las operaciones** que son necesarias
 - La función ***bake()*** **aplica** este último objeto a un **data frame** concreto, creando **otro data frame** con los datos procesados
- La **opción recomendable** es incluir directamente el objeto **recipe** dentro de un **workflow** sin tener que crear *dataframes* con los datos procesados
 - En este caso no es necesario usar las funciones ***prep()*** y ***bake()***, ya que se ejecutan internamente cuando se ejecute el **workflow**

Esquema de Análisis. Técnicas de remuestreo

Imagen del libro
“Tidy Modeling with R”



- Partición inicial en una **muestra de training** y **muestra de testing**
- **Optimización de los parámetros** del modelo dentro de la muestra de training, con técnicas de remuestreo, **validación cruzada**. A las submuestras que participan en este proceso se les llama de **análisis y evaluación**

Paquete *rsample*

- El paquete ***rsample*** de ***tidymodels*** contiene las funciones de **creación de las muestras** necesarias para usar las **técnicas de remuestreo**
- Todas las funciones de remuestreo permiten evaluar **todas las fases del proceso** (pre-procesamiento, optimización de parámetros, construcción, ...), usando ***workflows***
- Todas las funciones de remuestreo contienen un parámetro que es ***strata***= que permite realizar las particiones manteniendo las proporciones de las categorías de una variable categórica (**remuestreo estratificado**)
 - Si la variable por la que se desea estratificar es **continua**, se usan sus **cuartiles**
- La mayoría de los procesos de remuestreo permiten **paralelización**, ya que los modelos construidos son independientes unos de otros. ***tidymodels*** lo integra de una forma sencilla

Funciones del paquete *rsample*

| Función | Descripción |
|--|---|
| <i>initial_split()</i> | Divide la muestra en dos submuestras, training y testing |
| <i>training()</i> <i>testing()</i> | Construye los dataframes conteniendo las muestras de training y testing, especificadas por <i>initial_split()</i> |
| <i>vfold_cv()</i> | Validación cruzada con “v” particiones |
| <i>loo_cv()</i> | Validación cruzada leave-one-out |
| <i>mc_cv()</i> | Validación cruzada Monte Carlo (MCCV) |
| <i>bootstraps()</i> | Bootstrapping |
| <i>analysis()</i> <i>assessment()</i> | Construye los dataframes conteniendo los conjuntos de análisis y evaluación, dentro de una técnica de remuestreo |

- Todas estas funciones devuelven **dataframes**, con las particiones de análisis y evaluación construidas
- Esos dataframes se pueden salvar o integrar dentro de un **workflow**

Librerías

```
> library(survival)
> library(tidymodels)

— Attaching packages — tidymodels 1.2.0 —
✓ broom      1.0.6      ✓ recipes    1.0.10
✓ dials      1.2.1      ✓ rsample    1.2.1
✓ dplyr      1.1.4      ✓ tibble     3.2.1
✓ ggplot2    3.5.1      ✓ tidyr      1.3.1
✓ infer      1.0.7      ✓ tune       1.2.1
✓ modeldata  1.4.0      ✓ workflows  1.1.4
✓ parsnip    1.2.1      ✓ workflowsets 1.1.0
✓ purrr      1.0.2      ✓ yardstick  1.3.1

— Conflicts — tidymodels_conflicts() —
✖ purrr::discard() masks scales::discard()
✖ dplyr::filter()   masks stats::filter()
✖ dplyr::lag()      masks stats::lag()
✖ recipes::step()   masks stats::step()
• Search for functions across packages at https://www.tidymodels.org/find/
> library(tidyverse)
> library(censored)
> library(smoothHR)    ## Fichero de datos
> library(survminer)   ## KM Plot
Cargando paquete requerido: splines
> tidymodels_prefer()
```

- Se cargan las librerías
- ***tidymodels_prefer()*** resuelve los conflictos a favor de los paquetes de ***tidymodels***

Lectura del fichero de datos

```
> ## Lectura del fichero, que está en la librería "smoothHR"
> ## Selección de columnas
> xx_all <- whas500 %>% select(age, gender, hr, sysbp, diasbp, bmi, cvd, afb,
+                               sho, chf, av3, miord, mitype, lenfol, fstat)
> xx_all = as_tibble(xx_all)
> dim(xx_all)
[1] 500 15
>
> ## Tiempo de días a meses
> xx_all$lenfol = xx_all$lenfol / 30.4375
>
> ## Variable respuesta en Análisis de Supervivencia se construye con Surv()
> head( Surv( xx_all$lenfol, xx_all$fstat ) )
[1] 71.55646817+ 71.35934292+ 71.95071869+ 9.75770021 70.01232033+ 0.03285421
```

- Se seleccionan las variables que se van a usar, predictores y respuesta, del fichero **whas500** que está en la librería **smoothHR**
- Hay **500 observaciones** y 15 variables, de las cuales **13 son predictores**
- El **tiempo** de supervivencia se transforma a **meses**
- La función **Surv()** se usa para construir la **variable respuesta** en el análisis de datos censurados, especificando las variables **tiempo y status** (evento/censura). Las observaciones marcadas con el signo “+” son las censuras

Partición inicial en muestras de training y testing

```
> ## Se crea las especificaciones de la partición
> set.seed(47) ## Se fija una semilla, para reproducir la partición
> sp_split <- initial_split(xx_all, prop = 0.70, strata = fstat)
> sp_split
<Training/Testing/Total>
<349/151/500>
>
> ## Se crean los dataframe de training y testing
> xx_train <- training(sp_split)
> xx_test <- testing(sp_split)
> dim(xx_train)
[1] 349 15
> dim(xx_test)
[1] 151 15
> ## Se borra el dataframe original
> rm(xx_all)
```

- Se usa la función ***initial_split()*** del paquete ***rsample*** para crear **las especificaciones de la partición**, donde el 70% de las observaciones serán de training y el 30% de testing
- Se usa la **variable status (*fstat*)** como ***strata***, para que **los eventos** se repartan en las dos muestras, de una forma proporcional
- Las funciones ***training()*** y ***testing()*** generan las muestras, los *dataframes*
- Se borra el dataframe original, para que no participe en ninguna fase del proceso

Tratamiento de datos

```
> ## Se extraen todos los tiempos de supervivencia ordenados, de la muestra de training
> all_time_survival <-
+   xx_train %>%
+   filter(fstat == 1) %>%
+   pull(lenfol) %>% unique %>% sort
> all_time_survival
[1] 0.03285421 0.06570842 0.09856263 0.13141684 0.16427105 0.19712526
. . .
[115] 63.27720739 64.19712526 70.96509240 77.47022587
>
> ## Creación de la variable respuesta, para el Análisis de Supervivencia
> xx_train <- xx_train %>%
+   mutate(surv_var = Surv(lenfol , fstat), .keep = "unused")
> xx_test <- xx_test %>%
+   mutate(surv_var = Surv(lenfol , fstat), .keep = "unused")
```

- Se salvan todos los **tiempos de supervivencia** ordenados, donde se han producido eventos, en la muestra de **training**. Estos son los tiempos en los que se calcula el estimador Kaplan-Meier, y se utilizarán más tarde para gráficos y predicciones
- Se crea la **variable respuesta**, a la que llamamos **surv_var**, con la función **Surv()** en las muestras de training y testing, y se borran las dos originales, que indicaban el tiempo de supervivencia o de seguimiento, y el status.
- Es la recomendación de **tidymodels**. Esto se hace para facilitar el pre-procesamiento y el uso de algunas funciones

Esquema de remuestreo

```
> ## Especificaciones de la técnica de remuestreo
> set.seed(52) ## Se fija una semilla, para reproducir la partición
> cv_split <- vfold_cv(xx_train, v = 10, repeats=2)
> cv_split
# 10-fold cross-validation repeated 2 times
# A tibble: 20 × 3
   splits          id      id2
   <list>        <chr>   <chr>
1 <split [314/35]> Repeat1 Fold01
2 <split [314/35]> Repeat1 Fold02
3 <split [314/35]> Repeat1 Fold03
. . .
18 <split [314/35]> Repeat2 Fold08
19 <split [314/35]> Repeat2 Fold09
20 <split [315/34]> Repeat2 Fold10
```

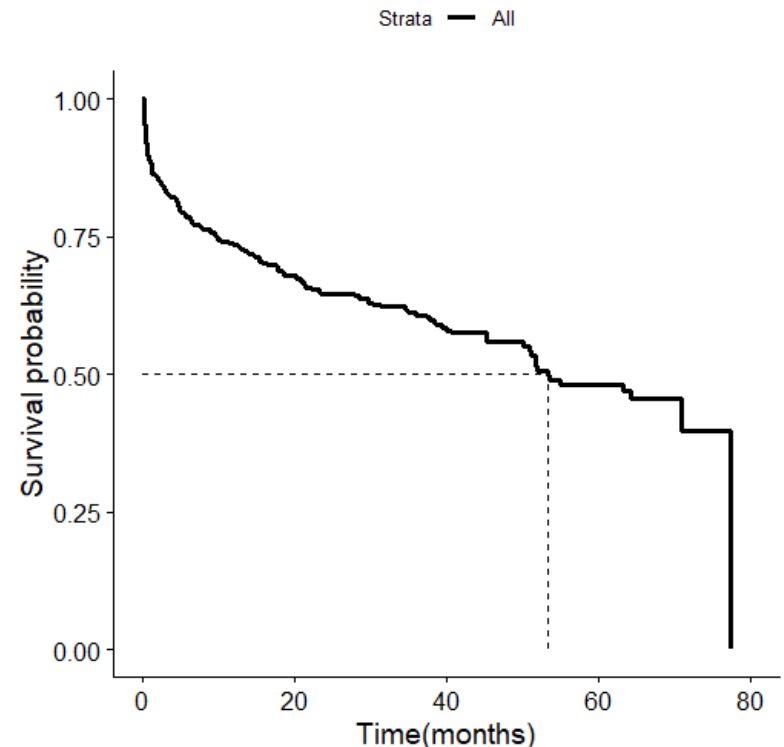
- Con la función ***vfold_cv()*** se fija la técnica de remuestreo, que va a ser **validación cruzada** con 10 particiones, **10-fold CV**, **repetida 2 veces** y se indica que las construya en la muestra de training
- Las **20 particiones** que se generan se van a usar en **todos los procesos** de optimización de parámetros de todas las técnicas (regresión penalizada, random forest, ..). Es importante si se quiere comparar los resultados

Estimador de Kaplan-Meier

```
> ## KM estimator
> surv_all <- survfit( surv_var ~ 1 , data=xx_train )
> ggsurvplot(surv_all, xlab="Time(months)", conf.int = FALSE, censor = FALSE,
+           size = 1.2, palette="black", surv.median.line = "hv" )
> surv_all
```

| | n | events | median | 0.95LCL | 0.95UCL |
|------|-----|--------|--------|---------|---------|
| [1,] | 349 | 150 | 53.4 | 50.2 | NA |

- Con la función **survfit()** se obtiene el **estimador de Kaplan-Meier** de la función de supervivencia, y se usa la función **ggsurvplot()** para mostrala
- Describe el **ritmo** en el que se produce el evento a lo largo del rango del tiempo
- Se observa una **caída brusca** al final. Esto sucede porque el tiempo de supervivencia mayor corresponde a un evento
- La **mediana de supervivencia** es **53.4 meses**. Es el tiempo en el que el 50% de los pacientes siguen vivos



Paquete *parsnip*

- El paquete ***parsnip*** proporciona una **interface estandarizada** para una gran variedad de **modelos**, que se usa también para sus extensiones, como el paquete ***censored***
- Todas las **funciones** de ***parsnip*** que construyen modelos tienen **dos parámetros** fundamentales:
 - ***set_engine()*** es el “motor”, dónde se especifica la **librería o función de R** con la que se va a construir el modelo
 - ***set_mode()*** es dónde se especifica **el tipo de variable respuesta**, si es un problema de “**classification**” o “**regression**”
 - Los modelos predictivos del paquete ***censored*** añaden un tercer tipo, que es “**censored regression**”
- Los **nombres de los parámetros** de las funciones de ***parsnip*** están unificados, para todos los paquetes que los usan

Modelos para Datos Censurados en *tidymodels*

- Los **modelos** (funciones) y los **engine** para **datos censurados** incluidos en ***censored / tidymodels*** están en esta tabla, donde se especifica también los **tipos de predicciones** que proporciona cada uno

| | model | engine | time | survival | linear_pred | raw | quantile | hazard |
|--|----------------------|----------------|------|----------|-------------|-----|----------|--------|
| Bagging y Boosting | bag_tree | rpart | ✓ | ✓ | × | × | × | × |
| | boost_tree | mboost | ✓ | ✓ | ✓ | × | × | × |
| Árboles | decision_tree | rpart | ✓ | ✓ | × | × | × | × |
| | decision_tree | partykit | ✓ | ✓ | × | × | × | × |
| Regresión de Cox | proportional_hazards | survival | ✓ | ✓ | ✓ | × | × | × |
| | proportional_hazards | glmnet | ✓ | ✓ | ✓ | ✓ | × | × |
| Random Forest | rand_forest | partykit | ✓ | ✓ | × | × | × | × |
| | rand_forest | aorsf | ✓ | ✓ | × | × | × | × |
| Regresión paramétrica y no paramétrica | survival_reg | survival | ✓ | ✓ | ✓ | × | ✓ | ✓ |
| | survival_reg | flexsurv | ✓ | ✓ | ✓ | × | ✓ | ✓ |
| | survival_reg | flexsurvspline | ✓ | ✓ | ✓ | × | ✓ | ✓ |

Parámetros de las funciones

| Modelo | Función | Parámetros |
|----------------------|------------------------|---|
| Regresión penalizada | proportional_hazards() | penalty mixture |
| Árboles de decisión | decision_tree() | tree_depth min_n cost_complexity |
| Random Forest | rand_forest() | trees mtry min_n |
| Bagging | bag_tree() | cost_complexity, tree_depth, class_cost, min_n |
| Boosting | boost_tree() | mtry, trees, min_n, tree_depth, learn_rate, loss_reduction, sample_size, stop_iter |


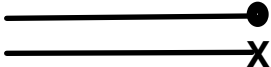

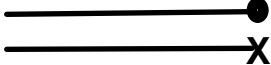
Funciones generales de *tidymodels*

| Función | Descripción |
|--|---|
| <i>fit()</i> | Ajusta un modelo |
| <i>extract_fit_engine()</i> | Extrae el modelo del paquete original de R usado |
| <i>predict()</i> | Obtiene las predicciones |
| <i>fit_resamples()</i> | Cálculo de medidas de capacidad predictiva con remuestreo |
| <i>tune_grid()</i> | Optimización de parámetros |
| <i>collect_metrics()</i> <i>show_best()</i> <i>select_best()</i> | Permite explorar los resultados de la optimización |

- Para construir modelos con **datos censurados**, se usan las mismas funciones de *tidymodels*

C-index

- El **C-index** es una **medida de concordancia** que se usa para evaluar la **capacidad predictiva** de un **modelo de supervivencia**
- El **C-index** es el número de **concordancias** entre lo observado y lo predicho, dividido por el número de comparaciones que se han establecido
 - Se omiten aquellos pares en los que el tiempo más corto es censurado

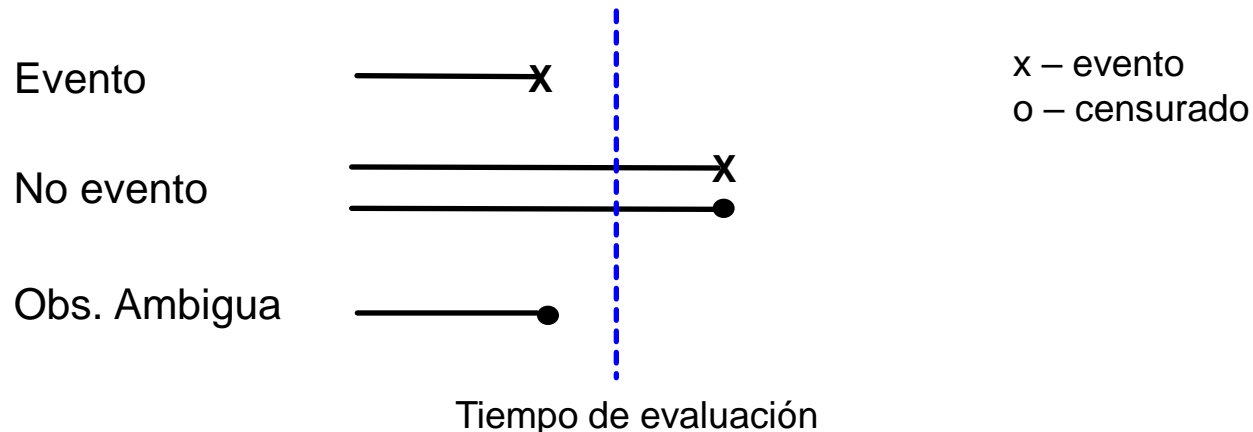
| Caso | Observado (tiempo y evento) | Predicho LP(A) > LP(B) | Predicho LP(B) > LP(A) |
|------|--|---------------------------|---------------------------|
| 1 | Obs A  Obs B  | Concordancia | Discordancia |
| 2 | Obs A  Obs B  | ?? | ?? |

x – evento
o – censurado

LP = predictor
lineal (Cox)

