GEICAM
investigación en cáncer de mama

Evaluación mediante un Test de Significación de la Capacidad Predictiva de un Modelo

X Jornadas de Usuarios de R
Murcia

Jesús Herranz Valera
jherranz@geicam.org
Bioestadístico GEICAM

22 de noviembre de 2018

Índice

- ✓ **Introducción**
- ✓ **Construcción de un Modelo Predictivo**
- ✓ **Evaluación de un Modelo Predictivo**

- ✓ **Significación Estadística del Modelo**
- ✓ **Test de Permutaciones**

- ✓ **Paralelización del Proceso**

- ✓ **Comparación de la Capacidad Predictiva de 2 Modelos**

Introducción

- Supongamos que hemos construido un **modelo predictivo de clasificación** y hemos evaluado su **capacidad predictiva** mediante una **técnica de remuestreo** (validación cruzada)

  - AUC = 0.54 El modelo predice muy poco
  - AUC = 0.87 El modelo predice muy bien

  - AUC = 0.62 o = 0.66 o = 0.70 ¿Cómo predice el modelo?
  - Posiblemente depende del contexto del problema

  - **¿Tenemos un test que nos diga si ese AUC es distinto de 0.5, que es el AUC de un modelo que predice de forma aleatoria?**

Introducción

- Supongamos que hemos construido un **modelo predictivo de regresión** y hemos evaluado su **capacidad predictiva** mediante una **técnica de remuestreo** (validación cruzada)

  - hemos obtenido un MSE = 1.23 (MSE = Error cuadrático medio)
  - ¿es mucho? ¿es poco?
  - El MSE se mide en las unidades de la variable respuesta al cuadrado
  - ¿hay algún “MSE basal”?
  - ¿El MSE=1.23 está lejos de un “MSE basal” si fuera 1.89, por ejemplo?

  - **¿Tenemos un test que nos diga si ese MSE es distinto del MSE que se obtendría en un modelo que predice de forma aleatoria?**

Introducción

- **Modelo 1**: modelo predictivo de clasificación basado en 10 predictores
  - AUC = 0.71

- **Modelo 2**: modelo predictivo de clasificación basado en 15 predictores, incluidos los 10 del modelo anterior
  - AUC = 0.74

- ¿Hemos mejorado el modelo? ¿merece la pena añadir esas 5 variables?
- Posiblemente depende del contexto del problema

- **¿Tenemos un test que nos compare los 2 AUCs?**

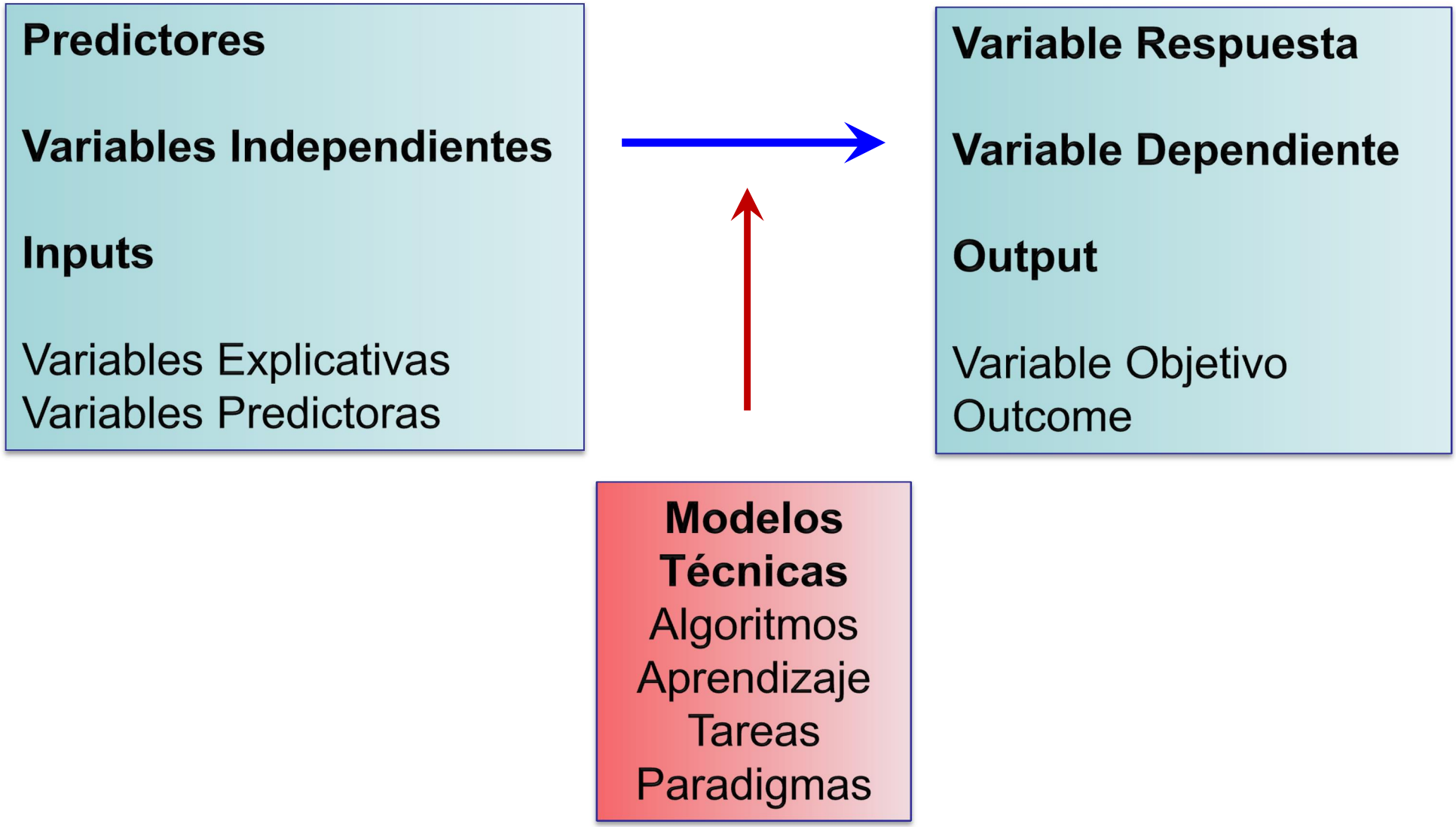
Aprendizaje Supervisado. Clasificación. Regresión

- **Aprendizaje Supervisado. Modelos Predictivos**
  - Técnicas donde el objetivo es **predecir** el valor de una **variable respuesta** basado en numerosas variables de entrada, llamadas **variables predictoras**

- Técnicas de **Clasificación**
  - Modelos predictivos cuando la **variable respuesta es categórica**
  - Cada observación pertenece a **una clase**
  - El modelo proporciona la **clase** predicha y la **probabilidad** de pertenecer a cada una de las clases

- Técnicas de **Regresión**
  - Modelos predictivos cuando la **variable respuesta es continua**
  - El modelo predice un **valor numérico** para cada observación

Modelos Predictivos



Evaluación de Modelos. Sobreajuste

- Evaluación de los modelos mediante su **capacidad predictiva**
  - Medidas objetivas basadas en la **comparación de los valores observados** para la variable respuesta y los **valores predichos** por el modelo

- Si la evaluación del modelo se hace sobre el propio conjunto de datos con el que se ha construido el modelo, se produce un **sobreajuste (overfitting)**

- Es necesario evaluar el modelo en un **conjunto de datos diferente** al que se ha utilizado para construirlo
  - Se dice que se obtienen **medidas de error "honestas"**

Medidas de la Capacidad Predictiva. Clasificación

- En el caso de una variable predictora categórica, la medida más extendida es la **Tasa de Error** (Error Rate), proporción de clasificaciones incorrectas

		Predicho			
Observ		**Y=1**	**Y=2**	**…**	**Y=k**
Y=1		F_{11}	F_{12}	…	F_{1k}
Y=2		F_{21}	F_{22}	…	F_{2k}
...		…	…	…	…
Y=k		F_{k1}	F_{k2}	…	F_{kk}

$$\text{Error Rate} = \frac{\sum_{i=1,j=1,i\neq j}^{k} F_{ij}}{N}$$

	Predicho	
Observado	**Y=1**	**Y=0**
Y=1	VP	FN
Y=0	FP	VN

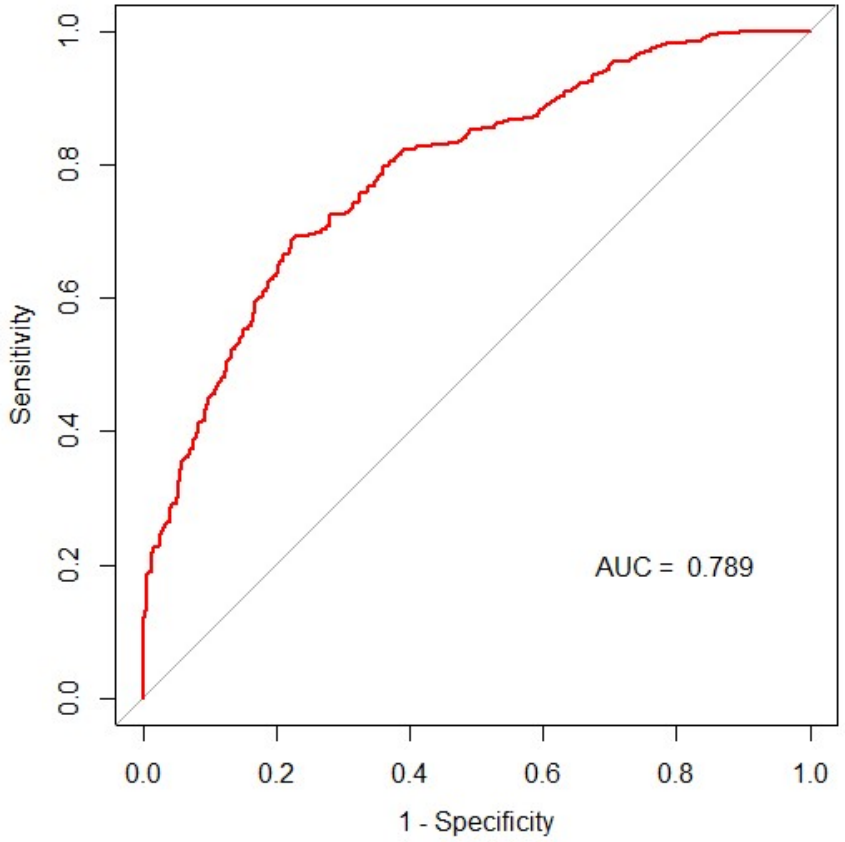
$$\text{Sensibilidad} = \frac{VP}{VP + FN}$$

$$\text{Especificidad} = \frac{VN}{VN + FP}$$

Medidas de la Capacidad Predictiva. Curvas ROC

- Las **curvas ROC** se utilizan para evaluar la capacidad predictiva de un **predictor continuo** en un problema de clasificación con **dos clases**
- La curva ROC se define con los puntos **(1 – especificidad, sensibilidad)** calculadas con las **probabilidades predichas** por el modelo

- El **área bajo la curva (AUC**) es considerado como una medida de la capacidad predictiva del predictor
  - Un predictor ideal tendría AUC = 1
  - Un predictor que predice de forma aleatoria tendría AUC = 0.5



Medidas de la Capacidad Predictiva. Regresión

- En Regresión, las medidas de capacidad predictiva de un modelo se basan en los **residuos**, diferencias entre los valores observados y predichos

- El **error cuadrático medio (MSE, mean squared error)** y su raíz cuadrada
  - El MSE se mide en unidades de la variable respuesta al cuadrado

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \qquad \mathrm{RMSE} = \sqrt{\mathrm{MSE}}$$

- Una propuesta para el **MSE basal** es considerar la media como la predicción más básica

$$\mathrm{MSE}_{\mathrm{basal}} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{y})^2$$

Métodos de Estimación. Training y Testing

- Un método para obtener una **estimación honesta** de la capacidad predictiva de un modelo es **dividir la muestra** original en dos submuestras:

  1. Una **muestra de entrenamiento** (**training set**) que se usa para la **construcción del modelo**
  2. Una **muestra de validación** (**testing set**) donde se realizan las **predicciones** y donde se **evalúa la capacidad predictiva** del modelo

- La **partición** se debe hacer de forma **aleatoria**

- Se ha extendido el uso de **Técnicas de remuestreo** que repiten este proceso de training y testing en varias submuestras
  - **Validación Cruzada**
  - **Bootstrapping**

Técnicas de remuestreo. Validación Cruzada

- El método de **Validación Cruzada (Cross-validation, CV)** consiste en dividir de **forma aleatoria** la muestra en **K subconjuntos de similar tamaño**, y ajustar **K modelos**, dejando cada vez una partición como conjunto de testing, y construyendo el modelo con las K-1 restantes