Medidas Dinámicas de Capacidad Predictiva

- Hay otras medidas que permiten evaluar la **capacidad predictiva** del modelo en un determinado instante del **tiempo t fijado**, que se llama **tiempo de evaluación**. Se llaman **medidas dinámicas**, porque se pueden calcular en todos los tiempos que se deseen
- La idea básica es que para el **tiempo de evaluación t fijado**, se crea una **variable binaria** que indica si se ha **producido o no el evento** en ese tiempo



- Las **observaciones censuradas ambiguas**, donde el tiempo de seguimiento es menor que el tiempo de evaluación, **no entran en el análisis**

Medidas Dinámicas de Capacidad Predictiva

- Las observaciones consideradas como “**evento**” o “**no evento**”, entran en el análisis con **un peso**
 - Se usa básicamente el **inverso de la probabilidad de ser censurado**, que se suele calcular con la curva “reverse KM”, que estima la probabilidad de ser censurado en cada instante
 - Se **pondera más alto** a las observaciones con más probabilidad de **mantenerse en la muestra**, es decir, con menor probabilidad de ser censuradas
- Una vez definida la **variable binaria**, en un tiempo de evaluación concreto, se usan las **probabilidades de supervivencia predichas** por el modelo en ese tiempo, ya que son **probabilidades** correspondientes a los “**no evento**”
- Una vez definida la variable binaria observada y las probabilidades predichas, se pueden usar las medidas de **capacidad predictiva de clasificación** para variables **binarias**

Medidas Dinámicas de Capacidad Predictiva

- El **AUC de la curva ROC**, basada en el cálculo de la sensibilidad y especificidad para todos los puntos de corte. Es una **medida de discriminación**, mide la capacidad del modelo para separar las clases
- El **Brier Score** es semejante al **error cuadrático medio** entre los **valores observados**, codificados como 0 y 1, y las **probabilidades predichas**

$$BS = \frac{1}{N} \sum_{t=1}^N (f_t - o_t)^2$$

o_t es la clase observada 0 / 1

f_t es la probabilidad de la clase 1

- El **Brier Score** es una **medida de calibración**, de precisión entre las probabilidades predichas y las observadas
- Un **modelo perfecto** tiene Brier Score de **0**, y un modelo **no informativo** tiene un Brier Score de **0.25 (0.5**2)**
- Si se calcula en varios tiempos de evaluación, se puede generar un gráfico, y el área bajo la curva se llama **Brier Score integrado**

Medidas Capacidad Predictiva en *tidymodels*

| Función | Descripción |
|---|--|
| <i>concordance_survival()</i> | C-index, medida de concordancia en supervivencia |
| <i>roc_auc_survival()</i> | AUC(t) de la curva ROC en un tiempo t |
| <i>brier_survival()</i> | Brier Score(t) en un tiempo t |
| <i>brier_survival_integrated()</i> | Brier Score integrado |

- Estas funciones son del paquete ***yardstick*** de ***tidymodels***, que contiene las medidas de capacidad predictiva, también para datos censurados

Métodos de regresión penalizados

- Los estimadores “clásicos” en regresión producen **estimadores muy inestables** (valores muy altos y con mucha varianza) en problemas de **alta dimensionalidad** ($p \gg n$) y/o con **variables correlacionadas**
- Los **métodos de regresión penalizados** introducen una **penalización** en la función de pérdida a ser optimizada (loss function)
- El **modelo de regresión de Cox** que se formula es el mismo, pero los coeficientes de regresión son estimados por un **método diferente** al de la verosimilitud parcial

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

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

$$pl_{\text{lasso}}(\beta) = pl(\beta) - \lambda \cdot \sum_{j=1}^p |\beta_j|$$

pl es la función de verosimilitud parcial

- **Lambda** es un parámetro que determina el **peso de la penalización**. Es un **parámetro de tuning**
- Muchos de los **coeficientes** estimados por Lasso alcanzan el valor 0, y solo **unos pocos** quedan con valores **distintos de 0**. Es una técnica adecuada a 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**

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

pl es la función de verosimilitud parcial

- El modelo se compone de todas las variables

- **Elastic Net** combina las 2 penalizaciones

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

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

Construcción de un modelo con *tidymodels*

- La **construcción de un modelo predictivo** con *tidymodels* se puede estructurar en los siguientes pasos:
 - Cargar las librerías y lectura de los datos
 - Especificaciones del **pre-procesamiento**
 - Especificaciones del **modelo** a ajustar, con los parámetros a optimizar
 - Integrar el pre-procesamiento y el modelo en un **workflow**
 - Especificaciones de la **técnica de remuestreo**
 - Especificaciones del conjunto de **parámetros** a explorar
 - Ejecución de la función de **optimización de parámetros**
 - Se finaliza el workflow y se ajusta el **modelo final** con los **parámetros óptimos**
 - Se evalúa el modelo en la **muestra de testing**: obtención de las **predicciones** y cálculo de las **medidas de capacidad predictiva**

Paralelización

```
> ## S.O. ---- Windows
> R.Version()$platform
[1] "x86_64-w64-mingw32"
> cores <- parallel::detectCores()
> cores
[1] 8
> if (!grepl("mingw32", R.Version()$platform)) {
+   ## Linux
+   library(doMC)
+   registerDoMC(cores = cores - 1)
+ } else {
+   ## Windows
+   library(doParallel)
+   cl <- makePSOCKcluster(cores - 1)
+   registerDoParallel(cl)
+ }
Cargando paquete requerido: foreach
. . .
Cargando paquete requerido: iterators
Cargando paquete requerido: parallel
```

- Se carga el paquete correspondiente: **doMC** para Linux o **doParallel** para Windows, aunque este paquete puede ser usado por cualquier sistema operativo
- Para que las funciones de **tidymodels** relacionadas con remuestreo y optimización de parámetros usen **paralelización**, solo es necesario **registrar el número de clusters**. Al final de todo el proceso, hay que pararlos con **stopCluster(cl)**

Preprocesamiento de datos. Recipe

```
> ## 1.- Se crea la receta con los pasos del pre-procesamiento
> obj_rec <-
+   recipe( surv_var ~ . , data=xx_train ) %>%
+   step_normalize(all_numeric_predictors())
> obj_rec
```

— Recipe

— Inputs

Number of variables by role

outcome: 1

predictor: 13

— Operations

- Centering and scaling for: all_numeric_predictors()

- Se crea una receta, un **recipe**, con las especificaciones del preprocesamiento de datos, recogidas en funciones de tipo **step_***() en el orden en el que se ejecutarán
- En **recipe()** se incluye una **formula**, que indica el papel de cada variable: **surv_var** es la **variable respuesta**, y el resto son **variables predictoras**
- Con **step_normalize()** se normalizan todos los predictores continuos a media 0 y desviación estándar 1

Preprocesamiento de datos. Recipe

```
> ## Se prepara la receta
> obj_prep <- prep(obj_rec, training = xx_train)
>
> ## Se aplica la receta a los dos data frame
> xx_train_proc <- bake(obj_prep, xx_train)
> xx_test_proc <- bake(obj_prep, xx_test)
> dim(xx_train)
[1] 349 14
> dim(xx_train_proc)
[1] 349 14
```

- Hay dos formas de usar el **preprocesamiento de datos**
- La primera, que NO vamos a usar aquí es crear **dos dataframes**, con las muestras de training y testing, con los **datos procesados** (este código que se explica aquí NO se ejecuta, está comentado en el script)
- Se “prepara” la receta con la función **prep()**, que realiza todos los **cálculos** necesarios en el dataframe especificado, el de training (medias y SDs, en este caso)
- La función **bake()** aplica esos cálculos a un dataframe, creando otro **dataframe**
- La segunda forma de usar el preprocesamiento de datos, es incluir **la receta en un workflow**, que va también a contener la optimización de parámetros con una técnica de remuestreo, y la receta se aplicará a **las muestras de análisis del remuestreo**

Regresión Penalizada. Elastic Net con tidymodels

```
> ## 2.- Especificaciones del modelos: elastic net con datos censurados
> enet_spec <-
+   proportional_hazards(penalty = tune(), mixture = tune()) %>%
+   set_engine("glmnet") %>%
+   set_mode("censored regression")
> enet_spec
Proportional Hazards Model Specification (censored regression)
Main Arguments:
  penalty = tune()
  mixture = tune()
Computational engine: glmnet

> ## Información de los parámetros del modelo elastic net
> penalty()
Amount of Regularization (quantitative)
Transformer: log-10 [1e-100, Inf]
Range (transformed scale): [-10, 0]
> mixture()
Proportion of Lasso Penalty (quantitative)
Range: [0, 1]
```

- Se usa la función ***proportional_hazards()*** para indicar las **especificaciones** del modelo, indicando en el ***set_engine()*** que use la función ***glmnet*** para construir el modelo de regresión penalizada **elastic net**
- Además, se indica que se desea **optimizar** (***tune()***), los parámetros **penalty** (penalización lambda) y **mixture** (alpha). Se pide información sobre estos parámetros

Regresión Penalizada. Optimización de parámetros

```
> ## 3.- Se crea el workflow
> wflow <- workflow() %>%
+   add_model(enet_spec) %>%
+   add_recipe(obj_rec)
>
> ## 4.- Se crea un grid con los parámetros a explorar
> enet_grid <- grid_regular(penalty(), mixture(),
+                           levels = list(penalty = 50, mixture = 6) )
> enet_grid %>% print(n=5)
# A tibble: 300 × 2
   penalty mixture
   <dbl>     <dbl>
1 1e-10      0
2 1.60e-10   0
3 2.56e-10   0
4 4.09e-10   0
5 6.55e-10   0
# i 295 more rows
```

- Se crea un **workflow** con la receta del **procesamiento** y el **modelo** elastic net
- Se usa la función **grid_regular()** para crear un grid con **todas las combinaciones** entre 50 parámetros de penalización y 6 alphas (0, 0.2, 0.4, 0.6, 0.8, 1) espaciados. En total, hay 300 combinaciones

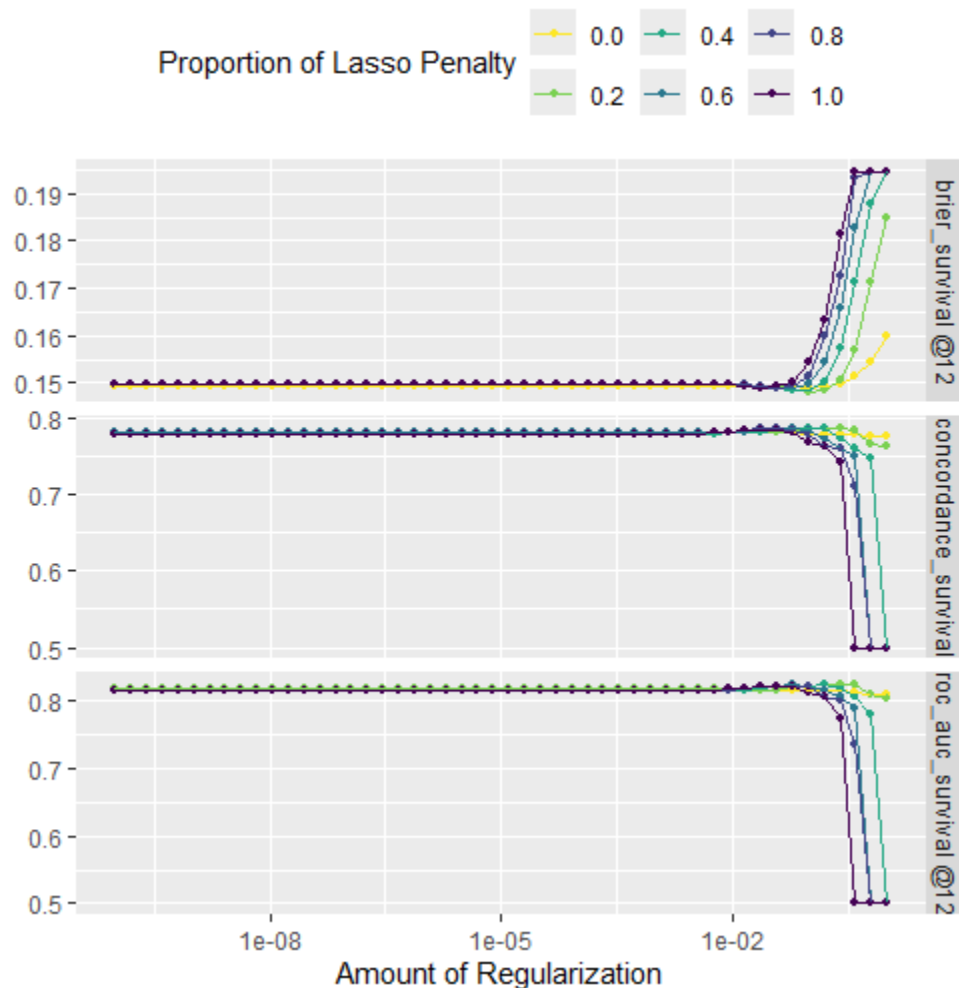
Regresión Penalizada. Optimización de parámetros

```
## Se ejecuta la optimización de parámetros
> keep_pred <- control_grid(save_pred = TRUE) ## salva las predicciones
+
> tune_result_enet <- wflow %>%
+   tune_grid( resamples = cv_split,
+             grid = enet_grid,
+             control = keep_pred,
+             metrics = metric_set(concordance_survival, brier_survival,
+                                 roc_auc_survival),
+             eval_time = 12)
```

- La función ***tune_grid()*** ejecuta la **optimización de los parámetros** del modelo, que se “aplica” al **workflow**, que recogía las especificaciones del modelo **elastic net** y el **preprocesamiento** de los datos
- Los **parámetros** de la función indican el esquema de remuestreo (***resamples=***), los valores de los parámetros a explorar (***grid=***), y que se salven las predicciones de todos los modelos evaluados (***control=***)
- Se indican las **medidas de capacidad predictiva** a evaluar (***metrics=***), que son el c-index, el Brier score y el AUC. Estas dos últimas evaluadas a 1 año (***eval_time=12***)
- Básicamente, en cada una de las **muestras de análisis** definidas por el remuestreo, se aplica el **preprocesamiento** de datos, y se ajusta un modelo **elastic net**. Se calculan las **medidas** con las **predicciones** obtenidas en las **muestras de evaluación**

Regresión Penalizada. Optimización de parámetros

```
> ## Plot  
> autoplot(tune_result_enet) +  
+   scale_color_viridis_d(direction = -1) +  
+   theme(legend.position = "top")
```



Regresión Penalizada. Optimización de parámetros

```
> ## Se analizan los resultados
> tune_result_enet %>%
+   collect_metrics()
# A tibble: 900 × 9
   penalty mixture .metric      .estimator .eval_time  mean      n std_err
   <dbl>   <dbl> <chr>      <chr>      <dbl> <dbl> <int> <dbl>
1 1     e-10    0 brier_survival standard    12 0.149    20 0.00671
2 1     e-10    0 roc_auc_survival standard    12 0.817    20 0.0144
3 1     e-10    0 concordance_survi... standard    NA 0.781    20 0.00803
4 1.60e-10    0 brier_survival standard    12 0.149    20 0.00671
5 1.60e-10    0 roc_auc_survival standard    12 0.817    20 0.0144
6 1.60e-10    0 concordance_survi... standard    NA 0.781    20 0.00803...
7 2.56e-10    0 brier_survival standard    12 0.149    20 0.00671
8 2.56e-10    0 roc_auc_survival standard    12 0.817    20 0.0144
9 2.56e-10    0 concordance_survi... standard    NA 0.781    20 0.00803
10 4.09e-10    0 brier_survival standard    12 0.149    20 0.00671
# i 890 more rows
```

- La función ***collect_metrics()*** devuelve un ***tibble*** con todas las medidas
- Contiene 900 filas (6 mixture x 50 penalty x 3 medidas de capacidad predictiva)
- Por ejemplo, para $\lambda = e-10$ y $\alpha = 0$, el AUC de la curva ROC evaluado en $t=12$, es 0.817 y esa medida tiene un error estándar de 0.0144, error estimado de las 20 veces que se ha calculado (10-fold CV, repetido 2 veces)

Regresión Penalizada. Mejores Modelos

```
> ## Los mejores modelos, con mayores c-index
> show_best(tune_result_enet, metric="concordance_survival")
# A tibble: 5 × 9
  penalty mixture .metric .estimator .eval_time mean n std_err
  <dbl> <dbl> <chr> <chr> <dbl> <dbl> <int> <dbl>
1 0.153 0.2 concordance_survival standard NA 0.787 20 0.00749
2 0.0373 0.8 concordance_survival standard NA 0.787 20 0.00714
3 0.0596 0.4 concordance_survival standard NA 0.787 20 0.00710
4 0.0373 0.6 concordance_survival standard NA 0.787 20 0.00681
5 0.244 0.2 concordance_survival standard NA 0.786 20 0.00756
> ## Modelo con máximo c-index
> tune_best_enet <- tune_result_enet %>% select_best(metric = "concordance_survival")
> tune_best_enet$penalty
[1] 0.1526418
> tune_best_enet$mixture
[1] 0.2
```

- Se pueden mostrar los **mejores modelos** con mayor **c-index** con **show_best()** y extraer el mejor, con el máximo, con **select_best()**
- El mejor modelo tiene un c-index de **0.787** (SE 0.00749), y corresponde al modelo de **elastic net** con **alpha=0.2** y **lambda = 0.153**

Regresión Penalizada. Mejores Modelos

```
> show_best(tune_result_enet, metric="roc_auc_survival", eval_time = 12 )
# A tibble: 5 × 9
  penalty mixture .metric      .estimator .eval_time  mean      n std_err
  <dbl>    <dbl> <chr>      <chr>      <dbl> <dbl> <int>  <dbl>
1  0.153    0.4 roc_auc_survival standard    12  0.825    20  0.0137
2  0.244    0.2 roc_auc_survival standard    12  0.824    20  0.0142
3  0.391    0.2 roc_auc_survival standard    12  0.824    20  0.0131
4  0.153    0.2 roc_auc_survival standard    12  0.824    20  0.0142
5  0.0596   0.6 roc_auc_survival standard    12  0.823    20  0.0139
>
```

- Se puede elegir otra métrica, por ejemplo, el AUC de la curva ROC de supervivencia en el tiempo = 12, y los mejores modelos son diferentes, pero algunas combinaciones de parámetros aparecen en el top 5 de ambas medidas
- En ambos casos, se observa que hay varias combinaciones de parámetros que prácticamente tienen el mismo poder predictivo

Regresión Penalizada. Modelo final

```
> ## Se crea al workflow final
> final_wflow <-
+   wflow %>%
+   finalize_workflow( select_best(tune_result_enet, metric="concordance_survival") )
> ## Se crea al model final
> enet_fit <-
+   final_wflow %>%
+   fit(xx_train)
> enet_fit
      Df  %Dev  Lambda
1     0  0.00 1.61400
2     1  0.27 1.47000
3     1  0.54 1.34000
4     2  0.93 1.22100
5     2  1.38 1.11200
.     .  .     .
```

- La función ***tune_grid()*** no construye el modelo final con los parámetros óptimos, y hay que crear un nuevo **workflow** con el **modelo final**, seleccionando los **parámetros óptimos** por la métrica deseada, lo que se hace con la función ***finalize_workflow()***
- El modelo especificado en este workflow, se ajusta con la función ***fit()*** detallando el *dataframe* sobre el que se desea hacer
- El modelo final proporcionado realmente se ajusta con varios lambdas (***glmnet*** lo hace siempre así, por defecto), pero hay un **control** del parámetro de penalización **óptimo**

Regresión Penalizada. Modelo final

```
> ## Coeficientes del modelo final
> tidy(enet_fit) %>% print(n=4)
# A tibble: 13 × 3
  term      estimate penalty
<chr>      <dbl>    <dbl>
1 age        0.422    0.153
2 gender      0      0.153
3 hr         0.146    0.153
4 sysbp       0      0.153
# i 9 more rows
> tidy(enet_fit) %>% filter( estimate != 0 ) %>% print(n=4)
# A tibble: 8 × 3
  term      estimate penalty
<chr>      <dbl>    <dbl>
1 age        0.422    0.153
2 hr         0.146    0.153
3 diasbp    -0.132    0.153
4 bmi       -0.155    0.153
# i 4 more rows
> ## Modelo final "glmnet"
> out_glmnet <- extract_fit_engine(enet_fit)
> class(out_glmnet)
[1] "coxnet" "glmnet"
```

- Con la función **tidy()** se muestran los coeficientes del modelo (con penalty = 0.153), pero hay que seleccionar los de las **8 variables** con coeficientes distintos de 0
- Con la función **extract_fit_engine()** se puede extraer el objeto del paquete **glmnet**

Regresión Penalizada. Predicciones

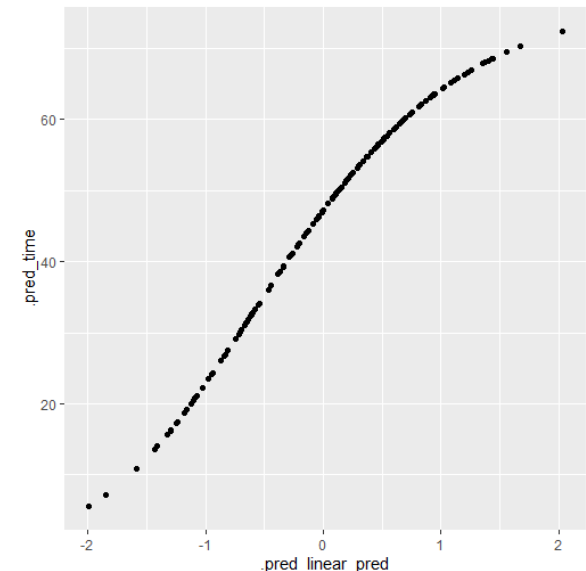
```
> ## Predicciones en Testing
> pred_enet_lin_pred <- predict(enet_fit, xx_test, type = "linear_pred")
> pred_enet_time <- predict(enet_fit, xx_test, type = "time")
> pred_enet_df <- bind_cols(xx_test %>% select(surv_var),
+                           pred_enet_lin_pred, pred_enet_time )
> pred_enet_df %>% print(n=4)
# A tibble: 151 × 3
  surv_var .pred_linear_pred .pred_time
  <Surv>    <dbl>          <dbl>
1 71.35934+ 0.929            63.3
2 49.14990 1.02             64.4
3 71.45791+ 0.646            59.4
4 77.30595 0.442            56.0
> ## Predicciones directas del modelo final proporcionado por "glmnet"
> head( predict( out_glmnet, newx = as.matrix( xx_test_proc %>% select( - surv_var ) ),
+             type="link", s = tune_best_enet$penalty ) [ , 1] )
[1] -0.9294891 -1.0228322 -0.6462901 -0.4418440 -1.2548174 -0.4093231
```

- La función ***predict()*** de ***tidymodels*** permite predecir el predictor lineal (***type="linear_pred"***)
- **Muy importante:** la función ***predict()*** de ***tidymodels*** cambia el signo al predictor lineal de la fórmula de Cox. Por tanto, se debe interpretar que los **valores altos en “este predictor lineal” tienen mejor supervivencia** (“low risk”)
- También, se puede obtener la predicción del **tiempo predicho** en el que se va a producir el evento (***type="time"***), pero es una predicción **difícil de interpretar**

Regresión Penalizada. Capacidad Predictiva

```
> ## c-index
> concordance_survival(pred_enet_df, truth = surv_var, estimate = .pred_linear_pred )
  .metric      .estimator .estimate
<chr>      <chr>      <dbl>
1 concordance_survival standard      0.750
> concordance_survival(pred_enet_df, truth = surv_var, estimate = .pred_time )
  .metric      .estimator .estimate
<chr>      <chr>      <dbl>
1 concordance_survival standard      0.750
>
> ## Relación entre el predictor lineal y el tiempo de supervivencia
> dev.new()
> pred_enet_df %>%
+   ggplot(aes(.pred_linear_pred, .pred_time)) +
+   geom_point()
```

- El **c-index**, en la muestra de testing es **0.750**, calculado con el predictor lineal, lo que es lo habitual con la regresión de Cox
- Si se usan las **predicciones del tiempo** de supervivencia, que es lo que hace *tidymodels*, el resultado es el mismo, ya que es una **transformación monótona** del predictor lineal



Regresión Penalizada. Capacidad Predictiva

```

> ## Tiempos de evaluación a explorar, hasta 5 años, con saltos de 6 meses
> time_points <- seq( 6, 60, by=6 )
> ## Predicciones en Testing
> pred_enet_time_df <- augment( enet_fit, xx_test, eval_time = time_points)
> pred_enet_time_df %>% print(n=4)
# A tibble: 151 × 16
  .pred      .pred_time    age gender    hr sysbp diasbp    bmi    cvd    afb    sho    chf
  <list>      <dbl> <int>  <int>  <int> <int>  <int>  <dbl> <int> <int> <int> <int>
1 <tibble>    63.3    49      0    84   120    60   24.0     1     0     0     0
2 <tibble>    64.4    55      0    91   147    95   27.1     1     0     0     0
3 <tibble>    59.4    54      0   104   166   106   25.5     1     0     0     0
4 <tibble>    56.0    51      1   133   166   134   24.4     0     0     0     1
# i 147 more rows
> pred_enet_time_df$.pred[[1]] ## 10 supervivencias predichas para la observación 1
# A tibble: 10 × 5
  .eval_time .pred_survival .weight_time .pred_censored .weight_censored
    <dbl>      <dbl>      <dbl>      <dbl>      <dbl>
1         6      0.925        6.00         1         1
2        12      0.904       12.0         1         1
3        18      0.880       18.0       0.800       1.25
4        24      0.857       24.0       0.736       1.36
5        30      0.848       30.0       0.736       1.36
. . .

```

- Para obtener las **medidas dinámicas de capacidad predictiva**, se fijan los tiempos de evaluación, hasta 5 años, con saltos de 6 meses (10 tiempos)
- Se muestran las **predicciones de supervivencia**, en esos tiempos de la observación 1

Regresión Penalizada. Brier Score

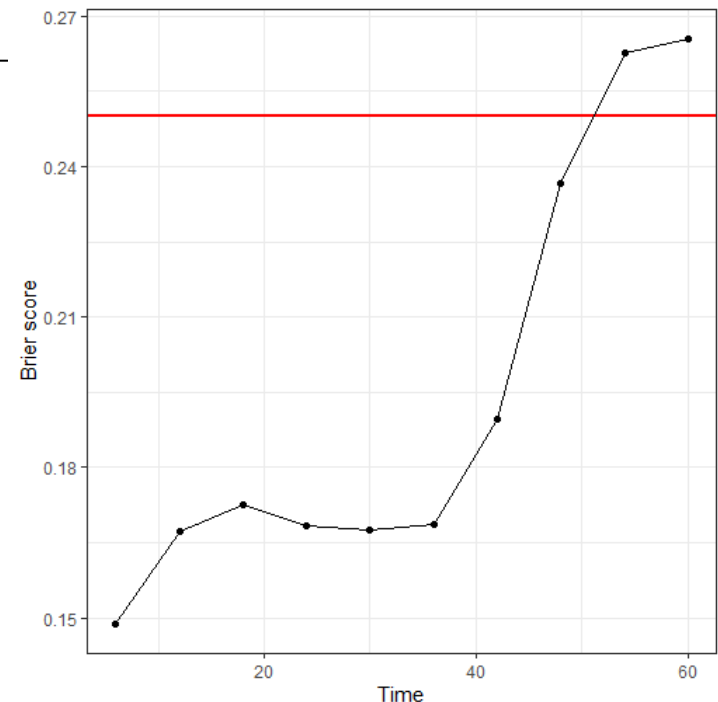
```
## Dynamic Brier Score
> brier_scores <-
+   brier_survival( pred_enet_time_df, truth = surv_var, .pred )
> brier_scores
# A tibble: 10 × 4
  .metric      .estimator .eval_time .estimate
  <chr>        <chr>      <dbl>    <dbl>
1 brier_survival standard      6    0.149
2 brier_survival standard     12    0.167
3 brier_survival standard     18    0.173
4 brier_survival standard     24    0.168
5 brier_survival standard     30    0.168
6 brier_survival standard     36    0.169
7 brier_survival standard     42    0.190
8 brier_survival standard     48    0.237
9 brier_survival standard     54    0.263
10 brier_survival standard     60    0.266
```

- Con la función ***brier_survival()*** se obtienen los **Brier Score** en los tiempos de evaluación, en los que se han calculado las predicciones (*.pred*)
- En los primeros tiempos, entre 6 y 36 meses, se obtiene los mejores resultados, valores menores del Brier Score

Regresión Penalizada. Brier Score

```
> brier_scores %>%
+   ggplot(aes(.eval_time, .estimate)) +
+   geom_hline(yintercept = 0.25, col="red", lwd = 1) +
+   geom_line() +
+   geom_point() +
+   labs(x = "Time", y="Brier score") +
+   theme_bw()
> ## Integrated Brier Scores
> brier_survival_integrated(pred_enet_time_df, truth = surv_var, .pred )
# A tibble: 1 × 3
  .metric          .estimator .estimate
  <chr>            <chr>      <dbl>
1 brier_survival_integrated standard    0.174
```

- Se ve claramente que el **Brier score** aumenta con el tiempo, es decir, el modelo pierde capacidad predictiva para predecir a largo plazo
- El modelo perfecto tiene 0, y un modelo no informativo tiene 0.25 (línea roja)
- El **Brier score integrado** (área de la curva), es de **0.174**, que es bastante moderado



Regresión Penalizada. AUC de la curva ROC

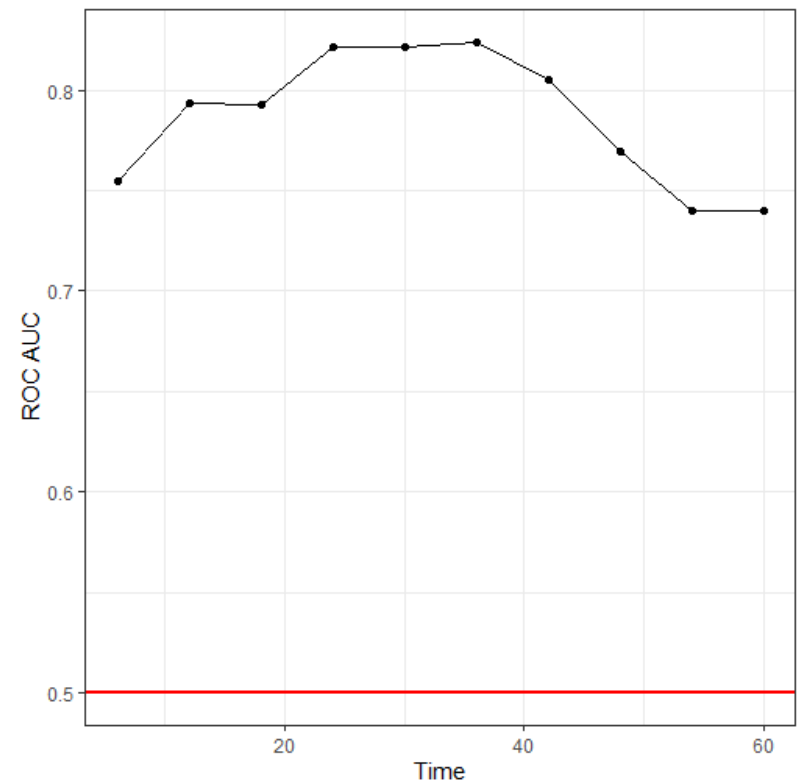
```
> ## Dynamic ROC curves
> roc_scores <-
+   roc_auc_survival( pred_enet_time_df, truth = surv_var, .pred )
> roc_scores
# A tibble: 10 × 4
  .metric      .estimator .eval_time .estimate
  <chr>        <chr>      <dbl>    <dbl>
1 roc_auc_survival standard      6    0.755
2 roc_auc_survival standard     12    0.793
3 roc_auc_survival standard     18    0.793
4 roc_auc_survival standard     24    0.822
5 roc_auc_survival standard     30    0.822
6 roc_auc_survival standard     36    0.824
7 roc_auc_survival standard     42    0.805
8 roc_auc_survival standard     48    0.769
9 roc_auc_survival standard     54    0.740
10 roc_auc_survival standard     60    0.740
```

- Con la función ***roc_auc_survival()*** se obtienen los **AUCs** de la curva ROC en los tiempos de evaluación, en los que se han calculado las predicciones (*.pred*)
- En los primeros, y sobre todo, en los tiempos intermedios, se obtienen las mejores predicciones

Regresión Penalizada. AUC de la curva ROC

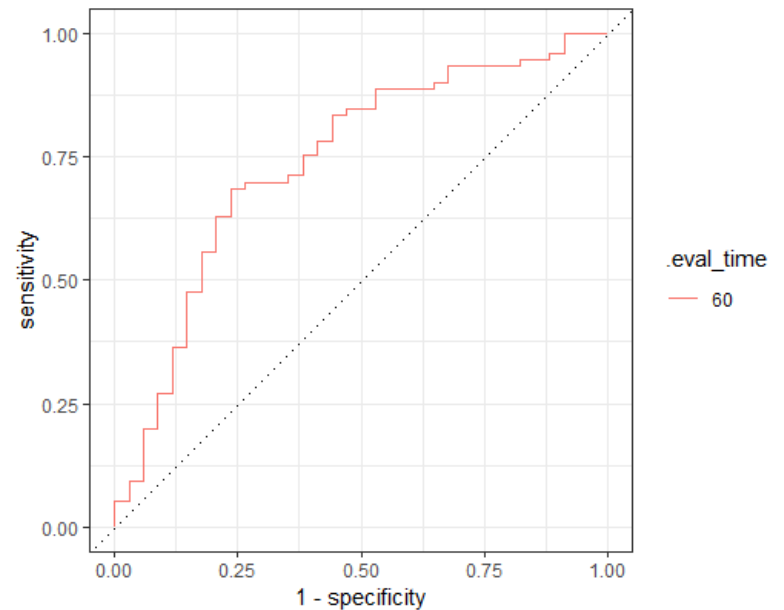
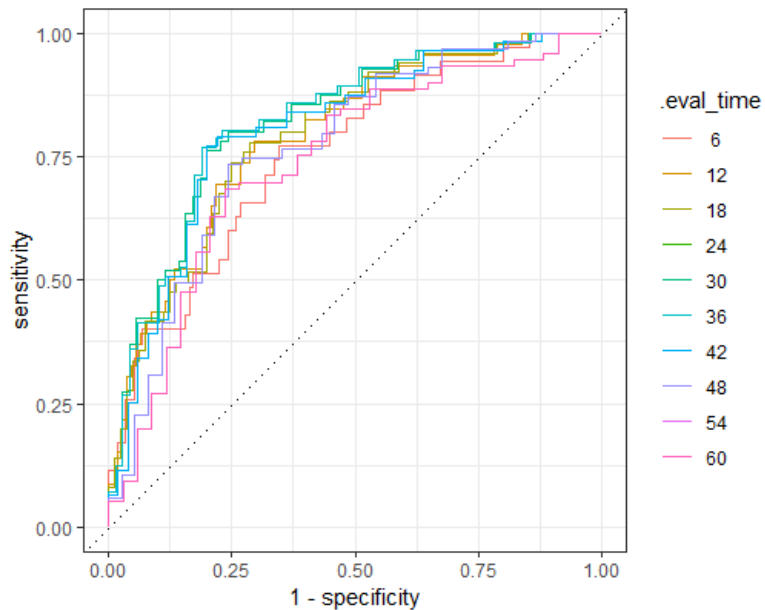
```
> ## Plot
> roc_scores %>%
+   ggplot(aes(.eval_time, .estimate)) +
+   geom_hline(yintercept = 0.5, col="red", lwd = 1) +
+   geom_line() +
+   geom_point() +
+   labs(x = "Time", y="ROC AUC") +
+   theme_bw()
```