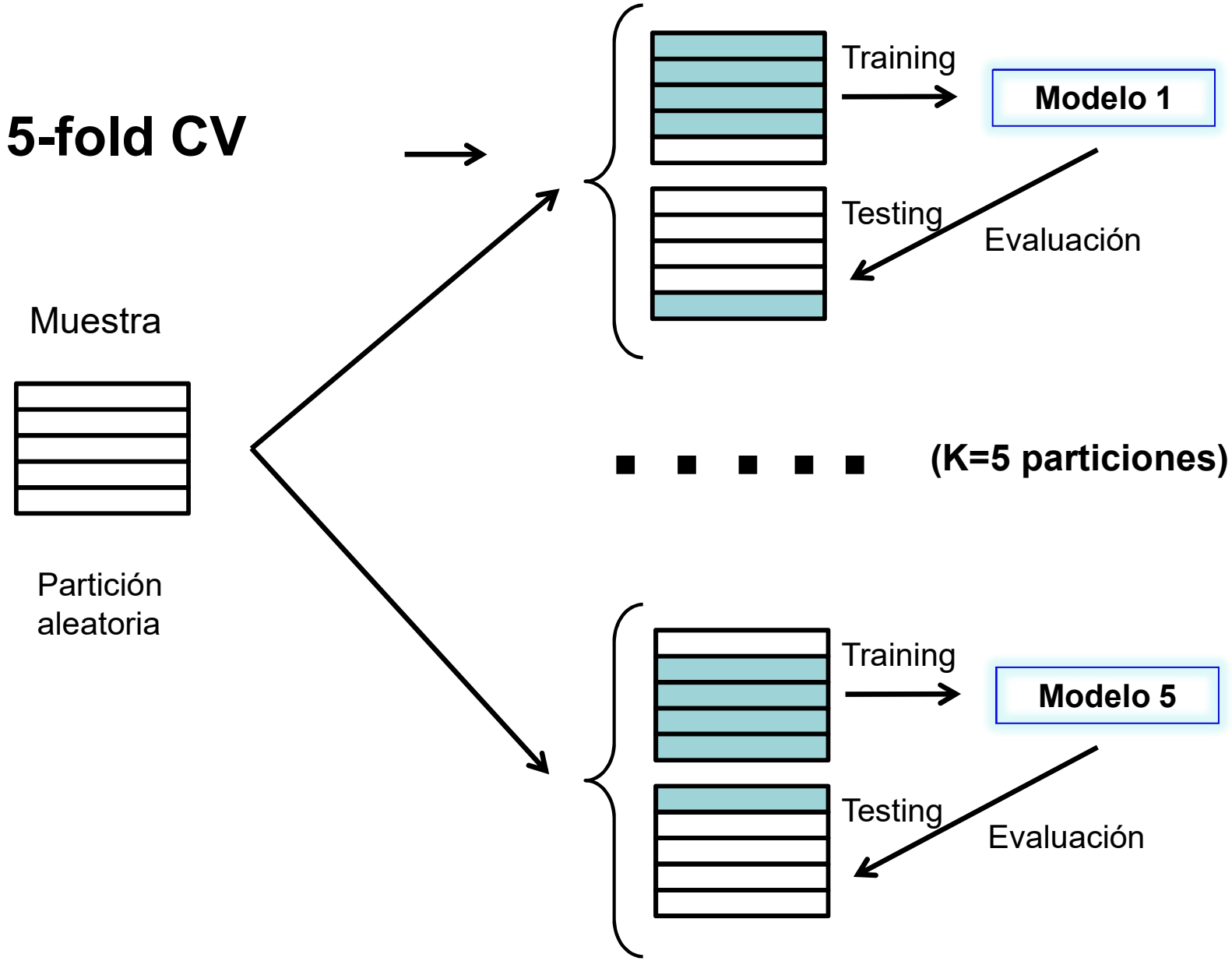
- La **estimación del error** se calcula como **promedio** de los K errores evaluados en las muestras de testing de las K particiones

- El **número K de particiones (folds)** se elige dependiendo del tamaño de la muestra. Se suele elegir **K=10** (**10-fold CV**) o **K=5**

- Se puede **repetir** varias veces el proceso de validación cruzada. Por ejemplo, si se repite 10 veces: **10 times 10-fold CV**

- Una variante en muestras pequeñas es **leave-one-out cross-validation** (LOOCV) donde K=N, y en cada subconjunto se deja fuera una observación

Técnicas de remuestreo. Validación Cruzada



Etapas. Construcción de Modelos Predictivos

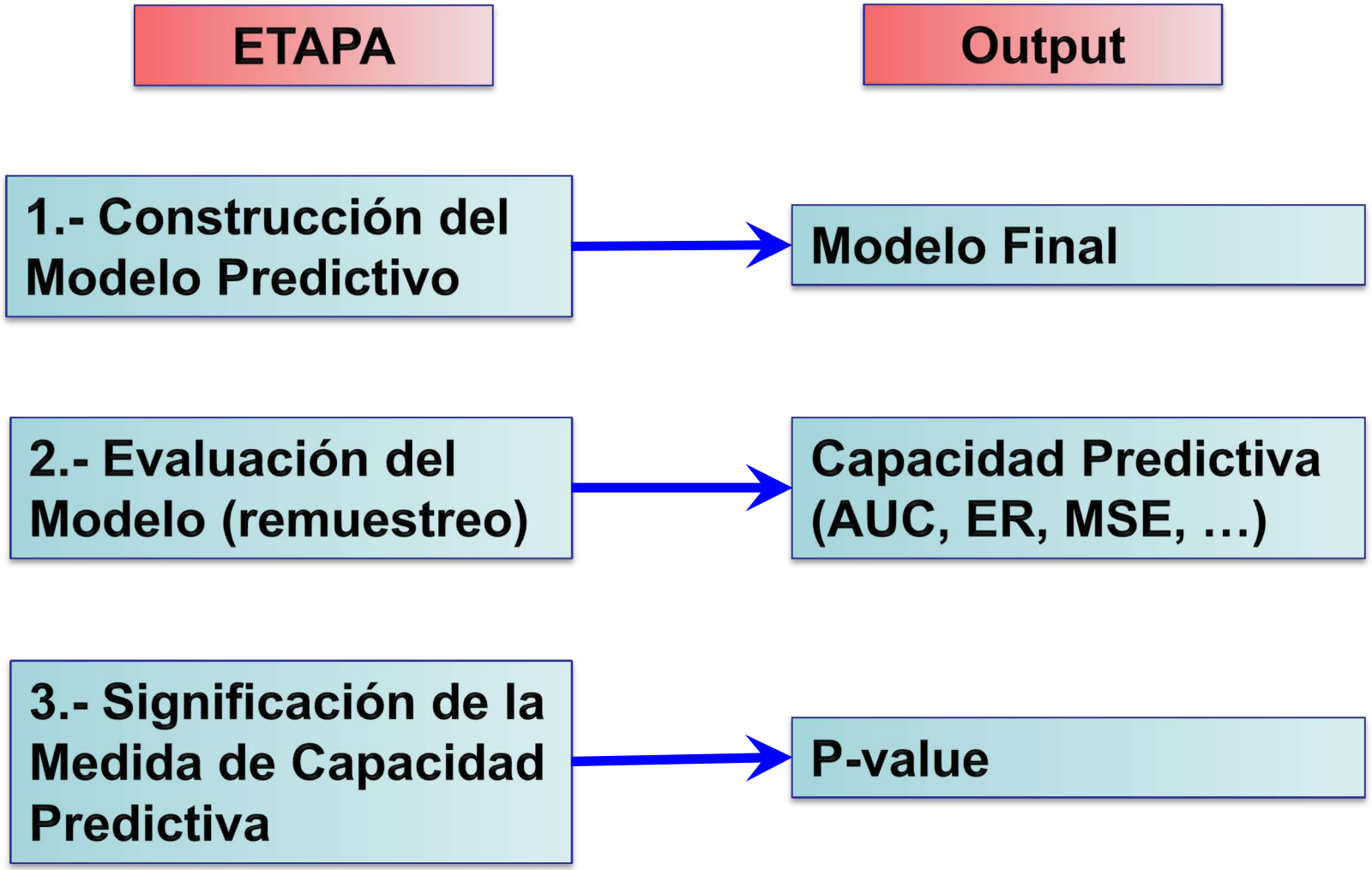
- **Pre-procesamiento de datos**
  - Se puede incluir un paso previo de selección de variables basado en **métodos no supervisados**, donde la variable respuesta no interviene en la selección

1. **Construcción del modelo predictivo**
   - **Selección de las variables** que van a formar parte del modelo (opcional)
   - **Optimización de los parámetros** del modelo (CV)
   - **Construcción** del modelo con los parámetros óptimos

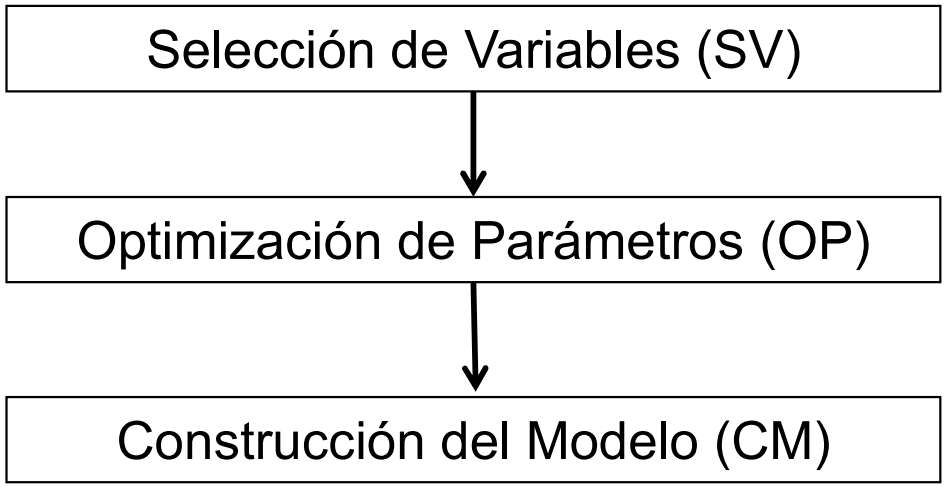
2. **Evaluación de la capacidad predictiva** del modelo
   - Técnicas de **remuestreo** (CV)

3. **Evaluación de la significación estadística** del modelo predictivo
   - Mediante **Test de Permutaciones**

Etapas. Construcción de Modelos Predictivos



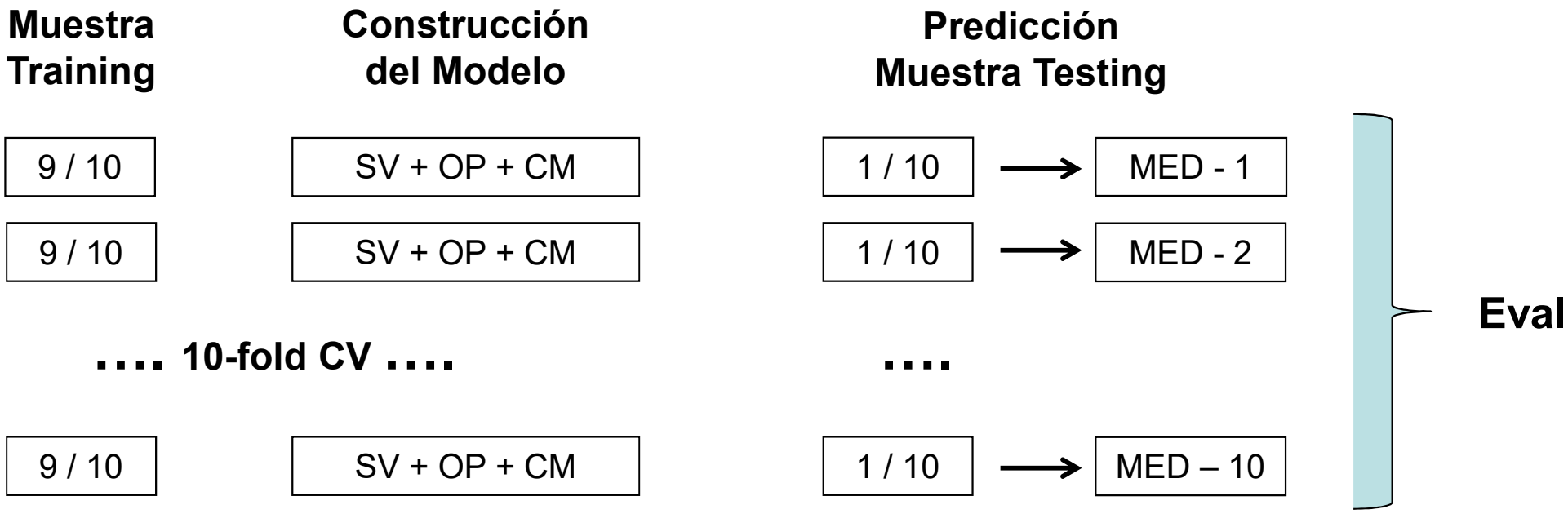
Etapa 1: Construcción del modelo predictivo



- El modelo predictivo se construye con la **muestra completa**
  - Una parte de la muestra se puede reservar como muestra “externa” de testing