- El **AUC** es mejor en los tiempos intermedios y en los más cortos, como ya indicaba el Brier Score, pero los valores del AUC fluctúan menos



Regresión Penalizada. Curva ROC

```
> ## Todas las curvas ROC
> all_roc_curve <-
+   roc_curve_survival(pred_enet_time_df, truth = surv_var, .pred )
> autoplot(all_roc_curve)
> ## Curva ROC en el tiempo t = 60
> pred_enet_time_60 <- augment( enet_fit, xx_test, eval_time = 60)
> roc_curve_60 <-
+   roc_curve_survival(pred_enet_time_60, truth = surv_var, .pred )
> autoplot(roc_curve_60)
```



- Se muestran las curvas ROC en los tiempos de evaluación, y se selecciona la de t=60

Grupos de Riesgo: High / Low Risk

- En la literatura clásica de Análisis de Supervivencia (Hosmer, Kleinbaum), se proponía evaluar los modelos de Cox definiendo **grupos de riesgo**, cuartiles o quintiles, definidos a partir del **predictor lineal**
 - Se mostraban las curvas KM de los grupos de riesgo de la muestra de training, y por tanto, esta evaluación estaba sobreajustada
- Actualmente, se suelen definir 2 grupos de riesgo: **High / Low Risk**
 - Los grupos se definen y evalúan en la **muestra de testing**, pero el punto de corte se elige con la **mediana del predictor lineal** en la **muestra de training**
 - Se muestran las **curvas Kaplan-Meier** y se calcula el **test log-rank**
- **Observación:** en estudios con **bajo tamaño muestral**, sin muestra de testing, esta evaluación se puede hacer con las predicciones obtenidas en el proceso de remuestro, obtenidas en la optimización de los parámetros (**cross-validated predictions**) (ANEXO)

Regresión Penalizada. Grupos de Riesgo

```
> ## Creación de los grupos en Testing con la mediana de Training
> pred_enet_lp_train <- predict(enet_fit, xx_train, type = "linear_pred")
> cutoff_lp <- median(pred_enet_lp_train$.pred_linear_pred)
>
> pred_enet_lin_pred_group <- as.integer( pred_enet_lin_pred$.pred_linear_pred >=
cutoff_lp )
> pred_enet_lin_pred_group <- factor(pred_enet_lin_pred_group, levels=0:1,
+                                   labels=c("High Risk", "Low Risk"))
> table(pred_enet_lin_pred_group)
pred_enet_lin_pred_group
High Risk  Low Risk
      73      78
```

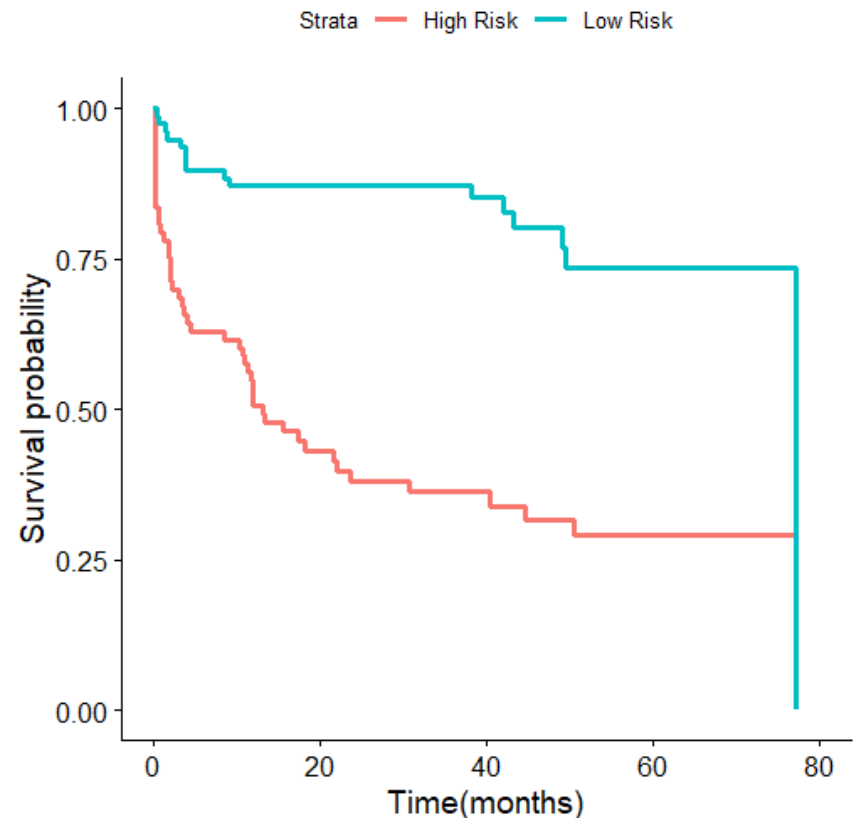
- Se obtiene la **mediana** del **predictor lineal** de las predicciones en la muestra de **training**
- Cada observación de la **muestra de testing** se asigna a los grupos de **High / Low Risk**, según esté por debajo o por encima de la mediana
- **Observación:** se está usando el **predictor lineal** de *tidymodels*, que tenía el **signo cambiado**, con lo que los **valores bajos** son los que corresponden a **riesgo alto**

Regresión Penalizada. Grupos de Riesgo

```
> ## KM Plot y Test log-rank
> surv_group <- survfit( surv_var ~ pred_enet_lin_pred_group, data=xx_test )
> survdiff( surv_var ~ pred_enet_lin_pred_group, data=xx_test)

Chisq= 35.9  on 1 degrees of freedom, p= 2e-09
>
> ggsurvplot(surv_group, xlab="Time(months)", conf.int = FALSE, censor = FALSE,
+           size=1.2, legend.labs = levels(pred_enet_lin_pred_group))
```

- Se obtiene las **gráficas KM** de los dos grupos de riesgo
- El **test log-rank** tiene **p.value < 0.001**



Regresión penalizada. Predicciones Supervivencia

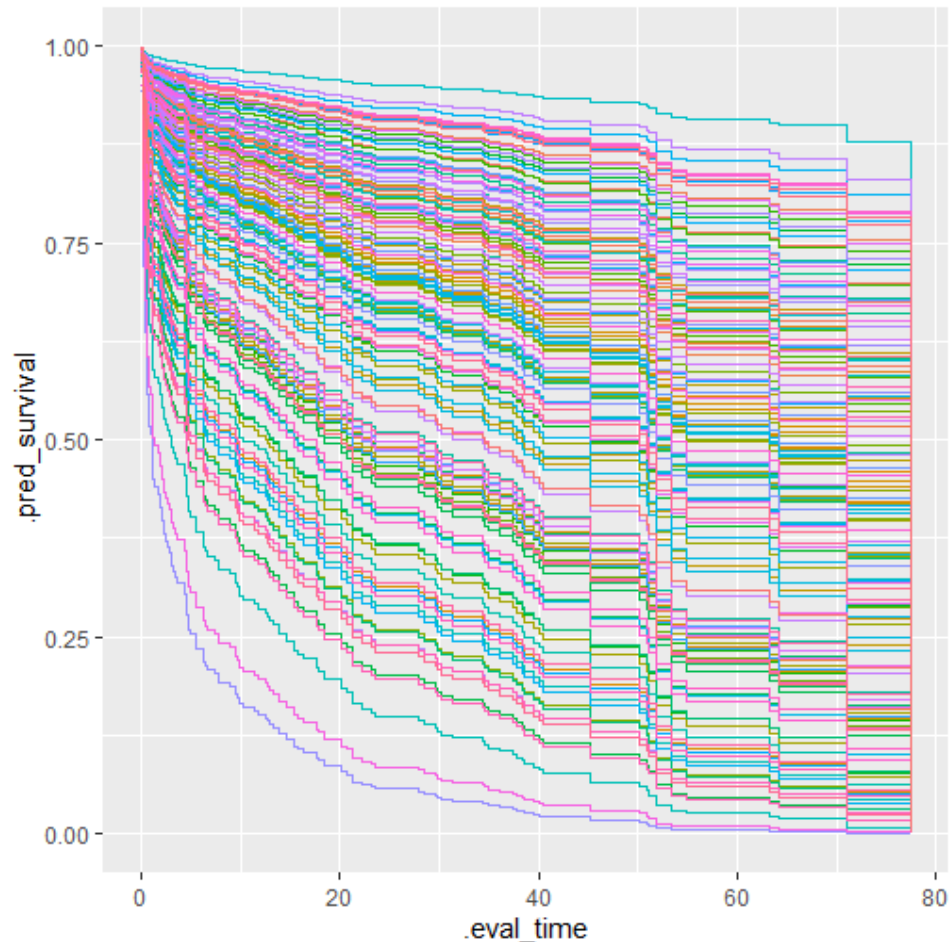
```
> ## Predicciones en todos los tiempos de supervivencia, donde se han producido eventos
> pred_enet_time_df_all <- augment( enet_fit, xx_test, eval_time = all_time_survival)
>
> ## Dataframe, con cada tiempo de cada observación en una fila, para el KM plot
> pred_enet_all_survival <- pred_enet_time_df_all %>%
+   mutate(id = factor(1:151)) %>%
+   unnest(cols = .pred)
```

- Se calculan, en la muestra de testing, las **predicciones del tiempo de supervivencia** en **todos los tiempos** de supervivencia donde se **han producido eventos** (almacenado en *all_time_survival*), con el modelo final de la **regresión penalizada**
- Se usa la función ***augment()*** que añade a un *dataframe* todas las predicciones, la de “*time*” y las de la función de supervivencia, en todos los tiempos de evaluación especificados
- Se prepara un *dataframe* para poder dibujar las **KM de todas las predicciones** en testing

Regresión penalizada. Predicciones Supervivencia

```
> ## KM Plot
> pred_enet_all_survival %>%
+   ggplot(aes(x = .eval_time, y = .pred_survival, col = id)) +
+   geom_step() +
+   theme(legend.position = "none")
```

- En esta gráfica se han representado las **151 predicciones** de testing
- Las **curvas no se cruzan** en ningún momento del tiempo, ya que están basadas en un modelo de regresión de Cox penalizado, que asume el **principio de riesgos proporcionales**
- **El orden** de las curvas de supervivencias es siempre el mismo si se toma un valor en un **tiempo puntual**



Árboles de Supervivencia. Survival trees

- Los **árboles de supervivencia** son la versión de los **árboles de decisión** de clasificación y de regresión, adaptados a los **datos censurados** o datos de supervivencia
- La construcción de los árboles 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

| | 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 |

Árboles de Supervivencia

- El **criterio de partición** más usado está basado en el **test log-rank**
 - En cada nodo, se calcula el **estadístico del test log-rank** para **todos los predictores**
 - 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, o combinación de categorías, con el **máximo estadístico** del test
- La **predicción** de los árboles de supervivencia en cada **nodo terminal** es la **función de supervivencia**, obtenida con el **estimador Kaplan-Meier**, de todas **las observaciones** que están en el nodo
- Se reporta normalmente la **mediana de la curva**, como **predicción numérica única** del nodo, pero la mediana no siempre se puede calcular

Árboles de Supervivencia con *tidymodels*

```
> ## 1.- Se crea la receta con los pasos del pre-procesamiento
> obj_rec_tree <-
+   recipe( surv_var ~ . , data=xx_train )
>
> ## 2.- Especificaciones del modelos: survival tree con datos censurados
> tree_spec <-
+   decision_tree( tree_depth = tune() , min_n = tune() ) %>%
+   set_engine("partykit") %>%
+   set_mode("censored regression")
>
> ## Información de los parámetros del modelo
> tree_depth()
Tree Depth (quantitative)
Range: [1, 15]
> min_n()
Minimal Node Size (quantitative)
Range: [2, 40]
>
```

- Se indica el **recipe** con la fórmula solamente, ya que en los árboles no hay que estandarizar las variables
- Se usa la función **decisión_tree()** especificando que use los árboles del paquete “**partykit**” y optimizando los dos parámetros: profundidad de los árboles (**tree_depth**) y el número mínimo de observaciones en los nodos terminales (**min_n**)

Árboles de Supervivencia con *tidymodels*

```
> ## 3.- Se crea el workflow
> wflow_tree <- workflow() %>%
+   add_model(tree_spec) %>%
+   add_recipe(obj_rec_tree)
>
> ## 4.- Se crea un grid con los parámetros a explorar
> tree_grid <- grid_regular(tree_depth(), min_n(), levels = 4 )
>
> ## Se ejecuta la optimización de parámetros
> tune_result_tree <- wflow_tree %>%
+   tune_grid( resamples = cv_split,
+             grid = tree_grid,
+             metrics = metric_set(concordance_survival, brier_survival,
+                                 roc_auc_survival),
+             eval_time = 12 )
```

- Se crea el **workflow** con el *recipe* y las especificaciones del modelo
- Se crea un **grid_regular** con 4 niveles para cada parámetro (**levels=4**)
- Se ejecuta la optimización de parámetros con **tune_grid()**
- Se indican las mismas **medidas de capacidad predictiva** a evaluar (**metrics=**), que son el c-index, el Brier score y el AUC. Éstas dos últimas evaluadas a 1 año (**eval_time=12**)

Árboles de Supervivencia con *tidymodels*

```
> show_best(tune_result_tree, metric="concordance_survival")
# A tibble: 5 × 9
  tree_depth min_n .metric .estimator .eval_time mean n std_err
    <int> <int> <chr>      <chr>      <dbl> <dbl> <int> <dbl>
1         5     2 concordance_surviv... standard      NA  0.713     20  0.0122
2        10     2 concordance_surviv... standard      NA  0.713     20  0.0122
3        15     2 concordance_surviv... standard      NA  0.713     20  0.0122
4         5    14 concordance_surviv... standard      NA  0.713     20  0.0122
5        10    14 concordance_surviv... standard      NA  0.713     20  0.0122
>
> ## Se crea al workflow final
> final_wflow_tree <-
+   wflow_tree %>%
+   finalize_workflow( select_best(tune_result_tree, metric="concordance_survival") )
>
> ## Se crea al model final
> tree_fit <-
+   final_wflow_tree %>%
+   fit(xx_train)
```

- El **mejor modelo** es el que tiene profundidad del árbol = 5 y mínimo tamaño en los nodos de 2. Este modelo tiene un **c-index** de **0.713** obtenido en el remuestreo
- Se finaliza el **workflow** final, y se ajusta el **modelo final** con los **parámetros óptimos**

Árboles de Supervivencia con *partykit*

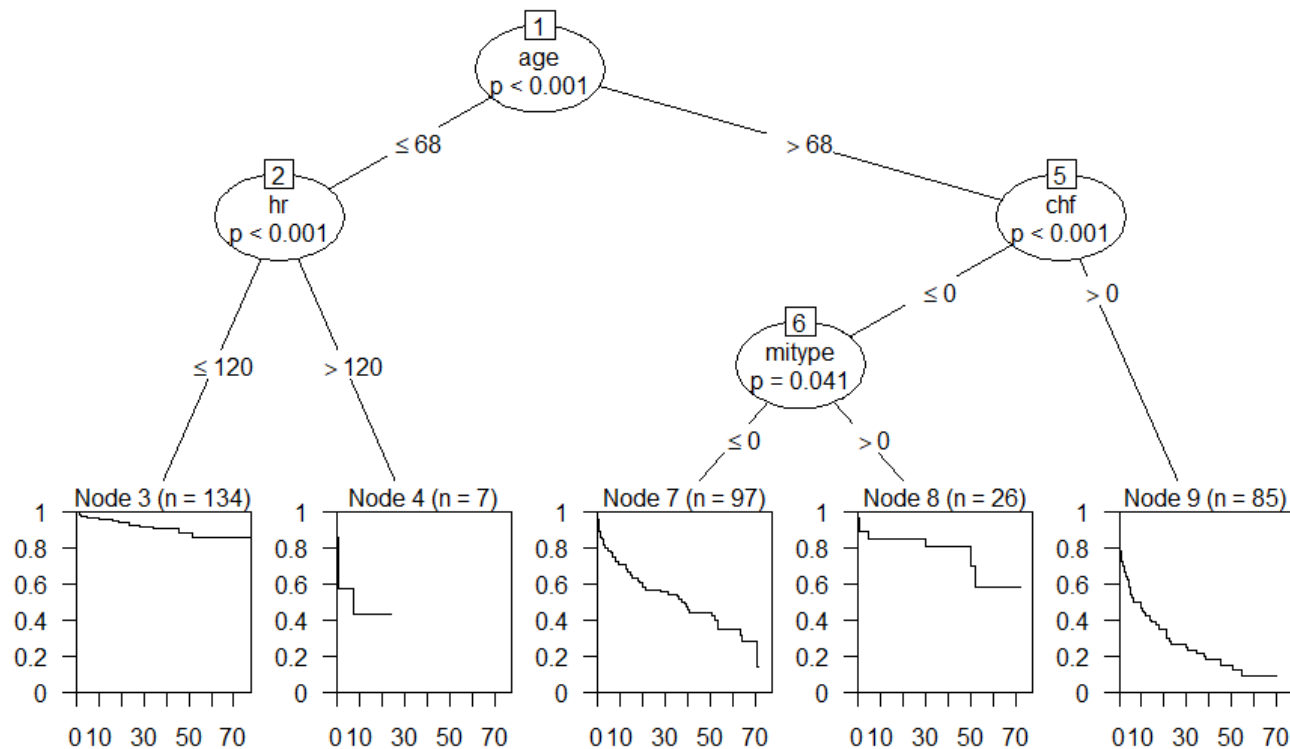
```
> tree_fit

Fitted party:
[1] root
|   [2] age <= 68
|   |   [3] hr <= 120: 77.470 (n = 134)
|   |   [4] hr > 120: 7.425 (n = 7)
|   [5] age > 68
|   |   [6] chf <= 0
|   |   |   [7] mitype <= 0: 38.571 (n = 97)
|   |   |   [8] mitype > 0: Inf (n = 26)
|   |   [9] chf > 0: 6.571 (n = 85)

Number of inner nodes:    4
Number of terminal nodes: 5
> ## Modelo final "tree"
> out_tree <- extract_fit_engine(tree_fit)
> class(out_tree)
[1] "constparty" "party"
> plot(out_tree)
```

- Se puede ver el **árbol de supervivencia**, con las **predicciones** para cada uno de los nodos terminales
- Para poder sacar un **gráfico del árbol**, usamos la función ***plot()*** del paquete ***party***, y para poder usar esa función, tenemos que extraer el objeto de este paquete, con la función ***extract_fit_engine()***

Árboles de Supervivencia con *partykit*



Medianas ---- >

77.5

7.4

38.6

Inf

6.6

- Las **predicciones** que se reportan son las **medianas** de las **curvas KM**, pero hay 2 valores **difíciles de interpretar**
- La predicción en el nodo 8 es “Infinite”, ya que no se puede calcular la mediana. La del nodo 3 es 77.47 debido a que la observación con el tiempo mayor es un evento, pero la supervivencia es mejor que la del nodo 8, ya que la curva está por encima

Árboles de Supervivencia con *tidymodels*

```
> ## Predicciones en Testing
> pred_tree_time <- predict(tree_fit, xx_test, type = "time")
> pred_tree_df <- bind_cols(xx_test %>% select(surv_var), pred_tree_time )
> pred_tree_df %>% count(.pred_time)
# A tibble: 5 × 2
  .pred_time      n
    <dbl> <int>
1     6.57     29
2     7.43      6
3    38.6     33
4    77.5     67
5     Inf     16
> ## c-index
> concordance_survival(pred_tree_df, truth = surv_var, estimate = .pred_time )
# A tibble: 1 × 3
  .metric          .estimator .estimate
  <chr>          <chr>      <dbl>
1 concordance_survival standard    0.639
```

- Se obtienen las predicciones, en la muestra de testing, del “tiempo hasta que se produce el evento” (*time*), y se comprueba que *tidymodels* proporciona los mismos valores que el árbol del objeto de *partykit*
- El **c-index** en la muestra de **testing** es **0.639**

Árbol de Supervivencia. Predicciones Supervivencia

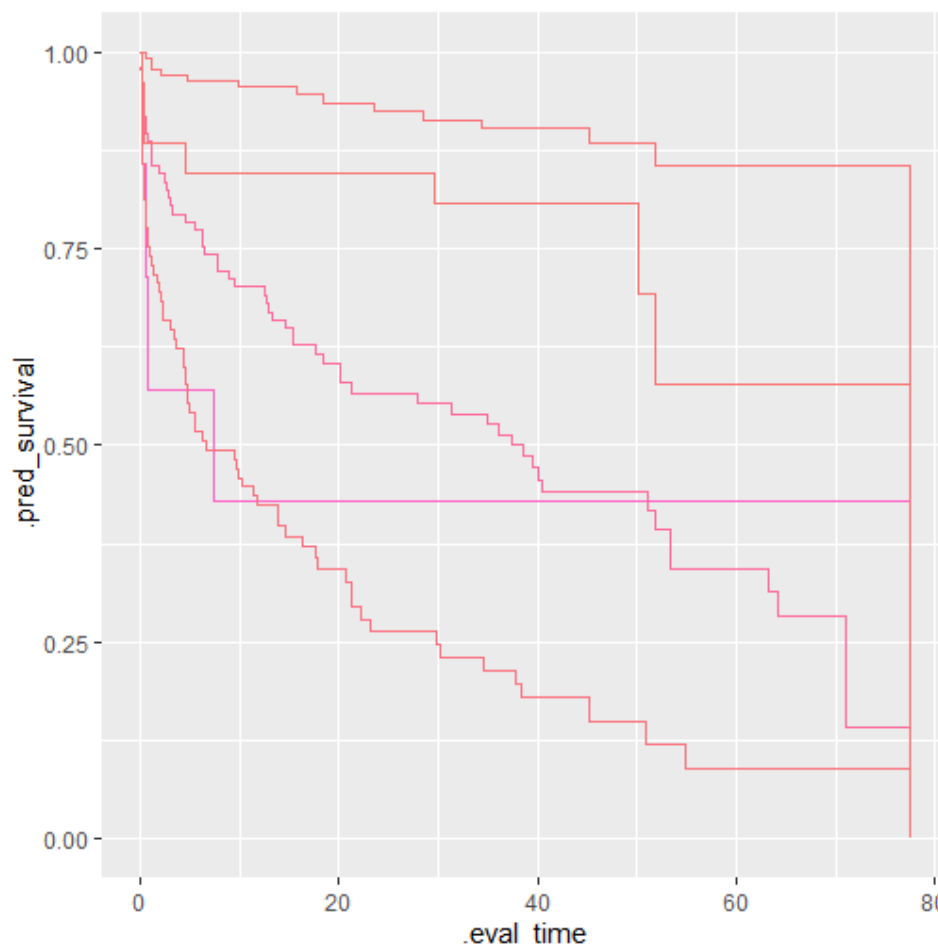
```
> ## Predicciones en todos los tiempos de supervivencia, donde se han producido eventos
> pred_tree_time_df_all <- augment( tree_fit, xx_test, eval_time = all_time_survival)
>
> ## Dataframe, con cada tiempo de cada observación en una fila, para el KM plot
> pred_tree_all_survival <- pred_tree_time_df_all %>%
+   mutate(id = factor(1:151)) %>%
+   unnest(cols = .pred)
```

- Se calculan, en la muestra de testing, las **predicciones del tiempo de supervivencia** en **todos los tiempos** de supervivencia donde se **han producido eventos** (almacenado en *all_time_survival*), con el modelo final del **árbol de supervivencia**
- Se usa la función ***augment()*** que añade a un *dataframe* todas las predicciones, la de “*time*” y las de la función de supervivencia, en todos los tiempos especificados
- Se prepara un *dataframe* para poder dibujar las **KM de todas las predicciones** en testing

Árbol de Supervivencia. Predicciones Supervivencia

```
> ## KM Plot
> pred_tree_all_survival %>%
+   ggplot(aes(x = .eval_time, y = .pred_survival, col = id)) +
+   geom_step() +
+   theme(legend.position = "none")
```

- En esta gráfica se han representado las **151 predicciones** de testing
- Había **5 nodos terminales**, y por eso muchas KM son iguales
- La predicción de la curva, basada en medianas, de arriba era **77.5** (la caída brusca) y de la segunda era **“Inf”**, pero la curva está por debajo, y debería tener **peor supervivencia**
- Se observa que **no siguen el principio de riesgos proporcionales**. De hecho, algunas curvas se cruzan



Random Forest

- **Random Forest (RF)** es una técnica de machine learning que consiste en la **construcción de un 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**
- **Random Survival Forest (RSF)** es la extensión de RF para datos de supervivencia, usando **árboles 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**

Random Forest

- Random Forest tiene varios **parámetros de tuning**, que hay que optimizar
 - el número de árboles (***trees***)
 - el número de predictores que son evaluados en cada división (***mtry***)
 - el número mínimo de observaciones en los nodos terminales (***min_n***)
- La **predicción de Random Survival Forest** para cada observación es un **estimador Kaplan-Meier**, calculado como la **media de todas las curvas KM** predichas en los nodos terminales, donde ha sido clasificada la observación en cada uno de **los árboles**
- **No asume el principio de riesgos proporcionales**, ya que está basado en árboles de supervivencia, que tampoco lo asumen

Random Forest con *tidymodels*

```
> ## 1.- Se crea la receta con los pasos del pre-procesamiento
> obj_rec_rf <-
+   recipe( surv_var ~ . , data=xx_train )
>
> ## 2.- Especificaciones del modelos: random forest con datos censurados
> rf_spec <-
+   rand_forest(trees = 200, mtry = tune(), min_n = tune()) %>%
+   set_engine("partykit") %>%
+   set_mode("censored regression")
>
> ## Parámetros del random forest
> trees()
# Trees (quantitative)
Range: [1, 2000]
> min_n()
Minimal Node Size (quantitative)
Range: [2, 40]
```

- Se indica el **recipe** con la fórmula solamente, ya que los árboles en los que está basado Random Forest no necesitan que se estandaricen las variables
- Se usa la función **rand_forest()** especificando que construya los árboles con el paquete “**partykit**”
- Se fija el número de árboles en 200 (**trees**) , y se optimiza el número de variables a evaluar en cada nodo (**mtry**) y el número mínimo de observaciones en los nodos terminales (**min_n**)

Random Forest con *tidymodels*

```
> mtry()
# Randomly Selected Predictors (quantitative)
Range: [1, ?]
> extract_parameter_set_dials(rf_spec)
  identifier type      object
    mtry    mtry nparam[?]
  min_n min_n nparam[+]

Model parameters needing finalization:
  # Randomly Selected Predictors ('mtry')

> extract_parameter_set_dials(rf_spec) %>%
+   finalize(xx_train)
  identifier type      object
    mtry    mtry nparam[+]
  min_n min_n nparam[+]
```

- Con ***extract_parameter_set_dials()*** se puede extraer la información de los parámetros en la especificación del modelo de Random Forest
- El ***mtry()***, número de variables a evaluar en cada nodo, depende del tamaño muestral, y por eso no tiene un valor prefijado por defecto. Se debe “finalizar” a partir de los datos concretos del *dataframe*, usando la función ***finalize()***

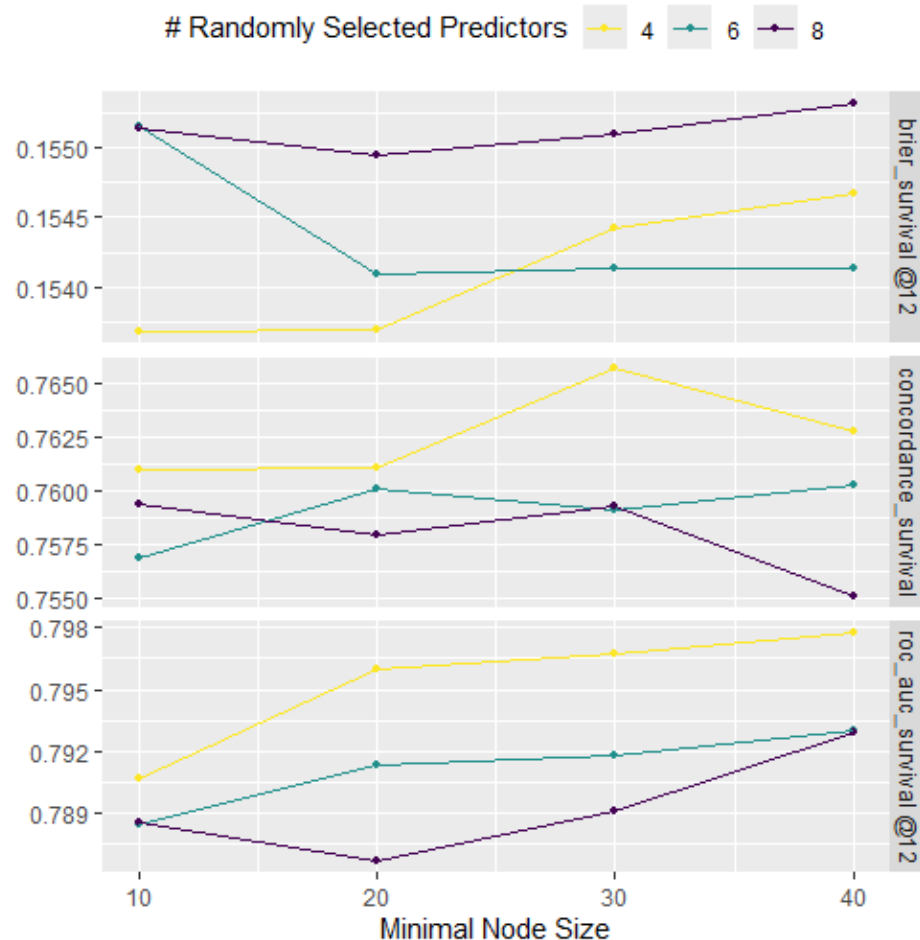
Random Forest con *tidymodels*

```
> ## 3.- Se crea el workflow
> wflow_rf <- workflow() %>%
+   add_model(rf_spec) %>%
+   add_recipe(obj_rec_rf)
>
> ## 4.- Se crea un grid con los parámetros a explorar
> rf_grid <- crossing( mtry = c(4, 6, 8), min_n = c(10, 20, 30, 40))
>
> ## Se ejecuta la optimización de parámetros
> tune_result_rf <- wflow_rf %>%
+   tune_grid( resamples = cv_split,
+             grid = rf_grid,
+             metrics = metric_set(concordance_survival, brier_survival,
+                                 roc_auc_survival),
+             eval_time = 12 )
```

- Se crea el **workflow** con el *recipe* y las especificaciones del modelo
- Se crea un “grid” con los valores concretos que se quieren explorar de **mtry** y de **min_n** usando la función **crossing()**
- Se ejecuta la **optimización de parámetros** con **tune_grid()**
- Se indican las mismas **medidas de capacidad predictiva** a evaluar (**metrics=**), que son el c-index, el Brier score y el AUC. Estas dos últimas medidas evaluadas a 1 año (**eval_time=12**)

Random Forest. Optimización de parámetros

```
> ## Plot  
> autoplot(tune_result_rf) +  
+   scale_color_viridis_d(direction = -1) +  
+   theme(legend.position = "top")
```



Random Forest. Modelo final

```
> ## Los mejores modelos, con máximo c-index
> show_best(tune_result_rf, metric="concordance_survival")
# A tibble: 5 × 9
  mtry min_n .metric .estimator .eval_time mean n std_err
<dbl> <dbl> <chr>      <chr>      <dbl> <dbl> <int> <dbl>
1     4    30 concordance_survival standard    NA  0.766    20  0.0122
2     4    40 concordance_survival standard    NA  0.763    20  0.0108
3     4    20 concordance_survival standard    NA  0.761    20  0.0122
4     4    10 concordance_survival standard    NA  0.761    20  0.0122
5     6    40 concordance_survival standard    NA  0.760    20  0.0119
>
> ## Se crea al workflow final
> final_wflow_rf <-
+   wflow_rf %>%
+   finalize_workflow( select_best(tune_result_rf, metric="concordance_survival") )
>
> ## Se crea al modelo final
> rf_fit <-
+   final_wflow_rf %>%
+   fit(xx_train)
```

- El **mejor modelo** es el que tiene 4 como número de variables que se evalúan en cada nodo, y un tamaño mínimo en los nodos de 30. Este modelo tiene un **c-index** de **0.766**, obtenido en el remuestreo
- Se finaliza el **workflow** final, y se ajusta el **modelo** con los **parámetros óptimos** en el *dataframe* de training

Random Forest. Capacidad Predictiva

```
> ## Predicciones en Testing
> pred_rf_time <- predict(rf_fit, xx_test, type = "time")
> pred_rf_df <- bind_cols(xx_test %>% select(surv_var), pred_rf_time)
> pred_rf_df %>% print(n=4)
# A tibble: 151 × 2
  surv_var .pred_time
  <Surv>    <dbl>
1 71.35934+    77.5
2 49.14990    77.5
3 71.45791+    77.5
4 77.30595    54.9
# i 147 more rows
>
> ## c-index
> concordance_survival(pred_rf_df, truth = surv_var, estimate = .pred_time )
# A tibble: 1 × 3
  .metric          .estimator .estimate
  <chr>            <chr>      <dbl>
1 concordance_survival standard    0.739
```

- Se obtienen las predicciones, en la muestra de testing, del “tiempo hasta que se produce el evento” (*time*)
- El **c-index** en la muestra de **testing** es **0.739**, un valor menor que el que se obtuvo con el modelo de regresión penalizada, que fue 0.750, lo que no suele ser habitual, aunque este es un análisis con pocas variables, muchas de ellas binarias, que no necesitan un modelado no lineal