- Este es el **“modelo final”,** el que se presenta como **resultado** del “problema de predicción”, y del que se pretende evaluar su capacidad predictiva y la significación estadística de dicha capacidad

Etapa 2: Evaluación de la capacidad predictiva (CV)



- La **selección de variables (SV)** se realiza en cada partición de la CV
- La **optimización de parámetros** (OP) se suele basar a su vez en un proceso de **CV**, que se repite de forma independiente en **cada partición de la CV**
- Se utilizan las **medidas** habituales: MSE, AUC, tasa de error, … que se **promedian** para obtener la **evaluación de la capacidad predictiva**

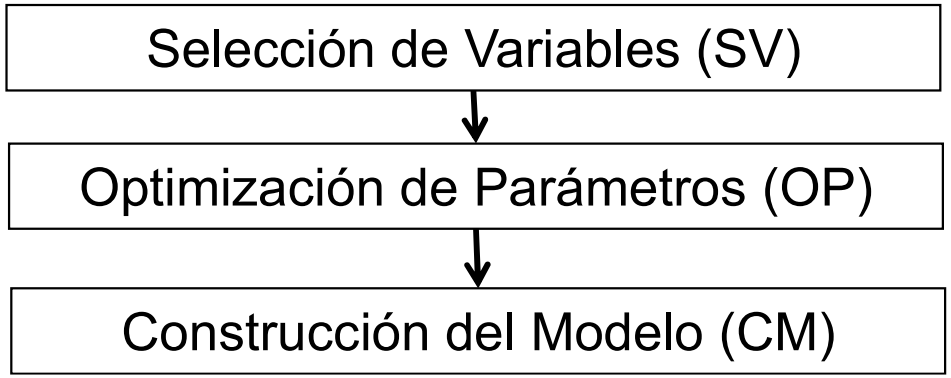
Ejemplo: Alzheimer

- **Objetivo del estudio:** analizar un conjunto de 124 **biomarcadores**, medidas de expresión de **proteínas**, asociados a las primeras etapas de **Alzheimer**

Nombre	Descripción	Categorías / Observaciones
124 Variables	Expresión de proteínas	Biomarcadores exploratorios
age	Edad	
male	Género	0 = mujer, 1 = hombre
tau, p_tau, Ab_42	3 proteínas conocidas que están asociadas a Alzheimer	
E4, E3, E2	3 alelos (SNPs) asociados a Alzheimer	0 = no presenta el alelo, 1 = presenta el alelo
Class	**Presencia de Alzheimer**	**1 = Alzheimer (primeras etapas) (“Impaired”)** **0 = Control**

- Problema de **Clasificación**, usando la variable respuesta binaria “**Class**”
- Ejemplo extraído de *“Applied Predictive Modeling”. Max Kuhn*

Ejemplo: Alzheimer



- La **selección de variables** se va a hacer mediante un **filtro univariante**
  - Se ajusta un modelo de regresión logística univariante para cada predictor
  - Se seleccionan aquellos donde p-value < threshold (= 0.10 por ejemplo)

- Se va a construir un modelo de **Support Vector Machine** con la función de **núcleo radial**
- Previamente, se optimiza el parámetro **gamma** del núcleo radial y el parámetro C, de **coste**, para penalizar los errores

Ejemplo: Función de un filtro univariante

```
## Selección de Variables. Filtro basado en P-valores con Respuesta binaria

FilterUni.Classif = function ( set.pred , var.resp )
{
  ## Se controla que Y es una variable binaria
  if( length(unique(var.resp)) != 2 ) stop("La variable respuesta debe ser binaria")

  ## Para cada variable se extrae el P-valor del LRT de la Regresión Logística
  p.val = apply( set.pred, 2 , function ( w.col )
                             { out.log = glm( var.resp ~ w.col, family=binomial )
                               anova(out.log, test="Chisq" )$"Pr(>Chi)"[2] } )

  ## Parámetros de salida: P-valores en el mismo orden que los predictores en set.pred
  list ( p.val = p.val )
}
```

- Se crea una **función** donde se ajusta un modelo de **regresión logística univariante** para cada una de las **variables predictoras** de un conjunto (set.pred) con una **variable respuesta binaria** (var.resp)
- Devuelve un vector con los **p-values** de todas las variables en el mismo orden de entrada

Ejemplo: Función con el proceso completo

```
> ProcessSVM = function ( set.train, set.test, name.resp , sign.level.sel = 0.10 )
+ {
+   ## Número de la columna donde está la variable respuesta y los predictores
+   w.column.Resp = which ( names(set.train) == name.resp  )
+   w.column.Pred = 1:ncol(set.train)
+   w.column.Pred = w.column.Pred [ - w.column.Resp ]     ## se quita la col de la resp
+
+   ## 1.- Seleccción de Variables. Filtro Univariante
+   p.values = FilterUni.Classif ( set.pred = set.train[ , w.column.Pred ] ,
+                                  var.resp = set.train [ , name.resp ] )$p.val
+
+   ## Columnas de los predictores que cumplen p-value < sign.level.sel
+   ## Si no hay ninguna se selecciona la del p-value mínimo
+   if ( min(p.values ) < sign.level.sel )
+   { column.sel = w.column.Pred [ which ( p.values < sign.level.sel ) ] }
+   else { column.sel = w.column.Pred [ which.min ( p.values ) ] }
+
+   ## Data frame con la Respuesta y los predictores con p-value < sign.level.sel
+   set.train.sel = set.train [  , c ( column.sel , w.column.Resp ) ]
```

- Se crea una **función** con el "**proceso completo**" ya que se va a utilizar en todas las etapas (construcción del modelo, evaluación y en el test)
- El **1º paso** es un **filtro** donde se seleccionan las variables predictoras que están asociadas con la respuesta con **p-value < 0.10 (regresión logística univariante)**
- Se crea un **dataframe** del **set de training** con las variables que han pasado **el filtro**

Ejemplo: Función con el proceso completo

```
+    ## 2.- Optimización de parámetros de SVM
+
+    tune.rad <- tune.svm( as.formula ( paste ( name.resp , " ~ . " ) ) ,
+                          data = set.train.sel, kernel="radial",
+                          gamma = c( 0.01, 0.005, 0.001 ) ,
+                          cost = c( 5, 10, 50 ) ,
+                          tunecontrol = tune.control ( cross = 5 ) )
+
+    ## 3.- SVM con los parámetros óptimos del paso anterior
+
+    svm.out = svm( as.formula ( paste ( name.resp , " ~ . " ) ) ,
+                   data = set.train.sel, kernel="radial",
+                   gamma = tune.rad$best.parameters$gamma,
+                   cost = tune.rad$best.parameters$cost,
+                   probability=TRUE )
```

- El **2º paso** es la **optimización de los parámetros** de **Support Vector Machines** (SVM) con el núcleo radial (*kernel="radial"*). Usamos la función ***tune.svm()*** del paquete ***e1071***. Se optimizan ***gamma*** y el ***coste*** usando 5-CV
- El **3º paso** es construir el modelo de **SVM con los parámetros óptimos.** Usamos la función ***svm()*** del paquete ***e1071***
- Estos dos pasos se aplican al dataframe extraído del set de training con las variables seleccionadas en el primer paso, las que han pasado el filtro univariante

Ejemplo: Función con el proceso completo

```
+    ## 4.- Predicción en el conjunto de testing
+
+    ## Clases
+    svm.class =  predict( svm.out, newdata = set.test )
+
+    ## Probabilidades
+    svm.pred = predict( svm.out, newdata = set.test, decision.values=TRUE,
                          probability=TRUE)
+    svm.prob = attr(svm.pred, "probabilities" )
+
+    ## OUTPUT: Modelo construido y Predicción en el conjunto de testing
+    list( out.mod = svm.out ,
+          pred.prob = svm.prob, pred.class = svm.class  )
+ }
```

- El **4º paso** es obtener **las predicciones** del modelo en el **conjunto de testing**
- Se obtienen las **clases** predichas para cada observación y las **probabilidades** de pertenecer a cada una de las clases con la función ***predict()***

- Los objetos de **salida de la función** se unen en una lista: el modelo y las predicciones (clases y probabilidades)

Ejemplo. Inicializar

```
> library(e1071)
> library(pROC)
> library(caret)
> ## Se cargan las funciones generales
> source ( "C://Taller Significación Modelos Predictivos/Funciones Generales.r")
> source ( "C://Taller Significación Modelos Predictivos/Funciones SVM.r")
>
> ## Fichero Datos: Alzheimer
> xx = read.csv("C://Taller Significación Modelos Predictivos/Alzheimer.csv", sep=";")
> dim(xx)
[1] 333 133
> xx$Class = as.factor(xx$Class)
> n.all = nrow(xx)                    ## Número de observaciones
>
> ## Distribución de la Variable Respuesta
> table ( xx$Class )
 Control Impaired
     242       91
> prop.table ( table ( xx$Class ) )
  Control  Impaired
0.7267267 0.2732733
```

- Se cargan las **librerías**, las **funciones** generales (ER, AUC), la función creada con el proceso completo con SVM
- Se leen **los datos** (333 observaciones y 133 variables) y se convierte la variable respuesta **“Class”** a factor. La categoría de interés es “Impaired” (27%)

Ejemplo. Inicializar

```
> ## Número de la columna donde está la variable respuesta
> column.Resp = which ( names(xx) == "Class" )
>
> ## Normalización de las Variables (con SVM es necesario)
> w.trans = preProcess (  xx [ , - column.Resp ] , method = c ("center", "scale") )
> xx [ , - column.Resp ]  = predict( w.trans , xx [ , - column.Resp ] )
```

- Se guarda en una variable el número de la columna donde está la variable respuesta
- Se normalizan las variables predictoras a media 0 y desviación estándar 1 con la función ***preProcess()*** del paquete ***caret***

- La **normalización** de los datos dependerá de la técnica que se use para la construcción del modelo. En el caso de **Support Vector Machines** los datos se normalizan porque la técnica está basada en distancias

Ejemplo: Etapa 1: Construcción del modelo

```
> ##############################################################################
> ## 1.- Construcción del Modelo Predictivo
> ##############################################################################
>
> ## Modelo "Final". Proceso completo (SV + OP + CM)
> mod.all = ProcessSVM ( xx, xx, "Class" )
> names(mod.all)
[1] "out.mod"    "pred.prob"  "pred.class"
> class(mod.all$out.mod)
[1] "svm.formula" "svm"
>
> ## Estimación sobreajustada de la Tasa de Error y el AUC. NO REPORTAR NUNCA !!!!
> ErrRate( xx$Class , mod.all$pred.class )$err.rate
[1] 0.07807808
> ROCAnalysis ( xx$Class, mod.all$pred.prob [ , "Impaired" ])$AUC
[1] 0.9686677
```

- Se construye el **"modelo predictivo final"** con la muestra observada, utilizando la función ***ProcessSVM*** que contiene el proceso completo (SV+OP+CM)
- El **"modelo final**" está almacenado en *mod.all$out.mod* , que es un modelo de Support Vector Machine
- El **AUC** y la **tasa de error** son evaluados en la misma muestra con la que se ha construido el modelo y están **sobreajustadas** . Se llama error "aparente" del modelo y **nunca se deben reportar**

Ejemplo: Etapa 2: Evaluación del modelo con CV

```
> ##############################################################################
> ## 2.- Evaluación de la Capacidad Predictiva con CV
> ##############################################################################
>
> ## Número de folder en la CV (X-fold CV)
> n.cv = 10
>
> ## Vectores para almacenar las Tasas de Error y los AUCs
> w.error = rep ( NA, n.cv )
> w.auc   = rep ( NA, n.cv )
>
> ## Crea el vector "folds" que contiene el grupo (fold) de cada observación
> folds = sample ( rep ( 1:n.cv, length=n.all ) )
>
> head(folds)
[1]  9  3  6 10  9  2
> table (folds)
folds
 1  2  3  4  5  6  7  8  9 10
34 34 34 33 33 33 33 33 33 33
```

- Se va a hacer **validación cruzada** con 10 particiones
- Se crean 2 vectores para almacenar los AUC y ER evaluadas en las 10 particiones de testing
- Se crean las **particiones** usando la función ***sample()*** (Ver anexo)

Ejemplo: Etapa 2: Evaluación del modelo con CV

```
> for ( ind.cv in 1:n.cv )
+ { ## Creamos los datasets de training y testing en este fold
+   cv.train = xx [ folds != ind.cv , ]
+   cv.test  = xx [ folds == ind.cv , ]
+
+   ## Proceso completo (SV + OP + CM)
+   ww.mod = ProcessSVM ( cv.train, cv.test, "Class" )
+
+   ## Error Rate and AUC
+   w.error [ind.cv] = ErrRate ( cv.test$Class, ww.mod$pred.class )$err.rate
+   w.auc [ind.cv] = ROCAnalysis ( cv.test$Class, ww.mod$pred.prob[ , "Impaired" ])$AUC
+ }
> ## Error y AUC medios
> ER.Obs = mean(w.error)
> AUC.Obs = mean(w.auc)
> ER.Obs
[1] 0.1319964
> AUC.Obs
[1] 0.8785916
```