Random Forest. Brier Score

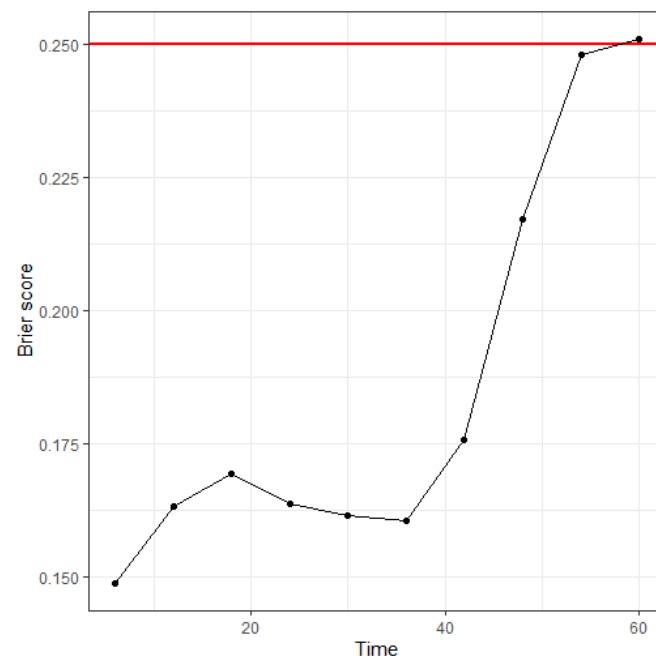
```
> ## Tiempos de evaluación a explorar, hasta 5 años, con saltos de 6 meses
> time_points <- seq( 6, 60, by=6 )
> ## Predicciones en Testing
> pred_rf_time_df <- augment( rf_fit, xx_test, eval_time = time_points)
> ## Dynamic Brier Score
> brier_scores <-
+   brier_survival( pred_rf_time_df, truth = surv_var, .pred )
> brier_scores
# A tibble: 10 × 4
  .metric      .estimator .eval_time .estimate
  <chr>        <chr>      <dbl>    <dbl>
1 brier_survival standard      6      0.149
2 brier_survival standard     12      0.163
3 brier_survival standard     18      0.169
4 brier_survival standard     24      0.164
5 brier_survival standard     30      0.161
6 brier_survival standard     36      0.160
7 brier_survival standard     42      0.176
8 brier_survival standard     48      0.217
9 brier_survival standard     54      0.248
10 brier_survival standard     60      0.251
```

- Para obtener las **medidas dinámicas de capacidad predictiva**, se fijan los tiempos de evaluación, hasta 5 años, con saltos de 6 meses (10 tiempos)
- Con la función ***brier_survival()*** se obtienen los **Brier Score** en los tiempos de evaluación
- En los primeros tiempos, entre 6 y 36 meses, se obtiene los mejores resultados

Random Forest. Brier Score

```
> brier_scores %>%
+   ggplot(aes(.eval_time, .estimate)) +
+   geom_hline(yintercept = 0.25, col="red", lwd = 1) +
+   geom_line() +
+   geom_point() +
+   labs(x = "Time", y="Brier score") +
+   theme_bw()
> ## Integrated Brier Scores
> brier_survival_integrated(pred_rf_time_df, truth = surv_var, .pred )
# A tibble: 1 × 3
  .metric          .estimator .estimate
  <chr>            <chr>      <dbl>
1 brier_survival_integrated standard    0.166
```

- Se ve claramente que el **Brier score** aumenta con el tiempo, es decir, el modelo pierde capacidad predictiva para predecir a largo plazo
- El **Brier score integrado** (área de la curva), es de **0.166**, que es bastante moderado
- Los resultados en **Random Forest** son muy **parecidos, ligeramente mejores**, a los de la regresión penalizada (Brier score integrado era 0.174)



Random Forest. AUC de la curva ROC

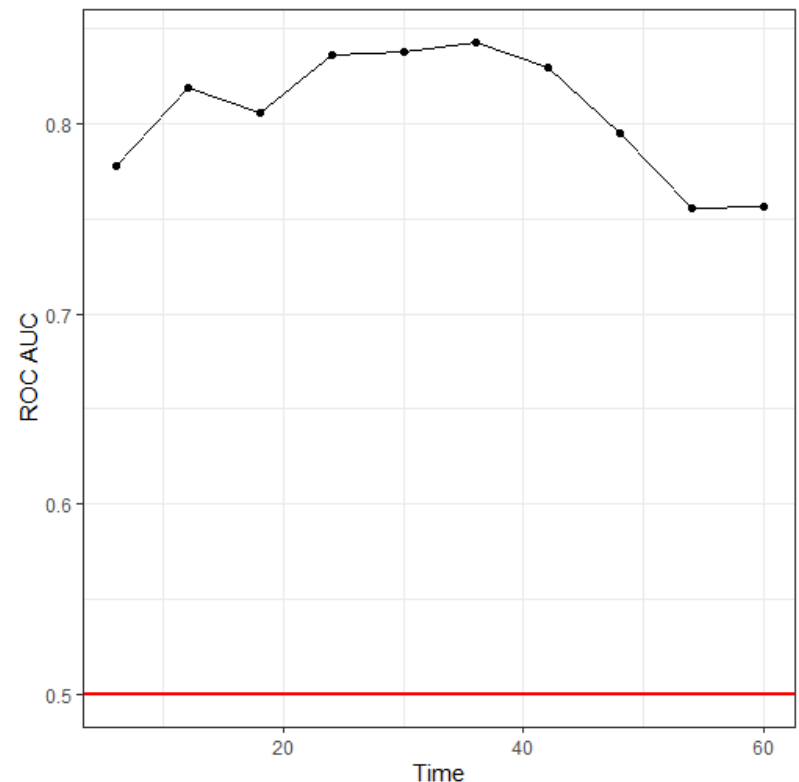
```
> ## Dynamic ROC curves
> roc_scores <-
+   roc_auc_survival( pred_rf_time_df, truth = surv_var, .pred )
> roc_scores
# A tibble: 10 × 4
  .metric      .estimator .eval_time .estimate
  <chr>        <chr>      <dbl>    <dbl>
1 roc_auc_survival standard      6    0.778
2 roc_auc_survival standard     12    0.819
3 roc_auc_survival standard     18    0.806
4 roc_auc_survival standard     24    0.836
5 roc_auc_survival standard     30    0.838
6 roc_auc_survival standard     36    0.843
7 roc_auc_survival standard     42    0.829
8 roc_auc_survival standard     48    0.795
9 roc_auc_survival standard     54    0.755
10 roc_auc_survival standard     60    0.756
```

- Con la función ***roc_auc_survival()*** se obtienen los **AUCs** de la curva ROC en los tiempos de evaluación
- En los primeros, y sobre todo, en los tiempos intermedios, se obtienen las mejores predicciones
- Los resultados del AUC de **Random Forest** son **ligeramente mejores** que los obtenidos en regresión penalizada (el AUC máximo era 0.824)

Random Forest. AUC de la curva ROC

```
> ## Plot
> roc_scores %>%
+   ggplot(aes(.eval_time, .estimate)) +
+   geom_hline(yintercept = 0.5, col="red", lwd = 1) +
+   geom_line() +
+   geom_point() +
+   labs(x = "Time", y="ROC AUC") +
+   theme_bw()
```

- El **AUC** es mejor en los tiempos intermedios y en los más cortos, como ya indicaba el Brier Score, pero los valores del AUC fluctúan menos



Random Forest. Predicciones Supervivencia

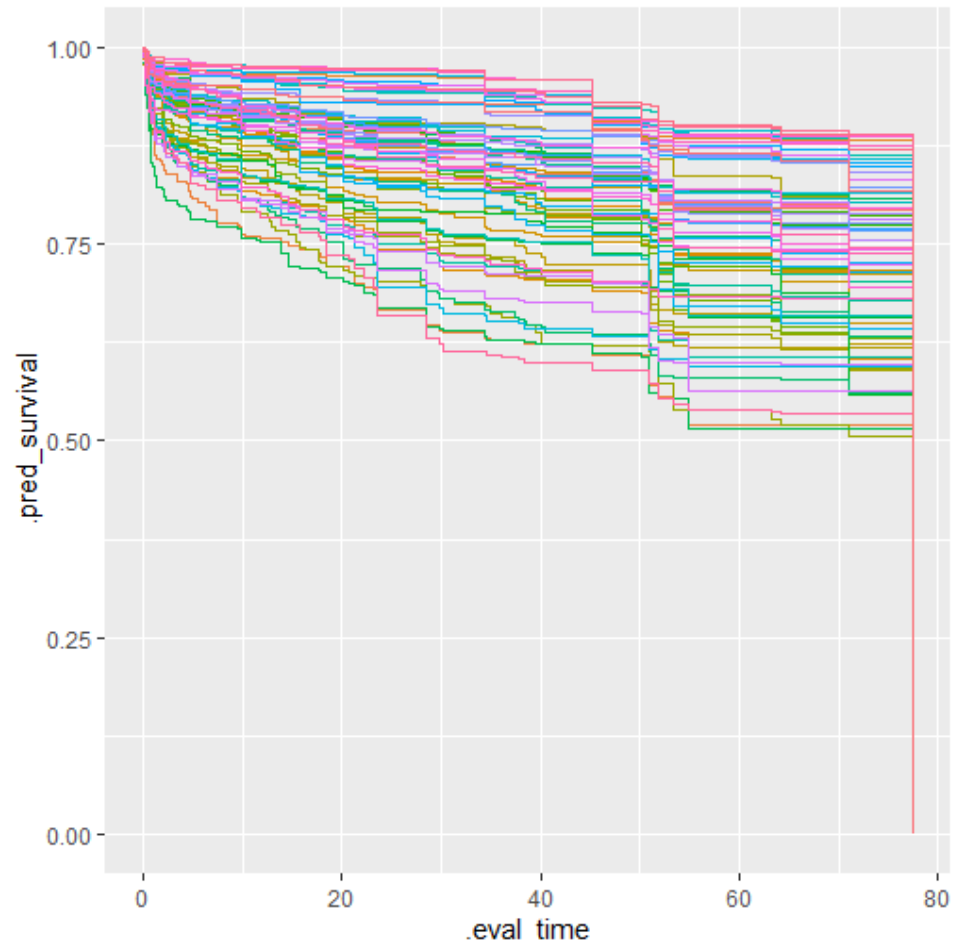
```
> ## Predicciones en todos los tiempos de supervivencia, donde se han producido eventos
> pred_rf_time_df_all <- augment( rf_fit, xx_test, eval_time = all_time_survival)
>
> ## Dataframe, con cada tiempo de cada observación en una fila, para el KM plot
> pred_rf_all_survival <- pred_rf_time_df_all %>%
+   mutate(id = factor(1:151)) %>%
+   unnest(cols = .pred)
>
> pred_rf_time_df_all %>% count(.pred_time) %>% tail
  .pred_time      n
2         63.3      5
3         64.2      2
4         71.0      2
5         77.5     70
```

- Se calculan, en la muestra de testing, las **predicciones del tiempo de supervivencia en todos los tiempos** de supervivencia donde **se han producido eventos** (almacenado en *all_time_survival*), con el modelo final de **Random Forest**
- Se usa la función ***augment()*** que añade a un *dataframe* todas las predicciones, la de “*time*” y las de la función de supervivencia, en todos los tiempos especificados
- Se prepara un *dataframe* para poder dibujar las **KM de todas las predicciones** en testing
- Se observa que hay **70 observaciones** de testing que tienen **77.5 como predicción**

Random Forest. Predicciones Supervivencia

```
> ## KM Plot de las que tienen mejor supervivencia
> pred_rf_all_survival %>% filter ( .pred_time > 77 ) %>%
+   ggplot(aes(x = .eval_time, y = .pred_survival, col = id)) +
+   geom_step() +
+   theme(legend.position = "none")
```

- En esta gráfica se han representado solo las 70 observaciones de testing cuya **predicción es igual a 77.5**
- Si no hubiera la caída brusca, todas hubieran sido etiquetas como “Inf”, ya que no se puede calcular la mediana de estas curvas
- Sin embargo, **la supervivencia** en estas curvas es **muy diferente**, y esto se podría corregir al **sumar todas las supervivencias** descritas en las curvas, lo que asigna **una predicción distinta** para cada una



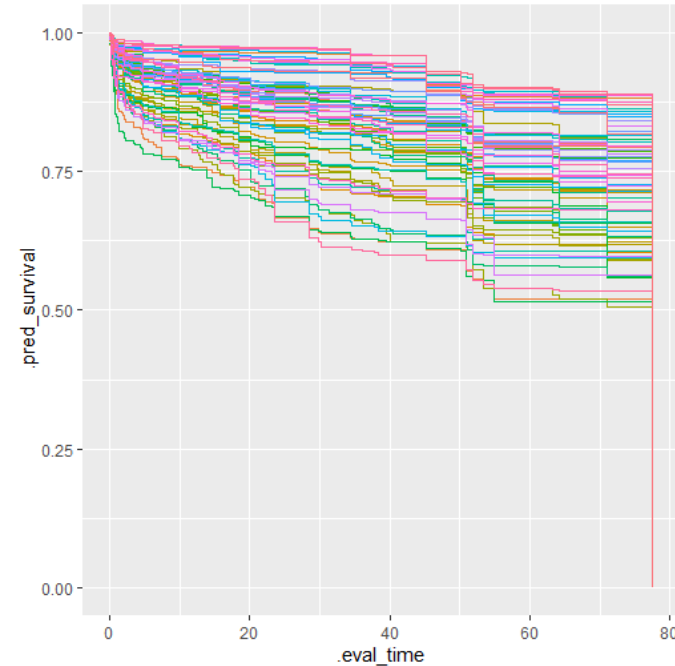
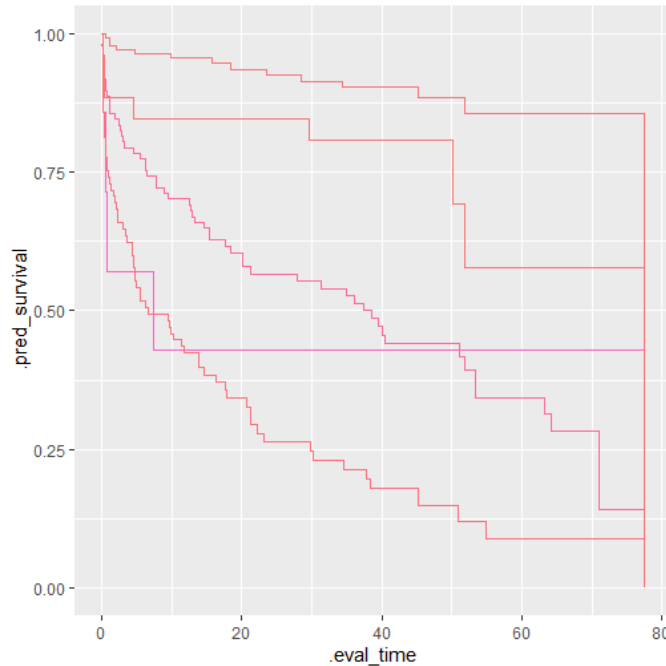
Predicciones con Datos Censurados. Comentarios

- La **predicción básica** que hacen todos los métodos de **análisis de datos censurados** es la **función de supervivencia predicha**, en todo el rango de tiempo, de cada observación
- La **medida de capacidad predictiva** más aceptada dentro del análisis de datos censurados es el **c-index**, pero para poder calcularlo hace falta un **único valor de la predicción** para cada observación
- En los **modelos de regresión**, esa predicción única por observación, está claramente determinada por el **predictor lineal**
- En otros modelos (árboles, random forest) se ha propuesto como predicción única la **mediana de la curva de supervivencia** predicha
 - En algunos conjuntos de datos, en las observaciones con mejor supervivencia **no se puede calcular la mediana**
 - Todas estas observaciones se unen en un **grupo genérico** (“Infinite”), que está incluyendo observaciones con muy **diferente supervivencia**

Predicciones con Datos Censurados. Comentarios

- Una alternativa a la mediana es tomar otro **punto de la curva**, el tiempo de supervivencia en un **tiempo concreto**
 - Si se cambia el tiempo, **el resultado será distinto**, pero con pequeñas diferencias
 - Se aconseja tomar tiempos en el **medio del rango del tiempo**
- Otra alternativa es resumir toda la información de la curva, **sumando la función de supervivencia** en todos los tiempos
 - Esta propuesta está basada en lo que hace el paquete “**randomForestSRC**” para calcular el **c-index**, aunque allí se hace con la función de **riesgo acumulado**, que es una transformación monótona del $S(t)$. **$H(t) = -\ln S(t)$**
 - **No tiene interpretación**, pero el predictor lineal tampoco
 - El problema de esta medida es la **falta de referencias** metodológicas para su uso
- Todas estas alternativas obtienen los **mismos resultados** en los modelos de regresión de **riesgos proporcionales**, cuando se usa el predictor lineal

Predicciones con Datos Censurados. Comentarios



- En el **Árbol**, al sumar las supervivencias, la curva de arriba tendrá ahora mejor supervivencia que la segunda
- En **Random Forest**, las curvas que tenían 77.5 como predicción, tienen una **supervivencia muy diferente**, y esto se corregiría al **sumar todas las supervivencias** descritas en las curvas, o al tomar la **supervivencia en un tiempo concreto**, que asignaría **una predicción distinta** a cada una de ellas

Árbol de Supervivencia. Predicciones Alternativas

```
> ## Se añade la suma de todos los tiempos de supervivencia
> pred_tree_time_df_all <- pred_tree_time_df_all %>%
+   rowwise() %>%
+   mutate(sum_survival = sum(.pred %>% select(.pred_survival))) %>%
+   ungroup()
> ## c-index con la predicción de "time" y con la suma de supervivencia
> concordance_survival(pred_tree_time_df_all, truth = surv_var, estimate = .pred_time )
1 concordance_survival standard      0.639
> concordance_survival(pred_tree_time_df_all, truth = surv_var, estimate = sum_survival )
1 concordance_survival standard      0.679
>
> pred_tree_time_df_all %>% select(.pred_time) %>% table
.pred_time
6.57084188911704  7.42505133470226  38.570841889117  77.4702258726899      Inf
          29              6              33              67              16
> pred_tree_time_df_all %>% select(sum_survival) %>% table
sum_survival
54.5309537132775  62.1428571428571  79.9440263495337  98.1538461538462 110.797121687596
          29              6              33              16              67
```

- Se calcula la **suma de todos estos tiempos de supervivencia predichos**, y se añade como una columna que se llama ***sum_survival***
- El **c-index** calculado con la suma de las supervivencias es **0.679**, mayor que el calculado con la predicción de tiempo (“time”) que era 0.639
- Al considerar la variable con la suma de las supervivencias, las observaciones etiquetadas con “Inf” tienen ahora mejor supervivencia que el otro grupo

Random Forest. Predicciones Alternativas

```
> ## Se añade la suma de todos los tiempos de supervivencia
> pred_rf_time_df_all <- pred_rf_time_df_all %>%
+   rowwise() %>%
+   mutate(sum_survival = sum(.pred %>% select(.pred_survival))) %>%
+   ungroup()
> ## c-index con la predicción de "time" y con la suma de supervivencia
> concordance_survival(pred_rf_time_df_all, truth = surv_var, estimate = .pred_time )
1 concordance_survival standard      0.739
> concordance_survival(pred_rf_time_df_all, truth = surv_var, estimate = sum_survival )
1 concordance_survival standard      0.760
> pred_rf_time_df_all %>% select(.pred_time)%>% round(1) %>% table %>% tail
.pred_time
53.5 54.9 63.3 64.2  71 77.5
   2   4   5   2   2  70
> pred_rf_time_df_all %>% select(sum_survival)%>% round(1) %>% table %>% tail
sum_survival
112.5 112.6 113.2 113.4 113.5  114
   2   1   1   1   1   1
```

- Se calcula la **suma de todos estos tiempos de supervivencia predichos**, y se añade como una columna que se llama ***sum_survival***
- El **c-index** calculado con la suma de las supervivencias es **0.760**, mayor que el calculado con la predicción de tiempo ("time") que era 0.739. En **Random Forest** se obtiene ahora **un resultado mejor** que con la regresión penalizada, que era 0.750
- Los 70 que estaban etiquetados como "77.5" tienen distintos valores en la suma de las supervivencias

Random Forest. Predicciones Alternativas

```
> ## c-index en tiempos concretos
> pred_rf_surv <- predict(rf_fit, xx_test, type = "survival", eval_time = c(12,60) )
>
> pred_rf_surv_12 <- pred_rf_surv %>%
+   tidyr::unnest(col = .pred) %>% filter( .eval_time == 12 )
> pred_rf_12_df <- bind_cols(xx_test %>% select(surv_var), pred_rf_surv_12 )
> concordance_survival(pred_rf_12_df, truth = surv_var, estimate = .pred_survival )
# A tibble: 1 × 3
  .metric          .estimator .estimate
  <chr>            <chr>      <dbl>
1 concordance_survival standard    0.761
>
> pred_rf_surv_60 <- pred_rf_surv %>%
+   tidyr::unnest(col = .pred) %>% filter( .eval_time == 60 )
> pred_rf_60_df <- bind_cols(xx_test %>% select(surv_var), pred_rf_surv_60 )
> concordance_survival(pred_rf_60_df, truth = surv_var, estimate = .pred_survival )
# A tibble: 1 × 3
  .metric          .estimator .estimate
  <chr>            <chr>      <dbl>
1 concordance_survival standard    0.758
```

- Si se considera como **predicción numérica única**, para cada observación, la **supervivencia a 12 o a 60 meses**, se obtienen c-index de **0.761** y **0.758**, muy parecidos a 0.760, obtenido con la suma
- En cada punto del tiempo, el cálculo del c-index puede **cambiar un poco**, porque **no se sigue el principio de riesgos proporcionales**, y algunas curvas se pueden cruzar

Regresión penalizada. Predicciones Alternativas

```
> ## Se añade la suma de todos los tiempos de supervivencia
> pred_enet_time_df_all <- pred_enet_time_df_all %>%
+   rowwise() %>%
+   mutate(sum_survival = sum(.pred %>% select(.pred_survival))) %>%
+   ungroup()
>
> ## c-index con la predicción de "time" y con la suma de supervivencia
> concordance_survival(pred_enet_df, truth = surv_var, estimate = .pred_linear_pred )
# A tibble: 1 × 3
  .metric          .estimator .estimate
  <chr>            <chr>      <dbl>
1 concordance_survival standard    0.750
> concordance_survival(pred_enet_time_df_all, truth = surv_var, estimate = sum_survival
)
# A tibble: 1 × 3
  .metric          .estimator .estimate
  <chr>            <chr>      <dbl>
1 concordance_survival standard    0.750
```

- Se calcula **la suma de todos estos tiempos de supervivencia predichos**, y se añade como una columna que se llama ***sum_survival***
- El **c-index** calculado con la suma de las supervivencias es **0.750**, exactamente igual que el basado en **el predictor lineal**

Regresión penalizada. Predicciones Alternativas

```
> ## c-index en tiempos concretos
> pred_enet_surv <- predict(enet_fit, xx_test, type = "survival", eval_time = c(12,60))
>
> pred_enet_surv_12 <- pred_enet_surv %>%
+   tidyr::unnest(col = .pred) %>% filter( .eval_time == 12 )
> pred_enet_12_df <- bind_cols(xx_test %>% select(surv_var), pred_enet_surv_12 )
> concordance_survival(pred_enet_12_df, truth = surv_var, estimate = .pred_survival )
# A tibble: 1 × 3
  .metric      .estimator .estimate
  <chr>        <chr>      <dbl>
1 concordance_survival standard    0.750
>
> pred_enet_surv_60 <- pred_enet_surv %>%
+   tidyr::unnest(col = .pred) %>% filter( .eval_time == 60 )
> pred_enet_60_df <- bind_cols(xx_test %>% select(surv_var), pred_enet_surv_60 )
> concordance_survival(pred_enet_60_df, truth = surv_var, estimate = .pred_survival )
# A tibble: 1 × 3
  .metric      .estimator .estimate
  <chr>        <chr>      <dbl>
1 concordance_survival standard    0.750
```

- Si se considera como **predicción numérica única**, para cada observación, la **supervivencia a 12 o a 60 meses**, se obtiene siempre el mismo c-index de **0.750**
- Como la regresión penalizada es un modelo de Cox de **riesgos proporcionales**, las **curvas no se cruzan**, y los resultados son siempre **los mismos**

Conclusiones. Comentarios

- El paquete ***censored*** es un paquete todavía en **versión 0**, que recoge ya todas las **funcionalidades importantes** de ***tidymodels*** para la construcción de modelos (remuestreo, optimización de parámetros, tratamiento de los resultados, ...)
- Reúne los **modelos predictivos** de análisis de datos **más importantes** de **análisis de datos censurados**
 - Se echa en falta algún “*engine*”, como el paquete “***randomForestSRC***”
- El punto más delicado es el **cálculo del c-index** basado en la predicción del **tiempo** hasta que se produce el evento
 - Es un problema de los paquetes de R, no tanto de ***tidymodels***
 - El **significado** de esta variable **es difícil**, y además cambia entre modelos
 - Hay que buscar **alternativas** para una **predicción** con un **valor único** por observación en los modelos que no son de regresión
 - En algunos casos, puede ser preferible la **optimización de los parámetros** basada en el **AUC** en un tiempo de evaluación concreto, en lugar del c-index

Curso de Formación Continua - UAM - 6ª Edición - 2025

Estadística Aplicada con



| Módulos | Fechas 2025 |
|---|----------------------|
| 1. Introducción a R | 24, 25 y 26 Marzo |
| 2. Visualización Interactiva de Datos con el paquete Shiny | 8 y 9 Abril |
| 3. Métodos de Regresión y Análisis Multivariante con R | 22, 23 y 24 Abril |
| 4. Métodos de Regresión Avanzados para la Investigación en Ciencias Naturales con R | 6, 7 y 8 Mayo |
| → 5. Estadística Aplicada a la Investigación Biomédica con R | 19, 20 y 21 Mayo |
| 6. Modelos Mixtos con R | 27, 28 y 29 Mayo |
| → 7. Machine Learning con R y <i>tidymodels</i> | 2, 3, 4, 5 y 6 Junio |
| 8. Deep Learning con R | 9, 10, 11 Junio |
| 9. Análisis de Datos Funcionales con R | 17 y 18 Junio |
| 10. Análisis Estadístico de Redes con R | 23 y 24 Junio |

ANEXO 1

Regresión de Cox con *tidymodels*

Regresión de Cox con *survival*

```
> ## 2.1- Regresión de Cox con survival
> out_cox <- coxph( surv_var ~ . , data = xx_train )
> summary(out_cox)
```

| | coef | exp(coef) | se(coef) | z | Pr(> z) | |
|--------|-----------|-----------|----------|--------|----------|-----|
| age | 0.042085 | 1.042983 | 0.008186 | 5.141 | 2.73e-07 | *** |
| gender | -0.159627 | 0.852461 | 0.174058 | -0.917 | 0.359094 | |
| hr | 0.012565 | 1.012645 | 0.004001 | 3.141 | 0.001686 | ** |
| sysbp | -0.001553 | 0.998448 | 0.003580 | -0.434 | 0.664424 | |
| diasbp | -0.009448 | 0.990597 | 0.006151 | -1.536 | 0.124532 | |
| bmi | -0.044956 | 0.956039 | 0.019362 | -2.322 | 0.020239 | * |
| cvd | 0.012062 | 1.012135 | 0.219932 | 0.055 | 0.956264 | |
| afb | -0.072251 | 0.930297 | 0.223184 | -0.324 | 0.746144 | |
| sho | 1.741522 | 5.706024 | 0.334088 | 5.213 | 1.86e-07 | *** |
| chf | 0.708297 | 2.030531 | 0.184894 | 3.831 | 0.000128 | *** |
| av3 | 1.196946 | 3.309993 | 0.494071 | 2.423 | 0.015409 | * |
| miord | -0.116590 | 0.889950 | 0.179932 | -0.648 | 0.517006 | |
| mitype | -0.703321 | 0.494939 | 0.263718 | -2.667 | 0.007655 | ** |

Concordance= 0.798 (se = 0.018)
Likelihood ratio test= 171.5 on 13 df, p=<2e-16
Wald test = 157.6 on 13 df, p=<2e-16
Score (logrank) test = 189 on 13 df, p=<2e-16

- Se ajusta, en la muestra de training, un modelo de **regresión de Cox** de riesgos proporcionales con la función ***coxph()*** del paquete ***survival***

Regresión de Cox con *tidymodels*

```
> ## 2.2.- Regresión de Cox con tidymodels
> ## Se crean las especificaciones del modelo
> ph_spec <-
+   proportional_hazards() %>%
+   set_engine("survival") %>%
+   set_mode("censored regression")
> ph_spec
Proportional Hazards Model Specification (censored regression)

Computational engine: survival

> ## Se ajusta el model
> ph_fit <- ph_spec %>%
+   fit( surv_var ~ ., data = xx_train)
```

- Con la función ***proportional_hazards()*** se especifica el modelo que vamos a ajustar
- ***set_engine()*** indica que se use la función del paquete ***survival***, que es la ***coxph()***
- ***set_mode()*** indica que se están analizando datos censurados (en este caso no sería necesario)
- La función ***fit()*** es la que ajusta el modelo, con las especificaciones anteriores, y donde se indica la fórmula y el *dataframe* de training

Regresión de Cox con *tidymodels*

```
> ph_fit
parsnip model object

Call:
survival::coxph(formula = surv_var ~ ., data = data, model = TRUE,
  x = TRUE)

      coef exp(coef) se(coef)      z      p
age      0.042085  1.042983  0.008186  5.141 2.73e-07
gender -0.159627  0.852461  0.174058 -0.917 0.359094
hr       0.012565  1.012645  0.004001  3.141 0.001686
sysbp   -0.001553  0.998448  0.003580 -0.434 0.664424
diasbp  -0.009448  0.990597  0.006151 -1.536 0.124532
bmi     -0.044956  0.956039  0.019362 -2.322 0.020239
cvd      0.012062  1.012135  0.219932  0.055 0.956264
afb     -0.072251  0.930297  0.223184 -0.324 0.746144
sho      1.741522  5.706024  0.334088  5.213 1.86e-07
chf      0.708297  2.030531  0.184894  3.831 0.000128
av3      1.196946  3.309993  0.494071  2.423 0.015409
miord   -0.116590  0.889950  0.179932 -0.648 0.517006
mitype  -0.703321  0.494939  0.263718 -2.667 0.007655

Likelihood ratio test=171.5  on 13 df, p=< 2.2e-16
n= 349, number of events= 150
```

- Se puede comprobar que el modelo es el mismo que se ajustó con ***coxph()***

Regresión de Cox con *tidymodels*

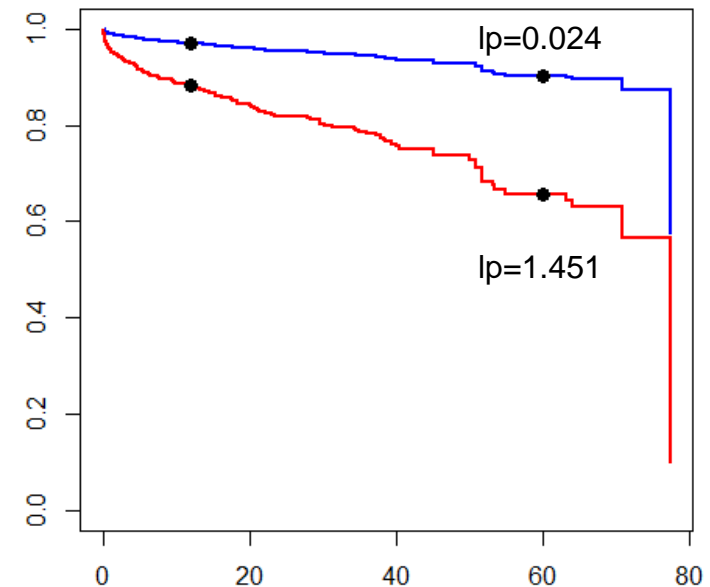
```
> tidy(ph_fit)
# A tibble: 13 × 5
  term      estimate std.error statistic    p.value
  <chr>      <dbl>      <dbl>      <dbl>    <dbl>
1 age        0.0421      0.00819      5.14 0.000000273
2 gender    -0.160        0.174     -0.917  0.359
3 hr         0.0126      0.00400      3.14  0.00169
4 sysbp     -0.00155      0.00358     -0.434  0.664
5 diasbp    -0.00945      0.00615     -1.54  0.125
6 bmi       -0.0450      0.0194     -2.32  0.0202
7 cvd        0.0121      0.220       0.0548 0.956
8 afb       -0.0723      0.223     -0.324  0.746
9 sho        1.74        0.334       5.21  0.000000186
10 chf       0.708        0.185       3.83  0.000128
11 av3        1.20        0.494       2.42  0.0154
12 miord     -0.117        0.180     -0.648  0.517
13 mitype    -0.703        0.264     -2.67  0.00765
```

- Se puede usar la función ***tidy()*** con este modelo como parámetro, que devuelve los coeficientes en un *dataframe* de tipo **tibble**, con nombres de columna “comprensibles” y “estandarizados”
- La función ***tidy()*** se puede aplicar a muchos modelos de regresión (lineal, logística, Cox ...) y siempre devuelve los resultados en este formato y nombres de columna

Predicciones del Modelo de Regresión de Cox

```
> ## Linear Predictor con survival
> pred_cox_lin_pred <- predict(out_cox, xx_test, type = "lp", reference = "zero" )
> head(pred_cox_lin_pred)
      1      2      3      4      5      6
0.59313268 0.42199362 1.18380327 1.04330042 0.02426664 1.45051481
>
> ## Función de supervivencia predicha para la Observacion 5 (buen pronóstico, azul)
> surv_obs_5 <- survfit( out_cox, newdata = xx_test %>% slice(5), se=F )
> plot( surv_obs_5, mark.time=F, conf.int=F, col="blue", lwd=2 )
> ## Función de supervivencia predicha para la Observacion 6 (mal pronóstico, rojo)
> surv_obs_6 <- survfit( out_cox, newdata = xx_test %>% slice(6), se=F )
> lines( surv_obs_6, mark.time=F, conf.int=F, col="red", lwd=2 )
```

- La función ***predict()*** del paquete ***survival*** permite obtener el **predictor lineal** (***type="lp"***) en la **muestra de testing** (***reference="zero"*** es para que no centre los predictores)
- Se obtienen **las curvas de supervivencias** predichas por el modelo, con la función ***survfit()***
- El predictor lineal de la observación 5 (azul) es menor que el de la 6 (rojo), lo que indica menos riesgo, y por tanto, mejor supervivencia, estando la curva por encima