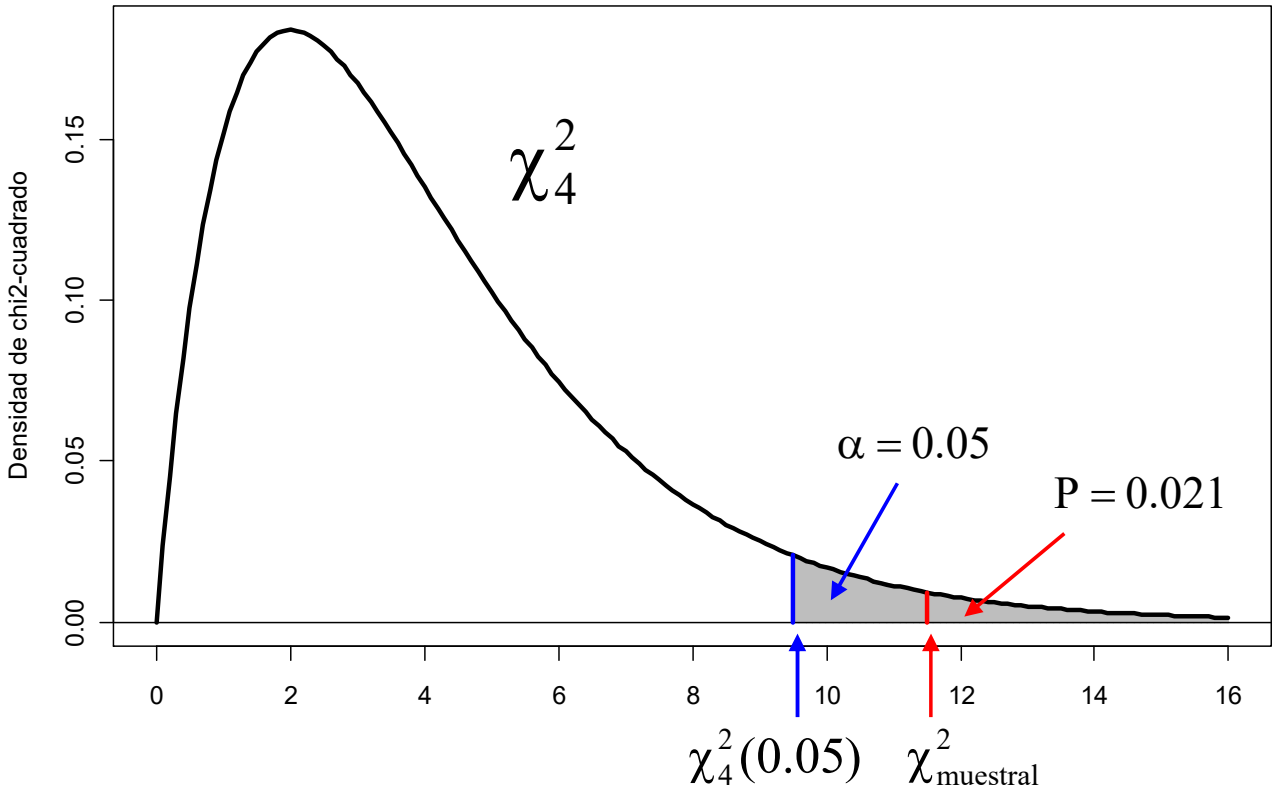
- Para cada partición de la CV, se crean los subconjuntos de **training y testing**
- Se ajusta el **proceso completo,** en el conjunto de **training**, y se **evalúa** el modelo en el conjunto de **testing**, donde se calculan el AUC y la ER
- Las estimaciones de la **capacidad predictiva del modelo** son las medias en las 10 particiones: **AUC= 0.879** y **ER=0.132**

Significación de un modelo predictivo

- Se pretende evaluar la **significación estadística** de las medidas de la **capacidad predictiva** de un modelo, obtenidas con técnicas de **remuestreo**
- Por ejemplo:
  - H_0: AUC = 0.5 que corresponde a un modelo que clasifica aleatoriamente
  - H_0: MSE es igual al de un modelo que predice aleatoriamente

- Se llama **P-valor de un test** a la probabilidad de obtener, suponiendo que la hipótesis nula H_0 sea cierta, un resultado al menos tan extremo como el que realmente se ha obtenido en la muestra observada
  - Se basa en la **distribución teórica** de un estadístico de contraste **bajo H_0** , y en evaluar el estadístico muestral, basado en las observaciones, en esa distribución
  - Si el P-valor es pequeño (< 0.05), significa que hay poca probabilidad de haber obtenido los valores observados si se cumpliese la H_0 y se rechaza la hipotesis

Significación de un modelo predictivo



- Rechazamos H_0 ya que:

$$\chi^2_4(0.05) = 9.49 < 11.5 = \chi^2_{\text{muestral}}$$

$$\alpha = 0.05 > 0.021 = \text{P - valor}$$

- El estadístico de contraste obtenido en la muestra tiene una probabilidad muy pequeña de proceder de la distribución teórica del estadístico si H_0 fuera cierta, y por tanto, rechazamos H_0

Significación de un modelo predictivo

- El **supuesto básico** para determinar la distribución teórica de los estadísticos de contraste de los tests es que se basan en **observaciones independientes**
- Sin embargo, las predicciones obtenidas por remuestreo no cumplen este supuesto ya que **las predicciones no son independientes**, puesto que se han obtenido con modelos muy relacionados

- Además, existen **diferentes escenarios** para generar las predicciones
  - Si se usa CV repetida, hay **varias predicciones** para cada observación
  - Las **medidas de capacidad predictiva** (AUC, MSE) se obtienen como **medias** de esas medidas en las submuestras del remuestreo
  - Las **predicciones** proporcionadas por cada **técnica estadística** tienen propiedades distintas

- En estas condiciones, las **distribuciones teóricas** de los **estadísticos de contraste** que se usan en los tests de hipótesis **no son válidas**

Test de permutaciones

- Para evaluar la **significación estadística** de un modelo predictivo se repetirá muchas veces el **procedimiento completo** con el que se ha construido y evaluado el modelo, **permutando los valores de la variable respuesta**
  - Se **asignan aleatoriamente** los valores de **la variable respuesta** a las observaciones, y así "se rompe" la posible relación que había entre la variable respuesta y los predictores
  - De esta forma, se construye la **distribución nula** (distribución bajo la hipótesis nula de no asociación) de la medida de la capacidad predictiva del modelo (AUC, MSE) de un **modelo que predice aleatoriamente**

- Este **proceso** se repite muchas veces (1.000 – 10.000), volviendo a permutar los valores de la variable respuesta, y se evalúa la **capacidad predictiva** de los modelos con la variable **respuesta permutada**

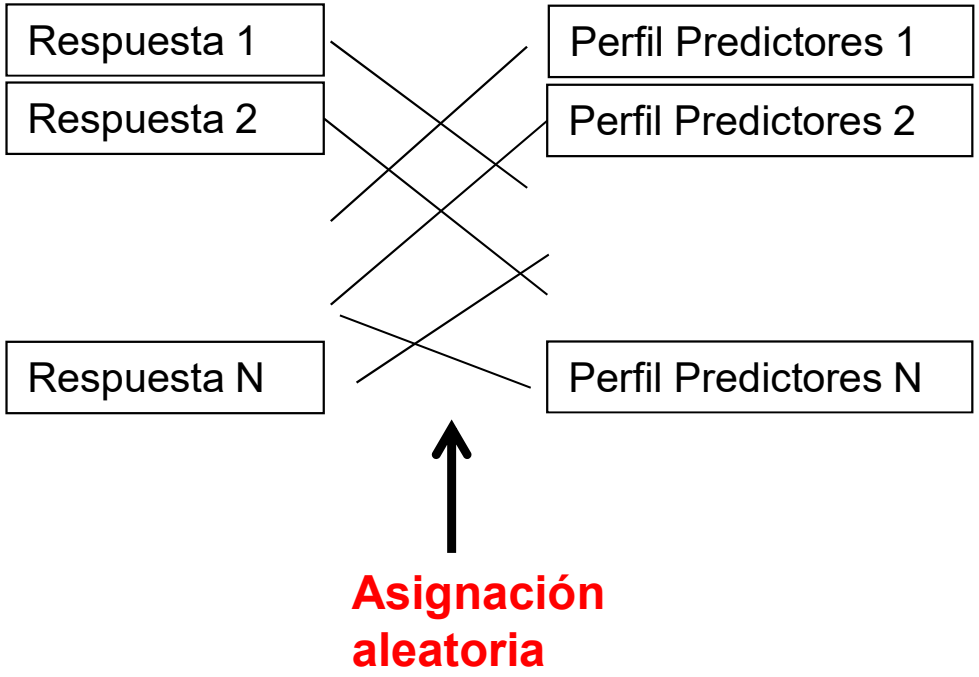
- Solo se permutan los valores de la variable respuesta para mantener la **estructura de correlaciones** entre las variables predictoras

Test de permutaciones

Muestra observada

Respuesta 1	Perfil Predictores 1
Respuesta 2	Perfil Predictores 2
.... N	
Respuesta N	Perfil Predictores N

Muestra con respuesta permutada



- El modelo construido con las respuestas permutadas **predice aleatoriamente** ya que no hay ninguna relación entre las respuestas permutadas y los predictores, y esperamos que no tenga **ninguna capacidad predictiva**

Test de permutaciones

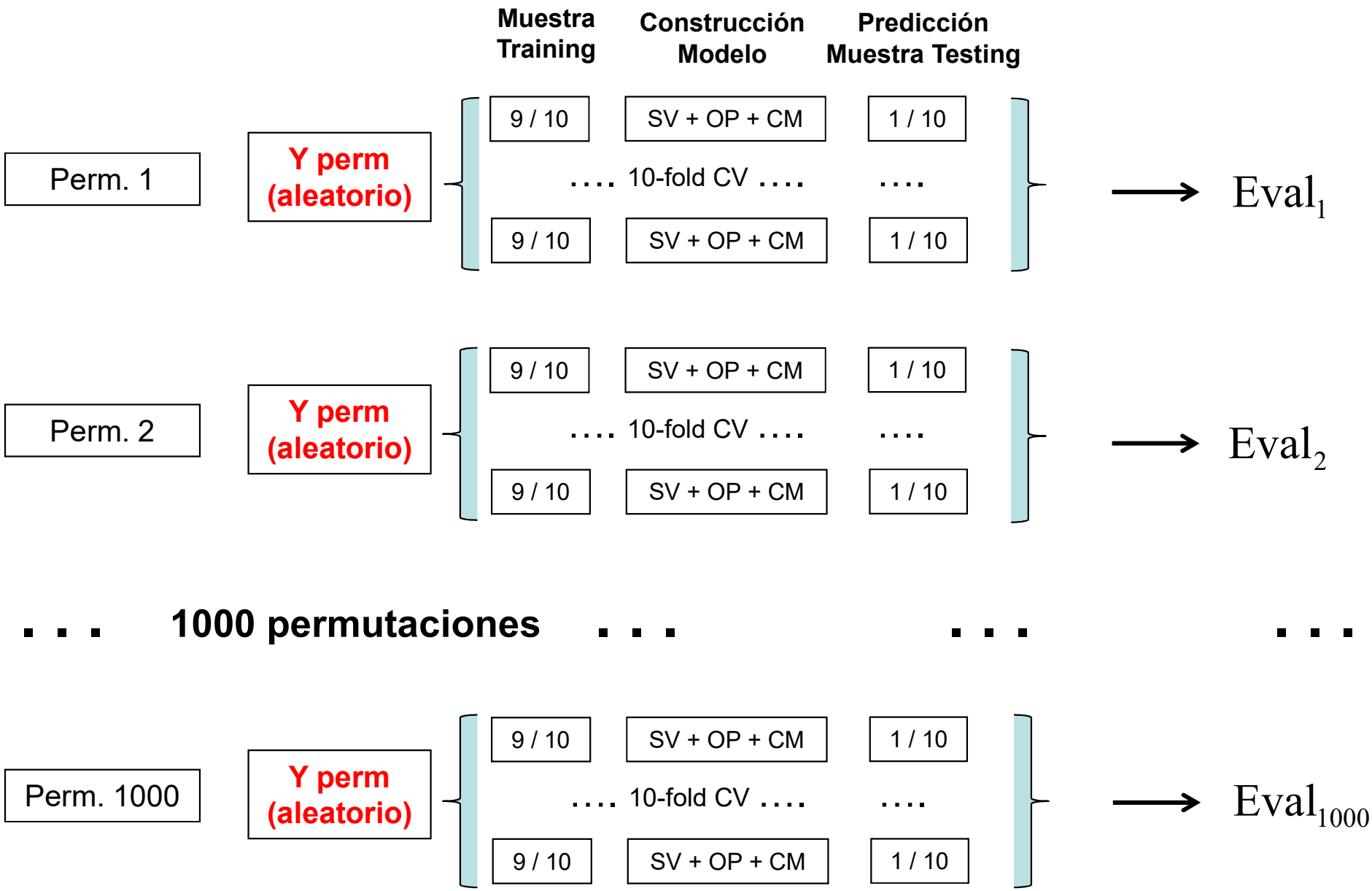
- Se pretende **cuantificar cuántas veces** un modelo generado con respuestas permutadas **predice mejor** que el modelo basado en la muestra observada

- El **P valor de un test de permutaciones** es el **número de veces** que mediante una asignación aleatoria de la variable respuesta, obtenemos **mejores predicciones** que la que se obtuvo con los datos observados, dividido por el número de permutaciones
  - El P-valor de un test es la probabilidad de obtener un resultado al menos tan extremo como el que se ha obtenido en la muestra observada

$$\text{p. value} = \frac{(\text{nº}\text{AUC}_\text{i} > \text{AUC}_\text{obs}) + 1}{\text{nº}\text{perm} + 1}$$

$$\text{p. value} = \frac{(\text{nº}\text{MSE}_\text{i} < \text{MSE}_\text{obs}) + 1}{\text{nº}\text{perm} + 1}$$

Etapa 3: Significación estadística del modelo



Ejemplo: Etapa 3: Significación estadística

```
> ## 3.- Significación Estadística con Test de Permutaciones con CV
> num.perm = 100     ## = 10000  ## Número de Permutaciones
> n.cv = 10          ## Número de folder en la CV (X-fold CV)
> ## Vectores para almacenar la capacidad de cada modelo con Respuesta permutada
> perm.AUC = rep ( NA, num.perm )
> perm.ER  = rep ( NA, num.perm )
> ## Proceso de Permutaciones
> for ( ind.perm in 1:num.perm )
+ {
+   ## Dataframe de trabajo (se mantienen los predictores y se permutará la respuesta)
+   ww = xx
+   ## Permutamos los valores de la Variable Respuesta
+   permut = sample ( 1:n.all )                ## Permutamos las Posiciones
+   ww$Class = xx$Class [ permut ]             ## Permutamos los Valores de Y
+   names(ww)[ column.Resp ] = "perm.Class"    ## Cambio de nombre para clarificar
+
+   ## Proceso de CV con la respuesta permutada
+   ## Vectores para almacenar los errores y los AUCs
+   w.error = rep ( NA, n.cv )
+   w.auc   = rep ( NA, n.cv )
.   .   .   .   .
```

- Se establece un **número de permutaciones**, que debería de ser 5.000 o 10.000
- En cada permutación, se asignan **aleatoriamente** los **valores de la variable respuesta** a las observaciones, con la función ***sample()***
- Por comodidad se usa otro dataframe (ww) con una variable Y llamada *"perm.Class"*

Ejemplo: Etapa 3: Significación estadística

```
+    ## Proceso de CV con la respuesta permutada
+    ## Vectores para almacenar los errores y los AUCs
+    w.error = rep ( NA, n.cv )
+    w.auc   = rep ( NA, n.cv )
+    ## Crea el vector "folds" que contiene el grupo (fold) de cada observación
+    folds = sample ( rep ( 1:n.cv, length=n.all ) )
+    ## Validación Cruzada
+    for ( ind.cv in 1:n.cv )
+    {
+        ## Creamos los datasets de training y testing en este fold
+        cv.train = ww [ folds != ind.cv , ]
+        cv.test  = ww [ folds == ind.cv , ]
+        ## Proceso con la Respuesta Permutada !!!!
+        ww.mod = ProcessSVM ( cv.train, cv.test, "perm.Class" )
+        ## Error Rate and AUC
+        w.error [ind.cv] = ErrRate ( cv.test$perm.Class, ww.mod$pred.class )$err.rate
+        w.auc [ind.cv] = ROCAnalysis ( cv.test$perm.Class, ww.mod$pred.prob[ ,
"Impaired" ])$AUC
+    } ## ind.cv
+    ## Tasa de Error y AUC para la permutación es la media de AUC y ER de la CV
+    perm.ER[ ind.perm ]  = mean(w.error)
+    perm.AUC[ ind.perm ] = mean(w.auc)
+ } ## ind.perm
```

- Se repite el **proceso de evaluación de la capacidad predictiva** con CV para los valores de la **variable respuesta permutados**.
- Para cada permutación, se obtiene un ER y AUC del modelo, que se van guardando

Ejemplo: Etapa 3: Significación estadística

```
> ######################################################
> ## Cálculo del P-value del Test de Permutaciones
> perm.p.val.ER  = ( sum ( perm.ER  <= ER.Obs  , na.rm=T ) + 1 ) / (num.perm+1)
> perm.p.val.AUC = ( sum ( perm.AUC >= AUC.Obs , na.rm=T ) + 1 ) / (num.perm+1)
> perm.p.val.ER
[1] 9.999e-05
> perm.p.val.AUC
[1] 9.999e-05
```

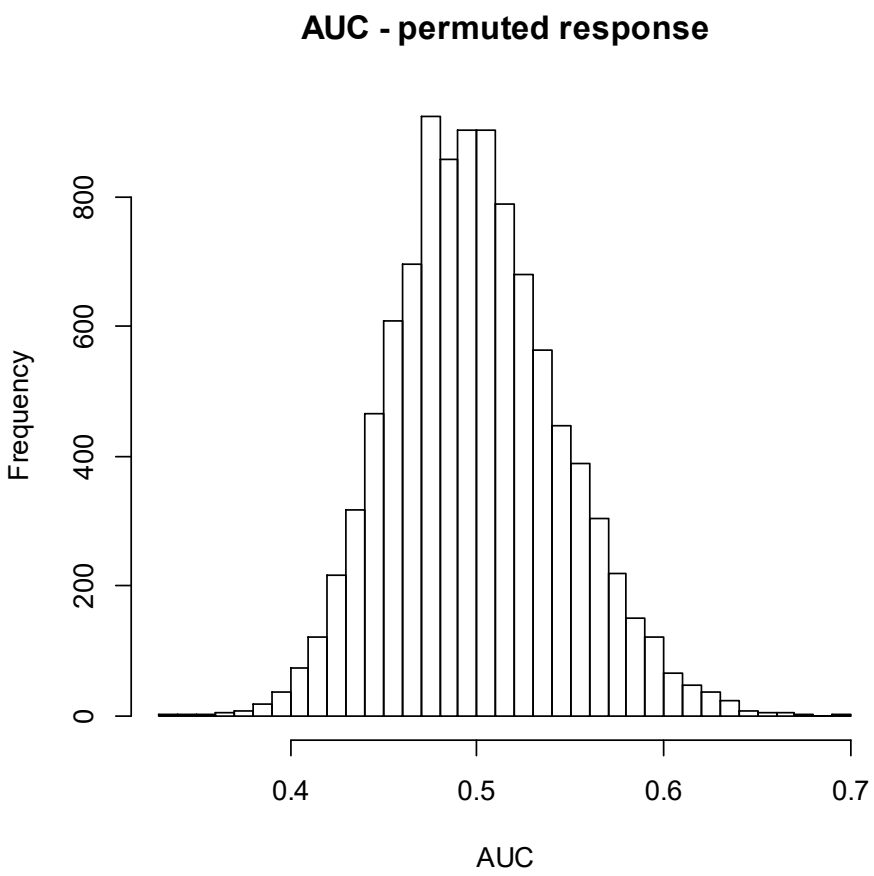
- El **p-value del AUC** se calcula como el número de veces que se ha obtenido un AUC mayor o igual que el observado, dividido por el número de permutaciones, con una ligera corrección
- No se ha obtenido **ningún modelo** con mayor AUC que el del modelo basado en la muestra observada, ni ningún modelo con menor tasa de error
- Los **P-valores** obtenidos con el test de permutaciones son **P=0.000099** para el **AUC** y para la **tasa de error,** basados en 10000 permutaciones
- Se podría reportar como **P<0.0001**

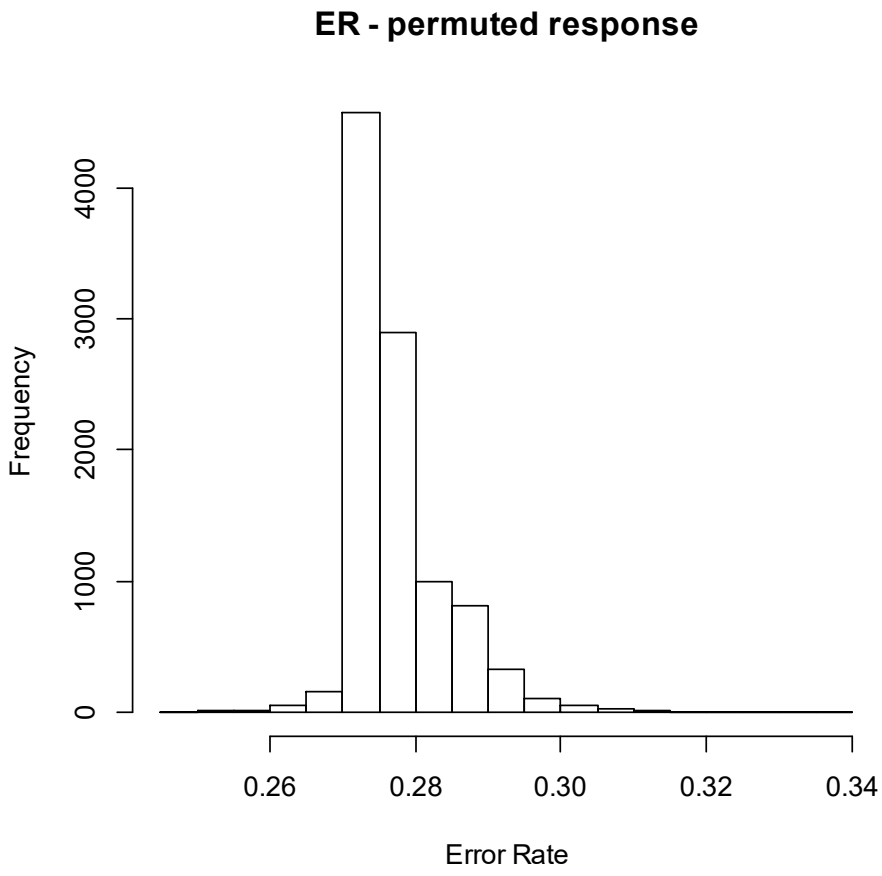
Ejemplo: Etapa 3: Significación estadística

```
> ## ER y AUC observados
> ER.Obs
[1] 0.1319964
> AUC.Obs
[1] 0.8785916
> ## Exploramos las Tasas de Error y AUCs con las respuestas permutadas
> sort(perm.ER)[1:8]
[1] 0.2459002 0.2516934 0.2519608 0.2519608 0.2554367 0.2575758 0.2580214 0.2580214
> mean(perm.ER)
[1] 0.2774287
> sort(perm.AUC, decreasing=T)[1:8]
[1] 0.6948059 0.6717741 0.6714204 0.6679272 0.6609010 0.6608380 0.6601685 0.6574993
> mean(perm.AUC)
[1] 0.4997926
> dev.new()
> hist( perm.AUC, main="AUC - permuted response" , xlab="AUC", breaks=30 )
> dev.new()
> hist( perm.ER, main="ER - permuted response" , xlab="Error Rate", breaks=30 )
```

- Los **mejores modelos** con la variable respuesta permutada tienen AUC=0.69 y error=0.25 , lejos de los obtenidos en la muestra observada (AUC=0.88 y error=0.13)
- Observamos que la **media** del AUC está cerca de 0.5 y la de la tasa de error 0.277, que son los valores esperados para un modelo que clasifica de una **forma aleatoria** (la frecuencia de la clase “Impaired” era 0.273)

Ejemplo: Etapa 3: Significación estadística





Significación estadística. Observaciones

- La metodología presentada es general y se puede usar para:
  - Cualquier **técnica de machine learning** para la construcción de modelos predictivos (SVM, NN, RF, GBM, Modelos de Regresión, ….)
  - Cualquier **técnica de remuestreo** (CV, LOOCV, bootstrapping, …)
  - Cualquier **medida de capacidad predictiva** (AUC, ER, Kappa, MSE, R2, …)