Predicciones del Modelo de Regresión de Cox

```
> ## predict tipo "survival" de tidymodels
> pred_ph_surv <- predict(ph_fit, xx_test, type = "survival",
+                          eval_time=seq(6, 60, by=6))
> pred_ph_surv %>% print(n=5)
# A tibble: 151 × 1
  .pred
  <list>
1 <tibble [10 × 2]>
2 <tibble [10 × 2]>
3 <tibble [10 × 2]>
4 <tibble [10 × 2]>
5 <tibble [10 × 2]>
# i 146 more rows
> ## predict tipo "survival" en tiempos concretos (1 y 5 años)
> pred_ph_surv_12 <- pred_ph_surv %>%
+   tidyr::unnest(col = .pred) %>% filter( .eval_time == 12 )
> pred_ph_surv_60 <- pred_ph_surv %>%
+   tidyr::unnest(col = .pred) %>% filter( .eval_time == 60 )
```

- La función ***predict()*** del paquete ***tidymodels*** permite obtener, en la muestra de testing, la supervivencia predicha (***type="survival"***), en distintos momentos del tiempo: hasta 6 años, en saltos de 6 meses (***eval_time=***)
- Devuelve un ***tibble*** con todas las **supervivencias predichas**, en esos 10 tiempos fijados, y se permite extraer las supervivencias en los tiempos concretos (***unnest*** convierte listas en columnas)

Predicciones del Modelo de Regresión de Cox

```
> pred_ph_surv_12 %>% print(n=6)
# A tibble: 151 × 2
  .eval_time .pred_survival
    <dbl>         <dbl>
1      12      0.948
2      12      0.956
3      12      0.907
4      12      0.919
5      12      0.970
6      12      0.881
# i 145 more rows
> pred_ph_surv_60 %>% print(n=6)
# A tibble: 151 × 2
  .eval_time .pred_survival
    <dbl>         <dbl>
1      60      0.836
2      60      0.860
3      60      0.723
4      60      0.755
5      60      0.903
6      60      0.655
# i 145 more rows
> ## Añadimos los valores de las observaciones 5 y 6 al gráfico de supervivencia
> points( pred_ph_surv_12 %>% slice(5,6), pch=16, cex=1.2 )
> points( pred_ph_surv_60 %>% slice(5,6), pch=16, cex=1.2 )
```

- Añadimos al gráfico, las supervivencias de las observaciones 5 y 6, en los tiempos 12 y 60. Se comprueba que las predicciones de la supervivencia son las del modelo de Cox

Predicciones del Modelo de Regresión de Cox

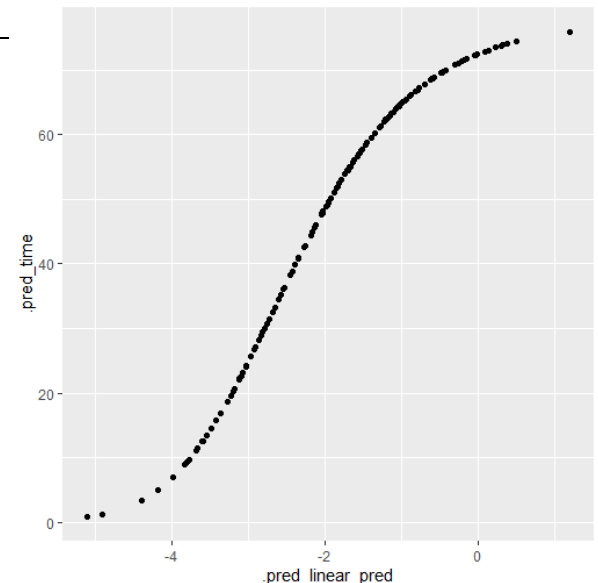
```
> ## predict tipo "linear_pred" y "time" de tidymodels
> pred_ph_lin_pred <- predict(ph_fit, xx_test, type = "linear_pred")
> pred_ph_time <- predict(ph_fit, xx_test, type = "time")
> pred_ph_df <- bind_cols(xx_test %>% select(surv_var),
+                          pred_ph_lin_pred, pred_ph_time )
> pred_ph_df %>% print(n=6)
# A tibble: 151 × 3
  surv_var .pred_linear_pred .pred_time
  <Surv>    <dbl>          <dbl>
1 71.35934+ -0.593            68.7
2 49.14990 -0.422            70.0
3 71.45791+ -1.18           62.6
4 77.30595 -1.04           64.3
5 70.27515+ -0.0243         72.4
> head(pred_cox_lin_pred)
      1      2      3      4      5      6
0.59313268 0.42199362 1.18380327 1.04330042 0.02426664 1.45051481
```

- La función ***predict()*** de ***tidymodels*** permite también predecir el predictor lineal (***type="linear_pred"***)
- **Muy importante:** la función ***predict()*** de ***tidymodels*** cambia el signo al predictor lineal de la fórmula de Cox. Por tanto, se debe interpretar que los **valores altos en este predictor lineal tienen mejor supervivencia** ("low risk")
- También, se puede obtener la predicción del **tiempo predicho** en el que se va a producir el evento (***type="time"***), pero es una estimación **difícil de interpretar**

C-index

```
> ## c-index en Testing
> concordance_survival(pred_ph_df, truth = surv_var, estimate = .pred_linear_pred )
# A tibble: 1 × 3
  .metric      .estimator .estimate
  <chr>        <chr>      <dbl>
1 concordance_survival standard    0.737
> concordance_survival(pred_ph_df, truth = surv_var, estimate = .pred_time )
# A tibble: 1 × 3
  .metric      .estimator .estimate
  <chr>        <chr>      <dbl>
1 concordance_survival standard    0.737
>
> ## Relación entre el predictor lineal y el tiempo de supervivencia
> pred_ph_df %>%
+   ggplot(aes(.pred_linear_pred, .pred_time)) +
+   geom_point()
```

- La función ***concordance_survival()*** permite calcular el **c-index**, y normalmente se usa el **predictor lineal**
- En este caso el **c-index** es **0.737**
- Si se usan las predicciones del tiempo de supervivencia, el resultado es el mismo, puesto que es una transformación monótona del predictor lineal



Evaluación con técnicas de remuestreo

```
> ## 2.5.- Capacidad Predictiva con Técnicas de Remuestreo
> ## Workflow
> ph_wflow <- workflow() %>%
+   add_model(ph_spec) %>%
+   add_formula(surv_var ~ .)
>
> ## Remuestreo
> ph_res <- ph_wflow %>%
+   fit_resamples( resamples = cv_split,
+                 metrics = metric_set(concordance_survival) )
>
> ## Resultados
> collect_metrics(ph_res)
# A tibble: 1 × 6
  .metric          .estimator mean      n std_err .config
  <chr>          <chr>    <dbl> <int>   <dbl> <chr>
1 concordance_survival standard  0.782    20 0.00847 Preprocessor1_Model1
```

- La función ***fit_resamples()*** calcula la capacidad predictiva del modelo, basada en el **c-index**, con las muestras del esquema de **remuestreo** que se ha creado (***resamples=***)
- Previamente se ha creado un ***workflow*** que incluye las especificaciones del modelo (***add_model***) y la fórmula (***add_formula***)
- En este caso, el **c-index** estimado con remuestreo es **0.782**

ANEXO 2

Cross-validated predictions

Cross-validated predictions

- En la **optimización de parámetros**, con **validación cruzada**, es posible almacenar **todas las predicciones** que se han realizado en las **muestras de evaluación**, con los modelos construidos en las muestras de análisis
- Para **cada observación de training**, dónde se ha ejecutado la optimización de parámetros, hay **una predicción**. Se llaman **cross-validated predictions**
 - Si la validación cruzada se ha **repetido varias veces**, se calcula **la media** de las predicciones, para cada observación
- Estas **predicciones de la validación cruzada** pueden ser usadas de forma semejante a las predicciones de una muestra de testing. Se pueden obtener **medidas de capacidad predictiva** y los **gráficos** derivados
 - C-index, AUC(t), Brier Score(t)
 - **Cross-Validated Kaplan-Meier Plot** con grupos de riesgo
- Esto tiene especial interés en **muestras pequeñas**, donde no se ha podido hacer una partición inicial, y por tanto, **no hay muestra de testing**

Cross-validated predictions

```
> ## Predicciones del proceso de tune. Cross-Validated Predictions
> assess_res <- collect_predictions(tune_result_enet)
> assess_res %>% print(n=4)
# A tibble: 209,400 × 9
  .pred      .pred_time id      id2      .row      penalty mixture  surv_var
1 <tibble [1 × 3]>    61.5 Repeat1 Fold01    16 0.0000000001      0 67.84394+
2 <tibble [1 × 3]>    51.3 Repeat1 Fold01    22 0.0000000001      0 64.22998+
3 <tibble [1 × 3]>    67.1 Repeat1 Fold01    27 0.0000000001      0 63.73717+
4 <tibble [1 × 3]>    60.6 Repeat1 Fold01    28 0.0000000001      0 63.50719+
> 349 * 300 * 2      ## 349 obs. en training, 300 parámetros, 2 repeticiones de CV
[1] 209400
> ## Predicciones con summarize ( 1 valor por observación )
> assess_res_summ <- collect_predictions(tune_result_enet, summarize=TRUE)
> assess_res_summ %>% print(n=4)
# A tibble: 104,700 × 7
  .pred      .pred_time .row  penalty mixture  surv_var .config
1 <tibble [1 × 3]>    39.7      1 1 e-10      0 71.55647+ Preprocessor1_Model1001
2 <tibble [1 × 3]>    39.7      1 1.60e-10    0 71.55647+ Preprocessor1_Model1002
3 <tibble [1 × 3]>    39.7      1 2.56e-10    0 71.55647+ Preprocessor1_Model1003
4 <tibble [1 × 3]>    39.7      1 4.09e-10    0 71.55647+ Preprocessor1_Model1004
```

- La función ***collect_predictions()*** permite recuperar todas las predicciones del proceso de la validación cruzada. El parámetro es el objeto de la optimización de parámetros, que se hizo con la función ***tune_grid()*** con la opción ***save_pred = TRUE***
- Si esta función se usa con el parámetro ***summarize=TRUE*** calcula las medias de las predicciones por observación

Cross-validated predictions

```
> ## Predicciones del mejor modelo
> assess_res_summ_best <-
+   assess_res_summ %>%
+   filter( penalty == tune_best_enet$penalty,
+           mixture == tune_best_enet$mixture )
> assess_res_summ_best %>% print(n=4)
# A tibble: 349 × 7
  .pred          .pred_time .row penalty mixture  surv_var .config
  <list>          <dbl> <int>   <dbl>   <dbl>   <Surv> <chr>
1 <tibble [1 × 3]>    40.7     1  0.153    0.2  71.55647+ Preprocessor1_Model1096
2 <tibble [1 × 3]>    56.2     2  0.153    0.2  71.95072+ Preprocessor1_Model1096
3 <tibble [1 × 3]>    59.1     3  0.153    0.2  70.01232+ Preprocessor1_Model1096
4 <tibble [1 × 3]>    68.9     4  0.153    0.2  69.71663+ Preprocessor1_Model1096
>
> ## Se calcula el c-index
> concordance_survival( assess_res_summ_best, truth = surv_var, .pred_time )
  .metric          .estimator .estimate
1 concordance_survival standard      0.782
```

- Se seleccionan las filas de los **parámetros óptimos**, y ahora se tiene **una única predicción por observación**, del proceso de validación cruzada que se hizo en la **muestra de training**
- Se calcula el **c-index** con esta predicción, y es **0.782**, que es muy parecido al que se obtuvo en la optimización de parámetros, 0.787, pero no tienen por qué ser igual, ya que este cálculo se obtiene de la **media de las predicciones**, y el primero era una media de 20 c-indexes, obtenidos en las 20 particiones del proceso de optimización

Cross-validated predictions. Grupos de Riesgo

```
> ## Cross-Validated Predictions - Grupos de High / Low Risks
> ## Creación de los grupos con la mediana de Training
> pred_enet_time_train <- predict(enet_fit, xx_train, type = "time")
> cutoff_time <- median(pred_enet_time_train$.pred_time)
>
> pred_enet_time_group <- as.integer( assess_res_summ_best$.pred_time >= cutoff_time )
> pred_enet_time_group <- factor(pred_enet_time_group, levels=0:1,
+                               labels=c("High Risk", "Low Risk"))
> table(pred_enet_time_group)
pred_enet_time_group
High Risk  Low Risk
    177      172
```

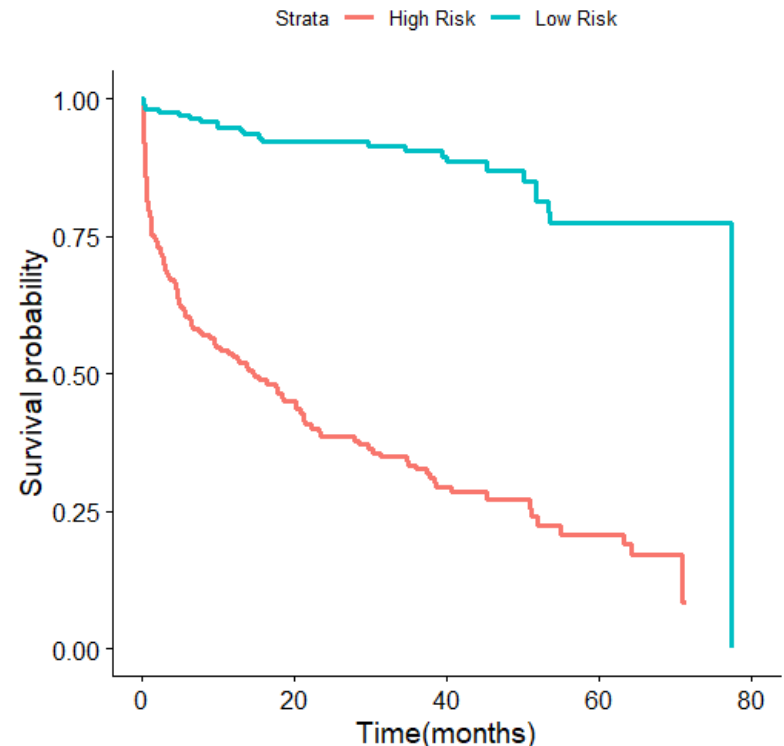
- Se obtiene la **mediana** de la predicción de tipo “***time***”, de las observaciones en la muestra de **training**
- Cada observación de la **muestra de training** se asigna a los grupos de **High / Low Risk**, según esté por debajo o por encima de la mediana, usando la **predicción** que se ha obtenido ahora de la **validación cruzada**

Cross-validated Kaplan-Meier Plot

```
> ## Cross-Validated KM Plot y Test log-rank, en xx_train
> surv_group <- survfit( surv_var ~ pred_enet_time_group, data=xx_train )
> survdiff( surv_var ~ pred_enet_time_group, data=xx_train)

Chisq= 131  on 1 degrees of freedom, p= <2e-16
>
> dev.new()
> ggsurvplot(surv_group, xlab="Time(months)", conf.int = FALSE, censor = FALSE,
+           size=1.2, legend.labs = levels(pred_enet_time_group))
```

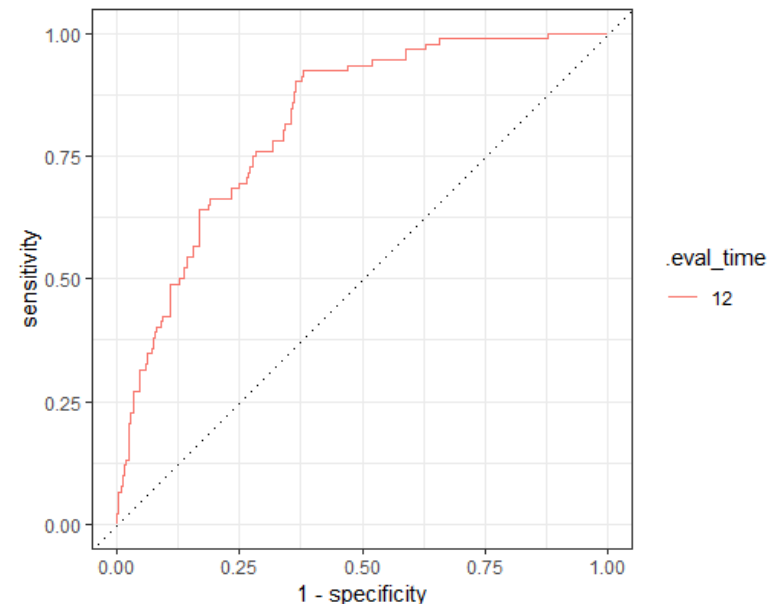
- Se obtienen las **gráficas KM** de los dos grupos de riesgo, que en este contexto se llaman **Cross-validated Kaplan-Meier Plot**
- El test log-rank tiene **p.value < 0.001**

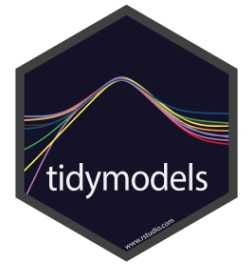


Cross-validated time-dependent ROC curves

```
> ## AUC
> roc_scores <-
+   roc_auc_survival(assess_res_summ_best, truth = surv_var, .pred )
> roc_scores
  .metric      .estimator .eval_time .estimate
1 roc_auc_survival standard         12      0.820
>
> ## Todas las curvas ROC
> all_roc_curve <-
+   roc_curve_survival(assess_res_summ_best, truth = surv_var, .pred )
> dev.new()
> autoplot(all_roc_curve)
```

- Con la función ***roc_auc_survival()*** se obtiene el AUC, que es **0.820** en **t = 12**, que es el único tiempo de evaluación que se introdujo en la función ***tune_grid()***
- Con la función ***roc_curve_survival()*** se obtiene la curva ROC en **t = 12**.
- En este contexto se llama **Cross-validated time-dependent ROC curve**





GRACIAS !!!!