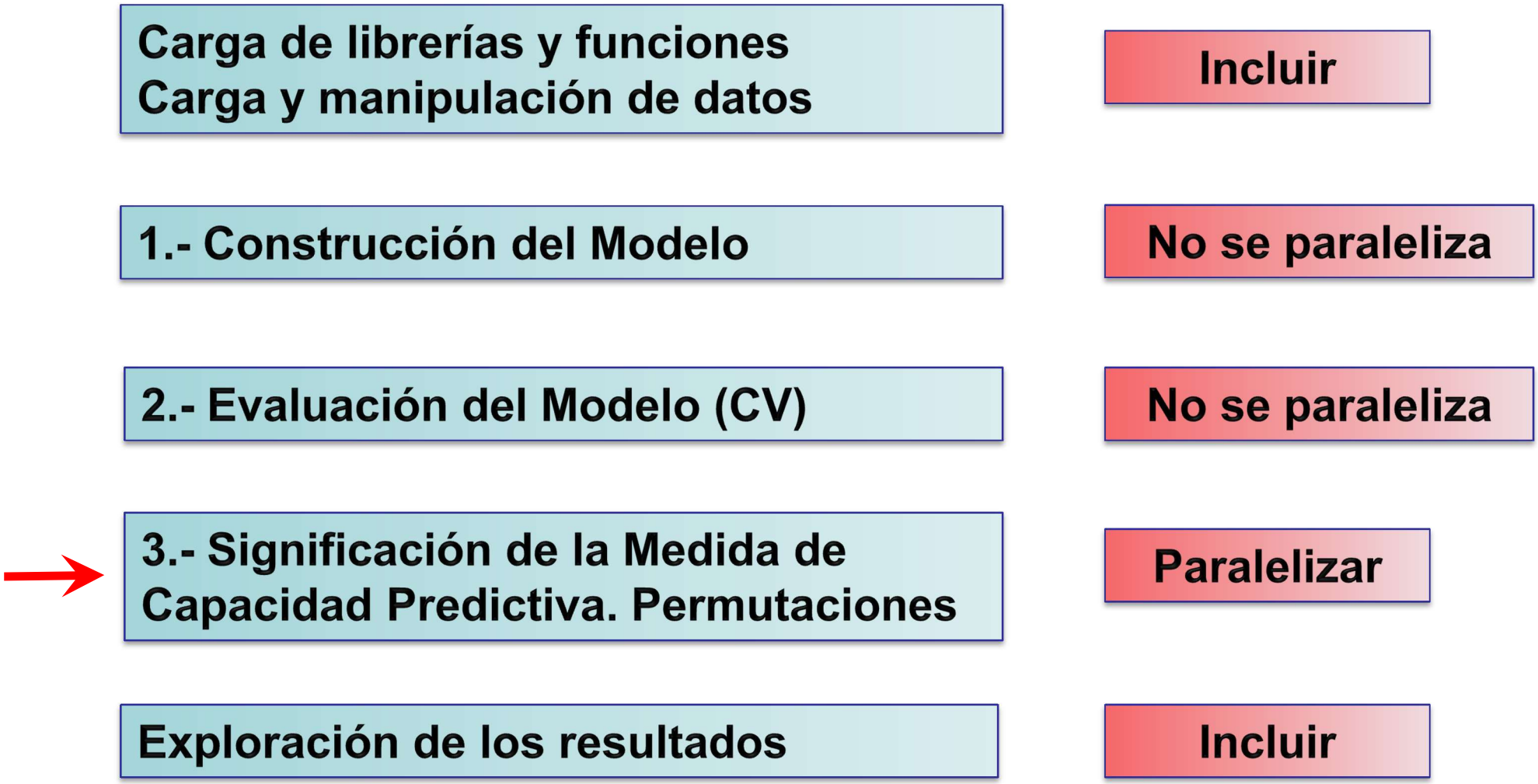
- Los test de permutaciones suelen actuar como los test clásicos: cuánto mayor es el **tamaño de la muestra** más fácil es rechazar la hipótesis nula

Significación estadística. Observaciones

- La exploración de las medidas predictivas obtenidas en **las permutaciones**, con respuestas aleatorias, puede indicar si el **proceso completo** de predicción **es correcto o si hay algún sesgo**
  - Se deberían obtener distribuciones centradas en las **medias esperadas** para modelos que **predicen aleatoriamente** (MSE basal, AUC=0.5, ….)
  - Si esto no ocurre, se debería revisarse el proceso que se ha creado (SV+OP+CM), y si este proceso se está incluyendo completo en la evaluación con CV y en el test de permutaciones

- Se puede utilizar una **muestra externa de validación** para una estimación adicional de la capacidad predictiva del modelo, pero no hay una metodología clara para estimar la significación de esa medida, ya que no se puede reproducir el proceso entero

Paralelización del Proceso



Paralelización con el paquete ***parallel***

- La función más importante del paquete ***parallel*** es ***clusterApply()*** que es la que ejecuta el proceso que se quiere paralelizar
  - Es una función de la familia *apply*

- La función ***clusterApply()*** tiene 3 parámetros:
  - Los clusters, previamente creados
  - Parámetros para la función
  - Una función que contiene el proceso a paralelizar

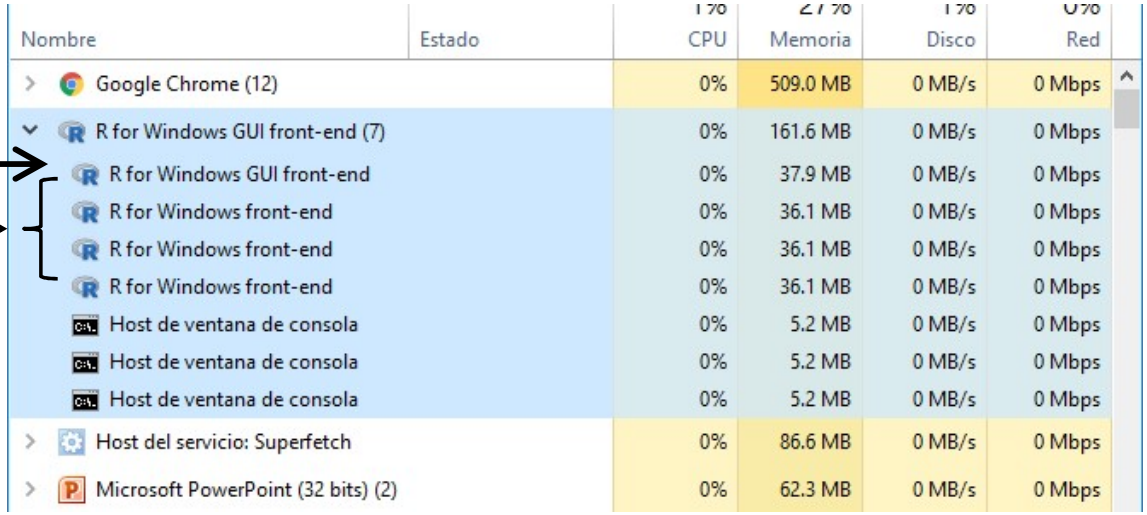
- Se desea paralelizar las permutaciones, es decir, enviar **a distintos clusters distintas permutaciones**
- Por tanto, hay que **crear una función con el código a paralelizar,** que contenga básicamente **el código de una permutación**, el que hemos incluido dentro del bucle

Paralelización con el paquete ***parallel***

- El primer paso es crear los **clusters** (workers) que se desean, que se hace con la función ***makeCluster()*** indicando el número de clusters

Sesión de R

3 clusters



- Abre distintos **procesos de R**, uno por cluster
- En cada uno de ellos hay que cargar todo lo que se necesita: **librerías**, **funciones** (incluida la nueva función con el proceso a paralelizar, la permutación). También, los **datos** y toda la manipulación sobre ellos
- El paquete ***parallel*** tiene diferentes funciones que hacen esto. Una de ellas es la función ***clusterEvalQ()***

Paralelización. Inicializar los clusters

```
> ## PARAL-1: Paquete que permite la paralelización
> library(parallel)
>
> ## PARAL-2: Inicializa los Cluster
>
> ## Se pregunta por el número de cores del ordenador y se inicializan, dejando
> ## un procesador libre
>
> nc <- detectCores() - 1
> nc
[1] 3
> w.cluster <- makeCluster( nc )
> w.cluster
socket cluster with 3 nodes on host 'localhost'
>
> ## Para declarar un número fijo, 28, se podría poner:
> ## w.cluster <- makeCluster( 28 )
>
```

- Se carga la librería ***parallel*** que viene con la instalación básica de R
- Con ***detectCores()*** se pregunta el número de clusters del ordenador
- Con ***makeCluster()*** se crean los clusters

Paralelización. Inicializar los clusters

```
> ## PARAL-3: Con la función "clusterEvalQ" se cargan los paquetes, los datos y
> ## las variables en cada Worker, que es el primer parámetro
>
> ignore = clusterEvalQ ( w.cluster,
+ {
+
+ ## Librerías y Funciones
+ library(e1071)
+ library(pROC)
+ library(caret)
+
+ ## Se cargan las funciones generales
+ source ( "C://Taller Significación Modelos Predictivos/Funciones Generales.r")
+ source ( "C://Taller Significación Modelos Predictivos/Funciones SVM.r")
+
+ ## Fichero Datos: Alzheimer
+
+ xx = read.csv("C://Taller Significación Modelos Predictivos/Alzheimer.csv", sep=";")
+
```

- La función ***clusterEvalQ()*** se usa para inicializar los clusters
- El primer parámetro son los clusters creados con ***makeCluster()***
- En el **segundo parámetro** hay que incluir **todas las órdenes de R** que necesitamos para inicializar el proceso: carga de librerías, carga de funciones externas, carga de los datos, manipulación y transformación de los datos, ….

Paralelización. Inicializar los clusters

```
+ xx$Class = as.factor(xx$Class)
+
+ ## Número de la columna donde está la variable respuesta
+ column.Resp = which ( names(xx) == "Class" )
+
+ ## Normalización de las Variables (con SVM es necesario)
+ w.trans = preProcess (  xx [ , - column.Resp ] , method = c ("center", "scale") )
+ xx [ , - column.Resp ]  = predict( w.trans , xx [ , - column.Resp ] )
+
+
+ ##############################################################################
+ ## PARAL-4: Se define dentro de cada Cluster una función que es la que hace UNA
+ ## PERMUTACIÓN, y que será llamado con una función de la familia "apply"
+ ## - Contiene básicamente el código del "for" del bucle de permutaciones
+ ## - El df "xx" aquí se introducirá en un parámetro "par.xx"
+
+ SVM.Perm.Paralelo = function ( par.ind.perm , par.xx , n.cv=10 )
+ {
```

- Dentro de este segundo parámetro de ***clusterEvalQ()*** hay que definir una función, que llamamos ***SVM.Perm.Paralelo()*** que incluye el código para hacer una permutación
- Los parámetros de esta función son el “número de la permutación” que se ejecuta, el dataframe y el número de carpetas en la CV

Paralelización. Inicializar los clusters

```
+ SVM.Perm.Paralelo = function ( par.ind.perm , par.xx , n.cv=10 )
+ {
+   ## Dataframe de trabajo (se mantienen los predictores y se permutará la respuesta)
+   ww = par.xx
+   n.all = nrow(par.xx)           ## Se ha añadido en la paralelización
+
+   ## Permutamos los valores de la Variable Respuesta
+   permut = sample ( 1:n.all )                ## Permutamos las Posiciones
+   ww$Class = par.xx$Class [ permut ]         ## Permutamos los Valores de Y
+   names(ww)[ column.Resp ] = "perm.Class"    ## Cambio de nombre para clarificar
+
+   ########################################################
+   ## Proceso de CV con la respuesta permutada
+
+   ## Matriz para almacenar los errores y los AUCs
+   w.error = rep ( NA, n.cv )
+   w.auc   = rep ( NA, n.cv )
+
+   ## Crea el vector "folds" que contiene el grupo (fold) de cada observación
+   folds = sample ( rep ( 1:n.cv, length=n.all ) )
+
```

- Esa función incluye **“una permutación”,** y para ello se crea un data frame que contiene la **variable respuesta permutada** en lugar de la observada, y se realiza la estimación del **AUC y ER** con **CV**
- Se crea el vector que contiene los folders de la CV

Paralelización. Inicializar los clusters

```
+    ## Validación Cruzada
+    for ( ind.cv in 1:n.cv )
+    {
+        ## Creamos los datasets de training y testing en este fold
+        cv.train = ww [ folds != ind.cv , ]
+        cv.test  = ww [ folds == ind.cv , ]
+
+        ## Proceso con la Respuesta Permutada !!!!
+        ww.mod = ProcessSVM ( cv.train, cv.test, "perm.Class" )
+
+        ## Error Rate and AUC
+        w.error [ind.cv] = ErrRate ( cv.test$perm.Class, ww.mod$pred.class )$err.rate
+        w.auc [ind.cv] = ROCAnalysis ( cv.test$perm.Class, ww.mod$pred.prob[ ,
"Impaired" ])$AUC
+    } ## ind.cv
+
+    ## OUTPUT: Los resultados de la permutación son el ER y AUC
+    list ( par.ind.perm = par.ind.perm, out.ER = mean(w.error), out.AUC = mean(w.auc) )
+ } ## end-function SVM.Perm.Paralelo
+
+ ## PARAL-5: aquí se cierra la función ""clusterEvalQ"
+    NULL       ## No devuelve nada
+
+ } ) ## final de las órdenes de R (2° parámetro) y final de la función "clusterEvalQ"
```

- Se hace la CV y cómo salida de esta función tenemos el AUC y el ER de la permutación
- Finalmente, se cierran las órdenes del 2º parámetro y la función ***clusterEvalQ()***

Paralelización. Ejecución de las permutaciones

```
> ## PARAL-6: EJECUCIÓN del Proceso:
> ## - Aquí se ejecuta la función en los Clusters que están inicializados
> ## PARAL-7: Esta es la función "apply" que ejecuta la función en cada cluster
> ## repartiendo el "set.perm" entre ellos
>
> ## Conjunto de permutaciones que se van a ejecutar, en un vector
> set.perm = 1:100
>
> perm.out  = clusterApply ( w.cluster, set.perm,
+     function(w.ind) SVM.Perm.Paralelo ( par.ind.perm = w.ind, par.xx = xx, n.cv=10 ))
>
> ## PARAL-8: MUY IMPORTANTE: Se cierran los clusters para liberar los procesadores
> stopCluster(w.cluster)
```

- La función ***clusterApply()*** ejecuta el **proceso completo de las permutaciones**
- En el 2º parámetro se le pasa un vector que contiene los números de permutaciones que se quieren ejecutar, y "los reparte" en los clusters definidos en el 1º parámetro
- El 3º parámetro es la función ***SVM.Perm.Paralelo ()*** que se había creado con "una permutación", que es la que se desea ejecutar
- El **resultado** de esta función se almacena en un objeto que contiene los resultados de **todas las permutaciones**
- Por último, se **cierran los clusters**

Paralelización. Ejecución de las permutaciones

```
> ## PARAL-9: Resultados del Proceso
> ## Explorando el objeto "perm.out" que devuelve: es una lista con una componente
> ## para cada permutación, que contiene la salida de la función "SVM.Perm.Paralelo"
> class( perm.out)
[1] "list"
> length ( perm.out )
[1] 100
> perm.out[[1]]
$`par.ind.perm`
[1] 1
$out.ER
[1] 0.278877
$out.AUC
[1] 0.4441529

> ## Para obtener vectores uniendo todos los resultados
> ## Estos resultados habría que salvarlos y en otro script calcular los ER y AUC
> ## de los datos observados
> perm.ER   = as.numeric( unlist( sapply( perm.out , '[' , 'out.ER'  )))
> perm.AUC  = as.numeric( unlist( sapply( perm.out , '[' , 'out.AUC' )))
>
```

- El objeto que devuelve la función ***clusterApply()*** es una lista con 100 componentes, que contiene los resultados de cada una de las permutaciones
- Utilizamos la función ***unlist()*** con ***sapply()*** para crear vectores que contienen el AUC y el ER

Paralelización. Ejecución de las permutaciones

```
> ## Exploramos las Tasas de Error y AUCs con las respuestas permutadas
> length(perm.ER)
[1] 100
> sort(perm.ER)[1:8]
[1] 0.2699643 0.2699643 0.2704991 0.2705882 0.2727273 0.2727273 0.2727273 0.2728164
> mean(perm.ER)
[1] 0.2767077
> length(perm.AUC)
[1] 100
> sort(perm.AUC, decreasing=T)[1:8]
[1] 0.6427700 0.6253205 0.6221313 0.6218020 0.6183120 0.5945862 0.5781213 0.5752614
> mean(perm.AUC)
[1] 0.4985192
>
```

- Las medias de la distribución del ER y del AUC están dentro de lo esperado

- Para obtener los **p-values** se necesita el ER y el AUC de los datos observados, obtenidos con CV, pero ese cálculo no está incluido en este script
- Este script proporciona solamente la distribución del ER y el AUC con las permutaciones, y deberíamos salvarlos para obtener los p-values

Comparación de 2 modelos predictivos

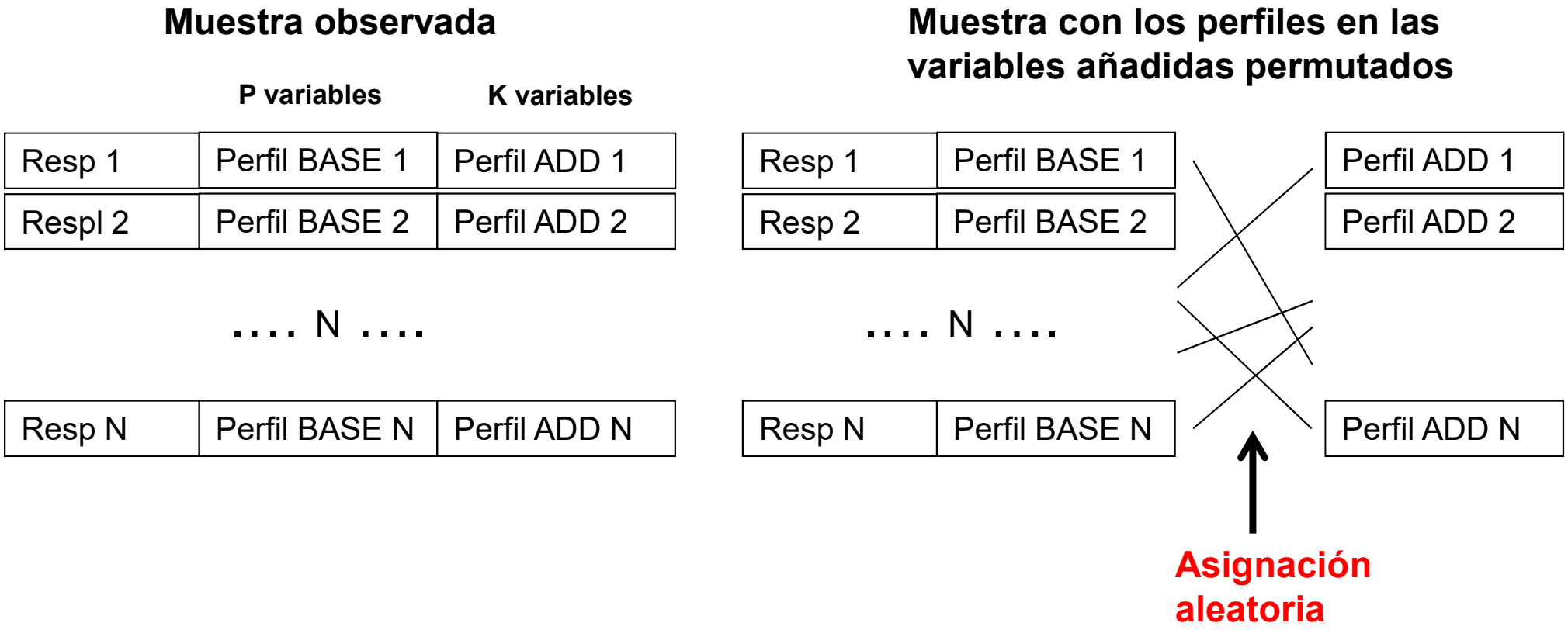
- Se pretende ahora comparar la capacidad predictiva de 2 modelos:
  - **Modelo base** con P predictores
  - **Modelo completo** donde a esos P predictores se les añade otros K predictores adicionales

- Es decir, se pretende contrastar si la **capacidad predictiva del modelo completo**, con más variables, es distinta a la del **modelo base**

- La **comparación** de estos 2 modelos predictivos tiene sentido si se usa la misma **construcción del modelo predictivo y** la misma **técnica de remuestreo** para obtener la capacidad predictiva

- El **test de permutaciones** se va a construir **permutando los perfiles de las K variables añadidas** entre las observaciones, **manteniendo la relación** que ya existe entre los P predictores del **modelo base y la respuesta**

Test de permutaciones. Comparación 2 modelos



- En estas condiciones, las K variables que se añaden “no aportan” nada al modelo base. Se está **añadiendo ruido al modelo de P variables**
- Se espera que el modelo base construido con las P variables, y el modelo con las P variables y las K variables añadidas de forma aleatoria, tengan “la misma” capacidad predictiva

Ejemplo: Alzheimer

- **Objetivo del estudio:** analizar un conjunto de 124 **biomarcadores**, medidas de expresión de **proteínas**, asociados a las primeras etapas de **Alzheimer**

Nombre	Descripción	Categorías / Observaciones
124 Variables	Expresión de proteínas	Biomarcadores exploratorios
age	Edad	
male	Género	0 = mujer, 1 = hombre
tau, p_tau, Ab_42	3 proteínas conocidas que están asociadas a Alzheimer	
E4, E3, E2	3 alelos (SNPs) asociados a Alzheimer	0 = no presenta el alelo, 1 = presenta el alelo
Class	**Presencia de Alzheimer**	**1 = Alzheimer (primeras etapas) (“Impaired”)** **0 = Control**

- El **modelo base** va a estar formado por: age, male, tau, p_tau, Ab_42, E4, E3 y E2, que son variables que ya se sabe que están **asociadas a Alzheimer**
- Se desea conocer si los **124 biomarcadores exploratorios** aportan algo al poder predictivo que tiene el modelo base, y se añadirían para formar el **modelo completo**

Comparación de 2 modelos predictivos

```
> library(e1071)
> library(pROC)
> library(caret)
> ## Se cargan las funciones generales
> source ( "C://Taller Significación Modelos Predictivos/Funciones Generales.r")
> source ( "C://Taller Significación Modelos Predictivos/Funciones SVM.r")
> ## Fichero Datos: Alzheimer
> xx = read.csv("C://Taller Significación Modelos Predictivos/Alzheimer.csv", sep=";")
> dim(xx)
[1] 333 133
> xx$Class = as.factor(xx$Class)
> n.all = nrow(xx)                    ## Número de observaciones
> ## Número de la columna donde está la variable respuesta
> column.Resp = which ( names(xx) == "Class" )
>
> ## Número de las columnas de los Predictores de cada uno de los modelos predictivos
> column.Mod.BASE = 125:132           ## Modelo Base
> column.Mod.ADD = 1:124              ## Variables a añadir
> column.Mod.COM = c ( column.Mod.BASE , column.Mod.ADD )  ## Modelo Completo
```

- Se cargan las librerías, las funciones y los datos
- Se crean vectores con los números de las columnas de las variables que forman:
  - **Modelo Base** (BASE)
  - **Variables que se van a Añadir** (ADD)
  - **Modelo Completo** (COM, uniendo los 2 vectores)

Comparación de 2 modelos predictivos

```
> ## Normalización de las Variables (con SVM es necesario)
> w.trans = preProcess (  xx [ , - column.Resp ] , method = c ("center", "scale") )
> xx [ , - column.Resp ]  = predict( w.trans , xx [ , - column.Resp ] )

> ######################################################################
> ## 1.- Construcción de los 2 Modelos Predictivos
>
> mod.base = ProcessSVM ( xx [ , c(column.Mod.BASE, column.Resp)] ,
+                         xx [ , c(column.Mod.BASE, column.Resp)] , "Class" )
> mod.completo = ProcessSVM ( xx [ , c(column.Mod.COM, column.Resp)] ,
+                           xx [ , c(column.Mod.COM, column.Resp)] , "Class" )
>
> ######################################################################
> ## 2.- Evaluación de la Capacidad Predictiva con CV de los 2 modelos
>
> ## Número de folder en la CV (X-fold CV)
> n.cv = 10
> ## Vectores para almacenar las Tasas de Error y los AUCs
> w.err.BASE  = rep ( NA, n.cv )
> w.auc.BASE  = rep ( NA, n.cv )
> w.err.COM  = rep ( NA, n.cv )
> w.auc.COM  = rep ( NA, n.cv )
```

- Se normalizan los datos (SVM)
- Se construyen los **2 modelos predictivos finales**
- Se preparan los vectores para evaluar la **capacidad predictiva** con validación cruzada

Comparación de 2 modelos predictivos

```
> ## Crea el vector "folds" que contiene el grupo (fold) de cada observación
> folds = sample ( rep ( 1:n.cv, length=n.all ) )
> ## Validación Cruzada
> for ( ind.cv in 1:n.cv )
+ {
+   ## Creamos los datasets de training y testing en este fold
+   cv.train = xx [ folds != ind.cv , ]
+   cv.test  = xx [ folds == ind.cv , ]
+
+   ## Proceso - Modelo Base
+   ww.mod.BASE  = ProcessSVM ( cv.train[ , c(column.Mod.BASE, column.Resp)],
+                               cv.test [ , c(column.Mod.BASE, column.Resp)], "Class" )
+   ## Error Rate and AUC - Modelo Base
+   w.err.BASE [ind.cv] = ErrRate ( cv.test$Class, ww.mod.BASE$pred.class )$err.rate
+   w.auc.BASE [ind.cv] = ROCAnalysis ( cv.test$Class, ww.mod.BASE$pred.prob[ ,
"Impaired" ])$AUC
+
+   ## Proceso - Modelo Completo
+   ww.mod.COM = ProcessSVM ( cv.train[ , c(column.Mod.COM, column.Resp)],
+                             cv.test [ , c(column.Mod.COM, column.Resp)], "Class" )
+   ## Error Rate and AUC - Modelo Completo
+   w.err.COM [ind.cv] = ErrRate ( cv.test$Class, ww.mod.COM$pred.class )$err.rate
+   w.auc.COM [ind.cv] = ROCAnalysis ( cv.test$Class, ww.mod.COM$pred.prob[ ,
"Impaired" ])$AUC
+ }
```

- El proceso de **validación cruzada** debe usar las **mismas particiones** para evaluar la **capacidad predictiva** del **modelo base** y del **modelo completo**

Comparación de 2 modelos predictivos

```
> ## Error y AUC medios
> ER.Obs.BASE = mean(w.err.BASE)
> AUC.Obs.BASE = mean(w.auc.BASE)
> AUC.Obs.BASE
[1] 0.84787
> ER.Obs.BASE
[1] 0.209893
> ER.Obs.COM = mean(w.err.COM)
> AUC.Obs.COM = mean(w.auc.COM)
> AUC.Obs.COM
[1] 0.8852264
> ER.Obs.COM
[1] 0.1495544
```

- En modelo base tiene un AUC = 0.848 y el modelo completo sube a AUC=0.885
- En modelo base tiene un ER = 0.210 y el modelo completo baja ER=0.150

- **El test de permutaciones** lo que contrasta es si las estimaciones de **capacidad predictiva** de estos dos modelos son **iguales**:
  - AUC=0.848 y AUC=0.885
  - ER=0.210 y ER=0.150

Comparación de 2 modelos predictivos

```
> ## 3.- Significación Estadística con Test de Permutaciones con CV
> num.perm = 100  ## = 10000  ## Número de Permutaciones
> n.cv = 10       ## Número de folder en la CV (X-fold CV)
> ## Vectores para almacenar la capacidad de cada modelo con Respuesta permutada
> perm.AUC = rep ( NA, num.perm )
> perm.ER  = rep ( NA, num.perm )
> ## Proceso de Permutaciones
> for ( ind.perm in 1:num.perm )
+ {
+   ## Permutamos las filas
+   permut = sample ( 1:n.all )                  ## Permutamos las Posiciones
+
+   # Dataframe de trabajo. Se mantienen los predictores del modelo Base y la Respuesta
+   # Se permutará los predictores añadidos
+   ww = cbind ( xx [         , c ( column.Mod.BASE , column.Resp ) ] ,
+                xx [ permut , c ( column.Mod.ADD ) ] )
+   names(ww)[ which ( names(ww) == "Class" ) ] = "perm.Class"  ## Cambio de nombre
```

- Dentro de **cada permutación**, se permutan las posiciones de las observaciones
- Se forma el **dataframe a analizar**, que está formado por las siguientes columnas:
  - **respuesta y predictores del modelo base**, en el **mismo orden** en el que están las observaciones en el **fichero original** (1º dimensión, las filas en blanco)
  - **predictores añadidos** en el **orden aleatorio** determinado por la permutación

Comparación de 2 modelos predictivos

```
+    ## Proceso de CV con los predictores añadidos permutados
+    w.error = rep ( NA, n.cv )
+    w.auc   = rep ( NA, n.cv )
+    ## Crea el vector "folds" que contiene el grupo (fold) de cada observación
+    folds = sample ( rep ( 1:n.cv, length=n.all ) )
+    ## Validación Cruzada
+    for ( ind.cv in 1:n.cv )
+    {
+        ## Creamos los datasets de training y testing en este fold
+        cv.train = ww [ folds != ind.cv , ]
+        cv.test  = ww [ folds == ind.cv , ]
+        ## Proceso con la Respuesta Permutada !!!!
+        ww.mod = ProcessSVM ( cv.train, cv.test, "perm.Class" )
+        ## Error Rate and AUC
+        w.error [ind.cv] = ErrRate ( cv.test$perm.Class, ww.mod$pred.class )$err.rate
+        w.auc [ind.cv] = ROCAnalysis ( cv.test$perm.Class, ww.mod$pred.prob[ ,
"Impaired" ])$AUC
+    } ## ind.cv
+    ## Tasa de Error y AUC para la permutación es la media de AUC y ER de la CV
+    perm.ER[ ind.perm ]  = mean(w.error)
+    perm.AUC[ ind.perm ] = mean(w.auc)
+ } ## ind.perm
```

- El proceso habitual de CV: se sortean las particiones, y para cada una de ellas se crean los datasets de training para construir el modelo y el de testing para evaluarlo
- Se almacena el **AUC y ER de la permutación** como la media obtenidas en las particiones

Comparación de 2 modelos predictivos

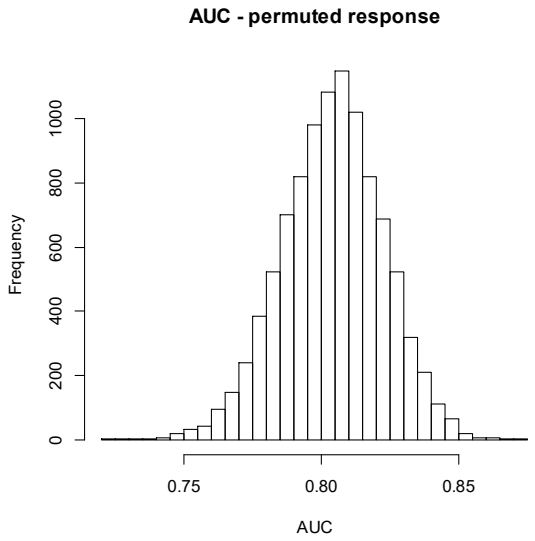
```
> ## Cálculo del P-value del Test de Permutaciones
> perm.p.val.ER  = ( sum ( perm.ER  <= ER.Obs.COM  , na.rm=T ) + 1 ) / (num.perm+1)
> perm.p.val.AUC = ( sum ( perm.AUC >= AUC.Obs.COM , na.rm=T ) + 1 ) / (num.perm+1)
> perm.p.val.ER
[1] 0.00019998
> perm.p.val.AUC
[1] 9.999e-05
>
> ## ER y AUC observados
> AUC.Obs.COM
[1] 0.8852264
> ER.Obs.COM
[1] 0.1495544
> ## Exploramos loas Tasas de Error y AUCs con las respuestas permutadss
> sort(perm.ER)[1:8]
[1] 0.1467914 0.1527629 0.1619430 0.1654189 0.1681818 0.1707665 0.1709447 0.1738859
> sort(perm.AUC, decreasing=T)[1:8]
[1] 0.8704304 0.8686414 0.8632590 0.8626271 0.8620319 0.8615943 0.8603762 0.8594116
```

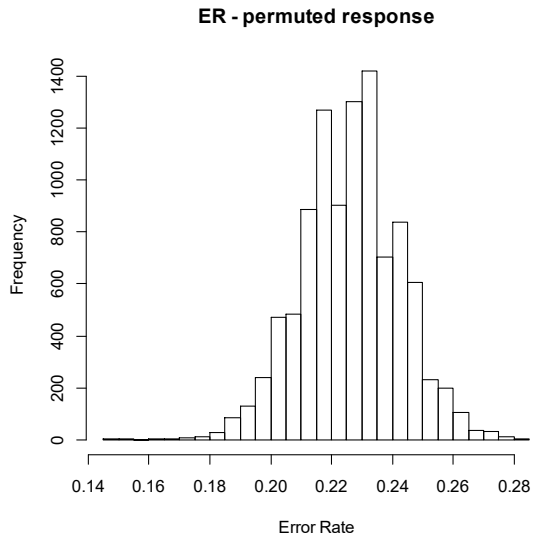
- Ningún modelo con los predictores añadidos ha obtenido un AUC mayor que el del modelo completo. Por tanto, el **p-value = 0.000099**
- Hay 1 modelo que ha obtenido un ER más bajo (0.147) . Por tanto, el **p-value = 0.000199**
- Se concluye que el **modelo completo**, que incluye las 124 variables adicionales, **mejora el modelo base** en capacidad predictiva

Comparación de 2 modelos predictivos

```
> AUC.Obs.A
[1] 0.84787
> ER.Obs.A
[1] 0.209893
> mean(perm.AUC)
[1] 0.8040767
> mean(perm.ER)
[1] 0.2262942
> dev.new()
> hist( perm.AUC, main="AUC - permuted response" , xlab="AUC", breaks=30 )
> dev.new()
> hist( perm.ER, main="ER - permuted response" , xlab="Error Rate", breaks=30 )
```

- El AUC medio obtenido en las permutaciones (0.804) es bastante menor que el observado en el modelo base (0.848)
- Algunas técnicas pierden capacidad predictiva al incluir ruido, en forma de variables no relacionadas con la respuesta





Anexo

Función: Tasa de Error

```
## Error Rate: Tasa de Error
ErrRate = function ( obs, pred )
{
  ## Deben tener la misma longitud
  if ( length(obs) != length(pred) )
  { stop ( "Los vectores deben tener la misma longitud" ) }

  ## Tabla
  tt = table( obs, pred )

  ## Proporción de Buenas y Malas clasificaciones
  accuracy = sum ( as.character(obs) == as.character(pred) )
  accuracy = accuracy / sum(tt)
  err.rate = 1 - accuracy

  list( accuracy = accuracy, err.rate = err.rate )
}
```

- La función ***ErrRate*** evalúa la precisión y la **tasa de error**, definidas como proporción de observaciones clasificadas correcta e incorrectamente

Función: Análisis de la curva ROC

```
ROCAnalysis = function( bin.var, cont.var, last = TRUE, ci = FALSE, plot = FALSE, ... )
{
  ## Comprobamos la variable binaria
  if ( length( unique (bin.var ) ) != 2 )
  { stop ( "La Variable bin.var debe ser binaria" ) }
  ## ROC Analysis
  ## Si se predice la última categoría (last=TRUE), la dirección del predictor
  ## es "<", los valores altos corresponden a los casos (controls < t <= cases)
  if ( last == TRUE ) { roc.out <- roc( bin.var , cont.var, direction="<" ) }
  else                { roc.out <- roc( bin.var , cont.var, direction=">" ) }
  ## AUC con IC95%
  w.auc  = auc (roc.out)
  if ( ci == TRUE ) { w.ci   = ci.auc (roc.out)  }

  ## Plot de la Curva ROC (opcional)
  if ( plot == TRUE )
  { dev.new(); plot( roc.out , legacy.axes = TRUE, col="red" , ... )
    text (0.2, 0.2, paste("AUC = ", round(w.auc ,3 )))
  }
  ## Output
  if ( ci == TRUE ) { list ( AUC = w.auc[1] , CI.AUC = c( w.ci[1] , w.ci[3] ) ) }
  else              { list ( AUC = w.auc[1] ) }
}
```

- La función ***ROCAnalysis*** devuelve el **AUC de la curva ROC** (paquete ***pROC***)
- El parámetro *last=T* indica que se está prediciendo la última categoría. Opcionalmente muestra el gráfico de la curva y el intervalo de confianza (bootstrapping)

Función: Evaluación de Modelos de Regresión

```
## Medidas de Evaluación de Modelos de Regresión (Var. Respuesta Cuantitativa)
EvalRegr = function ( obs, pred )
{
  ## Deben tener la misma longitud
  if ( length(obs) != length(pred) )
  { stop ( "Los vectores deben tener la misma longitud" ) }

  ## MSE y RMSE
  mse = mean ( ( obs - pred ) ** 2 )
  rmse = sqrt ( mse)
  ## Error Estandard del MSE
  ## El SE se calcula como el SD(res^2)/sqrt(n) porque el MSE es uma media de res^2
  se.mse = sd ( ( obs - pred ) ** 2 ) / sqrt(length(obs))

  ## MAE
  mae = mean ( abs( obs - pred ) )
  ## R2
  R2 = cor ( obs, pred ) ** 2

  ## Parámetros de salida
  list( MSE = mse, RMSE = rmse, MAE = mae, R2 = R2 , SE.MSE = se.mse )
}
```

- La función ***EvalRegr*** devuelve las medidas (**MSE, RMSE, MAE y R2**) para evaluar el error en los **modelos de Regresión**

Ejemplo: función *sample()*

```
> ## Muestra de (0,1) con reemplazamiento de tamaño 50 con proprociones
> ww = sample ( c(0,1), size=50 , replace=TRUE, prob=c(0.3, 0.7) )
 [1] 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1 1 0 0 0 0 1 0 1 0 1 1 1 0 1
[38] 1 0 0 0 0 0 1 0 1 0 0 1 1
> table(ww)
ww
 0  1
19 31
> ## Se construye una muestra de tamaño 52 con números del 1 al 5
> ww = sample ( rep ( 1:5, length=52 ) )
 [1] 5 3 1 5 4 4 1 5 2 1 4 2 2 3 4 3 1 5 4 4 3 2 3 3 1 2 4 4 5 2 2 1 1 3 3 4 2
[38] 5 1 2 2 1 3 3 2 5 1 5 4 5 1 5
> table(ww)
ww
 1  2  3  4  5
11 11 10 10 10
> ## Se extrae una muestra con reemplazamiento con números de 1 al 50
> ww = sample ( 1:50 , replace=T )
 [1] 28 37 21 18 38 47  3 22 38 29 46 43 11 30  1 33  2 24 18  2  2 29 23 19 41 44 37
[28]  6 12 32  2 17  8 39 22  7  8  8 49 25 32 14 49 25 36 10 37  3 43 47
> length ( unique( ww ) ) / 50
[1] 0.66
```

- La función ***sample()*** extrae una muestra de un tamaño determinado (*size*=) entre los elementos de un conjunto
- Permite hacerlo con reemplazamiento (*replace*=) y asignando probabilidades (*prob*=)